51st International Colloquium on Automata, Languages, and Programming

ICALP 2024, July 8–12, 2024, Tallinn, Estonia

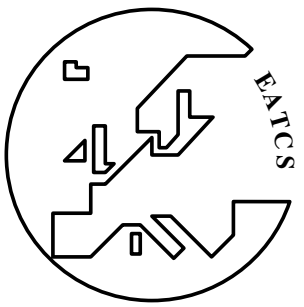
Edited by

Karl Bringmann

Martin Grohe

Gabriele Puppis

Ola Svensson



Editors

Karl Bringmann
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
bringmann@cs.uni-saarland.de

Martin Grohe
RWTH Aachen University, Germany
grohe@informatik.rwth-aachen.de

Gabriele Puppis
University of Udine, Italy
gabriele.puppis@uniud.it

Ola Svensson
EPFL, Lausanne, Switzerland
ola.svensson@epfl.ch

ACM Classification 2012
Theory of computation

ISBN 978-3-95977-322-5

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-322-5.

Publication date
July, 2024

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.ICALP.2024.0

ISBN 978-3-95977-322-5 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Track B: Automata, Logic, Semantics, and Theory of Programming

Preface

This volume contains the papers presented at the *51st EATCS International Conference on Automata, Languages and Programming (**ICALP 2024**)*, held in Tallinn, Estonia, during July 8–12, 2024. ICALP is a series of annual conferences of the *European Association for Theoretical Computer Science (EATCS)*, which first took place in 1972. This year, ICALP was co-located with the 39th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) and the 9th International Conference on Formal Structures for Computation and Deduction (FSCD).

The ICALP 2024 program consisted of two tracks:

Track A: Algorithms, Complexity, and Games

Track B: Automata, Logic, Semantics, and Theory of Programming

In response to the call for papers, a total of 519 eligible, anonymous submissions were received: 404 for Track A and 115 for Track B. The committees decided to accept 153 papers for inclusion in the scientific program: 119 papers for Track A and 34 for Track B. The selection was made by the program committees based on originality, quality, and relevance to theoretical computer science. The quality of the submissions was very high, and many deserving papers could not be selected.

The EATCS sponsored awards for both a best paper and a best student paper in each of the two tracks, selected by the program committees. The **best paper awards** were given to the following papers:

Track A: Yuda Feng and Shi Li. *A Note on Approximating Weighted Nash Social Welfare with Additive Valuations.*

Track B: Dmitry Chistikov, Alessio Mansutti, and Mikhail Starchak. *Integer Linear-Exponential Programming in NP by Quantifier Elimination.*

The **best student paper awards**, for papers that are solely authored by students, were given to the following papers:

Track A: Ce Jin and Hongxun Wu. *A Faster Algorithm for Pigeonhole Equal Sums.*

Track A: Kingsley Yung. *Limits of Sequential Local Algorithms on the Random k-XORSAT Problem.*

Track B: Roland Guttenberg. *Flattability of Priority Vector Addition Systems.*

ICALP 2024 included invited presentations by

- Anuj Dawar, University of Cambridge,
- Edith Elkind, University of Oxford (joint with LICS 2024),
- Danupon Nanongkai, MPI Saarbrücken,
- Merav Parter, Weizmann Institute,
- Stephanie Weirich, University of Pennsylvania (joint with LICS 2024 and FSCD 2024).

This volume contains all the contributed papers presented at the conference, and an abstract or paper accompanying some of the invited talks.

The program of ICALP 2024 also included presentations of

- the Gödel Prize 2024 (joint with ACM SIGACT) awarded to Ryan Williams (MIT) for the paper *Non-Uniform ACC Circuit Lower Bounds*: IEEE Conference on Computational Complexity (CCC) 2011. Journal of the ACM 61(1):1–32 (2014).
- the Alonzo Church Award 2024 (joint with LICS), awarded to Thomas Ehrhard (CNRS / IRIF) and Laurent Regnier (Université d'Aix-Marseille).

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- the EATCS Award 2024 to Samson Abramsky (Department of Computer Science, Oxford University),
- the Presburger Award 2024, awarded jointly to Justin Hsu (Cornell University) and Pravesh Kothari (Princeton University).

The EATCS Distinguished Dissertation Award 2023 was awarded jointly to the following PhD dissertations:

- William Kuszmaul (MIT). *Randomized Algorithms that Achieve the Unexpected.*
- Nathan Klein (University of Washington). *Finding Structure in Entropy: Improved Approximation Algorithms for TSP and other Graph Problems.*
- Ruiwen Dong (University of Oxford). *Algorithmic Problems for Subsemigroups of Infinite Groups.*

There was also the announcement of the new EATCS Fellows for 2024, who are:

- Yossi Azar (Blavatnik School of Computer Science, Tel-Aviv University),
- Friedhelm Meyer auf der Heide (Heinz Nixdorf Institute and Department of Computer Science, Paderborn University).

The following workshops were held as satellite events of ICALP 2024, LICS 2024, and FSCD 2024 during July 6-9, 2024:

- Algorithmic Aspects of Temporal Graphs VII (AATG 2024)
- Geometric and Topological Methods in Computer Science (GETCO 2024)
- Intersection Types and Related Systems (ITRS 2024)
- International Workshop on Confluence (IWC 2024)
- Learning and Automata (LearnAut 2024)
- Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2024)
- Logic Mentoring Workshop (LMW 2024)
- Mathematically Structured Functional Programming (MSFP 2024)
- Parameterized Approximation Algorithms Workshop (PAAW 2024)
- Parameterized Algorithms and Constraint Satisfaction (PACS 2024)
- Structure meets Power (SmP 2024)
- Trends in Arithmetic Theories (TAT 2024)
- Trends in Linear Logic and Applications (TLLA 2024)
- Women in Logic 2024

We wish to thank all authors who submitted extended abstracts for consideration, the program committees for their scholarly effort, and all the reviewers who assisted the program committees in the evaluation process.

We are very grateful to the Conference General Chair, Pawel Sobocinski, his colleagues from Tallinn University of Technology, and EATCS, for hosting ICALP 2024 in Tallinn.

Finally, we would like to thank Anca Muscholl, the Chair of the ICALP Steering Committee, for her continuous support, Artur Czumaj, the president of EATCS, for his generous advice on the organization of the conference, as well as the editorial office of LIPIcs for their support in editing these proceedings.

July 2024

Karl Bringmann
Martin Grohe
Gabriele Puppis
Ola Svensson

Organization

Program Committees

Track A

Nima Anari	Stanford University
Karl Bringmann (*co-chair*)	Saarland University
Parinya Chalermsook	Aalto University
Vincent Cohen-Addad	Google Research
Jose Correa	Universidad de Chile
Holger Dell	Goethe University Frankfurt
Ilias Diakonikolas	University of Wisconsin-Madison
Yuval Filmus	Technion
Arnold Filtser	Bar Ilan University
Naveen Garg	IIT Delhi
Pawel Gawrychowski	University of Wrocław
Anupam Gupta	Carnegie Mellon University
Samuel Hopkins	MIT
Sophie Huiberts	Columbia University
Giuseppe Italiano	LUISS University
Michael Kapralov	EPFL
Eun Jung Kim	Université Paris-Dauphine
Sándor Kisfaludi-Bak	Aalto University
Tomasz Kociumaka	Max-Planck-Institute for Informatics
Fabian Kuhn	University of Freiburg
Amit Kumar	IIT Delhi
William Kuszmaul	Harvard University
Rasmus Kyng	ETH Zurich
Kasper Green Larsen	Aarhus University
François Le Gall	Nagoya University
Pasin Manurangsi	Google Research
Daniel Marx	CISPA Helmholtz Center for Information Security
Yannic Maus	TU Graz
Nicole Megow	University of Bremen
Ruta Mehta	University of Illinois at Urbana-Champaign
Jakob Nordström	University of Copenhagen
Richard Peng	University of Waterloo
Seth Pettie	University of Michigan
Adam Polak	Bocconi University
Lars Rohwedder	Maastricht University
Eva Rotenberg	DTU Compute
Sushant Sachdeva	University of Toronto
Melanie Schmidt	University of Cologne
Sebastian Siebertz	University of Bremen
Shay Solomon	Tel Aviv University
Nick Spooner	University of Warwick

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Clifford Stein	Columbia University
Ola Svensson (*co-chair*)	EPFL
Luca Trevisan	Bocconi University
Ali Vakilian	Toyota Technological Institute Chicago
Jan van den Brand	Georgia Tech
Erik Jan van Leeuwen	Utrecht University
Oren Weimann	University of Haifa
Nicole Wein	University of Michigan
Andreas Wiese	TU Munich
John Wright	UC Berkeley

Track B

Arnold Beckmann	Swansea University
Manuel Bodirsky	TU Dresden
Patricia Bouyer	CNRS, LMF
Yijia Chen	Shanghai Jiao Tong University
Victor Dalmau	Universitat Pompeu Fabra
Laurent Doyen	CNRS, LMF
Marcelo Fiore	Cambridge University
Stefan Göller	University of Kassel
Martin Grohe (*chair*)	RWTH Aachen University
Sandra Kiefer	Oxford University
Aleks Kissinger	Oxford University
Bartek Klin	Oxford University
Antonin Kucera	Masaryk University Brno
Carsten Lutz	University of Leipzig
Jerzy Marcinkowski	University of Wrocław
Annabelle McIver	Macquaire University Sydney
Andrzej Murawski	Oxford University
Paweł Parys	University of Warsaw
Michał Pilipczuk	University of Warsaw
Joel Ouaknine	Max Planck Institute for Software Systems
Christian Riveros	Pontificia Universidad Catolica de Chile
Alexandra Silva	Cornell University
Balder ten Cate	ILLC Amsterdam
Szymon Toruńczyk	University of Warsaw
Igor Walukiewicz	CNRS, University of Bordeaux
Sarah Winter	IRIF, University Paris Cité
Georg Zetzsche	Max Planck Institute for Software Systems
Martin Ziegler	KAIST

Organizing Committee

Pawel Sobocinski (*chair*)	Tallinn University of Technology
Niccolò Veltri	Tallinn University of Technology
Amar Hadzihasanovic	Tallinn University of Technology
Fosco Loregian	Tallinn University of Technology
Matt Earnshaw	Tallinn University of Technology
Diana Kessler	Tallinn University of Technology
Ekaterina Zhuchko	Tallinn University of Technology
Kristi Ainen	Tallinn University of Technology

ICALP Steering Committee

Luca Aceto	Reykjavik University and Gran Sasso Science Institute, L'Aquila
Artur Czumaj	University of Warwick
Kousha Etessami	University of Edinburgh
Uriel Feige	Weizmann Institute
Sevag Gharibian	University of Paderborn
Thore Husfeldt (co-chair)	Lund University and IT University of Copenhagen
Anca Muscholl (chair)	University of Bordeaux
Luke Ong	NTU Singapore
Yuval Rabani	Hebrew University
Eva Rotenberg	DTU Compute
Paul Spirakis	University of Liverpool and University of Patras
Ola Svensson	EPFL

Financial Sponsors

We extend our gratitude to our generous sponsors for their support in ensuring the success of ICALP 2024:

External Reviewers

Anders Aamand
Mikkel Abrahamsen
Isolde Adler
Michal Ajdarow
Maryam Aliakbarpour
Jorge Almeida
Omar Amer
Antonios Antoniadis
Sepehr Assadi
Henry Austin
Amir Azarmehr
Ainesh Bakshi
Sayan Bandyapadhyay
Chris Barrett
Nicolas Basset
Soheil Behnezhad
Omri Ben-Eliezer
Ioana Bercea
Aaron Bernstein
Alexis Bes
Umang Bhaskar
Sayan Bhattacharya
Laurent Bienvenu
Jannis Blauth
Achim Blumensath
Greg Bodwin
Benedikt Bollig
Benjamin Bordais
Joshua Brakensiek
Adam Brown
Moritz Buchem
Mark Bun
Karthik C. S.
Clément Canonne
Antonio Casares
Ruoxu Cen
Karthekeyan Chandrasekaran
Panagiotis Charalampopoulos
Eshan Chattopadhyay
Shiri Chechik
Kuancheng Chen
Zongchen Chen
Eden Chlamtac
Lorenzo Clemente
Edith Cohen
Jonas Conneryd
Mohammad Ali Abam
Duncan Adamson
Arpit Agarwal
Hugo Akitaya
Shaull Almagor
Andris Ambainis
Alexandr Andoni
Simon Apers
Albert Atserias
Per Austrin
Giorgio Bacci
Nikhil Balaji
Kiril Bangachev
Libor Barto
Tugkan Batu
Amir Ben-Amram
Huck Bennett
Benjamin Bergougnoux
Raphaël Berthon
Amey Bhangale
Aditya Bhaskara
Sujoy Bhore
Davide Bilò
Joakim Blikstad
Markus Bläser
Jan Bok
Ilario Bonacina
Vitor Bosshard
Marco Bressan
Frederik Brüning
Maike Buchin
Jaroslaw Byrka
Michaël Cadilhac
Ioannis Caragiannis
Arnaud Casteigts
Jérémie Chalopin
Hsien-Chih Chang
Witold Charatonik
Vaggos Chatziafratis
Jingbang Chen
Li Chen
Ashish Chiplunkar
Valerio Cini
Raphael Clifford
Sarel Cohen
Jonathan Conroy
Amir Abboud
Deeksha Adil
Jungho Ahn
Hannaneh Akrami
Josh Alman
Afrouz Jabal Ameli
Spyros Angelopoulos
Noel Arteche
Clément Aubert
Yossi Azar
Aras Bacho
A. R. Balasubramanian
Kiarash Banihashem
Mateusz Basiak
Pascal Baumann
Shalev Ben-David
Matthias Bentert
Sebastian Berndt
Dietmar Berwanger
Koustav Bhanja
Anup Bhattacharya
Marcin Bienkowski
Andreas Björklund
Michael Blondin
Thomas Bläsius
Udi Boker
Itai Boneh
Nicolas Bousquet
Marcin Briański
Niv Buchbinder
Peter Buergisser
Kristóf Bérczi
Rutger Campbell
Arnaud Carayol
Javier Cembrano
Timothy M. Chan
Yi-Jun Chang
Krishnendu Chatterjee
Bhaskar Ray Chaudhury
Justin Chen
Yu Chen
Rajesh Chitnis
Emily Clement
Christian Coester
Amin Coja-Oghlan
Alex Conway

Martin Costa
Manuel Cáceres
Clément Dallard
Samir Datta
Max Deppert
Michael Dinitz
Michal Dory
Ran Duan
Paul Duetting
Julien Duron
Nico Döttling
Talya Eden
Eduard Eiben
Marek Elias
Ehsan Emamjomeh-Zadeh
Matthias Englert
Thomas Erlebach
Alireza Farhadi
Moran Feldman
Santiago Figueira
Simon D. Fink
Nick Fischer
Jacob Focke
András Frank
Daniele Friolo
Nicola Galesi
Yu Gao
Stéphane Gaubert
Guillaume Genestier
Frederik Geth
Yassine Ghannane
Suprovat Ghoshal
Ludmila Glinskih
Jesse Goodman
Mayank Goswami
Fabrizio Grandoni
Nathan Grosshans
Manoj Gupta
Waldo Gálvez
Serge Haddad
Lianna Hambardzumyan
Sariel Har-Peled
Koyo Hayashi
Markus Hecher
Danny Hermelin
Lukas Hintze
Petr Hlineny
Jędrzej Hodor

Bruno Courcelle
Rajni Dabas
Rathish Das
Niel De Beaudrap
Josee Desharnais
Yann Disser
Jan Dreier
Aditi Dudeja
Fabien Dufoulon
Pranjal Dutta
Anita Dürr
Klim Efremenko
Kord Eickmeyer
Michael Elkin
Jacopo Emmenegger
David Eppstein
Rolf Fagerberg
John Fearnley
Cristina Fernandes
Nathanaël Fijalkow
Bernd Finkbeiner
Noah Fleming
Florent Foucaud
Cody Freitag
Ameet Gadekar
Arnab Ganguly
Mohit Garg
Floris Geerts
Colin Geniet
Nadim Ghaddar
Surendra Ghentiyala
George Giakkoupis
Shay Golan
Gramoz Goranci
Themistoklis Gouleakis
Daniel Graça
Yuzhou Gu
Siddharth Gupta
Andreas Göbel
Magnús M. Halldórsson
Thekla Hamm
Tim A. Hartmann
Meng He
Reiko Heckel
D. Ellis Hershkowitz
Edward A. Hirsch
Duc A. Hoang
Ruben Hoeksma

Wojciech Czerwiński
Marcel Dall'Agnol
Shagnik Das
Adela DePavia
Martin Dietzfelbinger
Sally Dong
Marina Drygala
Bartlomiej Dudek
Vida Dujmovic
Simon Döring
Franziska Eberle
Charilaos Efthymiou
Friedrich Eisenbrand
Jonas Ellert
Christian Engels
Leah Epstein
Chenglin Fan
Sándor Fekete
Hendrik Fichtenberger
Omrit Filtser
Eldar Fischer
Krzysztof Fleszar
Emily Fox
Zachary Friggstad
Yotam Gafni
Ruiquan Gao
Leszek Gasieniec
Ran Gelles
Evangelia Gergatsouli
Mehrdad Ghadiri
Arka Ghosh
Amin Shiraz Gilani
Petr Golovach
Egor Gorbachev
Dishant Goyal
Joshua Grochow
Quentin Guilmant
Mohit Jayanti Gurumukhani
Inge Li Gørtz
Sean Hallgren
Kathrin Hanauer
Vojtech Havlicek
Qizheng He
Irene Heinrich
Karl Heuer
Richard Hladík
Matty Hoban
Charlotte Hoffmann

Piotr Hofman
Gary Hoppenworth
Pavel Hrubes
Brice Huang
Christopher Hugenroth
Tanmay Inamdar
Palak Jain
Ragesh Jaiswal
Petr Jancar
Klaus Jansen
Haotian Jiang
Yonggang Jiang
Ziyang Jin
Tobias Kaiser
John Kallaugher
Amin Karamlou
Sushrut Karmalkar
Joost-Pieter Katoen
Leon Kellerhals
Sanjeev Khanna
Robbie King
David Klaška
Boris Klemz
Katharina Klost
Alexander Knop
Yusuke Kobayashi
Jochen Koenemann
Hanna Komlos
Vasilis Kontonis
Tuukka Korhonen
Ivan Adrian Koswara
Matt Kovacs-Deak
Wiktor Kuchta
Shubhang Kulkarni
Andrey Kupavskii
O-Joung Kwon
Noleen Köhler
Bundit Laekhanukit
Martin Lange
Alexandra Lassota
Hung Le
Engel Lefaucheux
Jérôme Leroux
Asaf Levin
Bo Li
Jingwei Li
Shi Li
Yong Li

Felix Hommelsheim
Pierre Hosteins
Jun-Ting Hsieh
Chien-Chung Huang
Mikael Møller Høgsgaard
Radu Iosif
Pallavi Jain
Manuel Jakob
Wojciech Janczewski
Rajesh Jayaram
Shaofeng H.-C. Jiang
Ce Jin
Antoine Joux
Naonori Kakimura
Makoto Kanazawa
Neel Karia
Petteri Kaski
Telikepalli Kavitha
Dominik Kempa
Emanuel Kieronski
Evangelos Kipouridis
Pieter Kleer
Denis Kleyko
Alexander Knapp
Dušan Knop
Laura Vargas Koch
Gillat Kol
Christian Komusiewicz
Swastik Kopparty
Maria Kosche
Raoul Koudijs
Laszlo Kozma
Ariel Kulik
Mrinal Kumar
Greg Kuperberg
Jan Kynčl
Oded Lachish
Victor Lagerkvist
John Lapinskas
Rustam Latypov
Euiwoong Lee
Karoliina Lehtinen
Amit Levi
Roie Levin
Huan Li
Lawrence Li
Wenzheng Li
Yuhao Li

Chih-Duo Hong
Mathieu Hoyrup
Sihuang Hu
Neng Huang
Hannes Ihalainen
Taisuke Izumi
Rhea Jain
Arun Jambulapati
Duri Andrea Janett
Artur Jeż
Shunhua Jiang
Wenyu Jin
Marcin Jurdzinski
Iden Kalemaj
Ahmet Kara
Toghrul Karimov
Adam Kasperski
Phillip Keldenich
Arindam Khan
Zachary Kincaid
Peter Kiss
Kim-Manuel Klein
Max Klimm
Paul Knappe
Jakob Bæk Tejs Knudsen
Florent Koechlin
Ilan Komargodski
Athanasios Konstantinidis
Viktoriia Korchemna
Evangelos Kosinas
Martin Koutecky
Andrei Krokhin
Pooja Kulkarni
Nikhil Kumar
Martin Kurečka
Chris Köcher
Aditi Laddha
Michael Lampis
Alberto Larrauri
Michel Laurent
Jasper C.H. Lee
Christophe Lenté
Reut Levi
Nathan Lhote
Jason Li
Ray Li
Xingjian Li
Ya-Chun Liang

Jyun-Jie Liao
Ting-Chun Lin
Chun-Hung Liu
Siyue Liu
Yupan Liu
Guang Hao Low
Will Ma
James C. A. Main
Frederik Mallmann-Trenn
Richard Mandel
Mathieu Mari
Corto Mascle
Davide Mazzali
Simon Meierhans
Darya Melnyk
Andras Meszaros
Joseph Mitchell
Matthias Mnich
Benjamin Monmege
Tomoyuki Morimae
Amer Mouawad
Shay Mozes
Sayan Mukherjee
Aniket Murhekar
Tobias Mömke
Chaitanya Nalam
Anand Natarajan
Yasamin Nazari
Yakov Nekrich
Stefan Neumann
Hoai-An Nguyen
Joris Nieuwveld
Nicolas Nisse
André Nusser
Jack O'Connor
Eunjin Oh
Yoshio Okamoto
Tim Ophelders
Ly Orgo
Hussien Othman
Max Ovsiankin
Ioannis Panageas
Debmalya Panigrahi
Sewon Park
Dhrumil Patel
Angelos Pelecanos
Will Perkins
Giovanni Pighizzini
Bingkai Lin
Alexander Lindermayr
Mingmou Liu
Yang Liu
William Lochet
Xinhang Lu
Matthew Maat
Konstantin Makarychev
Guillaume Malod
Quentin Manière
Nicolas Markey
Elvira Mayordomo
Fionn Mc Inerney
Johannes Meintrup
Arturo Merino
Raphael Meyer
Parth Mittal
Sidhanth Mohanty
Fabrizio Montecchiani
Pat Morin
David Mount
Anna Mpanti
Tamalika Mukherjee
Anca Muscholl
Shivam Nadimpalli
Mikito Nanashima
Bento Natura
Jesper Nederlof
Daniel Neuen
Alantha Newman
Huy Nguyen
Milos Nikolic
Damian Niwinski
Pranav Nuti
Maciej Obremski
Pierre Ohlmann
Neil Olver
Michal Opler
George Osipov
Sang-il Oum
Joseph Paat
Anurag Pandey
Fahad Panolan
Nikos Parotsidis
Shyamal Patel
Vincent Penelle
Daniela Petrisan
Jakob Piribauer
Tao Lin
Allen Liu
Sihan Liu
Yanyi Liu
Bruno Loff
Christof Löding
Andreas Maggiori
Yury Makarychev
Nikhil Mande
Naren Manoj
Barnaby Martin
Filip Mazowiecki
Andrew McGregor
Nikolaos Melissinos
Ian Mertz
Pranabendu Misra
Masayuki Miyamoto
Hendrik Molter
Ryuhei Mori
Ron Mosenzon
Tamer Mour
Anish Mukherjee
Wolfgang Mulzer
Richard Mycroft
Viswanath Nagarajan
Shyam Narayanan
Inbal Livni Navon
Ofer Neiman
Eike Neumann
Hung Ngo
Lê Thành Dũng Nguyễn
Chinmay Nirkhe
Jakob Nogler
Zeev Nutov
Andy Oertel
Argyris Oikonomou
Tim Oosterwijk
Jakub Opršal
Piotr Ostropolski-Nalewaja
Xiating Ouyang
Alexandru Paler
Shuo Pang
Irene Parada
Anat Paskin-Cherniavsky
Ami Paz
Binghui Peng
Canh Pham
Maciej Piróg

Thanasis Pittas
Piotr Polesiuk
Aaron Potechin
Maximilian Probst
Manish Purohit
Luowen Qian
Ritam Raha
Fariba Ranjbar
Abhishek Rathod
Victor Oliveira Reis
Tatiana Rocha Avila
Dana Ron
Peter Rossmanith
Aviad Rubinstein
Janosch Ruff
Paweł Rzążewski
Kunihiko Sadakane
Mohammad Salavatipour
Sylvain Salvati
Ramprasad Saptharishi
Igal Sason
Rahul Savani
Kevin Schewior
Jens Schlöter
Daniel R. Schmidt
Pascal Schweitzer
Lia Schütze
Peter Selinger
Liren Shan
Alexander Sherstov
Xinkai Shu
Sebastian Siebertz
Abhishek Kr Singh
George Skretas
Dmitry Sokolov
Mehdi Soleimanifar
Thomas Soullard
Daniel Stefankovic
Hsin-Hao Su
Warut Suksompong
Varun Suriyanarayana
John Sylvester
Ewin Tang
Andrzej Tarlecki
Viet Cuong Than
K. S. Thejaswini
Mikkel Thorup
Kabir Tomer

Madhusudhan Reddy Pittu
Gleb Polevoy
Amaury Pouly
Kirk Pruhs
David Purser
Akbar Rafiey
Vijayaragunathan Ramamoorthi
Sujit Rao
Kavya Ravichandran
Kilian Risse
Liam Roditty
Will Rosenbaum
Jurriaan Rot
Mikhail Rudoy
Zhang Ruilong
Heiko Röglin
Irmak Saglam
Ville Salo
Arnaud Sangnier
Thatchaphol Saranurak
David Saulpic
Philipp Schepper
Šimon Schierreich
Markus L. Schmid
Sylvain Schmitz
Chris Schwiegelshohn
Adam Sealfon
Mark Sellke
Changpeng Shao
Kshiteej Sheth
Sudarshan Shyam
Jamie Sikora
Apoorv Vikram Singh
Michał Skrzypczak
Marek Sokołowski
Zhuoqing Song
Karteek Sreenivasaiah
Rafał Stefański
Sathyawageeswar Subramanian
Aurelio Sulser
Akira Suzuki
Toru Takisaka
Xueyan Tang
Jakub Tarnawski
Sharma V. Thankachan
Anthony Thomas
Cong Tian
Csaba Toth

Vladimir Podolskii
Tristan Pollner
John Power
Krišjānis Prūsis
Edward Pyne
Sharath Raghvendra
C Ramya
Jean-Francois Raskin
Vojtech Rehak
Peter Robinson
Mohammad Roghani
Benjamin Rossman
Arman Rouhani
Atri Rudra
Ignaz Rutter
Karthik C. S.
Jared Saia
Kai Salomaa
Raimundo Saona Urmeneta
Ankita Sarkar
Saket Saurabh
Sven Schewe
Aaron Schild
Todd Schmid
Jason Schoeters
Stefan Schwoon
Igor Sedlar
Rik Sengupta
Eklavya Sharma
Devansh Shringi
Anastasios Sidiropoulos
Kirill Simonov
Mohit Singh
Friedrich Slivovsky
Federico Soldà
José A. Soto
Aleksa Stankovic
Donald Stull
Ondrej Suchy
Xiaorui Sun
Chaitanya Swamy
Zihan Tan
Erasmo Tani
Sébastien Tavenas
Neil Thapen
Clayton Thomas
Konstantin Tikhomirov
Noam Touitou

Vera Traub
Ta-Wei Tu
Marc Uetz
Mihir Vahanwala
Virginia Vassilevska Williams
Moritz Venzin
José Verschae
Emmanouil Vlatakis
Thuy Duong Vuong
Jana Wagemaker
David Wajc
Stefan Walzer
Qisheng Wang
Rémi Watrigant
Alex Wein
Leo Wennmann
Sebastian Wiederrecht
Virginia Vassilevska Williams
Sampson Wong
Michał Wrona
Xudong Wu
Michalis Xefteris
Haifeng Xu
Jie Xue
Yongjie Yang
Penghui Yao
Taisuke Yasuda
Longhui Yin
Emre Yolcu
Huacheng Yu
Konstantin Zabarnyi
Chenyi Zhang
Hengjie Zhang
Ruizhe Zhang
Yibin Zhao
Weiqiang Zheng
Peilin Zhong
Samson Zhou
Marius Zimand
Goran Zuzic
Ronald de Haan
Jesse van Rhijn
Thijs van der Horst
Kenny Štorgel
Gilles Tredan
Malte Tutas
Jara Uitto
Manlio Valenti
Daniel Vaz
Oleg Verbitsky
Alexandre Vigny
Tjark Vredeveld
László Végh
Friedrich Wagner
Bartosz Walczak
Daochen Wang
Zhaozi Wang
Adam Bene Watts
Omri Weinstein
Klaus Wich
Marcus Wilhelm
Karl Wimmer
David R. Wood
Kaiyu Wu
Karol Węgrzycki
Zoe Xi
Yinzhan Xu
Anshu Yadav
Tal Yankovitz
Geva Yashfe
Guanghao Ye
Yitong Yin
Youngho Yoo
Nengkun Yu
Nikos Zarifis
Daniel Zhang
Peng Zhang
Tianyi Zhang
Yiming Zhao
Chenyang Zhong
Hang Zhou
Pawel Zielinski
Alexander Zlokapa
Uri Zwick
Ronald de Wolf
Geert van Wordragen
Tom van der Zanden
Stanislav Živný
Elias Tsigaridas
Nikos Tzevelekos
Seeun William Umboh
Pierre Vandenhove
Yde Venema
Victor Verdugo
Marc Vinyals
Hoa Vu
Maximilian Vötsch
Erik Waingarten
Nathan Wallheimer
Di Wang
Julian Wargalla
Hao-Ting Wei
Philip Wellnitz
Piotr Wieczorek
Gunnar Wilken
Petra Wolf
Marcin Wrochna
Xuan Wu
Michał Włodarczyk
Chao Xu
Zixuan Xu
Takashi Yamakawa
Mihalis Yannakakis
Reem Yassawi
Mingquan Ye
Sorrachai Yingchareonthawornchai
Yuichi Yoshida
Yuancheng Yu
Rico Zenklusen
Fred Zhang
Rachel Zhang
Yuhao Zhang
Da Wei Zheng
Mingxian Zhong
Renfei Zhou
Wieslaw Zielonka
Wiktor Zuba
Mark de Berg
Franck van Breugel
Ivor van der Hoog
Aleksander Łukasiewicz

List of Authors

Fateme Abbasi (5, 6)
University of Wrocław, Poland

Elie Abboud (7)
Department of Computer Science,
University of Haifa, Israel

Pritam Acharya (8)
Department of Mathematics, Indian Institute of Science Education and Research Pune, India

Rohan Acharya (124)
University of Warwick, Coventry, UK

Marek Adamczyk (5)
University of Wrocław, Poland

C. Aiswarya (125)
Chennai Mathematical Institute, India;
CNRS, ReLaX, IRL 2000, Chennai, India

Shyan Akmal (9)
MIT, EECS and CSAIL, Cambridge, MA, USA

Emile Anand (10)
Caltech, Pasadena, CA, USA

Konrad Anand (11)
School of Mathematical Sciences, Queen Mary University of London, London, UK

Noel Arteche (12)
Lund University, Sweden;
University of Copenhagen, Denmark

Srinivasan Arunachalam (13)
IBM Quantum, Thomas J Watson Research Center, Yorktown Heights, NY, USA

Vikraman Arvind (14)
The Institute of Mathematical Sciences (HBNI), Chennai, India;
Chennai Mathematical Institute, Siruseri, Kelambakkam, India

Yossi Azar (15)
School of Computer Science,
Tel Aviv University, Israel

Sandip Banerjee (6)
IDSIA, USI-SUPSI, Lugano, Switzerland

Omkar Baraskar (16)
University of Waterloo, Canada

Surender Baswana (17)
Department of Computer Science & Engineering, IIT Kanpur, India

MohammadHossein Bateni (18)
Google Research, New York, NY, USA

Pascal Baumann (126)
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Soheil Behnezhad (19, 121)
Northeastern University, Boston, MA, USA

Djamal Belazzougui (20)
CAPA, DTISI, Centre de Recherche sur l'Information Scientifique et Technique, Algiers, Algeria

Shalev Ben-David (21)
Institute for Quantum Computing, University of Waterloo, Canada

Michael Benedikt (127)
University of Oxford, UK

Massimo Benerecetti (128)
Università di Napoli Federico II, Italy

Matthias Bentert (22)
University of Bergen, Norway

Koustav Bhanja (17)
Department of Computer Science & Engineering, IIT Kanpur, India

Somnath Bhattacharjee (24)
Chennai Mathematical Institute, India

Chiranjib Bhattacharyya (25)
Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India

Sujoy Bhore (8)
Department of Computer Science and Engineering, Indian Institute of Technology Bombay, India

Andreas Björklund (26)
IT University of Copenhagen, Denmark

Manon Blanc (129)
Institut Polytechnique de Paris, Ecole Polytechnique, LIX, 91128 Palaiseau Cedex, France;
Université Paris-Saclay, LISN, 91190 Gif-sur-Yvette, France

Markus Bläser (24)
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Greg Bodwin (27, 28)
Computer Science and Engineering, University of Michigan, Ann Arbor, MI, USA

Mikołaj Bojańczyk (130)
University of Warsaw, Poland

Narek Bojikian (29)
Humboldt-Universität zu Berlin, Germany

Itai Boneh (30)
Reichman University, Herzliya, Israel; University of Haifa, Israel

Édouard Bonnet (31)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Michaela Borzechowski (32)
Department of Mathematics and Computer Science, Freie Universität Berlin, Germany

Miguel Bosch-Calvo (5)
IDSIA, USI-SUPSI, Lugano, Switzerland

Marin Bougeret (33)
LIRMM, Université de Montpellier, CNRS, France

Olivier Bournez (129)
Institut Polytechnique de Paris, Ecole Polytechnique, LIX, 91128 Palaiseau Cedex, France

Laura Bozzelli (128)
Università di Napoli Federico II, Italy

Jarosław Byrka (5, 6)
University of Wrocław, Poland

Kristóf Bérczi (23)
MTA-ELTE Matroid Optimization Research Group and HUN-REN-ELTE Egerváry Research Group, Department of Operations Research, Eötvös Loránd University, Budapest, Hungary

Barış Can Esmer (34)
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany; Saarbrücken Graduate School of Computer Science, Saarland Informatics Campus, Germany

Arnaud Carayol (131)
Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France

Charlie Carlson (35)
Department of Computer Science, University of California Santa Barbara, CA, USA

Maxime Cautrès (36)
Université Grenoble Alpes, CEA-Léti, F-38054 Grenoble, France; École Normale Supérieure de Lyon, F-69007 Lyon, France

Keren Censor-Hillel (37)
Department of Computer Science, Technion, Haifa, Israel

Parinya Chalermsook (6, 38)
Aalto University, Finland

Karthekeyan Chandrasekaran (23)
University of Illinois, Urbana-Champaign, IL, USA

Panagiotis Charalampopoulos (39)
School of Computing and Mathematical Sciences, Birkbeck, University of London, UK

Lucien Charamond (131)
Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France

Shiri Chechik (40, 41, 42)
Tel Aviv University, Israel

Chung Shue Chen (43)
Nokia Bell Labs, Nozay, France

Junjie Chen (44)
City University of Hong Kong, Hong Kong, China

Li Chen (45)
Carnegie Mellon University, Pittsburgh, PA, USA

Yu Chen (46, 47)
EPFL, Lausanne, Switzerland

Dmitry Chistikov (132)
Centre for Discrete Mathematics and its Applications (DIMAP) & Department of Computer Science, University of Warwick, Coventry, UK

Nathan Claudet (36)
Inria Mocqua, LORIA, CNRS, Université de Lorraine, F-54000 Nancy, France

Andrei Constantinescu (48)
ETH Zürich, Switzerland

Mónika Csikós (49)
Université Paris Cité, IRIF, CNRS UMR 8243 and DI-ENS, Université PSL, France

Artur Czumaj (50)
Department of Computer Science, University of Warwick, Coventry, UK

Clément Dallard (51)
Department of Informatics, University of Fribourg, Switzerland

Mohan Dantam (133)
School of Informatics, University of Edinburgh, UK

Ewan Davies (35)
Department of Computer Science, Colorado State University, Fort Collins, CO, USA

Sami Davies (52)
Department of EECS and Simons Institute, University of California at Berkeley, CA, USA

Anuj Dawar (1)
Department of Computer Science and Technology, University of Cambridge, UK

Arthur Azevedo de Amorim (157)
Rochester Institute of Technology, NY, USA

Élie de Panafieu (43)
Nokia Bell Labs, Nozay, France

Tijn de Vos (58)
Department of Computer Science, University of Salzburg, Austria

Argyrios Deligkas (53)
Department of Computer Science, Royal Holloway, University of London, Egham, UK

Holger Dell (54)
Goethe University Frankfurt, Germany; IT University of Copenhagen and Basic Algorithms Research Copenhagen (BARC), Denmark

Chengyuan Deng (27)
Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Agrim Dewan (16)
Indian Institute of Science, Bengaluru, India

Laxman Dhulipala (18)
University of Maryland, College Park, MD, USA

Konstantinos Dogeas (55)
Department of Computer Science, Durham University, UK

Ilan Doron-Arad (56, 57)
Computer Science Department, Technion, Haifa, Israel

Michal Dory (58)
University of Haifa, Israel

Amina Doumane (135)
Plume, LIP, CNRS, ENS de Lyon, France

Pål Grønås Drange (22)
University of Bergen, Norway

Aditi Dudeja (59)
University of Salzburg, Austria

Julien Duron (31)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Arkopal Dutt (13)
IBM Quantum, IBM Research Cambridge, MA, USA

Pranjal Dutta (24)
School of Computing, National University of Singapore, Singapore

Julian Dörfler (134)
Saarland Informatics Campus (SIC), Saarbrücken Graduate School of Computer Science, Saarland University, Germany

Talya Eden (60)
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Eduard Eiben (53)
Department of Computer Science, Royal Holloway, University of London, Egham, UK

Edith Elkind (2)
University of Oxford, UK; Alan Turing Institute, London, UK

Thomas Erlebach (55)
Department of Computer Science, Durham University, UK

Francisco Escudero Gutiérrez (13)
CWI & QuSoft, Amsterdam, The Netherlands

Tomer Even (37)
Department of Computer Science, Technion, Haifa, Israel

John Fearnley (32)
Department of Computer Science, University of Liverpool, UK

Andreas Emil Feldmann (61)
Department of Computer Science, University of Sheffield, UK

Weiming Feng (11, 62)
Institute for Theoretical Studies, ETH Zürich, Switzerland

Yuda Feng (63)
Department of Computer Science and Technology, Harbin Institute of Technology, Heilongjiang, China

Nick Fischer (64)
Weizmann Institute of Science, Rehovot, Israel

Cella Florescu (65)
ETH Zürich, Switzerland

Jacob Focke (34)
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Fedor V. Fomin (22, 51)
University of Bergen, Norway

Sebastian Forster (58)
Department of Computer Science, University of Salzburg, Austria

Florent Foucaud (66)
Université Clermont Auvergne, CNRS, Mines Saint-Étienne, Clermont Auvergne INP, LIMOS, 63000 Clermont-Ferrand, France

Graham Freifeld (11)
School of Informatics, University of Edinburgh, UK

Yuxi Fu (136)
BASICS, Shanghai Jiao Tong University, China

Marco Gaboardi (157)
Boston University, MA, USA

Ameet Gadekar (6)
Bar-Ilan University, Ramat-Gan, Israel

Jakub Gajarský (137)
University of Warsaw, Poland

Esther Galby (66, 67)
Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden;
University of Gothenburg, Sweden

Paul Gallot (138)
Universität Bremen, Germany

Robert Ganian (53, 68)
Algorithms and Complexity Group, TU Wien, Austria

Guichen Gao (50)
School of Computer Science, Peking University, Beijing, China

Jie Gao (27)
Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Serge Gaspers (69)
UNSW Sydney, Australia

Paweł Gawrychowski (39)
Institute of Computer Science, University of Wrocław, Poland

Mehrdad Ghadiri (10)
MIT, Cambridge, MA, USA

Sevag Gharibian (70)
Department of Computer Science and Institute for Photonic Quantum Systems (PhoQS), Paderborn University, Germany

Samah Ghazawi (39)
Department of Computer Science, University of Haifa, Israel;
Department of Software Engineering, Braude, College of Engineering, Karmiel, Israel

Prantar Ghosh (71)
Georgetown University, Washington, DC, USA

Yiannis Giannakopoulos (72)
School of Computing Science, University of Glasgow, UK

Matan Gilboa (73)
University of Oxford, UK

Raphael Douglas Giles (139)
The University of Melbourne, Australia

Shay Golan (30)
Reichman University, Herzliya, Israel;
University of Haifa, Israel

Guy Goldberg (74)
Weizmann Institute of Science, Rehovot, Israel

Petr A. Golovach (22, 51)
University of Bergen, Norway

Spencer Gordon (32)
Department of Computer Science, University of Liverpool, UK

Kishen N. Gowda (18)
University of Maryland, College Park, MD, USA

Fabrizio Grandoni (5)
IDSIA, USI-SUPSI, Lugano, Switzerland

Lucas Gretta (75)
University of California, Berkeley, CA, USA

Mario Grobler (76)
University of Bremen, Germany

Carla Groenland (77)
Delft Institute of Applied Mathematics, The Netherlands

Martin Grohe (78)
RWTH Aachen University, Germany

Alexander Grosz (72)
School of Computation, Information and Technology, Technical University of Munich, Germany

Quentin Guilmant (140)
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Heng Guo (11, 62)
School of Informatics,
University of Edinburgh, UK

Aaryan Gupta (8)
Department of Computer Science and Engineering, Indian Institute of Technology Bombay, India

Manoj Gupta (38)
IIT Gandhinagar, India

Leonid Gurvits (79)
City College New York, NY, USA

Maximilian Probst Gutenberg (65)
ETH Zürich, Switzerland

Roland Guttenberg (141, 156)
Technical University of Munich, Germany

Daniel Găină (143)
Kyushu University, Fukuoka, Japan

Christoph Haase (142)
Department of Computer Science,
University of Oxford, UK

Mohammad T. HajiAghayi (80)
University of Maryland,
College Park, MD, USA

Sariel Har-Peled (81)
Department of Computer Science, University of Illinois, Urbana, IL, USA

Elfarouk Harb (81)
Department of Computer Science, University of Illinois, Urbana, IL, USA

Go Hashimoto (143)
Kyushu University, Fukuoka, Japan

Hamed Hatami (82)
School of Computer Science, McGill University, Montreal, Canada

Klaus Heeger (83)
Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Sophia Heimann (84)
Research Institute for Discrete Mathematics, University of Bonn, Germany

Diptarama Hendrian (89)
Tokyo Medical and Dental University, Japan

Danny Hermelin (83)
Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva, Israel

D. Ellis Hershkowitz (18)
Brown University, Providence, RI, USA

Shuichi Hirahara (85)
National Institute of Informatics, Tokyo, Japan

Hung P. Hoang (84)
Algorithms and Complexity Group,
Faculty of Informatics, TU Wien, Austria

Gary Hoppenworth (27, 28)
Computer Science and Engineering, University of Michigan, Ann Arbor, MI, USA

Kaave Hosseini (82)
Department of Computer Science, University of Rochester, NY, USA

Stefan Hougardy (84)
Research Institute for Discrete Mathematics and Hausdorff Center for Mathematics, University of Bonn, Germany

Kevin Hua (87)
University of Michigan, Ann Arbor, MI, USA

Samuel Humeau (135)
Plume, LIP, CNRS, ENS de Lyon, France

Florian Hörsch (86)
Algorithms and Complexity Group, CISPA, Saarbrücken, Germany

Tomohiro I (89)
Kyushu Institute of Technology, Japan

Christian Ikenmeyer (134)
University of Warwick, Coventry, UK

András Imolay (86)
MTA-ELTE Matroid Optimization Research Group, Department of Operations Research, ELTE Eötvös Loránd University, Budapest, Hungary

Tanmay Inamdar (88)
Indian Institute of Technology Jodhpur, India

Kento Iseri (89)
Kyushu Institute of Technology, Japan

Vincent Jackson (139)
The University of Melbourne, Australia

Pallavi Jain (88)
Indian Institute of Technology Jodhpur, India

Bart M. P. Jansen (33)
Eindhoven University of Technology, The Netherlands

Rajesh Jayaram (18, 90)
Google Research, New York, NY, USA

Agastya Vibhuti Jha (91)
École polytechnique fédérale de Lausanne, Switzerland

Wanchote Jiamjitrak (38)
University of Helsinki, Finland

Shaofeng H.-C. Jiang (50, 92)
School of Computer Science, Peking University, Beijing, China

Ce Jin (93, 94)
MIT, Cambridge, MA, USA

Pushkar S. Joglekar (14)
Vishwakarma Institute of Technology, Pune, India

Marcin Jurdziński (124)
University of Warwick, Coventry, UK

Frank Kammer (55)
THM, University of Applied Sciences Mittelhessen, Gießen, Germany

Jonas Kamminga (70)
Department of Computer Science and Institute for Photonic Quantum Systems (PhoQS), Paderborn University, Germany

Iyad Kanj (53)
School of Computing, DePaul University, Chicago, IL, USA

Ravindran Kannan (25)
Department of Operations Research, Carnegie Mellon University, Pittsburgh, USA

Haim Kaplan (120)
Tel Aviv University, Israel

Michael Kapralov (46, 93)
EPFL, Lausanne, Switzerland

Adam Karczmarz (95)
University of Warsaw, Poland;
IDEAS NCBR, Warsaw, Poland

Petteri Kaski (26)
Aalto University, Finland

Yasushi Kawase (96)
University of Tokyo, Japan

Pavol Kebis (144)
Department of Computer Science, University of Oxford, UK

Peter Keevash (43)
Mathematical Institute, University of Oxford, UK

George Kenison (145)
School of Computer Science and Mathematics, Liverpool John Moores University, UK

Sean Kennedy (43)
Nokia Bell Labs, Ottawa, Canada

Yotam Kenneth (97)
Weizmann Institute of Science, Rehovot, Israel

Eren Keskin (126)
TU Braunschweig, Germany

Arindam Khan (8)
Department of Computer Science and Automation, Indian Institute of Science Bengaluru, India

Erfan Khaniki (12)
Institute of Mathematics of the Czech Academy of Sciences, Prague, Czech Republic

Sanjeev Khanna (98)
School of Engineering and Applied Sciences, University of Pennsylvania, Philadelphia, PA, USA

Liana Khazaliya (66)
Technische Universität Wien, Austria

Kamyar Khodamoradi (6)
University of Regina, Canada

Sandra Kiefer (156)
University of Oxford, UK

Tamás Király (23)
MTA-ELTE Matroid Optimization Research Group and HUN-REN-ELTE Egerváry Research Group, Department of Operations Research, Eötvös Loránd University, Budapest, Hungary

Sándor Kisfaludi-Bak (67)
Department of Computer Science, Aalto University, Finland

Nathan Klein (79)
Institute for Advanced Study, Princeton, NJ, USA

Boris Klemz (99)
Universität Würzburg, Germany

Yusuke Kobayashi (100)
Research Institute for Mathematical Sciences, Kyoto University, Japan

Alexandra Kolla (35)
Computer Science and Engineering, University of California Santa Cruz, CA, USA

Tsvi Kopelowitz (101)
Bar-Ilan University, Ramat-Gan, Israel

Tuukka Korhonen (22, 51)
University of Bergen, Norway

Ariel Korin (101)
Bar-Ilan University, Ramat-Gan, Israel

Dariusz R. Kowalski (80)
School of Computer and Cyber Sciences, Augusta University, GA, USA

Stefan Kratsch (29)
Humboldt-Universität zu Berlin, Germany

Robert Krauthgamer (50, 97)
Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel

Shankara Narayanan Krishna (142)
Department of Computer Science & Engineering, IIT Bombay, India

Gregory Kucherov (20)
LIGM, CNRS, Université Gustave Eiffel, Marne-la-Vallée, France

Ariel Kulik (56)
Computer Science Department, Technion, Haifa, Israel

Shubhang Kulkarni (23)
University of Illinois, Urbana-Champaign, IL, USA

Soh Kumabe (102)
CyberAgent, Tokyo, Japan

Akash Kumar (91)
Indian Institute of Technology, Bombay, India

Amit Kumar (25)
Department of Computer Science and Engineering, Indian Institute of Technology Delhi, India

Srijita Kundu (21)
Institute for Quantum Computing, University of Waterloo, Canada

William Kuszmaul (103)
Harvard University, Cambridge, MA, USA

Rasmus Kyng (65)
ETH Zürich, Switzerland

Dominik Köppl (89)
University of Yamanashi, Japan

Michael Lampis (61)
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

John Lapinskas (54)
University of Bristol, UK

Alberto Larrauri (146)
Department of Computer Science, University of Oxford, UK

Kasper Green Larsen (104)
Aarhus University, Denmark

Jonathan Leake (79)
University of Waterloo, Canada

Eunou Lee (105)
Korea Institute for Advanced Study, Seoul, South Korea

Engel Lefaucheux (140)
Loria, Nancy, France

Pascal Lenzner (48)
Hasso Plattner Institute, University of Potsdam, Germany

Reut Levi (60)
Efi Arazi School of Computer Science, Reichman University, Herzliya, Israel

Shahar Lewkowicz (15)
School of Computer Science, Tel Aviv University, Israel

Daniel Li (87)
University of Michigan, Ann Arbor, MI, USA

Jerry Zirui Li (69)
James Ruse Agricultural High School, Carlingford, Australia;
UNSW Sydney, Australia

Minming Li (44)
City University of Hong Kong, Hong Kong, China

Shaohua Li (66)
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Shi Li (63, 106)
Department of Computer Science and Technology, Nanjing University, Jiangsu, China

Shuangle Li (107)
State Key Laboratory for Novel Software Technology, Nanjing University, China

Xin Li (108)
Johns Hopkins University, Baltimore, MD, USA

Bingkai Lin (107)
State Key Laboratory for Novel Software Technology, Nanjing University, China

Yuwei Liu (107)
BASICS, Shanghai Jiao Tong University, China

Vasilis Livanos (81)
Department of Industrial Engineering, University of Chile, Santiago, Chile

Bruno Loff (147)
LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal

Daniel Lokshtanov (88)
University of California Santa Barbara, CA, USA

Yaowei Long (109)
University of Michigan, Ann Arbor, MI, USA

Shachar Lovett (82)
Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA, USA

Chia-Hsuan Lu (127)
University of Oxford, UK

Zhenjian Lu (110)
University of Warwick, UK

Florian Luca (144)
Mathematics Division, Stellenbosch University, Stellenbosch, South Africa

Stephanie Maaz (76)
University of Waterloo, Canada

Khushraj Madnani (142)
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Sepideh Mahabadi (93)
Microsoft Research-Redmond, WA, USA

Mikhail Makarov (46)
EPFL, Lausanne, Switzerland

Yury Makarychev (111)
Toyota Technological Institute at Chicago, IL, USA

Sebastian Maneth (138)
Universität Bremen, Germany

Isja Mannens (77)
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Alessio Mansutti (132)
IMDEA Software Institute, Madrid, Spain

Amaldev Manuel (125)
Indian Institute of Technology Goa, India

Dániel Marx (6, 34, 67)
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Corto Mascle (156)
LaBRI, Université de Bordeaux, France

Richard Mayr (133)
School of Informatics, University of Edinburgh, UK

Davide Mazzali (46)
EPFL, Lausanne, Switzerland

Fionn Mc Inerney (66)
Technische Universität Wien, Austria

Rose McCarty (137)
School of Mathematics and School of Computer Science, Georgia Institute of Technology, Altanta, GA, USA

Kitty Meeks (54)
University of Glasgow, UK

Nicole Megow (76)
University of Bremen, Germany

Johannes Meintrup (55)
HM, University of Applied Sciences Mittelhessen, Gießen, Germany

Themistoklis Melissourgos (72)
School of Computer Science and Electronic Engineering, University of Essex, UK

Roland Meyer (126)
TU Braunschweig, Germany

Mehdi Mhalla (36)
Université Grenoble Alpes, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France

Martin Milanič (51)
FAMNIT and IAM, University of Primorska, Koper, Slovenia

Om Swostik Mishra (142)
Department of Mathematics, IIT Bombay, India

Slobodan Mitrović (90)
University of California Davis, CA, USA

Ryuhei Mizutani (86)
Department of Mathematical Informatics, Graduate School of Information Science and Technology, The University of Tokyo, Japan

Matthias Mnich (83)
Institute for Algorithms and Complexity, Hamburg University of Technology, Hamburg, Germany

Augusto Modanese (112)
Aalto University, Finland

Fabio Mogavero (128)
Università di Napoli Federico II, Italy

Bratin Mondal (8)
Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India

Benjamin Moseley (52)
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA

William K. Moses Jr. (55)
Department of Computer Science, Durham University, UK

Boris Motik (127)
University of Oxford, UK

Antoine Mottet (148)
Research Group on Theoretical Computer Science, Hamburg University of Technology, Germany

Amer E. Mouawad (76)
American University of Beirut, Lebanon

Shay Mozes (30)
Reichman University, Herzliya, Israel

Saswata Mukherjee (24)
Chennai Mathematical Institute, India

Doron Mukhtar (40)
Tel Aviv University, Israel

Nabil H. Mustafa (49)
Université Sorbonne Paris Nord, Laboratoire LIPN, CNRS 7030, France

Haiko Müller (68)
School of Computing, University of Leeds, UK

Tomáš Nagy (148)
Theoretical Computer Science Department, Jagiellonian University, Kraków, Poland

Keisuke Nakano (138)
Tohoku University, Sendai, Japan

Danupon Nanongkai (3)
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany; KTH Royal Institute of Technology, Stockholm, Sweden

Joseph (Seffi) Naor (57)
Computer Science Department, Technion, Haifa, Israel

Yasamin Nazari (58)
Vrije Universiteit Amsterdam, The Netherlands

Jesper Nederlof (26, 77)
Utrecht University, The Netherlands

Daniel Neuen (78)
University of Regensburg, Germany

Heather Newman (52)
Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, PA, USA

Lê Thành Dũng (Tito) Nguyễn (130)
École normale supérieure de Lyon, France

Koichi Nishimura (96)
CRESCO LTD., Japan

Naoto Ohsaka (85, 113)
CyberAgent, Inc., Tokyo, Japan

Taihei Oki (86)
Department of Mathematical Informatics, Graduate School of Information Science and Technology, The University of Tokyo, Japan

Jan Olkowski (80)
University of Maryland, College Park, MD, USA

Krzysztof Onak (90)
Boston University, USA

Sebastian Ordyniak (68)
School of Computing, University of Leeds, UK

Anthony Ostuni (82)
Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA, USA

Joël Ouaknine (140, 144)
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Max Ovsiankin (111)
Toyota Technological Institute at Chicago, IL, USA

Giacomo Paesani (68)
School of Computing, University of Leeds, UK

Rasmus Pagh (104)
BARC, University of Copenhagen, Denmark

Carlos Palazuelos (13)
Dpto. Análisis Matemático y Matemática Aplicada, Fac. Ciencias Matemáticas, Universidad Complutense de Madrid, Spain; Instituto de Ciencias Matemáticas, Madrid, Spain

Akash Pareek (38)
IIT Gandhinagar, India

Ojas Parekh (105)
Sandia National Laboratories, Albuquerque, NM, USA

Jaewoo Park (87)
University of Michigan, Ann Arbor, MI, USA

Merav Parter (4)
Weizmann Institute of Science, Rehovot, Israel

Christophe Paul (114)
LIRMM, Univ Montpellier, CNRS, Montpellier, France

Simon Perdrix (36)
Inria Mocqua, LORIA, CNRS, Université de Lorraine, F-54000 Nancy, France

Adriano Peron (128)
Università di Trieste, Italy

Giuseppe Persiano (104)
Università di Salerno, Italy;
Google, New York, NY, USA

Charles Peyrat (138)
ENS Paris-Saclay, France

Ján Pich (12)
University of Oxford, UK

Marta Piecyk (77)
Warsaw University of Technology, Poland

Michael Pinsker (148)
Institut für Diskrete Mathematik und Geometrie, Technische Universität Wien, Austria

Toniann Pitassi (104)
Columbia University, New York, NY, USA

Orestis Plevrakis (115)
Department of Computer Science, Princeton University, NJ, USA

Vladimir V. Podolskii (116)
Tufts University, Medford, MA, USA

Aaron Potechin (117)
University of Chicago, IL, USA

Aditya Potukuchi (35)
Department of Electrical Engineering and Computer Science, York University, Toronto, Canada

Damien Pous (135)
Plume, LIP, CNRS, ENS de Lyon, France

Aditya Prakash (124)
University of Warwick, Coventry, UK

Eric Price (75)
University of Texas at Austin, TX, USA

Evangelos Protopapas (114)
LIRMM, Univ Montpellier, CNRS, Montpellier, France

Aaron (Louie) Putterman (98)
School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA

Kent Quanrud (118)
Dept. of Computer Science, Purdue University, West Lafayette, IN, USA

Seyoon Ragavan (115)
Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA, USA

Vijayaragunathan Ramamoorthi (76)
University of Bremen, Germany

M. S. Ramanujan (53)
Department of Computer Science, University of Warwick, Coventry, UK

Rebecca Reiffenhäuser (48)
University of Amsterdam, The Netherlands

Christine Rizkallah (139)
The University of Melbourne, Australia

Liam Roditty (101)
Bar-Ilan University, Ramat-Gan, Israel

Mohammad Roghani (19)
Stanford University, CA, USA

Dana Ron (60)
School of Electrical Engineering, Tel Aviv University, Israel

Noga Ron-Zewi (7)
Department of Computer Science, University of Haifa, Israel

Aviad Rubinstein (19)
Stanford University, CA, USA

Mateusz Rychlicki (68)
School of Computing, University of Leeds, UK

Jakub Rydval (150, 151)
Technische Universität Wien, Austria

Paweł Rzążewski (34, 77)
Warsaw University of Technology, Poland; University of Warsaw, Poland

Wojciech Różowski (149)
Department of Computer Science, University College London, UK

Amin Saberi (19)
Stanford University, CA, USA

Sushant Sachdeva (65, 119)
University of Toronto, Canada

Yaniv Sadeh (120)
Tel Aviv University, Israel

Chandan Saha (16)
Indian Institute of Science, Bengaluru, India

Abhishek Sahu (88)
National Institute of Science, Education and Research Bhubaneswar, HBNI, India

Mohammad Saneian (121)
Northeastern University, Boston, MA, USA

Piotr Sankowski (90)
IDEAS NCBR, University of Warsaw, Poland; MIM Solutions, Warsaw, Poland

Rahul Santhanam (12, 110)
University of Oxford, UK

Thatchaphol Saranurak (87)
University of Michigan, Ann Arbor, MI, USA

Ignasi Sau (33)
LIRMM, Université de Montpellier, CNRS, France

Saket Saurabh (88)
The Institute of Mathematical Sciences, HBNI, Chennai, India;
University of Bergen, Norway

Rahul Savani (32)
The Alan Turing Institute, London, UK;
and Department of Computer Science, University of Liverpool, UK

Valentin Savin (36)
Université Grenoble Alpes, CEA-Léti, F-38054 Grenoble, France

Benjamin Scheidt (152)
Humboldt-Universität zu Berlin, Germany

Niklas Schlomberg (122)
Research Institute for Discrete Mathematics and Hausdorff Center for Mathematics, University of Bonn, Germany

Daniel Schmand (48, 76)
University of Bremen, Germany

Sylvain Schmitz (153)
Université Paris Cité, CNRS, IRIF, Paris, France

Patrick Schnider (32)
Department of Computer Science, ETH Zürich, Switzerland

Tamás Schwarcz (86)
MTA-ELTE Matroid Optimization Research Group, Department of Operations Research, ELTE Eötvös Loránd University, Budapest, Hungary

Lia Schütze (153)
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Andrew Scoones (144)
Department of Computer Science, University of Oxford, UK

Žaneta Semanišinová (151)
Technische Universität Dresden, Germany

Dvir Shabtay (83)
Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Hadas Shachnai (56)
Computer Science Department, Technion, Haifa, Israel

Roohani Sharma (6, 66, 67)
University of Bergen, Norway

Ayumi Shinohara (89)
Tohoku University, Sendai, Japan

Sebastian Siebertz (76)
University of Bremen, Germany

Marie Diana Sieper (99)
Universität Würzburg, Germany

Pulkit Sinha (16)
University of Waterloo, Canada

Mateusz Skomra (147)
LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

Dmitrii Sluch (116)
Nebius, Tel Aviv, Israel

Marcin Smulewicz (95)
University of Warsaw, Poland

Krzysztof Sornat (5)
AGH University, Kraków, Poland

Joachim Spoerhase (6)
University of Sheffield, UK

Mikhail R. Starchak (132)
St. Petersburg State University, Russia

Rafał Stefański (130)
University of Warsaw, Poland

Manuel Stoeckl (71)
Dartmouth College, Hanover, NH, USA

Madhu Sudan (98)
School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA

Hanna Sumita (96)
Tokyo Institute of Technology, Japan

Saina Sunny (125)
Indian Institute of Technology Goa, India

John Sylvester (31)
Department of Computer Science, University of Liverpool, UK

Prafullkumar Tale (66)
Indian Institute of Science Education and Research Pune, India

Tony Tan (127)
University of Liverpool, UK

Zihan Tan (47)
Rutgers University, Piscataway, NJ, USA

Erasmo Tani (111)
University of Chicago, IL, USA

Tatsuya Terao (100)
Research Institute for Mathematical Sciences, Kyoto University, Japan

Guillaume Theyssier (154)
I2M, CNRS, Université Aix-Marseille, France

Dimitrios M. Thilikos (114)
LIRMM, Univ Montpellier, CNRS, Montpellier, France

Stéphan Thomassé (36)
Université de Lyon, École Normale Supérieure de Lyon, UCBL, CNRS, LIP, F-69007 Lyon, France

Anvith Thudi (119)
University of Toronto, Canada

Antoine Tinguely (5)
IDSIA, USI-SUPSI, Lugano, Switzerland

Yuya Uezato (155)
CyberAgent, Inc., Tokyo, Japan;
National Institute of Informatics, Tokyo, Japan

Jalaj Upadhyay (27)
Management Science and Information Systems, Rutgers University, Piscataway, NJ, USA

Anannya Upasana (88)
The Institute of Mathematical Sciences, HBNI, Chennai, India

Danny Vainstein (15)
School of Computer Science, Tel Aviv University, Israel;
Google Research, Tel Aviv, Israel

Ali Vakilian (93)
Toyota Technological Institute at Chicago (TTIC), IL, USA

Steffen van Bergerem (156)
Humboldt-Universität zu Berlin, Germany

Jan van den Brand (10)
Georgia Tech, Atlanta, GA, USA

Giovanna Varricchio (48)
University of Calabria, Rende, Italy

Virginia Vassilevska Williams (9, 28, 37)
MIT, EECS and CSAIL, Cambridge, MA, USA

Pavel Veselý (50)
Computer Science Institute of Charles University, Prague, Czech Republic

Adrian Vetta (43)
McGill University, Montreal, Canada

Nicolas Waldburger (156)
IRISA, Université de Rennes, France

Stefan Walzer (20)
Karlsruhe Institute of Technology, Germany

Chen Wang (27)
Department of Computer Science, Rice University, Houston, TX, USA;
Computer Science and Engineering, Texas A&M University, College Station, TX, USA

Jiaheng Wang (11)
School of Informatics,
University of Edinburgh, UK

Wenqian Wang (92)
School of Electronic, Information and Electrical Engineering, Shanghai Jiao Tong University, China

Yunfan Wang (109)
Tsinghua University, Beijing, China

Simon Weber (32)
Department of Computer Science, ETH Zürich, Switzerland

Chana Weil-Kennedy (156)
IMDEA Software Institute, Madrid, Spain

Oren Weimann (30)
University of Haifa, Israel

Nicole Wein (9, 28)
University of Michigan, Ann Arbor, MI, USA

S. Matthew Weinberg (115)
Department of Computer Science, Princeton University, NJ, USA

Leo Wennmann (64)
Maastricht University, The Netherlands

Sebastian Wiederrecht (114)
Discrete Mathematics Group, Institute for Basic Science, Daejeon, South Korea

Andreas Wiese (8)
Department of Mathematics, Technical University of Munich, Germany

James Worrell (140, 144)
Department of Computer Science,
University of Oxford, UK

Michał Wrona (151)
Jagiellonian University, Kraków, Poland

Hongxun Wu (94)
University of California Berkeley, CA, USA

Zoe Xi (103)
Massachusetts Institute of Technology, Cambridge, MA, USA

Chenyang Xu (106)
Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China

Haifeng Xu (44)
University of Chicago, IL, USA

Zixuan Xu (28)
Massachusetts Institute of Technology, Cambridge, MA, USA

Qizhe Yang (136)
Shanghai Normal University, China

Mingquan Ye (45)
University of Illinois at Chicago, IL, USA

Kevin Yeo (104)
Columbia University, New York, NY, USA;
Google, New York, NY, USA

Sorrachai Yingchareonthawornchai (38)
The Hebrew University of Jerusalem, Israel

Yuichi Yoshida (102, 112)
National Institute of Informatics, Tokyo, Japan

Ryo Yoshinaka (89)
Tohoku University, Sendai, Japan

Kingsley Yung (123)
The Chinese University of Hong Kong, Hong Kong, China

Viktor Zamaraev (31)
Department of Computer Science,
University of Liverpool, UK

Or Zamir (104)
Tel Aviv University, Israel

Georg Zetzsche (126, 142)
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Aaron Zhang (117)
The Voleon Group, Berkeley, CA, USA

Cheng Zhang (157)
Boston University, MA, USA

Daniel J. Zhang (10)
Georgia Tech, Atlanta, GA, USA

Ruilong Zhang (106)
Department of Computer Science, City University of Hong Kong, Hong Kong, China

Tianyi Zhang (40, 41, 42)
Tel Aviv University, Israel

Yubo Zhang (92)
School of Computer Science, Peking University, Beijing, China

Yuhao Zhang (92)
John Hopcroft Center for Computer Science, Shanghai Jiao Tong University, China

Yibin Zhao (119)
University of Toronto, Canada

Yangluo Zheng (136)
BASICS, Shanghai Jiao Tong University, China

Yan Zhong (108)
Johns Hopkins University, Baltimore, MD, USA

Maksim Zhukovskii (31)
Department of Computer Science, University of Sheffield, UK

Song Zuo (44)
Google Research, New York, NY, USA

Jakub Łącki (18, 90)
Google Research, New York, NY, USA

Ionuţ Ţuţu (143)
Simion Stoilow Institute of Mathematics of the Romanian Academy, Bucharest, Romania

Stanislav Živný (146)
Department of Computer Science, University of Oxford, UK

Limits of Symmetric Computation

Anuj Dawar
Department of Computer Science and Technology, University of Cambridge, UK

Abstract

I survey recent work on symmetric computation. A number of strands of work, from logic, circuit complexity, combinatorial optimization and other areas have converged on similar notions of symmetry in computation. This write-up of an invited talk gives a whirlwind tour through the results and pointers to the relevant literature.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Finite Model Theory; Theory of computation → Complexity classes

Keywords and phrases Logic, Complexity Theory, Symmetric Computation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.1

Category Invited Talk

Funding Research funded by UK Research and Innovation (UKRI) under the UK government's Horizon Europe funding guarantee: grant number EP/X028259/1.

Introduction

In the 1980s, descriptive complexity was a new approach to the problems of complexity theory. It carried the hope that methods from logic, particularly finite model theory, could be deployed to settle the difficult questions of complexity. It was one of many promising approaches at the time but the hard problems of complexity proved resistant to all of them. An amusing article from ACM Sigact News in 1996 [24] imagines the many possible titles of a paper announcing a resolution of the P vs.NP question. One of them is through Immerman's approach to descriptive complexity which recasts the question of separting P from NP as the question of separating the expressive power of fixed point logic FP from existential second-order logic on *ordered structures*. This captures the essential gap between the promise of descriptive complexity and its delivery. The methods from finite model theory that it makes available for proving inexpressibility in logics such as FP work well on *unordered* structures. But, the correspondence with complexity classes works well on *ordered* structures.

A more recent viewpoint on this is that the inexpressibility results from finite model theory establish lower bounds for restricted, *symmetric* models of computation. This is exemplified by the results of [2, 21], which show that the logic FPC (fixed-point logic with counting) corresponds to a natural model of *symmetric circuits*. The logic FPC is a natural and powerful logic within P for which unconditional lower bounds have been proved (see [11] for an overview).

Understanding the inexpressibilty methods of descriptive complexity as lower bounds for symmetric models of computation leads to a number of interesting further directions of investigation, which I review in the present talk. In particular, I look at the following directions.

1. We can extend the expressive power of FPC by considering more powerful operations than counting, while remaining within P. These give rise to further notions of symmetric computation, essentially weakening the symmetry restriction. Recently lower bounds have been obtained for these as well.
2. We can investigate what efficient algorithms can be implemented within these symmetric models. It turns out that many natural algorithmic methods are symmetric and therefore subject to the lower bound methods of descriptive complexity.


3. We can look at symmetry as it arises in other models of computation, and see to what extent *asymmetry* is used as a resource. I illustrate this with two cases, that of linear programs and of arithmetic circuits.
4. We relate the lower bounds to other classifications of problems according to their symmetries. In particular, the very successful classification of constraint satisfaction problems into tractable and intractable ones is based on a different, but related notion of symmetry.

In the following, after a brief introduction of the relationship between FPC and symmetric circuits, I give a summary of each of the above directions. The main aim is to provide pointers to the relevant literature and to identify fertile directions for future work. Formal definitions, statements of the results and proofs may be found in the cited literature.

FPC and symmetric circuits

The symmetry restriction we are considering is most clearly explained in a circuit model of computation. Recall that any decision problem $L \subseteq \{0,1\}^*$ can be seen as a family of Boolean functions $(f_n)_{n\in\omega}$ where $f_n : \{0,1\}^n \to \{0,1\}$. Moreover, each function f_n can be described by a circuit C_n: a directed acyclic graph with n inputs and gates labelled by Boolean functions from some fixed basis, such as $\wedge$, $\vee$, $\neg$ or extensions with a majority gate or threshold gates. The language L is decidable in polynomial time if, and only if, it has such a family of circuits that is P-uniform. In other words, the circuit C_n can be constructed in poly(n) time. One fact to note here is that when we consider what Boolean functions we can use in the basis, we are restricted to *symmetric* functions. These are functions whose value depends only on the number of 1s and 0s in the input and not on the order in which the inputs appear. The functions $\wedge$, $\vee$, $\neg$, majority and threshold all have this property. This is necessary in order for the circuit to be defined as a DAG with no further structure on it, such as an order of the gates.

When we consider decision problems on structures such as graphs, we are typically interested in deciding properties that are invariant under graph isomorphisms. A key perspective of descriptive complexity is that we consider formalisms in which only such properties can be expressed. In the context of circuit complexity, we can consider a property P of (directed) graphs as being given by a family of Boolean functions $(p_n)_{n\in\omega}$ where $p_n : \{0,1\}^{n\times n} \to \{0,1\}$ takes the adjacency matrix of a graph and maps it to 1 just in case the graph has property P. Such Boolean functions have natural symmetries in the sense that for any permutation π of $[n]$, we have $p_n(x) = p_n(x^\pi)$ where the string x^π is defined to have $x^\pi_{ij} = x_{\pi(i)\pi(j)}$. We say that a circuit C computing p_n is *symmetric* if this action of permutations $\pi \in S_n$ on the inputs always extends to an automorphism of the circuit C itself.

The key result from [2] is that a graph property is definable in FPC (for a formal definition of FPC, I refer the reader to [11]) if, and only if, it is decidable by a P-uniform family of symmetric circuits. The result there is stated for circuits with threshold gates (indeed, just a majority gate would suffice) but, as observed in [21], adding further symmetric Boolean functions to the basis does not change it. Thus, we get quite a robust notion of symmetric polynomial-time computation and it corresponds exactly to definability in FPC. Moreover, while I have stated it here for graphs, it is proved more generally for finite relational structures.

The important aspect of the connection between FPC definability and decidability with symmetric circuits is that we have methods for proving inexpressibility results in FPC and these yield proofs of unconditional lower bounds for symmetric circuits. In particular, FPC definable classes of graphs exhibit stronger invariance conditions than just being closed under

isomorphism. This is made precise by considering the Weisfeiler-Leman (WL) equivalences. For a precise definition of the k-dimensional Weisfeiler-Leman equivalence, see [17, Sec. 2]. Here we just note that this is, for any fixed k, a coarser relation than graph isomorphism. The connection with FPC definability comes from the fact that for any property P of graphs that is definable by a formula of FPC, there is a constant k such that P is invariant under k-dimensional WL equivalence.

Cai et al. [10] first showed that there is no fixed value of k for which k-dimensional WL equivalence is the same as isomorphism, and this leads to a construction of a class of graphs in P which is not FPC definable. This fundamental construction has been at the heart of many lower bounds since. That is, most results showing that some property P is not definable in FPC and therefore not decidable by symmetric circuits proceed by showing that P is not invariant under k-dimensional WL for any fixed k. Graph properties for which this has been shown include Hamiltonicity and 3-colourability.

Symmetric algorithms

The fact that we can prove unconditional lower bounds for classes of symmetric circuits would not be so interesting if these classes formed a very weak model of computation. It turns out, however, that many natural algorithmic techniques are in fact symmetric. First of all, it is worth recalling some of the original motivation for the interest of finite model theory in computer science, which came from the study of database query languages (see [31]). Languages for querying relational databases, based on first-order logic and its extensions, naturally give rise to symmetric algorithms (in the precise sense of symmetric circuits considered above) when automatically compiled (see, for instance, the connection to circuits given in [23, 29]). In this sense, FPC provides a good formal model of database query languages that extend the relational calculus with recursion and counting mechanisms. When FPC was first introduced [27] it was proposed as a possible language in which all polynomial-time decidable queries could be expressed. Even after Cai et al. showed that this was not the case, it was often said that all *natural* polynomial-time decidable properties are expressible in FPC. One way to understand this is that problems for which the obvious algorithm is in polynomial time can usually be formulated in FPC. However, the power of FPC, and hence of symmetric computation, is surprising and a number of problems for which the polynomial-time algorithms are far from trivial have been shown nonetheless to admit symmetric algorithms. A few are worth highlighting.

The most significant one is Grohe's monumental work [26] showing that any polynomial-time decidable property of graphs excluding some fixed minor is in FPC, and so invariant under k-WL equivalence for some fixed k. In [3], my co-authors and I show that the ellipsoid method for optimizing linear programs can be expressed in FPC, and so many natural combinatorial optimization problems have bounded WL dimension. In particular this is true of the problem of determining the size of a maximum matching in a graph. The result can be further extended to hierarchies of semi-definite programs [19, 6]. This shows that some of our most powerful techniques for constructing efficient algorithms can be implemented in a way that is symmetry preserving.

Linear-algebraic extensions

While many powerful algorithmic techniques can be implemented symmetrically, there are some simple efficient algorithms that just cannot be symmetrized without an exponential blow-up. It has been observed that the construction of Cai et al. essentially shows that

the problem of solving systems of linear equations over a a finite field cannot be expressed in FPC [4]. It follows that the Gaussian elimination algorithm cannot be implemented symmetrically. Indeed, since linear algebra, and more generally equation-solving, provides a rich source of examples of problems that cannot be expressed in FPC [12], research in descriptive complexity has investigated extensions of this logic with linear-algebraic operators. The resultung logics are provably more expressive than FPC. Here I want to point to connections of these with symmetric circuits, and with approximations of isomorphism stronger than the WL equivalences.

The first proposed extension of FPC by means of linear-algebraic operators was *fixed-point logic with rank* (FPR), which allows for operators that compute the rank of a matrix over a finite field [14, 25]. The expressive power of this logic has been shown to be characterized by symmetric circuits with *rank gates* [21]. To make this work, we need to modify the definition of circuit. To be precise, the Boolean function computed by a rank gate is not a fully symmetric function and so we can no longer think of a circuit as a DAG. It needs to have additional structure to give a matrix structure to the inputs of a rank gate. This relaxed notion of circuit gives a weaker requirement of symmetry which can be formalized and used to give a circuit characterization of FPR.

In order to study the expressiveness of FPR, a strengthening of the family of WL equivalences was defined in [16] which we call the *invertible map* (IM) equivalences. The WL equivalences can be seen as giving, on a fixed graph G, a partition of the k-tuples of vertices that approximates the partition into orbits of the automorphism group. The k-WL partition is the coarsest partition of the k-tuples of G into classes $P_1, \ldots, P_t$ satisfying a natural *stability condition*. This condition says that two tuples $\mathbf{u}$ and $\mathbf{v}$ in the same class P_i cannot be distinguished by counting the number of substitutions we can make in them to get a tuple in class P_j. The k-IM equivalences (denoted $\equiv^k_{\mathrm{IM}}$) are similarly defined but with a different stability condition. This essentially amounts to saying that the partition $P_1, \ldots, P_t$ cannot be further refined using *linear algebraic operators* over fields of characteristic p where p is a prime from some fixed set Ω. Technical details of the definition and characterization in terms of linear algebraic operators can be found in [13].

In a breakthrough result, Lichter [28] has shown that there is no constant k for which $\equiv^k_{\mathrm{IM}}$ is the same as isomorphism. The implications of this construction for the expressive power of any extension of FPC with linear-algebraic operators are spelled out in [15].

Symmetry and asymmetry in other models

Once we recognize symmetry as a feature of algorithms, it makes sense to identify how it appears in other models of computation, beyond logic and circuits. It is, of course, built in as a natural feature in the logics we study in descriptive complexity. And we have identified the corresponding notion in the context of circuits. One aim of identifying meaningful restrictions by symmetry in various models of computation is to see how asymmetry (or symmetry-breaking) is a resource and how it trades off with other computational resources. In this section, I aim to provide pointers to two specific models where we have obtained interesting insights by analysing symmetric restrictions to natural known computational paradigms.

The first is that of linear programming. I noted above that the ellipsoid method for solving linear programs can be implemented symmetrically. Moreover, when combinatorial problems are formualted as linear programs, the programs have natural symmetries inherited from the problem. Symmetric linear programs in this sense were studied by Yannakakis [32]. The focus there was on linear programs, or *extended formulations*. For example, consider

graphs over the vertex set $[n]$. We can consider these as functions $G : X \to \{0,1\}$ where $X = \{x_{ij} \mid i,j \in [n]\}$ is the set of potential edges. Equivalently, we can think of G as a 0-1 valued vector in the Euclidean space $\mathbb{R}^X$. A collection of graphs is then a set $P \subseteq \{0,1\}^X$ and various graph optimization problems can be expressed as optimizing a linear function over P. If we can represent the convex hull $\text{conv}(P)$ as the projection of a polytope $Q \subseteq \mathbb{R}^{X \times Y}$ using additional variables Y, with a number of facets polynomial in n, we can solve these problems in polynomial time. Yannakakis proved that the travelling salesman and matching polytopes do not have such polynomial-size *symmetric* extended formulations. The notion of symmetry is the natural one. Any permutation of $[n]$ has a natural action on X and hence on $\mathbb{R}^X$. The symmetry requirement says that for any such permutation $\pi \in S_n$ we can find a permutation σ of Y such that for $\mathbf{xy} \in R^{X \times Y}$, $\mathbf{xy} \in Q$ if, and only if, $\pi(\mathbf{x})\sigma(\mathbf{y}) \in Q$. While the lower bound proof of Yannakakis relies heavily on the notion of symmetry, it turns out that this is not essential to the result as Rothvoß [30] obtains exponential lower bounds without the assumption.

We can consider another way of representing the set $P \subseteq \{0,1\}^X$ as a linear program. We say that a polytope $\mathcal{P} \subseteq \mathbb{R}^X$ *recognizes* P if $P \subseteq \mathcal{P}$ and $\{0,1\}^X \setminus P$ is disjoint from $\mathcal{P}$. Now, a class of graphs that is decidable in polynomial time necessarily is recognized by a polynomial-size family of extended formulations. But, what classes are recognized by *symmetric* such families? It turns out that they are exactly the classes of bounded WL dimension. In other words those that are recognized by (possibly non-uniform) families of polynomial-size symmetric circuits with threshold gates [5].

Another computational model where the assumption of symmetry has revealed remarkable structure is that of *arithmetic circuits*. Formally, an arithmetic circuit over a field K and a set of variables X is a directed acyclic graph where every input (i.e. node of indegree 0) is labelled by an element of X or an element of K, and every internal node is labelled either $+$ (a sum gate) or $\times$ (a product gate). A distinguished *output* gate can then be seen as computing a polynomial in the ring $K[X]$. Two polynomials (strictly speaking they are families of polynomials) that are much studied in the field are the *determinant* and the *permanent*. They are both defined on a set of variables X representing the entries of an $n \times n$ matrix, so $X = \{x_{ij} \mid 1 \leq i,j \leq n\}$. It is known that there are polynomial-size circuits for computing the determinant $\det(X)$ while it is conjectured that there are no polynomial-size circuits for computing the permanent $\text{perm}(X)$. Both $\det(X)$ and $\text{perm}(X)$ are invariant under permutations of the variables X which are induced by the natural action of S_n. So, it makes sense to ask whether these polynomials can be computed by polynomial-size *symmetric* circuits. It turns out [20] that this is the case for the determinant but provably not for the permanent. Furthermore, these polynomials have symmetries that go beyond the action of S_n simultaneously on rows and columns. For example, the permanent is invariant under *any* permutation of the rows and columns of the matrix and the determinant under separate permutations of the rows and columns that have the same sign. We are able to establish lower bounds for arithmetic circuits assuming these more stringent symmetry requirements [22]. This provides an interesting case study in the tradeoff between symmetry and other resources, in this case circuit size. The lower bounds are obtained by adapting proof methods from finite model theory, even though the connection to logic is now remote.

Constraint satisfaction problems

One of the great breakthroughs in theoretical computer science in recent years was the dichotomy theorem for constraint satisfaction problems (CSP) proved independently by Bulatov [8] and Zhuk [33]. A specific CSP is given by a finite relational structure $\mathbb{D}$: the

domain D along with a collection of relations $R_1, \ldots, R_m$ on it. An instance to be solved is specified by a similar relational structure $\mathbb{V}$: the set V of variables and for each relation R_i, a set of tuples from V whose interpretation must be in the relation. A solution is just a homomorphism from $\mathbb{V}$ to $\mathbb{D}$. It turns out that the computational complexity of determining whether a given instance is solvable is completely determined by the algebraic structure of the so-called *clone of polymorphisms* of $\mathbb{D}$ [9]. Looking at the polymorphism clones of a structure, rather than the automorphism groups is a different notion of symmetry which is more relevant when we are interested in the homomorphisms between structures. Nonetheless, there is a potentially intriguing relationship between the two notions of symmetry.

While the Bulatov-Zhuk dichotomy theorem classifies all CSP into two classes: those that are solvable in polynomial time and those that are NP-complete, for our purposes a trifurcation of CSP is interesting. That is, we can further subdivide the polynomial-time solvable problems into those that have *bounded width* and those that do not. The CSP of bounded width can be solved by means of a simple algorithm, known as *local consistency* [7]. This is parameterized by a natural number k. Since, for fixed k, the k-local consistency check is a polynomial-time algorithm, all bounded-width CSP admit a polynomial-time algorithm. However, the converse is not true. A number of CSP are known which are efficiently solvable, but do not have bounded width. It turns out that a CSP has bounded width if, and only if, the collection of satisfiable instances has bounded WL dimension (see [4, 7, 18]). Thus, there is apparently a close relationship between the k-dimensional WL approximation of isomorphism and the k-local consistency method for approximating homomorphism. This relationship is made precise in the category theoretic framework we developed in [1].

It remains an open question whether we can similarly characterize *all* tractable CSP, i.e. those with a near-unanimity polymorphism by some (perhaps tractable) approximation of isomorphism. Indeed, it is conceivable that the invertible map equivalences serve this purpose. Could it be that for every such CSP there is a k such that the collection of satisfiable instances is invariant under $\equiv^k_{\mathrm{IM}}$? Converesely, could it be that for every CSP that does not admit such a polymorphism (i.e. those that we know to be NP-complete) is not invariant under $\equiv^k_{\mathrm{IM}}$ for any k?

Conclusion

We have discovered that symmetry arises in many forms in the analysis of computation, and is an important property of structured data, such as graphs and also of algorithms that work on this data. Symmetry in algorithms arises naturally when algorithms are automatically generated from high-level specifications. At the same time, symmetry-breaking can be an important method to improve efficiency of algorithmic procedures, and this is demonstrated by the unconditional lower bounds we have for symmetric algorithms for some problems, where efficient symmetry-breaking algorithms are known. The emerging theory of upper and lower bounds for symmetric computation pulls together a number of distinct strands within theoretical computer science, and draws in diverse mathematical methods with many promising directions to follow.

References

1 S. Abramsky, A. Dawar, and P. Wang. The pebbling comonad in finite model theory. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, 2017. doi:10.1109/LICS.2017.8005129.

2 M. Anderson and A. Dawar. On symmetric circuits and fixed-point logics. *Theory Comput. Syst.*, 60(3):521–551, 2017. doi:10.1007/s00224-016-9692-2.

3 M. Anderson, A. Dawar, and B. Holm. Solving linear programs without breaking abstractions. *J. ACM*, 62, 2015.

4 A. Atserias, A. Bulatov, and A. Dawar. Affine systems of equations and counting infinitary logic. *Theoretical Computer Science*, 410(18):1666–1683, 2009.

5 A. Atserias, A. Dawar, and J. Ochremiak. On the power of symmetric linear programs. *J. ACM*, 68:26:1–26:35, 2021. doi:10.1145/3456297.

6 A. Atserias and J. Fijalkow. Definable ellipsoid method, sums-of-squares proofs, and the graph isomorphism problem. *SIAM J. Comput.*, 52:1193–1229, 2023. doi:10.1137/20M1338435.

7 L. Barto and M. Kozik. Constraint satisfaction problems solvable by local consistency methods. *J. ACM*, 61:3:1–3:19, 2014.

8 A. A. Bulatov. A dichotomy theorem for nonuniform csps. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 319–330, 2017. doi:10.1109/FOCS.2017.37.

9 A. A. Bulatov, P. Jeavons, and A. A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34(3):720–742, 2005.

10 J-Y. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

11 A. Dawar. The nature and power of fixed-point logic with counting. *ACM SIGLOG News*, 2(1):8–21, 2015.

12 A. Dawar, E. Grädel, B. Holm, E. Kopczynski, and W. Pakusa. Definability of linear equation systems over groups and rings. *Logical Methods in Computer Science*, 9, 2013. doi:10.2168/LMCS-9(4:12)2013.

13 A. Dawar, E. Grädel, and W. Pakusa. Approximations of isomorphism and logics with linear-algebraic operators. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 112:1–112:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.112.

14 A. Dawar, M. Grohe, B. Holm, and B. Laubner. Logics with rank operators. In *2009 24th Annual IEEE Symposium on Logic In Computer Science*, pages 113–122. IEEE, 2009.

15 A. Dawar, E. Grädel, and M. Lichter. Limitations of the invertible-map equivalences. *Journal of Logic and Computation*, 33:961–969, 2023. doi:10.1093/logcom/exac058.

16 A. Dawar and B. Holm. Pebble games with algebraic rules. *Fundam. Informaticae*, 150:281–316, 2017. doi:10.3233/FI-2017-1471.

17 A. Dawar and K. Khan. Constructing hard examples for graph isomorphism. *J. Graph Algorithms Appl.*, 23:293–316, 2019. doi:10.7155/JGAA.00492.

18 A. Dawar and P. Wang. A definability dichotomy for finite valued CSPs. In *24th EACSL Annual Conference on Computer Science Logic, CSL 2015*, pages 60–77, 2015.

19 A. Dawar and P. Wang. Definability of semidefinite programming and Lasserre lower bounds for CSPs. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, 2017. doi:10.1109/LICS.2017.8005108.

20 A. Dawar and G. Wilsenach. Symmetric Arithmetic Circuits. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:18, 2020. doi:10.4230/LIPIcs.ICALP.2020.36.

21 A. Dawar and G. Wilsenach. Symmetric circuits for rank logic. *ACM Transactions on Computational Logic (TOCL)*, 23:1–35, 2021.

22 A. Dawar and G. Wilsenach. Lower bounds for symmetric circuits for the determinant. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 52:1–52:22, 2022. doi:10.4230/LIPIcs.ITCS.2022.52.

23 L. Denenberg, Y. Gurevich, and S. Shelah. Definability by constant-depth polynomial-size circuits. *Information and Control*, 70:216–240, 1986.

24 S. A. Fenner, L.ance Fortnow, and W. I. Gasarch. Complexity theory newsflash. *SIGACT News*, 27:126, 1996. doi:10.1145/235666.571629.

25 E. Grädel and W. Pakusa. Rank logic is dead, long live rank logic! *The Journal of Symbolic Logic*, 84, March 2019.

26 M. Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic. Cambridge University Press, 2017.

27 N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

28 M. Lichter. Separating rank logic from polynomial time. *J. ACM*, pages 14:1–14:53, 2023.

29 M. Otto. The logic of explicitly presentation-invariant circuits. In *Computer Science Logic, 10th International Workshop, CSL '96, Annual Conference of the EACSL*, pages 369–384, 1996.

30 T. Rothvoß. The matching polytope has exponential extension complexity. In *Symp. Theory of Computing, STOC 2014*, pages 263–272, 2014.

31 V. Vianu. Databases and finite-model theory. In N. Immerman and Ph. G. Kolaitis, editors, *Descriptive Complexity and Finite Models*, volume 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 97–148. AMS, 1996. doi:10.1090/DIMACS/031/04.

32 M. Yannakakis. Expressing combinatorial optimization problems by linear programs. *J. Comput. Syst. Sci.*, 43(3):441–466, 1991.

33 D. Zhuk. A proof of the CSP dichotomy conjecture. *Journal of the ACM*, 67:1–78, 2020.

Group Fairness: Multiwinner Voting and Beyond

Edith Elkind ✉
University of Oxford, UK
Alan Turing Institute, London, UK

Abstract

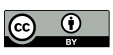
In multiwinner voting with approval ballots the agents are presented with a set of alternatives, each agent indicates which of these alternatives they approve, and the goal is to select a fixed-size subset of the alternatives, in a way that reflects the voters' preferences. This framework captures a variety of group decision-making scenarios, from choosing a list of speakers for an event to appointing a set of validators in a proof-of-stake blockchain. An important concern in many of these scenarios is group fairness: every sufficiently large group of agents with similar preferences should be represented in the winning set of alternatives. In this talk, we discuss how to formalise this idea and whether the resulting axioms can be satisfied by efficiently computable voting rules. We also discuss extensions of our framework to the more expressive setting of participatory budgeting, where the agents are presented with a slate of projects (which may have different costs) and the goal is to select a subset of projects subject to a budget constraint.

2012 ACM Subject Classification Applied computing

Keywords and phrases multiwinner voting, participatory budgeting, justified representation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.2

Category Invited Talk



51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 2; pp. 2:1–2:1

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Cross-Paradigm Graph Algorithms

Danupon Nanongkai
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
KTH Royal Institute of Technology, Stockholm, Sweden

Abstract

A goal of the theory of graph algorithms is algorithmic techniques that enable computing devices to process graph data with little resources (time, space, communication overhead, etc.). This led to extensive studies of graph algorithms in various models of computation (sequential algorithms, distributed algorithms, streaming algorithms, etc.) by many sub-communities. *Cross-paradigm graph algorithms* is an effort to attack the same problem in many models of computation *simultaneously*, with the goal to generate new insights that may not emerge from the isolated viewpoint of a single model and to ultimately develop techniques that can be used to solve graph problems near-optimally across many models of computation. In this talk, I will discuss some recent advances in graph algorithmic techniques for basic graph problems (e.g. minimum cut, shortest path, and maximum flow) in connection to this research program, especially some insights that led to cross-paradigm algorithms and to answering notorious open questions. No background will be assumed from the audience beyond familiarity with textbook graph algorithms.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Models of computation

Keywords and phrases Graph Algorithms and Complexity, Efficient Algorithms, Models of Computation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.3

Category Invited Talk

Funding *Danupon Nanongkai*: has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 715672 and was partially supported by the Swedish Research Council (Reg. No. 2019-05622).

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 3; pp. 3:1–3:1

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Graphs Shortcuts: New Bounds and Algorithms

Merav Parter ✉
Weizmann Institute of Science, Rehovot, Israel

Abstract

For an n-vertex digraph $G = (V, E)$, a *shortcut set* is a (small) subset of edges H taken from the transitive closure of G that, when added to G guarantees that the diameter of $G \cup H$ is small. Shortcut sets, introduced by Thorup in 1993, have a wide range of applications in algorithm design, especially in the context of parallel, distributed and dynamic computation on directed graphs. A folklore result in this context shows that every n-vertex digraph admits a shortcut set of linear size (i.e., of $O(n)$ edges) that reduces the diameter to $\widetilde{O}(\sqrt{n})$. Despite extensive research over the years, the question of whether one can reduce the diameter to $o(\sqrt{n})$ with $\widetilde{O}(n)$ shortcut edges has been left open.

In this talk, I will present the first improved diameter-sparsity tradeoff for this problem, breaking the $\sqrt{n}$ diameter barrier. Specifically, we show an $O(n^\omega)$-time randomized algorithm for computing a linear shortcut set that reduces the diameter of the digraph to $\widetilde{O}(n^{1/3})$. I also present time efficient algorithms for computing these shortcuts and explain the limitations of the current approaches. Finally, I will draw some connections between shortcuts and several forms of graph sparsification (e.g., reachability preservers, spanners). Based on a joint work with Shimon Kogan (SODA 2022, ICALP 2022, FOCS 2022, SODA 2023).

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Shortcuts, Spanners, Distance Preservers

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.4

Category Invited Talk

Funding This project is funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 949083), and by the Israeli Science Foundation (ISF), grant No. 2084/18.

An $O(\log \log n)$-Approximation for Submodular Facility Location

Fateme Abbasi
University of Wrocław, Poland

Marek Adamczyk
University of Wrocław, Poland

Miguel Bosch-Calvo
IDSIA, USI-SUPSI, Lugano, Switzerland

Jarosław Byrka
University of Wrocław, Poland

Fabrizio Grandoni
IDSIA, USI-SUPSI, Lugano, Switzerland

Krzysztof Sornat
AGH University, Kraków, Poland

Antoine Tinguely
IDSIA, USI-SUPSI, Lugano, Switzerland

Abstract

In the Submodular Facility Location problem (SFL) we are given a collection of n clients and m facilities in a metric space. A feasible solution consists of an assignment of each client to some facility. For each client, one has to pay the distance to the associated facility. Furthermore, for each facility f to which we assign the subset of clients S^f, one has to pay the opening cost $g(S^f)$, where $g(\cdot)$ is a monotone submodular function with $g(\emptyset) = 0$.

SFL is APX-hard since it includes the classical (metric uncapacitated) Facility Location problem (with uniform facility costs) as a special case. Svitkina and Tardos [35, SODA'06] gave the current-best $O(\log n)$ approximation algorithm for SFL. The same authors pose the open problem whether SFL admits a constant approximation and provide such an approximation for a very restricted special case of the problem.

We make some progress towards the solution of the above open problem by presenting an $O(\log \log n)$ approximation. Our approach is rather flexible and can be easily extended to generalizations and variants of SFL. In more detail, we achieve the same approximation factor for the natural generalizations of SFL where the opening cost of each facility f is of the form $p_f + g(S^f)$ or $w_f \cdot g(S^f)$, where $p_f, w_f \geq 0$ are input values.

We also obtain an improved approximation algorithm for the related Universal Stochastic Facility Location problem. In this problem one is given a classical (metric) facility location instance and has to a priori assign each client to some facility. Then a subset of active clients is sampled from some given distribution, and one has to pay (a posteriori) only the connection and opening costs induced by the active clients. The expected opening cost of each facility f can be modelled with a submodular function of the set of clients assigned to f.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Facility location and clustering; Theory of computation → Rounding techniques; Theory of computation → Online algorithms

Keywords and phrases approximation algorithms, facility location, submodular facility location, universal stochastic facility location

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.5

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2211.05474` [1]

Funding Fateme Abbasi and Jarosław Byrka were supported by Polish National Science Centre (NCN) Grant 2020/39/B/ST6/01641. Marek Adamczyk was supported by Polish National Science Centre (NCN) Grant 2019/35/D/ST6/03060. Miguel Bosch-Calvo, Fabrizio Grandoni, Krzysztof Sornat and Antoine Tinguely were supported by the SNSF Grant 200021_200731/1. Krzysztof Sornat was partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 101002854).

Acknowledgements We would like to thank the anonymous reviewers for their helpful comments, in particular for pointing out to the simpler and stronger lower bound construction by Gupta [23]. We would also like to thank Neil Olver for inspiring discussions about applications of their technique in [5] to various covering problems over time.

1 Introduction

In the Submodular Facility Location problem (SFL), we are given a set C of n clients and set F of m facilities, with metric distances $d : (C \cup F) \times (C \cup F) \to \mathbb{R}_{\geq 0}$. Furthermore, we are given a monotone submodular[1] (opening cost) function $g : 2^C \to \mathbb{R}_{\geq 0}$ with $g(\emptyset) = 0$. Notice that $g(\cdot)$ is non-negative. A feasible solution consists of an assignment $\varphi : C \to F$ of each client to some facility (we also say that $\varphi(c)$ *serves* c). The opening cost of $f \in F$ in this solution is $g(\varphi^{-1}(f))$. The cost of the solution, that we wish to minimize, is the sum of the distances from each client to the corresponding facility plus the total opening cost of the facilities, in other words

$$\text{cost}(\varphi) = \sum_{c \in C} d(c, \varphi(c)) + \sum_{f \in F} g(\varphi^{-1}(f)).$$

SFL captures practical scenarios where the cost of opening a facility is a (non-linear, still "tractable") function of the set of served clients. For example, each client might have different types of needs, and satisfying such needs might have a submodular impact on the opening cost (regardless of the facility location). As we will discuss, SFL is also closely related to certain stochastic optimization problems which recently attracted a lot of attention (see, e.g, [2, 17, 19, 22, 24] and references therein). In particular, there are scenarios where one has to pay (a posteriori) the connection and opening costs related only to a random subset of *activated* clients, and this naturally induces objective functions with submodular opening costs.

SFL is APX-hard since it includes the classical Facility Location problem (with uniform facility costs) as a special case [21]. Hence the best we can hope for, in terms of approximation algorithms, is a constant approximation. Finding such an approximation algorithm is explicitly posed as an open problem, e.g., by Svitkina and Tardos [35, 36]. The same authors present an $O(\log n)$ approximation, based on a greedy approach, for a generalization of SFL where each facility f has a distinct monotone submodular function $g_f(\cdot)$ (and this result is tight for this generalization due to a reduction from Set Cover by Shmoys, Swamy and Levi [33]). Chekuri and Ene [10] obtain an alternative $O(\log n)$ approximation for the same generalization of SFL based on rounding a convex relaxation exploiting Lovász extensions (see also the related work on submodular partitioning problems [9, 13]). Svitkina and Tardos also present a constant approximation for a rather restrictive (still practically motivated) special case of SFL where $g(\cdot)$ is induced by certain subtrees of a node-weighted tree over the clients.

1.1 Our Results and Techniques

We make some progress towards the resolution of the mentioned open problem by presenting an improved approximation algorithm for SFL.

[1] We recall that $g(\cdot)$ is submodular iff, for every $S, T \subseteq C$, $g(S) + g(T) \geq g(S \cap T) + g(S \cup T)$. The function is also monotone if $g(T) \leq g(S)$ for every $T \subseteq S \subseteq C$. As usual in this framework, we assume to have an oracle access to $g(\cdot)$: given $S \subseteq C$, we can obtain the value of $g(S)$ in polynomial time.

▶ **Theorem 1.1.** *There is a polynomial-time $O(\log\log n)$-approximation algorithm for SFL.*

Our approach is surprisingly simple (modulo exploiting some non-trivial results in the literature). By standard reductions (see Section 1.4) we can assume that $N = n + m$ is polynomial in n, hence it is sufficient to provide an $O(\log\log N)$ approximation. Our starting point is a natural (configuration) LP relaxation for the problem:

$$\begin{aligned}
\min \quad & \sum_{f\in F}\sum_{R\subseteq C} g(R)\cdot x_R^f + \sum_{c\in C}\sum_{f\in F}\sum_{R\ni c} d(c,f)\cdot x_R^f && \text{(Conf-LP)}\\
\text{s.t.} \quad & \sum_{f\in F}\sum_{R\ni c} x_R^f = 1 && \forall c\in C;\\
& \sum_{R\subseteq C} x_R^f = 1 && \forall f\in F;\\
& x_R^f \geq 0 && \forall R\subseteq C,\ \forall f\in F.
\end{aligned}$$

In an integral solution, we interpret $x_R^f = 1$ as assigning exactly the set of clients R to the facility f. Notice that we impose $\sum_{R\subseteq C} x_R^f = 1$. This is w.l.o.g. since $g(\emptyset) = 0$ (intuitively, $x_\emptyset^f = 1$ means that no client is assigned to f). We can solve the above LP in polynomial time (see Appendix A for a proof).

▶ **Lemma 1.2.** *In* $\mathrm{poly}(N)$ *time one can find an optimal solution to* (Conf-LP) *with* $\mathrm{poly}(N)$ *non-zero entries.*

Given an optimal solution $\dot{x} = (\dot{x}_R^f)_{f\in F, R\subseteq C}$ to (Conf-LP) of cost $\mathrm{cost}(\dot{x})$ as in Lemma 1.2, we proceed with two main stages. In the first stage (discussed in Section 2) we simply sample partial assignments of clients to facilities with the distribution induced by $\dot{x}$ for $\ln\ln N$ many times. This cost at most $\ln\ln N$ times the optimal LP cost in expectation, and leads to a partial solution that covers a random subset $C_1 \subseteq C$ of clients.

In the second stage (discussed in Section 3) we take care of the remaining uncovered clients $C_2 = C \setminus C_1$. Let us consider the restriction $\ddot{x}$ of $\dot{x}$ to C_2. The opening cost of $\ddot{x}$ might be as large as the opening cost of $\dot{x}$. However, in expectation, the connection cost of $\ddot{x}$ is only a $1/\ln N$ fraction of the connection cost of $\dot{x}$ (as we will show).

At this point, using the probabilistic tree embedding algorithm in [14], we embed the original metric d into a (rooted) tree metric d^T over a hierarchically well-separated tree (HST) T (see Section 1.4 for the details). The opening cost of $\ddot{x}$ w.r.t. to the new tree instance does not change, while its connection cost grows by a factor at most $O(\log N)$ in expectation. Altogether we obtain a feasible fractional solution $\ddot{x}$ over the tree instance whose expected cost is at most $O(\mathrm{cost}(\dot{x}))$. Hence it is sufficient to develop an $O(\log\log N)$-approximate LP-rounding algorithm for the considered tree instance.

The next step is at the heart of our approach. Using the properties of HSTs and losing a constant factor in the approximation, we can further reduce our SFL tree instance to the following Descendant-Leaf Assignment problem (DLA): the facilities are leaves of T and the clients are arbitrary nodes of T. Each client c must be served by a facility contained in the subtree T_c rooted at c. The opening cost of each facility is given by $g(\cdot)$, and there are no connection costs at all. Bosman and Olver [5] describe a reduction of Submodular Joint Replenishment and Inventory Routing problems to the Nice Subadditive Cover Over Time problem. We critically observed that DLA has some similarities with the latter problem (though this connection might not be obvious at first sight, see the discussion in Section 1.3). In particular, we were able to adapt their approach to achieve the desired $O(\log\log N)$ approximation for our DLA problem.

We remark that we do not know how to get an $O(1)$ approximation for SFL on trees (even on HSTs). Though such approximation would not imply an $O(1)$ approximation for SFL with our approach (due to the first stage), finding it seems to be a natural intermediate problem to address.

The first stage of our construction might be helpful in other related problems, in particular to reduce the input problem to one on HSTs while introducing an additive $O(\log \log n)$ term in the approximation ratio.

1.2 Generalizations and Variants

Our basic approach is rather flexible, and it can be applied to generalizations and variants of SFL. We next describe some other applications of our approach, and we expect to see a few more ones in the future. For example, we can handle the case where the opening cost of the facility f is $g_f(S^f) = w_f \cdot g(S^f)$, where $w_f \geq 0$ is some input value: we call this the SFL WITH MULTIPLICATIVE OPENING COSTS problem (MULTSFL).

▶ **Theorem 1.3.** *There is a polynomial-time $O(\log \log n)$-approximation algorithm for* MULTSFL.

Similarly, we can address the SFL WITH ADDITIVE OPENING COSTS problem (ADDSFL), where $g_f(S^f) = p_f + g(S^f)$ for $S^f \neq \emptyset$, $g_f(\emptyset) = 0$, and $p_f \geq 0$ is some input value.

▶ **Theorem 1.4.** *There is a polynomial-time $O(\log \log n)$-approximation algorithm for* ADDSFL.

The above generalizations are discussed in Appendix B. We remark that we do not know how to obtain an $O(\log \log n)$-approximation for the AFFINE SFL case, where the opening costs are submodular functions of the form $g_f(S^f) = p_f + w_f \cdot g(S^f)$. Notice that this generalizes both ADDSFL and MULTSFL. This is left as an interesting open problem.

As mentioned earlier, SFL is closely related to stochastic variants of FACILITY LOCATION. In particular, our approach also extends to the following UNIVERSAL STOCHASTIC FACILITY LOCATION problem (UNIVFL). Here we are given clients C and facilities F with metric distances d like in SFL, plus an opening cost w_f for each $f \in F$. Furthermore, we have an oracle access to a probability distribution $\pi : 2^C \to \mathbb{R}_{\geq 0}$ specifying the probability $\pi(A)$ that a given subset of clients $A \subseteq C$ is activated. A feasible solution is an (universal) mapping $\varphi : C \to F$. The cost of φ w.r.t. clients $A \subseteq C$ is $\text{cost}_A(\varphi) = \sum_{c \in A} d(c, \varphi(c)) + \sum_{f \in F : \varphi^{-1}(f) \cap A \neq \emptyset} w_f$. In words, this is the cost of connecting clients in A to the corresponding facilities, plus the cost of opening the facilities that serve at least one client in A. Our goal is to minimize $\mathbb{E}_{A \sim \pi}[\text{cost}_A(\varphi)]$. The main motivation for *universal* problems of this type is to allow a very quick (possibly distributed) reaction to requests that arrive over time. Let $\text{opt} : C \to F$ minimize $\mathbb{E}_{A \sim \pi}[\text{cost}_A(\text{opt})]$, in other words opt is an optimal (universal) mapping. We say that an algorithm for UNIVFL is α-approximate[2] if it returns a universal mapping φ satisfying $\mathbb{E}_{A \sim \pi}[\text{cost}_A(\varphi)] \leq \alpha \cdot \mathbb{E}_{A \sim \pi}[\text{cost}_A(\text{opt})]$.

Notice that the objective function of UNIVFL can be rewritten as

$$\sum_{c \in C} d(c, \varphi(c)) \cdot \mathbb{P}_{A \sim \pi}[\{c\} \cap A \neq \emptyset] + \sum_{f \in F} w_f \cdot \mathbb{P}_{A \sim \pi}[\varphi^{-1}(f) \cap A \neq \emptyset].$$

[2] In Section 1.3 we describe alternative ways to define the approximation ratio.

Hence univFL is almost identical to SFL since $g(R) = \mathbb{P}_{A\sim\pi}[R \cap A \neq \emptyset]$ is a monotone submodular function of R which is 0 for $R = \emptyset$. We can therefore adapt our techniques to achieve the following result (see Section 4). Let $\pi_{\min} := \min_{c\in C}\{\mathbb{P}_{A\sim\pi}[c \in A]\}$ be the smallest probability of any client to be activated. W.l.o.g. we will assume $\pi_{\min} > 0$.

▶ **Theorem 1.5.** *There is a polynomial-time $O(\log\log \frac{n}{\pi_{\min}})$-approximation algorithm for the* Universal Stochastic Facility Location *problem.*

For a comparison, Adamczyk, Grandoni, Leonardi and Włodarczyk [2] obtain an $O(\log n)$ approximation which also holds for non-metric distances. In the case of metric distances, they obtain an $O(1)$ approximation but only in the *independent activation case*, i.e., when the sampled set A of active clients is obtained by independently sampling each client c according to some input probability $\pi'(c)$ for k times.

1.3 Related Work

As mentioned earlier, Bosman and Olver [5] consider the Nice Subadditive Cover Over Time problem: roughly speaking, here we are given a set V of items and a time interval $\{1,\dots,L\}$. Each item $v \in V$ is associated with a time window $F_v = \{s,\dots,t\}$, $1 \le s \le t \le L$. The time windows altogether have a special *left-aligned* structure whose definition we skip here. A feasible solution consists of a subset $S_t \subseteq V$ for each $t \in \{1,\dots,L\}$, such that, for each $v \in V$, one has $v \in S_r$ for some $r \in F_v$. The goal is to minimize $\sum_{t=1}^{L} g(S_t)$, where $g(\cdot)$ is a monotone submodular set function with $g(\emptyset) = 0$. For this problem they give a $O(\log\log L)$ approximation, using a clever rounding algorithm for a convex optimization problem involving the Lovász extension of $g(\cdot)$. Intuitively, in our DLA problem (defined in Section 3.1) the time interval is replaced by the leaves (associated with some facility) of the tree $\tilde{T}$, and the time window of $c \in \tilde{C}$ by the set $\tilde{F}_c$. Our time windows naturally induce a laminar family, which is a special case of the left-aligned structure mentioned before. The parameter $\log L$ in their construction is replaced by the depth D of $\tilde{T}$ in our case.

In the (Metric Uncapacitated) Facility Location problem (FL) we are given a set of clients and a set of facilities in a metric space d, where each facility has an opening cost o_f. One has to select a subset of facilities $F' \subseteq F$ and assign each client c to the closest facility $F'(c)$ in F' so as to minimize $\sum_{c\in C} d(c, F'(c)) + \sum_{f\in F'} o_f$. FL is a special case of both addSFL and multSFL (and of SFL in the case of uniform opening costs). FL is among the best-studied problems in the literature from the point of view of approximation algorithms (see, e.g., [8, 30, 34]). It is known to be APX-hard [21] and the current best-known 1.488-approximation algorithm [28] is a randomized combination of the greedy JMS algorithm [26] with an LP-rounding algorithm from [6]. Lagrangian-multiplier preserving algorithms for FL are at the heart of several approximation algorithms for fundamental clustering problems, including k-Median [3, 7, 11, 12, 18, 26, 27, 29] and k-Means [3, 11, 20].

Various variants of FL were studied in the literature and for most of them (at least with metric connection costs) a constant approximation was eventually discovered. A notable example is the Capacitated Facility Location problem in which the number of clients that can be served from a facility is restricted by a location-specific bound. A local-search-based constant approximation for the latter problem is given in [37] (see also [4] for a more recent LP-based result). SFL is one of the most natural generalizations of (metric) FL where a constant approximation is still not known.

Grandoni, Gupta, Leonardi, Miettinen, Sankowski, and Singh [19], among other universal stochastic problems, studied univFL in the independent activation case. However, they compare the cost of their solution with $\mathbb{E}_{A\sim\pi}[\text{cost}_A(\text{opt}(A))]$, where $\text{opt}(A)$ is the optimal

facility location solution restricted to clients A (while we compare with $\mathbb{E}_{A\sim\pi}[\text{cost}_A(\text{opt})]$). For this setting they obtain a $O(\log n)$ approximation, which also holds for non-metric connection costs.

Gupta, Pál, Ravi, and Sinha [22] consider a 2-stage stochastic version of FL. Here in a first stage, one buys some facilities, then a subset of active clients is sampled from a given distribution. Finally, one can buy some more facilities, however at an opening cost which is increased by a multiplicative *inflation factor* σ. For this setting they present a constant approximation.

Universal stochastic problems have a natural online stochastic counterpart. For example, in the Online Stochastic Facility Location problem clients are sampled one by one, and when client c is sampled one has to connect c to an already open facility or open a new facility f and connect c to f. Garg, Gupta, Leonardi and Sankowski [17] consider this problem in the independent activation case, i.e. when the next client to be served is sampled from a probability distribution $\pi : C \to \mathbb{R}_{\geq 0}$. For this setting, they present an $O(1)$ approximation. Meyerson [31] studied a variant of the problem where an adversary chooses the set of input clients, and then a random permutation of them is presented in input (*random order model*).

We believe that it is plausible that SFL admits a constant approximation. In particular, one might consider greedy algorithms. Gupta [23] considered a natural set-cover type greedy algorithm for SFL. The same algorithm gives a 1.861-approximation when applied to the classical Facility Location problem [26]. Gupta [23, Section 2.3] showed that this algorithm produces an $\Omega(\log n)$ approximate solutions for SFL. Hence our algorithm is provably better than that one.

1.4 Preliminaries and Notation

We use log for the logarithm with base 2 and ln for the natural logarithm. Define $X = C \cup F$, and $N = |X| = |C \cup F|$. Given a metric d over X, we let $d_{\min}$ be the smallest non-zero distance and $d_{\max}$ be the largest distance (that we assume to be positive w.l.o.g). We use $g(c)$ as a shortcut for $g(\{c\})$.

We sometimes express a feasible solution to SFL in the form $S = (S^f)_{f\in F}$, where $S^f \subseteq C$ specifies the clients $\varphi^{-1}(f)$ assigned to f. Notice that for each $c \in C$ there is precisely one $f \in F$ with $c \in S^f$. We define a *partial assignment* as $S = (S^f)_{f\in F}$, where $S^f \subseteq C$. We say that S covers the clients $C' = \cup_{f\in F} S^f \subseteq C$. Notice that, for technical reasons, in a partial assignment we allow $S^f \cap S^{f'} \neq \emptyset$ for two distinct $f, f' \in F$ (i.e. we allow to simultaneously assign a client to more than one facility). The cost of a (partial) assignment S of the above type is defined as $\text{cost}(S) \coloneqq \text{conn}(S) + \text{open}(S)$, where $\text{conn}(S) \coloneqq \sum_{f\in F}\sum_{c\in S^f} d(c, f)$ is the connection cost of S and $\text{open}(S) \coloneqq \sum_{f\in F} g(S^f)$ is the opening cost of S. Given a (possibly infeasible) fractional solution x for (Conf-LP), we analogously define $\text{cost}(x) = \text{conn}(x) + \text{open}(x)$, where $\text{conn}(x) = \sum_{c\in C}\sum_{f\in F}\sum_{R\ni c} d(c, f) \cdot x^f_R$, and $\text{open}(x) = \sum_{f\in F}\sum_{R\subseteq C} g(R) \cdot x^f_R$.

It is convenient to define the merge $S = S_1 + S_2$ of two partial assignments S_1 and S_2 naturally as follows: (1) for each facility $f \in F$, we initially set $S^f \coloneqq S^f_1 \cup S^f_2$; (2) while there exist two distinct facilities f and f' with $S^f \cap S^{f'} \neq \emptyset$, replace $S^{f'}$ with $S^{f'} \setminus S^f$ (intuitively this second step guarantees that each client is assigned to no more than one facility). We observe that merging two partial assignments cannot increase the total cost.

▶ **Lemma 1.6.** *For any two partial assignments S_1 and S_2,* $\text{cost}(S_1 + S_2) \leq \text{cost}(S_1) + \text{cost}(S_2)$.

Proof. Let $S = S_1 + S_2$, and S' be the intermediate value of S obtained by executing only step (1) of the merge operation. One has $\mathrm{conn}(S') = \mathrm{conn}(S_1) + \mathrm{conn}(S_2)$. Furthermore, by the submodularity (hence subadditivity) of $g(\cdot)$, $\mathrm{open}(S') \leq \mathrm{open}(S_1) + \mathrm{open}(S_2)$. Clearly $\mathrm{conn}(S) \leq \mathrm{conn}(S')$, and the monotonicity of $g(\cdot)$ implies that $\mathrm{open}(S) \leq \mathrm{open}(S')$. The claim follows. ◀

We will exploit the following fairly standard reductions (proofs in Appendix A), thanks to which in the following it will be sufficient to obtain an $O(\log \log N)$ approximation for SFL. In order to distinguish between distinct instances J of the problem, we use $\mathrm{cost}_J(\varphi)$ to denote the cost of φ w.r.t. J and define similarly $\mathrm{open}_J(\varphi)$ etc.

▶ **Lemma 1.7.** *There is a 3-approximate reduction from SFL to the special case where $m = n$.*

▶ **Lemma 1.8.** *For any constant $\varepsilon > 0$, There is a $(1 + 4\varepsilon)$-approximate reduction from SFL to the special case where the metric d satisfies $d_{\min} = 2$ and $d_{\max} \leq \frac{2nN}{\varepsilon}$.*

One of the key tools that we use is the notion of probabilistic tree embedding, which we use to map the input metric into a metric on a *hierarchically well-separated tree* (HST) while stretching the distances by a small enough factor. We recall that an HST is an edge weighted rooted tree where all the leaves are at the same distance from the root r. Furthermore, on every path from a leaf to r the edge weights are $1, 2, 4, \ldots$ In particular, edges at the same level have the same weight. We will use the following construction[3] by Fakcharoenphol, Rao and Talwar [14].

▶ **Theorem 1.9** (FRT metric tree embedding [14])**.** *For any finite metric space (M, d) with $d_{\min} > 1$, there exists a randomized polynomial-time algorithm returning an HST T such that:*

1. *Every $a \in M$ is mapped to some leaf $map(a)$ of T (with elements at distance zero being mapped to the same leaf);*
2. *Let $d^T(a, b) := d^T(map(a), map(b))$ be the length of the path between the leaves $map(a)$ and $map(b)$ of T. Then $d^T(a, b) \geq d(a, b)$ and $\mathbb{E}\left[d^T(a, b)\right] \leq 8 \log |M| \cdot d(a, b)$;*
3. *T has depth $O(\log d_{\max})$.*

For a given set C, let $h\colon 2^C \to \mathbb{R}$ be a monotone submodular function with $h(\emptyset) = 0$. The *Lovász extension* $\hat{h}\colon [0, 1]^C \to \mathbb{R}$ of $h(\cdot)$ is defined as

$$\hat{h}(y) := \min \Big\{ \sum_{R \subseteq C} h(R)\mu_R : \sum_{R \subseteq C} \sum_{R \ni c} \mu_R = y_c \ \forall c \in C,\ \sum_{R \subseteq C} \mu_R = 1,\ \mu \geq 0 \Big\}. \tag{1}$$

The function $\hat{h}(\cdot)$ is convex. We remark that $\hat{h}(y)$ can be alternatively defined as

$$\hat{h}(y) := \sum_{k=1}^{n-1} h\left(\{c_1, \ldots, c_k\}\right)\left(y_{c_k} - y_{c_{k+1}}\right) + h(C) y_{c_n}, \tag{2}$$

where the components of y are sorted in decreasing order, i.e. $y_{c_1} \geq y_{c_2} \geq \cdots \geq y_{c_n}$ [16, Section 6.3]. By the monotonicity of $h(\cdot)$, $\hat{h}(\cdot)$ is also non-decreasing in the sense that $\hat{h}(y) \geq \hat{h}(y')$ if $y \geq y'$.

[3] We slightly and trivially extend their claim to consider nodes at distance 0.

2 Reducing the Connection Cost

In this section, we show how to compute a random partial assignment $S_1 = (S_1^f)_{f \in F}$ covering a random subset of clients $C_1 \coloneqq \cup_{f \in F} S_1^f \subseteq C$ with the following high-level properties: the expected cost of S_1 is "small enough" and (2) each client belongs to C_1 with "large enough" probability. In the next section, we will describe a different partial assignment $S_2 = (S_2^f)_{f \in F}$, again of small enough cost, covering the remaining clients $C_2 \coloneqq C \setminus C_1$. By merging these two partial assignments we obtain a feasible solution for the input problem of small enough total cost.

Let $\dot{x}$ be an optimal solution to (Conf-LP) with at most poly(N) non-zero entries that can be computed via Lemma 1.2. The basic idea behind the next lemma is fairly standard: we sample partial assignments according to the distribution induced by $\dot{x}$ for $\ln \ln N$ times, and merge them together.

▶ **Lemma 2.1.** *In polynomial time one can compute a random partial assignment S_1 covering a random subset of clients C_1 such that: (1) $\mathbb{E}[\text{cost}(S_1)] \leq \ln \ln(N) \cdot \text{cost}(\dot{x})$ and (2) For each $c \in C$, $\mathbb{P}[c \in C_1] \geq 1 - \frac{1}{\ln N}$.*

Proof. For $i \in \{1, 2, \ldots, \ln \ln N\}$ and for every $R \subseteq C$, we define a partial assignment $S(i, R)$ by setting $S^f(i, R) = R$ independently with probability $\dot{x}_R^f$ and $S^f(i, R) = \emptyset$ otherwise. Let $S_1 = \sum_{i=1}^{\ln \ln N} \sum_{R \subseteq C} S(i, R)$ be obtained by merging all these solutions, and let $C_1 = \cup_{f \in F} S_1^f$. Observe that

$$\mathbb{P}[c \notin C_1] = \prod_{f \in F} \prod_{R \ni c} (1 - \dot{x}_R^f)^{\ln \ln N} \leq e^{-\ln \ln N \sum_{f \in F} \sum_{R \ni c} \dot{x}_R^f} \leq e^{-\ln \ln N} = \frac{1}{\ln N}.$$

Furthermore, by Lemma 1.6, $\mathbb{E}[\text{cost}(S_1)]$ is upper-bounded by

$$\sum_{i=1}^{\ln \ln N} \sum_{R \subseteq C} \mathbb{E}[\text{cost}(S(i, R))] = \ln \ln N \cdot \sum_{f \in F, R \subseteq C} \dot{x}_R^f \cdot \left(g(R) + \sum_{c \in R} d(c, f) \right) = \ln \ln N \cdot \text{cost}(\dot{x}). \quad \blacktriangleleft$$

Consider the partial assignment S_1 covering the random subset of clients C_1 as in the previous lemma. Let $C_2 \coloneqq C \setminus C_2$ be the remaining (uncovered) clients. Let also $\ddot{x}$ be $\dot{x}$ restricted to C_2, i.e. $\ddot{x}_R^f = \sum_{R' \subseteq C_1} \dot{x}_{R \cup R'}^f$ for $R \subseteq C_2$ and $f \in F$. The following lemma upper bounds the expected opening and connection cost of $\ddot{x}$.

▶ **Lemma 2.2.** *One has* $\text{open}(\ddot{x}) \leq \text{open}(\dot{x})$ *and* $\mathbb{E}[\text{conn}(\ddot{x})] \leq \frac{1}{\ln N} \text{conn}(\dot{x})$.

Proof. We have $\text{open}(\ddot{x}) \leq \text{open}(\dot{x})$ by the monotonicity of $g(\cdot)$. For the connection cost, notice that the probability of a client c being in C_2 is at most $1/\ln N$, and only in that case one has to pay the associated connection cost. Thus by linearity of expectation, the expected connection cost of $\ddot{x}$ is at most $\text{conn}(\dot{x})/\ln N$. The claim follows. ◀

Notice that $\ddot{x}$ is a feasible fractional solution for (Conf-LP) limited to C_2. In the following section, we show how to randomly round $\ddot{x}$ to a partial assignment S_2 which covers C_2 at expected cost $O(\log \log N) \cdot \text{cost}(\ddot{x})$. It will then follow that $S_1 + S_2$ is a feasible $O(\log \log N)$-approximate solution to the input SFL instance.

3 Approximating SFL on an HST

Given an SFL instance and a tree embedding of $(C \cup F, d)$ into an HST T as in Theorem 1.9, we say that $(C \cup F, d^T, g(\cdot), map(\cdot))$ is the corresponding HST-type instance. We remark that we allow multiple clients $C(v)$ and facilities $F(v)$ to be colocated at each leaf v of T. In this section we will describe an $O(\log \log N)$-approximate LP-rounding algorithm for the considered instances w.r.t. (Conf-LP).

▶ **Lemma 3.1.** *Given a feasible fractional solution x to* (Conf-LP) *for an HST-type SFL instance, in polynomial time one can compute a feasible (integral) solution for the same instance with cost at most $O(\log\log N)\cdot \mathrm{cost}(x)$.*

Theorem 1.1 directly follows.

Proof of Theorem 1.1. By Lemma 1.7 it is sufficient to describe an $O(\log\log N)$-approximation. Furthermore by Lemma 1.8, we can assume that $d_{\min}=2$ and $d_{\max}\le \frac{2nN}{\varepsilon}$.

By applying the construction of Section 2 we compute a random partial assignment $S_1=(S_1^f)_{f\in F}$ covering the clients $C_1=\cup_{f\in F}S_1^f$ with expected cost at most $O(\log\log N)\cdot\mathrm{cost}(\dot{x})$, where $\dot{x}$ is an optimal solution to (Conf-LP). Furthermore, by Lemma 2.2, we obtain a feasible solution $\ddot{x}$ to (Conf-LP) restricted to clients $C_2\coloneqq C\setminus C_1$ which satisfies $\mathrm{open}(\ddot{x})\le\mathrm{open}(\dot{x})$ and $\mathbb{E}[\mathrm{conn}(\ddot{x})]\le\frac{1}{\ln N}\mathrm{conn}(\dot{x})$. By applying the probabilistic tree embedding from Theorem 1.9 to the metric $(C_2\cup F,d)$, we obtain an HST-type SFL instance $(C_2\cup F,d^T,g(\cdot),map(\cdot))$ where the tree has depth $D=O(\log d_{\max})=O(\log N)$. Observe that $\ddot{x}$ is a feasible fractional solution for (Conf-LP) restricted to C_2 on the HST-type instance. Furthermore, let $\mathrm{conn}_T(\ddot{x})$ denote the connection cost of $\ddot{x}$ w.r.t. the HST-type instance, and define similarly $\mathrm{open}_T(\ddot{x})$ and $\mathrm{cost}_T(\ddot{x})$. Then one has

$$\mathbb{E}[\mathrm{cost}_T(\ddot{x})]=\mathrm{open}(\ddot{x})+\mathbb{E}[\mathrm{conn}_T(\ddot{x})]\le\mathrm{open}(\dot{x})+O(\log N)\cdot\mathbb{E}[\mathrm{conn}(\ddot{x})]\le O(\mathrm{cost}(\dot{x})).$$

By applying the LP-rounding algorithm from Lemma 3.1 to $\ddot{x}$ one obtains a partial assignment $(S_2^f)_{f\in F}$ covering the clients C_2 of cost at most $O(\log\log N)\,\mathrm{cost}(\dot{x})$. The same solution has no larger cost in the original problem (on a non-tree metric). Altogether S_1+S_2 is a feasible solution to the input SFL problem of expected cost at most $O(\log\log N)\cdot\mathrm{cost}(\dot{x})\le O(\log\log N)\cdot\mathrm{cost}(\mathrm{opt})$. ◀

In the rest of this section, we prove Lemma 3.1. To this aim, we will first present a reduction to a different problem that we call the Descendent-Leaf Assignment problem (DLA) (see Section 3.1). Then, we will present a good-enough approximation algorithm for DLA (see Section 3.2).

3.1 A Reduction to DLA

In the Descendent-Leaf Assignment problem (DLA) we are given a rooted tree $\tilde{T}$ with depth D, a set of facilities $\tilde{F}$ and a set of clients $\tilde{C}$. Each $x\in\tilde{F}\cup\tilde{C}$ is mapped to some node $v(x)$ of $\tilde{T}$, with the restriction that facilities are mapped to leaves of $\tilde{T}$. By $\tilde{F}_c$ we denote the facilities which are mapped to nodes that are descendants of $v(c)$ in T ($v(c)$ included if it is a leaf). A feasible solution consists of an assignment $\tilde{\varphi}:\tilde{C}\to\tilde{F}$ of each $c\in\tilde{C}$ to some $f\in\tilde{F}_c$. The cost of this solution is $\sum_{f\in\tilde{F}}h(\tilde{\varphi}^{-1}(f))$, where $h(\cdot)$ is a monotone submodular function over $\tilde{C}$ with $h(\emptyset)=0$. Similarly to SFL, we also express a feasible solution as $S=(S^f)_{f\in\tilde{F}}$, where $S^f=\tilde{\varphi}^{-1}(f)$, and let $\mathrm{cost}_{\mathrm{DLA}}(S)=\sum_{f\in\tilde{F}}h(S^f)$ be the associated cost. We define a convex-programming (CP) relaxation for DLA as follows:

$$\begin{aligned}
\min\quad & \sum_{f\in\tilde{F}}\hat{h}(z^f) && && \text{(DLA-CP)}\\
\text{s.t.}\quad & \sum_{f\in\tilde{F}_c}z_c^f=1 && \forall c\in\tilde{C};\\
& z_c^f\ge 0 && \forall c\in\tilde{C},\ \forall f\in\tilde{F}.
\end{aligned}$$

In a 0-1 integral solution we interpret $z_c^f = 1$ as c being assigned to f. Recall that $\hat{h}(\cdot)$ is convex, which makes (DLA-CP) a convex program. We also notice that each feasible assignment $S = (S^f)_{f\in\tilde{F}}$ corresponds to a feasible integral solution $z = (z^f)_{f\in\tilde{F}}$ to (DLA-CP) with $\text{cost}_{\text{DLA}}(S) = \text{cost}_{\text{DLA}}(z) \coloneqq \sum_{f\in\tilde{F}} \hat{h}(z^f)$ and vice versa. Hence indeed (DLA-CP) is a CP-relaxation of DLA.

The next lemma provides the claimed reduction from SFL on HST-type instances to DLA.

▶ **Lemma 3.2.** *Given a polynomial-time $O(\log D)$-approximate CP-rounding algorithm for DLA w.r.t.* (DLA-CP)*, where D is the depth of the tree, there is polynomial-time $O(\log\log N)$-approximate LP-rounding algorithm for SFL on HST-type instances with tree-depth $O(\log N)$ w.r.t.* (Conf-LP)*.*

Proof. Let $(C \cup F, d^T, g(\cdot), map(\cdot))$ be the considered HST-type instance of SFL over an HST T, and x be an input feasible fractional solution to (Conf-LP) for this instance.

We build an instance $(\tilde{C}\cup\tilde{F}, \tilde{T}, h(\cdot), v(\cdot))$ of DLA as follows. First, let $y_c^f \coloneqq \sum_{R\subseteq C: c\in R} x_R^f$: intuitively this is the fractional amount by which c is assigned to f in x. We set $h(\cdot) = g(\cdot)$ and $\tilde{T} = T$. Notice that $D = O(\log N)$. We set $\tilde{F} = F$ and $v(f) = map(f)$ for each $f \in \tilde{F}$. We associate to each $c \in C$ a new client $\tilde{c} \in \tilde{C}$. Let T_v be the subtree rooted at v (containing v and all its descendants) and F_v be the facilities located in the leaves of T_v according to $map(\cdot)$. We map $\tilde{c}$ into the lowest ancestor $v(\tilde{c})$ of $map(c)$ such that $\sum_{f\in F_{v(\tilde{c})}} y_c^f \geq 1/2$. Notice that $v(\tilde{c}) = map(c)$ is possible (in which case there is at least one facility f colocated with c in T).

We next define a feasible fractional solution z for (DLA-CP) w.r.t this DLA instance as follows. For each $\tilde{c} \in \tilde{C}$ we set $z_{\tilde{c}}^f = y_c^f/(\sum_{f'\in F_{v(\tilde{c})}} y_c^{f'})$ if $f \in F_{v(\tilde{c})}$, and otherwise $z_{\tilde{c}}^f = 0$. Let $\tilde{\varphi}$ be a solution to the DLA instance obtained with the CP-rounding algorithm in the claim w.r.t. z. We obtain a feasible solution φ for the input instance by simply setting $\varphi(c) = \tilde{\varphi}(\tilde{c})$.

It remains to analyze the cost of φ. Define $\bar{z}_{\tilde{c}}^f = y_c^f/(\sum_{f'\in F_{v(\tilde{c})}} y_c^{f'})$ for all $f \in F$. Notice that $\bar{z} \geq z$. By the definition of $\hat{h}(\cdot)$ and its monotonicity, $\hat{h}(z^f) \leq \hat{h}(\bar{z}^f) = \hat{h}(y^f/(\sum_{f'\in F_{v(\tilde{c})}} y_c^{f'})) \leq 2\hat{h}(y^f) = 2\hat{g}(y^f)$. Notice that by plugging in x_R^f for μ_R in the set in (1) and by how y is defined w.r.t. x above, we get $\hat{g}(y^f) \leq \sum_{R\subseteq C} g(R)\cdot x_R^f$ and in particular $\sum_{f\in F}\hat{g}(y^f) \leq \text{open}(x)$. Thus, we have $\text{cost}_{\text{DLA}}(z) \leq 2\,\text{open}(x)$ and

$$\text{open}(\varphi) = \text{cost}_{\text{DLA}}(\tilde{\varphi}) = O(\log D)\cdot\text{cost}_{\text{DLA}}(z) \leq O(\log\log N)\cdot 2\,\text{open}(x). \tag{3}$$

Consider next the connection cost of a given $c \in C$. If $v(\tilde{c}) = map(c)$, i.e $v(\tilde{c})$ has no child, then $d^T(c,\varphi(c)) = 0 \leq \sum_{f\in F} d^T(c,f)y_c^f$. Otherwise, let $w(c)$ be the child of $v(\tilde{c})$ along the $v(\tilde{c})$-$map(c)$ path in T. Let Δ be the weight of the edge between $v(\tilde{c})$ and $w(c)$. Observe that the distance between $v(\tilde{c})$ and the leaves in $T_{v(\tilde{c})}$ is exactly $2\Delta - 1$. Furthermore, both c and $\varphi(c)$ are located in the leaves of $T_{v(\tilde{c})}$ in the HST mapping $map(\cdot)$. Hence $d^T(c,\varphi(c)) \leq 2(2\Delta-1)$.

By the definition of $v(\tilde{c})$, it must be the case that $\sum_{f\in F_{w(c)}} y_c^f < \frac{1}{2}$, and consequently $\sum_{f\in F\setminus F_{w(c)}} y_c^f \geq \frac{1}{2}$. For each $f \in F \setminus F_{w(c)}$, the $map(f)$-$map(c)$ path in T has length at least $2(2\Delta-1)$. Thus

$$\sum_{f\in F} d^T(c,f)y_c^f \geq \sum_{f\in F\setminus F_{w(c)}} d^T(c,f)y_c^f \geq \frac{1}{2}2(2\Delta-1).$$

Therefore, the connection cost of c in φ is at most 2 times its connection cost in x. We conclude that $\text{conn}(\varphi) \leq 2\,\text{conn}(x)$. Altogether we have $\text{cost}(\varphi) \leq 2\,\text{conn}(x) + O(\log\log N) \cdot 2\,\text{open}(x) \leq O(\log\log N) \cdot \text{cost}(x)$. ◀

3.2 An Approximation Algorithm for DLA

In this section, we present a CP-rounding algorithm for DLA. Lemma 3.1 follows by chaining Lemmas 3.2 and 3.3.

▶ **Lemma 3.3.** *Given a feasible fractional solution z to* (DLA-CP) *on an instance of DLA with tree-depth D, in polynomial time one can compute a feasible (integral) solution to the same instance of cost at most $O(\log D) \cdot \text{cost}_{\text{DLA}}(z)$.*

The CP-rounding algorithm from Lemma 3.3 is essentially the algorithm by Bosman and Olver [5] with minor modifications that we introduced to simplify our correctness analysis. Also, the analysis of its approximation ratio is essentially identical to [5], but we reproduce it for the sake of completeness. In particular, we will exploit the following definitions and lemma from [5]. Let $h : 2^{\tilde{C}} \to \mathbb{R}_{\geq 0}$ be a monotone submodular function with $h(\emptyset) = 0$. For a given $f \in \tilde{F}$ and a (possibly infeasible) solution z to (DLA-CP), let $L_\theta(z^f) \coloneqq \{c \in \tilde{C} : z_c^f \geq \theta\}$ be the set of clients that are served fractionally by at least some value θ by f. Let also $z^{f|\theta}$ be obtained from z^f by rounding down to θ the values larger than θ, i.e. $z_c^{f|\theta} \coloneqq \min\{z_c^f, \theta\}$ for each $c \in \tilde{C}$. Given $\theta \in [0,1]$ and $z^f \in [0,1]^{\tilde{C}}$, we say that the set $L_\theta(z^f)$ is α-supported (w.r.t. h) if $\hat{h}(z^f) - \hat{h}(z^{f|\theta}) \geq \alpha h(L_\theta(z^f))$.

▶ **Lemma 3.4** (Lemma 5.2 from [5])**.** *Given $z^f \in [0,1]^{\tilde{C}}$ and $\alpha \in (0,1]$, at least one of the following holds: (1) there exists $\theta \in [0,1]$, which can be computed in polynomial time, such that $L_\theta(z^f)$ is $\frac{\alpha}{32}$-supported; (2) $2^{1/\alpha} h(L_1(z^f)) \leq \hat{h}(z^f)$.*

Our algorithm is Algorithm 1 in the figure. Recall that $\tilde{T}_v$ is the subtree rooted at node v, where $\tilde{T}_v$ includes v and all its descendants. Furthermore, $\tilde{F}_v$ is the set of facilities mapped to the leaves of $\tilde{T}_v$. As usual the level of a node is its hop-distance from the root.

Algorithm 1 An algorithm used to prove Lemma 3.3.

Input: Feasible solution z to (DLA-CP)

$S^f \leftarrow \emptyset$ for all $f \in F$
for $i = 0, \ldots, D$ **do**
  For every node v at level $D - i$, choose an arbitrary $f_v \in \tilde{F}_v$ and set $z^{f_v} \leftarrow \sum_{f' \in \tilde{F}_v} z^{f'}$ and $z^{f'} \leftarrow 0$ for all $f' \in \tilde{F}_v \setminus \{f_v\}$
  if there exists $\theta \in [0,1]$ such that $L_\theta(z^{f_v})$ is $\frac{1}{32\log(D+1)}$-supported **then**
    For an arbitrary such θ, set $S^{f_v} \leftarrow S^{f_v} \cup L_\theta(z^{f_v})$ and $z_c^{f_v} \leftarrow 0$ for all $c \in L_\theta(z^{f_v})$
  else
    Set $S^{f_v} \leftarrow S^{f_v} \cup L_1(z^{f_v})$ and $z_c^{f_v} \leftarrow 0$ for all $c \in L_1(z^{f_v})$
For every $c \in \tilde{C}$ choose $f \in \tilde{F}_c$ such that $c \in S^f$ and set $S^{f'} \leftarrow S^{f'} \setminus \{c\}$ for all $f' \in \tilde{F} \setminus \{f\}$
return $(S^f)_{f \in \tilde{F}}$

Clearly Algorithm 1 runs in polynomial time. The next two lemmas analyze the correctness and the approximation ratio of Algorithm 1, hence proving Lemma 3.3.

▶ **Lemma 3.5.** *Algorithm 1 computes a feasible DLA solution.*

Proof. Consider a given client $c \in \tilde{C}$ such that $v(c)$ is at level $D - i$ in $\tilde{T}$. Let us show that the following invariant holds at the beginning of each iteration $j \leq i$: either $\sum_{f \in \tilde{F}_c} z_c^f = 1$ or $c \in S^f$ for some $f \in \tilde{F}_c$. The invariant trivially holds for $j = 0$. Assume that it holds

up to the beginning of iteration $j < i$, and consider what happens during that iteration. Notice that for every node v at level $D - j > D - i$, we either have that every $f \in \tilde{F}_v$ is a descendant of $v(c)$ or every $f \in \tilde{F}_v$ is not in $\tilde{F}_c$. Therefore, in Step (3) the value of $\sum_{f \in \tilde{F}_c} z_c^f$ does not change. In more detail, it remains 1 by inductive hypothesis. The same value can decrease in Steps (5) or (7), however, this can only happen if c is added to S^{f_v} for some $f_v \in \tilde{F}_c$. Thus the invariant holds at the end of the j-th iteration, hence at the beginning of the next iteration $j + 1$.

Due to the invariant, during the iteration i, when one considers the node $v = v(c)$, one has that either c already belongs to some S^f with $f \in \tilde{F}_c$, or $\sum_{f \in \tilde{F}_c} z_c^f = 1$. In the latter case, after Step (3), $z_c^{f_v} = 1$ where $f_v \in \tilde{F}_c$, so c belongs to every set $L_\theta(z^{f_v})$ with $\theta \in [0, 1]$. As a consequence, c is added to S^{f_v} either in Step (5) or in Step (7).

It might happen that a client c is assigned *also* to a facility not in $\tilde{F}_c$. Step (8) guarantees that the final assignment of c is correct and unique. ◀

▶ **Lemma 3.6.** *Algorithm 1 outputs a solution of cost at most* $O(\log D) \cdot \mathrm{cost}_{\mathrm{DLA}}(z)$.

Proof. Recall that $\mathrm{cost}_{\mathrm{DLA}}(z) = \sum_{f \in \tilde{F}} \hat{h}(z^f)$. We start by observing that the value of $\mathrm{cost}_{\mathrm{DLA}}(z)$ can not increase over time when z changes during the execution of the algorithm. Indeed, Steps (5) and (7) can only decrease the entries of z, hence $\mathrm{cost}_{\mathrm{DLA}}(z)$ by the monotonicity of $\hat{h}(\cdot)$. The only other changes of z happen in Step (3). Let us interpret this step as iteratively decreasing to zero $z^{f'}$ for each $f' \in \tilde{F}_v \setminus \{f_v\}$ and increasing z^{f_v} by the same amount. The decrease of the cost at each step is $\hat{h}(z^{f_v}) + \hat{h}(z^{f'}) - \hat{h}(z^{f_v} + z^{f'})$. By the alternative definition of $\hat{h}(\cdot)$ as in (2) and its convexity, one has $\hat{h}(z^{f_v} + z^{f'}) = 2\hat{h}\left(\frac{z^{f_v} + z^{f'}}{2}\right) \leq 2\left(\frac{1}{2}\hat{h}(z^{f_v}) + \frac{1}{2}\hat{h}(z^{f'})\right) = \hat{h}(z^{f_v}) + \hat{h}(z^{f'})$. Hence the decrease of the cost is non-negative as required.

For each facility f and level i, let $\Delta_i^\theta(f)$ be the clients added to S^f in Step (5) during iteration i (possibly $\Delta_i^\theta(f) = \emptyset$). We define similarly $\Delta_i^1(f)$ w.r.t. Step (7). Notice that, by the submodularity (hence subadditivity) of $h(\cdot)$, the increase of the cost of the solution due to adding Δ to S^f is at most $h(\Delta)$. Therefore we can upper bound the cost of the final solution $S = (S^f)_{f \in \tilde{F}}$ by

$$\mathrm{cost}_{\mathrm{DLA}}(S) \coloneqq \sum_{f \in \tilde{F}} h(S^f) \leq \sum_{i=0}^{D} \sum_{f \in \tilde{F}} \Big(h(\Delta_i^\theta(f)) + h(\Delta_i^1(f)) \Big).$$

Let us upper bound the right-hand side of the above inequality. Let $z(i)$ denote the value of z at the beginning of iteration i. From the previous observation, we have $\hat{h}(z(i)) \leq \hat{h}(z)$ for every i. By Lemma 3.4 with $\alpha = \frac{1}{\log(D+1)}$, for any $\Delta_i^1(f)$ one has $h(\Delta_i^1(f)) \leq \frac{1}{D+1}\hat{h}(z^f(i))$. Thus

$$\sum_{i=0}^{D} \sum_{f \in \tilde{F}} h\big(\Delta_i^1(f)\big) \leq \sum_{i=0}^{D} \sum_{f \in \tilde{F}} \frac{1}{D+1} \hat{h}\big(z^f(i)\big) \leq \sum_{i=0}^{D} \frac{1}{D+1} \mathrm{cost}_{\mathrm{DLA}}(z(i)) \leq \mathrm{cost}_{\mathrm{DLA}}(z). \quad (4)$$

Let $z(D+1)$ be the value of z at the end of the D-th iteration, hence in particular $\mathrm{cost}_{\mathrm{DLA}}(z(D+1)) \geq 0$. Notice that $z = z(0)$. We can lower bound $\mathrm{cost}_{\mathrm{DLA}}(z)$ by

$$\mathrm{cost}_{\mathrm{DLA}}(z) \geq \sum_{i=0}^{D} \Big(\mathrm{cost}_{\mathrm{DLA}}(z(i)) - \mathrm{cost}_{\mathrm{DLA}}(z(i+1)) \Big).$$

Let $z_1(i)$ be the value of z obtained from $z(i)$ after applying Step (3) for all nodes of level $D-i$. Let also $z_2(i)$ be the value obtained from $z_1(i)$ if, for all the facilities F'_i where Step (5) is applied during iteration i, instead of setting $z^f_c = 0$ one sets $z^f_c = \theta$ for the corresponding value of θ. For the facilities not in F'_i we simply let $z^f_2(i) = z^f_1(i)$. Observe that $z(i+1) \le z_2(i) \le z_1(i) \le z(i)$. One has

$$\begin{aligned}
\mathrm{cost}_{\mathrm{DLA}}(z(i)) - \mathrm{cost}_{\mathrm{DLA}}(z(i+1)) &\ge \mathrm{cost}_{\mathrm{DLA}}(z_1(i)) - \mathrm{cost}_{\mathrm{DLA}}(z(i+1)) \\
&\ge \mathrm{cost}_{\mathrm{DLA}}(z_1(i)) - \mathrm{cost}_{\mathrm{DLA}}(z_2(i)) \\
&= \sum_{f\in\tilde{F}} \hat{h}\Big(z^f_1(i)\Big) - \hat{h}\Big(z^f_2(i)\Big) = \sum_{f\in F'_i} \hat{h}\Big(z^f_1(i)\Big) - \hat{h}\Big(z^f_2(i)\Big) \\
&\ge \frac{\sum_{f\in F'_i} h\big(\Delta^\theta_i(f)\big)}{32\log(D+1)} = \frac{\sum_{f\in\tilde{F}} h\big(\Delta^\theta_i(f)\big)}{32\log(D+1)}.
\end{aligned}$$

In the first two inequalities above we used the monotonicity of $\hat{h}(\cdot)$, while in the last inequality the definition of α-supported. Altogether

$$\begin{aligned}
\sum_{i=0}^{D}\sum_{f\in\tilde{F}} h\Big(\Delta^\theta_i(f)\Big) &\le 32\log(D+1)\cdot\sum_{i=0}^{D}\Big(\mathrm{cost}_{\mathrm{DLA}}(z(i)) - \mathrm{cost}_{\mathrm{DLA}}(z(i+1))\Big) \\
&\le O(\log D)\cdot\mathrm{cost}_{\mathrm{DLA}}(z). \qquad (5)
\end{aligned}$$

By the monotonicity of $h(\cdot)$, Step (8) cannot increase the cost of the solution, hence the claim. ◀

4 Universal Stochastic Facility Location

In this section we sketch our approximation algorithm for UNIVFL. We first present a weaker approximation factor $O(\log\log N + \log\log \frac{d_{\max}}{d_{\min}})$. Later we will show how to refine it.

Define $g(R) \coloneqq \mathbb{P}_{A\sim\pi}[R \cap A \neq \emptyset]$. We observe that this function is monotone submodular and $g(\emptyset) = 0$. Recall that $g(c) = g(\{c\})$ for every $c \in C$. W.l.o.g. we can assume $g(c) > 0$ since otherwise we can discard c. We can define the objective function of UNIVFL for a given assignment $\varphi : C \to F$ as

$$\mathrm{cost}(\varphi) = \mathrm{conn}(\varphi) + \mathrm{open}(\varphi) = \sum_{c\in C} d(c,\varphi(c))\cdot g(c) + \sum_{f\in F} w_f \cdot g(\varphi^{-1}(f)).$$

Notice that only the connection cost changes w.r.t. MULTSFL. In more detail, the connection cost of each client c is scaled by the factor $g(c)$.

We can similarly define a configuration LP for UNIVFL, and solve it by the same arguments as in Lemma 1.2. We next use an analogous notation as for SFL. Let $\dot{x}$ be an optimal solution to this LP with poly(N) many non-zero variables. We can apply the first stage of our algorithm for SFL (described in Section 2) with essentially no changes. This will lead to a partial assignment S_1 of expected cost $\mathbb{E}[\mathrm{cost}(S_1)] \le \ln\ln N \cdot \mathrm{cost}(\dot{x})$ and serving the clients C_1, where $\mathbb{P}[c \notin C_1] \le \frac{1}{\ln N}$. Mapping the metric over an HST T and considering the restriction $\ddot{x}$ of $\dot{x}$ to $C_2 \coloneqq C \setminus C_1$, we obtain that $\mathbb{E}[\mathrm{cost}_{HST}(\ddot{x})] = O(\mathrm{cost}(\dot{x}))$. A reduction similar to the one in Lemma 3.2 works also in this case (since the scaling of the fractional solution is done on a per-client base). However in this case $D = O(\log \frac{d_{\max}}{d_{\min}})$ (since we did not reduce the ratio $\frac{d_{\max}}{d_{\min}}$ in a preprocessing step). Hence we can apply the result from Lemma 3.3 to obtain an assignment covering C_2 of expected cost $O(\log\log \frac{d_{\max}}{d_{\min}})\cdot\mathrm{cost}(\dot{x})$. This concludes the sketch of the $O(\log\log N + \log\log \frac{d_{\max}}{d_{\min}})$ approximation.

We next improve this bound via a preprocessing step. Recall that $0 < \pi_{\min} := \min_{c \in C}\{g(c)\}$. We first scale the ratio $d_{\max}/d_{\min}$. Let us guess[4] the largest distance $L = \max_{c \in C}\{d(c, \mathrm{opt}(c))\}$ in some optimal (universal) solution opt. Notice that $\mathrm{cost}(\mathrm{opt}) \geq \pi_{\min} L$. We use essentially the same arguments as in Lemma 1.8, we can enforce that $d_{\max} \leq NL$ and $d_{\min} \geq \frac{\varepsilon}{n}\pi_{\min} L$. Hence we obtain $\frac{d_{\max}}{d_{\min}} \leq \frac{nN}{\varepsilon \pi_{\min}}$.

Now let us reduce the number of facilities m to $O(n + \log \frac{1}{\pi_{\min}})$ (hence N as well). Here we use essentially the same argument as in the proof of Lemma B.1 (with $p_f = 0$). In more detail, we can assume that $m \leq 2^n$. Indeed, otherwise we can reduce the input instance to a Weighted Set Cover instance (that we can solve exactly in polynomial time) in the same way as in the mentioned lemma, with the difference that now, for $R \neq \emptyset$, we set $\kappa_R = \min_{f \in F}\{w_f \cdot g(R) + \sum_{c \in R} d(c, f) \cdot g(c)\}$. By the rest of the construction in the same lemma, we can reduce (with a constant loss in the approximation factor) our instance to one where there are $O(\log \frac{d_{\max}}{d_{\min}}) = O(\log \frac{n2^n}{\varepsilon \pi_{\min}}) = O(n + \log \frac{1}{\pi_{\min}})$ facilities per client. Altogether we reduce N to $N' = O(n(n + \log \frac{1}{\pi_{\min}}))$. Now we can apply again the above scaling trick over the distances (with N replaced by N') to obtain distances d' which satisfy:

$$\frac{d'_{\max}}{d'_{\min}} \leq \frac{nN'}{\varepsilon \pi_{\min}} = O\left(\frac{n^3 + n^2 \log \frac{1}{\pi_{\min}}}{\pi_{\min}}\right).$$

This leads to the approximation factor

$$O\left(\log \log \frac{d'_{\max}}{d'_{\min}} + \log \log N'\right) = O\left(\log \log \frac{n}{\pi_{min}}\right).$$

References

1 Fateme Abbasi, Marek Adamczyk, Miguel Bosch-Calvo, Jarosław Byrka, Fabrizio Grandoni, Krzysztof Sornat, and Antoine Tinguely. An $O(\log \log n)$-approximation for submodular facility location. *CoRR*, abs/2211.05474, 2022. doi:10.48550/arXiv.2211.05474.

2 Marek Adamczyk, Fabrizio Grandoni, Stefano Leonardi, and Michal Włodarczyk. When the optimum is also blind: A new perspective on universal optimization. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 35:1–35:15, 2017. doi:10.4230/LIPIcs.ICALP.2017.35.

3 Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and Euclidean k-median by primal-dual algorithms. *SIAM J. Comput.*, 49(4), 2020. doi:10.1137/18M1171321.

4 Hyung-Chan An, Mohit Singh, and Ola Svensson. LP-based algorithms for capacitated facility location. *SIAM J. Comput.*, 46(1):272–306, 2017. doi:10.1137/151002320.

5 Thomas Bosman and Neil Olver. Improved approximation algorithms for inventory problems. In *Integer Programming and Combinatorial Optimization - 21st International Conference, IPCO 2020*, pages 91–103, 2020. doi:10.1007/978-3-030-45771-6_8.

6 Jarosław Byrka and Karen Aardal. An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. *SIAM J. Comput.*, 39(6):2212–2231, 2010. doi:10.1137/070708901.

7 Jarosław Byrka, Thomas W. Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Trans. Algorithms*, 13(2):23:1–23:31, 2017. doi:10.1145/2981561.

8 Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for facility location problems. *SIAM J. Comput.*, 34(4):803–824, 2005. doi:10.1137/S0097539701398594.

[4] Throughout this paper, by guessing we mean trying all the (polynomially many) possible options. Each such options leads to a different solution, and we return the best one.

9 Chandra Chekuri and Alina Ene. Approximation algorithms for submodular multiway partition. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011*, pages 807–816, 2011. doi:10.1109/FOCS.2011.34.

10 Chandra Chekuri and Alina Ene. Submodular cost allocation problem and applications. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011*, pages 354–366, 2011. doi:10.1007/978-3-642-22006-7_30.

11 Vincent Cohen-Addad, Hossein Esfandiari, Vahab S. Mirrokni, and Shyam Narayanan. Improved approximations for Euclidean k-means and k-median, via nested quasi-independent sets. In *54th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2022*, pages 1621–1628, 2022. doi:10.1145/3519935.3520011.

12 Vincent Cohen-Addad, Fabrizio Grandoni, Euiwoong Lee, and Chris Schwiegelshohn. Breaching the 2 LMP approximation barrier for facility location with applications to k-median. In *34th ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 940–986, 2023. doi:10.1137/1.9781611977554.ch37.

13 Alina Ene, Jan Vondrák, and Yi Wu. Local distribution and the symmetry gap: Approximability of multiway partitioning problems. In *24th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013*, pages 306–325, 2013. doi:10.1137/1.9781611973105.23.

14 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004. doi:10.1016/J.JCSS.2004.04.011.

15 Fedor V. Fomin, Dieter Kratsch, and Gerhard J. Woeginger. Exact (exponential) algorithms for the dominating set problem. In *Graph-Theoretic Concepts in Computer Science, 30th International Workshop, WG 2004*, pages 245–256, 2004. doi:10.1007/978-3-540-30559-0_21.

16 Satoru Fujishige. *Submodular functions and optimization*, volume 58 of *Annals of Discrete Mathematics*. Elsevier, 2nd edition, 2005.

17 Naveen Garg, Anupam Gupta, Stefano Leonardi, and Piotr Sankowski. Stochastic analyses for online combinatorial optimization problems. In *19th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008*, pages 942–951, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347185.

18 Kishen N. Gowda, Thomas W. Pensyl, Aravind Srinivasan, and Khoa Trinh. Improved bi-point rounding algorithms and a golden barrier for k-median. In *34th ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 987–1011, 2023. doi:10.1137/1.9781611977554.ch38.

19 Fabrizio Grandoni, Anupam Gupta, Stefano Leonardi, Pauli Miettinen, Piotr Sankowski, and Mohit Singh. Set covering with our eyes closed. *SIAM J. Comput.*, 42(3):808–830, 2013. doi:10.1137/100802888.

20 Fabrizio Grandoni, Rafail Ostrovsky, Yuval Rabani, Leonard J. Schulman, and Rakesh Venkat. A refined approximation for Euclidean k-means. *Inf. Process. Lett.*, 176:106251, 2022. doi:10.1016/j.ipl.2022.106251.

21 Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *J. Algorithms*, 31(1):228–248, 1999. doi:10.1006/JAGM.1998.0993.

22 Anupam Gupta, Martin Pál, R. Ravi, and Amitabh Sinha. Sampling and cost-sharing: Approximation algorithms for stochastic optimization problems. *SIAM J. Comput.*, 40(5):1361–1401, 2011. doi:10.1137/080732250.

23 Shalmoli Gupta. *Approximation algorithms for clustering and facility location problems*. PhD thesis, University of Illinois Urbana-Champaign, USA, 2018. URL: https://hdl.handle.net/2142/102419.

24 Nicole Immorlica, David R. Karger, Maria Minkoff, and Vahab S. Mirrokni. On the costs and benefits of procrastination: Approximation algorithms for stochastic combinatorial optimization problems. In *15th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004*, pages 691–700, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982898.

25 Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM (JACM)*, 48(4):761–777, 2001. doi:10.1145/502090.502096.

26 Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM (JACM)*, 50(6):795–824, 2003. doi:10.1145/950620.950621.

27 Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM (JACM)*, 48(2):274–296, 2001. doi:10.1145/375827.375845.

28 Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Inf. Comput.*, 222:45–58, 2013. doi:10.1016/J.IC.2012.01.007.

29 Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. *SIAM J. Comput.*, 45(2):530–547, 2016. doi:10.1137/130938645.

30 Mohammad Mahdian, Yinyu Ye, and Jiawei Zhang. Approximation algorithms for metric facility location problems. *SIAM J. Comput.*, 36(2):411–432, 2006. doi:10.1137/S0097539703435716.

31 Adam Meyerson. Online facility location. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001*, pages 426–431, 2001. doi:10.1109/SFCS.2001.959917.

32 Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1999.

33 David B. Shmoys, Chaitanya Swamy, and Retsef Levi. Facility location with service installation costs. In *15th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004*, pages 1088–1097, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982953.

34 David B. Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems (extended abstract). In *29th Annual ACM Symposium on the Theory of Computing, STOC 1997*, pages 265–274, 1997. doi:10.1145/258533.258600.

35 Zoya Svitkina and Éva Tardos. Facility location with hierarchical facility costs. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006*, pages 153–161, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109576.

36 Zoya Svitkina and Éva Tardos. Facility location with hierarchical facility costs. *ACM Trans. Algorithms*, 6(2):37:1–37:22, 2010. doi:10.1145/1721837.1721853.

37 Jiawei Zhang, Bo Chen, and Yinyu Ye. A multiexchange local search algorithm for the capacitated facility location problem. *Math. Oper. Res.*, 30(2):389–403, 2005. doi:10.1287/MOOR.1040.0125.

A Some Omitted Proofs about SFL

Proof of Lemma 1.2. Considering the dual of (Conf-LP):

$$\max \Big\{ \sum_{c \in C} \alpha_c + \sum_{f \in F} \beta_f : \sum_{c \in R} \alpha_c + \beta_f \leq g(R) + \sum_{c \in R} d(c,f),\ \forall R \subseteq C,\ \forall f \in F \Big\}. \tag{Conf-DLP}$$

Notice that for fixed α and β, the functions $g_f(R) \coloneqq g(R) + \sum_{c \in R} d(c,f) - \sum_{c \in R} \alpha_c - \beta_f$ are submodular. Thus, a call of a separation oracle on (Conf-DLP) is equivalent to a minimization of all functions $g_f(\cdot)$, which can be done using polynomially many oracle calls of $g(\cdot)$ [25]. Therefore, an optimal primal solution with $\mathrm{poly}(N)$ many non-zero variables for (Conf-LP) can be found in polynomial time [32, Corollary 14.1g(v)]. ◀

Proof of Lemma 1.7. Let $I = (C, F, d, g(\cdot))$ be the considered instance of SFL. Consider the complete weighted graph on nodes $C \cup F$, with weights induced by d. For each client c, let $f(c)$ be the facility closest to c. We create a dummy facility $f'(c)$ and add a dummy edge $\{c, f'(c)\}$ of weight $d(c, f(c))$. Let F' be the set of newly created facilities. Observe

that $|F'| = n$. Finally we remove F and consider the metric d' over $C \cup F'$ induced by the distances over the resulting graph. Let $I' = (C, F', d', g(\cdot))$ be the obtained instance of SFL. Given a solution φ' for I', we obtain a solution φ for I by simply assigning to $f(c)$ each client c' assigned to $f'(c)$ in φ'.

Let us analyze the approximation factor introduced by this reduction. We first observe that $\mathrm{cost}_I(\varphi) \leq \mathrm{cost}_{I'}(\varphi')$. Indeed, $\mathrm{open}_I(\varphi) = \mathrm{open}_{I'}(\varphi')$. Furthermore, for each each client c' assigned to $f'(c)$ by φ', the associated connection cost w.r.t. I is $d(c', f(c)) \leq d(c', c) + d(c, f(c)) = d'(c', f'(c))$. Hence $\mathrm{conn}_I(\varphi) \leq \mathrm{conn}_{I'}(\varphi')$.

Next consider an optimal solution opt for I. For each facility f with $\mathrm{opt}^{-1}(f) \neq \emptyset$, let $c \in \mathrm{opt}^{-1}(f)$ be the client closest to f. We define a solution opt$'$ for I' by assigning all the clients in $\mathrm{opt}^{-1}(f)$ to $f'(c)$. Again, $\mathrm{open}_I(\varphi) = \mathrm{open}_{I'}(\varphi')$. For each client c' assigned to f in opt, its connection cost in I' is

$$d'(c', f'(c)) = d(c, c') + d(c, f(c)) \leq d(c', f) + d(c, f) + d(c, f(c)) \leq d(c', f) + 2d(c, f) \leq 3d(c', f).$$

Hence $\mathrm{conn}_{I'}(\mathrm{opt}') \leq 3\,\mathrm{conn}_I(\mathrm{opt})$. The claim follows. ◀

Proof of Lemma 1.8. Let us guess the value $L = \max_{c \in C} d(c, \mathrm{opt}(c))$ for some optimal solution opt. W.l.o.g. assume $L > 0$, otherwise the problem is trivial. Consider the complete weighted graph on nodes $C \cup F$ with weights induced by d. Remove the edges of weight larger than L. We next compute a feasible solution in each connected component of the resulting graph separately. Notice that this part of the reduction is approximation preserving since no client can be assigned to a facility in a different connected component in opt.

Let C' and F' be the clients and facilities, resp., in one such connected component G', $X' = C' \cup F'$, and d' be the metric induced by the distances in G'. Consider the corresponding SFL instance $I' = (C', F', d', g(\cdot))$. Notice that in each such instance I' one has $d'_{\max} \leq NL$. We next change the location of elements of X' as follows. We consider the ball $B(x) := \{y \in X' : d'(x, y) \leq \frac{\varepsilon}{2n} L\}$ of radius $\frac{\varepsilon}{2n} L$ around each $x \in X'$. Let $\mathcal{I}$ be a maximal (independent) set of such balls so that, if $B(x), B(y) \in \mathcal{I}$ for $x \neq y$, then $B(x) \cap B(y) = \emptyset$. For each y with $B(y) \notin \mathcal{I}$, we consider any $B(x) \in \mathcal{I}$ with $B(x) \cap B(y) \neq \emptyset$ (which must exist since $\mathcal{I}$ is maximal) and colocate y with x. Let $I'' = (C', F', d'', g(\cdot))$ be the resulting instance of SFL. Observe that $d''_{\max} \leq NL$ and $d''_{\min} \geq \frac{\varepsilon}{n} L$.

Let $\tilde{I}$ be the union of all the instances I'', and $\tilde{d}$ be the associated distances (where inter-component distances can be considered to be $+\infty$). Given a solution φ for $\tilde{I}$ (obtained by the union of all the solutions obtained for each instance I''), we return exactly the same solution φ for I.

Let us analyze the approximation factor. Notice that $\mathrm{open}_I(\varphi) = \mathrm{open}_{\tilde{I}}(\varphi)$. Furthermore, for each client c, $d(c, \varphi(c)) \leq \tilde{d}(c, \varphi(c)) + \frac{2\varepsilon}{n} L$, where in the latter term we consider the fact that each client and facility is moved at most at distance $\frac{\varepsilon}{n} L$ from the original location. Hence $\mathrm{conn}_I(\varphi) \leq \mathrm{conn}_{\tilde{I}}(\varphi) + 2\varepsilon L$. Given an optimum solution opt for I, by a symmetric argument one has $\mathrm{cost}_{\tilde{I}}(\mathrm{opt}) \leq \mathrm{cost}_I(\mathrm{opt}) + 2\varepsilon L \leq (1 + 2\varepsilon)\,\mathrm{cost}_I(\mathrm{opt})$, where we used the fact that $\mathrm{cost}_I(\mathrm{opt}) \geq L$. Altogether an $\alpha \geq 1$ approximation algorithm for each instance I'' implies an $\alpha(1 + 2\varepsilon) + 2\varepsilon \leq \alpha(1 + 4\varepsilon)$ approximation for I.

Finally, we scale the distance d'' and $g(\cdot)$ by the same factor $\frac{2n}{\varepsilon L}$ so that $d''_{\min} = 2$ and $d''_{\max} \leq \frac{2nN}{\varepsilon}$. Clearly this final scaling is approximation preserving. ◀

B Generalizations of SFL

In this section we discuss some generalizations of SFL.

B.1 Reduction of the Number of Facilities

In this section we consider the generalization of SFL, next called Affine SFL, where the opening cost of each facility f with assigned clients $R \neq \emptyset$ is $g_f(R) \coloneqq p_f + w_f \cdot g(R)$, where $p_f, w_f \geq 0$ are input values. Notice that this generalizes SFL with Additive (resp., Multiplicative) Opening Costs. We also observe that each $g_f(\cdot)$ is non-negative monotone submodular.

We show how to reduce to the case where $m = \text{poly}(n)$ (hence $N = \text{poly}(n)$) while loosing a constant factor in the approximation. We will use this reduction in the following sections to convert an $O(\log \log N)$ approximation into an $O(\log \log n)$ one.

▶ **Lemma B.1.** *For any constant $\varepsilon > 0$, there is a $(3 + 37\varepsilon)$-approximate reduction from* Affine *SFL to the special case where the number of facilities is $O_\varepsilon(n^3)$.*

Proof. First of all, consider the case $m \geq 2^n$. In this case we can solve the problem optimally in polynomial time via the following reduction to the Weighted Set Cover problem. For an instance $I = (C, F, d, g(\cdot))$ of Affine SFL, consider the instance $J = (\mathcal{U}, \mathcal{R}, \kappa)$ of Weighted Set Cover with universe $\mathcal{U} = C$, set collection $\mathcal{R} = 2^C$ and weight function κ given as $\kappa_R = 0$ if $R = \emptyset$ and $\kappa_R = \min_{f \in F}(p_f + w_f \cdot g(R) + \sum_{c \in R} d(c, f))$ for $R \in 2^C \setminus \{\emptyset\}$ (which can be computed in $\text{poly}(N)$ time). Notice that $2^{|\mathcal{U}|} = 2^n$ which is polynomially bounded in the input size of I. The optimal solution to J induces a solution of exactly the same cost to I and vice versa. There is a simple dynamic program which solves Weighted Set Cover in time $O(2^{|U|} \cdot |U| \cdot |\mathcal{R}|)$ [15, Lemma 2]. Applying this algorithm to J, one obtains an optimal solution for the input instance I in time $O(2^n \cdot \text{poly}(n, m))$, which is polynomial in m.

Hence it remains to consider the case $m \leq 2^n$. We show how to reduce the number of facilities to $O_\varepsilon(n^2 \log(nN)) = O_\varepsilon(n^3)$, while losing the approximation factor in the claim. By exactly the same reduction as in Lemma 1.8, we can assume that in the input metric d the maximum distance is $0 < d_{\max} \leq NL$ and the minimum non-zero distance is $d_{\min} \geq \frac{\varepsilon}{n} L$ while loosing a factor $(1 + 4\varepsilon)$ in the approximation. Here L is some value that lower bounds the cost of a given optimum solution opt. Let us guess the largest value P of p_f over the facilities with at least one assigned client in opt. We discard all the facilities f with $p_f > P$. Now, assuming $P > 0$, we replace each p_f with the value $p'_f \coloneqq \lceil \frac{p_f \cdot n}{\varepsilon P} \rceil \cdot \frac{\varepsilon P}{n}$ ($p'_f = p_f$ for $P = 0$). Notice that this can only increase the cost of a given solution φ, however this increase is upper bounded by $n \cdot \frac{\varepsilon P}{n} \leq \varepsilon \cdot \text{cost}_I(\text{opt})$, where I is the input instance of the problem. Hence this reduction preserves the approximation guarantee up to a factor $1 + \varepsilon$. After this reduction, the set $\mathcal{P}'$ of different possible values of p'_f has cardinality at most $\frac{n}{\varepsilon}$.

Let $I = (C, F, d, p', w, g(\cdot))$ be the instance of Affine SFL obtained after the above two reductions. Consider the complete edge-weighted graph on nodes $C \cup F$, with weights induced by d. We modify this graph as follows. For each client c and value $p' \in \mathcal{P}'$, we consider the set of facilities $F_{p'}$ with $p'_f = p'$. Let $F_{p'}(c, i)$, $i \geq 0$, be the facilities in $F_{p'}$ whose distances from c are in the range $[\frac{\varepsilon}{n} L \cdot (1 + \varepsilon)^i, \frac{\varepsilon}{n} L \cdot (1 + \varepsilon)^{i+1})$. We also define the set $F_{p'}(c, -1)$ of the facilities in $F_{p'}$ at distance 0 from c. Notice that there are at most $1 + \lceil \log_{1+\varepsilon} \frac{nN}{\varepsilon} \rceil$ sets $F_{p'}(c, i)$ which are non-empty. For each $F_{p'}(c, i) \neq \emptyset$, we choose a facility $f = f_{p'}(c, i)$ with minimum value of w_f. We create a dummy facility $f' = f'_{p'}(c, i)$ with opening cost $g'_{f'}(C') = p' + w_f \cdot g(C')$ for $C' \neq \emptyset$, and add a dummy edge $\{c, f'\}$ of weight $d(c, f)$. Let F'

be the set of dummy facilities. Notice that, considering also the previous reduction, one has $|F'| \leq n \cdot \frac{n}{\varepsilon} \cdot (1 + \lceil \log_{1+\varepsilon} \frac{nN}{\varepsilon} \rceil) = O(n^2 \log(nN))$. We remove the original facilities F, and let d' be the metric given by the distances in the resulting graph G' on nodes $C \cup F'$. We solve the problem on the resulting instance $I' = (C, F', d', p', w, g(\cdot))$. Given a solution φ' for I', we obtain a solution φ for I naturally as follows: if $\varphi'(c') = f'_{p'}(c, i)$, we assign c' to $f_{p'}(c, i)$.

Let us analyze the approximation factor of this final reduction. The opening costs of φ and φ' are identical. Furthermore, for each client c' assigned to $f = f_{p'}(c, i)$ in φ, and for $f' = f'_{p'}(c, i)$, one has $d(c', f) \leq d(c', c) + d(c, f) = d'(c', c) + d'(c, f') = d'(c', f')$. Hence $\mathrm{cost}_I(\varphi) = \mathrm{cost}_{I'}(\varphi')$.

Next consider an optimum solution opt for I. We construct a feasible solution opt$'$ for I' as follows. Let $S^f \neq \emptyset$ be the clients assigned to some $f \in F$ in opt. Recall that the opening cost of f is $g'_f(S^f) = p'_f + w_f \cdot g(S^f)$. Let $c \in S^f$ be the client at minimum distance $d(c, f)$ from f. Define i as -1 if $d(c, f) = 0$, and otherwise, i such that $d(c, f) \in [\frac{\varepsilon}{n} L \cdot (1+\varepsilon)^i, \frac{\varepsilon}{n} L \cdot (1+\varepsilon)^{i+1})$. In opt$'$ we reassign all the clients in S^f to $f' = f'_{p'_f}(c, i)$. The opening cost associated with f' in opt$'$ is no larger than the corresponding cost in opt since

$$p'_{f'} + w_{f'} \cdot g(S^{f'}) = p'_f + w_{f'} \cdot g(S^f) \leq p'_f + w_f \cdot g(S^f).$$

In the last inequality above we used the fact that $f \in F_{p'_f}(c, i)$ and $f_{p'_f}(c, i)$ is the facility in the latter set with minimum w_f value. The connection cost of each $c' \in S^f$ w.r.t. opt$'$ satisfies

$$\begin{aligned} d'(c', f') &= d'(c', c) + d'(c, f') = d(c, c') + d(c, f_{p'_f}(c, i)) \\ &\leq d(c', f) + d(c, f) + (1 + \varepsilon) d(c, f) \leq (3 + \varepsilon) d(c', f). \end{aligned}$$

Altogether, $\mathrm{cost}_{I'}(\mathrm{opt}') \leq (3 + \varepsilon)\, \mathrm{cost}_I(\mathrm{opt})$. Considering also the first two reductions, we obtain a global reduction which preserves the approximation guarantee up to a factor $(1 + 4\varepsilon)(1 + \varepsilon)(3 + \varepsilon) \leq 3 + 37\varepsilon$. ◀

B.2 SFL with Multiplicative Opening Costs

In this section we sketch the proof of Theorem 1.3. By Lemma B.1, it is sufficient to provide an $O(\log \log N)$ approximation.

For $f \in F$ and $R \subseteq C$ let $g_f(R) \coloneqq w_f \cdot g(R)$. Note that $g_f(\cdot)$ is submodular, monotone and has $g(\emptyset) = 0$ for every $f \in F$. For any (partial) assignment $S = (S^f)$ and any vector $(x^f_R)^{f \in F}_{R \subseteq C}$ let also $\mathrm{open}'(S) \coloneqq \sum_{f \in F} g_f(S^f)$, resp. $\mathrm{open}'(x) \coloneqq \sum_{f \in F} \sum_{R \subseteq C} g_f(R) \cdot x^f_R$ and $\mathrm{cost}'(S) \coloneqq \mathrm{open}'(S) + \mathrm{conn}(S)$ resp. $\mathrm{cost}'(x) \coloneqq \mathrm{open}'(x) + \mathrm{conn}(x)$.

By these definitions, the LP-relaxation of the MULTSFL is given by the constraints from (Conf-LP) and the objective $\mathrm{cost}'(\cdot)$. In particular, the LP-relaxation of MULTSFL can be solved with the approach from Lemma 1.2. We keep the merging rule defined in Section 1.4 and the sampling procedure from Section 2. It is easy to verify that the vector $\ddot{x}$ resulting from this procedure fulfills Lemma 2.2 w.r.t. open$'$ instead of open.

We reduce MULTSFL to a similar problem to DLA which we call DLA* which is the same problem as DLA and with the same input variables as DLA, additional inputs $\tilde{w}_f \geq 0$ for every $f \in \tilde{F}$ and cost $\mathrm{cost}^*_{\mathrm{DLA}}(\varphi) = \sum_{f \in \tilde{F}} h_f(\varphi^{-1}(f))$ where $h_f(\cdot) \coloneqq \tilde{w}_f h(\cdot)$ for every $f \in \tilde{F}$. Its convex relaxation is given by the constraints in (DLA-CP) with the cost function $\mathrm{cost}^*_{\mathrm{DLA}}(z) \coloneqq \sum_{f \in \tilde{F}} \hat{h}_f(z^f)$ (where $\hat{h}_f$ is the Lovász extension of h_f). The reduction described in Lemma 3.2 can be reproduced to reduce MULTSFL to DLA*. We define the input values

of DLA* w.r.t. MULTSFL in the same way we define the input values of DLA w.r.t. SFL, with additionally $\tilde{w}_f = w_f$ for every $f \in F$. Notice that $h_f(\cdot) = \tilde{w}_f h(\cdot) = g_f(\cdot) = w_f g(\cdot)$. Every reasoning made in the proof of Lemma 3.2 stays valid.

We now adjust Algorithm 1 for DLA* as follows: in Step 3, we select the facility $f_v \in \tilde{F}_v$ with minimum weight $\tilde{w}_{f_v}$. In the if-clause 4, we search and verify for supportedness w.r.t. h_{f_v} instead of h (which is equivalent unless $\tilde{w}_{f_v} = 0$, in which case $L_\theta(z^{f_v})$ is supported for every θ). Since the new algorithm functions exactly like Algorithm 1, except for an arbitrary selection step becoming determined (in particular, the new algorithm is a possible implementation of Algorithm 1), its correctness is implied by the correctness of Algorithm 1.

Notice that since f_v in Step 3 is now chosen to have minimal weight, we have for any $f' \in \tilde{F}_v \setminus \{f_v\}$

$$\hat{h}_{f_v}\big(z^{f_v} + z^{f'}\big) \leq \hat{h}_{f_v}\big(z^{f_v}\big) + \hat{h}_{f_v}\big(z^{f'}\big) \leq \hat{h}_{f_v}\big(z^{f_v}\big) + \hat{h}_{f'}\big(z^{f'}\big),$$

which means that the cost of z does not increase at any time by the arguments as before. Also, notice that since h_f is submodular, monotone and $h_f(\emptyset) = 0$ we can apply Lemma 3.4 with respect to h_{f_v} instead of h. Thus, the cost of the sets added at Step 5 and Step 7 is still bounded as in (4) and (5).

B.3 SFL with Additive Opening Costs

In this section we sketch the proof of Theorem 1.4. As in the previous section, by Lemma B.1, it is sufficient to provide an $O(\log \log N)$ approximation.

Similarly to the previous section, we define the set function $g_f(\cdot)$ as $g_f(R) = g(R) + p_f$ for $R \neq \emptyset$ and $g_f(\emptyset) = 0$. As argued in the previous section, we can find an optimum to the LP relaxation of ADDSFL and reduce it to the problem DLA* as defined in the last section, but with input weights $\tilde{p}_f$ instead of $\tilde{w}_f$ and $h_f(\cdot)$ as $h_f(R) \coloneqq h(R) + p_f$ for $R \neq \emptyset$, and $h_f(\emptyset) = 0$.

We adapt Algorithm 1 like in the previous section: in Step 3, we select the facility $f_v \in \tilde{F}_v$ with minimum weight $\tilde{p}_{f_v}$. In the if-clause 4, we search and verify for supportedness w.r.t. h_{f_v} instead of h. The correctness of the new algorithm here is given by the same argument as in the previous section. Notice that by (2) we have $\hat{h}_f(z) = \hat{h}(z) + p_f \cdot \max_{c \in \bar{C}} z_c$, which implies $\hat{h}_{f_v}(z^{f_v} + z^{f'}) \leq \hat{h}_{f_v}(z^{f_v}) + \hat{h}_{f'}(z^{f'})$ with f_v chosen as in Step 3 in Algorithm 1. The cost of z does therefore not increase throughout the algorithm. Bounding the cost of sets added to the solution at Step 5 and Step 7 can be done, like for MULTSFL, by applying Lemma 3.4 to h_{f_v}.

Parameterized Approximation For Robust Clustering in Discrete Geometric Spaces

Fateme Abbasi
University of Wrocław, Poland

Sandip Banerjee
IDSIA, USI-SUPSI, Lugano, Switzerland

Jarosław Byrka
University of Wrocław, Poland

Parinya Chalermsook
Aalto University, Finland

Ameet Gadekar
Bar-Ilan University, Ramat-Gan, Israel

Kamyar Khodamoradi
University of Regina, Canada

Dániel Marx
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Roohani Sharma
University of Bergen, Norway

Joachim Spoerhase
University of Sheffield, UK

Abstract

We consider the well-studied Robust (k, z)-Clustering problem, which generalizes the classic k-Median, k-Means, and k-Center problems and arises in the domains of robust optimization [Anthony, Goyal, Gupta, Nagarajan, Math. Oper. Res. 2010] and in algorithmic fairness [Abbasi, Bhaskara, Venkatasubramanian, 2021 & Ghadiri, Samadi, Vempala, 2022]. Given a constant $z \geq 1$, the input to Robust (k, z)-Clustering is a set P of n points in a metric space (M, δ), a weight function $w : P \to \mathbb{R}_{\geq 0}$ and a positive integer k. Further, each point belongs to one (or more) of the m many different groups $S_1, S_2, \ldots, S_m \subseteq P$. Our goal is to find a set X of k centers such that $\max_{i \in [m]} \sum_{p \in S_i} w(p)\delta(p, X)^z$ is minimized.

Complementing recent work on this problem, we give a comprehensive understanding of the parameterized approximability of the problem in geometric spaces where the parameter is the number k of centers. We prove the following results:

- **(i)** For a universal constant $\eta_0 > 0.0006$, we devise a $3^z(1-\eta_0)$-factor FPT approximation algorithm for Robust (k, z)-Clustering in *discrete* high-dimensional Euclidean spaces where the set of potential centers is finite. This shows that the lower bound of 3^z for general metrics [Goyal, Jaiswal, Inf. Proc. Letters, 2023] no longer holds when the metric has geometric structure.
- **(ii)** We show that Robust (k, z)-Clustering in discrete Euclidean spaces is $(\sqrt{3/2} - o(1))$-hard to approximate for FPT algorithms, even if we consider the special case k-Center in logarithmic dimensions. This rules out a $(1+\epsilon)$-approximation algorithm running in time $f(k, \epsilon)\mathsf{poly}(m, n)$ (also called efficient parameterized approximation scheme or EPAS), giving a striking contrast with the recent EPAS for the *continuous* setting where centers can be placed anywhere in the space [Abbasi et al., FOCS'23].
- **(iii)** However, we obtain an EPAS for Robust (k, z)-Clustering in discrete Euclidean spaces when the dimension is sublogarithmic (for the discrete problem, earlier work [Abbasi et al., FOCS'23] provides an EPAS only in dimension $o(\log \log n)$). Our EPAS works also for metrics of sub-logarithmic doubling dimension.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Facility location and clustering

Keywords and phrases Clustering, approximation algorithms, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.6

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2305.07316 [1]

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 6; pp. 6:1–6:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding Fateme Abbasi and Jarosław Byrka were supported by the Polish National Science Centre (NCN) Grant 2020/39/B/ST6/01641. Sandip Banerjee acknowledges the support by SNSF Grant 200021 200731/1 and also the support of Polish National Science Centre (NCN) Grant 2020/39/B/ST6/01641 while at the University of Wrocław, Poland. Parinya Chalermsook, Kamyar Khodamoradi, and Joachim Spoerhase were supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 759557). Ameet Gadekar was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 759557) while at Aalto University, and by the Israel Science Foundation (grant No. 1042/22).

1 Introduction

Clustering is a crucial method in the analysis of massive datasets and has widespread applications in operations research and machine learning. As a consequence, optimization problems related to clustering have received significant attention from the theoretical computer science community over the years. Within the framework of center-based clustering, k-Center, k-Means, and k-Median [25, 26, 10, 4, 27] are widely regarded as the most fundamental problems.

A general notion that captures various classic clustering problems is referred to as (k, z)-Clustering in the literature, where $z \geq 1$ is a constant. In this type of problem, the input is a set P of data points (clients), a set F of centers (facilities), a metric δ on $P \cup F$, and a positive integer k. The goal is to find a set $C \subseteq F$ of k facilities that minimizes the following cost function:

$$cost(C) = \sum_{p \in P} cost(p, C)$$

where $cost(p, C) = \delta(p, C)^z$ and $\delta(p, C) = \min_{c \in C} \delta(p, c)$. Note that (k, z)-Clustering encapsulates the classical k-Median, and k-Means for $z = 1$ and $z = 2$, respectively .

Center-based clustering has cemented its place as an unsupervised learning method that has proven effective in modeling a variety of real-world problem. In most of the practical machine learning applications however, it is observed that the input data is rarely of high quality.

To tackle this challenge, we study a robust version of (k, z)-Clustering in this paper which can handle uncertainty in the input: Consider a situation where we do not have complete knowledge about the clients that will be served. In order to perform well despite this uncertainty, Anthony et al. [5] defined a concept of robustness for the k-Median problem, in which each possible scenario is represented by a group of clients and the goal is to find a solution that performs best possible even in the worst scenario. In this paper, we address the following robust version of the (k, z)-Clustering problem (called Robust (k, z)-Clustering):

Robust (k, z)-Clustering
Input: Instance (P, F, δ) with δ being a metric on $P \cup F$, positive integer k, a weight function $w\colon P \to \mathbb{R}_+$, and m groups $S_1, \ldots, S_m$ such that $S_i \subseteq P, P = \cup_{i \in [m]} S_i$.
Output: A k-element subset $X \subseteq F$ that minimizes $\max_{i \in [m]} \sum_{p \in S_i} w(p)\delta(p, X)^z$.

Let $n = |P|$. We remark that, in addition to generalizing k-Median and k-Means, the Robust (k, z)-Clustering problem encapsulates k-Center, when each group contains a distinct singleton. A similar objective has been studied in the context of *fairness*, in which

we aim to create a solution that will be appropriate for each of the specified groups of people. This problem is known in the literature as Socially Fair k-Median, recently introduced independently by Abbasi et al. [3] and Ghadiri et al. [20]. Notice that Abbasi et al. [3] introduce fair clustering with client weights being inversely proportional to the group size as a normalization. On the other hand, Anthony et al. [5] introduce robust clustering with unweighted clients. Since our definition allows arbitrary client weights, we capture both of these settings.

While k-Means, k-Median, and k-Center admit constant-factor approximations, it is not very surprising that Robust (k, z)-Clustering is harder due to its generality: Makarychev and Vakilian [29] design a polynomial-time $\mathcal{O}(\log m/\log\log m)$-approximation algorithm, which is tight under a plausible complexity assumption [8][1]. As this precludes the existence of efficient constant-factor approximation algorithms, recent works have focused on designing constant factor *parameterized* (FPT) approximation algorithms[2]. Along these lines, an FPT time $(3^z + \epsilon)$-approximation algorithm has been proposed and shown to be tight under the Gap Exponential-Time Hypothesis (Gap-ETH) [22]. When allowing a parameterization on the number of groups m (instead of k), Ghadiri et al. designed a $(5 + 2\sqrt{6} + \epsilon)^z$-approximation algorithm in $n^{\mathcal{O}(m^2)}$ time [21].

Motivated by the tight lower bounds for general discrete metrics, we focus on *geometric* spaces. Geometric spaces have a particular importance in real-world applications because data can often be represented via a (potentially large) collection of numerical attributes, that is, by vectors in a (possibly high-dimensional) geometric space. For example, in the bag-of-words model a document is represented by a vector where each coordinate specifies the frequency of a given word in that document. Such representations naturally lead to very high-dimensional data. A setting of particular interest is the high-dimensional *Euclidean space* where the metric is simply the Euclidean metric $\delta(x, y) = ||x - y||_2$.

The study of clustering problems in high-dimensional Euclidean space is an important line of research that has received significant attention in the algorithms community. It may seem intuitive to believe that it should generally (for almost any problem) be possible to algorithmically leverage the geometric structure to separate high-dimensional Euclidean from general metrics. For clustering, however, this turns out to be either false or highly non-trivial in many cases. For example, it is a long-standing open question [19] whether k-Center admits a polynomial time $(2 - \epsilon)$-approximation algorithm even in $\mathbb{R}^2$, improving the tight bound of 2 in general metrics. Interestingly enough, for the more general Euclidean k-Supplier problem, Nagarajan et al. [30] obtain an improvement over the tight bound of 3 in general metrics. The improved bounds for Euclidean k-Median and k-Means by Ahmadian et al. [4], Grandoni et al. [23], and recently by Cohen-Addad et al. [11] were breakthroughs. Concerning the more general Robust (k, z)-Clustering, the tight inapproximability bound of $\Omega(\log m/\log\log m)$ in general metric continues to hold even in the line metric [8].

Similarly, the regime of FPT approximation algorithms for Euclidean clustering problems has received significant attention. Classic works design an Efficient Parameterized Approximation Scheme (EPAS), that is, a $(1 + \epsilon)$-approximation in $f(k, \epsilon)\mathsf{poly}(n)$ time, for k-Center [6] as well as for k-Median and k-Means [28]. Recent research focuses on the design of so-called coresets [31, 16] whose existence implies an EPAS if their size only depends on k and the error parameter ϵ.

[1] Note that they proved this factor for Robust k-Median, and the hardness result holds even in the line metric, unless $\mathsf{NP} \subseteq \cap_{\delta>0}\mathsf{DTIME}(2^{n^\delta})$.

[2] Throughout the paper, parameterization refers to the natural parameter k.

In the real space $\mathbb{R}^d$, it is important to distinguish between the *discrete* and the *continuous* settings. In the discrete setting, both the point set P and the candidate center set F are finite subsets of $\mathbb{R}^d$ while in the continuous setting, centers can be chosen anywhere in the metric space, that is, $F = \mathbb{R}^d$. A separate line of research has studied the contrast between continuous and discrete versions. For example, while discrete clustering variants are clearly polynomial-time solvable for constant k by trivial enumeration, the continuous versions of k-Center and k-Median are known to be NP-hard even for $k = 2$ [18] in high-dimensional Euclidean space. Also in terms of polynomial-time approximability, stronger lower bounds were shown by Cohen-Addad et al. [14] for the continuous versions. Indeed, there have been systematic research efforts in understanding these geometric clustering problems [15, 13, 14]. A recent result [2] implies an EPAS for Robust (k, z)-Clustering in continuous Euclidean spaces (of any dimension), as well as in discrete Euclidean spaces in "relatively low" dimension, that is, dimension $o(\log \log n)$.

The main goal of this paper is to develop comprehensive understanding for Robust (k, z)-Clustering in high-dimensional discrete Euclidean spaces, in particular, when the dimension is at least $\Omega(\log \log n)$.

1.1 Our contributions

First, motivated by a factor-$(3^z - o(1))$ hardness of FPT approximation for Robust (k, z)-Clustering in general metrics [22], a natural question is whether the structures of Euclidean spaces can be leveraged to obtain better results in high dimensions. While it is intuitive to believe that such an improvement should generally (for almost any problem) be possible in geometric spaces, we note that this is sometimes not the case: The polynomial time inapproximability of Robust (k, z)-Clustering remains $\Omega(\log m / \log \log m)$ even in the line metric [8].

Our first result gives an affirmative answer to this question.

▶ **Theorem 1.1** (High-Dimensional Euclidean Space). *There exists a universal constant $\eta_0 > 0.0006$ such that for any constant positive integer z, there is a factor $3^z(1 - \eta_0)$ FPT approximation algorithm for Robust (k, z)-Clustering in discrete Euclidean space $\mathbb{R}^d$ that runs in time $2^{\mathcal{O}(k \log k)} \mathsf{poly}(m, n, d)$.*

We remark that, first, our running time has only a polynomial dependency on d. Secondly, the key take-home message for Theorem 1.1 is not about a concrete approximation factor, but rather a "proof of concept" that the factor of 3^z can be improved. Conceptually, this result shows that geometric spaces are indeed easier for Robust (k, z)-Clustering than general metric spaces in the FPT world, in contrast to the polynomial-time world, where they seem to be equally hard [8]. The proof of this theorem relies on a new geometric insight that leverages the properties of Euclidean spaces (that do not hold in general metric spaces). The analysis of our algorithms "reduces" the global analysis of approximation factor to a "local" geometric instance, in which it suffices to merely analyze the behavior of three points in the Euclidean spaces.

Next, we focus on obtaining a complete characterization of the existence of EPAS in discrete Euclidean spaces. Recall that an EPAS exists in continuous Euclidean spaces of any dimensions and in discrete Euclidean spaces of dimension $o(\log \log n)$ [2], so to complete the landscape, we need to understand the discrete Euclidean spaces of dimension $\Omega(\log \log n)$.

In the next theorem, we prove that even the special case of k-Center does not admit an EPAS. This hardness holds for any ℓ_q metric and even in dimension $O(k \log n)$. More formally, we prove the following theorem.

▶ **Theorem 1.2** (Hardness in Discrete Euclidean Space). *For any constant positive integer q and any positive constant $\eta > 0$, there exists a function $d(k,n) = O(k \log n)$ such that there is no factor-$(3/2-\eta)^{1/q}$ FPT approximation algorithm for the discrete k-Center problem in $\mathbb{R}^{d(k,n)}$ under the ℓ_q metric unless W[1]= FPT. Moreover, for the ℓ_2 metric this hardness holds even for some dimension $O(\log n)$, that is, independently of k.*

Our result therefore highlights the interesting contrast between the discrete and continuous settings in high-dimensional Euclidean spaces, which has been systematically studied in recent years [15, 13, 14]. As mentioned, the continuous setting admits an EPAS [2], so our hardness result implies that the discrete setting is harder than the continuous counterpart. This is contrast to the results Cohen-Addad et al. [14] mentioned earlier showing that continuous variants of k-Median and k-Means in geometric spaces are apparently harder to approximate (in polynomial time) than their discrete part as well as the different complexity status of continuous and discrete clustering in high-dimensional spaces even for $k = 2$ [18]. This shows a rather mysterious behavior of clustering problems in geometric spaces.

Our next theorem completes the FPT-approximability landscape by designing an EPAS for the problem in doubling metrics of dimension $d = o_k(\log n)$[3]. We remark that the doubling dimension of the d-dimensional discrete Euclidean metric is $\Theta(d)$, that is, we obtain an EPAS for discrete Euclidean $o_k(\log n)$-dimensional spaces in particular.

▶ **Theorem 1.3** (EPAS for Doubling Metric of Sub-Logarithmic Dimension). *There is an algorithm that computes $(1+\epsilon)$-approximate solution, for every $\epsilon > 0$, for Robust (k,z)-Clustering in the metric of doubling dimension d in time $f(k,d,\epsilon,z)\mathsf{poly}(m,n)$, where $f(k,d,\epsilon,z) = \left(\left(\frac{2^z}{\epsilon}\right)^d k \log k\right)^{\mathcal{O}(k)}$.*

Note that the above theorem yields an EPAS for Robust (k,z)-Clustering when $d = o_k(\log n)$. Together with Theorem 1.2, this theorem gives (almost) a dichotomy result for the existence of EPAS: An EPAS exists for Robust (k,z)-Clustering in $o_k(\log n)$ dimension, while obtaining an EPAS is W[1]-hard in $\Omega_k(\log n)$ dimension. This leads to an almost complete understanding on the existence of EPAS in continuous and discrete Euclidean spaces.

2 Overview of Techniques

Improved FPT Approximation in High-Dimensional Discrete Euclidean Space

Our algorithm underlying Theorem 1.1 is a slight modification of the factor-$(3^z+\epsilon)$ FPT approximation algorithm for general metrics by Goyal and Jaiswal [22]. Our main technical contribution lies in the improved analysis. A key component of the analysis by Goyal and Jaiswal is a simple projection property of metric spaces (see Lemma 2.1 below). We argue that under minor additional assumptions, this property can be strengthened in Euclidean space. The resulting *assigment lemma* (see Lemma 3.1) is at the heart of our analysis and its proof relies on several new ideas and technically involved ingredients.

We briefly review the algorithm by Goyal and Jaiswal [22]. Their algorithm consists of two main steps. First, they compute a (κ,λ)-bicriteria solution $B \subseteq F$, that is, the cost of B is bounded by $\kappa\mathsf{OPT}$ and the cardinality of B is bounded by λk. Specifically, they obtain guarantees $\kappa = 1+\epsilon$ and $\lambda = \mathcal{O}\left(\log^2 n/\epsilon^2\right)$ for sufficiently small $\epsilon > 0$. In the second step, they extract a feasible solution from the (infeasible) bi-criteria solution B by enumerating all k-subsets of B and outputting the one of minimum cost.

[3] We use notation $o_k(\cdot)$ to hide multiplicative factors depending only on k.

Their analysis is based on proving the existence of a k-subset of B whose cost is at most $(3^{z-1}(\kappa+2))\mathsf{OPT}$, which can be bounded by $(3^z+\epsilon)\mathsf{OPT}$ assuming z being constant. Since the algorithm enumerates all k-subsets, this provides an upper bound on the cost of the algorithm. The key component of their existential argument is the following simple property of metric spaces, which we call *projection lemma*. It is convenient to think of O as an optimal solution and B as a bicriteria solution with $|B| > |O|$ but the lemma holds for any sets B, O.

▶ **Lemma 2.1** (Projection Lemma). *Let (Y,δ) be a metric space, and $B \subseteq Y$. Then for any set $O \subseteq Y$, there exists an assignment $\sigma\colon O \to B$ such that, for all $o \in O$ and $y \in Y$, we have*

$$\delta(y,\sigma(o)) \leq 2\delta(y,o) + \delta(y,B)\,. \tag{1}$$

Intuitively, their lemma allows them to "project" the optimal solution O onto a k-subset $\sigma(O) \subseteq B$ of the bicriteria solution so that for any client $y \in Y$, the distance $\delta(y,\sigma(O))$ can be charged to $\delta(y,O)$ and $\delta(y,B)$. If fact, the number 3 in the approximation factor $3^z+\epsilon$ corresponds to the sum $(2+1)$ of the coefficients in front of $\delta(y,o)$ and $\delta(y,B)$.

In this paper, we study the setting where Y is a discrete Euclidean metric (P,F,δ), that is, where P, F are finite subsets of $\mathbb{R}^d$ and δ is the Euclidean distance. A natural attempt to improve the approximation factor in the Euclidean setting is to reduce the coefficients in front of the terms $\delta(y,o)$ and $\delta(y,B)$ in the projection lemma. Unfortunately, this straightforward approach fails: The projection lemma is tight even on the line metric; see Figure 1.

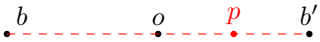


Figure 1 This example shows that the projection lemma is tight even for the 1-dimensional Euclidean space. Let $o = 0$ be the optimum facility located at the origin and serving client $p = 1/2$. Let $b' = 1$ be the facility in B that serves p and let $b = \sigma(o) = -1$ be the facility in B nearest to o. We have $\mathsf{OPT} = 1/2$, which also equals the cost of B. However $\delta(p,\sigma(o)) = 3/2 = 2\times\delta(p,o)+1\times\delta(p,b')$. Combining multiple such examples in orthogonal directions and sharing facility b shows that the approximation ratio of the algorithm of Goyal and Jaiswal [22] approaches 3 in the discrete Euclidean space.

It turns out that slightly enlarging the projection space is already sufficient to bypass this obstacle. More specifically, we project onto the *midpoint closure*

$$\mathsf{cl}(B) = B \cup \left\{ \pi_F\left(\frac{b+b'}{2}\right) : b, b' \in B \right\}, \tag{2}$$

of the bicriteria solution where $\pi_F(p)$ represents the closest facility in F to point p. This step exploits that the metric space is embedded into $\mathbb{R}^d$ (so that the midpoints exist).

While on the algorithmic side a slight modification of the original algorithm is sufficient for the improvement, the analysis requires several new ideas and technically involved ingredients. To prove a strengthened version of the projection lemma (called assignment lemma) we set up a factor-revealing geometric optimization problem in the plane; see (3) in Definition 3.2 below. We call the optimum objective γ_β of this problem *displacement ratio*. Roughly speaking, this ratio corresponds to the maximum ratio between the left-hand and the right-hand side of (1) in Lemma 2.1. However, we project to $\mathsf{cl}(B)$ rather than B and impose some additional minor restrictions. By a careful and technically involved analysis of this optimization problem we can upper bound the displacement ratio in the Euclidean setting by $1-\epsilon_0$ for some universal constant $\epsilon_0 > 0$ as long as two obstructions are avoided. The first obstruction occurs in any configuration similar to the one in Figure 1 above where the bi-criteria solution contains two

facilities b, b' so that o is near to the mid-point of b and b'. However, in such a configuration facility o certifies that $b'' = \pi_F((b + b')/2)$ must be close to o allowing us to assign o to b'' contained in the mid-point closure. The second obstruction arises if p is β-*near*, that is, within a small distance β from o (but there is no facility in B such as b' as in the first obstruction). For β approaching 0, the displacement ratio of β-near points can approach 1 even if when projecting to the mid-point closure of B. To account for β-near points, we therefore cannot resort to the assignment lemma. However, the overall contribution of β-near points to the cost of the projected solution can be shown to be very small. More details of the algorithm and its analysis are provided in Section 3.1. The full proof of the assignment lemma is technically more involved and can be found in the full version [1].

Hardness of Discrete k-Center

Our proof constructs an instance of the discrete k-Center from an instance of Multi-Colored Independent Set problem, which is known to be $W[1]$-hard. In Multi-Colored Independent Set, we are given a k-partite graph G with a k-partition of the vertices $V_1, \ldots, V_k$, and the goal is to determine if there is an independent set that contains precisely one node from each set V_i, $i \in [k]$. The gadget in our construction is a set of nearly equidistant binary code words. Such code words with relative Hamming distance roughly $1/2$ and logarithmic length are known to exist (see Ta-Shma [32]). The high level idea is as follows. We associate each vertex of G with a unique code word of suitable length t. Then, we generate a data point in P for each vertex and edge of G by using code word(s) associated with the corresponding vertices. The construction guarantees the following crucial properties: (i) The Hamming distance between the data points of vertices is roughly t. (ii) The Hamming distance between a data point of vertex $v \in V_i$ and a data point of an edge e is roughly t if e is incident on $V_i \setminus \{v\}$ and is roughly $3t/2$ otherwise. (iii) The Hamming distance between the data points of edges is at least (close to) $3t/2$. Thus, the construction forces us to pick data points of vertices as centers in our solution and guarantees that the optimum cost of the k-Center instance is roughly t if and only if there is an independent set in G. As a result, approximating the cost of the k-Center instance better than a (roughly) $(3/2)^{1/q}$ factor would imply $W[1] = \mathsf{FPT}$. That is because the cost of a k-Center instance is the maximum ℓ_q distance between a data point and its closest selected center, and hence, approximating this cost better than the mentioned factor allows us to distinguish between Yes and No cases of an arbitrary instance of Multi-Colored Independent Set.

Approximation Scheme for Metrics of Sub-Logarithmic Doubling Dimension

Our algorithm comprises two main components, both based on standard techniques from the literature: instance compression and decomposition of the doubling metric into smaller balls. However, it becomes evident that a natural construction based on these standard techniques for Robust (k, z)-Clustering faces serious information-theoretic limitations, as explained below. One natural idea for compressing a Robust (k, z)-Clustering instance is to reduce the number of groups, as each group can be further compressed using a (k, z)-Clustering coreset (such coresets exist [16]). This reduction yields a significantly smaller instance. If we could reduce the number of groups to $m' \ll m$ while approximately preserving the cost for every solution, we could obtain an EPAS as follows. First, apply a (k, z)-Clustering coreset to every group of the compressed instance to obtain another Robust (k, z)-Clustering instance with m' groups, each containing $g(k, \epsilon)$ points, where g is some function that represents the size of (k, z)-Clustering coreset. It is essential to note that this compression

is acceptable for obtaining an EPAS since the coreset of a group approximately preserves the (k, z)-Clustering cost of the group. Next, enumerate all k-partitions of the points within each group to find potential solutions. Finally, return the solution that has the minimum Robust (k, z)-Clustering cost. Unfortunately, because Robust (k, z)-Clustering captures k-Center (and consequently faces a coreset lower bound of $2^{\Omega(d)}$ in Euclidean space of dimension d [9]), the number of new groups must satisfy $m' \geq 2^{\Omega(d)}$. Consequently, the running time of this algorithm is $k^{2^{\Omega(d)}}\mathsf{poly}(n, m)$, which is doubly exponential in d. It is worth noting that this algorithm matches the running time of [2] and does not yield an EPAS for sub-logarithmic dimension.

Furthermore, if we explore an alternative approach and utilize the coreset of k-Center, it is not immediately clear how to extend the coreset of k-Center to reduce the number of groups in an instance of Robust (k, z)-Clustering. This is because, firstly, we would require a mapping between the old groups and the new groups, and secondly, this mapping should ideally approximately preserve the Robust (k, z)-Clustering cost for every solution.

Another potential method for compressing the instance involves reducing the number of points in set P, rather than altering the groups, with the hope of designing an EPAS that can exploit the smaller P (without concern for the number of groups). However, for this approach to succeed, it is essential to establish a bijection between the old and new groups. Yet, it remains uncertain whether such a bijection exists. In typical coreset constructions, each point in the coreset P' of P has a weight that is the sum of the weights of the points in its local neighborhood in P which it is supposed to represent in P'. However, these points in P could potentially belong to different groups, making it challenging to establish the mapping between groups.

The core idea of our approach is to work with an alternative and more general definition of groups that permits a point to participate in different groups with varying weights. In this revised definition, instead of viewing groups as subsets of points, we treat each group as a weight function that assigns non-negative real values to points. This flexibility allows different weights to be assigned to the same point by different groups, which can, in fact, be of practical interest. Utilizing this new definition, we can devise an approach for compressing the points such that each point in the compressed instance can have a weight for group g that represents the sum of the weights of nearby points in g that were filtered out during compression. Essentially, this enables us to approximately preserve the group costs. With this approach and additional technical work that leverages the standard ball decomposition technique for doubling metrics, we derive a coreset for Robust (k, z)-Clustering that can be employed to construct an EPAS for doubling metrics with sub-logarithmic dimension.

▶ Remark 2.2. Due to lack of space, we move some of the proofs to the full version [1]. The proofs of the Theorems and Lemmas with corresponding $(\star)$ marked are provided in the full version [1].

3 High-Dimensional Discrete Euclidean Space

3.1 FPT Approximation Algorithm for Robust (k, z)-Clustering

In this section, we exploit non-trivial properties of the Euclidean metric to prove the following result that breaches the barrier of 3^z-approximation for Robust (k, z)-Clustering in general metrics.

▶ **Theorem 1.1** (High-Dimensional Euclidean Space). *There exists a universal constant $\eta_0 > 0.0006$ such that for any constant positive integer z, there is a factor $3^z(1 - \eta_0)$ FPT approximation algorithm for Robust (k, z)-Clustering in discrete Euclidean space $\mathbb{R}^d$ that runs in time $2^{\mathcal{O}(k \log k)}\mathsf{poly}(m, n, d)$.*

Recall from Section 2 that our approach begins with computing a (κ, λ)-bicriteria solution B to the Robust (k, z)-Clustering instance employing the algorithm proposed by Goyal-Jaiswal [22]. As we argued, it is sufficient to prove the existence of a k-subset of B whose cost is within a constant factor of optimal. The result by Goyal and Jaiswal [22] is based on the following simple projection lemma for general metrics whose proof we state here for the sake of later reference.

▶ **Lemma 2.1** (Projection Lemma). *Let (Y, δ) be a metric space, and $B \subseteq Y$. Then for any set $O \subseteq Y$, there exists an assignment $\sigma\colon O \to B$ such that, for all $o \in O$ and $y \in Y$, we have*

$$\delta(y, \sigma(o)) \leq 2\delta(y, o) + \delta(y, B)\,. \tag{1}$$

Proof. For each $o \in O$, define $\sigma(o)$ as $\pi_B(o)$, the point in B closest in distance to o. Notice that for any $o \in O$, $y \in Y$, we have $\delta(y, \sigma(o)) \leq \delta(y, o) + \delta(o, \sigma(o))$ by triangle inequality. The lemma follows by combining this with $\delta(o, \sigma(o)) = \delta(o, B) \leq \delta(o, \pi_B(y)) \leq \delta(y, o) + \delta(y, B)$. ◀

This lemma itself is tight even in 1-dimensional Euclidean space (as we showed in Figure 1). In order to get around this issue, we make use of the property of our geometric space. Given the instance (P, F, δ) embedded into the Euclidean space and the bicriteria solution B, we project to the mid-point closure $\mathsf{cl}(B)$ as defined in (2).

Notice that $|\mathsf{cl}(B)| = \mathcal{O}\left(|B|^2\right)$. Let O be the optimal solution. For $\beta > 0$ we say that client $p \in P$ is *β-far* (from O w.r.t. B) if $\delta(p, O) \geq \beta \cdot \delta(p, B)$, and we say that client p is *β-near* otherwise. The key of our analysis is the following strengthening of the projection lemma for Euclidean space, which we call assignment lemma.

▶ **Lemma 3.1** (Assignment Lemma) (⋆). Let $\beta_0 = 0.05$ and let $B \subseteq \mathbb{R}^d$. Then, for any $O \subseteq \mathbb{R}^d$, there exists an assignment $\sigma\colon O \to \mathsf{cl}(B)$ such that, for all β_0-far points $p \in \mathbb{R}^d$, we have $\delta(p, \sigma(O)) \leq (1 - \epsilon_0)(2\delta(p, O) + \delta(p, B))$ where $\epsilon_0 > 0.002$.

Proof Sketch. We start with defining the assignment function σ. Take any facility $o \in O$ and let $b = \pi_B(o)$. We assume w.l.o.g. that the instance is rotated so that p, b, and o lie in the plane spanned by the first two coordinates. For the sake of easier notation, we identify p, b, o by points in $\mathbb{R}^2$. Further, by translation and scaling, we assume that o coincides with the origin and that $b = (-1, 0)$. Let $q = (0, 1)$ be the mirror image of b. Let α be a parameter to be fixed (we later set it to 0.6). We define $\sigma(o)$ based on the position of o relative to an α-ball. Specifically, $\sigma(o) = b$ if the α-ball centered at a point q contains no facility from B; otherwise, $\sigma(o)$ is the projection $\pi_{\mathsf{cl}(B)}(o)$ of o onto the mid-point closure of B.

Our goal is to analyze the displacement of a client p under the assignment rule σ. Recall from the proof of Lemma 2.1 that if $\sigma(o)$ is simply the projection onto B, then a client p, when served by facilities o and b' in sets O and B respectively, incurs a cost of at most $2||p - o|| + ||p - b'||$. We wish to show that the assignment cost in our algorithm is strictly smaller than this upper bound (under certain assumptions). We prove this by bounding the ratio of these two quantities.

▶ **Definition 3.2** (Displacement Ratio). *For a given small constant $\beta > 0$, let the* displacement ratio *be defined as*

$$\gamma_\beta = \max_{\substack{p \in \mathbb{R}^d \setminus \mathsf{ball}(o, \beta),\\ b' \in \mathbb{R}^d \setminus \mathsf{ball}(o, 1)}} \left\{ \frac{||p - \sigma(o)||}{2||p - o|| + ||p - b'||} \right\}. \tag{3}$$

Let S be the plane spanned by b, p, and o. After the appropriate rotations and translations we mentioned earlier, S would coincide with the x-y plane. In what follows, we also restrict b' to lie in $\mathbb{R}^2$ as well. We omit the argument why this assumption is without loss of generality from this sketch, and defer it to [1].

To show the lemma, we demonstrate that γ_β can be upper-bounded by $1 - f(\alpha, \beta)$ for some $f(\alpha, \beta) > 0$, where $f(\cdot)$ is a function dependent on α, β and the geometry of O and B. We distinguish two cases. First, suppose that B contains a facility b'' lying inside the α-ball around q. Recall that in this case $\sigma(o) = \pi_{\mathrm{cl}(B)}(o)$. Hence $\sigma(o)$ is no farther from o than the facility $\pi_F((b + b'')/2)$ nearest to the midpoint of b and b''. This allows us to bound the displacement ratio γ_β by $1 - \frac{1-\alpha}{2}$. See Figure 2 for an illustration. Notice that the optimal center o certifies the existence of a point in F nearby the mid-point of b and b''.

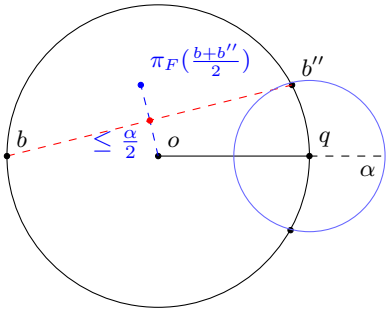


Figure 2 The midpoint of b and b'' is shown by red dot, $||(b + b'')/2 - o|| \leq \frac{\alpha}{2}$ and thus $||\sigma(o) - o|| \leq \alpha$.

In the second case, where the α-ball does not contain a facility from B, we argue that the points o, $\sigma(o) = b$, and b' are far enough from a co-linear position. This allows us to argue that the triangle inequality in the proof of Lemma 2.1 is not tight. Towards this, we divide the space into four regions R_1, R_2, R_3 and R_4 that could contain client p, we assume that p lies in the half plane above the x-axis (The case where p lies below the x-axis is symmetric.). Let q_1 be the intersection point of the surfaces of $\mathsf{ball}(o, 1)$ and $\mathsf{ball}(q, \alpha)$ above the x-axis. Let q_3 be the midpoint of q and q_1, the region H is defined as the area above the lines passing through (q_3, o) and (o, b), we define $R_1 = H \backslash \mathsf{ball}(o, \beta)$. Next, consider $(1 - \omega)$ and $(1 + \omega)$ balls around o, H' is defined as the area below the line passing through (o, q_3) and above the line passing through (o, q), we define $R_2 = (\mathsf{ball}(o, 1 - \omega) \backslash \mathsf{ball}(o, \beta)) \cap H'$, $R_3 = (\mathsf{ball}(o, 1 + \omega) \backslash \mathsf{ball}(o, 1 - \omega)) \cap H'$, and $R_4 = H' \backslash \mathsf{ball}(o, 1 + \omega)$, the regions are indicated in Figure 3. Below, we provide full proof for one of these regions.

Assume that client p lies in region R_1 (see Figure 4). Let b'' be the closest point to p not in the interior of $\mathsf{ball}(o, 1)$, and let p' be the point on the boundary of $\mathsf{ball}(o, \beta)$ that is closet to p. Let p'' be the point where the segment (o, q_3) intersects the boundary of $\mathsf{ball}(o, \beta)$, that is, $p'' = (\beta \cos\theta, \beta \sin\theta)$ where $\theta = \angle q_3 o q_1$. Notice that $\cos\theta = 1 - \frac{\alpha^2}{4}$. First, we assume p is inside $\mathsf{ball}(o, 1 + 2\beta)$ in the region of R_1.

▶ **Observation 3.3.** *For any* $\epsilon_1, \epsilon_2, X, Y \geq 0$ *:*

$$\frac{X - \epsilon_1 + Y}{X + Y} \leq \frac{X - \epsilon_1 + Y + \epsilon_2}{X + Y + \epsilon_2}$$

Consider assigning p via p' to b. We bound the displacement cost as follows:

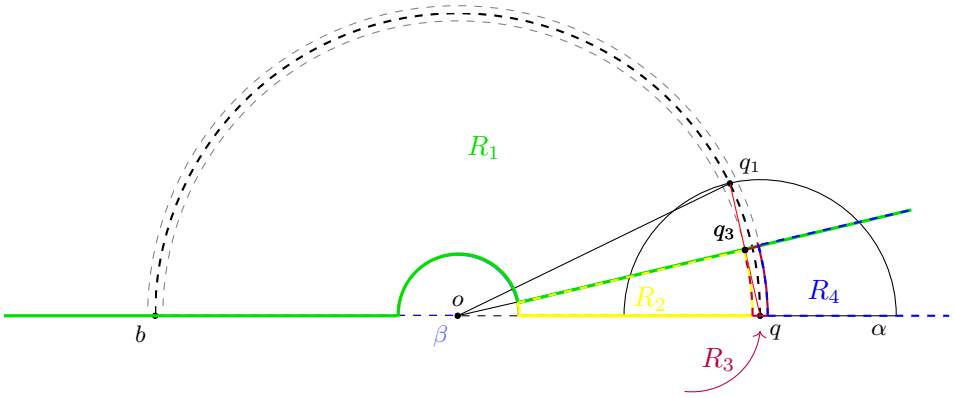


Figure 3 The dashed black circle depicts $\mathsf{ball}(o, 1)$, while the dashed gray circles represent $\mathsf{ball}(o, 1-\omega)$ and $\mathsf{ball}(o, 1+\omega)$. Regions R_1, R_2, R_3, and R_4 are outlined with green, yellow, purple, and blue borders respectively.

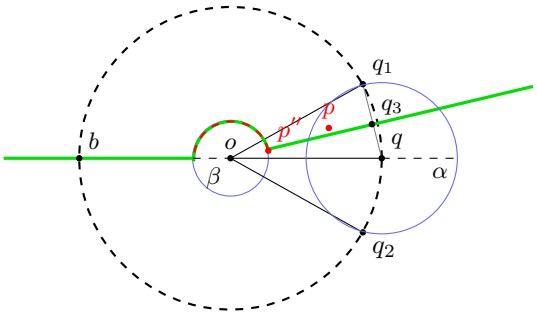


Figure 4 The dashed circle depicts $\mathsf{ball}(o, 1)$, α-ball around q and β-ball around o are shown in blue. The region R_1 is specified by green borders. The red dashed area represents the potential location of point p'.

$$\begin{aligned}
\gamma_\beta = \frac{||\sigma(o) - p||}{2||p-o|| + ||p-b'||} &\leq \frac{||b-p'|| + ||p-p'||}{2||p'-o|| + ||p'-b''|| + ||p-p'||} \\
&\leq \frac{||b-p''|| + ||p-p'||}{2\beta + 1 - \beta + ||p-p'||} \\
&\leq \frac{\sqrt{(\beta\cos(\theta)+1)^2 + (\beta\sin(\theta))^2} + ||p-p'||}{1+\beta+||p-p'||} \\
&= \frac{\sqrt{\beta^2 + 2\beta\cos\theta + 1} + ||p-p'||}{1+\beta+||p-p'||} \\
&= \frac{\sqrt{(1+\beta)^2 - \frac{\beta\alpha^2}{2}} + ||p-p'||}{1+\beta+||p-p'||}
\end{aligned}$$

We assume $||p - p'|| \leq 1 + \beta$, and by observation 3.3, we obtain:

$$\gamma_\beta \leq \frac{(1+\beta)(1+\sqrt{(1-\frac{\beta\alpha^2}{2})})}{2(1+\beta)} \leq \frac{1}{2} + \frac{\sqrt{1^2 - \frac{2\beta\alpha^2}{4} + \frac{\beta^2\alpha^4}{16}}}{2} \leq \frac{1}{2} + \frac{1 - \frac{\beta\alpha^2}{4}}{2}$$
$$= 1 - \frac{4 + \beta\alpha^2}{8}$$

Second, let's assume that the client p is distant from o and positioned within region R_1 outside $\mathsf{ball}(o, 1 + 2\beta)$, we can bound γ_β as follows:

$$\gamma_\beta \leq \frac{1 + ||o - p||}{2||o - p||} \leq \frac{1 + 1 + 2\beta}{2(1 + 2\beta)} = \frac{1 + \beta}{1 + 2\beta} = 1 - \frac{\beta}{1 + 2\beta}$$

Therefore, by examining the position of p in the regions, we establish that γ_β is upper-bounded by $1 - f(\alpha, \beta)$. Consequently, Lemma 3.1 is substantiated by showing the existence of an $\alpha_0 \leq 0.6$ and a sufficiently small $\beta_0 \leq 0.05$ such that $\gamma_{\beta_0} \leq 1 - f(\alpha_0, \beta_0) = 1 - \epsilon_0 \leq 0.9978$. The proofs for the other regions as well as the full details of the rest of the argument are provided in the full version [1]. ◀

In the proof of Theorem 1.1, we show that this new assignment property is enough to derive an improved FPT approximation for Robust (k, z)-Clustering in Euclidean space. Since the assignment σ maps every facility in O uniquely to a facility in $\mathsf{cl}(B)$, this implies that $\sigma(O)$ is a feasible solution of cost at most $(3^z \cdot (1 - \eta_0))\mathsf{OPT}$. This certifies the existence of a feasible solution being a subset of $\mathsf{cl}(B)$ with the desired approximation factor. Hence, we can find such a solution in FPT time by enumeration. The complete proof of Theorem 1.1 is provided in the full version [1].

3.2 Hardness of Discrete k-Center

For this section, we use the following explicit construction of the so-called *η-balanced error-correcting codes* from a recent result of Ta-Shma [32] which we rephrase for our purposes as follows:

▶ **Theorem 3.1.** *Let $\eta \in (0, 1/2)$ be a positive constant. Then there is an algorithm that computes, for any given number $s \in \mathbb{N}$, an s-element set $B \subseteq \{0,1\}^t$ of binary vectors of dimension $t = \mathcal{O}(\log s/\eta^{2+o(1)})$ such that for any $b \in B$, its Hamming weight $||b||_1$ and for any $b' \in B \setminus \{b\}$, the Hamming distance $||b - b'||_1$ both lie in the interval $[(1/2-\eta)t, (1/2+\eta)t]$. The running time of the algorithm is $\mathcal{O}(st)$.*

Proof. Ta-Shma [32] gives an explicit construction of a $t \times \lceil \log_2 s \rceil$ binary matrix generating a linear, binary, error-correcting code of message length $\lceil \log_2 s \rceil$, block length $t = \mathcal{O}(\log s/\eta^{2+o(1)})$, and pairwise Hamming distance between $(1/2 - \eta)t$ and $(1/2 + \eta)t$. Since the code is linear, it contains the zero code word. Hence each code word has Hamming weight in $[(1/2 - \eta)t, (1/2 + \eta)t]$. The time for constructing the matrix is polynomial in $\log s$ and t. Using the generating matrix, at least s many non-zero code words can be enumerated in time $\mathcal{O}(st)$, which dominates the time for computing the matrix. ◀

We leverage balanced error correcting codes as gadget in our hardness proof for discrete k-Center. For any binary vector $b \in \{0,1\}^t$, we denote by $\bar{b}$ the binary vector obtained by flipping each coordinate in b.

▶ **Theorem 1.2** (Hardness in Discrete Euclidean Space). *For any constant positive integer q and any positive constant $\eta > 0$, there exists a function $d(k,n) = O(k \log n)$ such that there is no factor-$(3/2-\eta)^{1/q}$ FPT approximation algorithm for the discrete k-Center problem in $\mathbb{R}^{d(k,n)}$ under the ℓ_q metric unless W[1]= FPT. Moreover, for the ℓ_2 metric this hardness holds even for some dimension $O(\log n)$, that is, independently of k.*

Proof. We show a reduction from Multi-Colored Independent Set, which is known to be $W[1]$-hard [17]. The input is a k-partite graph $G = (V, E)$ with k-partition $V_1, \ldots, V_k$. The question is if there is an independent set that is *multi-colored*, that is, it has precisely one node from each set V_i, $i \in [k]$. W.l.o.g. we assume that each V_i contains at least one node that is adjacent to all nodes $V \setminus V_i$. Adding such nodes, we can additionally assume that $|V_i| = n/k$ for each $i \in [k]$ where $n = |V|$.

Fix some constant $\eta \in (0, 1/2)$. Using Theorem 3.1, we construct a set $B \subseteq \{0,1\}^t$ of n nearly equidistant code words of dimension $t = \mathcal{O}(\log n/\eta^{2+o(1)})$. We map each node $u \in V$ uniquely to some non-zero code word $b(u) \in B$. We construct a k-Center instance in $\mathbb{R}^{k \cdot t}$ as follows. We subdivide the coordinates of each point in $\mathbb{R}^{k \cdot t}$ into k *blocks* each containing t consecutive coordinates. In our set P of data points, we introduce for each node $v_i \in V_i$, $i \in [k]$, the point $p(v_i) \in P$ in which the ith block equals $b(v_i)$ and all other coordinates are zero. For each edge $(v_i, v_j) \in E$, $v_i \in V_i$, $v_j \in V_j$ for distinct $i, j \in [k]$ we create a point $p(v_i, v_j) \in P$ in which the ith block equals $\overline{b(v_i)}$, the jth block equals $\overline{b(v_j)}$, and all other coordinates are zero. No further points are added to P. We set the number of centers to be k completing the construction of the k-Center instance.

Let $i \in [k]$ and $v_i, v_i' \in V_i$ be distinct vertices. We have that $||p(v_i) - p(v_i')||_q^q \leq ||b(v_i) - b(v_i')||_1 \leq (1/2 + \eta)t$ by Theorem 3.1. Let $v_j \in V_j$, $j \in [k]$ such that $(v_i, v_j) \in E$. By Theorem 3.1, we have that

$$\begin{aligned} ||p(v_i') - p(v_i, v_j)||_q^q &\leq ||b(v_i') - \overline{b(v_i)}||_1 + ||\overline{b(v_j)}||_1 \\ &\leq (t - ||b(v_i') - b(v_i)||_1) + (t - (1/2 - \eta)t) \\ &\leq (t - (1/2 - \eta)t) + (1/2 + \eta)t \\ &\leq (1 + 2\eta)t\,. \end{aligned}$$

Hence if there is a multi-colored independent set I for G then $X = \{\, p(u) \mid u \in I \,\}$ is a k-element set such that $\delta(p, X)^q \leq (1 + 2\eta)t$ for any $p \in P$ under the ℓ_q metric, which gives an upper bound of $(1 + 2\eta)t$ on the k-Center objective in the completeness case.

For analyzing the soundness case, assume that there is no multi-colored independent set for G. Consider an arbitrary k-element set $X \subseteq V$. We say that $x \in X$ *covers* $p \in P$ if $\delta(p, x)^q < (3/2 - 3\eta)t$. We claim that there is some $p \in P$ not covered by any center in X. The correctness of this claim implies that any parameterized approximation algorithm with approximation ratio strictly better than $((3/2 - 3\eta)/(1 + 2\eta))^{1/q}$ implies that $W[1] = \mathsf{FPT}$ and thus the theorem.

In order to prove this claim, we assume for the sake of contradiction, that all $p \in P$ are covered by some center in X. First, we argue that w.l.o.g. X contains no point of the form $p(v_i, v_j)$ where $(v_i, v_j) \in E$. In fact, for any $g \notin \{i, j\}$, we have that

$$\begin{aligned} ||p(v_g') - p(v_i, v_j)||_q^q &\geq ||b(v_g')||_1 + ||\overline{b(v_i)}||_1 + ||\overline{b(v_j)}||_1 \\ &\geq (1/2 - \eta)t + 2(t - (1/2 + \eta)t) \\ &= (3/2 - 3\eta)t\,. \end{aligned} \tag{4}$$

Hence $p(v_i, v_j)$ can cover $p(v_g')$ only if $g = i$ or $g = j$. Similarly, $p(v_i, v_j)$ can cover $p(v_g', v_h')$ only if $i = g$ and $j = h$. But then these points would be covered by $p(v_i)$ as well and hence we could replace $p(v_i, v_j)$ with $p(v_i)$. We therefore assume that X contains only points of the form $p(v_i)$.

We claim that X is multi-colored. Otherwise, there would be some V_i that contains no point from X. By our initial assumption, V_i contains some point v_i that is adjacent to all points $V \setminus V_i$. Assuming $k \geq 3$ there exists at least one V_j, $j \neq i$ that contains at most one node from X. If V_j intersects X then let $v_j \in V_j \cap X$, and otherwise let v_j be an arbitrary node in V_j. By our assumption $(v_i, v_j) \in E$. If $v_j \in X$ then

$$\begin{aligned} ||p(v_j, v_i) - p(v_j)||_q^q &\geq ||\overline{b(v_j)} - b(v_j)||_1 + ||\overline{b(v_i)}||_1 \\ &\geq t + (t - (1/2 + \eta)t) \\ &= (3/2 - \eta)t \end{aligned} \tag{5}$$

as the jth block of $p(v_j)$ equals $b(v_j)$ and the ith block of $p(v_j, v_i)$ equals $\overline{b(v_j)}$. If $v_j \notin X$ then for any it$v_h \in X$ we have $h \notin \{i, j\}$. Thus $||p(v_i, v_j) - p(v_h)||_q^q \geq (3/2 - 3\eta)t$, which follows as in (4). Hence $p(v_i, v_j)$ would not be covered showing that X is multi-colored. Since X is multi-colored it can not be an independent set. Hence there exists some edge (v_i, v_j) such that $v_i, v_j \in X$ but then $||p(v_i) - p(v_i, v_j)||_q^q \geq (3/2 - \eta)t$, $||p(v_j) - p(v_i, v_j)||_q^q \geq (3/2 - \eta)t$, and $||p(v_h) - p(v_i, v_j)||_q^q \geq (3/2 - 3\eta)t$ for any $v_h \in X$, $h \notin \{i, j\}$, which follows as in (5) and (4), respectively. Hence $\delta(p(v_i, v_j), X) \geq (3/2 - 3\eta)t$, implies that $p(v_i, v_j)$ is not covered.

We complete the proof by noting that the dimension of the instance can be reduced to $O(\log n)$ for Euclidean metrics by using the Johnson-Lindenstrauss transform with sufficiently small (constant) error parameter. ◀

4 EPAS for Metrics of Sub-Logarithmic Doubling Dimension

In this section, we show an EPAS for Robust (k, z)-Clustering in metrics of sub-logarithmic doubling dimension. This result complements the hardness result of Section 3 (Theorem 1.2). Towards our goal, we prove the following result.

▶ **Theorem 1.3** (EPAS for Doubling Metric of Sub-Logarithmic Dimension). *There is an algorithm that computes $(1 + \epsilon)$-approximate solution, for every $\epsilon > 0$, for Robust (k, z)-Clustering in the metric of doubling dimension d in time $f(k, d, \epsilon, z)\mathsf{poly}(m, n)$, where $f(k, d, \epsilon, z) = \left(\left(\frac{2^z}{\epsilon}\right)^d k \log k\right)^{\mathcal{O}(k)}$.*

Note that the above algorithm runs in FPT time for $d = o(\log n)$. We also remark that the above result can be extended to the continuous $\mathbb{R}^d$. Throughout this section, we assume that the weight aspect ratio $\frac{\max_{p \in P} w(p)}{\min_{p' \in P} w(p')}$ and the distance aspect ratio $\frac{\max_{p,p' \in P} \delta(p,p')}{\min_{p \neq p' \in P} \delta(p,p')}$ are bounded by $\mathsf{poly}(n)$, some polynomial in n. For $p \in P$ and any number $r \geq 0$, denote by $\mathsf{ball}(p, r)$ to be the closed ball centered at p of radius r. We prove the theorem in two steps: first, in Section 4.1 we show an algorithm to obtain a coreset for the problem, and then, in Section 4.2 we show how to use this coreset to get the algorithm of Theorem 1.3.

4.1 Coreset for Robust (k, z)-Clustering

The key idea for constructing coresets for Robust (k, z)-Clustering crucially relies on the following alternate but equivalent definition of the problem. In this definition, we are given $\mathcal{I} = (F, P \subset \mathcal{M}, \mathcal{W})$, where either $F = \mathcal{M}$ or $F \subseteq \mathcal{M}$, where $\mathcal{M}$ is doubling metric of dimension d, defined by the metric function δ. A *group* is a weight vector $\mathbf{w} \in \mathcal{W}$ such that $\mathbf{w} : P \to \mathbb{R}_{\geq 0}$. Given $X \subseteq F$, the distance vector $\boldsymbol{\delta}_P(X)$ is defined as $\boldsymbol{\delta}_P(X)[p] = \delta(p, X)^z$, for each $p \in P$. The cost of X for a group $\mathbf{w} \in \mathcal{W}$ is defined as $c(\mathbf{w}, X) = \mathbf{w} \cdot \boldsymbol{\delta}_P(X)$. For a Robust (k, z)-Clustering instance $\mathcal{I} = (F, P, \mathcal{W})$, the cost of X is defined as $\mathsf{cost}(\mathcal{I}, X) = \max_{\mathbf{w} \in \mathcal{W}} \mathsf{cost}(\mathbf{w}, X)$. The cost of the instance $\mathcal{I} = (F, P, \mathcal{W})$ is

$$\mathsf{OPT}(\mathcal{I}) = \min_{X \subseteq F, |X|=k} \max_{\mathbf{w} \in \mathcal{W}} \mathsf{cost}(\mathbf{w}, X)$$

Whenever the instance $\mathcal{I}$ is clear from context, we will just write OPT. Notice that, in the original Robust (k,z)-Clustering, a group is given by $S \subseteq P$, and this can be captured by weight vector $\mathbf{w}[p] = 0$ for $p \notin S$ and $w(p)$ otherwise. We prove the following coreset exists for Robust (k,z)-Clustering.

▶ **Theorem 4.1** (Coreset for Robust (k,z)-Clustering) $(\star)$**.** Given an instance $\mathcal{I} = (F, P, \mathcal{W})$ of Robust (k,z)-Clustering in doubling metric of dimension d and $0 < \epsilon \leq 1$, there is an algorithm that, in time $\left(\frac{2^z}{\epsilon}\right)^{\mathcal{O}(d)} \mathsf{poly}(n,m)$, computes another instance $\mathcal{I}' = (F, P', \mathcal{W}')$ of Robust (k,z)-Clustering with $P' \subseteq P : |P'| = \left(\frac{2^z}{\epsilon}\right)^{\mathcal{O}(d)} kz \log n$ such that for any $X \subseteq F$ with $|X| = k$,

$$(1-\epsilon)\mathsf{cost}(\mathcal{I}, X) \leq \mathsf{cost}(\mathcal{I}', X) \leq (1+\epsilon)\mathsf{cost}(\mathcal{I}, X).$$

We remark that the above theorem yields a coreset of clients, and not of groups, and hence, the total size of coreset is comparable to the original instance. However, we will show later that such coreset is sufficient to get a parameterized approximation scheme with parameters k and d. We would also like to point out that the exponential dependency on d on the point set size of the coreset is inevitable since Robust (k,z)-Clustering captures k-Center, for which such a lower bound is known [9, 7]. To see that our notion of coreset for Robust (k,z)-Clustering coincides with the regular notion of coreset for k-Center, note that in this setting each group contains a single distinct point.

In the next section, we describe the algorithm of Theorem 4.1. Due to space constraints, we defer the analysis of our algorithm to the full version [1].

The Algorithm

Our algorithm is inspired by the grid construction approach of [24] that yields coresets for k-Median and k-Means. Given an instance $\mathcal{I} = (F, P, \mathcal{W})$ of Robust (k,z)-Clustering, the first step is to start with an (α, β)-bicriteria solution $B = \{b_i\}_{i \in [\beta k]}$ that opens at most βk facilities with the guarantee that $\mathsf{cost}(\mathcal{I}, B) \leq \alpha \cdot \mathsf{OPT}$, for some constants $\alpha, \beta \geq 1$. Let $R = \sqrt[z]{\frac{\mathsf{cost}(\mathcal{I},B)}{\alpha\tau}}$, where $\tau := \max_{\mathbf{w} \in \mathcal{W}} ||\mathbf{w}||_1$. Let $\Delta = \frac{\max_{p \in P, \mathbf{w} \in \mathcal{W}} \mathbf{w}[p]}{\min_{p \in P, \mathbf{w} \in \mathcal{W}} \mathbf{w}[p]}$ be the weight aspect ratio of $\mathcal{I}$. Then, for each $b_i \in B$, consider the balls $\mathcal{B}_i^j := \mathsf{ball}(b_i, 2^j R)$, for $j \in \{0, \cdots, \lceil 2\log(\alpha n \Delta)\rceil\}$. Note that, for $\mathbf{w} \in \mathcal{W}$ and $p \in P$ with $\mathbf{w}[p] > 0$, it holds that $\delta(p, B) \leq R\sqrt[z]{\alpha n \tau}$, since $\delta(p,B) \leq \sqrt[z]{\frac{\mathsf{cost}(\mathcal{I},B)}{\mathbf{w}[p]}} \leq \sqrt[z]{\frac{\alpha\tau}{\mathbf{w}[p]}} R \leq R\sqrt[z]{\alpha n \Delta}$. Hence, we have that every point $p \in P$ is contained in some ball $\mathcal{B}_i^j$. For $b_i \in B$, let $\mathcal{Q}_i^j = \mathcal{B}_i^j - \mathcal{B}_i^{j-1}$, for $j = \{1, \cdots, \lceil 2\log(\alpha\Delta)\rceil\}$, be the ring between $\mathcal{B}_i^j$ and $\mathcal{B}_i^{j-1}$, with $\mathcal{Q}_i^0 = \mathcal{B}_i^0$. Decompose every ball $\mathcal{B}_i^j$ into smaller balls each of radius $\frac{\epsilon}{40\alpha} R 2^j$ using the fact that the metric is a doubling metric. These balls can intersect, so we assign every point $p \in P$ to exactly one ball (for example, by associating p to the smallest ball containing p, breaking ties arbitrarily). For every ball $\mathcal{B}_i^j$ and every smaller ball t of $\mathcal{B}_i^j$ with $|t \cap \mathcal{Q}_i^j| \neq \emptyset$, pick an arbitrary point $p' \in t \cap \mathcal{Q}_i^j$ as the *representative* of (the points in) $t \cap \mathcal{Q}_i^j$, and add p' to the coreset P' with group weight vectors as follows. Corresponding to every group vector $\mathbf{w} \in \mathcal{W}$, create a new group vector $\mathbf{w}' \in \mathcal{W}'$. Then,$\mathbf{w}'[p'] := \sum_{p \in t \cap Q_i^j} \mathbf{w}(p)$. Intuitively, $\mathbf{w}'[p']$ captures the total weight of points of $\mathbf{w}$ in $t \cap \mathcal{Q}_i^j$. This concludes the coreset construction. For detailed pseudocode of the algorithm, please refer to the full version of the paper [1].

The high-level idea above is to decompose each ball $\mathcal{B}_i^j$ into smaller balls and pick a distinct point as the representative of points in the non-empty decomposed ball. Additionally, such representative p' participates in the group $\mathbf{w}'$ with weight which is sum of the weights of points in $\mathbf{w}$ that are represented by p'. However, we want to decompose the ball $\mathcal{B}_i^j$ into smaller balls in a way that the total number of balls remains the same, irrespective of the radius of the ball. This is necessary as for higher values of j, this number would depend on n, if we are not careful. While this does not seem to help much, as the radius of the decomposed balls is much larger for higher values j, it actually does the trick: since the points in these balls are far from b_i, and hence their connection cost to b_i is also large. This allows us to represent the radii of larger balls in terms of the connection cost of its points to B, thus bounding the error in terms of the cost of B, which in turn is bounded by $\alpha\mathsf{OPT}$, which gives us the desired guarantee.

4.2 EPAS for Robust (k, z)-Clustering

In this section, we show how to use the coreset obtained from Theorem 4.1 to get a $(1+\epsilon)$-approximate solution to the Robust (k, z)-Clustering problem and provide an EPAS with respect to k and d, when $|P|$ is small. By scaling the distances in the instance of Robust (k, z)-Clustering, we assume that the distances are between 1 and Δ', for some number Δ'. Our algorithm (see Algorithm 1) uses the leader guessing idea of [12]. In the leader guessing approach, we guess the leader of every partition of a fixed optimal solution, where the leader of a partition is a closest point (client) in P to the corresponding optimal center. However, each point can participate in multiple groups, resulting in the total number of points being dependent on the number of groups, $|\mathcal{W}|$. In the full version [1], we show that guessing the leaders from P without considering the groups in $\mathcal{W}$ is, in fact, sufficient. Further, to get a $(1+\epsilon)$-approximate solution, we use a standard ball decomposition lemma (for e.g., see the full version [1]).

▶ **Theorem 4.2.** *For any $0 < \epsilon \leq 1$, Algorithm 1, on input $\mathcal{I} = (F, P, \mathcal{W})$, computes $X \subseteq F : |X| \leq k$ such that $\mathsf{cost}(\mathcal{I}, X) \leq (1+\epsilon)\mathsf{OPT}(\mathcal{I})$ in time $\left((\frac{z}{\epsilon})^d \log n\right)^{\mathcal{O}(k)} |P|^k \mathsf{poly}(n, m)$.*

We conclude this section by proving the main claim of this section (Theorem 1.3) by using the results of Theorem 4.1 and Theorem 4.2 as follows.

Proof of Theorem 1.3. Given an instance $\mathcal{I} = (F, P, \mathcal{W})$ of Robust (k, z)-Clustering, and the accuracy parameter $\epsilon > 0$, we invoke Theorem 4.1 on $\mathcal{I}$ with parameter $\epsilon/10$ to obtain an coreset $(P', \mathcal{W}')$ such that $P' \subseteq P : |P'| = \left(\frac{2^z}{\epsilon}\right)^{\mathcal{O}(d)} kz \log n$. Let $\mathcal{I}' = (F, P', \mathcal{W}')$ be the resulting instance. Then, we invoke Theorem 4.2 on $\mathcal{I}'$ with parameter $\epsilon/10$ to obtain $X \subseteq F : |X| \leq k$ such that $\mathsf{cost}(\mathcal{I}', X) \leq (1+\epsilon/10)\mathsf{OPT}(\mathcal{I}')$.

First, we analyze the overall running time. With $|P'| = \left(\frac{2^z}{\epsilon}\right)^{\mathcal{O}(d)} kz \log n$, Theorem 4.2 runs in time $\left(\left(\frac{2^z}{\epsilon}\right)^d kz \log n\right)^{\mathcal{O}(k)} \mathsf{poly}(n, m)$, leading to $\left(\left(\frac{2^z}{\epsilon}\right)^d zk \log k\right)^{\mathcal{O}(k)} \mathsf{poly}(n, m)$ as the overall running time as desired. For correctness, consider

$$\begin{aligned}
\mathsf{cost}(\mathcal{I}, X) &\leq (1+\epsilon/10)\mathsf{cost}(\mathcal{I}', X) && \text{by the coreset property} \\
&\leq (1+\epsilon/10)^2\mathsf{OPT}(\mathcal{I}') && \text{by Algorithm 1} \\
&\leq (1+\epsilon/10)^3\mathsf{OPT}(\mathcal{I}) && \text{by the coreset property} \\
&\leq (1+\epsilon)\mathsf{OPT}(\mathcal{I}). &&
\end{aligned}$$

◀

Algorithm 1 $(1+\epsilon)$-approximation algorithm for Robust (k,z)-Clustering.

Data: Instance $\mathcal{I} = (F, P, \mathcal{W})$ of Robust (k,z)-Clustering
Result: $(1+\epsilon)$-approximate solution $X \subseteq F$

Let $X \leftarrow \emptyset$;
forall *k-tuples* $(\ell_1, \cdots, \ell_k)$ *of* P **do**
  forall *k-tuples* $(\lambda_1, \cdots, \lambda_k)$ *radii of* $(\ell_1, \cdots, \ell_k)$ *that are power of* $(1 + \epsilon/10z)$ **do**
    for $i \in [k]$ **do**
      $\mathcal{B}_i \leftarrow \{\frac{\epsilon}{20z}$-ball decomposition of $\mathsf{ball}(\ell_i, \lambda_i)\}$;
    end
    $T_i \leftarrow \{f \in F \mid f$ is an arbitrary facility in ball $b \in \mathcal{B}_i\}$ [a] ;
    forall *k-tuples* $(t_1, \cdots, t_k)$ *of* $T_1 \times \cdots \times T_k$ **do**
      if $\mathit{cost}(\mathcal{I}, \{t_1, \cdots, t_k\}) < \mathit{cost}(\mathcal{I}, X)$ **then**
        $X \leftarrow \{t_1, \cdots, t_k\}$
      end
    end
  end
end
return X

[a] If $F = \mathbb{R}^d$ then $T_i \leftarrow \{x_b \in F \mid x_b$ is the center of ball $b \in \mathcal{B}_i\}$

References

1. Fateme Abbasi, Sandip Banerjee, Jarosław Byrka, Parinya Chalermsook, Ameet Gadekar, Kamyar Khodamoradi, Dániel Marx, Roohani Sharma, and Joachim Spoerhase. Parameterized approximation for robust clustering in discrete geometric spaces, 2023. arXiv:2305.07316.
2. Fateme Abbasi, Sandip Banerjee, Jarosław Byrka, Parinya Chalermsook, Ameet Gadekar, Kamyar Khodamoradi, Dániel Marx, Roohani Sharma, and Joachim Spoerhase. Parameterized approximation schemes for clustering with general norm objectives. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1377–1399, 2023. doi:10.1109/FOCS57990.2023.00085.
3. Mohsen Abbasi, Aditya Bhaskara, and Suresh Venkatasubramanian. Fair clustering via equitable group representations. In *Proc. ACM Conference on Fairness, Accountability, and Transparency (FAccT '21)*, pages 504–514, 2021.
4. Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. In *Proc. 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS'17)*, pages 61–72, 2017.
5. Barbara M. Anthony, Vineet Goyal, Anupam Gupta, and Viswanath Nagarajan. A plant location guide for the unsure: Approximation algorithms for min-max location problems. *Math. Oper. Res.*, 35(1):79–101, 2010. doi:10.1287/moor.1090.0428.
6. Mihai Badŏiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proc. 34th Annual ACM Symposium on Theory of Computing (STOC'04)*, pages 250–257, 2002.
7. Daniel Baker, Vladimir Braverman, Lingxiao Huang, Shaofeng H.-C. Jiang, Robert Krauthgamer, and Xuan Wu. Coresets for clustering in graphs of bounded treewidth. In *Proc. 37th International Conference on Machine Learning (ICML'20)*, volume 119, pages 569–579, 2020.
8. Sayan Bhattacharya, Parinya Chalermsook, Kurt Mehlhorn, and Adrian Neumann. New approximability results for the robust k-median problem. In *Proc. Scandinavian Workshop on Algorithm Theory (SWAT'14)*, pages 50–61, 2014.
9. Vladimir Braverman, Shaofeng H-C Jiang, Robert Krauthgamer, and Xuan Wu. Coresets for ordered weighted clustering. In *Proc. International Conference on Machine Learning (ICML'19)*, pages 744–753, 2019.
10. T.W. Byrka, J.and Pensyl, B. Rybicki, A. Srinivasan, and K. Trinh. An improved approximation algorithm for k-median and positive correlation in budgeted optimization. *ACM Trans. Algorithms*, 13(2)(23):1–31, 2013.

11 Vincent Cohen-Addad, Hossein Esfandiari, Vahab S. Mirrokni, and Shyam Narayanan. Improved approximations for Euclidean k-means and k-median, via nested quasi-independent sets. In *Proc. 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC'22)*, pages 1621–1628, 2022.

12 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight FPT Approximations for k-Median and k-Means. In *Proc. 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132, pages 42:1–42:14, 2019.

13 Vincent Cohen-Addad and CS Karthik. Inapproximability of clustering in lp metrics. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 519–539. IEEE, 2019.

14 Vincent Cohen-Addad, CS Karthik, and Euiwoong Lee. On approximability of clustering problems without candidate centers. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2635–2648. SIAM, 2021.

15 Vincent Cohen-Addad and Euiwoong Lee. Johnson coverage hypothesis: Inapproximability of k-means and k-median in lp-metrics. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1493–1530. SIAM, 2022.

16 Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. A new coreset framework for clustering. In Samir Khuller and Virginia Vassilevska Williams, editors, *Proc. 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC'21)*, pages 169–182, 2021.

17 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

18 Petros Drineas, Alan M. Frieze, Ravi Kannan, Santosh S. Vempala, and V. Vinay. Clustering large graphs via the singular value decomposition. *Mach. Learn.*, 56(1-3):9–33, 2004.

19 Tomás Feder and Daniel H. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th Annual ACM Symposium on Theory of Computing (STOC'88)*, pages 434–444, 1988.

20 Mehrdad Ghadiri, Samira Samadi, and Santosh Vempala. Socially fair k-means clustering. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 438–448, 2021.

21 Mehrdad Ghadiri, Mohit Singh, and Santosh S Vempala. Constant-factor approximation algorithms for socially fair k-clustering. *arXiv preprint arXiv:2206.11210*, 2022.

22 Dishant Goyal and Ragesh Jaiswal. Tight fpt approximation for socially fair clustering. *Information Processing Letters*, 182:106383, 2023. doi:10.1016/j.ipl.2023.106383.

23 Fabrizio Grandoni, Rafail Ostrovsky, Yuval Rabani, Leonard J. Schulman, and Rakesh Venkat. A refined approximation for euclidean k-means. *Inf. Process. Lett.*, 176:106251, 2022. doi:10.1016/j.ipl.2022.106251.

24 Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proc. 36th Annual ACM Symposium on Theory of Computing (STOC'04)*, page 291–300, 2004.

25 D.S. Hochbaum and D. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operation Research*, 10(2):180–184, 1985.

26 K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001.

27 Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. A local search approximation algorithm for k-means clustering. *Computational Geometry*, 28(2-3):89–112, 2004.

28 Amit Kumar, Yogish Sabharwal, and Sandeep Sen. Linear-time approximation schemes for clustering problems in any dimensions. *Journal of the ACM (JACM)*, 57(2):1–32, 2010.

29 Yury Makarychev and Ali Vakilian. Approximation algorithms for socially fair clustering. In Mikhail Belkin and Samory Kpotufe, editors, *Proceedings of Thirty Fourth Conference on Learning Theory*, volume 134 of *Proceedings of Machine Learning Research*, pages 3246–3264.

PMLR, 15–19 August 2021. URL: https://proceedings.mlr.press/v134/makarychev21a.html.

30 Viswanath Nagarajan, Baruch Schieber, and Hadas Shachnai. The Euclidean k-supplier problem. *Math. Oper. Res.*, 45(1):1–14, 2020.

31 Christian Sohler and David P. Woodruff. Strong coresets for k-median and subspace approximation: Goodbye dimension. In *Proc. 59th IEEE Annual Symposium on Foundations of Computer Science (FOCS'18)*, pages 802–813, 2018.

32 Amnon Ta-Shma. Explicit, almost optimal, ϵ-balanced codes. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proc. 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC'17)*, pages 238–251. ACM, 2017. doi:10.1145/3055399.3055408.

Finer-Grained Reductions in Fine-Grained Hardness of Approximation

Elie Abboud
Department of Computer Science, University of Haifa, Israel

Noga Ron-Zewi
Department of Computer Science, University of Haifa, Israel

Abstract

We investigate the relation between δ and ϵ required for obtaining a $(1+\delta)$-approximation in time $N^{2-\epsilon}$ for closest pair problems under various distance metrics, and for other related problems in fine-grained complexity.

Specifically, our main result shows that if it is impossible to (exactly) solve the (bichromatic) inner product (IP) problem for vectors of dimension $c \log N$ in time $N^{2-\epsilon}$, then there is no $(1+\delta)$-approximation algorithm for (bichromatic) Euclidean Closest Pair running in time $N^{2-2\epsilon}$, where $\delta \approx (\epsilon/c)^2$ (where $\approx$ hides polylog factors). This improves on the prior result due to Chen and Williams (SODA 2019) which gave a smaller polynomial dependence of δ on ϵ, on the order of $\delta \approx (\epsilon/c)^6$. Our result implies in turn that no $(1+\delta)$-approximation algorithm exists for Euclidean closest pair for $\delta \approx \epsilon^4$, unless an algorithmic improvement for IP is obtained. This in turn is very close to the approximation guarantee of $\delta \approx \epsilon^3$ for Euclidean closest pair, given by the best known algorithm of Almam, Chan, and Williams (FOCS 2016). By known reductions, a similar result follows for a host of other related problems in fine-grained hardness of approximation.

Our reduction combines the hardness of approximation framework of Chen and Williams, together with an MA communication protocol for IP over a small alphabet, that is inspired by the MA protocol of Chen (Theory of Computing, 2020).

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Fine-grained complexity, conditional lower bound, fine-grained reduction, Approximation algorithms, Analysis of algorithms, Computational geometry, Computational and structural complexity theory

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.7

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2311.00798

Funding *Elie Abboud*: Research supported in part by ISF grant 735/20, and by the European Union (ERC, ECCC, 101076663). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.
Noga Ron-Zewi: Research supported in part by ISF grant 735/20, and by the European Union (ERC, ECCC, 101076663). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

1 Introduction

Traditionally, the approach to determine whether a computational problem is tractable was to find out whether it has a polynomial-time algorithm. Finding such an algorithm implies that the problem is in P, and thus it was considered efficiently computable. Otherwise, if one is interested in proving that the problem is intractable, we usually lack the tools to

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 7; pp. 7:1–7:17


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

prove lower bounds; instead one relies on hardness assumptions which allow us to prove conditional lower-bounds. In the classical theory of NP-hardness, the hardness assumption is that P $\neq$ NP, which is known to imply that no polynomial-time algorithm exists for many central computational problems.

In fine-grained complexity, one is interested in pinning down the *precise* complexity of *tractable* computational problems. In particular, a central objective in fine-grained complexity is to determine the exact exponent in the time complexity of problems already known to be in P. More concretely, given a problem with input length n known to be solvable in $t(n)$-time, is it possible to solve the problem in time $t(n)^{1-\epsilon}$ for some $\epsilon > 0$? This is motivated by the fact that despite rigorous study of many central computational problems in P, we have failed to improve on the running time of their best-known algorithms (see for example the survey [21] for a list of such problems). This motivates the question of whether there is an inherent difficulty in the problem that prevents us from finding faster algorithms.

Once more, we typically lack the tools to prove lower bounds, and we thus instead rely on hardness assumptions to obtain conditional lower bounds for problems in P. One popular such conjecture has been the *Strong Exponential Time Hypothesis* (SETH), which postulates that for any $\epsilon > 0$, there exists an integer $k = k(\epsilon)$ so that it is impossible to solve k-SAT on n variables in time $2^{(1-\epsilon)n}$ [13].

Another popular conjecture is the *Orthogonal Vector Conjecture* (OVC) which in the low-dimensional regime posits that for any $\epsilon > 0$, there exists a $c_{\text{ov}} = c_{\text{ov}}(\epsilon)$ such that given a pair of sets $A, B \subseteq \{0,1\}^d$ of cardinality N each and of dimension $d = c_{\text{ov}} \cdot \log N$, it is impossible to determine whether there exists a pair $(a, b) \in A \times B$ satisfying that $\langle a, b\rangle = 0$ in $N^{2-\epsilon}$ time [11]. It is known that SETH implies OVC [20], and so OVC is at least as plausible as SETH. In terms of algorithms, it is known how to solve the OV problem in time $N^{2-\epsilon}$ with $c = \exp(1/\epsilon)$ [3, 7], which implies that $c_{\text{ov}} \geq \exp(1/\epsilon)$.

A related assumption is the *inner product* (IP) *assumption* which postulates that for any $\epsilon > 0$, there exists a $c_{\text{ip}} = c_{\text{ip}}(\epsilon)$ such that given a pair of sets $A, B \subseteq \{0,1\}^d$ of cardinality N each and of dimension $d = c_{\text{ip}} \cdot \log N$, and an integer $\sigma \in \{0, 1, \ldots, d\}$, it is impossible to determine whether there exists a pair $(a, b) \in A \times B$ satisfying that $\langle a, b\rangle = \sigma$ in $N^{2-\epsilon}$ time. Once more, since the OV problem is a special case of the IP problem, the IP assumption is at least as plausible as OVC[1]. Indeed, the best known algorithms for the IP problem are only able to solve this problem in time $N^{2-\epsilon}$ with $c \approx 1/\epsilon$ [5], and this only imposes that $c_{\text{ip}} \gtrapprox 1/\epsilon$.[2]

In recent years, there has been a flurry of work showing fine-grained lower bounds for many central computational problems in P, based on the above assumptions. A main challenge in showing such fine-grained lower bounds based on these assumptions is that one must carefully design the reductions so that they run fast enough as not to supersede the lower bound assumptions.

One fundamental problem for which such fine-grained reductions were shown is the *Closest Pair* (CP) *problem*. In this problem, given a distance metric $\text{dist} : \{0,1\}^d \times \{0,1\}^d \to \mathbb{R}^+$, and given a pair of sets $A, B \subseteq \{0,1\}^d$, the goal is to find a pair $(a, b) \in A \times B$ which minimizes $\text{dist}(a, b)$. This problem was studied for various metrics such as Hamming, ℓ_p, and edit distance, and it has many applications, for example in computational geometry, geographic information systems [12], clustering [24, 6], and matching problems [23], to name a few. For concreteness, in what follows we restrict our attention only to the Euclidean ℓ_2 metric, though many of the results we mention hold also for other metrics.

---

[1] The IP assumption is at least as plausible as OVC if we allow an arbitrary dependence of c_{ip} on epsilon.
[2] We use $\approx$, $\gtrsim$, $\lesssim$ to hide polylog factors.

One can naïvely solve the (Euclidean) Closest Pair problem in $O(N^2 d)$ time. On the other hand, algorithms have been developed which solve the problem in time $\approx N^{O(c)}$ in the low-dimensional regime $d = c \log N$, [16, 14]; Thus, a truly sub-quadratic algorithm is only known for smaller values of c. On the other hand, in [5] it was shown that assuming OVC, for any $\epsilon > 0$ there exists $c = c(\epsilon)$ so that no algorithm can solve this problem in time $N^{2-\epsilon}$.

1.1 Fine-grained hardness of approximation

Given the above state of affairs, it is natural to ask whether relaxing the requirements and settling for an approximate "close-enough" answer can help in designing faster fine-grained algorithms. For example, it is known that for the (Euclidean) CP problem, one can obtain a $(1+\delta)$-approximation with running time $N^{2-\epsilon}$ for $\delta \approx \epsilon^3$ (for any dimension $d \leq N^{1-\epsilon}$) [4], which is much faster than the best-known exact algorithm.

In terms of impossibility results, known fine-grained reductions can typically be adapted to the approximate setting, based on appropriate *gap assumptions*, such as Gap-SETH.[3] In the theory of NP-hardness, it is often possible to base hardness of gap-problems on hardness of exact problems using PCPs. However, a major barrier in applying this approach in the fine-grained setting (for example for the purpose of reducing SETH to Gap-SETH) is the large (super-constant) blow-up in the length of existing PCPs, which translates into a large (super-constant) blow-up in the number of variables n in the reduction.

Nevertheless, in a recent breakthrough, Abboud, Rubinstein, and Williams [2] have shown how to utilize PCP machinery (specifically, the sumcheck protocol) for showing fine-grained hardness of approximation results based on *non-gap assumptions*. Since then, many works have utilized this framework for showing fine-grained hardness of approximation results for many central problems in P, based on non-gap assumptions such as SETH or OVC (see the recent surveys [18, 10] for a description of this line of work).

In particular, for the CP problem, Rubinstein [17] has shown that assuming OVC, for any $\epsilon > 0$ there exists $\delta = \delta(\epsilon)$ such that there is no $(1+\delta)$-approximation algorithm for (Euclidean) CP running in time $N^{2-\epsilon}$. This rules out truly sub-quadratic approximation algorithms, running, say, in time $f(\delta) \cdot N^{1.99}$. However, the obtained dependence of δ on ϵ is far from optimal, specifically $\delta = \exp(-c_{\text{ov}}/\epsilon)$, where $c_{\text{ov}} = c_{\text{ov}}(\epsilon) \geq \exp(1/\epsilon)$ is the constant guaranteed by the OVC conjecture.

In a follow-up work, Chen and Williams [9] have shown an improved hardness of approximation result for CP in which δ only depends polynomially on ϵ. Specifically, they showed that if the IP assumption holds, then for any $\epsilon > 0$ there is no $(1+\delta)$-approximation algorithm for (Euclidean) CP running in time $N^{2-\epsilon}$, where $\delta = \text{poly}(\epsilon/c_{\text{ip}})$ and $c_{\text{ip}} = c_{\text{ip}}(\epsilon) \gtrapprox 1/\epsilon$ is the constant guaranteed by the IP assumption. However, the obtained dependence of δ on ϵ was still quite small, on the order of $\delta \approx (\epsilon/c_{\text{ip}})^6 \lessapprox \epsilon^{12}$. This is still quite far from the dependence obtained by the best known approximation algorithm for CP which gives an $(1+\delta)$-approximation in time $N^{2-\epsilon}$ for $\delta \approx \epsilon^3$.

In this work, we investigate the question of whether the dependence of δ on ϵ can even be further improved, potentially to match the best known approximation algorithm.

[3] The Gap-SETH assumption asserts that for any $\epsilon > 0$, there are k and $\delta > 0$, so that no $2^{(1-\epsilon)n}$-time algorithm can, given a k-CNF on n variables, distinguish between the case that it is satisfiable, and the case that any assignment satisfies at most an $(1-\delta)$-fraction of its clauses.

1.2 Our results

Recall that by the discussion above, the best known approximation algorithm for (Euclidean) CP gives an $(1+\delta)$-approximation in time $N^{2-\epsilon}$ for $\delta \approx \epsilon^3$, while the best-known hardness of approximation result shows that if the IP assumption holds, then no $(1+\delta)$-approximation algorithm running in time $N^{2-\epsilon}$ exists for $\delta \approx (\epsilon/c_{\text{ip}})^6 \lessapprox \epsilon^{12}$, where $c_{\text{ip}} = c_{\text{ip}}(\epsilon) \gtrapprox 1/\epsilon$ is the constant guaranteed by the IP assumption. Thus there remains a large polynomial gap between the upper and lower bounds, and our main result narrows this gap.

▶ **Theorem 1.1.** *Suppose that the IP assumption holds, i.e., for any $\epsilon' > 0$, there exists a $c_{\text{ip}} = c_{\text{ip}}(\epsilon')$ such that given a pair of sets $A, B \subseteq \{0,1\}^d$ of cardinality N each and of dimension $d = c_{ip}(\epsilon') \cdot logN$, and an integer $\sigma \in \{0, 1, \ldots, d\}$, it is impossible to find a pair $(a, b) \in A \times B$ satisfying that $\langle a, b\rangle = \sigma$ in $N^{2-\epsilon'}$ time.*

Then for any $\epsilon > 0$, there is $\delta = \tilde{\Theta}((\frac{\epsilon}{c_{\text{ip}}(\epsilon/2)})^2)$, so that any algorithm running in time $N^{2-\epsilon}$ cannot $(1+\delta)$-approximate Euclidean CP.

Recall that it is known how to solve the IP problem in time $N^{2-\epsilon}$ for dimension $d = c \log N$ with $c \approx 1/\epsilon$, and so it must hold that $c_{\text{ip}} \gtrapprox 1/\epsilon$. If we assume that $c_{\text{ip}} \approx 1/\epsilon$, then the above theorem gives a dependence of δ on ϵ of the form $\delta \approx \epsilon^4$, which is very close to the dependence of $\delta \approx \epsilon^3$ given by the best known algorithm. Moreover, improving the dependence in the above theorem to $\delta \approx \frac{\epsilon}{c_{\text{ip}}}$ would imply an algorithmic improvement on the IP problem. We leave the question of determining the exact dependence of δ on ϵ as an interesting open problem for future research.

By known reductions, the above theorem gives a similar improvement for a host of other problems in fine-grained hardness of approximation such as closest pair with respect to other metrics such as Hamming, ℓ_p-norm for any constant $p > 0$, and edit distance, Furthest Pair and approximate nearest neighbor in these metrics, and additive approximations to Max-IP and Min-IP, see Appendix A for more details.

Finally, we remark that the above theorem also holds under OVC (or SETH), but is less meaningful, since as discussed above, for the OV problem we have that $c_{\text{ov}} \geq \exp(1/\epsilon)$.

1.3 Proof overview

Next we give an overview of our proof method, and how it improves on prior work. To this end, we first describe the general framework presented in [2] for obtaining fine-grained hardness of approximation results based on MA communication protocols. Then we discuss the work of Rubinstein [17] who relied on this framework to give the first fine-grained hardness of approximation result for CP, albeit with an exponential dependence of δ on ϵ, and the work of Chen and Williams [9] who improved this dependence to polynomial. Following this, we turn to discuss our proof method that obtains a tighter polynomial relation.

Fine-grained hardness of approximation via MA communication [2]. In a Merlin-Arthur (MA) communication protocol for a function $f : \{0,1\}^d \times \{0,1\}^d \to \{0,1\}$, two players Alice and Bob wish to compute $f(a, b)$, where Alice is given as input only $a \in \{0,1\}^d$, and Bob is given as input only $b \in \{0,1\}^d$. To this end, Alice and Bob engage in a randomized (public coin) communication protocol, where their goal is to use as little communication as possible. To aid them with this task, there is also a (potentially malicious) prover Merlin who sees Alice's and Bob's inputs, and before any communication begins Merlin sends Alice a short message m, which can be thought of as a "proof" or "advice". The requirement is that if

$f(a,b)=1$, then there must exist some message m from Merlin on which Alice accepts with probability 1. Otherwise, if $f(a,b)=0$, then for any possible message $\tilde{m}$ from Merlin, Alice accepts with probability at most $\frac{1}{2}$ on $\tilde{m}$.[4]

In [2], it was shown that an efficient MA communication protocol for *set disjointness*[5] implies a fine-grained reduction from OV to an approximate version of Max-IP in which given two sets $A,B\subseteq\{0,1\}^d$ of cardinality N each, the goal is to output a number sufficiently close to $M:=\max_{a\in A,b\in B}\langle a,b\rangle$.

To see how a reduction as above can be constructed, suppose that there exists an MA communication protocol for set disjointness with Merlin's message length L, communication complexity cc between Alice and Bob, and randomness complexity R. Suppose furthermore that we are given an instance $A,B\subseteq\{0,1\}^d$ of OV, where $|A|=|B|=N$. Then for each possible Merlin's message $m\in\{0,1\}^L$, we construct an instance $A_m,B_m\subseteq\{0,1\}^{2^{\mathrm{cc}+R}}$ of Max-IP, where $|A_m|=|B_m|=N$.

Fix $m\in\{0,1\}^L$. Then the set A_m is obtained from A by mapping each element $a\in A$ to a binary vector a_m that contains an entry for each possible transcript $\Gamma\in\{0,1\}^{\mathrm{cc}}$ and randomness string $r\in\{0,1\}^R$ (so a_m has length $2^{\mathrm{cc}+R}$), and whose (Γ,r)-entry equals 1 if and only if Γ is consistent with r and a, and Alice accepts on input a, randomness string r, and transcript Γ. The set B_m is obtained analogously from B.

Then the main observation is that for some $m\in\{0,1\}^L$, we have that the (Γ,r)-entry of both a_m and b_m equals 1 if and only if Alice accepts on Merlin's message m, inputs a and b, and randomness string r. Consequently, if there exists $(a,b)\in A\times B$ so that $\langle a,b\rangle=0$ (i.e., $f(a,b)=1$), then there exists a Merlin's message m on which Alice accepts with probability 1 on inputs a and b, and consequently we have that the corresponding vectors $(a_m,b_m)\in A_m\times B_m$ satisfy that $\langle a_m,b_m\rangle=2^R$. On the other hand, if $\langle a,b\rangle\neq 0$ (i.e., $f(a,b)=0$) for any $(a,b)\in A\times B$, then for any Merlin's message $\tilde{m}$, and on any inputs $(a,b)\in A\times B$, Alice accepts with probability at most $\frac{1}{2}$, and so $\langle a_{\tilde{m}},b_{\tilde{m}}\rangle\leq\frac{1}{2}\cdot 2^R$ for any pair $(a_{\tilde{m}},b_{\tilde{m}})\in A_{\tilde{m}}\times B_{\tilde{m}}$. This gives the desired gap, showing that OV reduces to 2^L instances of approximate Max-IP.

To obtain a fine-grained reduction, one must make sure that cc, R and L are not too large, so that the total construction time of the reduction is at most N^ϵ. To achieve this, one can use the MA communication protocol of Aaronson and Wigderson [1] for set disjointness in which all these quantities are upper bounded by $\approx\sqrt{d}$.

For $d=c\log N$, this gives that 2^{cc}, 2^R, and 2^L are all upper bounded by $2^{\tilde{O}(\sqrt{\log N})}\ll N^\epsilon$, and so the reduction can be constructed in time N^ϵ. Next we describe the MA communication protocol of [1], as hardness of approximation results for CP (including ours) crucially rely on its properties.

MA communication protocol for set disjointness [1]. The MA communication protocol for set disjointness of Aaronson and Wigderson [1] relies on the influential sumcheck protocol of [15], and it proceeds as follows.

Let $a\in\{0,1\}^d$ be Alice's input. Slightly abusing notation, we view a as a $\sqrt{d}\times\sqrt{d}$ binary matrix in the natural way, and we let $\hat{a}$ denote the $p\times\sqrt{d}$ matrix obtained by encoding each column of a with a systematic Reed-Solomon code $\mathrm{RS}_{\sqrt{d},p}:\mathbb{F}_p^{\sqrt{d}}\to\mathbb{F}_p^p$ of degree $\sqrt{d}$ over a prime field of size $p\approx 4\sqrt{d}$.[6] Let $\hat{b}$ be defined analogously.

[4] The accept probability can be increased by executing the communication phase between Alice and Bob independently for multiple times and accepting if and only if all invocations accept.

[5] Recall that *set disjointness* is the function $\mathrm{disj}:\{0,1\}^d\times\{0,1\}^d\to\{0,1\}$ which satisfies that $\mathrm{disj}(a,b)=1$ if and only if the supports of a and b are disjoint, i.e., if $\langle a,b\rangle=0$.

[6] The systematic Reed-Solomon code $\mathrm{RS}_{d,p}:\mathbb{F}_p^d\to\mathbb{F}_p^p$ of degree d over a prime field of size $p>d$ is a linear

In the protocol, Merlin first computes the pointwise product $\hat{a} \star \hat{b} \in \mathbb{F}_p^{p \times \sqrt{d}}$, and then sends Alice the sum $m \in \mathbb{F}_p^p$ of the columns of $\hat{a} \star \hat{b}$ (where arithemtic is performed mod p). Alice first checks that m is a codeword of $\mathrm{RS}_{2\sqrt{d},p}$, and that the first $\sqrt{d}$ entries of m are all zero, otherwise she rejects and aborts. Then Alice and Bob jointly sample a random index $i \in [p]$, Bob sends Alice the i'th row of $\hat{b}$, Alice computes its inner product with the i'th row of $\hat{a}$, and accepts if and only if this product equals $m(i)$ (where once more, arithmetic is performed mod p).

To see that the protocol is complete, note first that if a and b are disjoint, then $a \star b$ is the all-zero matrix. Consequently, by the systematic property of the Reed-Solomon encoding, the first $\sqrt{d}$ rows of $\hat{a} \star \hat{b}$ are also identically zero, which implies in turn that the first $\sqrt{d}$ entries of m are identically zero. Furthermore, since the product of two polynomials of degree at most $\sqrt{d}$ is a polynomial of degree at most $2\sqrt{d}$, it follows that m is a codeword of $\mathrm{RS}_{2\sqrt{d},p}$. Thus, both Alice's checks will clearly pass. It can also be verified that by construction, the inner product of the i'th rows of $\hat{a}$ and $\hat{b}$ equals $m(i)$, and so Alice accepts with probability 1.

To show soundness, suppose that a and b intersect, and let $\tilde{m}$ denote Merlin's message. We may assume that $\tilde{m}$ is a codeword of $\mathrm{RS}_{2\sqrt{d},p}$, and that the first $\sqrt{d}$ entries of $\tilde{m}$ are all zero, since otherwise Alice clearly rejects. But on the other hand, since a and b intersect, then $a \star b$ has a 1-entry, say in the j-th row, and since $p > \sqrt{d}$, the sum of entries in the j'th row of $a \star b$ is non-zero mod p, which implies in turn that $m(j) \neq 0$. Thus, we conclude that $\tilde{m}$ and m are distinct codewords of $\mathrm{RS}_{2\sqrt{d},p}$ – a code of distance at least $\frac{p}{2}$ – and so they must differ by at least $\frac{1}{2}$ of their entries. But this implies in turn that with probability at least $\frac{1}{2}$ over the choice of i, it holds that $\tilde{m}(i) \neq m(i)$, in which case the inner product of the i'th row of $\hat{a}$ and $\hat{b}$ will be different than $\tilde{m}(i)$, which will cause Alice to reject.

Finally, it can also be verified that in this protocol, cc, R, and L are all upper bounded by $\approx \sqrt{d}$.

Hardness of approximation for CP with exponential dependence [17]. In [17], Rubinstein utilized the above framework to show fine-grained hardness of approximation for CP. The starting point of [17] is a simple linear-time reduction from δ-additive approximation for Max-IP[7] to an $(1+\Theta(\delta))$-approximation for (Euclidean) CP. Thus, to show that no algorithm can find an $(1+\Theta(\delta))$-approximation for (Euclidean) CP in time $N^{2-\epsilon}$, it suffices to show that no algorithm can find a δ-additive approximation for Max-IP in time $N^{2-\epsilon}$.

The [2] framework discussed above generates instances of Max-IP of dimension $2^{\mathrm{cc}+R}$ and additive gap of $\frac{1}{2} \cdot 2^R$, which gives $\delta := \Theta(2^{-\mathrm{cc}})$. However, the MA protocol of [1] described above only gives cc $\approx \sqrt{d}$ which is super-constant for a super-constant dimension d, and consequently only yields a *sub-constant* δ.

To deal with this, [17] first utilized the fact (previously utilized also in [2]) that the [1] protocol described above works equally well on skewed matrices of dimensions $\frac{d}{T} \times T$, in which case we have that $L = \frac{d}{T} \cdot \log p$ and $\mathrm{cc} = T \cdot \log p$. Thus, assuming $d = c \log N$, to achieve $2^L \leq N^\epsilon$, one can set $T = \frac{c}{\epsilon} \cdot \log p$, which gives in turn $\mathrm{cc} = \frac{c}{\epsilon} \cdot \log^2 p$.

---

map, defined as follows. To encode a message $m = (m(0), \ldots, m(d-1)) \in \mathbb{F}_p^d$, one finds the (unique) degree $d-1$ polynomial $P_m(X) \in \mathbb{F}_p[X]$ which satisfies that $P_m(i) = m(i)$ for any $i = 0, \ldots, d-1$, and lets $\mathrm{RS}_{d,p}(m) = (P_m(0), \ldots, P_m(p-1))$. The code is called *systematic* since the message is a prefix of its encodings. The code has distance at least $p-d+1$ since any pair of distinct degree $d-1$ polynomials can agree on at most $d-1$ points.

[7] In a *δ-additive approximation* for Max-IP, given $A, B \subseteq \{0,1\}^d$ of cardinality N each, the goal is to output a number in $[M - \delta \cdot d, M]$, where $M := \max_{a \in A, b \in B} \langle a, b \rangle$.

However, this is still not quite enough since the MA protocol of [1] requires setting $p > \sqrt{d}$ because of the use of Reed-Solomon codes that are only defined over a large alphabet, and consequently the communication complexity is still super-constant. However, the main observation in [1] is that the protocol can actually be executed using any error-correcting code with a *multiplication property*[8]. Relying on this observation, Rubinstein replaced the Reed-Solomon codes in the protocol of [1] with algebraic-geometric (AG) codes that satisfy the multiplication property over a constant-size alphabet. This reduced the communication complexity to $\approx c/\epsilon$, yielding in turn an approximation factor of $\delta = 2^{-\tilde{\Theta}(c/\epsilon)}$.

Polynomial dependence [9]. While [17] gave the first non-trivial hardness of approximation result for CP, a downside of this result was that the approximation factor δ depended exponentially on the running time parameter ϵ. In the follow-up work [9], Chen and Williams showed how to reduce this dependence to just polynomial.

The main observation of Chen and Williams was that instead of thinking of the output of the MA protocol of [17] as being just accept or reject, one can view the output as being *short vectors* $a', b' \in \mathbb{F}_p^T$ (namely, the i'th row of $\hat{a}, \hat{b}$, respectively), and $\sigma' \in \{0, 1, \ldots, T \cdot p^2\}$ (namely, the i'th entry of Merlin's message m), where a' only depends on Alice's input a and the randomness string, b' only depends on Bob's input b and the randomness string, and σ' only depends on Merlin's message and the randomness string. The requirement then is that if $\langle a, b\rangle = 0$ for some $(a, b) \in A \times B$, then for some Merlin's message m, $\langle a', b'\rangle = \sigma'$ with probability 1, while if $\langle a, b\rangle \neq 0$ for any $(a, b) \in A \times B$, then for any Merlin's message $\tilde{m}$, then $\langle a', b'\rangle \neq \sigma'$ with probability at least $\frac{1}{2}$.

Chen and Williams then suggested to create an instance A_m, B_m for any Merlin's message m, where the set A_m is obtained from A by simply mapping each element $a \in A$ to a vector $a_m \in \mathbb{F}_p^{T \times 2^R}$ that is the concatenation of all possible output vectors a' for all possible randomness strings, and analogously for B_m. The advantage is that now the dimension of the vectors in A_m and B_m is much shorter than in [17]. However, a disadvantage is that now the alphabet is not binary anymore, and an even more serious problem is that the soundness guarantee is only that $\langle a', b'\rangle \neq \sigma'$, so the reduction does not seem to produce any gap.

To deal with these issues, Chen and Williams use an *encoding lemma* which gives mappings g, h and a value Γ, where g, h, and Γ only depend on p and T, so that $g(a', \sigma')$ and $h(b', \sigma')$ are binary vectors of length $\text{poly}(p, T)$ satisfying that if $\langle a', b'\rangle = \sigma'$ then $\langle g(a', \sigma'), h(b', \sigma')\rangle = \Gamma$, while if $\langle a', b'\rangle \neq \sigma'$ then $\langle g(a', \sigma'), h(b', \sigma')\rangle < \Gamma$ (see Lemma 4.2 for a formal statement). This produces the desired additive gap, on the order of $\Omega(2^R)$. Since the encoding lemma increases the dimension of the vectors only by a factor of $\text{poly}(p, T)$, this yields an approximation factor of $\delta = \frac{1}{\text{poly}(p,T)} = \text{poly}(\frac{\epsilon}{c})$.

We note that a delicate issue that should be dealt with in the reduction is that the encoding lemma works over the integers, while the protocol works over finite fields, and in particular, over non-prime fields, as AG codes are only known to exist over non-prime fields. Additionally, Chen and Williams show that the reduction works equally well when using the IP problem instead of OV as its starting point, and using an MA communication protocol for IP similar to that of [1]. This can potentially allow for a smaller value of c as it is only known how to solve (exact) IP in time $N^{2-\epsilon}$ up to a dimension of $c \log N$ for $c \approx 1/\epsilon$.

[8] Informally, we say that a linear code $C : \mathbb{F}^k \to \mathbb{F}^n$ has a *multiplication property* if the set $\text{span}\{C(m) \star C(m') \mid m, m' \in \mathbb{F}^k\}$ has sufficiently large distance.

This work – tighter polynomial dependence. While [9] obtained a polynomial dependence of δ on ϵ, the dependence was quite small, on the order of $\delta \approx (\frac{\epsilon}{c})^6$, and in the current work we show how to improve the dependence to $\delta \approx (\frac{\epsilon}{c})^2$.

To this end, we first observe that one reason for the small polynomial dependence obtained in [9] was the large polynomial dependence of the dimension of the resulting vectors in the encoding lemma on the alphabet size p. While in the protocol of [17] the field size p can be made constant using AG codes, the field still needs to be of characteristic at least T, since otherwise the sum of entries in a non-zero row of $a \star b$ may sum to zero over $\mathbb{F}_p$, and consequently the soundness analysis will not go through.

To reduce the alphabet size, we first design a new MA protocol in which the alphabet size is only *polylogarithmic* in T (see Theorem 3.1). This protocol is inspired by the MA protocol of [8] for IP which achieved communication complexity $O(\sqrt{d \log d \log \log d})$, improving on the communication complexity of $O(\sqrt{d} \log d)$ of [1]. In a nutshell, Chen's idea was to execute the original MA protocol of [1] multiple times over different small prime fields, hoping that if $\langle a, b \rangle \neq 0$, then $\langle a, b \rangle$ is also non-zero modulo many of the primes, and so the protocol will be executed correctly. Chen showed that this is indeed possible to achieve using $O(\log d)$ distinct primes of cardinality at most polylog(d) each.

We observe that for skewed matrices of dimensions $\frac{d}{T} \times T$, it in fact suffices to execute the protocol with $O(\log T)$ distinct primes of cardinality at most polylog(T) each. While this choice does not necessarily guarantee the property above that if $\langle a, b \rangle \neq 0$, then $\langle a, b \rangle$ is also non-zero modulo many of the primes, this turns out to still suffice for a correct execution of the protocol.

We then further observe that such a protocol can be used in the framework of [9] to obtain an improved hardness of approximation result for Max-IP. Once more, a delicate issue is how to use the encoding lemma in the presence of many different non-prime fields.

To the best of our knowledge, this is the first use of the techniques underlying the improved MA protocol of [8] for showing a fine-grained hardness of approximation result.[9]

Paper organization. The rest of the paper is organized as follows. We begin in Section 2 below with the required notation and terminology with respect to fine-grained complexity problems and error-correcting codes. Then in Section 3 we present our improved MA protocol over a small alphabet, while in Section 4 we show how to use this protocol for obtaining an improved reduction from IP to approximate Max-IP. Finally, in Appendix A we show implications of our latter result to showing hardness of approximation results for closest pair, as well as other related problems in fine-grained complexity.

2 Preliminaries

We start by setting some general notation. For a positive integer d, we let $[d] := \{1, 2, \ldots, d\}$. For convenience, we often view a vector $a \in \Sigma^d$ as a function $a : [d] \to \Sigma$, and we let $a(i)$ denote the i-th entry of a. For a pair of vectors $a, b \in \mathbb{N}^d$, we let $\langle a, b \rangle := \sum_{i=1}^{d} a(i) \cdot b(i)$ denote their inner product, and we let $a \star b \in \mathbb{N}^d$ denote their pointwise product, given by $(a \star b)(i) = a(i) \cdot b(i)$ for $i \in [d]$. For $a, b \in \Sigma^d$, we let $\Delta(a, b) := |\{i \in [d] \mid a(i) \neq b(i)\}|$ denote their Hamming distance. For an $n \times k$ matrix A and $i \in [n]$ ($j \in [k]$, respectively), we let $\text{row}_i(A)$ ($\text{col}_j(A)$, respectively) denote the i-th row (j-th column, respectively) of A.

[9] The paper [8] contains various hardness of approximation results for Max-IP, as well as the improved MA protocol for IP. To the best of our knowledge, the improved MA protocol presented in this paper was not used in this paper or in any subsequent work as the basis for hardness of approximation results.

2.1 Problems in fine-grained complexity

Below we list the main fine-grained problems that we will be concerned with in this paper.

▶ **Definition 2.1** (Inner Product (IP)). *In the inner product* $\mathrm{IP}_{N,d}$ *problem, given two sets* $A, B \subseteq \{0,1\}^d$ *of cardinality* N *each, and an integer* $\sigma \in \{0,1,\ldots,d\}$*, the goal is to determine whether there exists a pair* $(a,b) \in A \times B$ *satisfying that* $\langle a,b\rangle = \sigma$*.*

▶ **Definition 2.2** (Maximum Inner Product (Max-IP)). *In the maximum inner product* Max-$\mathrm{IP}_{N,d}$ *problem, given two sets* $A, B \subseteq \{0,1\}^d$ *of cardinality* N *each, the goal is to compute* $M := \max_{a\in A, b\in B} \langle a,b\rangle$*.*

For the approximate version of Max-IP, defined next, we will consider the less standard *additive* approximation version that will be useful for obtaining hardness of approximation for the closest pair problem.

▶ **Definition 2.3** (Approximate Maximum Inner Product (Apx-Max-IP)). *Let* $\delta > 0$ *be a parameter. In the (additive) approximate maximum inner product* δ-Apx-Max-$\mathrm{IP}_{N,d}$ *problem, given two sets* $A, B \subseteq \{0,1\}^d$ *of cardinality* N *each, the goal is to output a number in* $[M - \delta \cdot d, M]$*, where* $M := \max_{a\in A, b\in B} \langle a,b\rangle$*.*

▶ **Definition 2.4** (Closest Pair (CP)). *Let* $\mathrm{dist} : \{0,1\}^d \times \{0,1\}^d \to \mathbb{R}^+$ *be a distance function. In the closest pair* $\mathrm{CP}_{N,d,\mathrm{dist}}$ *problem, given two sets* $A, B \subseteq \{0,1\}^d$ *of cardinality* N *each, the goal is to compute* $M := \min_{a\in A, b\in B} \mathrm{dist}(a,b)$*.*

▶ **Definition 2.5** (Approximate Closest Pair (Apx-CP)). *Let* $\mathrm{dist} : \{0,1\}^d \times \{0,1\}^d \to \mathbb{R}^+$ *be a distance function, and let* $\delta > 0$ *be a parameter. In the approximate closest pair* δ-Apx-$\mathrm{CP}_{N,d,\mathrm{dist}}$ *problem, given two sets* $A, B \subseteq \{0,1\}^d$ *of cardinality* N *each, the goal is to output a number in* $[M, (1+\delta)M]$*, where* $M := \min_{a\in A, b\in B} \mathrm{dist}(a,b)$*.*

2.2 Error-correcting codes

Our reduction will make use of error-correcting codes. In what follows, we first present some general notation and terminology with respect to error-correcting codes, and then describe the kind of codes we shall use for our reduction.

Let Σ be a finite alphabet, and k, n be positive integers (the message length and the codeword length, respectively). An (error-correcting) code is an injective map $C : \Sigma^k \to \Sigma^n$. The elements in the domain of C are called messages, and the elements in the image of C are called codewords. We say that C is systematic if the message is a prefix of the corresponding codeword, i.e., for every $x \in \Sigma^k$ there exists $z \in \Sigma^{n-k}$ such that $C(x) = (x,z)$. The rate of a code $C : \Sigma^k \to \Sigma^n$ is the ratio $\rho := \frac{k}{n}$. The relative distance $\mathrm{dist}(C)$ of C is the maximum $\delta > 0$ such that for every pair of distinct messages $x, y \in \Sigma^k$ it holds that $\Delta(C(x), C(y)) \geq \delta$.

If $\Sigma = \mathbb{F}$ for some finite field $\mathbb{F}$, and C is a linear map between the vector spaces $\mathbb{F}^k$ and $\mathbb{F}^n$ then we say that C is linear. The generating matrix of a linear code $C : \mathbb{F}^k \to \mathbb{F}^n$ is a matrix $G \in \mathbb{F}^{n\times k}$ such that $C(x) = G \cdot x$ for any $x \in \mathbb{F}^k$. We say that a linear code C is explicit if G can be generated in time $\mathrm{poly}(n)$.

For our reduction, we shall require linear codes over a small (constant-size, independent of the codeword length) alphabet, satisfying the *multiplication property*, which informally says that all pointwise products of pairs of codewords span a code of large distance. Such codes can be obtained from the AG codes of [19] (see also [17, Theorem 2.4]).

▶ **Theorem 2.6** ([19]; [17], Theorem 2.4). *There exists a constant integer* p_0 *so that for any prime* $p \geq p_0$*, there exist two explicit code families* $\mathcal{C} = \{C_k\}_{k\in\mathbb{N}}$ *and* $\mathcal{C}_\star = \{(C_\star)_k\}_{k\in\mathbb{N}}$ *so that the following hold for any* $k \in \mathbb{N}$*:*

- $C_k, (C_\star)_k$ *are systematic linear codes over* $\mathbb{F}_{p^2}$ *of relative distance at least* 0.1 *and rate at least* 0.1.
- C_k *has message length* k.
- *For any* $x, y \in (\mathbb{F}_{p^2})^k$, $C_k(x) \star C_k(y)$ *is a codeword of* $(C_\star)_k$.

3 MA protocol for IP over a small alphabet

In this section, we will provide an MA protocol for IP over a small alphabet. The protocol will be later used in Section 4 below to show a reduction from IP to Apx-Max-IP.

▶ **Theorem 3.1** (MA Protocol for IP over a small alphabet). *For any sufficiently large integer* T*, there is an integer* $q = O(\log^2 T)$*, so that for any integer* d *which is a multiple of* T *there is an* MA *Protocol which satisfies the following:*

1. *Alice is given as input a vector* $a \in \{0,1\}^d$ *and an integer* $\sigma \in \{0, 1, \ldots, d\}$*, Bob is given as input a vector* $b \in \{0,1\}^d$*, and Merlin is given as input* a*,* b*, and* σ*.*
2. *Merlin sends Alice a message* m *of (bit) length* $L = O(\frac{d}{T} \cdot \log^2 T)$*. Alice reads Merlin's message, and based on this message and* σ*, decides whether to reject and abort, or continue.*
3. *Alice and Bob sample a joint random string* r *of (bit) length* $R = \log(\frac{d}{T}) + \log\log T + O(1)$*.*
4. *Alice outputs a string* $a' \in \{0, 1, \ldots, q\}^T$ *and an integer* $\sigma' \in \{0, 1, \ldots, T \cdot q^2\}$*, where* a' *only depends on Alice's input* a *and the randomness string* r*, and* σ' *only depends on Merlin's message* m *and* r*, and Bob outputs a string* $b' \in \{0, 1, \ldots, q\}^T$*, which only depends on Bob's input* b *and* r*, so that the following hold:*
   - *(Completeness) If* $\langle a, b\rangle = \sigma$*, then on Merlin's message* m*, Alice and Bob output* a', b'*, and* σ' *so that* $\langle a', b'\rangle = \sigma'$ *with probability* 1*.*
   - *(Soundness) If* $\langle a, b\rangle \neq \sigma$*, then for any Merlin's message* $\tilde{m}$*, Alice and Bob output* a', b'*, and* σ' *so that* $\langle a', b'\rangle = \sigma'$ *with probability at most* 0.98*.*

   Moreover, the running time of both Alice and Bob is $\mathrm{poly}(d)$*.*

The main difference between the above protocol and that of [9], is that instead of working over a field of characteristic $\Theta(T)$, we perform the protocol of [9] simultaneously over $O(\log T)$ different fields of size $O(\log^2 T)$ each.

To this end, we start by fixing some notation. Let t be an integer such that $t^t = T$. By Lemma 2.4 in [8], for a large enough integer t, there exist $10t$ distinct primes $p_1 < p_2 < \cdots < p_{10t}$, where the value of each prime is bounded in the interval $[t, t^2]$. Let $q := t^2$, and note that $t = O(\log T)$ and $q = O(\log^2 T)$. For each $\ell \in [10t]$, let $C^{(\ell)}, C_\star^{(\ell)}$ be the systematic linear codes over $\mathbb{F}_{p_\ell^2}$ guaranteed by Theorem 2.6, where $C^{(\ell)}$ has message length $\frac{d}{T}$ and codeword length $n_\ell := O(\frac{d}{T})$. Finally, recall that the elements of $\mathbb{F}_{p_\ell^2}$ can be viewed as degree 1 polynomials over $\mathbb{F}_{p_\ell}$, where multiplication is performed modulo an irreducible polynomial Q_ℓ of degree 2 over $\mathbb{F}_{p_\ell}$.

Let $a \in \{0,1\}^d$ be Alice's input. Slightly abusing notation, we view a as a $\frac{d}{T} \times T$ binary matrix in the natural way. For $\ell \in [10t]$, let $a^{(\ell)}$ denote the $n_\ell \times T$ matrix over $\mathbb{F}_{p_\ell^2}$ obtained by encoding each column of a with the code $C^{(\ell)}$. View each entry of $a^{(\ell)}$ as a degree 1 polynomial over $\mathbb{F}_{p_\ell}$, and let $a^{(\ell,0)}, a^{(\ell,1)}$ denote the $n_\ell \times T$ matrices over $\mathbb{F}_{p_\ell}$, obtained from $a^{(\ell)}$ by keeping in each entry only the free coefficient and linear coefficient, respectively. Let $b, b^{(\ell)}, b^{(\ell,0)}, b^{(\ell,1)}$ be defined analogously for $\ell \in [10t]$. In what follows, all arithmetic operations are performed over the reals, unless otherwise stated.

The protocol. The protocol proceeds as follows:

1. **a.** Merlin sends

$$m_0 := \sum_{j=1}^{T} \mathrm{col}_j(a) \star \mathrm{col}_j(b) \in \{0,1,\ldots,T\}^{d/T}.$$

   b. For $\ell = 1,\ldots,10t$ and $\alpha,\beta \in \{0,1\}$, Merlin sends

$$m_{\ell,\alpha,\beta} := \sum_{j=1}^{T} \mathrm{col}_j(a^{(\ell,\alpha)}) \star \mathrm{col}_j(b^{(\ell,\beta)}) \in \{0,1,\ldots,T\cdot q^2\}^{n_\ell}.$$

2. **a.** Alice checks that $\sum_{i=1}^{d/T} m_0(i) = \sigma$.
   b. Alice checks that $m_{\ell,0,0}(i) = m_0(i)$ and $m_{\ell,0,1}(i) = m_{\ell,1,0}(i) = m_{\ell,1,1}(i) = 0$ for $\ell = 1,\ldots,10t$ and $i = 1,\ldots,\frac{d}{T}$.
   c. For $\ell = 1,\ldots,10t$, let $m_\ell \in (\mathbb{F}_{p_\ell^2})^{n_\ell}$ given by

$$m_\ell = m_{\ell,0,0} + (m_{\ell,0,1} + m_{\ell,1,0}) \cdot X + m_{\ell,1,1} \cdot X^2 \pmod{Q_\ell}.$$

   Alice checks that m_ℓ is a codeword of $C_\star^{(\ell)}$ for $\ell = 1,\ldots,10t$.
   If any of the checks is unsatisfied, then Alice rejects and aborts.
3. Alice and Bob jointly sample $\ell_* \in [10t]$, $i_* \in [n_{\ell_*}]$, and $\alpha_*,\beta_* \in \{0,1\}$.
4. Alice outputs $a' := \mathrm{row}_{i_*}(a^{(\ell_*,\alpha_*)}) \in \{0,1,\ldots,q\}^T$ and $\sigma' := m_{\ell_*,\alpha_*,\beta_*}(i_*) \in \{0,1,\ldots,T\cdot q^2\}$, and Bob outputs $b' := \mathrm{row}_{i_*}(b^{(\ell_*,\beta_*)}) \in \{0,1,\ldots,q\}^T$.

It can be verified that the protocol has the required structure, and that the running times of Alice and Bob are as claimed. Next we show completeness and soundness.

Completeness. Suppose that $\langle a,b\rangle = \sigma$, we shall show that in this case Alice and Bob output a', b', and σ' so that $\langle a',b'\rangle = \sigma'$ with probability 1.

We first show that in this case all of Alice's checks on Step 2 always pass.

To this end, first note that by assumption that $\langle a,b\rangle = \sigma$, we have that

$$\sum_{i=1}^{d/T} m_0(i) = \sum_{i=1}^{d/T}\sum_{j=1}^{T} a(i,j)\cdot b(i,j) = \langle a,b\rangle = \sigma, \tag{1}$$

so Alice's check on Step 2a will pass.

We now show that Alice's check on Step 2b passes. Fix $\ell \in [10t]$, and recall that $a^{(\ell)}$ is obtained by encoding each column of the matrix $a \in \{0,1\}^{\frac{d}{T}\times T}$ with a systematic linear code. Consequently, a is the restriction of $a^{(\ell)}$ to the first $\frac{d}{T}$ rows, and similarly for b. This implies in turn that for any $i \in [\frac{d}{T}]$, we have that

$$m_{\ell,0,0}(i) = \left\langle \mathrm{row}_i(a^{(\ell,0)}), \mathrm{row}_i(b^{(\ell,0)}) \right\rangle = \langle \mathrm{row}_i(a), \mathrm{row}_i(b)\rangle = m_0(i), \tag{2}$$

and

$$m_{\ell,0,1}(i) = m_{\ell,1,0}(i) = m_{\ell,1,1}(i) = 0. \tag{3}$$

So Alice's check on Step 2b will pass as well.

Finally, we show that Alice's check on Step 2c passes. Fix $\ell \in [10t]$, and note that

$$\begin{aligned} m_\ell &= m_{\ell,0,0} + (m_{\ell,0,1} + m_{\ell,1,0}) \cdot X + m_{\ell,1,1} \cdot X^2 \pmod{Q_\ell} \\ &= \sum_{j=1}^{T} \Big[\mathrm{col}_j(a^{(\ell,0)}) \star \mathrm{col}_j(b^{(\ell,0)}) \\ &\quad + \Big(col_j(a^{(\ell,0)}) \star \mathrm{col}_j(b^{(\ell,1)}) + col_j(a^{(\ell,1)}) \star \mathrm{col}_j(b^{(\ell,0)}) \Big) \cdot X \\ &\quad + col_j(a^{(\ell,1)}) \star \mathrm{col}_j(b^{(\ell,1)}) \cdot X^2 \Big] \pmod{Q_\ell} \\ &= \sum_{j=1}^{T} (\mathrm{col}_j(a^{(\ell,0)}) + \mathrm{col}_j(a^{(\ell,1)}) \cdot X) \star (\mathrm{col}_j(b^{(\ell,0)}) + \mathrm{col}_j(b^{(\ell,1)}) \cdot X) \pmod{Q_\ell} \\ &= \sum_{j=1}^{T} \mathrm{col}_j(a^{(\ell)}) \star \mathrm{col}_j(b^{(\ell)}) \pmod{Q_\ell}. \end{aligned} \tag{4}$$

Now, since each column of $a^{(\ell)}$ and $b^{(\ell)}$ is a codeword of $C^{(\ell)}$, we have that $\mathrm{col}_j(a^{(\ell)}) \star \mathrm{col}_j(b^{(\ell)}) \pmod{Q_\ell}$ is a codeword of $C_\star^{(\ell)}$ for any $j \in [T]$. By linearity of $C_\star^{(\ell)}$, this implies in turn that m_ℓ is a codeword of $C_\star^{(\ell)}$, and so Alice's check on Step 2c will also pass.

Thus, we conclude that all of Alice's checks on Step 2 pass. Furthermore, we clearly have that

$$\langle a', b' \rangle = \left\langle \mathrm{row}_{i_*}(a^{(\ell_*,\alpha_*)}), \mathrm{row}_{i_*}(b^{(\ell_*,\beta_*)}) \right\rangle = m_{\ell_*,\alpha_*,\beta_*}(i) = \sigma'.$$

We conclude that in the case that $\langle a, b \rangle = \sigma$, we have that $\langle a', b' \rangle = \sigma'$ with probability 1, as required.

Soundness. Assume that $\langle a, b \rangle \neq \sigma$, and let $\tilde{m}_0$ and $\tilde{m}_{\ell,\alpha,\beta}$ for $\ell = 1, \ldots, 10t$ and $\alpha, \beta \in \{0,1\}$ be Merlin's messages on Step 1. We shall show that in this case Alice and Bob output a', b', and σ' so that $\langle a', b' \rangle = \sigma'$ with probability at most 0.98.

To this end, first note that we may assume that $\sum_{i=1}^{d/T} \tilde{m}_0(i) = \sigma$, since otherwise Alice clearly rejects on Step 2a. On the other hand, by (1) and by assumption that $\langle a, b \rangle \neq \sigma$, we have that $\sum_{i=1}^{d/T} m_0(i) = \langle a, b \rangle \neq \sigma$. Consequently, there exists $i \in [\frac{d}{T}]$ so that $m_0(i) \neq \tilde{m}_0(i)$.

Moreover, since $m_0(i), \tilde{m}_0(i) \in \{0, 1, \ldots, T\}$, we have that $|m_0(i) - \tilde{m}_0(i)| \leq T$. Recalling that $t^t = T$, and that $p_\ell \geq t$ for any $\ell \in [10t]$, we conclude that at most t of the p_ℓ's can divide $|m_0(i) - \tilde{m}_0(i)|$. Thus, with probability at least 0.9 over the choice of ℓ_*, it holds that $p_{\ell *}$ does not divide $|m_0(i) \neq \tilde{m}_0(i)|$, and so $m_0(i) \neq \tilde{m}_0(i) \pmod{p_{\ell_*}}$. In what follows, assume that this event holds.

Let $\tilde{m}_{\ell_*} \in (\mathbb{F}_{p_{\ell_*}^2})^{n_{\ell *}}$ be given by

$$\tilde{m}_{\ell_*} = \tilde{m}_{\ell_*,0,0} + (\tilde{m}_{\ell_*,0,1} + \tilde{m}_{\ell^*,1,0}) \cdot X + \tilde{m}_{\ell^*,1,1} \cdot X^2 \pmod{Q_{\ell_*}}.$$

Next observe that we may assume that for any $i \in [\frac{d}{T}]$,

$$\tilde{m}_{\ell_*}(i) = \tilde{m}_{\ell_*,0,0}(i) + (\tilde{m}_{\ell_*,0,1}(i) + \tilde{m}_{\ell^*,1,0}(i)) \cdot X + \tilde{m}_{\ell^*,1,1}(i) \cdot X^2 \pmod{Q_{\ell_*}} = \tilde{m}_0(i) \pmod{p_{\ell_*}},$$

since otherwise Alice clearly rejects on Step 2b. On the other hand, by (2) and (3) we have that for any $i \in [\frac{d}{T}]$,

$$m_{\ell_*}(i) = m_{\ell_*,0,0}(i) + (m_{\ell_*,0,1}(i) + m_{\ell^*,1,0}(i)) \cdot X + m_{\ell^*,1,1}(i) \cdot X^2 \pmod{Q_{\ell_*}} = m_0(i) \pmod{p_{\ell_*}}.$$

Consequently, by assumption that $m_0(i) \neq \tilde{m}_0(i) \pmod{p_{\ell_*}}$ for some $i \in [\frac{d}{T}]$, we have that $\tilde{m}_{\ell_*}(i) \neq m_{\ell_*}(i)$.

Finally, note that we may assume that $\tilde{m}_{\ell_*}$ is a codeword of $C_\star^{(\ell_*)}$, since otherwise Alice clearly rejects on Step 2c. Moreover, by (4) we also have that m_{ℓ^*} is a codeword of $C_\star^{(\ell_*)}$. Since $C_\star^{(\ell_*)}$ has relative distance at least 0.1, and by assumption that $\tilde{m}_{\ell_*} \neq m_{\ell_*}$, we have that $\tilde{m}_{\ell^*}$ and m_{ℓ^*} differ on at least a 0.1-fraction of their entries, and so with probability at least 0.1 over the choice of i_* it holds that $\tilde{m}_{\ell_*}(i_*) \neq m_{\ell_*}(i_*)$. In what follows, assume that this event holds as well.

By assumption that $\tilde{m}_{\ell_*}(i_*) \neq m_{\ell_*}(i_*)$, there exist $\alpha, \beta \in \{0,1\}$ so that $\tilde{m}_{\ell_*,\alpha,\beta}(i_*) \neq m_{\ell_*,\alpha,\beta}(i_*)$. Consequently, with probability at least 0.25 over the choice of α_*, β_*, it holds that $\tilde{m}_{\ell_*,\alpha_*,\beta_*}(i_*) \neq m_{\ell_*,\alpha_*,\beta_*}(i_*)$. But assuming that this latter event holds, we have that

$$\langle a', b' \rangle = \left\langle \mathrm{row}_{i_*}(a^{(\ell_*,\alpha_*)}), \mathrm{row}_{i_*}(b^{(\ell_*,\beta_*)}) \right\rangle = m_{\ell_*,\alpha_*,\beta_*}(i_*) \neq \tilde{m}_{\ell_*,\alpha_*,\beta_*}(i_*) = \sigma'.$$

We conclude that in the case that $\langle a, b \rangle \neq \sigma$, for any Merlin's message, we have that Alice either rejects or $\langle a', b' \rangle \neq \sigma'$ with probability at least $0.9 \cdot 0.1 \cdot 0.25 \geq 0.02$ over the choice of ℓ_*, i_*, α_*, and β_*. So $\langle a', b' \rangle = \sigma'$ with probability at most 0.98 over the choice of ℓ_*, i_*, α_*, and β_*.

4 From IP to Apx-Max-IP

In this section we use Theorem 3.1 from the previous section which gives an MA protocol for IP over a small alphabet to give a fine-grained reduction from IP to Apx-Max-IP with a tighter polynomial dependence of the approximation parameter δ on the running time parameter ϵ.

▶ **Lemma 4.1** (From IP to Apx-Max-IP). *The following holds for any $\epsilon > 0$ and integer $c \geq 1$. Suppose that $\mathrm{IP}_{N,d}$ cannot be solved in time $N^{2-\epsilon}$ for $d = c \log N$. Then there exists d' such that δ-Apx-Max-$\mathrm{IP}_{N,d'}$ cannot be solved in time $N^{2-2\epsilon}$ for $\delta = \tilde{\Theta}((\frac{\epsilon}{c})^2)$.*

To prove the above lemma, we shall use the following encoding lemma from [9], which can be used to turn the (non-binary) vectors a', b' from the protocol given in Theorem 3.1 into (binary) vectors, whose inner product exhibits a gap.

▶ **Lemma 4.2** (Encoding Lemma, [9]). *For any non-negative integers T and q, there exist mappings $g, h : \{0,1,\ldots,q\}^T \times \{0,1,\ldots,T \cdot q^2\} \to \{0,1\}^{O(T^2 q^4)}$ and an integer $\Gamma \leq O(T^2 \cdot q^4)$, so that for any $a, b \in \{0,1,\ldots,q\}^T$ and $\sigma \in \{0,1,\ldots,T \cdot q^2\}$:*

- *If $\langle a, b \rangle = \sigma \Rightarrow \langle g(a,\sigma), h(b,\sigma) \rangle = \Gamma$.*
- *If $\langle a, b \rangle \neq \sigma \Rightarrow \langle g(a,\sigma), h(b,\sigma) \rangle < \Gamma$.*

Moreover, g, h can be computed in time $\mathrm{poly}(T, q)$.

The reduction. We shall show a reduction from IP to many instances of Apx-Max-IP, based on our MA protocol for IP over a small alphabet given in Theorem 3.1, and the above encoding Lemma 4.2.

Let $A, B \subseteq \{0,1\}^d$ and $\sigma \in \{0,1,\ldots,d\}$ be an instance of $\mathrm{IP}_{N,d}$. Let T be a sufficiently large integer, to be determined later on, and let π be the protocol guaranteed by Theorem 3.1 for T, $q = O(\log^2 T)$, and d (without loss of generality assume that T divides d). Let g, h, and Γ be the mappings and the integer guaranteed for T and q by Lemma 4.2.

Let $\mathrm{Rej} \subseteq \{0,1\}^L$ denote the subset of Merlin's messages $m \in \{0,1\}^L$ in π on which Alice rejects on input σ. For $b \in B$ and $r \in \{0,1\}^R$, let b'_r denote the string output by Bob in the protocol π on input b and randomness string r. Similarly, for $a \in A$ and $r \in \{0,1\}^R$, let a'_r denote the string output by Alice in the protocol π on input a and randomness string r. For $m \in \{0,1\}^L \setminus \mathrm{Rej}$ and $r \in \{0,1\}^R$, let $\sigma'_{m,r}$ denote the integer output by Alice on Merlin's message m and randomness string r.

For any $m \in \{0,1\}^L \setminus \text{Rej}$, we create an instance A_m, B_m of Apx-Max-IP$_{N,d'}$, given by

$$A_m := \{(g(a'_r, \sigma'_{m,r}))_{r\in\{0,1\}^R} \mid a \in A\},$$

and

$$B_m := \{(h(b'_r, \sigma'_{m,r}))_{r\in\{0,1\}^R} \mid b \in B\},$$

where

$$d' = O(2^R \cdot T^2 \cdot q^4).$$

Let $\delta := \frac{0.01\cdot 2^R}{d'}$. Given an algorithm $\mathcal{A}$ for δ-Apx-Max-IP$_{N,d'}$, we show an algorithm $\mathcal{A}'$ for IP$_{N,d}$: Given an instance A, B, σ for IP$_{N,d}$, the algorithm $\mathcal{A}'$ generates all instances A_m, B_m for $m \in \{0,1\}^L \setminus \text{Rej}$, and runs $\mathcal{A}$ on any of these instances. If on any of the instances the algorithm $\mathcal{A}$ outputs a value at least $2^R \cdot (\Gamma - 0.01)$ then the algorithm $\mathcal{A}'$ accepts, otherwise it rejects.

Correctness. Correctness relies on the following claim.

▷ Claim.

- If there exists $(a, b) \in A \times B$ so that $\langle a, b\rangle = \sigma$, then there exist $m \in \{0,1\}^L \setminus \text{Rej}$ and $(a'', b'') \in A_m \times B_m$ so that $\langle a'', b''\rangle = 2^R \cdot \Gamma$.
- If $\langle a, b\rangle \neq \sigma$ for any $(a, b) \in A \times B$, then $\langle a'', b''\rangle \leq 2^R \cdot (\Gamma - 0.02)$ for any $m \in \{0,1\}^L \setminus \text{Rej}$ and $(a'', b'') \in A_m \times B_m$.

Proof. For the first item, suppose that there exists $(a, b) \in A \times B$ so that $\langle a, b\rangle = \sigma$. Let m be Merlin's message in the protocol π on inputs a, b, and σ, and let $(a'', b'') \in A_m \times B_m$ be given by $a'' = (g(a'_r, \sigma'_{m,r}))_{r\in\{0,1\}^R}$ and $b'' = (h(b'_r, \sigma'_{m,r}))_{r\in\{0,1\}^R}$. By the completeness property of π, we have that $m \notin \text{Rej}$, and $\langle a'_r, b'_r\rangle = \sigma'_{m,r}$ for any $r \in \{0,1\}^R$. Consequently, by Lemma 4.2, $\left\langle g(a'_r, \sigma'_{m,r}), h(b'_r, \sigma'_{m,r})\right\rangle = \Gamma$ for any $r \in \{0,1\}^R$. But this implies in turn that

$$\langle a'', b''\rangle = \sum_{r\in\{0,1\}^R} \left\langle g(a'_r, \sigma'_{m,r}), h(b'_r, \sigma'_{m,r})\right\rangle = 2^R \cdot \Gamma.$$

For the second item, suppose that $\langle a, b\rangle \neq \sigma$ for any $(a, b) \in A \times B$. Fix $m \in \{0,1\}^L \setminus \text{Rej}$ and $(a'', b'') \in A_m \times B_m$. Then by construction, $a'' = (g(a'_r, \sigma'_{m,r}))_{r\in\{0,1\}^R}$ and $b'' = (h(b'_r, \sigma'_{m,r}))_{r\in\{0,1\}^R}$. By the soundness property of π, for at least a 0.02-fraction of the randomness strings $r \in \{0,1\}^R$, it holds that $\langle a'_r, b'_r\rangle \neq \sigma'_{m,r}$. Consequently, by Lemma 4.2 for at least a 0.02-fraction of the randomness strings $r \in \{0,1\}^R$, it holds that $\left\langle g(a'_r, \sigma'_{m,r}), h(b'_r, \sigma'_{m,r})\right\rangle \leq \Gamma - 1$. But this implies in turn that

$$\langle a'', b''\rangle = \sum_{r\in\{0,1\}^R} \left\langle g(a'_r, \sigma'_{m,r}), h(b'_r, \sigma'_{m,r})\right\rangle \leq 0.98\cdot 2^R\cdot\Gamma + 0.02\cdot 2^R\cdot(\Gamma-1) = 2^R\cdot(\Gamma-0.02).\triangleleft$$

Now, if there exists $(a, b) \in A \times B$ so that $\langle a, b\rangle = \sigma$, then by the above claim there exists $m \in \{0,1\}^L \setminus \text{Rej}$ so that $\max_{a''\in A_m, b''\in B_m} \langle a'', b''\rangle \geq 2^R \cdot \Gamma$. Consequently, the algorithm $\mathcal{A}$ will output a value greater than $2^R \cdot \Gamma - \delta \cdot d' = 2^R \cdot (\Gamma - 0.01)$ on the instance A_m, B_m, and so the algorithm $\mathcal{A}'$ will accept.

If on the other hand, $\langle a, b\rangle \neq \sigma$ for any $(a, b) \in A \times B$, then by the above claim $\max_{a''\in A_m, b''\in B_m} \langle a'', b''\rangle \leq 2^R \cdot (\Gamma - 0.02)$ for any $m \in \{0,1\}^L \setminus \text{Rej}$. Consequently, the algorithm $\mathcal{A}$ will output a value at most $2^R \cdot (\Gamma - 0.02) < 2^R \cdot (\Gamma - 0.01)$ on any of the instances, an so the algorithm $\mathcal{A}'$ will reject.

Running time. Suppose that the algorithm $\mathcal{A}$ for δ-Apx-Max-IP$_{N,d'}$ runs in time $N^{2-2\epsilon}$, we shall show that for an appropriate choice of T, the running time of the algorithm $\mathcal{A}'$ for IP$_{N,d}$ is at most $N^{2-\epsilon}$.

The algorithm $\mathcal{A}'$ enumerates over all possible Merlin's messages $m \in \{0,1\}^L$, and for each such message checks whether Alice rejects m in π, which takes time poly(d), and if she does not reject, it generates the instance A_m, B_m which takes time $N \cdot 2^R \cdot \text{poly}(d) \cdot \text{poly}(T,q) \leq N \cdot \text{poly}(d)$, and runs the algorithm $\mathcal{A}$ on A_m, B_m which takes time $N^{2-2\epsilon}$.

Hence the total running time of the algorithm $\mathcal{A}'$ is at most

$$\begin{aligned} 2^L \cdot (N \cdot \text{poly}(d) + N^{2-2\epsilon}) &\leq 2^{O(\frac{d}{T} \cdot \log^2 T)} \cdot (N \cdot \text{poly}(d) + N^{2-2\epsilon}) \\ &= 2^{O(\frac{c \log N}{T} \cdot \log^2 T)} \cdot (N \cdot \text{poly}(c \log N) + N^{2-2\epsilon}). \\ &\leq 2^{O(\frac{c \log N}{T} \cdot \log^2 T)} \cdot N^{2-2\epsilon}. \end{aligned}$$

Finally, it can be verified that the latter expression is at most $N^{2-\epsilon}$ for choice of $T = \tilde{\Theta}(c/\epsilon)$ which divides d.

Approximation parameter. By choice of $\delta = \frac{0.01 \cdot 2^R}{d'}$, $d' = O(2^R \cdot T^2 \cdot q^4)$, $T = \tilde{\Theta}(c/\epsilon)$, and $q = O(\log^2 T)$, we have that

$$\delta = \Theta\left(\frac{1}{T^2 q^4}\right) = \tilde{\Theta}\left(\left(\frac{\epsilon}{c}\right)^2\right).$$

References

1. Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Trans. Comput. Theory*, 1(1):2:1–2:54, 2009.
2. Amir Abboud, Aviad Rubinstein, and Ryan Williams. Distributed pcp theorems for hardness of approximation in p. In *FOCS*, pages 25–36. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.12.
3. Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In Piotr Indyk, editor, *SODA*, pages 218–230. SIAM, 2015.
4. Josh Alman, Timothy Chan, and Ryan Williams. Polynomial representations of threshold functions and algorithmic applications. In *FOCS*, pages 467–476. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.57.
5. Josh Alman and Ryan Williams. Probabilistic polynomials and hamming nearest neighbors. In Venkatesan Guruswami, editor, *FOCS*, pages 136–150. IEEE Computer Society, 2015. URL: http://www.computer.org/csdl/proceedings/focs/2015/8191/00/index.html.
6. Ethem Alpaydin. *Introduction to Machine Learning*. MIT Press, Cambridge, Massachusetts, 2014.
7. Timothy M. Chan and R. Ryan Williams. Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. *ACM Trans. Algorithms*, 17(1):2:1–2:14, 2021.
8. Lijie Chen. On the hardness of approximate and exact (bichromatic) maximum inner product. *Theory of Computing*, 16:1–50, 2020.
9. Lijie Chen and Ryan Williams. An equivalence class for orthogonal vectors. In *SODA*, pages 21–40. SIAM, 2019.
10. Andreas Emil Feldmann, C. S. Karthik, Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020.
11. Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and Ryan Williams. Completeness for first-order properties on sparse structures with algorithmic applications. *ACM Trans. Algorithms*, 15(2):23:1–23:35, 2019.

12 Tomislav Hengl. Finding the right pixel size. *Computers and Geosciences*, 32(9):1283–1298, 2006.

13 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci*, 62(2):367–375, 2001.

14 Samir Khuller and Yossi Matias. A simple randomized sieve algorithm for the closest-pair problem. *Inf. Comput*, 118(1):34–37, April 1995.

15 C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992.

16 M. O. Rabin. *Probabilistic Algorithms*, pages 21–39. Academic Press, NY, 1976.

17 Aviad Rubinstein. Hardness of approximate nearest neighbor search. In *STOC*, pages 1260–1268. ACM, 2018. URL: `http://dl.acm.org/citation.cfm?id=3188745`.

18 Aviad Rubinstein and Virginia Vassilevska Williams. Seth vs approximation. *SIGACT News*, 50(4):57–76, 2019.

19 Kenneth W. Shum, Ilia Aleshnikov, P. Vijay Kumar, Henning Stichtenoth, and Vinay Deolalikar. A low-complexity algorithm for the construction of algebraic-geometric codes better than the gilbert-varshamov bound. *IEEE Transactions on Information Theory*, 47(6):2225–2241, 2001. `doi:10.1109/18.945244`.

20 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci*, 348(2-3):357–365, 2005.

21 Virginia Vassilevska Williams. Some open problems in fine-grained complexity. *SIGACT News*, 49(4):29–35, 2018.

22 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018.

23 Raymond Chi-Wing Wong, Yufei Tao, Ada Wai-Chee Fu, and Xiaokui Xiao. On efficient spatial matching. In *VLDB*, pages 579–590. ACM, 2007. URL: `http://www.vldb.org/conf/2007/papers/research/p579-wong.pdf`.

24 Charles Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers*, 20(1):68–86, January 1971.

A Applications

In this section we show a couple of consequences of Lemma 4.1 to obtaining tighter fine-grained hardness of approximation results based on the IP assumption.

Closest pair in Hamming metric. The following reduction from Max-IP to CP in the Hamming metric Δ is implicit in [17].

▶ **Lemma A.1** (From Apx-Max-IP to Apx-CP$_\Delta$, [17]). *Suppose that δ-Apx-Max-IP$_{N,d}$ cannot be solved in time $N^{2-\epsilon}$. Then δ'-Apx-CP$_{N,d',\Delta}$ cannot be solved in time $N^{2-2\epsilon}$ for $\delta' = \frac{\delta}{2}$.*

The above lemma and Lemma 4.1 readily imply the following.

▶ **Corollary A.2** (From IP to Apx-CP$_\Delta$). *Suppose that IP$_{N,d}$ cannot be solved in time $N^{2-\epsilon}$ for $d = c \log N$. Then δ-Apx-CP$_{N,d',\Delta}$ cannot be solved in time $N^{2-2\epsilon}$ for $\delta = \tilde{\Theta}((\frac{\epsilon}{c})^2)$.*

In contrast, it is known how to obtain an $(1+\delta)$-approximation for CP over the Hamming metric in time $N^{2-\epsilon}$ for $\delta = \tilde{\Theta}(\epsilon^3)$ [4].

Closest pair in ℓ_p metric. The following reduction from CP in the Hamming metric to CP in the ℓ_p metric is also implicit in [17].

▶ **Lemma A.3** (From Apx-CP$_\Delta$ to Apx-CP$_{\ell_p}$, [17]). *Suppose that δ-Apx-CP$_{N,d,\Delta}$ cannot be solved in time $N^{2-\epsilon}$. Then for any $p > 0$, δ'-Apx-CP$_{N,d,\ell_p}$ cannot be solved in time $N^{2-2\epsilon}$ for $\delta' = \Theta_p(\delta)$.*

The following corollary is a consequence of the above lemma and Corollary A.2, and implies Theorem 1.1.

▶ **Corollary A.4** (From IP to Apx-CP$_{\ell_p}$). *Suppose that* IP$_{N,d}$ *cannot be solved in time $N^{2-\epsilon}$ for $d = c \log N$. Then for any $p > 0$, δ-Apx-CP$_{N,d',\ell_p}$ cannot be solved in time $N^{2-2\epsilon}$ for $\delta = \tilde{\Theta}_p((\frac{\epsilon}{c})^2)$.*

In contrast, it is known how to obtain an $(1+\delta)$-approximation for CP over the ℓ_p metric in time $N^{2-\epsilon}$ for $\delta = \tilde{O}(\epsilon^3)$ and $p \in \{1, 2\}$ [4].

Closest pair in edit distance metric. For $a, b \in \Sigma^d$, we let $ED(a, b)$ denote their edit distance which is the minimum number of character deletion, insertion, and substitution operations needed to transform a into b. The following Lemma is also implicit in [17].

▶ **Lemma A.5** (From Apx-CP$_\Delta$ to Apx-CP$_{ED}$, [17]). *Suppose that δ-Apx-CP$_{N,d,\Delta}$ cannot be solved in time $N^{2-\epsilon}$. Then δ'-Apx-CP$_{N,d',ED}$ cannot be solved in time $N^{2-2\epsilon}$ for $\delta' = \Theta(\delta)$.*

The above lemma and Corollary A.2 imply the following corollary.

▶ **Corollary A.6** (From IP to Apx-CP$_{ED}$). *Suppose that* IP$_{N,d}$ *cannot be solved in time $N^{2-\epsilon}$ for $d = c \log N$. Then δ-Apx-CP$_{N,d',ED}$ cannot be solved in time $N^{2-2\epsilon}$ for $\delta' = \tilde{\Theta}((\frac{\epsilon}{c})^2)$.*

To the best of our knowledge, it is not known how to solve $(1+\delta)$-Apx-CP$_{\text{ED}}$ in sub-quadratic time.

▶ Remark (Apx-Min-IP and Furthest-Pair). It is not hard to show (see e.g., [9], Lemma 5.3) that there is a simple linear-time reduction from δ-Apx-Max-IP$_{N,d}$ to δ-Apx-Min-IP$_{N,d}$ (and vice versa), and so the same result as in Lemma 4.1 also holds for δ-Apx-Min-IP$_{N,d}$ (where the goal is to output a number in $[M, M + \delta \cdot d]$, where $M := \min_{a \in A, b \in B} \langle a, b \rangle$).

Using Apx-Min-IP as the starting point for the reductions cited above instead of Apx-Max-IP implies the same results as in Corollaries A.2, A.4, and A.6 for Furthest Pair (where the goal is to output a number in $[(1-\delta)M, M]$, where $M := \max_{a \in A, b \in B} dist(a, b)$).

Data structure setting. Our results extend to the data structure setting.

▶ **Definition A.7** (Approximate Nearest Neighbor (Apx-NN)). *Let* dist : $\{0,1\}^d \times \{0,1\}^d \to \mathbb{R}^+$ *be a distance function, and let $\delta > 0$ be a parameter. In the Approximate Nearest Neighbor δ-Apx-NN$_{N,d,dist}$ problem, given a set $A \subseteq \{0,1\}^d$ of cardinality N, the goal is to pre-process the set, so that given a vector $b \in \{0,1\}^d$ it is possible to quickly output a number in $[M, (1+\delta)M]$, where $M := \min_{a \in A}$* dist(a, b).

It is known that Apx-CP can be reduced to Apx-NN [22] (see also proof of Corollary 1.4 in [2]).

▶ **Lemma A.8** (From Apx-CP$_\Delta$ to Apx-NN, [22]). *Let* dist : $\{0,1\}^d \times \{0,1\}^d \to \mathbb{R}^+$ *be a distance function. Suppose that δ-Apx-CP$_{N,d,dist}$ cannot be solved in $N^{2-\epsilon}$ time. Then for any $r > 0$, δ-Apx-NN$_{N,d,dist}$ cannot be solved with N^r preprocessing time and $N^{1-2r\epsilon}$ time.*

▶ **Corollary A.9** (From IP to Apx-NN). *Suppose that* IP$_{N,d}$ *cannot be solved in time $N^{2-\epsilon}$ for $d = c \log N$. Then for any distance function* dist $\in \{\Delta, \ell_p, \text{ED}\}$ *and $r > 0$, δ-Apx-NN$_{N,d',dist}$ cannot be solved with N^r preprocessing time and $N^{1-3r\epsilon}$ query time for $\delta = \tilde{\Theta}((\frac{\epsilon}{c})^2)$.*

Approximation Schemes for Geometric Knapsack for Packing Spheres and Fat Objects

Pritam Acharya
Department of Mathematics, Indian Institute of Science Education and Research Pune, India

Sujoy Bhore
Department of Computer Science and Engineering, Indian Institute of Technology Bombay, India

Aaryan Gupta
Department of Computer Science and Engineering, Indian Institute of Technology Bombay, India

Arindam Khan
Department of Computer Science and Automation, Indian Institute of Science Bengaluru, India

Bratin Mondal
Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India

Andreas Wiese
Department of Mathematics, Technical University of Munich, Germany

Abstract

We study the geometric knapsack problem in which we are given a set of d-dimensional objects (each with associated profits) and the goal is to find the maximum profit subset that can be packed non-overlappingly into a given d-dimensional (unit hypercube) knapsack. Even if $d = 2$ and all input objects are disks, this problem is known to be NP-hard [Demaine, Fekete, Lang, 2010]. In this paper, we give polynomial time $(1 + \varepsilon)$-approximation algorithms for the following types of input objects in any constant dimension d:

- disks and hyperspheres,
- a class of fat convex polygons that generalizes regular k-gons for $k \geq 5$ (formally, polygons with a constant number of edges, whose lengths are in a bounded range, and in which each angle is strictly larger than $\pi/2$),
- arbitrary fat convex objects that are sufficiently small compared to the knapsack.

We remark that in our PTAS for disks and hyperspheres, we output the computed set of objects, but for a $O_\varepsilon(1)$ of them we determine their coordinates only up to an exponentially small error. However, it is not clear whether there always exists a $(1 + \varepsilon)$-approximate solution that uses only rational coordinates for the disks' centers. We leave this as an open problem which is related to well-studied geometric questions in the realm of circle packing.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Approximation Algorithms, Polygon Packing, Circle Packing, Sphere Packing, Geometric Knapsack, Resource Augmentation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.8

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2404.03981

Funding Arindam Khan's research is supported in part by Google India Research Award, SERB Core Research Grant (CRG/2022/001176) on "Optimization under Intractability and Uncertainty", and the Walmart Center for Tech Excellence at IISc (CSR Grant WMGT-23-0001). A part of this work was done when Aaryan Gupta, Bratin Mondal, and Pritam Acharya were undergraduate interns at Indian Institute of Science, supported by Arindam Khan's Pratiksha Trust Young Investigator Award.

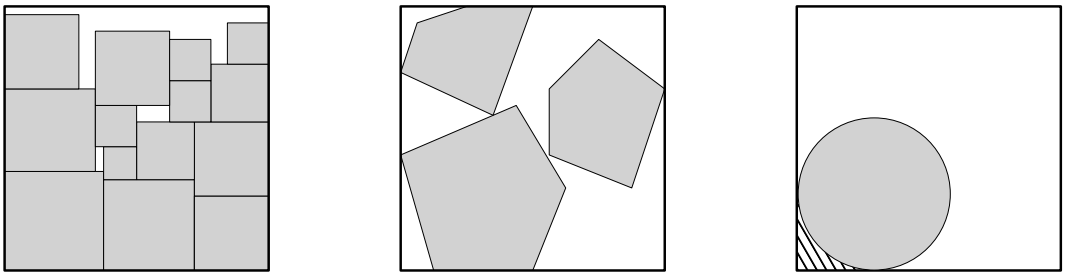


Figure 1 Left: The squares are stacked compactly inside the knapsack. Middle: The pentagons cannot be stacked as tightly inside the knapsack as the squares. Right: The space in the corner (striped area) cannot be covered by any large circle.

1 Introduction

One of the cornerstones of geometry is the problem of packing circles and spheres into a container, e.g., a square or a hypercube. It dates back to the 17th century when Kepler conjectured his famous bound on the average density of any packing of spheres in the three-dimensional Euclidean space [27]. The problem has been investigated, for example, by Lagrange [11] who solved it in the setting of two dimensions, by Hales and Ferguson [24] who proved Kepler's original conjecture, and by Viazovska [39] who studied the problem in dimension 8 and was awarded the Fields medal in 2022 for her work.

A natural corresponding optimization question is the *geometric knapsack problem*, where we are given a set of d-dimensional objects for some constant $d \in \mathbb{N}$, e.g., circles or (hyper-)spheres, but possibly also other shapes (like squares, pentagons, hexagons, etc. for the case of $d = 2$) with each of them having a given profit. The goal is to find the subset of maximum total profit that can be packed non-overlappingly into a given square or (hyper-)cube. In this work, we consider the translations of the objects but do not allow rotations.

Geometric knapsack is a natural mathematical problem and it is well-motivated by practical applications in several areas, including radio tower placement [38], origami design [30], cylinder pallet assembly [8, 17], tree plantation [38], cutting industry [38], bundling tubes or cables [40], layout of control panels [8], or design of digital modulation schemes [36].

The problem is known to be NP-hard, already for $d = 2$ and if all input objects are axis-aligned squares or disks [14, 2]. This motivates designing approximation algorithms for it. For hypercubes in any constant dimension d, there is a polynomial time $(1+\varepsilon)$-approximation algorithm known for any constant $\varepsilon > 0$ [26], i.e., a polynomial time approximation scheme (PTAS). Thus, this is the best possible approximation guarantee, unless P=NP.

However, for other classes of (fat) objects, the best known results either have approximation ratios that are (far) from their respective lower bounds or they require resource augmentation, i.e., increase in the size of the given knapsack. One intuitive reason for this is that axis-aligned squares and cubes can be stacked nicely without wasting space and the resulting coordinates are well-behaved, while for more general shapes this might not be the case, see Figure 1. For circles, the best known result is a $(3 + \varepsilon)$-approximation [35] but the best known lower bound is only NP-hardness. Still, the membership in NP is wide open for the following question: given a set of n circles of $O(1)$ number of different sizes, decide whether they can be packed into a unit square. It is also open whether packing circles into a square knapsack is $\exists\mathbb{R}$-complete or not [1]. Even for n unit circles. we do not know the exact value of the smallest size squares that can pack them. See [37] for the current status of upper and lower bounds for $n \leq 1000$.

There is a PTAS in any constant dimension d for this case, but it requires resource augmentation [9]. For triangles, there is a $O(1)$-approximation algorithm (assuming it is allowed to rotate the triangles arbitrarily) whose precise approximation ratio is not explicitly specified [33]. Also for this case, it is still possible that there is a PTAS. On the other hand, there are settings of geometric knapsack that do *not* admit a PTAS, e.g., axis-parallel cuboids in three dimensions [12].

Furthermore, in practical applications (e.g., loading cargo into a truck or cutting pieces out of raw material like cloth or metal) the objects do not necessarily all have the same shape. For example, Bennell and Oliveira [6] consider a mix of different shapes of objects (their primary objects are circles, rectangles, regular polygons, and convex polygons). However, the previous papers in the theoretical literature for geometric knapsack mostly assume that all input objects are of the same type, e.g., only squares, only circles, or only rectangles, etc. Thus, from a theoretical point of view, it is interesting to see how the problem behaves when the input objects might be of different types.

This raises the following natural question that we study in this paper:

What are the best approximation ratios we can achieve for the geometric knapsack problem, depending on the type of the input objects? For which type of objects does a PTAS exist?

1.1 Our contribution

In this paper, we present a polynomial time $(1+\varepsilon)$-approximation algorithm for geometric knapsack problem for packing d-dimensional spheres into d-dimensional hypercube knapsack, for any constant dimension $d \geq 2$. For spheres, there is a complication that possibly any (near-)optimal packing for a given instance require irrational coordinates. Therefore, our output consists of a set of spheres that can be packed non-overlappingly inside the given knapsack and whose profit is at least $(1+\varepsilon)^{-1}w(\mathrm{OPT})$, where $w(\mathrm{OPT})$ denotes the profit of the optimal solution OPT. Moreover, for all but at most $O_\varepsilon(1)$ spheres, our algorithm outputs the precise (rational) coordinates of the packing. [1] For the other $O_\varepsilon(1)$ spheres it outputs them up to an exponentially small error in each dimension. We remark that there are related packing problems for which it is known that irrational coordinates are sometimes necessary and that computing them is $\exists\mathbb{R}$-complete (and hence possibly even harder than NP-hardness) [1]. On the other hand, if we knew that there always exists a $(1+\varepsilon)$-approximate solution in which all coordinates are rational with only a polynomial number of bits, our algorithm would find such coordinates in polynomial time. We stress that our returned set of spheres is always guaranteed to fit into the given knapsack with appropriate (possibly irrational) coordinates but *without* resource augmentation.

Our second result is a polynomial time $(1+\varepsilon)$-approximation algorithm for the geometric knapsack problem for wide classes of convex geometric polygons. Our first result is a PTAS for a class of fat convex polygons which generalizes pentagons, hexagons, and regular k-gons for constant $k > 4$ (see Figure 2). Formally, we require for each polygon that the angle between any two adjacent edges is at least $\pi/2 + \delta$ for some constant $\delta > 0$ and that each polygon has a constant number of edges with similar lengths (up to a constant factor). Note that in contrast to many prior results, we allow that each input object has a different shape, e.g., with a different number of edges, different angles formed by them, and a different orientation. Also, the polygons may differ arbitrarily in size.

[1] The notation $O_\varepsilon(f(n))$ means that the implicit constant hidden in big-O notation can depend on ε.

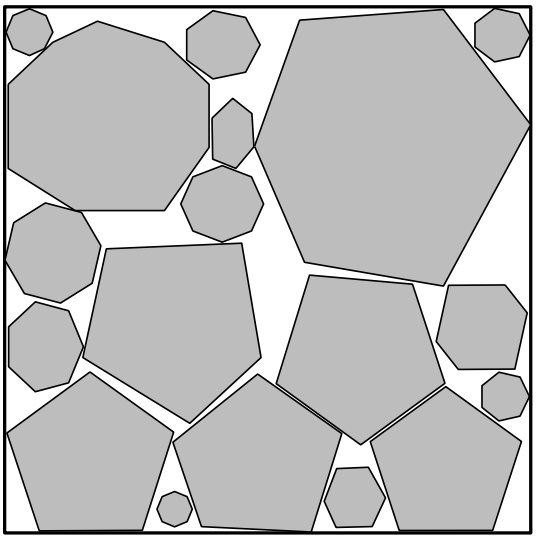


Figure 2 Packing of fat convex polygons in a knapsack.

If each input object is sufficiently small compared to the knapsack, we obtain even a polynomial time $(1+\varepsilon)$-approximation for *arbitrary* fat convex objects in any constant dimension d. We remark that for other packing problems like one-dimensional Knapsack or Bin Packing, near-optimal solutions can easily be achieved via greedy algorithms if the input objects are sufficiently small. Even for sufficiently small d-dimensional axis-aligned hypercuboids, it is known that simple algorithms like NFDH [13, 4] has negligible wasted space. However, for other geometric objects this is much harder since we might not be able to place the input objects compactly without wasting space. For example, classical result by Thue [11] showed that one can pack at most $\frac{\pi}{2\sqrt{3}} \approx 0.9069$ fraction of the total area, even in the case of packing of unit circles. Furthermore, for circles and other similar convex objects, irrational coordinates may arise in the packing and the optimal solution may use a very complicated packing to minimize the wasted space.

1.2 Our techniques

We discuss now the techniques of our results, starting with our PTAS for spheres. To compute our packing, we first enumerate all the large spheres in the optimal solution, i.e., the spheres whose radius is at least a constant fraction of the side length of the knapsack. Also, we guess their placement up to a polynomially small error, which yields a small range of possible placements for each of them. Note that we cannot guess these coordinates precisely, since we cannot even exclude that they are irrational. However, we guarantee that such coordinates *exist*, by solving a system of polynomial equations *exactly* in polynomial time.

Next, we want to place small spheres into the remaining part of the knapsack. Unfortunately, we do not know precisely which part of the knapsack is available for them since we do not know the precise coordinates of the large spheres. Thus, there is some area of the knapsack that is *maybe* used by the large spheres in our packing; however, potentially, the optimal solution uses it for placing small spheres. Our key insight is that this area is small compared to the area that is for sure *not* used by large spheres in the optimal solution. Using the fact that objects are spheres, we show that some area in each corner of the knapsack cannot be covered by any large sphere in *any* solution and whose size is at least a constant

fraction of the knapsack (see the bottom-left empty corner in Figure 1). We use this area to compensate the fact that we do not know the precise coordinates of our large spheres and we waste space because of this.

When we select and pack the small spheres, we define a constant number of (small) identical knapsacks that fit into the given knapsack together with the large spheres and into which we place our small spheres. For the remaining task of placing the small spheres, we argue that it is sufficient to have an algorithm that uses resource augmentation, i.e., that increases the size of each knapsack by a factor of $1 + \varepsilon$ (in each dimension). Thus, on a high level, we reduce the problem of packing arbitrary spheres into *one* knapsack to the problem of packing small spheres into a *constant number* of knapsacks *with resource augmentation*.

This remaining problem can be solved via an algorithm in [9]; however, we present a more general routine that works even for arbitrary convex fat objects. Also, it is arguably simpler than the corresponding algorithm in [9]. On a high level, we prove that there is a well-structured solution based on a hierarchical decomposition of the knapsacks into grid cells. The grid cells are partitioned such that each placed object P is contained in a constant number of grid cells whose size is comparable to P. Importantly, these grid cells are used *exclusively* by P and not by any other placed object (not even partially). This allows us to devise a dynamic program (DP) that computes the optimal structured packing of this type. Our DP has a subproblem for each combination of a level (corresponding to a size range of the input objects) and a number of available grid cells corresponding to this level. Given such a subproblem, it suffices to enumerate a polynomial number of possibilities for selecting and placing objects of this level, which reduces the given subproblem to a subproblem corresponding to the next level. This DP might have applications in other related packing problems.

In our algorithm for fat convex polygons (with the properties described above), we extend our algorithm for spheres as follows. For the guessed large polygons, we compute their coordinates *exactly* in polynomial time. Here, we use the (known) fact that there exists a placement for them that corresponds to an extreme point solution of a suitable linear program, which has rational coordinates. Then, intuitively we use the condition for the polygons' angles to ensure that the large objects leave a certain area of the knapsack empty. We use this empty area in a similar way as in the setting of circles. Again, we place the small objects into a constant number of knapsacks under resource augmentation, using our new subroutine described above.

If all input objects are sufficiently small compared to the size of the knapsack (formally, we assume that each of them fits in a smaller knapsack with side length $\Theta(\varepsilon)$) there are no large objects and, hence, we can omit the step of enumerating them. In particular, we do not need the conditions of the polygons' edges anymore. Since the input objects are so small, we can show that by losing a factor of $1 + \varepsilon$ in the approximation ratio, we may pretend that we have resource augmentation available. Hence, we can directly call our subroutine for small objects under resource augmentation.

We leave it as an open question to determine whether irrational coordinates are sometimes necessary for optimal or $(1 + \varepsilon)$-approximate solutions for geometric knapsack for spheres. If yes, it would be interesting to determine the best possible approximation ratio one can achieve with rational coordinates only. Note that this question is related to the well-studied problem of determining the size of the smallest knapsack needed to pack a given number of unit circles. For that problem, it is known that for some number of unit circles the smallest knapsack has irrational edge lengths [18]. On the other hand, recall that if rational coordinates with a polynomial number of bits always suffice, our algorithm for spheres can compute the coordinates of *all* returned spheres of our $(1 + \varepsilon)$-approximation algorithm *exactly*.

1.3 Other related work

For geometric knapsack for axis-parallel rectangles (i.e., when $d = 2$), the best known polynomial time algorithm has an approximation ratio of $17/9 + \varepsilon$ [19]. There is a pseudo-polynomial time algorithm with a ratio of $4/3 + \varepsilon$ [20] and a pseudo-polynomial time approximation scheme if we require guillotine-separable packing [28] . If it is allowed to rotate the rectangles by 90 degrees, there is also a polynomial time $(1.5 + \varepsilon)$-approximation algorithm known [19]. Moreover, the problem admits a QPTAS if the input data are quasi-polynomially bounded integers [3]. For the setting of packing circles, there is a PTAS under resource augmentation in one dimension, assuming that the profit of each circle equals its area, due to Lintzmayer, Miyazawa, and Xavier [31]. This was improved to the above mentioned PTAS under resource augmentation in one dimension for spheres with arbitrary profits in any constant dimension d, due to Chagas, Dell'Arriva, and Miyazawa [9]. In addition, there have been many attempts to develop heuristics and other optimization methods on circle packing, see e.g., [38, 25, 32].

A related problem is the geometric bin packing problem in which we want to place a given set of geometric objects into the smallest number of unit size bins. For the settings of squares or (hyper-)cubes [4] or skewed rectangles [29], the problem admits an asymptotic PTAS. In the case of general rectangles, the best known result is an asymptotic 1.405-approximation [5] but an asymptotic PTAS cannot exist unless $\mathsf{P} = \mathsf{NP}$ [12]. Maximum independent set in geometric intersection graphs [10, 34, 21] is another well-studied related problem.

In a recent paper, Abrahamsen, Miltzow, and Seiferth [1] developed a framework to show that for many combinations of allowed pieces, containers, and motions, the resulting packing problem is $\exists\mathbb{R}$-complete. For example, they showed that it is $\exists\mathbb{R}$-complete problem to decide if a set of convex polygons with at most seven corners each can be packed into a square if arbitrary rotations are allowed. However, it is not known if the setting of packing circles into a square knapsack is $\exists\mathbb{R}$-complete.

There is also a large body of work on questions about the optimal packings of unit circles into unit squares or equilateral triangles. We refer to [15, 16, 22, 25] for an overview.

1.4 Organization of this paper

In Section 2, we discuss the PTAS when the input items are sufficiently small fat convex objects. In Section 3, we give our algorithm for spheres. In Section 4, we consider the case of convex polygons. Finally, in Section 5 we end with conclusions. Due to space constraints, many proofs have been omitted. The corresponding lemmas and theorems are marked with $(\star)$. Please see the full version of the paper for the complete proofs.

2 PTAS under Resource Augmentation

In this section we present a PTAS when the input items are fat convex objects, and we are allowed to increase the size of the given knapsack by a factor of $1 + \varepsilon$ in each dimension. Chagas et al. [9] presented a PTAS for circles with resource augmentation in one dimension. Their result is based on a combination of multiple integer programs with variables for different configurations for packing parts of the given knapsack. Our result is arguably simpler and purely based on dynamic programming.

Let P_i be a two-dimensional convex object and let $r_i^{\text{out}}(P_i)$ and $r_i^{\text{in}}(P_i)$ be the radius of the smallest circle containing P_i and the radius of the largest circle contained in P_i, respectively. We will drop P_i when it is clear from the context. We say that P_i is *f-fat* if $r_i^{\text{out}}/r_i^{\text{in}} \leq f$ for some value $f \geq 1$. In the remainder of this section, we prove the following theorem.

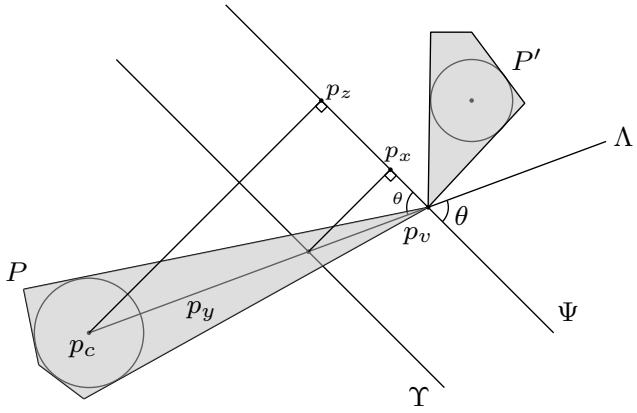


Figure 3 Line Ψ separates the two f-fat and convex objects P and P'. We construct line Υ such that it has intersects no common grid cells with line Ψ. We proceed to shrink the two objects P, P' by a factor of $1+\varepsilon$ such that they cannot intersect the space between lines Υ and Ψ.

▶ **Theorem 1.** *Let $f \geq 1$, $\varepsilon > 0$, and $d \in \mathbb{N}$ be constants. Given a set of d-dimensional f-fat convex input objects, there exists a polynomial time algorithm that can pack a subset of them with a total profit of $w(\mathsf{OPT})$ into a knapsack $K' := [0, 1+\varepsilon]^d$, where $w(\mathsf{OPT})$ is the optimal profit that can be packed into a knapsack $K := [0,1]^d$.*

For simplicity, we first describe our algorithm in the setting where $d = 2$. Given a packing of a set of f-fat objects $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ in our knapsack $K = [0,1] \times [0,1]$, we want to show that there is also a structured packing of these objects into an augmented knapsack $K' = [0, 1+\varepsilon] \times [0, 1+\varepsilon]$, defined via a discrete grid. Let $\delta_{cell} > 0$ be a constant to be defined later such that $1/\delta_{cell} \in \mathbb{N}$. We place a two-dimensional grid inside K' such that each grid cell has an edge length of δ_{cell}. Let $\mathcal{G}$ denote the set of all resulting grid cells. We assume first that each object $P_i \in \mathcal{P}$ is δ_{large}-*large*, meaning that $r^{in}(P_i) \geq \delta_{large}$ for some given constant $\delta_{large} > 0$. We say that our given packing for $\mathcal{P}$ is *discretized* if there is a partition of $\mathcal{G}$ into sets $\{\mathcal{G}_{P_1}, \mathcal{G}_{P_2}, \ldots, \mathcal{G}_{P_n}\}$ such that for each $P_i \in \mathcal{P}$, we have that P_i is contained in the union of the cells in $\mathcal{G}_{P_i}$. Therefore, each object $P_i \in \mathcal{P}$ has "its own" set of grid cells $\mathcal{G}_{P_i}$ that contain P_i and that do not intersect with any other object $P_j \in \mathcal{P} \setminus \{P_i\}$.

We show that for an appropriate choice of δ_{cell}, there is a discretized packing for $\mathcal{P}$ in K', i.e., if we can increase the size of our knapsack K by a factor of $1+\varepsilon$ in each dimension.

▶ **Lemma 2.** *For each $f \geq 1$, $\varepsilon > 0$, and $\delta_{large} > 0$, there is a value $\delta_{cell} > 0$ such that for any set of δ_{large}-large f-fat convex objects $\mathcal{P}$ that can be placed nonoverlappingly inside a knapsack $K = [0,1] \times [0,1]$, there is a discretized packing for $\mathcal{P}$ inside knapsack $K' = [0, 1+\varepsilon] \times [0, 1+\varepsilon]$ based on a grid in which each edge of each grid cell has a length of δ_{cell}.*

Proof. Let $P, P' \in \mathcal{P}$ be two f-fat convex δ_{large}-large objects packed inside the knapsack. Then by *separating hyperplane theorem for convex objects* [7], we know that there is a line Ψ containing a point p_v on the boundary of P, and Ψ separates P from P' (see Figure 3). Now, intuitively, increasing the size of the knapsack by a factor of $1+\varepsilon$ is equivalent to shrinking the objects in $\mathcal{P}$ by a factor of $1+\varepsilon$. So, we want to find the right constraints such that after shrinking P and P' do not share any grid cell. Let the center of the incircle (of radius r^{in}) contained in P be p_c and the line joining p_c and p_v be Λ. Let the foot of the image of the point p_c on line Ψ be the point p_z. Now the length of line segment $\overline{p_c p_z} := |\overline{p_c p_z}| \geq r^{in}$

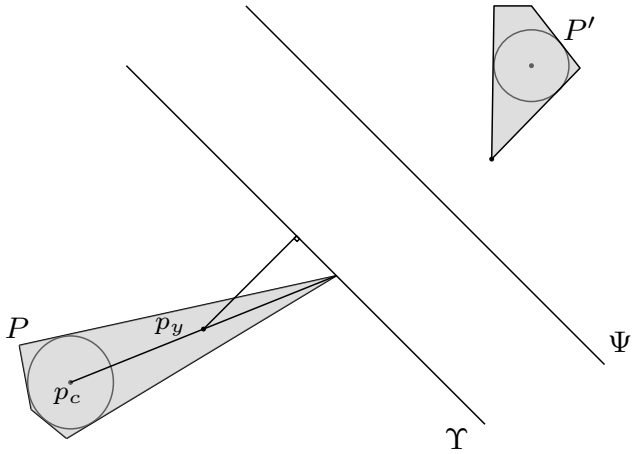


Figure 4 P, P' are shrunk so that they cannot intersect the space between the lines Υ and Ψ.

and $|\overline{p_c p_v}| \leq 2fr^{in}$, due to fatness. Hence, the angle θ between Ψ and Λ is at least $\sin^{-1}(\frac{1}{2f})$. Consider points p_x on Ψ and p_y on Λ such that $|\overline{p_x p_y}| = \sqrt{2}\delta_{cell}$ and the line Υ joining p_x, p_y is parallel to the line joining $p_c p_z$. Then any point on Ψ does not share a gridcell with any point on Υ. Also, $|\overline{p_y p_v}| \leq 2\sqrt{2}f\delta_{cell}$. Now we want to shrink P by $(1+\varepsilon)$ factor keeping p_c at the same position such that the shrunk version of P lies completely within one side of Υ (see Figure 4). After shrinking, $\overline{p_c p_v}$ gets smaller by $\varepsilon|\overline{p_c p_v}| \geq \varepsilon r^{in} \geq \varepsilon\delta_{large}$. Now we choose δ_{cell} such that $\delta_{cell} \leq \frac{\varepsilon}{2\sqrt{2}f} \cdot \delta_{large}$. Thus we satisfy $\varepsilon|\overline{p_c p_v}| \geq \varepsilon\delta_{large} \geq 2\sqrt{2}f\delta_{cell} \geq |\overline{p_y p_v}|$ and this ensures that the shrunk down version of P and P' do not share any grid cell. We assign each polygon to the grid cells that it intersects with. Hence this process leads to a discretization such that the no grid cell is intersected by two polygons. ◀

Next, we argue that there is also a structured packing for fat objects that are not necessarily all (relatively) large. Let $\delta_{large}, \delta_{cell}, \delta_{small} > 0$ be constants to be defined later (they will depend on ε which will denote the amount by which we increase the size of our knapsack). We place now a *hierarchical* two-dimensional grid with multiple levels. We define that the whole knapsack K is one grid cell of level 0 of side length $\delta_{c,0} := 1$. For each level $\ell \geq 1$, we define grid cells whose edges all have a length of $\delta_{c,\ell} := \delta_{cell}\delta_{c,(\ell-1)}$. Recursively, for each $\ell \geq 1$ we partition each grid cell of level $\ell - 1$ into $1/\delta_{cell}^2$ grid cells of level ℓ with side length $\delta_{c,\ell}$ each. Similarly as before, we want that there is a partition of the grid cells such that for each object $P \in \mathcal{P}$ there is a set of grid cells $\mathcal{G}_P$ that contain P and that are disjoint from the grid cells $\mathcal{G}_{P'}$ for each object $P' \in \mathcal{P} \setminus \{P\}$. Also, we want that all grid cells in $\mathcal{G}_P$ are of the same level, that their size is comparable to the size of P, and that the number of grid cells in $\mathcal{G}_P$ is bounded. To ensure this, we group the objects $P \in \mathcal{P}$ according to their respective values $r^{in}(P)$ which we use as a proxy for their sizes. Formally, we define $\delta_{small,0} := 1$ and for each level $\ell \geq 1$ we define $\delta_{large,\ell} := \delta_{large}\delta_{c,(\ell-1)}$ and $\delta_{small,\ell} := \delta_{small}\delta_{c,(\ell-1)}$. For each level ℓ we define

- L_ℓ to be all objects $P \in \mathcal{P}$ with $r^{in}(P) \in (\delta_{large,\ell}, \delta_{small,(\ell-1)}]$; intuitively, they are "large" for level ℓ,
- M_ℓ to be all objects $P \in \mathcal{P}$ with $r^{in}(P) \in (\delta_{small,\ell}, \delta_{large,\ell}]$.

Note that our grid and the sets L_ℓ and M_ℓ depend on the (initial) choice of $\delta_{large}, \delta_{small}$ and δ_{cell}. In the next lemma, we show via a shifting argument that there are choices for these values such that the total area of the objects in $\bigcup_\ell M_\ell$ is very small. This will allow us later to pack them separately via a simple greedy algorithm. In particular, these choices are from a set of $O_\varepsilon(1)$ candidate values D, and thus we will be able to guess them later easily.

▶ **Lemma 3** (⋆). *Let $f \geq 1$. There is a global set D with $|D| \leq O_\varepsilon(1)$ such that for any set of f-fat convex objects $\mathcal{P}$ that can be packed in a knapsack $K = [0,1] \times [0,1]$, there are values $\delta_{large}, \delta_{small}, \delta_{cell} > 0$ that are all contained in D such that for the resulting hierarchical grid and the corresponding sets $\{L_\ell, M_\ell\}_\ell$ we have that the total area of all objects in $\bigcup_\ell M_\ell$ is bounded by ε.*

We generalize now our notion of discretized packings. Intuitively, like before, we require that there is a partition of the grid cells such that for each object $P \in \mathcal{P}$ there is a set of grid cells $\mathcal{G}_P$ that contain P and that are disjoint from the grid cells $\mathcal{G}_{P'}$ for each object $P' \in \mathcal{P} \setminus \{P\}$. Formally, we define that our packing of $\mathcal{P}$ is *discretized* if

- for each level ℓ and for each object $P \in \mathcal{P} \cap L_\ell$ there is a set of $O(1/\delta_{cell}^2)$ grid cells $\mathcal{G}_P$ of level ℓ such that P is contained in $\mathcal{G}_P$, and there is a single grid cell of level $\ell - 1$ that contains all grid cells in $\mathcal{G}_P$, and
- for any two objects $P, P' \in \mathcal{P}$ (not necessarily of the same level) and for any two grid cells $C \in \mathcal{G}_P$ and $C' \in \mathcal{G}'_P$ their relative interiors are disjoint.

We show that by increasing the size of our knapsack by a factor of $1+\varepsilon$, there is a discretized packing for all objects in $\bigcup_\ell L_\ell$. As mentioned above, we will pack the objects in $\bigcup_\ell M_\ell$ separately later.

▶ **Lemma 4.** *Let each $f \geq 1$ and $\varepsilon > 0$. There is a global set D with $|D| \leq O_\varepsilon(1)$ such that for any set of f-fat objects $\mathcal{P}$ that can be placed non-overlappingly inside a knapsack $K = [0,1] \times [0,1]$, there is a choice for the grid with parameters $\delta_{large}, \delta_{small}, \delta_{cell} > 0$ such that all these values are contained in D and there is a discretized packing for $\mathcal{P} \cap (\bigcup_\ell L_\ell)$ inside an (augmented) knapsack $[0, 1 + O(\varepsilon)] \times [0, 1 + O(\varepsilon)]$.*

Proof. We start with the given packing of $\mathcal{P}$ in K and do a sequence of refinements which leads to our discretized packing for $\mathcal{P} \cap \bigcup_\ell L_\ell$. First, we use the increased size of the knapsack to ensure that for each level ℓ and any two objects $P, P' \in L_\ell$, the distance between P and P' is at least $2\delta_{c,\ell}$. Intuitively, increasing the size of the knapsack by a factor of $1+\varepsilon$ is equivalent to shrinking the objects in $\mathcal{P}$ by a factor of $1+\varepsilon$. Therefore, we can achieve this required minimum distance of $2\delta_{c,\ell}$ by choosing δ_{cell} appropriately according to the multiple constraints given in the proof of Lemma 2.

Next, we would like that for each level ℓ, each object in $\mathcal{P} \cap L_\ell$ is contained in a grid cell of level $\ell - 1$. This might not be the case, however, via a shifting argument (giving the grid a random shift) we can argue that this is the case for almost all objects in $\mathcal{P}$. The probability that an object in level ℓ intersects a grid line from level $\ell - 1$ is at most $8\delta_{small,\ell-1}/\delta_{cell,\ell-1} = 8\delta_{small}/\delta_{cell}$. Let this probability be smaller than $\varepsilon^2/2$, leading to a constraint $\delta_{small} \leq \varepsilon^2 \delta_{cell}/16$ on the choice of $\delta_{cell}, \delta_{small}$. Then the total area of intersecting objects must also be smaller than ε^2 as the area of all packed objects can be at most the area of the augmented knapsack. Thus we can easily pack these intersected objects into extra space that we gain via increasing the size of the knapsack (i.e., for a second time).

After this preparation, we process the objects $P \in \mathcal{P}$ level by level and define their corresponding sets $\mathcal{G}_P$, starting with the highest level. Consider a level ℓ. For each object $P \in \mathcal{P}$ of level ℓ we define $\mathcal{G}_P$ to be the set of all grid cells of level ℓ that intersect with P. Due to our minimum distance between any two objects in $\mathcal{P}$ of level ℓ, for any two different objects $P, P' \in \mathcal{P}$ of level ℓ we have that $\mathcal{G}_P \cap \mathcal{G}_{P'} = \emptyset$. Now it could be that a cell $\mathtt{C} \in \mathcal{G}_P$ for some $P \in \mathcal{P}$ intersects not only with P, but also with another object $P' \in \mathcal{P}$ of some deeper level $\ell' > \ell$. We call such a cell $\mathtt{C}$ *problematic*; recall that we wanted the cells in $\mathcal{G}_P$ to be used exclusively by P. Therefore, we move all objects $P' \in \mathcal{P}$ of some level $\ell' > \ell$ that intersect a problematic grid cell in $\mathcal{G}_P$. We pack them into extra space that we gain due to resource augmentation. We do this operation for all levels ℓ.

In the process above, we move objects that intersect problematic grid cells. We need to argue that the total area of these moved objects is small compared to the size of the knapsack and that, therefore, we can pack them into additional space that we gain due to our resource augmentation. In particular, we need to argue this globally, over all levels. The key insight is that if we define a set $\mathcal{G}_P$ for some object $P \in \mathcal{P}$ as above, then each problematic cell $\mathtt{C} \in \mathcal{G}_P$ must intersect the boundary of P and, since P is fat, the number of problematic cells $\mathtt{C} \in \mathcal{G}_P$ is very small compared to the number of cells $\mathtt{C}' \in \mathcal{G}_P$ that are contained in P and, thus, for sure *not* problematic.

By the classical *Barbier's theorem*, we know the perimeter of a convex set P of level ℓ is at most $\pi \cdot \text{diameter}(P) \leq 2\pi r^{out} \leq 2\pi f r^{in}$. A curve of length $\delta_{cell,\ell}$ is bound to be contained inside a circle of radius $\delta_{cell,\ell}$. This implies that this curve can intersect at most 9 grid cells as any circle of radius $\delta_{cell,\ell}$ can be bounded in a 3×3 grid square. Hence, the number of grid cells (of level ℓ) N_1 that the perimeter can intersect is at most $18\pi f r^{in}/\delta_{cell,\ell}$. On the other hand, P completely contains at least grid cells of area $\pi(r^{in} - 2\delta_{cell,\ell})^2$, i.e., the number of such gridcells N_2 is at least $\frac{\pi(r^{in}-2\delta_{cell,\ell})^2}{(\delta_{cell,\ell})^2}$. We need $N_1 \leq \varepsilon N_2$. Equivalently, we want to show, $\delta_{cell,\ell} \leq \frac{\varepsilon}{18f} \cdot \frac{(r^{in}-2\delta_{cell,\ell})^2}{r^{in}}$. For this we impose the condition that $\delta_{large} \geq \frac{72f}{\varepsilon}\delta_{cell}$. Then, $\frac{\varepsilon}{18f} \cdot \frac{(r^{in}-2\delta_{cell,\ell})^2}{r^{in}} \geq \frac{\varepsilon}{18f} \cdot \frac{(r^{in}/2)^2}{r^{in}} \geq \frac{\varepsilon}{18f} \cdot \frac{\delta_{large,\ell}}{4} \geq \delta_{cell,\ell}$.

Using this, we derive a *global* argumentation, stating that the total area of *all* problematic grid cells over all objects of all levels is at most an ε-fraction of the area of the knapsack. Also, if an object P' of some level ℓ' intersects a problematic grid cell $\mathtt{C}$ of some level $\ell < \ell'$, then P' is very small compared to $\mathtt{C}$. Thus, the total area of these objects intersecting a problematic grid cell $\mathtt{C}$ is essentially the same as the area of $\mathtt{C}$. Thus, we can pack all these objects into our additional space due to resource augmentation.

Finally, we can afford to increase the space of our knapsack such that this additional space is even by a constant factor larger than the total area of the objects we need to pack into it. Therefore, it is easy to find a discretized packing for them in this extra space. ◀

Algorithm. Now we describe our algorithm. First, we *correctly* guess (i.e., by brute-force enumeration of all possible cases) the values $\delta_{large}, \delta_{small}, \delta_{cell} > 0$ from set D due to Lemma 4. Note that we still do not know $\bigcup_\ell L_\ell$ or $\bigcup_\ell M_\ell$, i.e., which objects are there in the optimal packing. So, for each level ℓ we define $\tilde{L}_\ell$ to be all input objects P with $r^{in}(P) \in (\delta_{large,\ell}, \delta_{small,(\ell-1)}]$ and $\tilde{M}_\ell$ to be all input objects P with $r^{in}(P) \in (\delta_{small,\ell}, \delta_{large,\ell}]$.

Then, we compute an optimal discretized packing via a dynamic program. Intuitively, our DP computes an optimal subset of $\bigcup_\ell \tilde{L}_\ell$ for which there is a discretized packing. We introduce a DP-cell DP$[\ell, m]$ for each combination of a level ℓ and a value $m \in \{1, ..., n\}$. This cell corresponds to the subproblem of packing a maximum profit subset of the objects in $\tilde{L}_\ell, \tilde{L}_{\ell+1}, \tilde{L}_{\ell+2}, \ldots$. via a discretized packing into at most m grid cells of level $\ell - 1$, i.e., with side length $\delta_{c,(\ell-1)}$ each. Recall that each object in $\tilde{L}_\ell$ is relatively large compared to the grid cells of level $\ell - 1$. Therefore, we can pack only constantly many items from $\tilde{L}_\ell$ into each of these m grid cells of level $\ell - 1$. Therefore, there are only constantly many options how the set $\mathcal{G}_P$ of an object $P \in \tilde{L}_\ell$ in the optimal solution to our subproblem can look like. We say that a *configuration* is a partition of a grid cell of level $\ell - 1$ into sets of grid cells of level ℓ. Each grid cell of level $\ell - 1$ contains $O(1/\delta_{cell}^2)$ many grid cells of level ℓ. Hence, there are only constantly many configurations. We assume two configurations to be *equivalent* if they are identical up to translation by an integral multiple of $\delta_{c,(\ell-1)}$, i.e., by an integral multiple of the edge length of a grid cell of level $\ell - 1$. Denote by C the total number of resulting equivalence classes. We guess in time $m^{O(C)} \leq n^{O(C)}$ how many grid cells have each of the

at most C configurations (up to equivalences). Then, we assign the items in $\tilde{L}_\ell$ into the grid cells according to this guess. We can do this by weighted bipartite matching. For each object $P \in \tilde{L}_\ell$, each possible configuration $\mathcal{G}'$, and each set in the partition of $\mathcal{G}'$, we can check easily whether P fits into $\mathcal{G}'$. In the bipartite graph, one side will contain the objects in $\tilde{L}_\ell$ and other side will contain the sets in the partition of $\mathcal{G}'$. If P fits into set Q in the partition of $\mathcal{G}'$, then there is an edge with edge cost as $w(P)$. Our guess yields a certain number m' of empty grid cells of level $\ell+1$ into which we need to pack items in $\tilde{L}_{\ell+1}, \tilde{L}_{\ell+2}, \ldots$. We assign these items according to the solution in the DP-cell $\mathrm{DP}[\ell+1, \min\{m', n\}]$. Note that there are at most n items and, hence, we never need more than n grid cells of level $\ell+1$. Also, since our input data is polynomially bounded, the number of classes $\tilde{L}_i$ is bounded by $n^{O(1)}$. Thus, our DP runs in time $n^{O(C)}$.

Additionally, we pack medium objects from the set $\bigcup_\ell \tilde{M}_\ell$ separately in a strip of the form $[0,1] \times [1, 1+O(\varepsilon)]$. We select the most profitable subset of $\bigcup_\ell \tilde{M}_\ell$ (up to a factor of $1+\varepsilon$) whose total area is bounded by ε (see Lemma 3). We replace each of these objects by the smallest square that contains it, which increases its area only by a constant factor. We can pack these squares efficiently into a (slightly larger) strip $[0,1] \times [1, 1+O(\varepsilon)]$ using the `NFDH` algorithm [13]. This yields the following lemma.

▶ **Lemma 5.** ($\star$) *In polynomial time we can compute a set $\mathcal{P}' \subseteq \bigcup_\ell \tilde{M}_\ell$ and a non-overlapping placement of $\mathcal{P}'$ inside $[0,1] \times [1, 1+O(\varepsilon)]$ such that $w(\mathcal{P}')$ is at least the profit of any subset of $\bigcup_\ell \tilde{M}_\ell$ whose total area is at most ε.*

One can easily extend our algorithm above to any constant dimension d. This completes the proof of Theorem 1. A consequence is that we obtain a polynomial time $(1+\varepsilon)$-approximation *without* resource augmentation if all input objects are small, i.e., if $r^{out}(P) \leq \varepsilon$ for each given object $P \in \mathcal{P}$. Using this property, we can argue that there is a $(1+\varepsilon)$-approximate solution in which only the area $[0, 1-\Theta(\varepsilon)] \times [0, 1-\Theta(\varepsilon)]$ of the knapsack is used. Thus, we can use the free space for the resource augmentation that is required by our algorithm due to Theorem 1.

▶ **Theorem 6.** ($\star$) *Let $d \in \mathbb{N}$ be a constant. There is a polynomial time $(1+O(\varepsilon))$-approximation for the geometric knapsack problem if the set of input objects $\mathcal{P}$ consists of convex fat d-dimensional objects such that $r^{out}(P) \leq \varepsilon$ for each $P \in \mathcal{P}$.*

3 Spheres

In this section we present our $(1+\varepsilon)$-approximation algorithm for the case of d-dimensional spheres. Let $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$ be a set of n number of d-dimensional hyperspheres. We denote the radius and profit of each hypersphere $C_i \in \mathcal{C}$ by r_i and w_i. For an object C_i we denote its volume (or area in 2-dimension) to be $a(C_i)$. For a collection of objects $\mathcal{A}$, we define its volume and profit to be $a(\mathcal{A}) := \sum_{C_i \in A} a(C_i)$ and $w(\mathcal{A}) := \sum_{C_i \in A} w(C_i)$, respectively. We are given a unit knapsack $K := [0,1]^d$.

We first consider the case of circles, i.e., $d = 2$. Let OPT be an optimal solution and $\mathcal{C}_{\mathsf{OPT}}$ be the circles in OPT. Let $\varepsilon \in (0, 1/2]$ be a constant and assume that $1/\varepsilon \in \mathbb{N}$. First, we want to classify the input circles into *small* and *large* circles such that each large circle is much larger than any small circle. Due to the following lemma, we can do this such that we can ignore all circles that are neither large nor small by losing only a factor of $1+\varepsilon$. We will use this standard shifting argument throughout the paper.

▶ **Lemma 7** ($\star$)**.** *There is a set of global constants $\varepsilon^{(0)}, \ldots, \varepsilon^{(1/\varepsilon)} \geqslant 0$ such that $\varepsilon^{(j)} = (\varepsilon^{(j-1)})^{24}$ for each $j \in \{1, \ldots, 1/\varepsilon\}$ and a value $k \in \{1, \ldots, 1/\varepsilon - 1\}$ with the following property: if we define $\varepsilon_{large} := \varepsilon^{(k)}$ and $\varepsilon_{small} := \varepsilon^{(k+1)}$, then sum of profits of all circles C_i in OPT with radii $\varepsilon_{small} \leq r_i \leq \varepsilon_{large}$ is at most $\varepsilon \cdot w(\mathsf{OPT})$.*

We guess the value $k \in \{0, ..., 1/\varepsilon - 1\}$ due to Lemma 7. We define that a circle $C_i \in \mathcal{C}$ is *large* if $r_i > \varepsilon_{large}$ and *small* if $r_i \leqslant \varepsilon_{small}$. Also, note that $\varepsilon_{small} = \varepsilon_{large}^{24}$.

3.1 Guessing large circles

We observe that in OPT there can be only a constant number of large circles since each large circle covers a constant fraction of the available area in the knapsack.

▶ **Proposition 8** ($\star$). *Any feasible solution can contain at most* $(1/\varepsilon)^{O_k(1)}$ *large circles.*

We guess a feasible solution of the large circles in OPT that satisfy the packing constraints in time $n^{(1/\varepsilon)^{O_k(1)}}$, denote them by $\mathcal{C}_L^*$. In related problems, like the two-dimensional knapsack problem with squares or rectangles, one can easily guess the correct placement of the guessed large circles (assuming rational input data). For packing circles, it is not clear that if there is a packing in which the centers of the circles are placed at rational coordinates. However, in the following section, when we pack polygons, we can guarantee that there is an optimal solution in which the corner of each polygon has a rational coordinate.

Therefore, instead we first guess for each circle $C_i \in \mathcal{C}_L^*$ an *estimate* for its placement in OPT. Denote by $\hat{x}_i^{(1)}, \hat{x}_i^{(2)} \in [0, 1]$ the coordinates of the center of C_i in OPT. We guess values $\tilde{x}_i^{(1)}, \tilde{x}_i^{(2)} \in \{0, \frac{\varepsilon}{n}, \frac{2\varepsilon}{n}, ..., 1\}$ such that $\hat{x}_i^{(1)} \in [\tilde{x}_i^{(1)}, \tilde{x}_i^{(1)} + \frac{\varepsilon}{n})$ and $\hat{x}_i^{(2)} \in [\tilde{x}_i^{(2)}, \tilde{x}_i^{(2)} + \frac{\varepsilon}{n})$. Note that there are only $O(n^2/\varepsilon^2)$ possibilities for each $C_i \in \mathcal{C}_L^*$, and hence only $n^{(1/\varepsilon)^{O_k(1)}}$ possibilities overall for all circles $C_i \in \mathcal{C}_L^*$.

Given these guessed values $\tilde{x}_i^{(1)}, \tilde{x}_i^{(2)}$ for each circle $C_i \in \mathcal{C}_L^*$, we verify that our guess was correct or not, i.e., confirm that there exists, indeed a corresponding placement for each circle $C_i \in \mathcal{C}_L^*$ such that the circles in $\mathcal{C}_L^*$ do not overlap. Therefore, we define a system of quadratic inequalities that describes the problem of finding such a placement. We require that this placement is consistent with our guesses $\tilde{x}_i^{(1)}, \tilde{x}_i^{(2)}$ for each $C_i \in \mathcal{C}_L^*$.

$$\begin{aligned}
\max\{\tilde{x}_i^{(1)}, r_i\} \leqslant x_i^{(1)} &\leqslant \min\left\{\tilde{x}_i^{(1)} + \frac{\varepsilon}{n}, 1 - r_i\right\} && \forall C_i \in \mathcal{C}_L^* \\
\max\{\tilde{x}_i^{(2)}, r_i\} \leqslant x_i^{(2)} &\leqslant \min\left\{\tilde{x}_i^{(2)} + \frac{\varepsilon}{n}, 1 - r_i\right\} && \forall C_i \in \mathcal{C}_L^* \\
(x_i^{(1)} - x_j^{(1)})^2 + (x_i^{(2)} - x_j^{(2)})^2 &\geqslant (r_i + r_j)^2 && \forall C_i, C_j \in \mathcal{C}_L^* \\
x_i^{(1)}, x_i^{(2)} &\geqslant 0 && \forall C_i \in \mathcal{C}_L^*
\end{aligned} \tag{1}$$

Let $|\mathcal{C}_L^*| =: t$. Then, the above system has $2t$ variables and $k := O(t^2)$ constraints. It is not clear how to compute a solution to this system in polynomial time. It is not even clear whether it has a solution in which each variable has a rational value. However, in polynomial time, we can *decide* whether it has a solution (without computing the solution itself) using an algorithm from [23].

Note that the set of solutions satisfying system (1) is a semi-algebraic set in the field of real numbers. Thus, whether a given set of circles can be packed or not (the decision problem) reduces to a decision problem of whether this semi-algebraic set is nonempty or not. Here, each constraint $i \in [k]$ in (1) can be written as a function $f_i(x_1^{(1)}, x_1^{(2)}, \ldots, x_t^{(1)}, x_t^{(2)}) \geqslant 0$ where each $f_i \in \mathbb{Q}[x_1^{(1)}, x_1^{(2)}, \ldots, x_t^{(1)}, x_t^{(2)}]$ is a polynomial with rational coefficients of degree at most two. Thus deciding the circle packing problem is equivalent to deciding the truth of the following formula: $F := (\exists x_1^{(1)})(\exists x_1^{(2)}) \ldots (\exists x_t^{(1)})(\exists x_t^{(2)}) \wedge_{i=0}^{k} f_i(x_1^{(1)}, x_1^{(2)}, \ldots, x_t^{(1)}, x_t^{(2)}) \geqslant 0$. To solve this decision problem, we use the following result.

▶ **Theorem 9** ([23]). *Let $f_1, f_2, \ldots, f_k \in \mathbb{Q}[x_1^{(1)}, x_1^{(2)}, \ldots, x_t^{(1)}, x_t^{(2)}]$ be polynomials with absolute value of any coefficient to be represented by M bits and maximum degree Δ. There is an algorithm that decides whether the formula $F := (\exists x_1^{(1)})(\exists x_1^{(2)}) \ldots (\exists x_t^{(1)})(\exists x_t^{(2)}) \wedge_{i=0}^{k} f_i(x_1, y_1, \ldots, x_n, y_n) \geqslant 0$ is true, with a running time of $M^{O(1)}(k\Delta)^{O(t^2)}$.*

If it is true, the algorithm also returns polynomials $f, g_1, h_1, \ldots, g_t, h_t \in \mathbb{Q}[x]$ with coefficients of bit size at most $M^{O(1)}(k\Delta)^{O(t)}$ and maximum degree $k^{O(t)}$, such that for a root x of $f(x)$, the assignment $x_1^{(1)} = g_1(x), x_1^{(2)} = h_1(x), \ldots, x_t^{(1)} = g_n(x), x_t^{(2)} = h_n(x)$ satisfies the formula F.

Moreover, for any rational $\alpha > 0$, it returns values $\bar{x}_1^{(1)}, \bar{x}_1^{(2)} \ldots, \bar{x}_t^{(1)}, \bar{x}_t^{(2)} \in \mathbb{Q}$ such that $|\bar{x}_i^{(1)} - x_i^{(1)}| \leqslant \alpha$ and $|\bar{x}_i^{(2)} - x_i^{(2)}| \leqslant \alpha$, for $1 \leqslant i \leqslant t$, in time at most $(\log(1/\alpha)M)^{O(1)}(k\Delta)^{O(t^2)}$.

We crucially use here that our system has only constantly many variables and constraints, i.e., in (1), we have that Δ, k, t are constants and that M is polynomially bounded in n. From Theorem 9, we see that in polynomial time we can decide whether (1) has a solution.

If the system (1) does not have a solution, then we reject this guessed combination of $\mathcal{C}_L^*$ and values $\tilde{x}_i^{(1)}, \tilde{x}_i^{(2)}$ for each circle $C_i \in \mathcal{C}_L^*$. We assume in the following that it has a solution. Observe that the guessed values $\tilde{x}_i^{(1)}, \tilde{x}_i^{(2)}$ yield an estimate for $\hat{x}_i^{(1)}, \hat{x}_i^{(2)}$ up to a (polynomially small) error of ε/n.

3.2 Placing small circles

We want to select small circles from $\mathcal{C}$ and place them inside the knapsack, so that they do not overlap with each other or with the circles in $\mathcal{C}_L^*$. To this end, we define $\varepsilon_{cell} := \varepsilon_{large}^{12}$ (i.e., $\varepsilon_{small} = \varepsilon_{large}^{12}\varepsilon_{cell} = \varepsilon_{cell}^2$) to subdivide the knapsack into a grid with $1/\varepsilon_{cell}^2$ square grid cells of side length ε_{cell} . Our choice of parameters ensures that each small circle is small compared to each grid cell and each large circle is big compared to each grid cell. Formally, for each $\ell, \ell' \in \{0, 1, \ldots, \frac{1}{\varepsilon_{cell}} - 1\}$ we define a grid cell $G_{\ell,\ell'} := [\ell \cdot \varepsilon_{cell}, (\ell+1) \cdot \varepsilon_{cell}) \times [\ell' \cdot \varepsilon_{cell}, (\ell'+1) \cdot \varepsilon_{cell})$. We define the set of all grid cells by $\mathcal{G} := \{G_{\ell,\ell'} : \ell, \ell' \in \{0, 1, \ldots, 1/\varepsilon_{cell} - 1\}\}$.

We say that a placement of a circle $C_i \in \mathcal{C}_L^*$ is *legal* if its center is placed at a point $(x_i^{(1)}, x_i^{(2)})$ such that $\max\{\tilde{x}_i^{(s)}, r_i\} \leqslant x_i^{(s)} \leqslant \min\left\{\tilde{x}_i^{(s)} + \frac{\varepsilon}{n}, 1 - r_i\right\}$ for each $s \in \{1, 2\}$. We show that there is a structured packing with near-optimal profit in which each small circle is contained in a grid cell that does not intersect with any large circle in $\mathcal{C}_L^*$ in any legal packing of them. This will allow us to decouple the remaining problem for the small circles from the large circles, even though we do not know the exact placement for the latter. Moreover, in each grid cell the small circles use only a reduced area of size $(1-\varepsilon)\varepsilon_{cell} \times (1-\varepsilon)\varepsilon_{cell}$. Let $\mathcal{C}_S^*$ denote the small circles in OPT.

▶ **Lemma 10.** *In polynomial time, we can compute a set of grid cells $\mathcal{G}_w$ such that no grid cell in $\mathcal{G}_w$ intersects with any circle $C_i \in \mathcal{C}_L^*$ for any legal placement of C_i. Moreover, there is a set of small circles $\mathcal{C}_S \subseteq \mathcal{C}_S^*$ such that $w(\mathcal{C}_S) \geqslant (1-\varepsilon)w(\mathcal{C}_S^*)$ and the circles in $\mathcal{C}_S$ can be packed non-overlappingly inside $|\mathcal{G}_w|$ grid cells of size $(1-\varepsilon)\varepsilon_{cell} \times (1-\varepsilon)\varepsilon_{cell}$ each.*

We will prove Lemma 10 later in Section 3.3. Using it, we compute an approximation to the set $\mathcal{C}_S$ via Theorem 1.

We pack the computed circles into our grid cells $\mathcal{G}_w$, denote them by $\mathcal{C}_S'$. In particular, they do not intersect any of the large circles in $\mathcal{C}_L^*$ in any legal placement of them. Our solution (corresponding to the considered guesses) consists of $\mathcal{C}_L^* \cup \mathcal{C}_S'$. Recall that we can guarantee that these circles can be packed non-overlappingly inside the knapsack. Also, for the circles in $\mathcal{C}_S'$ we computed their placement *exactly* and for the circles in $\mathcal{C}_L^*$ we computed their placement up to our polynomially small error of ε/n. In Appendix A we show how to reduce this error to an exponentially small error.

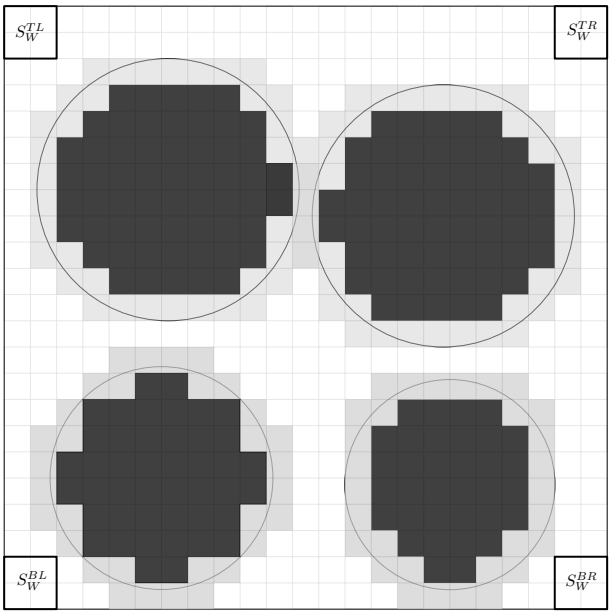


Figure 5 Partitioning grid cells into white, black and gray cells. Later corner regions $S_W^{BL}, S_W^{TL}, S_W^{BR}, S_W^{TR}$ are used to to pack items in gray cells.

3.3 Structural packing for small circles

In this section, we prove Lemma 10. First, we show that intuitively almost every small circle in $\mathcal{C}_S^*$ is contained inside some grid cell. Formally, we show that the total area of all other small circles in $\mathcal{C}_S^*$ is small. For any set $\mathcal{S}$ of circles or grid cells, we define $a(\mathcal{S})$ to be the total area of the elements in $\mathcal{S}$.

▶ **Lemma 11** ($\star$). *Let $\mathcal{C}_{cut} \subseteq \mathcal{C}_S^*$ be the set of all small circles in $\mathcal{C}_S^*$ that intersect more than one grid cell. We have that $a(\mathcal{C}_{cut}) \leqslant 8\varepsilon_{small}/\varepsilon_{cell} \leqslant \varepsilon\varepsilon_{large}^2/64$.*

We will repack the circles in $\mathcal{C}_{cut}$ later such that each of them is contained inside one single grid cell. Thus, for each small circle $C_i \in \mathcal{C}_{\mathsf{OPT}} \setminus \mathcal{C}_{cut}$ there is a grid cell $G_{\ell,\ell'}$ for some $\ell, \ell' \in \{0, 1, ..., 1/\varepsilon_{cell} - 1\}$ such that C_i is contained in $G_{\ell,\ell'}$ in OPT. When we select and place small circles, we must be careful that they do not intersect any large circles from $\mathcal{C}_L^*$. One difficulty for this is that we do not know the precise coordinates of the large circles. Therefore, we place small circles only into grid cells that do not overlap with any large circle from $\mathcal{C}_L^*$ in any legal placement of them. Formally, we partition the cells in $\mathcal{G}$ into three types: white, gray, and black cells (see Figure 5).

▶ **Definition 12.** Let $G_{\ell,\ell'} \in \mathcal{G}$ for some $\ell, \ell' \in \{0, 1, ..., 1/\varepsilon_{cell} - 1\}$. The cell $G_{\ell,\ell'}$ is

- *white* if $G_{\ell,\ell'}$ does not intersect with any circle $C_i \in \mathcal{C}_L^*$ for any legal placement of C_i,
- *black* if $G_{\ell,\ell'}$ is contained in some circle $C_i \in \mathcal{C}_L^*$ for any legal placement of C_i,
- *gray* if $G_{\ell,\ell'}$ is neither white nor black.

The gray cells are problematic for us since a gray cell might be (partially) covered by a large circle in $\mathcal{C}_L^*$ but we do not know by how much (and which part of the cell). Therefore, we do not place any small circles into gray cells. However, OPT might place small circles into these cells (and obtain the profit of these circles). On the other hand, we can show that the number of gray cells is very small, only a small fraction of all grid cells can be gray. In

order to do this, we use the fact that the values $\tilde{x}_i^{(1)}, \tilde{x}_i^{(2)}$ for each circle $C_i \in \mathcal{C}_L^*$ estimate the placement of each large circle relatively accurately, and that the grid cells are relatively small. This allows us to prove that almost all cells are black or white. Also, we can compute all gray cells efficiently. Let $\mathcal{G}_g \subseteq \mathcal{G}$ denote the set of all gray grid cells in $\mathcal{G}$.

▶ **Lemma 13** (⋆). *The total area of gray cells $a(\mathcal{G}_g)$ is at most $\varepsilon\varepsilon_{large}^2/5$. We can compute $\mathcal{G}_g$ in polynomial time.*

Unfortunately, it is not sufficient for us that there are only few gray cells. It might be that almost all cells are either gray or black and, hence, we need to place most of the selected small circles into gray cells (in order to obtain an $(1+\varepsilon)$-approximate solution).

However, we can show that this is not the case. We prove that the number of white cells (which we can safely use for small circles) is at least by a factor $1/\varepsilon$ larger than the number of gray cells. To show this, we exploit the geometry of the circles. In each corner of the knapsack, there are cells that cannot intersect with any large circle, simply because the grid cells are small compared to the large circles and because of the shape of the large circles (see the corner regions in Figure 5). Hence, these grid cells are white. Let $\mathcal{G}_w \subseteq \mathcal{G}$ denote the set of all white grid cells in $\mathcal{G}$.

▶ **Lemma 14** (⋆). *The total area of white grid cells $a(\mathcal{G}_w)$ is at least $\varepsilon_{large}^2/4$. We can compute $\mathcal{G}_w$ in polynomial time.*

Using Lemmas 11, 13, and 14, we show that there is a $(1+\varepsilon)$-approximate solution in which each small circle is contained in some white cell; in particular, no small circle is placed inside a gray cell. To prove this, we delete all small circles in the $O(\varepsilon|\mathcal{G}_w|)$ white grid cells with the smallest total profit among all white cells and place all circles from gray cells and all circles from $\mathcal{C}_{cut}$ into those.

▶ **Lemma 15** (⋆). *There is a set $\mathcal{C}_S' \subseteq \mathcal{C}_S^*$ of small circles with $p(\mathcal{C}_S') \geqslant (1-\varepsilon)p(\mathcal{C}_S^*)$ such that there is a packing for $\mathcal{C}_S'$ using the grid cells in $\mathcal{G}_w$ only.*

We complete the proof of Lemma 10 by applying the following lemma to $\mathcal{C}_S := \mathcal{C}_S''$ which shows that we can sacrifice a factor of $1 + O(\varepsilon)$ to be able to use resource augmentation when we pack the small circles.

▶ **Lemma 16** (⋆). *There is a set of small circles $\mathcal{C}_S'' \subseteq \mathcal{C}_S'$ such that $w(\mathcal{C}_S'') \geq (1-\varepsilon)w(\mathcal{C}_S')$ and it is possible to place the circles in $\mathcal{C}_{\text{sml}}''$ non-overlappingly inside $|\mathcal{G}_w|$ square knapsacks of size $(1-\varepsilon)\varepsilon_{cell} \times (1-\varepsilon)\varepsilon_{cell}$ each.*

3.4 Higher dimensions

Our techniques from the previous section extend directly to the problem of packing hyperspheres in any (constant) dimension d which yields our main theorem for the setting of packing (hyper-)spheres.

▶ **Theorem 17** (⋆). *Let $d \in \mathbb{N}$ be a fixed constant. For the geometric knapsack problem with d-dimensional hyperspheres, there is a polynomial time algorithm that computes a set of hyperspheres $\tilde{\mathcal{C}}$ with $p(\tilde{\mathcal{C}}) \geqslant (1-\varepsilon)\mathsf{OPT}$ that can be placed non-overlappingly inside the knapsack. For all but $O_\varepsilon(1)$ hyperspheres in $\tilde{\mathcal{C}}$ we compute the precise coordinate of the corresponding packing; for the other $O_\varepsilon(1)$ circles we compute an estimate of the packing with an additive error of at most $\frac{1}{2^{n/\varepsilon}}$ in each dimension.*

4 Polygons

In this section, we adjust our techniques from the previous sections to obtain a PTAS for the case that each input object is a fat and convex polygon with at most a constant number of edges whose lengths differ by at most a constant factor, and such that each angle between adjacent edges is larger than $\pi/2$. These objects generalize regular polygons with greater than 4 sides.

Formally, we assume that we are given a set $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ of n polygons that are (f, α, q, t)-*well-behaved*, i.e., for each polygon $P_i \in \mathcal{P}$ we assume that

- P_i is fat, i.e., $r_i^{\text{out}}/r_i^{\text{in}} \leq f$ for some (global) constant $f \geq 1$, where r_i^{out} and r_i^{in} is the radius of the smallest circle containing P_i and the radius of the largest circle contained in P, respectively,
- the angle between any two consecutive edges of P_i is at least $\pi/2 + \alpha$, for some (global) constant $\alpha > 0$,
- P_i has at most q edges for some (global) constant q such that the lengths of any two of its edges differ at most by a factor of t.

For example, regular pentagons are $(2, \pi/10, 5, 1)$-well-behaved. For each polygon $P_i \in \mathcal{P}$ we denote by w_i its profit, and for a set of polygons $\mathcal{P}' \subseteq \mathcal{P}$ we denote by $w(\mathcal{P}') := \sum_{P_i \in \mathcal{P}'} w_i$ their total profit. For any object C we define its area to be $a(C)$, and for any collection of objects $\mathcal{A}$ we denote their total area by $a(\mathcal{A}) := \sum_{P_i \in A} a(P_i)$. We want to pack a subset of $\mathcal{P}$ non-overlappingly into the unit knapsack $K := [0,1]^2$. We do not allow rotations in our packing.

Let $\varepsilon > 0$. We require that $\varepsilon < g(f, \alpha, q, t)$ for a function g to be defined later. In contrast to the case with hyperspheres, we show that we can compute each coordinate of our packing exactly. We classify each polygon $P_i \in \mathcal{P}$ as large or small according to the respective value r_i^{in}. For this, we define values ε_{large} and ε_{small}. For technical reasons, we need that $\varepsilon_{small} \leq h(\varepsilon_{large})$ for some decreasing function $h : \mathbb{R} \to \mathbb{R}$ to be defined later.

▶ **Lemma 18.** ($\star$) *There is a set of global constants $\varepsilon^{(0)}, \dots, \varepsilon^{(1/\varepsilon)} \geqslant 0$ such that $\varepsilon^{(j)} = h(\varepsilon^{(j-1)})$ for each $j \in \{0, \dots, 1/\varepsilon - 1\}$ and a value $k \in \{0, \dots, 1/\varepsilon - 1\}$ with the following properties. If we define $\varepsilon_{large} := \varepsilon^{(k)}$ and $\varepsilon_{small} := \varepsilon^{(k+1)}$, then by losing a factor of $1 + \varepsilon$ in our approximation guarantee, we can assume that each polygon $P_i \in \mathcal{P}$ satisfies that $r_i^{\text{in}} \leq \varepsilon_{small}$ or $r_i^{\text{in}} > \varepsilon_{large}$.*

We guess the value $k \in \{0, \dots, 1/\varepsilon - 1\}$ due to Lemma 18 and define that a polygon $P \in \mathcal{P}$ is *large* if $r_i^{\text{in}} \geq \varepsilon^{(k-1)} = \varepsilon_{large}$ and *small* if $r_i^{\text{in}} < \varepsilon^{(k)} = \varepsilon_{small}$. We discard all input polygons that are neither large nor small. Similar to the case of hyperspheres, we guess the large polygons in OPT. Since they are fat, there can be only constantly many of them.

▶ **Proposition 19.** *Any feasible solution can contain at most $(1/\varepsilon)^{O(1)}$ large polygons.*

We now calculate the placement of the large polygons using a linear program. Define $\mathcal{P}_L^*$ to be the set of large polygons in OPT. Note that, since they are convex polygons instead of circles or hyperspheres, we can compute an *exact* placement of the polygons with *rational* coordinates. For this, we use the following approach, which was also noted by Abrahamsen et al. in [1]. Consider the placement of the polygons $\mathcal{P}_L^*$ in OPT. Each side e of each polygon $P_i \in \mathcal{P}_L^*$ is contained in a line $\{x : a_e x = b_e\}$ for some vector a_e and a scalar b_e. For each corner vertex v of each polygon $P_j \in \mathcal{P}_L^*$, we have that $a_e x \geq b_e$ or $a_e x \leq b_e$ (or both); we guess which of these cases applies. Let v_i be a special vertex for each polygon P_i defined as

a vertex with the least value of $x_i^{(1)}$. Then, the coordinates $(x_i^{(1)}, x_i^{(2)})$ of the special vertex v_i of each polygon $P_i \in \mathcal{P}_L^*$ satisfy a system of linear inequalities defined as follows. There are three types of inequalities:

- *Positivity constraints*: $x_i^{(1)}, x_i^{(2)} \geq 0, \forall P_i \in \mathcal{P}_L^*$
- *Packing constraints*: $\forall P_i, P_j \in \mathcal{P}_L^*$ any vertex v in polygon $P_j \neq P_i$ cannot lie inside P_i. A vertex v lies inside polygon P_i if it satisfies the inequalities described above.
- *Container constraints*: $\forall P_i \in \mathcal{P}_L^*$, $0 \leq x_i^{(1)} \leq a_i$ and $b_i \leq x_i^{(2)} \leq 1 - c_i$, where a_i, b_i, and c_i can be calculated exactly in constant time for a given polygon. They represent the maximum value of $x_{i,v}^{(1)} - x_i^{(1)}$, maximum value of $x_i^{(2)} - x_{i,v}^{(2)}$, and maximum value of $x_{i,v}^{(2)} - x_i^{(2)}$, where $x_{i,v}^{(1)}, x_{i,v}^{(2)}$ vary over all vertices v of polygon P_i.

From Proposition 19, we know that there can only be at most $(1/\varepsilon)^{O(1)}$ large polygons in OPT. We take all possible subsets $\mathcal{P}_L^*$ of this size and smaller from the set $\mathcal{P}$, which is polynomial in number. We compute a feasible solution of packing of these subsets P_L^* to it which is easy since it has only $O_{\varepsilon,k,f,\alpha,t,q}(1)$ variables and constraints for each subset for polynomially many subsets, by using the ellipsoid method.

Now for each guessed large subset P_L^*, we compute a near-optimal packing of the small polygons P_S^*. Our goal is to pack the small polygons in the bin with only a loss of ε-fraction of profit, corresponding to the guessed P_L^*. We then return the solution $P_L^* \cup P_S^*$ which has maximum weight over all guessed values of P_L^* initially and claim that this packing is near-optimal.

In order to pack small polygons, we need a corresponding version of Lemma 10. We define grid cells again similarly such that ε_{cell} is much smaller compared to ε_{large} and ε_{small} is much smaller compared to ε_{cell}. Intuitively, since our input polygons are well-behaved, we can prove that a certain amount of space is not used by the large polygons, similar to Lemma 14. To ensure this, we require that ε is sufficiently small, which in particular also yields a bound on ε_{cell}. Using this, we show that there are many grid cells that are disjoint from any large polygon (similarly as the white grid cells in Section 3).

▶ **Lemma 20.** ($\star$) *There is a function $g : \mathbb{R}^4_{\geq 0} \to \mathbb{R}_{\geq 0}$ such that if all given polygons are (f, α, q, t)-well-behaved and $\varepsilon < g(f, \alpha, q, t)$ then*

- *in polynomial time we can compute a set of grid cells $\mathcal{G}_w$ such that no grid cell in $\mathcal{G}_w$ intersects with any polygon $P_i \in \mathcal{P}_L^*$,*
- *there is a set of small polygons $\mathcal{P}_S \subseteq \mathcal{P}_S^*$ such that $w(\mathcal{P}_S) \geq (1 - O(\varepsilon))w(\mathcal{P}_S^*)$ for the optimal packing of large polygons, and*
- *the polygons in $\mathcal{P}_S$ can be packed non-overlappingly inside $|\mathcal{G}_w|$ grid cells that have size $(1-\varepsilon)\varepsilon_{cell} \times (1-\varepsilon)\varepsilon_{cell}$ each.*

We can prove Lemma 20 with similar techniques as we used in the proof of Lemma 10. In order to select and place the small polygons, we use the algorithm due to Theorem 1.

Let P_S^* denote our computed solution for the small polygons. We return the solution $\tilde{P} = P_L^* \cup P_S^*$ which has maximum weight over all initially guessed combinations for the polygons in P_L^* and their approximate coordinates.

▶ **Theorem 21.** *For any constants $f, q \geq 1$ and $t, \alpha > 0$ there is a PTAS for the geometric knapsack problem for (f, α, q, t)-*well-behaved *polygons.*

5 Conclusion

We almost settle the approximability of the geometric knapsack problem in the setting of packing spheres into a hypercube knapsack. However, it remains an open problem whether rational coordinates always suffice in an optimal packing. If not, it would be an interesting

question to determine the best approximation ratio one can obtain if we allow only rational coordinates for the centers of the circles (while the optimal packing has no such restrictions). It would be also interesting to obtain a PTAS for the case of d-dimensional fat convex objects. Another interesting but difficult open question is whether the case of convex but not necessarily fat input objects in the plane admits a PTAS. The best known result for this setting is only an $O(1)$-approximation in quasi-polynomial time (assuming polynomially bounded integral input data [33]). Already for the special case of axis-parallel rectangles, it is open whether a PTAS exists.

References

1 Mikkel Abrahamsen, Tillmann Miltzow, and Nadja Seiferth. Framework for ER-completeness of two-dimensional packing problems. In *IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1014–1021. IEEE, 2020.

2 Anna Adamaszek, Tomasz Kociumaka, Marcin Pilipczuk, and Michał Pilipczuk. Hardness of approximation for strip packing. *ACM Transactions on Computation Theory*, 9(3):14:1–14:7, 2017.

3 Anna Adamaszek and Andreas Wiese. A quasi-ptas for the two-dimensional geometric knapsack problem. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1491–1505. SIAM, 2014.

4 Nikhil Bansal, José R. Correa, Claire Kenyon, and Maxim Sviridenko. Bin packing in multiple dimensions: inapproximability results and approximation schemes. *Mathematics of Operations Research*, 31(1):31–49, 2006.

5 Nikhil Bansal and Arindam Khan. Improved approximation algorithm for two-dimensional bin packing. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 13–25. SIAM, 2014.

6 Julia A. Bennell and José F. Oliveira. A tutorial in irregular shape packing problems. *Journal of the Operational Research Society*, 60:S93–S105, 2009.

7 Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

8 Ignacio Castillo, Frank J. Kampas, and János D. Pintér. Solving circle packing problems by global optimization: numerical results and industrial applications. *European Journal of Operational Research*, 191(3):786–802, 2008.

9 Vítor Gomes Chagas, Elisa Dell'Arriva, and Flávio Keidi Miyazawa. Approximation schemes under resource augmentation for knapsack and packing problems of hyperspheres and other shapes. In *International Workshop on Approximation and Online Algorithms (WAOA)*, pages 145–159. Springer, 2023.

10 Timothy M. Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012.

11 Hai-Chau Chang and Lih-Chung Wang. A simple proof of Thue's theorem on circle packing, 2010. `arXiv:1009.4322`.

12 Miroslav Chlebík and Janka Chlebíková. Hardness of approximation for orthogonal rectangle packing and covering problems. *J. Discrete Algorithms*, 7(3):291–305, 2009.

13 E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, 1980.

14 Erik D. Demaine, Sándor P. Fekete, and Robert J. Lang. Circle packing for origami design is hard. *CoRR*, 2010. `arXiv:1008.1224`.

15 Sándor P. Fekete, Phillip Keldenich, and Christian Scheffer. Packing disks into disks with optimal worst-case density. *Discrete & Computational Geometry*, 69(1):51–90, 2023.

16 Ferenc Fodor. The densest packing of 13 congruent circles in a circle. *Beiträge zur Algebra und Geometrie*, 44(2):431–440, 2003.

17 Hamish J. Fraser and John A. George. Integrated container loading software for pulp and paper industry. *European Journal of Operational Research*, 77(3):466–474, 1994.

18 E. Friedman. Circles in squares. `https://erich-friedman.github.io/packing/cirinsqu/`. Accessed: 2024-03-03.

19 Waldo Gálvez, Fabrizio Grandoni, Salvatore Ingala, Sandy Heydrich, Arindam Khan, and Andreas Wiese. Approximating geometric knapsack via L-packings. *ACM Trans. Algorithms*, 17(4):33:1–33:67, 2021.

20 Waldo Gálvez, Fabrizio Grandoni, Arindam Khan, Diego Ramírez-Romero, and Andreas Wiese. Improved Approximation Algorithms for 2-Dimensional Knapsack: Packing into Multiple L-Shapes, Spirals, and More. In *Symposium on Computational Geometry (SoCG)*, pages 39:1–39:17, 2021.

21 Waldo Gálvez, Arindam Khan, Mathieu Mari, Tobias Mömke, Madhusudhan Reddy Pittu, and Andreas Wiese. A 3-approximation algorithm for maximum independent set of rectangles. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 894–905. SIAM, 2022.

22 Michael Goldberg. Packing of 14, 16, 17 and 20 circles in a circle. *Mathematics Magazine*, 44(3):134–139, 1971.

23 Dima Grigoriev and Nicolai N. Vorobjov Jr. Solving systems of polynomial inequalities in subexponential time. *J. Symb. Comput.*, 5(1/2):37–64, 1988.

24 Thomas C. Hales and Samuel P. Ferguson. A formulation of the kepler conjecture. *Discrete & Computational Geometry*, 36:21–69, 2006.

25 Mhand Hifi and Rym M'hallah. A literature review on circle and sphere packing problems: Models and methodologies. *Advances in Operations Research*, 2009.

26 Klaus Jansen, Arindam Khan, Marvin Lira, and K. V. N. Sreenivas. A PTAS for packing hypercubes into a knapsack. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 78:1–78:20, 2022.

27 Johannes Kepler. *The six-cornered snowflake*. Paul Dry Books, 2010.

28 Arindam Khan, Arnab Maiti, Amatya Sharma, and Andreas Wiese. On guillotine separable packings for the two-dimensional geometric knapsack problem. In *Symposium on Computational Geometry (SoCG)*, pages 48:1–48:17, 2021.

29 Arindam Khan and Eklavya Sharma. Tight approximation algorithms for geometric bin packing with skewed items. *Algorithmica*, 85(9):2735–2778, 2023.

30 Robert J. Lang. A computational algorithm for origami design. In *Symposium on Computational Geometry (SoCG)*, pages 98–105, 1996.

31 Carla Negri Lintzmayer, Flávio Keidi Miyazawa, and Eduardo Candido Xavier. Two-dimensional knapsack for circles. In *Latin American Theoretical Informatics Symposium (LATIN)*, pages 741–754. Springer, 2018.

32 Boris D. Lubachevsky and Ronald L. Graham. Curved hexagonal packings of equal disks in a circle. *Discrete & Computational Geometry*, 18(2):179–194, 1997.

33 Arturo I. Merino and Andreas Wiese. On the two-dimensional knapsack problem for convex polygons. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 84:1–84:16, 2020.

34 Joseph S. B. Mitchell. Approximating maximum independent set for rectangles in the plane. In *IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 339–350. IEEE, 2021.

35 Flávio K. Miyazawa and Yoshiko Wakabayashi. Techniques and results on approximation algorithms for packing circles. *São Paulo Journal of Mathematical Sciences*, 16(1):585–615, May 2022.

36 Ronald Peikert, Diethelm Würtz, Michael Monagan, and Claas de Groot. Packing circles in a square: a review and new results. In *System Modelling and Optimization*, pages 45–54. Springer, 1992.

37 E. Specht. The best known packings of equal circles in a square. `http://hydra.nat.uni-magdeburg.de/packing/csq/csq.html`. Accessed: 2024-03-03.

38 Péter Gábor Szabó, Mihaly Csaba Markót, Tibor Csendes, Eckard Specht, Leocadio G Casado, and Inmaculada García. *New approaches to circle packing in a square: with program codes*, volume 6. Springer Science & Business Media, 2007.

39 Maryna S. Viazovska. The sphere packing problem in dimension 8. *Annals of Mathematics*, pages 991–1015, 2017.

40 Huaiqing Wang, Wenqi Huang, Quan Zhang, and Dongming Xu. An improved algorithm for the packing of unequal circles within a larger containing circle. *European Journal of Operational Research*, 141(2):440–453, 2002.

A Improving the precision of the large spheres

Recall that for each large circle $C_i \in \mathcal{C}_L^*$ we guessed its center in the optimal packing up to a polynomial error of $\frac{\varepsilon}{n}$. We improve this to only an exponential error of at most $\frac{1}{2^{n/\varepsilon}}$. To do this, we apply Theorem 9 with $\alpha := \Theta(\frac{1}{2^{n/\varepsilon}})$. This yields more precise estimates $\bar{x}_i^{(1)}, \bar{x}_i^{(2)}$ for each $C_i \in \mathcal{C}_L^*$. There is an important subtlety though: for our guessed coordinates $\tilde{x}_i^{(1)}, \tilde{x}_i^{(2)}$ we can assume that they differ from the coordinates of OPT by at most our polynomial error of $\frac{\varepsilon}{n}$. For the new estimates $\bar{x}_i^{(1)}, \bar{x}_i^{(2)}$ we can *not* guarantee this: our subroutine from Theorem 9 possibly returns a solution that is (close to) feasible for the large circles, but not (close to) a solution that is feasible for the large and for the small circles. Because of this, we guessed the estimates $\tilde{x}_i^{(1)}, \tilde{x}_i^{(2)}$ for each $C_i \in \mathcal{C}_L^*$, so that we can assume that these estimates really correspond to OPT and not just to some arbitrary solution to (1).

If it were true that there is always a $(1+\varepsilon)$-approximate solution in which the center of each circle has rational coordinates that can be encoded with a polynomially bounded number of bits, then we could choose α appropriately to compute it. More precisely, we could compute a range for each coordinate that contains only one rational number with a bounded number of bits, and we could compute this number afterward.

Detecting Disjoint Shortest Paths in Linear Time and More

Shyan Akmal
MIT, EECS and CSAIL, Cambridge, MA, USA

Virginia Vassilevska Williams
MIT, EECS and CSAIL, Cambridge, MA, USA

Nicole Wein
University of Michigan, Ann Arbor, MI, USA

Abstract

In the k-Disjoint Shortest Paths (k-DSP) problem, we are given a graph G (with positive edge weights) on n nodes and m edges with specified source vertices $s_1, \ldots, s_k$, and target vertices $t_1, \ldots, t_k$, and are tasked with determining if G contains vertex-disjoint (s_i, t_i)-shortest paths. For any constant k, it is known that k-DSP can be solved in polynomial time over undirected graphs and directed acyclic graphs (DAGs). However, the *exact* time complexity of k-DSP remains mysterious, with large gaps between the fastest known algorithms and best conditional lower bounds. In this paper, we obtain faster algorithms for important cases of k-DSP, and present better conditional lower bounds for k-DSP and its variants.

Previous work solved 2-DSP over weighted undirected graphs in $O(n^7)$ time, and weighted DAGs in $O(mn)$ time. For the main result of this paper, we present optimal *linear time* algorithms for solving 2-DSP on weighted undirected graphs and DAGs. Our linear time algorithms are algebraic however, and so only solve the detection rather than search version of 2-DSP (we show how to solve the search version in $O(mn)$ time, which is faster than the previous best runtime in weighted undirected graphs, but only matches the previous best runtime for DAGs).

We also obtain a faster algorithm for k-Edge Disjoint Shortest Paths (k-EDSP) in DAGs, the variant of k-DSP where one seeks edge-disjoint instead of vertex-disjoint paths between sources and their corresponding targets. Algorithms for k-EDSP on DAGs from previous work take $\Omega(m^k)$ time. We show that k-EDSP can be solved over DAGs in $O(mn^{k-1})$ time, matching the fastest known runtime for solving k-DSP over DAGs.

Previous work established conditional lower bounds for solving k-DSP and its variants via reductions from detecting cliques in graphs. Prior work implied that k-Clique can be reduced to $2k$-DSP in DAGs and undirected graphs with $O((kn)^2)$ nodes. We improve this reduction, by showing how to reduce from k-Clique to k-DSP in DAGs and undirected graphs with $O((kn)^2)$ nodes (halving the number of paths needed in the reduced instance). A variant of k-DSP is the k-Disjoint Paths (k-DP) problem, where the solution paths no longer need to be shortest paths. Previous work reduced from k-Clique to p-DP in DAGs with $O(kn)$ nodes, for $p = k + k(k-1)/2$. We improve this by showing a reduction from k-Clique to p-DP, for $p = k + \lfloor k^2/4 \rfloor$.

Under the k-Clique Hypothesis from fine-grained complexity, our results establish better conditional lower bounds for k-DSP for all $k \geq 4$, and better conditional lower bounds for p-DP for all $p \leq 4031$. Notably, our work gives the first nontrivial conditional lower bounds 4-DP in DAGs and 4-DSP in undirected graphs and DAGs. Before our work, nontrivial conditional lower bounds were only known for k-DP and k-DSP on such graphs when $k \geq 6$.

2012 ACM Subject Classification Theory of computation → Shortest paths

Keywords and phrases disjoint shortest paths, algebraic graph algorithms, disjoint paths, fine-grained complexity, clique

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.9

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2404.15916 [2]

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 9; pp. 9:1–9:17

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding *Shyan Akmal*: Supported by Virginia Vassilevska Williams' Simons Investigator Award. *Virginia Vassilevska Williams*: Supported by NSF Grant CCF-2330048, BSF Grant 2020356, and a Simons Investigator Award.

Acknowledgements We thank an anonymous reviewer for pointing out issues with a proof in a previous version of this work, and another anonymous reviewer for simplifying our construction of covering families. We thank Malte Renken and André Nichterlein for answering questions about previous work, as well as Matthias Bentert for telling us about his and his coauthors' work [3], which independently proves Theorem 6 from this paper. The first author additionally thanks Ryan Williams and Zixuan Xu for conversations about early versions of ideas presented here, and Yinzhan Xu for nice discussions about the arguments in this paper.

1 Introduction

Routing disjoint paths in graphs is a classical problem in computer science. For positive integer k, in the k-Disjoint Paths (k-DP) problem, we are given a graph G with n vertices and m edges, with specified source nodes $s_1, \ldots, s_k$ and target nodes $t_1, \ldots, t_k$, and are tasked with determining if G contains (s_i, t_i)-paths which are internally vertex-disjoint. Beyond being a natural graph theoretic problem to study, k-DP is important because of its deep connections with various computational tasks from the Graph Minors project [22].

Following a long line of research, the polynomial-time complexity of k-DP has essentially been settled: in directed graphs the 2-DP problem is NP-hard [18, Lemma 3], and so is unlikely to admit a polynomial time algorithm, while in undirected graphs k-DP can be solved in $\tilde{O}(m+n)$ time for $k = 2$ [24], and in $O(n^2)$ time or $m^{1+o(1)}$ time for any constant $k \geq 3$ [19, 20].

In this work we study an optimization variant of k-DP, the k-Disjoint Shortest Paths (k-DSP) problem. In k-DSP we are given the same input as in k-DP, but are now tasked with determining if the input contains (s_i, t_i)-*shortest* paths which are internally vertex-disjoint. This problem is interesting both because it is a natural graph algorithms question to investigate from the perspective of combinatorial optimization, and because understanding the complexity of k-DSP could lead to a deeper understanding of the interaction between shortest paths structures in graphs (analogous to how studying k-DP helped develop the rich theory surrounding forbidden minors in graphs).

Compared to k-DP, not much is known about the exact time complexity of k-DSP. In directed graphs, 2-DSP can be solved in polynomial time [6], but no polynomial-time algorithm (or NP-hardness proof) is known for k-DSP for *any* constant $k \geq 3$. In undirected graphs, it was recently shown that for any constant k, k-DSP can be solved in polynomial time [21]. However, the current best algorithms for k-DSP in undirected graphs run in $n^{O(k \cdot k!)}$ time, so in general this polynomial runtime is quite large for $k \geq 3$. For example, the current fastest algorithm for 3-DSP in undirected graphs takes $O(n^{292})$ time [4].

Significantly faster algorithms are known for detecting $k = 2$ disjoint shortest paths. The paper which first introduced the k-DSP problem in 1998 also presented an $O(n^8)$ time algorithm for solving 2-DSP in weighted[1] undirected graphs [15]. The first improvement for this problem came over twenty years later in [1], which presented an algorithm solving 2-DSP in weighted undirected graphs in $O(n^7)$ time, and in unweighted undirected graphs in $O(n^6)$

[1] Throughout, we assume that weighted graphs have positive edge weights.

time. Soon after, [4, Theorem 1] presented an even faster $O(mn)$ time algorithm for solving 2-DSP in the special case of unweighted undirected graphs.[2] The main result of our work is an optimal algorithm for 2-DSP in weighted undirected graphs.

▶ **Theorem 1.** *2-DSP can be solved in weighted undirected graphs in $O(m+n)$ time.*

This result pins down the true time complexity of 2-DSP in undirected graphs, and (up to certain limitations of our algorithm, which we discuss later) closes the line of research for this specific problem, initiated twenty-five years ago in [15].

As discussed previously, over directed graphs the 2-DP problem is NP-hard, and the complexity of k-DSP is open even for $k = 3$. This lack of algorithmic progress in general directed graphs has motivated researchers to characterize the complexity of disjoint path problems in restricted classes of directed graphs. In this context, studying algorithms for routing disjoint paths in directed acyclic graphs (DAGs) has proved to be particularly fruitful. For example, the only known polynomial time algorithm for 2-DSP on general directed graphs works by reducing to several instances of 2-DP on DAGs [6]. Similarly, the fastest known algorithm for k-DSP on undirected graphs works by reducing to several instances of disjoint paths on DAGs [4].

It is known that 2-DP in DAGs can be solved in linear time [25]. More generally, since 1980 it has been known that k-DP in DAGs can be solved in $O(mn^{k-1})$ time, and this remains the fastest known algorithm for these problems for all $k \geq 3$ [18, Theorem 3].

As observed in [6, Proposition 10], the algorithm of [18] for k-DP on DAGs can be modified to solve k-DSP in weighted DAGs in the same $O(mn^{k-1})$ runtime. This is the fastest known runtime for k-DSP in DAGs. In particular, the fastest algorithm for 2-DSP from previous work runs in $O(mn)$ time.

The second result of our work is an optimal algorithm for 2-DSP in weighted DAGs.

▶ **Theorem 2.** *2-DSP can be solved in weighted DAGs in $O(m+n)$ time.*

This settles the time complexity of 2-DSP in DAGs, and marks the first improvement over the $O(mn)$ time algorithm implied by [18] from over thirty years ago. The 2-DSP problem in weighted DAGs generalizes the 2-DP problem in DAGs, and so Theorem 2 also offers an alternate linear time algorithm for 2-DP in DAGs, which is arguably simpler than the previous approaches leading up to [25], many of which involved tricky case analyses and carefully constructed data structures.

Our algorithms for solving 2-DSP in undirected graphs and DAGs are algebraic, and work by testing whether certain polynomials, whose terms encode pairs of disjoint shortest paths in the input graph, are nonzero. As a consequence, the algorithms establishing Theorems 1 and 2 are randomized, and solve 2-DSP with high probability. Moreover, these algorithms determine whether the input graph has a solution, but do not explicitly return solution paths. So while our algorithms solve the decision problem 2-DSP, they do not solve the search problem of returning two disjoint shortest paths if they exist. This is a common limitation for algebraic graph algorithms.

The 2-DSP problem does admit a search to decision reduction – with $O(m)$ calls to an algorithm which detects whether a graph contains two disjoint shortest paths, we can actually find two disjoint shortest paths if they exist. Thanks to the algebraic nature of our algorithms, we can get a slightly better reduction, and find two disjoint shortest paths when they exist with only $O(n)$ calls to the algorithms from Theorems 1 and 2.

[2] It seems plausible that the method of [4] could be adapted to handle weighted undirected graphs as well, but such a generalization does not appear to currently be known.

▶ **Theorem 3.** *We can solve 2-DSP over weighted DAGs and undirected graphs, and find a solution if it exists, in $O(mn)$ time.*

So we can *find* two disjoint shortest paths in weighted undirected graphs in $O(mn)$ time (which still beats the previous fastest $O(n^7)$ time algorithm for weighted undirected graphs, and matches the previous fastest algorithm for *unweighted* undirected graphs), and in weighted DAGs in $O(mn)$ time (which only matches, rather than beats, the previous fastest runtime for 2-DSP in DAGs).

Finally, one can also consider *edge-disjoint* versions of all the problems discussed thus far. The k-Edge Disjoint Paths (k-EDP) and k-Edge Disjoint Shortest Paths (k-EDSP) problems are the same as the respective k-DP and k-DSP problems, except the solutions paths merely need to be edge-disjoint instead of internally vertex-disjoint.

For any constant k, there is a simple reduction from k-EDSP on n nodes and m edges to k-DSP on $O(m+n)$ nodes and $O(m)$ edges. Combining this reduction with Theorems 1 and 2, we get that we can solve 2-EDSP over weighted DAGs and undirected graphs in linear time as well.

▶ **Corollary 4.** *There is an algorithm solving 2-EDSP over weighted DAGs and undirected graphs in $O(m+n)$ time.*

More generally, for all $k \geq 3$ the fastest known algorithms for k-EDSP on DAGs in the literature work by reducing this problem to k-DSP using the reduction mentioned in the previous paragraph. Consequently, the current fastest algorithm for k-EDSP in DAGs runs in $O(m^k)$ time, which in dense graphs is much slower than the $O(mn^{k-1})$ time algorithm known for k-DSP. For the same reason, the fastest known algorithm for k-EDP in DAGs for $k \geq 3$ runs in $O(m^k)$ time.

Our final algorithmic result is that we can solve k-EDSP in weighted DAGs as quickly as the fastest known algorithms for k-DSP.

▶ **Theorem 5.** *The k-EDSP problem can be solved in weighted DAGs in $O(mn^{k-1})$ time.*

Since k-EDSP in weighted DAGs generalizes the k-EDP problem in DAGs, Theorem 5 also implies faster algorithms for this latter problem. Our algorithm is simple and employs the same general approach used by previous routines [18, 6] for this problem.

Lower Bounds

For $k \geq 3$, the known $O(mn^{k-1})$ algorithms for k-DP and k-DSP in DAGs have resisted any improvements over the past three decades. Thus, it is natural to wonder whether there is complexity theoretic evidence that solving these problems significantly faster would be difficult. Researchers have presented some evidence in this vein, in the form of reductions from the conjectured hard problem of detecting cliques in graphs.

Let $k = \Theta(1)$ be a positive integer. A *k-clique* is a collection of k mutually adjacent vertices in a graph. In the k-Clique problem,[3] we are given a k-partite graph G with vertex set $V_1 \sqcup \cdots \sqcup V_k$, where each part V_i has n vertices, and are tasked with determining if G contains a k-clique.

[3] This problem is sometimes referred to in the literature as k-Multicolored Clique. A folklore argument reduces from detecting a k-clique in an arbitrary n-node graph to the k-Clique problem as defined here, by making k copies of the input graph, and only including edges between different copies, e.g. as in [11, Proof of Theorem 13.7].

We can of course solve k-Clique in $O(n^k)$ time, just by trying out all possible k-tuples of vertices. Better algorithms for k-Clique are known, which employ fast matrix multiplication. Let ω denote the exponent of matrix multiplication (i.e., ω is the smallest real such that two $n \times n$ matrices can be multiplied in $n^{\omega+o(1)}$ time). Given positive reals a, b, c, we more generally write $\omega(a, b, c)$ to denote the smallest real such that we can compute the product of an $n^a \times n^b$ matrix and an $n^b \times n^c$ matrix in $n^{\omega(a,b,c)+o(1)}$ time. Then it is known that k-Clique can be solved in

$$C(n, k) = \Theta(n^{\omega(\lfloor k/3 \rfloor, \lceil (k-1)/3 \rceil, \lceil k/3 \rceil)})$$

time [16]. The current fastest matrix multiplication algorithms yield $\omega < 2.37156$ [28]. A popular fine-grained hardness hypothesis posits (e.g., in [27, 13]) that current algorithms for k-Clique are optimal.

▶ Hypothesis 1 (k-Clique Hypothesis). For any integer constant $k \geq 3$, solving k-Clique requires at least $C(n, k)^{1-o(1)}$ time.

In this context, previous work provided reductions from k-Clique to disjoint path problems. For example, [4] presents a reduction from k-Clique to $2k$-DSP on undirected graphs with $O((kn)^2)$ vertices (and this reduction easily extends to DAGs). Our first conditional lower bound improves this result for DAGs, by halving the number of paths needed in the reduction.

▶ **Theorem 6.** *There is a reduction from k-Clique to k-DSP on unweighted DAGs with $O((kn)^2)$ vertices, that runs in $O((kn)^2)$ time.*

▶ **Corollary 7.** *Assuming the k-Clique Hypothesis, k-DSP requires $C(n^{1/2}, k)^{1-o(1)}$ time to solve on unweighted DAGs.*

The previous reduction of [4] yields a weaker bound of $C(n^{1/2}, \lfloor k/2 \rfloor)^{1-o(1)}$ for the time needed to solve k-DSP, assuming the k-Clique Hypothesis. If $\omega > 2$, this earlier result only gives nontrivial (that is, superquadratic) lower bounds for $k \geq 10$, and if $\omega = 2$ is only nontrivial for $k \geq 14$. In comparison, if $\omega > 2$, Corollary 7 is nontrivial for $k \geq 5$, and if $\omega = 2$, Corollary 7 is still nontrivial for $k \geq 7$. See Table 1 for the concrete lower bounds we achieve for small k.

As mentioned before, the k-DSP problem in weighted DAGs generalizes k-DP in DAGs. However, the current fastest algorithms for k-DP have the same time complexity as the current best algorithms for the more general k-DSP problem. To explain this behavior, it is desirable to show conditional lower bounds for k-DP in DAGs, which are similar in quality to the known lower bounds for k-DSP in DAGs.

Such lower bounds have been shown by [10]. In particular, [10] together with standard reductions in parameterized complexity [11, Proofs of Theorems 14.28 and 14.30] implies that there is a reduction from k-Clique to $8k$-EDSP on graphs with $O((kn)^4)$ nodes. One can easily modify this reduction, using the idea in the construction from [5, Section 6], to instead reduce from k-Clique to $8k$-DSP on graphs with $O((kn)^4)$ nodes.

This reduction implies that k-DSP requires $C(n^{1/4}, \lfloor k/8 \rfloor)^{1-o(1)}$ time to solve on DAGs, assuming the k-Clique Hypothesis. For large k, this is the current best conditional lower bound for k-DP in DAGs. However, the blow-up in the graph size and parameter value in this reduction makes this lower bound irrelevant for small k (in fact, the reduction only yields nontrivial lower bounds under the k-Clique Hypothesis for $k \geq 96$).

For small values of k, better conditional lower bounds are known for k-DP. In particular, [23] presents reductions from k-Clique to p-DP and p-DSP on DAGs with $O(kn)$ vertices, for parameter value $p = k + \binom{k}{2}$. For our final conditional lower bound, we improve this reduction, by reducing the number of paths needed to $k + \lfloor k^2/4 \rfloor$.

▶ **Theorem 8.** *Let $k \geq 3$ be a constant integer, and set $p = k + \lfloor k^2/4 \rfloor$. There are $O((kn)^2)$ time reductions from k-Clique to p-DP and p-DSP on unweighted DAGs with $O(kn)$ vertices.*

Table 1 A list of lower bounds implied by Corollary 7 for k-DSP when $5 \leq k \leq 9$. Each row corresponds to a value of k. An entry of α in the left column of the row for a given k value indicates that solving k-DSP in $O(n^{\alpha-\delta})$ time for any constant $\delta > 0$ would require refuting the k-Clique Hypothesis or designing faster matrix multiplication algorithms. An entry of β in the right column in the row for a given k value indicates that assuming the k-Clique Hypothesis, k-DSP requires $n^{\beta-o(1)}$ time to solve. **The previous reduction of [4] gave no nontrivial lower bound for k-DSP for any value of k in this table**, and the reduction of [23] matches our lower bound for $k = 6$, but is worse everywhere else. Table entry values are based off rectangular matrix multiplication exponents from [28, Table 1].

k	k-DSP Exponent Lower Bound	
	(for current ω)	(if $\omega = 2$)
5	2.042	Trivial
6	2.371	Trivial
7	2.794	2.5
8	3.198	3
9	3.557	3

For each integer $p \geq 5$, we can find the largest integer $k \geq 3$ such that $k + \lfloor k^2/4 \rfloor \leq p$, and then apply Theorem 8 to obtain conditional lower bounds for p-DP and p-DSP on DAGs.

▶ **Corollary 9.** *Assuming the k-Clique Hypothesis, the p-DSP and p-DP problems require*

$$\max\left(C(n, k_{even}(p)), C(n, k_{odd}(p))\right)^{1-o(1)}$$

time to solve on unweighted DAGs for all integers $p \geq 5$, where

$$k_{even}(p) = 2\lfloor\sqrt{p+1}\rfloor - 2$$

and

$$k_{odd}(p) = 2\left\lfloor\frac{\sqrt{p+5}-1}{2}\right\rfloor - 1$$

are the largest even and odd integers k such that $k + \lfloor k^2/4 \rfloor \leq p$ respectively.

Assuming the k-Clique Hypothesis, Corollary 9 shows that 5-DSP requires at least $n^{\omega-o(1)}$ time and 8-DSP requires at least $C(n,4)^{1-o(1)}$ time to solve. For the current value of ω, these yield lower bounds of $n^{2.371-o(1)}$ for 5-DSP and $n^{3.198-o(1)}$ for 8-DSP, which are better than the lower bounds implied by Corollary 7 (see Table 1). If $\omega = 2$ however, Corollary 9 does not yield better lower bounds than Corollary 7 for k-DSP.

Previous reductions give nontrivial lower bounds for p-DP only when $p \geq 6$ if $\omega > 2$, and $p \geq 10$ if $\omega = 2$. In comparison, Corollary 9 yields nontrivial lower bounds for p-DP under the k-Clique Hypothesis for $p \geq 5$ if $\omega > 2$, and $p \geq 8$ if $\omega = 2$.

Previously, the reduction of [23] yielded the best lower bounds for p-DP for $p \leq 2016$, and otherwise the reduction of [10] yielded better lower bounds. In comparison, Corollary 9 yields lower bounds matching the reduction from [23] for $p \in \{6, 7, 10\}$, and otherwise, for $\omega > 2$, yields strictly better lower bounds for p-DP for all $p \geq 5$. Moreover, for $\omega = 2$, Corollary 9 yields the best lower bounds for p-DP for all $p \leq 4031$ (with [10] yielding better lower bounds only for larger p). To see quantitatively how Corollary 9 improves the best conditional lower bounds for p-DP from previous work at various concrete values of p, see Table 2.

Table 2 A list of lower bounds implied by Corollary 9 (and previous work) for p-DP at various values of p. Rows correspond to values of p. For a given such row, the entries α, β, γ in the three columns collected under the heading of "p-DP Exponent Lower Bound," read from left to right, indicate that Corollary 9, the reduction of [23], and the reduction of [10] imply that p-DP requires $n^{\alpha-o(1)}$, $n^{\beta-o(1)}$, and $n^{\gamma-o(1)}$ time to solve respectively, assuming the k-Clique Conecture.

p	p-DP Exponent Lower Bound (if $\omega = 2$)		
	From Corollary 9	Reduction of [23]	Reduction of [10]
9	3	Trivial	Trivial
24	6	4	Trivial
89	12	8	Trivial
239	20	14	5
929	40	28	19.5
2016	58	42	42
2969	72	51	62
4031	84	60	84

1.1 Comparison with Previous Algorithms

Previous algorithms for 2-DSP and 2-DP in DAGs are combinatorial in nature: they observe certain structural properties of candidate solutions, and then leverage these observations to build up pairs of disjoint paths. In the special case of *unweighted* undirected graphs, [8] presented an algebraic algorithm for solving a generalization of 2-DSP, but all other prior algorithms for 2-DSP and 2-DP in undirected graphs are combinatorial. Our work is the first to employ algebraic methods to tackle the general weighted 2-DSP problem: our algorithms for 2-DSP on undirected graphs and DAGs work by checking that a certain polynomial, whose monomials correspond uniquely to pairs of disjoint shortest paths in the input graph, is nonzero. To obtain the fast runtimes in Theorems 1 and 2, we evaluate this polynomial over a field of characteristic two, and crucially exploit certain symmetries which make efficient evaluation possible when working modulo two.

Such "mod 2 vanishing" methods have appeared previously in the literature for algebraic graph algorithms, but the symmetries we exploit in our algorithms for 2-DSP differ in interesting ways from those of previous approaches. For example, previous methods tend to work exclusively in undirected graphs (relying on the ability to traverse cycles in both the forwards and backwards directions to produce terms in polynomials which cancel modulo 2), while our approach is able to handle 2-DSP in both undirected graphs and DAGs. It is also interesting that our algorithms solve 2-DSP in *weighted* graphs without any issue, since the previous algebraic graph algorithms we are aware of are efficient in unweighted graphs, but in weighted graphs have a runtime which depends polynomially on the value of the maximum edge weight.

Below, we compare our techniques to previous algebraic algorithms in the literature.

Two Disjoint Paths with Minimum Total Length

The most relevant examples of algebraic graph algorithms in the literature to our work are previous algorithms for the MinSum 2-DP problem: in this problem, we are given a graph G on n vertices, with specified sources s_1, s_2 and targets t_1, t_2, and are tasked with finding internally vertex-disjoint paths P_i from s_i to t_i, such that the sum of the lengths of P_1 and P_2 is minimized, or reporting that no such paths exists.

In *unweighted* undirected graphs, [7] showed that MinSum 2-DP can be solved in polynomial time, with [8, Section 6] providing a faster implementation of this approach running in $\tilde{O}(n^{4+\omega})$ time. Similar to our work, these algorithms check if a certain polynomial enumerating disjoint pairs of paths in G is nonzero or not. These methods rely on G being undirected, and are based off computing determinants of $n \times n$ matrices.

Our approach for 2-DSP differs from these arguments because we seek linear time algorithms, and so **avoid computing determinants** (which would yield $\Omega(n^\omega)$ runtimes). We instead directly enumerate pairs of intersecting paths and subtract them out. This alternate approach also allows us to obtain algorithms which apply to both undirected graphs and DAGs, whereas the cycle-reversing arguments of [8] do not appear to extend to DAGs.

Paths and Linkages with Satisfying Length Conditions

Given sets S and T of p source and target vertices respectively, an (S, T)-linkage is a set of p vertex-disjoint paths, beginning at different nodes in S and ending at different nodes in T. The length of such a linkage is the sum of the lengths of the paths it contains. Recent work has presented algorithms for the problem of finding (S, T)-linkages in undirected graphs of length at least k, fixed-parameter tractable in k. In particular, [17, Section 4] presents an algorithm solving this problem in $2^{k+p}\,\mathrm{poly}(n)$ time. Their algorithm enumerates collections of p walks beginning at different nodes in S and ending at different nodes in T. They then argue that all terms in this enumeration with intersecting walks cancel modulo 2, leaving only the (S, T)-linkages. One idea used in the above cancellation argument is that if two paths P and Q in a collection intersect at a vertex v, then we can pair this collection with a new collection obtained by swapping the suffixes of P and Q after vertex v.

In the 2-DSP problem, solution paths must connect sources s_i to corresponding targets t_i instead of to arbitrary targets, and so we cannot use the above suffix-swapping argument to get cancellation. So to enumerate disjoint shortest paths in our algorithms, we employ somewhat trickier cancellation arguments than what was previously used.

More recently, [14, Section 6] presented an algorithm solving the linkage problem discussed above in $2^k\,\mathrm{poly}(n)$ time (with runtime independent of p). Their approach uses determinants to enumerate (S, T)-linkages. As mentioned previously, we explicitly avoid using determinants so that we can obtain linear time algorithms.

Additional Related Work

There are many additional examples of algebraic graph algorithms in the literature. For example, [9] presents an efficient algorithm for finding shortest cycles through specified subsets of vertices, [12] presents algorithms for finding shortest cycles and perfect matchings in essentially matrix multiplication time, and [8] presents a polynomial time algorithm for finding a shortest cycle of even length in a directed graph. Even more examples of algebraic methods in parameterized algorithms are listed in [14, Table 1].

Bibliographic Remark

While the current paper was under submission, the work [3] of Bentert, Fomin, and Golovach was posted online. The reduction they use to establish [3, Theorem 1] is essentially the same as the reduction we use to prove Theorem 6, so this result was independently shown by [3].

Organization

In Section 2 we introduce notation and recall useful facts about graphs and polynomials used in our results. In Section 3 we provide some informal overviews for the proofs of our results. Full proofs of the results claimed in this paper can be found in the full version of this work [2]. We conclude in Section 4 by highlighting some open problems motivated by this work.

2 Preliminaries

General Notation

Given a positive integer a, we let $[a] = \{1, \ldots, a\}$ denote the set of the first a positive integers. Given positive integers a and b, we let $[a, b] = \{a, \ldots, b\}$ denote the set of consecutive integers from a to b inclusive (if $a > b$, then $[a, b]$ is the empty set).

Throughout, we let k denote a constant positive integer parameter.

Graph Notation and Assumptions

Throughout, we let G denote the input graph on n vertices and m edges. We let $s_1, \ldots, s_k$ denote the source vertices of G, and $t_1, \ldots, t_k$ denote the target vertices of G. A *terminal* is a source or target node. We assume without loss of generality that G is weakly connected (we can do this because we only care about solving disjoint path problems on G, and if terminals of G are in separate weakly connected components, we can solve smaller disjoint path problems on each component separately).

Given an edge $e = (u, v)$, we let $\ell(u, v)$ denote the weight of e in G. We assume all edge weights are positive. We let $\text{dist}(u, v)$ denote the distance of a shortest path (i.e., the sum of the weights of the edges used in a shortest path) from u to v. When we write "path P traverses edge (u, v)" we mean that P first enters u, then immediately goes to v.

We represent paths $P = \langle v_0, \ldots, v_r \rangle$ as sequences of vertices. If the path P passes through vertices u and v in that order, we let $P[u, v]$ denote the subpath of P which begins at u and ends at v. We let $\overleftarrow{P}$ denote the *reverse path* of P, which traverses the vertices of P in reverse order. Given two paths P and Q such that the final vertex of P is the same as the first vertex of Q, we let $P \diamond Q$ denote the concatenation of P and Q.

Shortest Path DAGs

Given a graph G and specified vertex s, the *s-shortest paths DAG of G* is the graph with the same vertex set as G, which includes edge (u, v) if and only if (u, v) is an edge traversed by an (s, v)-shortest path in G. From this definition, it is easy to see that a sequence of vertices is an (s, v)-shortest path of G if and only if it is an (s, v)-path in the s-shortest paths DAG of G. Indeed, every edge of an (s, v)-shortest path in G is contained in the s-shortest paths DAG by definition, and so forms a path in this graph. Conversely, if the sequence of vertices $P = \langle v_0, \ldots, v_r \rangle$ is an (s, v)-path in the s-shortest paths DAG of G, then we can inductively show that $P[s, v_i]$ is a shortest path in G for each index i.

We observe that shortest paths DAGs can be constructed in linear time.

▶ **Proposition 10** (Shortest Path DAGs). *Let G be a weighted DAG or undirected graph with distinguished vertex s. Then we can construct the s-shortest paths DAG of G in linear time.*

Proof. By definition, an edge (u, v) is in the s-shortest paths DAG of G if and only if (u, v) is the last edge of some (s, v)-shortest path in G. This is equivalent to the condition that (u, v) is an edge in G, and

$$\text{dist}(s, v) = \text{dist}(s, u) + \ell(u, v). \tag{1}$$

So, we can construct the s-shortest paths DAG of G by computing the values of $\text{dist}(s, v)$ for all vertices v, and then going through each edge (u, v) in G (if G is undirected, we try out both ordered pairs (u, v) and (v, u) of an edge $\{u, v\}$) and checking if Equation (1) holds.

So to prove the claim, it suffices to compute $\text{dist}(s, v)$ for all vertices v in linear time.

When G is a weighted DAG, we can compute a topological order of G in linear time, and then perform dynamic programming over the vertices in this order to compute $\text{dist}(s, v)$ for all vertices v in linear time (this procedure is just a modified breadth-first search routine).

When G is an undirected graph, we instead use Thorup's linear-time algorithm for single-source shortest paths in weighted undirected graphs [26] to compute $\text{dist}(s, v)$ for all vertices v. ◀

Finite Fields

Our algorithms for 2-DSP in undirected graphs and DAGs involve working over a finite field $\mathbb{F}_{2^q}$ of characteristic two, where $q = O(\log n)$. We work in the Word-RAM model with words of size $O(\log n)$, so that addition and multiplication over this field take constant time.

We make use of the following classical result, which shows that we can test if a polynomial is nonzero by evaluating it at a random point of a sufficiently large finite field.

▶ **Proposition 11** (Schwartz-Zippel Lemma)**.** *Let f be a nonzero polynomial of degree at most d. Then a uniform random evaluation of f over $\mathbb{F}$ is nonzero with probability at least $1 - d/|\mathbb{F}|$.*

3 Technical Overview

3.1 2-DSP Algorithms

We first outline a linear time algorithm solving 2-DP in DAGs. We then discuss the changes needed to solve the 2-DSP problem in weighted DAGs, and then the additional ideas used to solve 2-DSP in weighted undirected graphs.

Let G be the input DAG. For each edge (u, v) in G, we introduce an indeterminate x_{uv}. We assign each pair of paths in G a certain monomial over the x_{uv} variables, which records the pairs of consecutive vertices traversed by the paths. These monomials are constructed so that any pair of disjoint paths has a unique monomial.

Let F be the sum of monomials corresponding to all pairs of paths $\langle P_1, P_2 \rangle$ such that P_i is an (s_i, t_i)-path in G. Let F_{disj} and $F_{\cap}$ be the sums of monomials corresponding to all such pairs of paths which are disjoint and intersecting respectively. Since each disjoint pair of paths produces a distinct monomial, we can solve 2-DP by testing whether F_{disj} is a nonzero polynomial. We can perform this test by evaluating F_{disj} at a random point, by the Schwartz-Zippel lemma (Proposition 11).

Since every pair of paths is either disjoint or intersecting, we have

$$F = F_{\text{disj}} + F_{\cap}$$

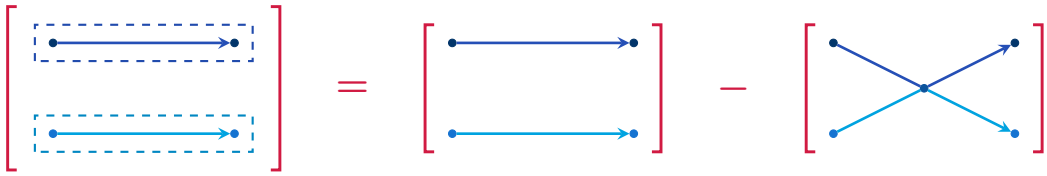


Figure 1 To enumerate the family of disjoint pairs of paths on the left (the dashed borders around the paths indicate that the paths do not intersect), it suffices to enumerate all pairs of paths and subtract out those pairs in the family which intersect at some point.

which implies that

$$F_{\text{disj}} = F - F_{\cap}.$$

This relationship is pictured in Figure 1.

Thus, in order to evaluate F_{disj}, it suffices to evaluate F and $F_{\cap}$. Since F enumerates pairs of paths from the sources to their corresponding targets with no constraints, it turns out that F is easy to evaluate. So solving 2-DP amounts to evaluating $F_{\cap}$ efficiently.

To evaluate $F_{\cap}$, we need a way of enumerating over all pairs of intersecting paths. Each pair of intersecting paths overlaps at a unique earliest vertex v (with respect to the topological order of G). Consequently, if we let F_v be the sum of monomials of pairs of intersecting paths with first intersection at v, we have

$$F_{\cap} = \sum_{v \in V} F_v \tag{2}$$

as depicted in Figure 3.

We evaluate each F_v by relating it to a seemingly simpler polynomial. Let $\tilde{F}_v$ be the polynomial enumerating pairs of paths $\langle P_1, P_2 \rangle$ where P_i is an (s_i, t_i)-path in G such that

1. P_1 and P_2 intersect at vertex v, and
2. the vertices appearing immediately before v on P_1 and P_2 are distinct.

We can think of property 2 as a relaxation of the condition that P_1 and P_2 have v as their earliest intersection point: instead of requiring that $P_1[s_1, v]$ and $P_2[s_2, v]$ never overlap before v, we merely require that these subpaths do not overlap at the position *immediately* before v. It turns out evaluating $\tilde{F}_v$ is easy, because we can enforce property 2 above by enumerating over all pairs of paths which intersect at v, and then subtracting out all such pairs which overlap at some edge ending at v. Simultaneously evaluating all $\tilde{F}_v$ can then be done in $O(m)$ time, roughly because we perform one subtraction for each possible edge the paths could overlap at.

So far, we have explained how to compute all $\tilde{F}_v$ values in linear time. Now comes the key idea behind our algorithm: over fields of characteristic two, the polynomials $\tilde{F}_v$ and F_v are actually identical! Indeed, consider a pair of paths $\langle P_1, P_2 \rangle$ enumerated by $\tilde{F}_v$, which intersects before v. Let the first intersection point of these paths be some vertex u. Then by condition 2 above, the subpaths $P_1[u, v]$ and $P_2[u, v]$ are distinct, because their penultimate vertices are distinct. So if we define new paths

$$Q_1 = P_1[s_1, u] \diamond P_2[u, v] \diamond P_1[v, t_1] \quad \text{and} \quad Q_2 = P_2[s_2, u] \diamond P_1[u, v] \diamond P_2[v, t_2]$$

obtained by swapping the u to v subpaths in P_1 and P_2, we get a new pair of paths $\langle Q_1, Q_2 \rangle$ satisfying conditions 1 and 2 from before, such that each Q_i is an (s_i, t_i)-path in G, which produces the same monomial as $\langle P_1, P_2 \rangle$. This subpath swapping operation is depicted in Figure 2, for $u = a$ and $v = b$. Then modulo two, the contributions of the pairs $\langle P_1, P_2 \rangle$

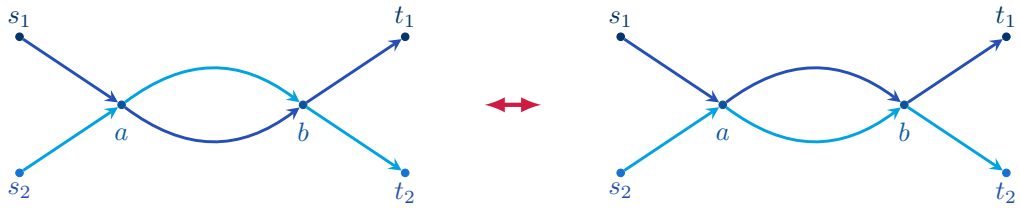


Figure 2 Given paths P_1 and P_2 which intersect at nodes $a = \alpha(P_1, P_2)$ and $b = \beta(P_1, P_2)$, such that a appears before b on both paths, if we swap the a to b subpaths of of P_1 and P_2 to produce new paths Q_1 and Q_2 respectively, then these pairs $f(P_1, P_2) = f(Q_1, Q_2)$ have the same monomials. Moreover, swapping the a to b subpaths of Q_1 and Q_2 recovers P_1 and P_2.

and $\langle Q_1, Q_2 \rangle$ to $\tilde{F}_v$ will cancel out. It follows that all pairs of paths which intersect before v have net zero contribution to $\tilde{F}_v$, and so $\tilde{F}_v = F_v$ as claimed. This congruence is depicted in Figure 4.

Given this observation, we can use our evaluations of $\tilde{F}_v$ in Equation (2) to evaluate $F_\cap$ and thus, by the previous discussion, solve the 2-DP problem.

From Disjoint Paths to Disjoint Shortest Paths

To solve 2-DSP in weighted DAGS, we can modify the 2-DP algorithm sketched above as follows. First, for $i \in [2]$, we compute G_i, the s_i-shortest paths DAG of G. We then construct polynomials as above, but with the additional constraint that they only enumerate pairs of paths $\langle P_1, P_2 \rangle$ with the property that every edge in path P_i lies in G_i. This ensures that we only enumerate pairs of paths which are shortest paths between their terminals.

With this change, the above algorithm for 2-DP generalizes to solving 2-DSP.

▶ Remark 12 (Enumeration Makes Generalization Easy). Previous near-linear time algorithms for 2-DP in DAGs and undirected graphs do not easily generalize to solving 2-DSP. In contrast, as outlined above, in our approach moving from 2-DP to 2-DSP is simple. Why is this?

Intuitively, this happens because our algorithms take an *enumerative* perspective on 2-DSP, rather than the detection-based strategy of previous algorithms. Older algorithms iteratively build up solutions to 2-DP or 2-DSP. Depending on the problem, this involves enforcing different sorts of constraints, since partial solutions to these problems may look quite different. In our approach, we just need to enumerate paths to solve 2-DP and enumerate shortest paths to solve 2-DSP. Enumerating paths and shortest paths are both easy in DAGs by dynamic programming. Hence algorithms for these two problems end up being essentially the same in our framework.

From DAGs to Undirected Graphs

When solving 2-DSP in DAGs, we used the fact that DAGs have a topological order, so that any pair of paths intersects at a unique earliest vertex v in this order. This simple decomposition does not apply to solving 2-DSP in undirected graphs, since we cannot rely on a fixed topological order.

Instead, we perform casework on the first vertex v in P_1 lying in $P_1 \cap P_2$. We observe that in undirected graphs, there are two possibilities: v is either the first vertex in P_2 lying in $P_1 \cap P_2$, or it is the final vertex in P_2 lying in $P_1 \cap P_2$. Intuitively, the paths either "agree" and go in the same direction, or "disagree" and go in opposite directions.

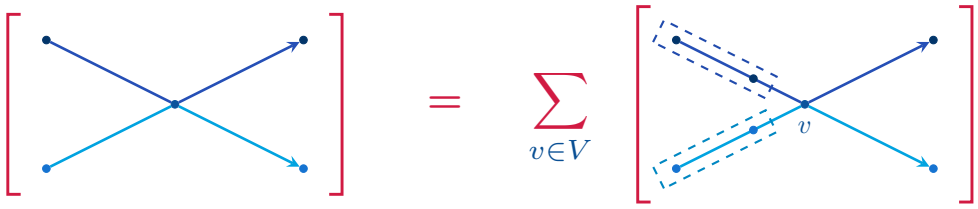


Figure 3 To enumerate the family of intersecting pairs of paths on the left, we can perform casework on the earliest intersection point v for the paths (the dashed border on the subpaths on the right indicates that the paths do not intersect before v).

We then argue that over a field of characteristic two, we can efficiently enumerate over pairs of paths in each of these cases. As with DAGs, we make this enumeration efficient by arguing that modulo 2 we can relax the (a priori difficult to check) condition of v being the first intersection point on P_1 to some simpler "local" condition. When the paths agree, this argument is similar to the reasoning used for solving 2-DSP in DAGs.

For the case where the paths disagree, this enumeration is more complicated, because there is no consistent linear ordering of the vertices neighboring v on the two shortest paths, but can still be implemented in linear time using a more sophisticated local condition. Specifically, if we let a_i and b_i denote the nodes appearing immediately before and after v on path P_i, then to enumerate the "disagreeing paths" modulo two, we prove that it suffices to enumerate paths P_1 and P_2 which intersect at v and have the properties that $a_1 \neq a_2$, $b_1 \neq b_2$, and $a_1 \neq b_2$. Intuitively, using the subpath swapping idea depicted in Figure 4, the conditions that $a_1 \neq a_2$ and $a_1 \neq b_2$ ensure that v is the first vertex of P_1 lying in $P_1 \cap P_2$, and the condition that $b_1 \neq b_2$ ensures that the paths disagree. To implement this idea, we need a slightly more complicated subpath swapping argument, which can also handle the case where two paths P_1 and P_2 intersect at vertices u and v, with u appearing before v on P_1 but u appearing after v on P_2 (this situation does not occur in DAGs, but can occur in undirected graphs). We do this by combining the previous subpath swapping idea with the observation that in undirected graphs we can also traverse subpaths in the reverse direction (so it is possible to swap the subpaths $P_1[u, v]$ and $P_2[v, u]$ in P_1 and P_2, even though u and v appear in different orders on P_1 and P_2).

By combining the enumerations for both cases, we can evaluate F_{disj}, and thus solve 2-DSP over undirected graphs.

3.2 k-EDSP Algorithm

The previous algorithm of [6, Proposition 10] for k-EDSP works by constructing a graph G' encoding information about k-tuples of edge-disjoint shortest paths in the original graph G. This new graph G' has special nodes $\vec{s}$ and $\vec{t}$, such that there is a path from $\vec{s}$ to $\vec{t}$ in G' if and only if G contains k edge-disjoint shortest paths connecting its terminals. The nodes of the new graph G' are k-tuples of edges $(e_1, \ldots, e_k)$ where each e_i is an edge in G. So constructing G' already takes $\Omega(m^k)$ time.

Our algorithm for k-EDSP uses the same general idea. We construct an alternate graph G' which still has the property that finding a single path between two specified vertices of G' solves the k-EDSP problem in G. However, we design G' to have nodes of the form $(v_1, \ldots, v_k)$, where each v_i is a *vertex* in G. Our construction produces a graph with n^k nodes and $O(mn^{k-1})$ edges, which yields the speed-up. We avoid the $\Omega(m^k)$ bottleneck of the previous algorithm by showing how to encode edge-disjointness information simply through the k-tuples of vertices, rather than edges, that the k potential solution paths in G traverse.

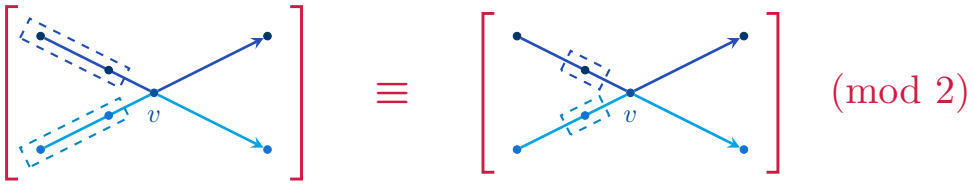


Figure 4 If we work modulo two, then we can enumerate pairs of paths which have common first intersection at node v by enumerating pairs of paths which intersect at v and have the property that the vertices appearing immediately before v on each path are distinct.

3.3 Lower Bounds

Disjoint Shortest Paths

Our proof of Theorem 6 is based on the reduction of [5, Proposition 1] from k-Clique to $2k$-DSP on undirected graphs, which also easily extends to DAGs. Our contribution is a transformation that reduces the number of paths in their reduction from $2k$ to k by exploiting the symmetry of the construction.

The reduction of [5] maps each vertex v in the k-Clique instance to a horizontal path P_v and a vertical path Q_v, each of length n. These paths are arranged so that for each pair of vertices (v, w) in the input graph, the paths P_v and Q_w intersect if and only if (v, w) is not an edge in the input graph. To achieve this, the paths are placed along a grid, and at the intersection point in the grid between paths P_v and Q_w, these two paths are modified to bypass each other to avoid intersection if (v, w) is an edge in the input graph.

The main idea of our transformation is the following. Since the known reduction is symmetric along the diagonal of the grid, it contains some redundancy. We remove this redundancy by only keeping the portion of the grid below the diagonal. To do this, we only have one path P_v for each vertex v in the input graph, and each such path has both a horizontal component and a vertical component. Each path turns from horizontal to vertical when it hits the diagonal. As a result, each pair of paths (P_v, P_w) has exactly one intersection point in the grid (which we bypass if (v, w) is an edge in the input graph). Since we produce only a single path P_v for each vertex v, we obtain a reduction to k-DSP instead of $2k$-DSP.

Disjoint Paths

The starting point for Theorem 8 is the work of [23], which reduces from k-Clique to p-EDP in a DAG with $O(kn)$ nodes, for $p = k + \binom{k}{2}$. The parameter blows up from k to p in this way because the reduction uses k solution paths to pick k vertices in the original graph, and then for each of the $\binom{k}{2}$ pairs of vertices chosen, uses an additional solution path to verify that the vertices in that pair are adjacent in the original graph.

We improve upon this by modifying the reduction graph to allow some solution paths to check multiple edges simultaneously. This lets us avoid using $\binom{k}{2}$ solution paths to separately check for edges between each pair of nodes in a candidate k-clique. Instead, we employ just $\lfloor k^2/4 \rfloor$ solution paths in the reduction, roughly halving the number of paths needed.

To do this, we need to precisely identify which paths can check for multiple edges without compromising the correctness of the reduction. To this end, we examine the structure of the reduction and define a notion of a *covering family* which characterizes which paths can safely check for multiple edges at once. Formally, a k-covering family is a collection $\mathcal{L}$ of increasing lists of positive integers, with the property that for all integers i, j with $1 \leq i < j \leq k$, some list in $\mathcal{L}$ contains i and j as consecutive members.

We show that for any k, the smallest number of lists in a k-covering family is $\lambda(k) = \lfloor k^2/4 \rfloor$ (note that merely obtaining asymptotically tight bounds would not suffice for designing interesting conditional lower bounds). We then insert this construction of a minimum size covering family into the framework of the reduction and prove that the reduction remains correct. Intuitively, given lists in a covering family, we can map each list L to a path which checks edges between vertex parts V_i and V_j for each (i, j) pair appearing as consecutive members of L.

The original reduction of [23] corresponds to implementing this strategy with the trivial k-covering family using $\binom{k}{2}$ lists, achieved by taking a single increasing list of two elements for each unordered pair of integers from $[k]$. Our improved reduction comes from implementing this framework with the optimal bound of $\lfloor k^2/4 \rfloor$ lists.

This yields reductions from k-Clique to p-DP and p-DSP for $p = k + \lambda(k) = k + \lfloor k^2/4 \rfloor$.

4 Conclusion

In this work, we obtained linear time algorithms for 2-DSP in undirected graphs and DAGs. These algorithms are based off algebraic methods, and as a consequence are *randomized* and only solve the *decision*, rather than search, version of 2-DSP. This motivates the following questions:

▶ Open 1. Is there a *deterministic* linear time algorithm solving 2-DSP?

▶ Open 2. Given a DAG or undirected graph G with sources s_1, s_2 and targets t_1, t_2, is there a linear time algorithm *finding* disjoint (s_i, t_i)-shortest paths in G for $i \in \{1, 2\}$?

It is also an interesting research direction to see if algebraic methods can help design faster algorithms for k-DSP in undirected graphs and DAGs when $k \geq 3$, or help tackle this problem in the case of general directed graphs.

In this work, we also established tighter reductions from finding cliques to disjoint path and shortest path problems. There still remain large gaps however, between the current best conditional lower bounds and current fastest algorithms for these problems.

▶ Open 3. Is there a fixed integer $k \geq 3$ and constant $\delta > 0$ such that k-DSP in DAGs can be solved in $O(n^{k+1-\delta})$ time? Or does some popular hypothesis rule out such an algorithm?

Since k-Clique admits nontrivial algorithms by reduction to matrix multiplication, it is possible that k-DSP can be solved faster using fast matrix multiplication algorithms. On the other hand, if we want to rule out this possibility and obtain better conditional lower bounds for k-DSP, we should design reductions from problems which are harder than k-Clique. In this context, a natural strategy would be to reduce from Negative k-Clique and 3-Uniform k-Hyperclique instead, since these problems are conjectured to require $n^{k-o(k)}$ time to solve (and it is not known how to leverage matrix multiplication to solve these problems faster than exhaustive search).

For all $k \geq 3$, the current fastest algorithm for k-DSP in undirected graphs takes $n^{O(k \cdot k!)}$ time, much slower than the $O(mn^{k-1})$ time algorithm known for the problem in DAGs. Despite this, every conditional lower bound that has been established for k-DSP in undirected graphs so far also extends to showing the same lower bound for the problem in DAGs. This is bizarre behavior, and suggests we should try establishing a lower bound which separates the complexities of k-DSP in undirected graphs and DAGs. If designing such a lower bound proves difficult, that would offer circumstantial evidence that far faster algorithms for k-DSP in undirected graphs exist.

▶ Open 4. Can we show a conditional lower bound for k-DSP in undirected graphs, which is stronger than any conditional lower bound known for k-DSP in DAGs?

Finally, for large k, the best conditional time lower bounds we have for k-DP in DAGs are far weaker than the analogous lower bounds we have for k-DSP in DAGs. This is despite the fact that the fastest algorithms we have for both problems run in the same time. It would nice to resolve this discrepancy, either by designing faster algorithms for the latter problem, or showing better lower bounds for the former problem.

▶ Open 5. Is there a fixed integer $k \geq 3$ such that we can solve k-DP in DAGs faster than we can solve k-DSP in weighted DAGs?

▶ Open 6. Can we show a conditional lower bound for k-DP in DAGs matching the best known conditional lower bound for k-DSP in DAGs?

References

1 Maxim Akhmedov. Faster 2-disjoint-shortest-paths algorithm. In *Computer Science – Theory and Applications*, pages 103–116. Springer International Publishing, 2020. doi:10.1007/978-3-030-50026-9_7.

2 Shyan Akmal, Virginia Vassilevska Williams, and Nicole Wein. Detecting Disjoint Shortest Paths in Linear Time and More, 2024. arXiv:2404.15916.

3 Matthias Bentert, Fedor V. Fomin, and Petr A. Golovach. Tight approximation and kernelization bounds for vertex-disjoint shortest paths, 2024. arXiv:2402.15348.

4 Matthias Bentert, André Nichterlein, Malte Renken, and Philipp Zschoche. Using a Geometric Lens to Find k Disjoint Shortest Paths. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:14, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2021.26.

5 Matthias Bentert, André Nichterlein, Malte Renken, and Philipp Zschoche. Using a geometric lens to find k disjoint shortest paths, 2020. doi:10.48550/arXiv.2007.12502.

6 Kristof Berczi and Yusuke Kobayashi. The Directed Disjoint Shortest Paths Problem. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2017.13.

7 Andreas Björklund and Thore Husfeldt. Shortest two disjoint paths in polynomial time. *SIAM Journal on Computing*, 48(6):1698–1710, January 2019. doi:10.1137/18m1223034.

8 Andreas Björklund, Thore Husfeldt, and Petteri Kaski. The shortest even cycle problem is tractable. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. ACM, June 2022. doi:10.1145/3519935.3520030.

9 Andreas Björklund, Thore Husfeldt, and Nina Taslaman. Shortest cycle through specified elements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, January 2012. doi:10.1137/1.9781611973099.139.

10 Rajesh Chitnis. A tight lower bound for edge-disjoint paths on planar dags, 2021. doi:10.48550/arXiv.2101.10742.

11 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer International Publishing, 2015. doi:10.1007/978-3-319-21275-3.

12 Marek Cygan, Harold N. Gabow, and Piotr Sankowski. Algorithmic applications of baur-strassen's theorem. *Journal of the ACM*, 62(4):1–30, September 2015. doi:10.1145/2736283.

13 Mina Dalirrooyfard and Virginia Vassilevska Williams. Induced cycles and paths are harder than you think. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 531–542. IEEE, 2022.

14 Eduard Eiben, Tomohiro Koana, and Magnus Wahlström. Determinantal sieving, 2023. arXiv:2304.02091.

15 Tali Eilam-Tzoreff. The disjoint shortest paths problem. *Discrete Applied Mathematics*, 85(2):113–138, June 1998. doi:10.1016/s0166-218x(97)00121-2.

16 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1-3):57–67, October 2004. doi:10.1016/j.tcs.2004.05.009.

17 Fedor V. Fomin, Petr A. Golovach, Tuukka Korhonen, Kirill Simonov, and Giannos Stamoulis. Fixed-parameter tractability of maximum colored path and beyond, 2022. arXiv:2207.07449.

18 Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, February 1980. doi:10.1016/0304-3975(80)90009-2.

19 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2):424–435, March 2012. doi:10.1016/j.jctb.2011.07.004.

20 Tuukka Korhonen, Michał Pilipczuk, and Giannos Stamoulis. Minor containment and disjoint paths in almost-linear time, 2024. arXiv:2404.03958.

21 Willian Lochet. A polynomial time algorithm for the k-disjoint shortest paths problem. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 169–178. Society for Industrial and Applied Mathematics, January 2021. doi:10.1137/1.9781611976465.12.

22 N. Robertson and P.D. Seymour. Graph minors .XIII. the disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, January 1995. doi:10.1006/jctb.1995.1006.

23 Aleksandrs Slivkins. Parameterized tractability of edge-disjoint paths on directed acyclic graphs. *SIAM Journal on Discrete Mathematics*, 24(1):146–157, January 2010. doi:10.1137/070697781.

24 Torsten Tholey. Solving the 2-disjoint paths problem in nearly linear time. *Theory of Computing Systems*, 39(1):51–78, November 2005. doi:10.1007/s00224-005-1256-9.

25 Torsten Tholey. Linear time algorithms for two disjoint paths problems on directed acyclic graphs. *Theoretical Computer Science*, 465:35–48, December 2012. doi:10.1016/j.tcs.2012.09.025.

26 M. Thorup. Undirected single source shortest paths in linear time. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*. IEEE Comput. Soc, 1997. doi:10.1109/sfcs.1997.646088.

27 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the international congress of mathematicians: Rio de janeiro 2018*, pages 3447–3487. World Scientific, 2018.

28 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega, 2023. arXiv:2307.07970.

The Bit Complexity of Dynamic Algebraic Formulas and Their Determinants

Emile Anand
Caltech, Pasadena, CA, USA

Jan van den Brand
Georgia Tech, Atlanta, GA, USA

Mehrdad Ghadiri
MIT, Cambridge, MA, USA

Daniel J. Zhang
Georgia Tech, Atlanta, GA, USA

Abstract

Many iterative algorithms in computer science require repeated computation of some algebraic expression whose input varies slightly from one iteration to the next. Although efficient data structures have been proposed for maintaining the solution of such algebraic expressions under low-rank updates, most of these results are only analyzed under exact arithmetic (real-RAM model and finite fields) which may not accurately reflect the more limited complexity guarantees of real computers. In this paper, we analyze the stability and bit complexity of such data structures for expressions that involve the inversion, multiplication, addition, and subtraction of matrices under the word-RAM model. We show that the bit complexity only increases linearly in the number of matrix operations in the expression. In addition, we consider the bit complexity of maintaining the determinant of a matrix expression. We show that the required bit complexity depends on the logarithm of the condition number of matrices instead of the logarithm of their determinant. Finally, we discuss rank maintenance and its connections to determinant maintenance. Our results have wide applications ranging from computational geometry (e.g., computing the volume of a polytope) to optimization (e.g., solving linear programs using the simplex algorithm).

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Data Structures, Online Algorithms, Bit Complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.10

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2401.11127

Funding *Jan van den Brand*: Supported by NSF award CCF-2338816.
Daniel J. Zhang: Supported by NSF award CCF-2338816.

Acknowledgements The authors would like to thank Richard Peng for advice and comments.

1 Introduction

Computing algebraic expressions is a workhorse of many iterative algorithms in modern optimization, computational geometry, and dynamic algorithms. Examples include but are not limited to interior point methods for solving linear programs [30, 15, 29, 8, 10], iterative refinement for solving p-norm regression problems [13, 1, 2, 3, 28], semi-definite programming [26, 27], and many algorithmic graph theory problems [33, 11, 6, 14].

Such algebraic expressions are usually represented as matrix formulas involving matrices and operations such as inversion, multiplication, and addition/subtraction. In many iterative algorithms, the algebraic expression does not change over the course of the algorithm, and

only low-rank updates occur to the corresponding matrices from one iteration to the next. For example, for $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$, if we have $\mathbf{AB}$ from a previous iteration and one column of $\mathbf{B}$ changes in the next iteration, we can update $\mathbf{AB}$ in $O(n^2)$ time which is much faster than computing $\mathbf{AB}$ from scratch again. This has been exploited in many iterative algorithms to reduce the amortized cost of iteration and, therefore, the total running time of the algorithms.

A main component of this approach is the Sherman-Morrison-Woodbury (SMW) identity (see (1)), which informally states that the inverse of a rank-k perturbation of a matrix $\mathbf{A}$ can be obtained by a rank-k perturbation of $\mathbf{A}^{-1}$. Although this identity (also called *inverse maintenance*) has been used from the early days of optimization and control theory [31, 32], it was recently shown that *any matrix formula* involving only inversion, multiplication, and addition/subtraction operations can be maintained under low-rank updates with the Sherman-Morrison-Woodbury identity [9]. The main idea is to inductively construct a large matrix whose inverse contains a block that is precisely the output of the formula.

The result of [9] is under the *real-RAM model*, which assumes each arithmetic operation is carried over to infinite precision in constant time. Although this is a valid assumption for finite fields, it does not hold over real numbers. For example, in modern computers, floating-points are the number system of choice that only has a finite precision. Then, it is unclear whether such inverse maintenance techniques are sufficiently stable so that the downstream iterative algorithm outputs the correct solution (e.g., whether the iterative optimization algorithm converges).

Very recently, [24] analyzed the SMW identity over fixed-point arithmetic and showed that a bit complexity proportional to the logarithm of the condition number (ratio of the largest singular value to the smallest singular value) of the corresponding matrix is sufficient to guarantee the stability of the inverse maintenance over the *word-RAM model* in which the running time of arithmetic operations is proportional to the number of bits of corresponding numbers and only finite precision is guaranteed (and the precision itself depends on the number of utilized bits).

This implies that in order to show that the techniques of [9] also hold over the word-RAM model, we need to bound the condition number of the inductively constructed matrix for arbitrary matrix formulas. Indeed, we affirmatively show that the condition number of the constructed matrix is $\kappa^{O(s)}$, where κ is an upper bound for the condition numbers of the input matrices and s is the number of input matrices. This implies that a bit complexity of $O(s \log \kappa)$ is sufficient to guarantee the stability of dynamically updating the matrix formulas. We point out that a naive analysis would give a bound of $O(2^s \log \kappa)$ for the bit complexity, and our bound on the condition number is asymptotically tight since one can easily see that the product of s matrices each with condition number κ results in a matrix with condition number κ^s, e.g., consider $\mathbf{A}^s$.

In addition, we consider the stability and bit complexity of maintaining the determinant and rank of matrix formulas with inversion, multiplication, and addition/subtraction. An application of maintaining the determinant is in the faster computation of the volume of a polytope [22], and an application of the rank maintenance is in dynamic maximum matching [34].

To maintain the determinant of a matrix formula up to a multiplicative error of $(1 \pm \epsilon)$ for $0 < \epsilon < 1$, in addition to the inductively constructed matrix $\mathbf{N}$ of [9], we construct another matrix $\widehat{\mathbf{N}}$ and show that the determinant of the matrix formula is the ratio of $\det(\widehat{\mathbf{N}})$ to $\det(\mathbf{N})$. This then allows us to use the *matrix determinant lemma* (see (3)) to maintain the determinant. Although one might expect that we would require $\log \det(\mathbf{N})$ number of bits for determinant maintenance, we show that $O(s \log(\kappa/\epsilon))$ bits are sufficient. Note that

$\log \kappa$ is preferable to $\log \det$ since, for random matrices, the condition number is polynomial in the dimension of the matrix with high probability [19, 20] while the determinant is exponential [35].

We also consider rank maintenance over finite fields. This is because, under fixed-point arithmetic, we can multiply our matrices by a large number to obtain integer matrices and then perform all operations modulo a sufficiently large prime number ($\text{poly}(n)$ is sufficient). Then the rank of such matrix formula over $\mathbb{Z}_p$ is the same as the rank of the original matrix formula with high probability.

We believe optimizers and algorithm designers can use our results as black boxes to analyze their algorithms under the word-RAM model. The only additional part on their side is to analyze what error bounds can be tolerated in the corresponding algorithm while guaranteeing the returned outputs are correct. Then, our results provide the corresponding running time and bit complexity bounds for the required errors.

Our algorithmic results are presented as dynamic data structures in the next Section 1.1. They cover the most common update schemes occurring in iterative algorithms, such as updating one entry of the matrix and querying one entry or updating a column and querying a row.

Finally, to illustrate the effectiveness of our approach and results, we discuss two example applications. The first one, discussed in the full version, considers finding a basic solution of a set of linear constraints in the standard form $\mathbf{Ax} = \mathbf{b}$, $\mathbf{x} \geq 0$ where $\mathbf{A} \in \mathbb{R}^{d \times n}$ and $n \geq d$. The operations involved in this algorithm are similar to the simplex algorithm. Beling and Megiddo [5] presented two algorithms, a simple one with $O(d^2 n) = O(n^3)$ time, and a more complicated one with $O(d^{1.528} n) = O(n^{2.528})$ time. Both these algorithms assumed the real-RAM model ($O(1)$ time per arithmetic operation with infinite precision). We show that for $n = O(d)$, by simply plugging our data structures into the simple algorithm, the time complexity becomes $\widetilde{O}(n^{2.528} \log(\kappa \cdot \max \det))$ in bit complexity (i.e., number of bit operations). Here $\max \det$ is the maximum determinant over each square $d \times d$ submatrix, and κ is the maximum condition number over each $d \times d$ submatrix. In particular, for matrices with $\log \max \det = \text{poly} \log(n)$ (as is the case when modeling many combinatorial problems as linear programs), our worst-case running time (i.e., number of bit operations) is $\widetilde{O}(n^{2.528} \log \kappa)$. Thus, not only does the simple algorithm become competitive with the more complicated algorithm, but we also show that it can be efficiently implemented without the real-RAM assumptions.

The second example application is for dynamically maintaining the size of the maximum matching of a graph that goes through edge deletion, edge insertion, turning vertices on and off, and merging vertices. We show in the full-version that our rank maintenance data structure can be used for this purpose with a cost of $O(n^{1.405})$ arithmetic operations per update.

1.1 Our Results

Our first result is the following generic data structure that can maintain the value of any matrix formula. Here a matrix formula is any expression that can be written using the basic matrix operations of addition, subtraction, multiplication, and inversion.

▶ **Theorem 1.** *Suppose we are given a matrix formula $f(\mathbf{M}_1, \ldots, \mathbf{M}_s)$ with respective input matrices $\mathbf{M}_1, \ldots \mathbf{M}_s$, where $\|\mathbf{M}_i\|_{\mathrm{F}} \leq \kappa$ for all $i \in [s]$. Let n denote the sum of the number of rows and columns of all $\mathbf{M}_1, \ldots, \mathbf{M}_s$. We further assume that the result of each inversion within f also has Frobenius-norm bounded by κ: in other words, we assume that every*

internal inversion-node of the computation tree has a bounded condition number. Then, for $\epsilon > 0, \kappa > n$*, there exists data structures that are each initialized in time* $\widetilde{O}(n^\omega s \log(\kappa/\epsilon))$ *and have the following operations.*

The data structures have the following update and query operations (where each bullet is a different data structure)

- *Support entry updates and entry queries in* $\widetilde{O}(n^{1.405} s \log(\kappa/\epsilon))$ *time.*
- *Support entry updates in* $\widetilde{O}(n^{1.528} s \log(\kappa/\epsilon))$ *time and entry queries in* $O(n^{0.528} s \log(\kappa/\epsilon))$ *time.*
- *Support column updates and row queries in* $\widetilde{O}(n^{1.528} s \log(\kappa/\epsilon))$ *time.*
- *Support rank-1 updates and returning all entries of* $f(\mathbf{M}_1, ..., \mathbf{M}_s)$ *in* $\widetilde{O}(n^2 s \log(\kappa/\epsilon))$ *time.*
- *Support column updates and row queries in the offline model (the entire sequence of column indices and row queries is given at the start) in* $\widetilde{O}(n^{\omega-1} s \log(\kappa/\epsilon))$ *update and query time.*

The outputs are all ϵ*-approximate, i.e. each entry is off by at most an additive* ϵ*. The stated time complexities depend on current bounds on fast matrix multiplication [37]. The precise dependencies are stated in Theorem 4.*

A similar result was previously proven in [9] using data structures from [33, 12], assuming $O(1)$ time per arithmetic operation and infinite precision. We extend this to the word-RAM model by analyzing the stability of this data structure under the fixed-point arithmetic.

In addition, we show that we can also maintain other properties of $f(\mathbf{M}_1, ..., \mathbf{M}_s)$ while receiving updates to the input matrices. We can maintain the determinant and the rank of $f(\mathbf{M}_1, ..., \mathbf{M}_s)$. The following Theorem 2 and Theorem 3 are proven in the full version.

▶ **Theorem 2.** *Let* $\mathbf{M}_1 \in \mathbb{R}^{n_1 \times d_1}, ..., \mathbf{M}_s \in \mathbb{R}^{n_s \times d_s}$ *and* $n = \sum_{i=1}^{s} n_i + d_i$*. Then, there exists a dynamic determinant data structure that initializes in* $\widetilde{O}(n^\omega s \log(\kappa/\epsilon))$ *time on given accuracy parameters* $\epsilon > 0, \kappa > 2n$*, matrix formula* $f(\mathbf{M}_1, ..., \mathbf{M}_s)$*, and respective input matrices* $\mathbf{M}_1, \ldots, \mathbf{M}_s$*.*

The data structures support the maintenance of $\det(f(\mathbf{M}_1, ..., \mathbf{M}_s))$ *up to a multiplicative factor of* $1 \pm \epsilon$*. They have the following update operations (each bullet is a different data structure)*

- *Support entry updates to any* $\mathbf{M}_i$ *in* $\widetilde{O}(n^{1.405} s \log(\kappa/\epsilon))$ *time.*
- *Support column updates to any* $\mathbf{M}_i$ *in* $\widetilde{O}(n^{1.528} s \log(\kappa/\epsilon))$ *time.*
- *Support rank-1 updates to any* $\mathbf{M}_i$ *in* $\widetilde{O}(n^2 s \log(\kappa/\epsilon))$ *time.*

We assume that throughout all updates, $\|f(\mathbf{M}_1, ..., \mathbf{M}_s)\|_{\mathrm{F}} \leq \kappa$*,* $\|(f(\mathbf{M}_1, ..., \mathbf{M}_s))^{-1}\|_{\mathrm{F}} \leq \kappa$*, and* $\|\mathbf{M}_i\|_{\mathrm{F}} \leq \kappa$ *for all* i*, and the result of each inversion within* f *also has the Frobenius norm bounded by* κ*.*

▶ **Theorem 3.** *There exists a dynamic rank data structure that initializes in* $O(n^\omega)$ *arithmetic operations on given matrix formula* $f(\mathbf{M}_1, ..., \mathbf{M}_s)$*, and respective input matrices. Here,* n *is the sum of the number of rows and columns of all* $\mathbf{M}_1, ..., \mathbf{M}_s$*. The data structure maintains* $\mathrm{rank}(f(\mathbf{M}_1, ..., \mathbf{M}_s))$ *subject to entry updates to any* $\mathbf{M}_i$ *in* $O(n^{1.405})$ *arithmetic operations per update.*

This implies, for example, maintaining the size of the maximum matching in a dynamic graph undergoing edge insertions and deletions, turning vertices on/off, and also merging of vertices (see full version for details). Each such update to the graph takes $O(n^{1.405})$ time. This was previously achieved for edge insertions/deletions only [34, 12].

Theorem 3 gives bounds for rank maintenance over finite fields. It is usually assumed that operations over finite fields take $O(1)$ time. However, a more realistic running time is $\text{poly}\log(|\mathbb{F}|)$ if an isomorphism to polynomials of degree less than d over $\mathbb{Z}_p$ is given, where $|\mathbb{F}| = p^d$ is the size of the field. One approach to go beyond the finite fields for rank maintenance is to perform the operations modulo a random prime, which preserves the rank with constant probability. This has been leveraged in communication complexity literature [36, 23]. See Lemma 4.1 on [36].

1.2 Preliminaries

Notation. We denote matrices with bold uppercase letters and vectors with bold lowercase letters. We denote the Frobenius norm and the operator norm by $\|\cdot\|_{\text{F}}$ and $\|\cdot\|_2$, respectively. We define the condition number of an invertible matrix $\mathbf{M}$ as $\kappa(\mathbf{M}) := \|\mathbf{M}\|_2 \cdot \left\|\mathbf{M}^{-1}\right\|_2$. For simplicity of presentation, we use κ as an upper bound for the Frobenius norm of matrices and their inverses. However, since the condition number of matrices is scale-free, up to polynomial factors such an upper bound is equal to the condition number. When the corresponding matrix is clear from the context, we drop the argument and simply write κ. We denote entry (i, j) of $\mathbf{M}$ by $\mathbf{M}_{i,j}$, row i of $\mathbf{M}$ by $\mathbf{M}_{i:}$ and column j of $\mathbf{M}$ by $\mathbf{M}_{:j}$. For sets I and J, we write $(\mathbf{A})_{I,}$ to denote the rows with indices in I, $(\mathbf{A})_{,J}$ to denote the column with indices in J, and $(\mathbf{A})_{I,J}$ to denote the submatrix with rows with indices in I and columns with indices in J. We denote the $n \times n$ identity matrix by $\mathbf{I}^{(n)}$, and use $\mathbf{0}^{(i,j)}$ to denote the $i \times j$ all-zeros matrix. We denote the transposition of matrix $\mathbf{M}$ by $\mathbf{M}^\top$. We use $\widetilde{O}$ notation to omit polylogarithmic factors in n and polyloglog factors in κ/ϵ from the complexity, i.e., for function f, $\widetilde{O}(f) := O(f \cdot (\log n \cdot \log\log \frac{\kappa}{\epsilon})^c)$, where c is a constant. Further, we denote the set $\{1, \ldots, n\}$ by $[n]$. We denote the number of operations for multiplying an $n^a \times n^b$ matrix with an $n^b \times n^c$ matrix by $O(n^{\omega(a,b,c)})$ and use $O(n^\omega)$ as shorthand for $O(n^{\omega(1,1,1)})$. Finally, for $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $i \in [\min(m, n)]$, let σ_i denote the i'th largest singular value of $\mathbf{A}$.

Sherman-Morrison-Woodbury Identity [38]. Consider an invertible matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$, and matrices $\mathbf{U} \in \mathbb{R}^{n \times r}, \mathbf{D} \in \mathbb{R}^{r \times r}, \mathbf{V} \in \mathbb{R}^{r \times n}$. If $\mathbf{D}$ and $(\mathbf{M} + \mathbf{UDV})^{-1}$ are invertible, then:

$$(\mathbf{M} + \mathbf{UDV})^{-1} = \mathbf{M}^{-1} - \mathbf{M}^{-1}\mathbf{U}(\mathbf{D}^{-1} + \mathbf{VM}^{-1}\mathbf{U})^{-1}\mathbf{VM}^{-1} \tag{1}$$

Schur complement. Consider the block matrix $\mathbf{M}$ given by:

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix},$$

where $\mathbf{A}$ and $\mathbf{D}$ are square matrices. Then if $\mathbf{D}$ is invertible, $\mathbf{M}/\mathbf{D} := \mathbf{A} - \mathbf{BD}^{-1}\mathbf{C}$ is called the *Schur complement* of block $\mathbf{D}$ of matrix $\mathbf{M}$. Similarly, if $\mathbf{A}$ is invertible, $\mathbf{M}/\mathbf{A} := \mathbf{D} - \mathbf{CA}^{-1}\mathbf{B}$ is the Schur complement of block $\mathbf{A}$ of $\mathbf{M}$. The Schur complement gives an inversion formula for block matrices. Particularly, we have the following fact:

Fact. If $\mathbf{A}$ and $\mathbf{M}/\mathbf{A}$ are invertible, then $\mathbf{M}$ is invertible and

$$\mathbf{M}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}(\mathbf{M}/\mathbf{A})^{-1}\mathbf{CA}^{-1} & -\mathbf{A}^{-1}\mathbf{B}(\mathbf{M}/\mathbf{A})^{-1} \\ (\mathbf{M}/\mathbf{A})^{-1}\mathbf{CA}^{-1} & (\mathbf{M}/\mathbf{A})^{-1} \end{bmatrix}.$$

This can be easily verified by multiplication with $\mathbf{M}$.

The Frobenius norm over $\mathbb{R}$ satisfies non-negativity, homogeneity, and the triangle inequality. Specifically, we have that $\|\mathbf{A}\|_{\mathrm{F}} \geq 0$, $\|\beta\mathbf{A}\|_{\mathrm{F}} = |\beta| \|\mathbf{A}\|_{\mathrm{F}}$ and $\|\mathbf{A}+\mathbf{B}\|_{\mathrm{F}} \leq \|\mathbf{A}\|_{\mathrm{F}} + \|\mathbf{B}\|_{\mathrm{F}}$. Finally, when the product $\mathbf{AB}$ is defined, the Frobenius norm obeys the following sub-multiplicative property: $\|\mathbf{AB}\|_{\mathrm{F}} \leq \|\mathbf{A}\|_{\mathrm{F}} \|\mathbf{B}\|_{\mathrm{F}}$.

Our Computational Model. Our algorithms and analysis are under the fixed-point arithmetic. We present all of our analysis under fixed-point arithmetic except for the result of [17] for QR decomposition which is under floating-point arithmetic but we only use that result in a black-box way. In fixed-point arithmetic, each number is represented with a fixed number of bits before and after the decimal point, e.g., under fixed-point arithmetic with 6 bits, we can only present integer numbers less than 64. Addition/subtraction and multiplication of two numbers with n bits can be done in $\widetilde{O}(n)$ time in this model by using fast Fourier transform (FFT) – see [16, Chapter 30]. Division to an additive error of ϵ can also be performed in $\widetilde{O}(n + \log(1/\epsilon))$ time again with the help of FFT. In general, when we mention running time, we mean the number of bit operations. Otherwise, we specify the complexity is about the number of arithmetic operations.

Matrix Formula. Intuitively, a matrix formula is any formula involving matrices and the basic matrix operations of adding, subtracting, multiplying, or inverting matrices. E.g., $f(\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D}) = (\mathbf{AB}+\mathbf{C})^{-1}\mathbf{D}$ is a matrix formula.

Formally, a matrix formula $f(\mathbf{M}_1,...,\mathbf{M}_s)$ is a directed tree, where each input $\mathbf{M}_i$ is a leaf, and each matrix operation (addition, subtraction, multiplication, inversion) is an internal node. Nodes that represent addition, subtraction, or multiplication have two children, i.e. the two terms that are being added, subtracted, or multiplied. Inversion has only one child, the term being inverted. For example, for node v labeled "+", the subtree rooted at the left child and the subtree rooted at the right child represent formulas $g_{\text{left}}(\mathbf{M}_1,...,\mathbf{M}_k), g_{\text{right}}(\mathbf{M}_{k+1},...,\mathbf{M}_s)$, and the tree rooted at v represents $f(\mathbf{M}_1,...,\mathbf{M}_s) = g_{\text{left}}(\mathbf{M}_1,...,\mathbf{M}_k) + g_{\text{right}}(\mathbf{M}_{k+1},...,\mathbf{M}_s)$.

Note that since a formula is a tree, and not a DAG, and because there is no point in inverting something twice in succession, a formula (i.e., tree) with s input matrices (i.e., leaves) has at most $O(s)$ operations (i.e., internal nodes). Further, note that by formulas being trees, a leaf (input) can be used only once. For example, $(\mathbf{A}+\mathbf{B})\mathbf{A}$ is a formula with 3 inputs.

2 Technical Overview

Here we outline how we obtain our three main results Theorems 1–3. [9] proved a variant of Theorem 1 that assumed the "real-RAM model", i.e., exact arithmetic in only $O(1)$ time per operation. Modern computers do not provide this guarantee unless one uses up to $\Omega(n)$ bit, substantially slowing down each arithmetic operation and the overall algorithm. We show that the techniques by [9] also work with bounded accuracy and much smaller bit complexity. In particular, only $\widetilde{O}(s\log\kappa)$ bits are enough, as stated in Theorem 1.

Before we can outline how to obtain these results, we need to give a brief recap of [9]. This is done in Section 2.1. We outline in Section 2.2 how to prove that $\widetilde{O}(s\log\kappa)$-bit accuracy suffice, resulting in Theorem 1. At last, Section 2.3 outlines how to extend Theorem 1 to maintain determinant and rank of a matrix formula, i.e., prove Theorems 2 and 3.

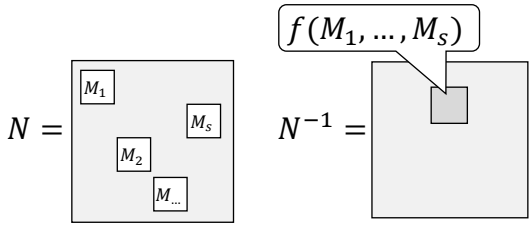


Figure 1 Maintaining $\mathbf{N}^{-1}$ allows us to maintain $f(\mathbf{M}_1, ..., \mathbf{M}_s)$.

2.1 Setting the Stage: How to Maintain Dynamic Algebraic Formulas

Dynamic Matrix Formula is the following data structures task: We are given a formula $f(\mathbf{M}_1, ..., \mathbf{M}_s)$ and respective input matrices $\mathbf{M}_1, ..., \mathbf{M}_s$. The entries of these matrices change over time, and the data structure must maintain $f(\mathbf{M}_1, ..., \mathbf{M}_s)$ (see Theorem 1). *Dynamic Matrix Inverse* is the special case $f(\mathbf{M}) = \mathbf{M}^{-1}$, i.e., given a matrix that changes over time, we must maintain information about its inverse. The latter problem has been studied in, e.g., [33, 12] and there exist several data structures for this task (see Theorem 7).

Previously, [9] showed that the dynamic matrix formula for any formula can be reduced to the special case of matrix inversion, i.e., dynamic matrix inverse. In particular, this means all the previous data structures for dynamic matrix inverse [33, 12] or simple application of the Sherman-Morrison-Woodbury identity (1), can also be used to maintain any general formula $f(\mathbf{M}_1, ..., \mathbf{M}_s)$. [9] shows that for any formula $f(\mathbf{M}_1, ..., \mathbf{M}_s)$, there is a large block matrix $\mathbf{N}$, where $\mathbf{M}_1, ..., \mathbf{M}_s$ occur as subblocks. Further, $\mathbf{N}^{-1}$ contains a block[1] that is precisely $f(\mathbf{M}_1, ..., \mathbf{M}_s)$. See Figure 1. When $\mathbf{M}_1, ..., \mathbf{M}_s$ change over time, matrix $\mathbf{N}$ changes over time, and running a dynamic matrix inverse data structures on this $\mathbf{N}$ allows us to maintain $\mathbf{N}^{-1}$ and its subblock containing $f(\mathbf{M}_1, ...\mathbf{M}_s)$.

The issue of the reduction is that we do not know if $\mathbf{N}$ is well-conditioned. Under which conditions to f and $\mathbf{M}_1, ..., \mathbf{M}_s$ can we guarantee that matrix $\mathbf{N}$ is well-conditioned? Once we can guarantee that $\mathbf{N}$ is well-conditioned, we can give good error guarantees for the dynamic matrix inverse data structures that maintain $\mathbf{N}^{-1}$ via the result of [24] regarding the numerical stability of SMW identity (see full version for details).

We will bound both $\|\mathbf{N}\|_{\mathrm{F}}$ and $\|\mathbf{N}^{-1}\|_{\mathrm{F}}$, which gives a bound on the condition number. In particular, we show that under reasonable assumptions on formula f and input $\mathbf{M}_1, ..., \mathbf{M}_s$, both Frobenius-norms are bounded by $\kappa^{O(s)}$ (Lemma 6). With the dynamic matrix inverse data structures' complexities scaling in the log of these Frobenius norms (see Theorem 7), this leads to the $\widetilde{O}(s \log \kappa)$ factors in Theorem 1.

2.2 Bounding the Frobenius Norms

As outlined in the previous subsection, when given a matrix formula $f(\mathbf{M}_1, ..., \mathbf{M}_s)$, [9] constructs a matrix $\mathbf{N}$ with the following properties. Matrix $\mathbf{N}$ contains $\mathbf{M}_1, ..., \mathbf{M}_s$ as subblock, and the inverse $\mathbf{N}^{-1}$ contains a subblock that is precisely $f(\mathbf{M}_1, ..., \mathbf{M}_s)$, see Figure 1. Our task is to bound the Frobenius-norm of both $\mathbf{N}$ and $\mathbf{N}^{-1}$, which then implies a bound on the condition number of $\mathbf{N}$. Further, data structures for maintaining $\mathbf{N}^{-1}$ have a runtime that scales in those norms (Theorem 7).

[1] The proof is constructive. Given $f, \mathbf{M}_1, ..., \mathbf{M}_s$, we know $\mathbf{N}$ and we know which submatrix of $\mathbf{N}^{-1}$ contains $f(\mathbf{M}_1, ..., \mathbf{M}_s)$, i.e., we do not have to search for the subblock.

Let us briefly recap how matrix $\mathbf{N}$ is constructed, so we can then analyze the Frobenius-norms of $\mathbf{N}$ and $\mathbf{N}^{-1}$.

Construction of N. The given formula $f(\mathbf{M}_1, ..., \mathbf{M}_s)$ can be represented as a tree (where the operations like matrix product, sum, or inversion are nodes, the input matrices are leaves, and the output is the root). The construction of $\mathbf{N}$ follows by induction over the size of the tree: E.g., given some $f(\mathbf{M}_1, ..., \mathbf{M}_s) = g_1(\mathbf{M}_1, ..., \mathbf{M}_k) \cdot g_2(\mathbf{M}_{k+1}, ..., \mathbf{M}_s)$, by induction hypothesis there are matrices $\mathbf{N}_1, \mathbf{N}_2$ where $\mathbf{N}_i$ contains the input matrices of g_i as blocks, and a block of $\mathbf{N}_i^{-1}$ contains the evaluation of f_i. These two matrices are then combined into one larger matrix $\mathbf{N}$ (i.e., $\mathbf{N}$ contains $\mathbf{N}_1, \mathbf{N}_2$ as sub-blocks and thus $\mathbf{N}$ contains $\mathbf{M}_1, ..., \mathbf{M}_s$ as sub-blocks) with the property that $\mathbf{N}^{-1}$ contains a sub-block that is precisely $g_1(\mathbf{M}_1, ..., \mathbf{M}_k) \cdot g_2(\mathbf{M}_{k+1}, ...\mathbf{M}_s)$.

Here we will not go into the precise construction of $\mathbf{N}$ for the different arithmetic operations $f(\mathbf{M}_1, ..., \mathbf{M}_s) = g_1(\mathbf{M}_1, ..., \mathbf{M}_k) \circ g_2(\mathbf{M}_{k+1}, ..., \mathbf{M}_s)$. To follow the outline, it is only important to know that we perform induction by splitting the formula $f = g_1 \circ g_2$ at its root into g_1 and g_2 to obtain two smaller matrices $\mathbf{N}_1, \mathbf{N}_2$. If the root is an inversion, i.e., $f(\mathbf{M}_1, ..., \mathbf{M}_s) = (g(\mathbf{M}_1, ..., \mathbf{M}_s))^{-1}$, then we only have one matrix $\mathbf{N}'$ where $\mathbf{N}'^{-1}$ contains a sub-block that is $g(\mathbf{M}_1, ..., \mathbf{M}_s)$.

Bounding the Frobenius Norm (Section 3.1). For each possible operation $\circ \in \{+, -, \cdot\}$ at the root: $f(\mathbf{M}_1, ..., \mathbf{M}_s) = g_1(\mathbf{M}_1, ..., \mathbf{M}_k) \circ g_2(\mathbf{M}_{k+1}, ..., \mathbf{M}_s)$ there is a different construction for how to combine $\mathbf{N}_1, \mathbf{N}_2$ into a single $\mathbf{N}$, such that $\mathbf{N}^{-1}$ contains $f(\mathbf{M}_1, ..., \mathbf{M}_s)$ as a submatrix. We bound the Frobenius-norm of $\mathbf{N}$ and $\mathbf{N}^{-1}$, with respect to the Frobenius-norms of $\mathbf{N}_1, \mathbf{N}_2$. This then implies a bound by induction. Since the construction of $\mathbf{N}$ differs depending on the operation $\circ \in \{+, -, \cdot\}$, we need slightly different proof for each operation. The proofs will all follow via simple applications of triangle inequalities. Since $\mathbf{N}$ is constructed so that $\mathbf{N}_1, \mathbf{N}_2$ are submatrices of $\mathbf{N}$, simple arguments via triangle inequality suffice. However, for the special case of $f(\mathbf{M}_1, ..., \mathbf{M}_s) = (g(\mathbf{M}_1, ..., \mathbf{M}_s))^{-1}$, a more careful analysis is required, which we outline below.

For $\mathbf{N}'$ being the matrix constructed for formula $g(\mathbf{M}_1, ..., \mathbf{M}_s)$, we have sets $I', J' \subset \mathbb{N}$ where $(\mathbf{N}'^{-1})_{I',J'} = g(\mathbf{M}_1, ..., \mathbf{M}_s)$ (i.e., this is the subblock that contains the evaluation of $g(\mathbf{M}_1, ..., \mathbf{M}_s)$). The reduction by [9] then constructs $\mathbf{N}$ as follows, and we state its inverse:

$$\mathbf{N} = \begin{bmatrix} \mathbf{N}' & -\mathbf{I}^{(n_{N'})}_{,J'} \\ \mathbf{I}^{(n_{N'})}_{I',} & \mathbf{0}^{(n_w, n_w)} \end{bmatrix}$$

$$\mathbf{N}^{-1} = \begin{bmatrix} \mathbf{N}'^{-1} - (\mathbf{N}'^{-1})_{,J'}(\mathbf{N}'^{-1})^{-1}_{I',J'}(\mathbf{N}'^{-1})_{I',} & (\mathbf{N}'^{-1})_{,J'}(\mathbf{N}'^{-1})^{-1}_{I',J'} \\ -(\mathbf{N}'^{-1})^{-1}_{I',J'}(\mathbf{N}'^{-1})_{I',} & ((\mathbf{N}'^{-1})_{I',J'})^{-1} \end{bmatrix}$$

Note that the bottom right block of $\mathbf{N}^{-1}$ is precisely $g(\mathbf{M}_1, ..., \mathbf{M}_s)^{-1}$ since $(\mathbf{N}'^{-1})_{I',J'} = g(\mathbf{M}_1, ..., \mathbf{M}_s)$. To bound the Frobenius-norm of $\mathbf{N}$ and $\mathbf{N}^{-1}$, we apply the triangle inequality to

$$\begin{aligned} \|\mathbf{N}\|_{\mathrm{F}} &\leq \left\| \begin{bmatrix} \mathbf{N}' & 0 \\ 0 & 0 \end{bmatrix} \right\|_{\mathrm{F}} + \left\| \begin{bmatrix} 0 & -\mathbf{I}^{(n_{N'})}_{,J'} \\ \mathbf{I}^{(n_{N'})}_{I',} & 0 \end{bmatrix} \right\|_{\mathrm{F}} \\ &\leq \|\mathbf{N}'\|_F + \sqrt{|J'| + |I'|} \end{aligned}$$

We will have an upper bound on $\|\mathbf{N}'\|_{\mathrm{F}}$ by the inductive hypothesis, so this yields an upper bound on $\|\mathbf{N}\|_{\mathrm{F}}$ as well.

Using the triangle inequality similarly, we also can also bound $\|\mathbf{N}^{-1}\|_{\mathrm{F}}$ by

$$\begin{aligned}\|\mathbf{N}^{-1}\|_{\mathrm{F}} \leq &\left\|\mathbf{N}'^{-1} - (\mathbf{N}'^{-1})_{,J'}(\mathbf{N}'^{-1})^{-1}_{I',J'}(\mathbf{N}'^{-1})_{I',}\right\|_{\mathrm{F}} \\ &+ \left\|(\mathbf{N}'^{-1})_{,J'}(\mathbf{N}'^{-1})^{-1}_{I',J'}\right\|_{\mathrm{F}} + \left\|(\mathbf{N}'^{-1})^{-1}_{I',J'}(\mathbf{N}'^{-1})_{I',}\right\|_{\mathrm{F}} + \left\|(\mathbf{N}'^{-1})^{-1}_{I',J'}\right\|_{\mathrm{F}} \end{aligned} \tag{2}$$

We can split the sum by the triangle inequality, and products using $\|\mathbf{AB}\|_{\mathrm{F}} \leq \|\mathbf{A}\|_{\mathrm{F}}\|\mathbf{B}\|_{\mathrm{F}}$. If we naively upper bound $\|(\mathbf{N}'^{-1})_{I',}\|_{\mathrm{F}}$ and $\|(\mathbf{N}'^{-1})_{,J'}\|_{\mathrm{F}}$ using $\|(\mathbf{N}'^{-1})\|_{\mathrm{F}}$, we will get

$$\begin{aligned}&\|\mathbf{N}'^{-1} - (\mathbf{N}'^{-1})_{,J'}(\mathbf{N}'^{-1})^{-1}_{I',J'}(\mathbf{N}'^{-1})_{I',}\|_{\mathrm{F}} \\ &\quad\leq \|\mathbf{N}'^{-1}\|_{\mathrm{F}} + \|(\mathbf{N}'^{-1})_{,J'}\|_{\mathrm{F}}\|(\mathbf{N}'^{-1})^{-1}_{I',J'}\|_{\mathrm{F}}\|(\mathbf{N}'^{-1})_{I',}\|_{\mathrm{F}} \\ &\quad\leq \|\mathbf{N}'^{-1}\|_{\mathrm{F}} + \|(\mathbf{N}'^{-1})^{-1}_{I',J'}\|_{\mathrm{F}}\|(\mathbf{N}'^{-1})\|^2_{\mathrm{F}}\end{aligned}$$

and similarly for the other three terms of (2). This will yield an upper bound for $\|\mathbf{N}^{-1}\|_{\mathrm{F}}$, but it involves $\|(\mathbf{N}'^{-1})\|^2_{\mathrm{F}}$. In particular, the upper bound gets squared for every nested inverse gate, which will yield a bound that is in the order of κ^{2^s} (with $O(s)$ being a bound on the number of gates).

To improve this, we bound $\|(\mathbf{N}'^{-1})_{I',}\|_{\mathrm{F}}$ and $\|(\mathbf{N}'^{-1})_{,J'}\|_{\mathrm{F}}$ inductively as well. This removes the dependence on $\|(\mathbf{N}'^{-1})\|^2_{\mathrm{F}}$, so the upper bound no longer gets squared in every iteration, and becomes $\kappa^{O(s)}$ instead.

2.3 Dynamic Rank and Determinant of Matrix Formulas

So far, we outlined how to maintain $f(\mathbf{M}_1, ...\mathbf{M}_s)$ within finite precision. This is based on a reduction by [9] from the dynamic matrix formula to the dynamic matrix inverse. We now explain how to extend the reduction, allowing us to also maintain $\det(f(\mathbf{M}_1, ..., \mathbf{M}_s))$ and $\mathrm{rank}(f(\mathbf{M}_1, ..., \mathbf{M}_s))$.

Maintaining the Determinant. First, note that given a block matrix, we can represent its determinant as follows

$$\text{For } \mathbf{M} = \begin{bmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{C}^\top & \mathbf{D}\end{bmatrix} \text{ we have } \det(\mathbf{M}) = \det(\mathbf{A}) \cdot \det(\mathbf{D} - \mathbf{C}^\top\mathbf{A}^{-1}\mathbf{B}).$$

This allows for the following observation: Given $n \times n$ matrix $\mathbf{N}$ and sets $I, J \subset \mathbb{N}$ with $(\mathbf{N}^{-1})_{I,J} = f(\mathbf{M}_1, ..., \mathbf{M}_s)$, we have $(\mathbf{N}^{-1})_{I,J} = \mathbf{I}^{(n)}_{I,[n]}\mathbf{N}^{-1}\mathbf{I}^{(n)}_{[n],J}$ and thus

$$\widehat{\mathbf{N}} = \begin{bmatrix}\mathbf{N} & \mathbf{I}^{(n)}_{[n],J}\\ \mathbf{I}^{(n)}_{I,[n]} & \mathbf{0}\end{bmatrix} \text{ with } \det(f(\mathbf{M}_1, ..., \mathbf{M}_s)) = \det(\mathbf{I}^{(n)}_{I,[n]}\mathbf{N}^{-1}\mathbf{I}^{(n)}_{[n],J}) = \det(\widehat{\mathbf{N}})/\det(\mathbf{N}).$$

Thus, to maintain $\det(f(\mathbf{M}_1, ..., \mathbf{M}_s))$, we just need to maintain $\det(\widehat{\mathbf{N}})$ and $\det(\mathbf{N})$. Maintaining these determinants can be done via the determinant lemma, which states:

$$\det(\mathbf{N} + \mathbf{uv}^\top) = \det(\mathbf{N})(1 + \mathbf{v}^\top\mathbf{N}^{-1}\mathbf{u}). \tag{3}$$

Here adding $\mathbf{uv}^\top$ is a rank-1 update and can capture updates such as changing an entry of $\mathbf{N}$ (when $\mathbf{u}, \mathbf{v}$ have only 1 nonzero entry each) or changing a column of $\mathbf{N}$ (when v has only 1 nonzero entry). In particular, the task of maintaining $\det(\mathbf{N})$ reduces to the task of repeatedly computing $\mathbf{v}^\top\mathbf{N}^{-1}\mathbf{u}$. This is a dynamic matrix formula (since $\mathbf{u}, \mathbf{v}, \mathbf{N}$ change over time). For example, maintaining $\det(\mathbf{N})$ while $\mathbf{N}$ receives entry updates, requires us to

maintain $f(\mathbf{u}, \mathbf{v}, \mathbf{N}) = \mathbf{v}^\top \mathbf{N}^{-1}\mathbf{u}$ while $\mathbf{u}, \mathbf{v}, \mathbf{N}$ receive entry updates. Thus data structures for maintaining $\det(f(\mathbf{M}_1, ..., \mathbf{M}_s))$ (Theorem 2) are implied by data structures for dynamic matrix formula (Theorem 7), together with some additional error analysis performed in the full-version. A key observation in the error analysis is that the determinant is the product of the eigenvalues, and therefore, if we guarantee (with a sufficient number of bits) that the eigenvalues are preserved up to a multiplicative error factor of $1 \pm \frac{\epsilon}{10n}$, then we have determinant computation up to a multiplicative error factor of $1 \pm \epsilon$. We formalize this idea by bounding the determinant of a matrix perturbed by a small amount – see full version for details.

Maintaining the Rank. Let us assume that $\mathbf{M}_1, ..., \mathbf{M}_s$ are integer matrices, so $\mathbf{N}$ is an integer matrix as well. Note that w.h.p. $\text{rank}(\mathbf{N})$ is the same over $\mathbb{Z}$ and $\mathbb{Z}_p$ for prime $p \sim n^c$ and large enough constant c. So for the rank, we do not need to worry about rounding errors and can just focus on finite fields.

Sankowski [34] proved the following statement about matrix ranks. For any $n \times n$ matrix $\mathbf{M}$, and random $n \times n$ matrices $\mathbf{X}$ and $\mathbf{Y}$ (each entry is chosen uniformly at random from $\mathbb{Z}_p$), and $\mathbf{I}_k$ being a partial identity (the first k diagonal entries are 1, the remaining diagonal entries are 0), let

$$\overline{\mathbf{M}} = \begin{bmatrix} \mathbf{M} & \mathbf{X} & \mathbf{0} \\ \mathbf{Y} & \mathbf{0} & \mathbf{I}_n \\ \mathbf{0} & \mathbf{I}_n & \mathbf{I}_k \end{bmatrix}$$

Then with high probability, $\det(\overline{\mathbf{M}}) \neq 0 \iff \text{rank}(\mathbf{M}) \geq n - k$. In [34], this was used to maintain the rank of $\mathbf{M}$. We now generalize this to maintaining the rank of $f(\mathbf{M}_1, ..., \mathbf{M}_s)$.

Given a formula $f(\mathbf{M}_1, ..., \mathbf{M}_s)$, let $g(\mathbf{M}_1, ..., \mathbf{M}_s, \mathbf{P}, \mathbf{Q}, \mathbf{R}_k) = \mathbf{P} f(\mathbf{M}_1, ..., \mathbf{M}_s)\mathbf{Q} + \mathbf{R}_k$ where

$$\mathbf{P} = \mathbf{Q} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}, \quad \mathbf{R}_k = \begin{bmatrix} \mathbf{0} & \mathbf{X} & \mathbf{0} \\ \mathbf{Y} & \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{I} & \mathbf{I}_k \end{bmatrix},$$

we have

$$g(\mathbf{M}_1, ..., \mathbf{M}_s, \mathbf{P}, \mathbf{Q}, \mathbf{R}_k) = \begin{bmatrix} f(\mathbf{M}_1, ..., \mathbf{M}_s) & \mathbf{X} & \mathbf{0} \\ \mathbf{Y} & \mathbf{0} & \mathbf{I}_n \\ \mathbf{0} & \mathbf{I}_n & \mathbf{I}_k \end{bmatrix}$$

Thus, $\det(g(\mathbf{M}_1, ..., \mathbf{M}_s, \mathbf{P}, \mathbf{Q}, \mathbf{R}_k)) \neq 0 \iff \text{rank}(f(\mathbf{M}_1, ..., \mathbf{M}_s)) \geq n - k$. So we can track the rank of $f(\mathbf{M}_1, ..., \mathbf{M}_s)$ by finding and maintaining the smallest k where $\det(g(\mathbf{M}_1, ..., \mathbf{M}_s, \mathbf{P}, \mathbf{Q}, \mathbf{R}_k)) \neq 0$. Note that with each changed entry to any $\mathbf{M}_i$, the rank can change by at most 1. So we can simply try increasing/decreasing k by performing a single entry update to $\mathbf{R}_k$ (and potentially reverting the update) to check if the $\det(g(\mathbf{M}_1, ..., \mathbf{M}_s, \mathbf{P}, \mathbf{Q}, \mathbf{R}_k))$ becomes 0. Note that the determinant lemma (3) breaks once the matrix is no longer invertible.

Thus we must increase k whenever $\det(g(\mathbf{M}_1, ..., \mathbf{M}_s, \mathbf{P}, \mathbf{Q}, \mathbf{R}_k)) = 0$. If an update causes the determinant to become 0, we must revert that update by reverting any internal changes of the data structure, then increase k, and then perform the reverted update again. This way, we always guarantee $\det(g(\mathbf{M}_1, ..., \mathbf{M}_s, \mathbf{P}, \mathbf{Q}, \mathbf{R}_k)) \neq 0$ and that (3) never breaks. By always maintaining the largest k where $\det(g(\mathbf{M}_1, ..., \mathbf{M}_s, \mathbf{P}, \mathbf{Q}, \mathbf{R}_k)) \neq 0$, we know $n - k$ is the rank of $f(\mathbf{M}_1, ..., \mathbf{M}_s)$.

3 Dynamic Matrix Formula

In this section, we prove the first main result: a generic data structure that can maintain the evaluation of any matrix formula $f(\mathbf{M}_1, ..., \mathbf{M}_s)$ while supporting updates to the input matrices.

▶ **Theorem 4** (Detailed variant of Theorem 1). *Suppose we are given a matrix formula $f(\mathbf{M}_1, \ldots, \mathbf{M}_s)$ with respective input matrices $\mathbf{M}_1, \ldots \mathbf{M}_s$, where $\|\mathbf{M}_i\|_{\mathrm{F}} \leq \kappa$ for all $i \in [s]$. Let n denote the sum of the number of rows and columns of all $\mathbf{M}_1, \ldots, \mathbf{M}_s$. We further assume that the result of each inversion within f also has Frobenius-norm bounded by κ: in other words, we assume that every internal inversion-node of the computation tree has a bounded condition number. Then, for $\epsilon > 0, \kappa > n$ and any $0 \leq \nu \leq \mu \leq 1$, there exists data structures that are each initialized in time $\widetilde{O}(n^{\omega} s \log(\kappa/\epsilon))$ and have the following operations.*

The data structures have the following update and query operations (where each bullet is a different data structure)

- *Support entry updates and entry queries in $\widetilde{O}((n^{\omega(1,1,\mu)-\mu} + n^{\omega(1,\mu,\nu)-\nu} + n^{\mu+\nu}) s \log(\kappa/\epsilon))$ time. This is $\widetilde{O}(n^{1.405} s \log(\kappa/\epsilon))$ for $\nu \approx 0.543, \mu \approx 0.8612$.*
- *Support entry updates in $\widetilde{O}((n^{\omega(1,1,\mu)-\mu} + n^{1+\mu}) s \log(\kappa/\epsilon))$ time and entry queries in $O(n^{\mu} s \log(\kappa/\epsilon))$ time. This is $\widetilde{O}(n^{1.528} s \log(\kappa/\epsilon))$ and $O(n^{0.528} s \log(\kappa/\epsilon))$ for $\mu \approx 0.528$.*
- *Support column updates and row queries in $\widetilde{O}((n^{\omega(1,1,\mu)-\mu} + n^{1+\mu}) s \log(\kappa/\epsilon))$ time. This is $\widetilde{O}(n^{1.528} s \log(\kappa/\epsilon))$ for $\mu \approx 0.528$.*
- *Support rank-1 updates and returning all entries of $f(\mathbf{M}_1, ..., \mathbf{M}_s)$ in $\widetilde{O}(n^2 s \log(\kappa/\epsilon))$ time.*
- *Support column updates and row queries in the offline model (the entire sequence of column indices and row queries is given at the start) in $\widetilde{O}(n^{\omega-1} s \log(\kappa/\epsilon))$ update and query time.*

The outputs are all ϵ-approximate, i.e. each entry is off by at most an additive ϵ.

The explicit upper bound exponents were obtained via the tool of [7] and use the upper bounds on fast matrix multiplication by [37].

This result is obtained by reducing the task to the special case $g(\mathbf{N}) = \mathbf{N}^{-1}$, where the structure of matrix $\mathbf{N}$ depends on the formula f and its inputs $\mathbf{M}_1, ..., \mathbf{M}_s$.

We then run data structures (Theorem 7) that can maintain $\mathbf{N}^{-1}$ while supporting updates to $\mathbf{N}$. The accuracy of these data structures depends on $\|\mathbf{N}\|_F$ and $\|\mathbf{N}^{-1}\|_F$, so we must bound these Frobenius-norms. These bounds are given in Section 3.1. We then combine the bounds with the previous reduction to obtain Theorem 1 in Section 3.2.

3.1 Norm Bounds on Construction

In this section, we bound the Frobenius norm of the matrix produced by the reduction of [8], as well as its inverse. Formally, [8] has proven the following.

▶ **Theorem 5** (Theorem 3.1 of [9]). *Given a matrix formula $f(\mathbf{A}_1, ..., \mathbf{A}_p)$ over field $\mathbb{F}$, define $n := \sum_{i \in V} n_i + m_i$ where $n_i \times m_i$ is the dimension of $\mathbf{A}_i$.*

Then there exists a square matrix $\mathbf{N}$ of size at most $O(n) \times O(n)$, where each $\mathbf{A}_i$ is a block of $\mathbf{N}$. Further, if $f(\mathbf{A}_1, ..., \mathbf{A}_p)$ does not attempt to invert a non-invertible matrix then $(\mathbf{N}^{-1})_{I,J} = f(\mathbf{A}_1, \ldots, \mathbf{A}_p)$. Constructing $\mathbf{N}$ and I, J takes $O(n^2)$ time.

In the following Lemma 6 we retread the construction of matrix $\mathbf{N}$ and bound the Frobenius-norm.

▶ **Lemma 6.** *Let $f(\mathbf{A}_1, \ldots, \mathbf{A}_p)$ be a matrix formula over $\mathbb{R}$ using s gates. Suppose matrix $\mathbf{N}$, and index sets I, and J are constructed as in Theorem 5 so that $(\mathbf{N}^{-1})_{I,J} = f(\mathbf{A}_1, \ldots, \mathbf{A}_p)$. Let $\kappa \geq \max_i n_i + m_i \geq 2$, where $n_i \times m_i$ are the dimensions of $\mathbf{A}_i$.*

Then, if $\|\mathbf{A}_1\|_{\mathrm{F}}, \ldots, \|\mathbf{A}_p\|_{\mathrm{F}} \leq \kappa$, and the Frobenius norms of outputs of intermediate inverse gates are also bounded by κ, we have

$$\begin{aligned} \|\mathbf{N}\|_{\mathrm{F}} &\leq \kappa^s \\ \|\mathbf{N}^{-1}\|_{\mathrm{F}} &\leq (10\kappa)^{2s+1}. \end{aligned}$$

Proof. We bound $\|\mathbf{N}\|_{\mathrm{F}}$ and $\|\mathbf{N}^{-1}\|_{\mathrm{F}}$ by induction on the number of gates s. Note that the given formula f can be represented as a tree, where the input matrices are leaves and each operation is an internal node. For example $f(\mathbf{M}_1 ... \mathbf{M}_p) = g(\mathbf{M}_1, ..., \mathbf{M}_q) + h(\mathbf{M}_{q+1}, ..., \mathbf{M}_p)$ can be seen as a tree where the root node is a "+" with two subtrees for the formulas g and h. Each node that represents an operation has 2 children (or 1 child in case of inversion). We call the nodes also gates, e.g., inversion gate or addition gate, depending on what operation they represent.

Theorem 5 ([8]) constructs the matrix $\mathbf{N}$ by induction over the number of gates, i.e., for each gate w some matrix $\mathbf{N}_w$ is constructed. This $\mathbf{N}_w$ is constructed as a block matrix where some blocks are $\mathbf{N}_u, \mathbf{N}_v$ where u, v are the child gates of w. We also say that "$\mathbf{N}_w$ is returned by gate w".

Suppose matrix $\mathbf{N}$, and index sets I, and J are constructed as in Theorem 5 so that $(\mathbf{N}^{-1})_{I,J} = f(\mathbf{A}_1, \ldots, \mathbf{A}_p)$. We will show by induction on $s \geq 1$ (the number of gates in f) that:

$$\begin{aligned} \|\mathbf{N}\|_{\mathrm{F}} &\leq \kappa^s \\ \|(\mathbf{N}^{-1})_{I,J}\|_{\mathrm{F}} &\leq 2s\kappa \\ \|(\mathbf{N}^{-1})_{I,}\|_{\mathrm{F}} &\leq (5\kappa)^s \\ \|(\mathbf{N}^{-1})_{,J}\|_{\mathrm{F}} &\leq (5\kappa)^s \\ \|\mathbf{N}^{-1}\|_{\mathrm{F}} &\leq (10\kappa)^{2s+1} \end{aligned}$$

We now prove bounds on the output of our gates by assuming the induction hypothesis that their inputs have bounded norms. We start with the base case.

Input gate. The base case is when $s = 1$, in which case the formula $f(\mathbf{M}_v) = \mathbf{M}_v$ consists of a single input gate. The construction by [8] for Theorem 5 defines $\mathbf{N}$ as

$$\mathbf{N}^{-1} = \mathbf{N} = \begin{bmatrix} \mathbf{I}^{(n_v)} & \mathbf{M}_v \\ \mathbf{0}^{(m_v, n_v)} & -\mathbf{I}^{(m_v)} \end{bmatrix}$$

where $\mathbf{M}_v$ is the input matrix to the formula f and $n_v \times m_v$ are its dimensions. Selecting rows with indices in $I = \{1, ..., m_v\}$, and columns with indices in $J = \{n_v + 1, ..., n_v + m_v\}$, we get

$$\begin{aligned} (\mathbf{N}^{-1})_{,J} &= \begin{bmatrix} \mathbf{M}_v \\ -\mathbf{I}^{(m_v)} \end{bmatrix}, \\ (\mathbf{N}^{-1})_{I,} &= \begin{bmatrix} \mathbf{I}^{(n_v)} & \mathbf{M}_v \end{bmatrix}, \\ (\mathbf{N}^{-1})_{I,J} &= \begin{bmatrix} \mathbf{M}_v \end{bmatrix} \end{aligned}$$

Applying the triangle inequality to

$$\mathbf{N} = \begin{bmatrix} \mathbf{I}^{(n_v)} & \mathbf{0}^{(n_v,m_v)} \\ \mathbf{0}^{(m_v,n_v)} & -\mathbf{I}^{(m_v)} \end{bmatrix} + \begin{bmatrix} \mathbf{0}^{(n_v,n_v)} & \mathbf{M}_v \\ \mathbf{0}^{(m_v,n_v)} & \mathbf{0}^{(m_v,m_v)} \end{bmatrix}$$

and using the fact that $\sqrt{n_v + m_v} \leq \kappa$ and $\|\mathbf{M}_v\|_{\mathrm{F}} \leq \kappa$, we get

$$\begin{aligned} \|\mathbf{N}\|_{\mathrm{F}} &\leq \sqrt{n_v + m_v} + \|\mathbf{M}_v\|_{\mathrm{F}} \\ &\leq 2\kappa \end{aligned}$$

Similarly, we have that

$$\begin{aligned} \|(\mathbf{N}^{-1})_{I,J}\| &\leq \|\mathbf{M}_v\|_{\mathrm{F}} \leq \kappa^1 \\ \|(\mathbf{N}^{-1})_{I,}\| &\leq \|\mathbf{M}_v\|_{\mathrm{F}} + \sqrt{n_v} \leq 2\kappa \leq (5\kappa)^1 \\ \|(\mathbf{N}^{-1})_{,J}\| &\leq \|\mathbf{M}_v\|_{\mathrm{F}} + \sqrt{m_v} \leq 2\kappa \leq (5\kappa)^1 \\ \|\mathbf{N}^{-1}\|_{\mathrm{F}} &\leq \sqrt{n_v + m_v} + \|\mathbf{M}_v\|_{\mathrm{F}} \leq 2\kappa \leq (10\kappa)^3 \end{aligned}$$

We now consider the operation gates for the inductive step.

Inversion. Suppose the root gate is an inversion gate. Suppose $\mathbf{N}'$ is a $n_{N'} \times n_{N'}$ matrix and $I', J' \subset \mathbb{Z}$ are sets, such that $(\mathbf{N}'^{-1})_{I',J'}$ is the $n_w \times n_w$ matrix that the child gate w returns. The child gate is the root of a subtree with $a = s - 1 \geq 1$ gates, which implies bounds on the Frobenius-norm of $\mathbf{N}'$. The construction of Theorem 5 defines

$$\mathbf{N} = \begin{bmatrix} \mathbf{N}' & -\mathbf{I}^{(n_{N'})}_{,J'} \\ \mathbf{I}^{(n_{N'})}_{I',} & \mathbf{0}^{(n_w,n_w)} \end{bmatrix}$$

By block matrix inversion (Section 1.2), we have

$$\mathbf{N}^{-1} = \begin{bmatrix} \mathbf{N}'^{-1} - (\mathbf{N}'^{-1})_{,J'}(\mathbf{N}'^{-1})^{-1}_{I',J'}(\mathbf{N}'^{-1})_{I',} & (\mathbf{N}'^{-1})_{,J'}(\mathbf{N}'^{-1})^{-1}_{I',J'} \\ -(\mathbf{N}'^{-1})^{-1}_{I',J'}(\mathbf{N}'^{-1})_{I',} & (\mathbf{N}'^{-1})^{-1}_{I',J'} \end{bmatrix}$$

Selecting the rows with indices in I, and columns with indices in J, we get

$$\begin{aligned} (\mathbf{N}^{-1})_{I,} &= \begin{bmatrix} -(\mathbf{N}'^{-1})^{-1}_{I',J'}(\mathbf{N}'^{-1})_{I',} & (\mathbf{N}'^{-1})^{-1}_{I',J'} \end{bmatrix} \\ (\mathbf{N}^{-1})_{,J} &= \begin{bmatrix} (\mathbf{N}'^{-1})_{,J'}(\mathbf{N}'^{-1})^{-1}_{I',J'} \\ (\mathbf{N}'^{-1})^{-1}_{I',J'} \end{bmatrix} \\ (\mathbf{N}^{-1})_{I,J} &= \begin{bmatrix} (\mathbf{N}'^{-1})^{-1}_{I',J'} \end{bmatrix} \end{aligned}$$

By the triangle inequality and the assumption that $\kappa \geq n_w \geq 1$,

$$\begin{aligned} \|\mathbf{N}\|_{\mathrm{F}} &\leq \|\mathbf{N}'\|_{\mathrm{F}} + \sqrt{2n_w} \\ &\leq 2a\kappa + 2\kappa \leq 2s\kappa \end{aligned}$$

By the assumption that the output of each inversion gate has Frobenius norm at most κ,

$$\|(\mathbf{N}^{-1})_{I,J}\|_{\mathrm{F}} = \|(\mathbf{N}'^{-1})^{-1}_{I',J'}\|_{\mathrm{F}} \leq \kappa \leq \kappa^s$$

For the remaining matrices, we can bound their Frobenius norms by the sum of Frobenius norms of their blocks, use the triangle inequality to split sums, and $\|\mathbf{AB}\|_{\mathrm{F}} \leq \|\mathbf{A}\|_{\mathrm{F}}\|\mathbf{B}\|_{\mathrm{F}}$ to split products, and then bound the resulting terms by the inductive hypothesis:

$$\begin{aligned}
\|(\mathbf{N}^{-1})_{I,}\|_{\mathrm{F}} &\leq \|(\mathbf{N}'^{-1})^{-1}_{I',J'}\|_{\mathrm{F}}(1 + \|(\mathbf{N}'^{-1})_{I',}\|_{\mathrm{F}}) \\
&\leq \kappa(1 + (5\kappa)^a) \leq \kappa \cdot 2(5\kappa)^a \leq (5\kappa)^{a+1} = (5\kappa)^s \\
\|(\mathbf{N}^{-1})_{,J}\|_{\mathrm{F}} &\leq \|(\mathbf{N}'^{-1})^{-1}_{I',J'}\|_{\mathrm{F}}(1 + \|(\mathbf{N}'^{-1})_{,J'}\|_{\mathrm{F}}) \\
&\leq \kappa(1 + (5\kappa)^a) \leq \kappa \cdot 2(5\kappa)^a \leq (5\kappa)^{a+1} = (5\kappa)^s \\
\|\mathbf{N}^{-1}\|_{\mathrm{F}} &\leq \|\mathbf{N}'^{-1}\|_{\mathrm{F}} + \|(\mathbf{N}'^{-1})^{-1}_{I',J'}\|_{\mathrm{F}}(\|(\mathbf{N}'^{-1})_{I',}\|_{\mathrm{F}} + 1)(\|(\mathbf{N}'^{-1})_{,J'}\|_{\mathrm{F}} + 1) \\
&\leq (10\kappa)^{2a+1} + \kappa((5\kappa)^a + 1)^2 \leq (10\kappa)^{2a+1} + \kappa(10\kappa)^{2a} \\
&\leq (10\kappa)^{2a+3} = (10\kappa)^{2s+1}
\end{aligned}$$

Addition and Subtraction. Suppose the root gate w is an addition gate, adding two $n_w \times m_w$ matrices. Suppose the subtree of the left child has $a \geq 1$ gates and the subtree of the right child has $b \geq 1$ gates, where $s = a + b + 1$. Let $\mathbf{L}$ be the $n_L \times n_L$ matrix and $\mathbf{R}$ be the a $n_R \times n_R$ matrix returned by the child gates, where $(\mathbf{L}^{-1})_{I_L,J_L}$ and $(\mathbf{R}^{-1})_{I_R,J_R}$ are $n_w \times m_w$ matrices. The matrix $\mathbf{N}$ for parent (addition) gate w is defined as

$$\mathbf{N} = \begin{bmatrix} \mathbf{L} & \mathbf{0} & \mathbf{I}^{(n_L)}_{,J_L} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} & \mathbf{I}^{(n_R)}_{,J_R} & \mathbf{0} \\ \mathbf{I}^{(n_L)}_{I_L,} & \mathbf{I}^{(n_R)}_{I_R,} & \mathbf{0} & \mathbf{I}^{(n_w)} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}^{(m_w)} & \mathbf{0} \end{bmatrix}$$

$$\mathbf{N}^{-1} = \begin{bmatrix} \mathbf{L}^{-1} & \mathbf{0} & \mathbf{0} & -(\mathbf{L}^{-1})_{,J_L} \\ \mathbf{0} & \mathbf{R}^{-1} & \mathbf{0} & -(\mathbf{R}^{-1})_{,J_R} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}^{(m_w)} \\ -(\mathbf{L}^{-1})_{I_L,} & -(\mathbf{R}^{-1})_{I_R,} & \mathbf{I}^{(n_w)} & (\mathbf{L}^{-1})_{I_L,J_L} + (\mathbf{R}^{-1})_{I_R,J_R} \end{bmatrix}$$

Selecting the rows of $\mathbf{N}^{-1}$ with indices in I and columns with indices in J, we get

$$(\mathbf{N}^{-1})_{I,} = \begin{bmatrix} -(\mathbf{L}^{-1})_{I_L,} & -(\mathbf{R}^{-1})_{I_R,} & \mathbf{I}^{(n_w)} & (\mathbf{L}^{-1})_{I_L,J_L} + (\mathbf{R}^{-1})_{I_R,J_R} \end{bmatrix}$$

$$(\mathbf{N}^{-1})_{,J} = \begin{bmatrix} -(\mathbf{L}^{-1})_{,J_L} \\ -(\mathbf{R}^{-1})_{,J_R} \\ \mathbf{I}^{(m_w)} \\ (\mathbf{L}^{-1})_{I_L,J_L} + (\mathbf{R}^{-1})_{I_R,J_R} \end{bmatrix}$$

$$(\mathbf{N}^{-1})_{I,J} = \begin{bmatrix} (\mathbf{L}^{-1})_{I_L,J_L} + (\mathbf{R}^{-1})_{I_R,J_R} \end{bmatrix}$$

We now bound the Frobenius norms of these matrices using the bounds on $\|\mathbf{L}\|_{\mathrm{F}}$, $\|\mathbf{L}^{-1}\|_{\mathrm{F}}$, $\|(\mathbf{L}^{-1})_{I_L,}\|_{\mathrm{F}}$, $\|(\mathbf{L}^{-1})_{,J_L}\|_{\mathrm{F}}$, $\|(\mathbf{L}^{-1})_{I_L,J_L}\|_{\mathrm{F}}$, and similarly for $\mathbf{R}^{-1}$, that we get from the inductive hypothesis. Using the triangle inequality and the assumption that $\kappa \geq n_w + m_w$, and that $\|\mathbf{L}\|_{\mathrm{F}} \leq 2a\kappa$ and $\|\mathbf{R}\|_{\mathrm{F}} \leq 2b\kappa$ by the inductive hypothesis,

$$\begin{aligned}
\|\mathbf{N}\|_{\mathrm{F}} &\leq \|\mathbf{L}\|_{\mathrm{F}} + \|\mathbf{R}\|_{\mathrm{F}} + \sqrt{3n_w + 3m_w} \\
&\leq 2a\kappa + 2b\kappa + 2\kappa \leq 2(a + b + 1)\kappa = 2s\kappa
\end{aligned}$$

Similarly, we get:

$$\begin{aligned}
\|(\mathbf{N}^{-1})_{I,J}\|_{\mathrm{F}} &\le \|(\mathbf{L}^{-1})_{I_L,J_L}\|_{\mathrm{F}} + \|(\mathbf{R}^{-1})_{I_R,J_R}\|_{\mathrm{F}} \\
&\le \kappa^a + \kappa^b \le 2\kappa^{a+b} \le \kappa^{a+b+1} = \kappa^s \\
\|(\mathbf{N}^{-1})_{I,}\|_{\mathrm{F}} &\le \|(\mathbf{L}^{-1})_{I_L,}\|_{\mathrm{F}} + \|(\mathbf{R}^{-1})_{I_R,}\|_{\mathrm{F}} + \|(\mathbf{L}^{-1})_{I_L,J_L}\|_{\mathrm{F}} + \|(\mathbf{R}^{-1})_{I_R,J_R}\|_{\mathrm{F}} + \sqrt{n_w} \\
&\le (5\kappa)^a + (5\kappa)^b + \kappa^a + \kappa^b + \kappa \le 5(5\kappa)^{a+b} \le (5\kappa)^{a+b+1} = (5\kappa)^s \\
\|(\mathbf{N}^{-1})_{,J}\|_{\mathrm{F}} &\le \|(\mathbf{L}^{-1})_{,J_L}\|_{\mathrm{F}} + \|(\mathbf{R}^{-1})_{,J_R}\|_{\mathrm{F}} + \|(\mathbf{L}^{-1})_{I_L,J_L}\|_{\mathrm{F}} + \|(\mathbf{R}^{-1})_{I_R,J_R}\|_{\mathrm{F}} + \sqrt{m_w} \\
&\le (5\kappa)^a + (5\kappa)^b + \kappa^a + \kappa^b + \kappa \le 5(5\kappa)^{a+b} \le (5\kappa)^{a+b+1} = (5\kappa)^s
\end{aligned}$$

$$\begin{aligned}
\|\mathbf{N}^{-1}\|_{\mathrm{F}} &\le \|\mathbf{L}^{-1}\|_{\mathrm{F}} + \|\mathbf{R}^{-1}\|_{\mathrm{F}} + \|(\mathbf{L}^{-1})_{I_L,}\|_{\mathrm{F}} + \|(\mathbf{L}^{-1})_{,J_L}\|_{\mathrm{F}} \\
&\qquad + \|(\mathbf{R}^{-1})_{I_R,}\|_{\mathrm{F}} + \|(\mathbf{R}^{-1})_{,J_R}\|_{\mathrm{F}} + \sqrt{n_w + m_w} \\
&\le (10\kappa)^{2a+1} + (10\kappa)^{2b+1} + 2(5\kappa)^a + 2(5\kappa)^b + \kappa \\
&\le 7(10\kappa)^{2a+2b+1} \le (10\kappa)^{2a+2b+3} = (10\kappa)^{2s+1}
\end{aligned}$$

Subtraction gates are the same as addition gates except that the $\mathbf{I}^{(n_R)}_{,J_R}$ in the second row, third column of $\mathbf{N}$ is replaced by $-\mathbf{I}^{(n_R)}_{,J_R}$. The norm-bounding computations are then the same except for irrelevant sign changes.

Multiplication. Suppose the root gate is a multiplication gate. Suppose the left child has $a \ge 1$ gates and the right child has $b \ge 1$ gates, where $s = a + b + 1$. Let $\mathbf{L}$ be the $n_L \times n_L$ matrix and $\mathbf{R}$ be the $n_R \times n_R$ matrix such that the outputs of the child gates are $(\mathbf{L}^{-1})_{I_L,J_L}$ and $(\mathbf{R}^{-1})_{I_R,J_R}$.

$$\begin{aligned}
\mathbf{N} &= \begin{bmatrix} \mathbf{L} & -\mathbf{I}^{(n_L)}_{[n_L],J_L}\mathbf{I}^{(n_R)}_{I_R,[n_R]} \\ \mathbf{0}^{(n_R,n_L)} & \mathbf{R} \end{bmatrix} \\
\mathbf{N}^{-1} &= \begin{bmatrix} \mathbf{L}^{-1} & (\mathbf{L}^{-1})_{,J_L}(\mathbf{R}^{-1})_{I_R,} \\ \mathbf{0} & \mathbf{R}^{-1} \end{bmatrix}
\end{aligned}$$

Selecting the rows of $\mathbf{N}^{-1}$ with indices in I is the same as taking the first row of blocks and left-multiplying by $\mathbf{I}^{(n_L)}_{I_L,}$. Selecting columns with indices in J is the same as taking the second column of blocks and right-multiplying by $\mathbf{I}^{(n_R)}_{,J_R}$. Hence,

$$\begin{aligned}
(\mathbf{N}^{-1})_{I,} &= \begin{bmatrix} (\mathbf{L}^{-1})_{I_L,} & (\mathbf{L}^{-1})_{I_L,J_L}(\mathbf{R}^{-1})_{I_R,} \end{bmatrix} \\
(\mathbf{N}^{-1})_{,J} &= \begin{bmatrix} (\mathbf{L}^{-1})_{,J_L}(\mathbf{R}^{-1})_{I_R,J_R} \\ (\mathbf{R}^{-1})_{,J_R} \end{bmatrix} \\
(\mathbf{N}^{-1})_{I,J} &= \begin{bmatrix} (\mathbf{L}^{-1})_{I_L,J_L}(\mathbf{R}^{-1})_{I_R,J_R} \end{bmatrix}
\end{aligned}$$

We again bound the Frobenius norms of these matrices using the bounds on $\|\mathbf{L}\|_{\mathrm{F}}$, $\|\mathbf{L}^{-1}\|_{\mathrm{F}}$, $\|(\mathbf{L}^{-1})_{I_L,}\|_{\mathrm{F}}$, $\|(\mathbf{L}^{-1})_{,J_L}\|_{\mathrm{F}}$, $\|(\mathbf{L}^{-1})_{I_L,J_L}\|_{\mathrm{F}}$, and similarly for $\mathbf{R}^{-1}$, that we get from the inductive hypothesis. The Frobenius norm of each block matrix is bounded by the sum of the Frobenius norms of its blocks. Using this together with the fact that $\|\mathbf{AB}\|_{\mathrm{F}} \le \|\mathbf{A}\|_{\mathrm{F}}\|\mathbf{B}\|_{\mathrm{F}}$, we get

$$\begin{aligned}
\|\mathbf{N}\|_{\mathrm{F}} &\leq \|\mathbf{L}\|_{\mathrm{F}} + \|\mathbf{R}\|_{\mathrm{F}} + \sqrt{\min(n_L, n_R)} \\
&\leq 5a\kappa + 5b\kappa + \kappa \leq 5(a+b+1)\kappa = 5s\kappa \\
\|(\mathbf{N}^{-1})_{I,J}\|_{\mathrm{F}} &\leq \|(\mathbf{L}^{-1})_{I_L,J_L}\|_{\mathrm{F}}\|(\mathbf{R}^{-1})_{I_R,J_R}\|_{\mathrm{F}} \\
&\leq \kappa^a\kappa^b \leq \kappa^{a+b+1} = \kappa^s \\
\|(\mathbf{N}^{-1})_{I,}\|_{\mathrm{F}} &\leq \|(\mathbf{L}^{-1})_{I_L,}\|_{\mathrm{F}} + \|(\mathbf{L}^{-1})_{I_L,J_L}\|_{\mathrm{F}}\|(\mathbf{R}^{-1})_{I_R,}\|_{\mathrm{F}} \\
&\leq (5\kappa)^a + \kappa^a(5\kappa)^b \leq (5\kappa)^{a+b} + (5\kappa)^{a+b} \leq (5\kappa)^{a+b+1} = (5\kappa)^s \\
\|(\mathbf{N}^{-1})_{,J}\|_{\mathrm{F}} &\leq \|(\mathbf{R}^{-1})_{,J_R}\|_{\mathrm{F}} + \|(\mathbf{R}^{-1})_{I_R,J_R}\|_{\mathrm{F}}\|(\mathbf{L}^{-1})_{,J_L}\|_{\mathrm{F}} \\
&\leq (5\kappa)^b + \kappa^b(5\kappa)^a \leq (5\kappa)^{a+b} + (5\kappa)^{a+b} \leq (5\kappa)^{a+b+1} = (5\kappa)^s \\
\|\mathbf{N}^{-1}\|_{\mathrm{F}} &\leq \|\mathbf{L}^{-1}\|_{\mathrm{F}} + \|\mathbf{R}^{-1}\|_{\mathrm{F}} + \|(\mathbf{L}^{-1})_{,J_L}\|_{\mathrm{F}}\|(\mathbf{R}^{-1})_{I_R,}\|_{\mathrm{F}} \\
&\leq (10\kappa)^{2a+1} + (10\kappa)^{2b+1} + (5\kappa)^a(5\kappa)^b \\
&\leq 3(10\kappa)^{2a+2b+1} \leq (10\kappa)^{2a+2b+3} = (10\kappa)^{2s+1}
\end{aligned}$$

Then by the inductive hypothesis, the claim is proven. ◀

3.2 Proof of Theorem 1

To obtain Theorem 1, we will use the following data structures (Theorem 7) by Sankowski [33], v.d.Brand, Nanongkai and Saranurak [12]. This previous work only considered finite fields or the real-RAM model, i.e., infinite precision with $O(1)$ time per arithmetic operation. In the full version, we prove that these data structures also work with finite precision and $\widetilde{O}(\log(\kappa/\epsilon))$-bit fixed-point arithmetic, as stated in Theorem 7.

▶ **Theorem 7.** *There exist several dynamic matrix inverse algorithms with the following operations. For any update vs. query time trade-off parameters $0 \leq \nu \leq \mu \leq 1$, each data structure initializes in $O(n^\omega \log \kappa/\epsilon)$ time on given accuracy parameters $\epsilon > 0, \kappa > n$, and dynamic matrix $\mathbf{Z} \in \mathbb{R}^{n\times n}$ that is promised to stay invertible throughout all updates with $\|\mathbf{Z}\|_F, \|\mathbf{Z}^{-1}\|_F \leq \kappa$.*

The data structures have the following update/query operations

1. *Support entry updates and entry queries in $\widetilde{O}((n^{\omega(1,1,\mu)-\mu} + n^{\omega(1,\mu,\nu)-\nu} + n^{\mu+\nu})s\log(\kappa/\epsilon))$ time. This is $\widetilde{O}(n^{1.405}s\log(\kappa/\epsilon))$ for $\nu \approx 0.543, \mu \approx 0.8612$.*
2. *Support entry updates in $\widetilde{O}((n^{\omega(1,1,\mu)-\mu} + n^{1+\mu})s\log(\kappa/\epsilon))$ time and entry queries in $O(n^\mu s\log(\kappa/\epsilon))$ time. This is $\widetilde{O}(n^{1.528}s\log(\kappa/\epsilon))$ and $O(n^{0.528}s\log(\kappa/\epsilon))$ for $\mu \approx 0.528$.*
3. *Support rank-1 updates and returning all entries of $f(\mathbf{M}_1, ..., \mathbf{M}_s)$ in $\widetilde{O}(n^2 s\log(\kappa/\epsilon))$ time.*
4. *Support column updates and row queries in the offline model (the entire sequence of column indices and row queries is given at the start) in $\widetilde{O}(n^{\omega-1}s\log(\kappa/\epsilon))$ update and query time.*

The outputs are all ϵ-approximate, i.e., each entry is off by at most an additive $\pm\epsilon$.

By running the data structures of Theorem 7 on the matrix obtained from Theorem 5, we obtain Theorem 1.

Proof of Theorem 4 (Theorem 1). Given the formula $f(\mathbf{M}_1, ..., \mathbf{M}_s)$ and its input matrices $\mathbf{M}_1, ..., \mathbf{M}_s$, where each $\mathbf{M}_i$ is of size $n_i \times m_i$, let $n = \sum_s n_i + m_i$.

Initialization. By Theorem 5, we can construct in $O(n^2)$ time a square $O(n) \times O(n)$ matrix, where each $\mathbf{M}_i$ is a sub-block of $\mathbf{N}$, and two sets $I, J \subset \mathbb{Z}$ with $(\mathbf{N}^{-1})_{I,J} = f(\mathbf{M}_1, ..., \mathbf{M}_s)$.

The assumption of Theorem 4 states that each $\|\mathbf{M}_i\|_F \leq \kappa$ and each result of an inversion gate within f also has Frobenius norm bounded by κ. Thus, by Lemma 6, we have $\log \|\mathbf{N}\|_F$ and $\log \|\mathbf{N}^{-1}\|_F$ bounded by $O(s \log \kappa)$.

Depending on the type of update and query that we want (i.e., entry, column, row, etc), we run the respective data structure from Theorem 7 on $\mathbf{N}$.

In total, the initialization takes $\widetilde{O}(n^\omega s \log(\kappa/\epsilon))$ time, dominated by the initialization of the data structure from Theorem 7.

Updates and Queries. Since each $\mathbf{M}_i$ is a submatrix of $\mathbf{N}$, entry, column, row, or rank-1 updates to any $\mathbf{M}_i$ can be modeled by an entry, column, row, or rank-1 update to $\mathbf{N}$.

Likewise, querying an entry, a row, or column of $f(\mathbf{M}_1, ..., \mathbf{M}_s)$ can be performed by querying an entry, or row, or column of $\mathbf{N}^{-1}$, because submatrix $(\mathbf{N}^{-1})_{I,J} = f(\mathbf{M}_1, ..., \mathbf{M}_s)$. Further, the queries all have accuracy ϵ by Theorem 7.

Thus, the update and query complexity of Theorem 1 is exactly as stated in Theorem 7. ◀

4 Conclusion

We discussed the bit complexity and stability of maintaining arbitrary matrix formulas (with inversion, multiplication, and addition/subtraction) and their determinants. In addition, we provided data structures for maintaining the rank of matrices under finite fields and discussed a few applications for these. We believe our data structures and analysis would provide a useful and easy-to-use toolbox for designing iterative algorithms under the word-RAM model. For example, to extend optimization algorithms to the word-RAM model, one is only required to provide an analysis of what amount of errors in each step can be tolerated without affecting the convergence. Some other applications are in computational geometry and computer algebra problems.

A compelling future direction is to analyze the bit complexity of more complex algorithms that use algebraic and matrix formulas and the required error bounds for these algorithms. One interesting example is the Gram-Schmidt walk introduced for discrepancy minimization [4] that has many applications including experimental design [25]. It is not clear how many bits are required to guarantee such a random walk constructs a good distribution.

Another compelling direction is to investigate whether our bit complexity bounds can be improved for certain problems. For example, in the basic solution application, we presented bounds that depend on both the maximum determinant and maximum condition number over all d-by-d submatrices. We know that the maximum determinant is small for combinatorial problems and the maximum condition number is small for random matrices with high probability. Therefore it would be interesting to investigate whether one of these terms can be eliminated from the bit complexity bound. Another case of special problems is in tensor Tucker decomposition, where linear regression problems with Kronecker structure are solved [18, 21]. Although the matrices involved have a condition number exponential in the order of the tensor, such matrices are usually not constructed explicitly.

Finally, our results hint that inverse maintenance approaches might not be as unstable as previously assumed. It would be interesting to investigate their performance in practice. This might require modifications to plain vanilla Sherman-Morrison-Woodbury identity.

References

1 Deeksha Adil, Rasmus Kyng, Richard Peng, and Sushant Sachdeva. Iterative Refinement for ℓ_p-norm Regression. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1405–1424. SIAM, 2019. doi:10.1137/1.9781611975482.86.

2 Deeksha Adil, Richard Peng, and Sushant Sachdeva. Fast, Provably convergent IRLS algorithm for p-norm Linear Regression. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 14166–14177, 2019. URL: http://papers.nips.cc/paper/9565-fast-provably-convergent-irls-algorithm-for-p-norm-linear-regression.

3 Deeksha Adil and Sushant Sachdeva. Faster p-norm minimizing flows, via smoothed q-norm problems. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 892–910. SIAM, 2020. doi:10.1137/1.9781611975994.54.

4 Nikhil Bansal, Daniel Dadush, Shashwat Garg, and Shachar Lovett. The Gram-Schmidt Walk: A Cure for the Banaszczyk Blues. In *Proceedings of the 50th annual acm sigact symposium on theory of computing*, pages 587–597, 2018. URL: https://theoryofcomputing.org/articles/v015a021/v015a021.pdf.

5 Peter A. Beling and Nimrod Megiddo. Using fast matrix multiplication to find basic solutions. *Theoretical Computer Science*, 205(1):307–316, 1998. doi:10.1016/S0304-3975(98)00003-6.

6 Thiago Bergamaschi, Monika Henzinger, Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New techniques and fine-grained hardness for dynamic near-additive spanners. In *SODA*, pages 1836–1855. SIAM, 2021. URL: https://epubs.siam.org/doi/10.1137/1.9781611976465.110.

7 Jan van den Brand. Complexity term balancer. URL: https://www.ocf.berkeley.edu/~vdbrand/complexity/. Tool to balance complexity terms depending on fast matrix multiplication.

8 Jan van den Brand. A deterministic linear program solver in current matrix multiplication time. In *SODA*, pages 259–278. SIAM, 2020. URL: https://dl.acm.org/doi/abs/10.5555/3381089.3381105.

9 Jan van den Brand. Unifying matrix data structures: Simplifying and speeding up iterative algorithms. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 1–13. SIAM, 2021. arXiv:2010.13888.

10 Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 775–788. ACM, 2020. doi:10.1145/3357713.3384309.

11 Jan van den Brand and Danupon Nanongkai. Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 436–455. IEEE, 2019. arXiv:1909.10850.

12 Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *FOCS*, pages 456–480. IEEE Computer Society, 2019. arXiv:1905.05067.

13 Sébastien Bubeck, Michael B Cohen, Yin Tat Lee, and Yuanzhi Li. An homotopy method for lp regression provably beyond self-concordance and in input-sparsity time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1130–1137, 2018. arXiv:1711.01328.

14 Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623. IEEE, 2022. URL: https://ieeexplore.ieee.org/document/9996881.

15 Michael B. Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *J. ACM*, 68(1):3:1–3:39, 2021. doi:10.1145/3424305.

16 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, third edition edition, 2009. URL: https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/.

17 James Demmel, Ioana Dumitriu, and Olga Holtz. Fast linear algebra is stable. *Numerische Mathematik*, 108(1):59–91, 2007. doi:10.1007/s00211-007-0114-x.

18 Huaian Diao, Rajesh Jayaram, Zhao Song, Wen Sun, and David Woodruff. Optimal sketching for kronecker product regression and low rank approximation. *Advances in neural information processing systems*, 32, 2019.

19 Alan Edelman. Eigenvalues and condition numbers of random matrices. *SIAM journal on matrix analysis and applications*, 9(4):543–560, 1988.

20 Alan Edelman. *Eigenvalues and condition numbers of random matrices*. PhD thesis, Massachusetts Institute of Technology, 1989. URL: https://math.mit.edu/~edelman/homepage/papers/Eig.pdf.

21 Matthew Fahrbach, Gang Fu, and Mehrdad Ghadiri. Subquadratic kronecker regression with applications to tensor decomposition. In *Advances in Neural Information Processing Systems*, 2022.

22 Vissarion Fisikopoulos and Luis Mariano Peñaranda. Faster geometric algorithms via dynamic determinant computation. *Comput. Geom.*, 54:1–16, 2016. URL: https://www.sciencedirect.com/science/article/pii/S0925772115001261.

23 Mehrdad Ghadiri, Yin Tat Lee, Swati Padmanabhan, William Swartworth, David Woodruff, and Guanghao Ye. Improving the bit complexity of communication for distributed convex optimization. In *56th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2024.

24 Mehrdad Ghadiri, Richard Peng, and Santosh Vempala. The bit complexity of efficient continuous optimization. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2023. URL: https://www.computer.org/csdl/proceedings-article/focs/2023/189400c059/1T9796LmQ80.

25 Christopher Harshaw, Fredrik Sävje, Daniel Spielman, and Peng Zhang. Balancing covariates in randomized experiments with the gram-schmidt walk design. *arXiv preprint*, 2019. arXiv:1911.03071.

26 Baihe Huang, Shunhua Jiang, Zhao Song, Runzhou Tao, and Ruizhe Zhang. Solving SDP faster: A robust ipm framework and efficient implementation. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 233–244. IEEE, 2022. URL: https://www.computer.org/csdl/proceedings-article/focs/2022/551900a233/1JtvWgBUn8A.

27 Haotian Jiang, Tarun Kathuria, Yin Tat Lee, Swati Padmanabhan, and Zhao Song. A faster interior point method for semidefinite programming. In *2020 IEEE 61st annual symposium on foundations of computer science (FOCS)*, pages 910–918. IEEE, 2020. URL: https://ieeexplore.ieee.org/document/9317892.

28 Shunhua Jiang, Binghui Peng, and Omri Weinstein. The complexity of dynamic least-squares regression. In *FOCS*, 2023. URL: https://www.computer.org/csdl/proceedings-article/focs/2023/189400b605/1T972gjnp4I.

29 Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. A faster algorithm for solving general LPs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 823–832, 2021.

30 Yin Tat Lee and Aaron Sidford. Path finding I: Solving linear programs with $\widetilde{O}(\sqrt{\text{rank}})$ linear system solves. *arXiv preprint*, 2013. URL: https://ieeexplore.ieee.org/document/6979027, arXiv:1312.6677.

31 Yiheng Lin, James A Preiss, Emile Timothy Anand, Yingying Li, Yisong Yue, and Adam Wierman. Learning-augmented control via online adaptive policy selection: No regret via contractive perturbations. In *ACM SIGMETRICS, Workshop on Learning-augmented Algorithms: Theory and Applications 2023*, 2023. URL: https://learning-augmented-algorithms.github.io/papers/sigmetrics23-lata-posters-paper5.pdf.

32 Yiheng Lin, James A Preiss, Emile Timothy Anand, Yingying Li, Yisong Yue, and Adam Wierman. Online adaptive policy selection in time-varying systems: No-regret via contractive perturbations. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL: https://openreview.net/forum?id=hDajsofjRM.

33 Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse (extended abstract). In *FOCS*, pages 509–517. IEEE Computer Society, 2004. URL: https://ieeexplore.ieee.org/document/1366271.

34 Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 118–126, USA, 2007. Society for Industrial and Applied Mathematics. URL: https://dl.acm.org/doi/10.5555/1283383.1283397.

35 Terence Tao and Van Vu. On random ±1 matrices: singularity and determinant. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 431–440, 2005. URL: https://dl.acm.org/doi/10.1145/1060590.1060655.

36 Santosh S Vempala, Ruosong Wang, and David P Woodruff. The communication complexity of optimization. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1733–1752. SIAM, 2020.

37 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *SODA*, pages 3792–3835. SIAM, 2024.

38 Max A Woodbury. *Inverting modified matrices*. Statistical Research Group, 1950. URL: https://books.google.com/books/about/Inverting_Modified_Matrices.html?id=_zAnzgEACAAJ.

Approximate Counting for Spin Systems in Sub-Quadratic Time

Konrad Anand
School of Mathematical Sciences, Queen Mary University of London, London, UK

Weiming Feng
Institute for Theoretical Studies, ETH Zürich, Switzerland

Graham Freifeld
School of Informatics, University of Edinburgh, UK

Heng Guo
School of Informatics, University of Edinburgh, UK

Jiaheng Wang
School of Informatics, University of Edinburgh, UK

Abstract

We present two randomised approximate counting algorithms with $\widetilde{O}(n^{2-c}/\varepsilon^2)$ running time for some constant $c > 0$ and accuracy ε:

1. for the hard-core model with fugacity λ on graphs with maximum degree Δ when $\lambda = O(\Delta^{-1.5-c_1})$ where $c_1 = c/(2-2c)$;
2. for spin systems with strong spatial mixing (SSM) on planar graphs with quadratic growth, such as $\mathbb{Z}^2$.

For the hard-core model, Weitz's algorithm (STOC, 2006) achieves sub-quadratic running time when correlation decays faster than the neighbourhood growth, namely when $\lambda = o(\Delta^{-2})$. Our first algorithm does not require this property and extends the range where sub-quadratic algorithms exist.

Our second algorithm appears to be the first to achieve sub-quadratic running time up to the SSM threshold, albeit on a restricted family of graphs. It also extends to (not necessarily planar) graphs with polynomial growth, such as $\mathbb{Z}^d$, but with a running time of the form $\widetilde{O}\left(n^2\varepsilon^{-2}/2^{c(\log n)^{1/d}}\right)$ where d is the exponent of the polynomial growth and $c > 0$ is some constant.

2012 ACM Subject Classification Theory of computation → Randomness, geometry and discrete structures

Keywords and phrases Randomised algorithm, Approximate counting, Spin system, Sub-quadratic algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.11

Category Track A: Algorithms, Complexity and Games

Related Version *Preprint*: `https://arxiv.org/abs/2306.14867`

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 947778). Weiming Feng acknowledges support of Dr. Max Rössler, the Walter Haefner Foundation and the ETH Zürich Foundation.

Acknowledgements We would like to thank Chunyang Wang for pointing out how to shave a factor of e from Lemma 24.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 11; pp. 11:1–11:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The study of counting complexity was initiated by Valiant [33] with the introduction of the complexity class #**P**. An intriguing phenomenon emerging in counting complexity is that many #**P**-complete problems admit fully polynomial-time randomised approximation schemes (FPRAS), which output an ε-approximation in time polynomial in n and $1/\varepsilon$ with n being the input size. This is most commonly found for the so-called partition function of spin systems, as demonstrated by the pioneering work of Jerrum and Sinclair [23, 24]. Spin systems are physics models for nearest neighbour interactions, and the partition functions are the normalizing factors for their Gibbs distributions. This quantity can express the count of combinatorial objects such as the number of matchings, independent sets, or colourings in a graph, and is much more expressible by allowing real parameters of the system.

In this paper we are most interested in the fine-grained aspects of the complexity of estimating partition functions. While for most spin systems, exact counting is #**P**-hard [6], many of them admit efficient approximation schemes when strong spatial mixing (SSM) holds. Roughly speaking, SSM states that correlation or influence between vertices decays quickly as their distance increases (detailed definitions are given in Section 2 and Definition 7). When SSM fails, the partition function is usually **NP**-hard to approximate [31, 19, 18].

Efficient approximate counting was first enabled by the work of Jerrum, Valiant, and Vazirani [25] who gave self-reductions from approximate counting to sampling for a large class of problems. The sampling task is then most commonly solved via Markov chains. The efficiency of a Markov chain is measured by its mixing time (i.e. how long it takes to get close to the target distribution). For spin systems with SSM, in many situations, the standard chain, namely the Glauber dynamics, mixes in $O(n \log n)$ time [10, 2, 8, 9].

Another later technique, simulated annealing, provides a more efficient counting to sampling reduction [32, 22, 27]. Together with the $O(n \log n)$ mixing time mentioned above, this leads to $\widetilde{O}((n/\varepsilon)^2)$[1] approximate counting algorithms. These Markov chain Monte Carlo (MCMC) algorithms are the fastest for estimating partition functions in general, but $\Omega(n^2)$ appears to be a natural barrier to this approach. This is because generating a sample would take at least linear time (and there are $\Omega(n \log n)$ lower bounds for the mixing time of Markov chains [21] for many spin systems), and, restricted to the standard way of using the samples, the number of samples required for simulated annealing is at least $\Omega(n/\varepsilon^2)$ [27, Theorem 10].

On the other hand, when we relax the parameters, $\Omega(n^2)$ is no longer a barrier to algorithms. Let us take the hard-core gas model as an example. Here the Gibbs distribution μ is over the set $\mathcal{I}$ of independent sets of a graph G. For an independent set I, $\mu(I) := \lambda^{|I|}/Z(G)$, where λ is a parameter of the system (so-called fugacity), and $Z(G) := \sum_{I \in \mathcal{I}} \lambda^{|I|}$ is the partition function. For graphs with degree bound Δ, SSM holds when $\lambda < \lambda_c(\Delta) := \frac{(\Delta-1)^{\Delta-1}}{(\Delta-2)^{\Delta}} \approx \frac{e}{\Delta}$. The aforementioned MCMC results [10, 8, 9] imply FPRASes running in time $\widetilde{O}((n/\varepsilon)^2)$ as long as $\lambda < \lambda_c(\Delta)$. Yet much earlier, Weitz [34] gave the first fully polynomial-time approximation scheme (FPTAS, the deterministic counterpart to FPRAS) for the partition function of the hard-core model when $\lambda < \lambda_c(\Delta)$, which is not based on Markov chains. While Weitz's algorithm has a running time $n^{O(\log \Delta)}$ in general, it has an interesting feature that it gets faster as λ decreases. Roughly speaking, for $k > 0$ and $\lambda = O((1/\Delta)^{1+k})$, Weitz's FPTAS runs in time $O(n^{1+1/k}/\varepsilon^2)$. In particular, if $\lambda = o(\Delta^{-2})$, Weitz's algorithm passes the $\Omega(n^2)$ barrier, whereas the aforementioned MCMC method still takes $\Omega(n^2)$ time. This leads to an intriguing question:

When can we achieve sub-quadratic running time for approximate counting? (1)

[1] The notation $\widetilde{O}(\cdot)$ hides logarithmic factors.

In this paper we make some progress towards this question. For hard-core models, Weitz's algorithm uses the self-reduction [25] to reduce approximate counting to estimating marginal probabilities. We provide a quadratic speedup for the marginal estimation step for λ well below $\lambda_c(\Delta)$, albeit with the introduction of randomness. The result is summarized as follows.

▶ **Theorem 1.** *Fix a constant $k > 0$. Let $\Delta \geq 2$ be an integer and $\lambda < \frac{1}{\Delta^k(\Delta-1)}$. For graphs with maximum degree Δ, there exists an FPRAS for the partition function of the hard-core model with parameter λ in time $\widetilde{O}(n^{1+\frac{1}{2k}}/\varepsilon^2)$, where n is the number of vertices.*

▶ Remark 2 (Decay rate vs. neighbourhood growth). For a constant ε, the running time of Theorem 1 is sub-quadratic if $\lambda = o(\Delta^{-1.5})$, and $\widetilde{O}(n^{1.5})$ if $\lambda = O(\Delta^{-2})$. In contrast, to achieve sub-quadratic running-time, Weitz's algorithm requires $\lambda = o(\Delta^{-2})$, which is also the threshold when correlation decays faster than the growth of the neighbourhood. This threshold has algorithmic significance in other contexts [17, 1], but Theorem 1 implies that it is not essential to achieve sub-quadratic approximate counting.

Figure 1 is a sketch comparing the running times of MCMC,[2] Weitz's algorithm, and Theorem 1.[3] For the limiting case of $k = 0$, our algorithm works when $\lambda < \frac{1}{\Delta-1}$ and still presents a quadratic speedup comparing to Weitz's algorithm. However in this case the running time is $\left(\frac{n}{\varepsilon}\right)^{O(\log \Delta)}$ and thus our speedup is hidden in the big-O notation and is less significant. The parameter constraint $\frac{1}{\Delta-1}$ is imposed by the running-time tail bound of a subroutine we used, namely the recursive marginal sampler of Anand and Jerrum [1].

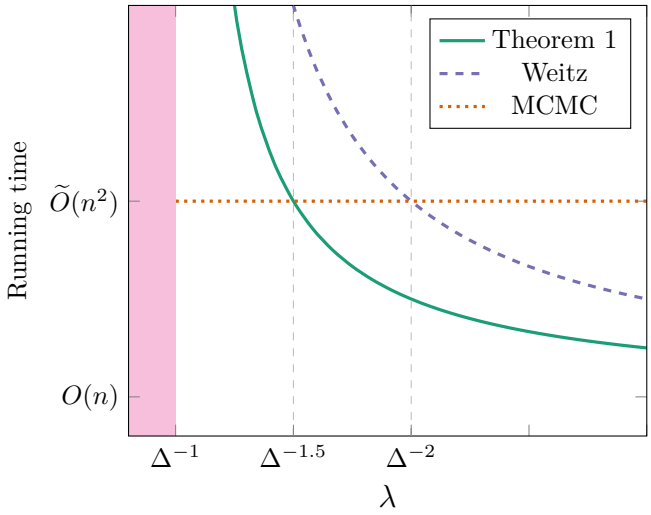


Figure 1 Running time comparison among MCMC, Weitz's algorithm, and Theorem 1.

The key to our method is to find a new estimator of the marginal probability that simultaneously has low variance and can be evaluated very fast. Our technique combines Weitz's self-avoiding walk (SAW) tree construction and the $\widetilde{O}(1)$ marginal sampler of Anand

[2] The running time of MCMC usually also depends on the parameter λ, but changing λ does not change the exponent of n. The effect of λ is usually a small polynomial factor hidden in the $\widetilde{O}(\cdot)$ notation, and the sketch in Figure 1 ignores this effect.

[3] Another notable FPTAS is via zeros of polynomials [3, 29]. It can achieve similar subquadratic running time when $\lambda = o(\Delta^{-2})$, but it is apparently no faster than Weitz's correlation decay algorithm.

and Jerrum [1]. The marginal of the root of the SAW tree preserves the desired marginal probability, and can be evaluated in time linear in the size of the tree via standard recursion. We use the marginal sampler to draw a random boundary condition at a suitable depth on the SAW tree, and compute the marginal of the root using recursion under this boundary condition. Both steps can be computed in time near-linear in the size of the sub-tree. The depth of our boundary condition is roughly half of where Weitz truncates the SAW tree, and yet we show that our estimator has $O(1/n)$ variance under SSM, which is essential to get an FPRAS. This leads to our quadratic improvement on the marginal estimation over Weitz's algorithm. This method also extends to other anti-ferromagnetic 2-spin systems.

Our second contribution is about graphs with polynomial growth. In particular, for planar graphs with quadratic growth, we provide $\widetilde{O}(n^{2-c}/\varepsilon^2)$ algorithms for some constant $c > 0$. An informal statement is as follows. (The detailed statement is Theorem 13.)

▶ **Theorem 3.** *Let $\mathbb{G}$ be a family of planar graphs with quadratic growth. For a spin system exhibiting SSM on $\mathbb{G}$, there exists an FPRAS for the partition function of $G \in \mathbb{G}$ with n vertices. The run-time is $\widetilde{O}(n^{2-c}/\varepsilon^2)$ for some constant $c > 0$.*

We note that one of the most important graph in statistical physics, the 2D integer grid $\mathbb{Z}^2$, indeed has quadratic growth. More generally, any planar graph with a bounded radius circle packing has quadratic growth. Thus Theorem 3 covers many important families of planar graphs, including most lattices. (A *non*-example would be the Cayley tree.) Specialized to the hard-core model, Theorem 3 works up to the critical threshold, which is at least $\lambda_c(\Delta)$,[4] when the graph satisfies the condition in the theorem and has maximum degree Δ.

The key to Theorem 3 is once again a suitable estimator for marginal probabilities. We choose a distance ℓ boundary around a vertex v in G with a carefully chosen ℓ, and our estimator is the marginal under random boundary conditions. This boundary condition is yet again sampled using the algorithm of Anand and Jerrum [1]. Our main observation is that due to quadratic growth, the number of possible boundary conditions do not grow very fast. It turns out to be more efficient to create a look-up table by enumerating all boundary conditions first, and instead of computing the marginal for each sample, we simply find it in this table. Since planar graphs have linear local tree-width, the table can be created efficiently. This last step is inspired by the work of Yin and Zhang [35].

This method extends to any (not necessarily planar) graph families with polynomial growth. Without planarity, we use brute-force enumeration instead to create the table. This makes our gain on the running time smaller. Again an informal statement is as follows, with the full version in Theorem 22.

▶ **Theorem 4.** *Let $\mathbb{G}$ be a family of graphs with polynomial growth. For a spin system exhibiting SSM on $\mathbb{G}$, there exists an FPRAS for the partition function of $G \in \mathbb{G}$ with n vertices. The run-time is $\widetilde{O}\left(\frac{n^2}{\varepsilon^2 2^{c(\log n)^{1/d}}}\right)$ where $c > 0$ is some constant and d is the exponent of the polynomial growth.*

An example of such graphs would be the d-dimensional integer lattice $\mathbb{Z}^d$. Note that Theorem 3 is better than Theorem 4 for $d = 2$ but requires the extra assumption of planarity. The speedup factor $2^{c(\log n)^{1/d}}$ in Theorem 4 is slower than any polynomial in n but faster than any polynomial in $\log n$.

We note an interesting related work by Chu, Gao, Peng, Sachdeva, Sawlani, and Wang [11], who give an approximate counting algorithm with running time $\widetilde{O}(m^{1+o(1)} + n^{1.875+o(1)}/\varepsilon^{1.75})$ for spanning trees of graphs, where m is the number of edges and n is the number of vertices.

[4] For a given graph family, such as subgraphs of $\mathbb{Z}^2$, the critical threshold may be well above $\lambda_c(\Delta)$.

Notice that the input size here is $O(m)$ and $m = \Omega(n)$. Thus their running time is also sub-quadratic. However, there are some key differences between this work and ours. Aside from not being a spin system, spanning trees can be counted exactly in polynomial time, thanks to Kirchhoff's matrix-tree theorem. This allows them to use various efficient exact counting subroutines, whereas the problems we consider are #**P**-hard in general and no such subroutine is likely to exist.

Another more recent related result is the sub-quadratic all-terminal unreliability estimation algorithm by Cen, He, Li, and Panigrahi [7], which runs in sub-quadratic time $m^{1+o(1)}\varepsilon^{-3} + \widetilde{O}(n^{1.5}\varepsilon^{-2})$. This problem, while #**P**-hard, is not a spin system either. Their method features a recursive Monte Carlo estimator that is very different from ours, and not applicable to spin systems.

A crucial ingredient of our algorithm is the recursive marginal sampler of Anand and Jerrum [1]. This type of local / marginal samplers allows us partial access to a large random object with substantially less information than traditional samplers. It has found applications in local computation algorithms [4], and in derandomising Markov chains [16]. Our results offer yet another application, namely to accelerate computation of the global partition function.

We hope that our results are just the first step towards answering the question (1). In particular, it is not clear whether an $O(n^{2-c}/\varepsilon^2)$ algorithm exists for the hard-core model when $\lambda = \Theta(1/\Delta)$ on graphs with maximum degree Δ, or if more efficient algorithms exist for graphs with polynomial or sub-exponential growth. We leave these questions as open problems.

2 Preliminaries

We are interested in spin systems which exhibit strong spatial mixing.

▶ **Definition 5.** *A q state spin system (or q-spin system for short) is given by a graph $G = (V, E)$, a q-by-q interaction matrix A, and a field $b : [q] \to \mathbb{R}$. A configuration of G is an assignment of states to vertices, $\sigma : V \to [q]$. The weight of a configuration σ is determined by the assignments to the vertices and the interactions between them,*

$$w(\sigma) := \prod_{(u,v)\in E} A_{\sigma(u),\sigma(v)} \prod_{v\in V} b_{\sigma(v)}.$$

The Gibbs distribution μ is one where the probability of each configuration is proportional to its weight, namely, $\mu(\sigma) := \frac{w(\sigma)}{Z(G)}$, where the partition function $Z(G) = \sum_\sigma w(\sigma)$ is a normalizing factor.

In this paper, we consider the following permissive spin system, which says any locally feasible configuration can be extended to a globally feasible configuration.

▶ **Definition 6.** *A q-spin system on $G = (V, E)$ is permissive if for any $\Lambda \subseteq V$, any $\sigma \in [q]^\Lambda$, if $b_{\sigma(v)} > 0$ for all $v \in \Lambda$ and $A_{\sigma(u),\sigma(v)} > 0$ for all $u, v \in \Lambda$ satisfying $(u, v) \in E$, then σ can be extended to a full configuration $\sigma' \in [q]^V$ such that $w(\sigma') > 0$.*

Many natural spin systems are permissive. Examples include the hard-core model, the graph q-colouring with $q \geq \Delta + 1$, where Δ is the maximum degree of the graph, and all spin systems with soft constraints (e.g. the Ising model and the Potts model).

We call the problem of evaluating Z the counting problem for the q-spin system. The standard algorithmic aim here is a *fully-polynomial randomised approximation scheme* (FPRAS), where given the spin system and an accuracy $\varepsilon > 0$, the algorithm outputs $\widetilde{Z}$

such that $1-\varepsilon \le \frac{\widetilde{Z}}{Z} \le 1+\varepsilon$ with probability at least $3/4$, and runs in time polynomial in the size of the system and $1/\varepsilon$. To understand the requirement of an FPRAS, note that the probability $3/4$ can be boosted arbitrarily close to 1 via standard means. The accuracy can also be boosted by taking many disjoint copies of the system. In fact, any polynomial accuracy can be boosted to an arbitrarily small ε in polynomial-time.

Also note that if G is disconnected, then $Z(G) = \prod_i Z(G_i)$ where $G_i's$ are the connected components of G. Thus, we always consider connected graphs in the paper.

Similar to $\mu(\sigma)$ for the probability of a configuration, for an $S \subseteq V$ and a partial configuration σ_S on S, we use $\mu(\sigma_S)$ for the marginal probability of σ_S under μ. We denote the marginal distribution induced by μ on S by μ_S. When $S = \{v\}$, we also write μ_v. For the distribution conditioned on a partial configuration σ_S, we use μ^{σ_S} or $\mu_v^{\sigma_S}$.

Strong spatial mixing is a property of the spin system where a partial configuration of G does not significantly influence the assignment of a distant vertex.

▶ **Definition 7** (SSM). *A q-spin system is said to have strong spatial mixing with decay rate $f(\ell)$ for a family of graphs if for any $G = (V, E)$ in the family, any $v \in V, S \subset V$, and two configurations σ_S, τ_S,*

$$d_{\mathrm{TV}}(\mu_v^{\sigma_S}, \mu_v^{\tau_S}) \le f(\ell),$$

where d_{TV} denotes the total variation distance, $T \subseteq S$ is the subset where the configurations are different, and $\ell = \mathrm{dist}(v, T)$ is the minimum distance from v to a vertex in T.

Strong spatial mixing is a very strong form of correlation decay. When $f(\ell) = \exp(-\Omega(\ell))$ we say we have strong spatial mixing with exponential decay.

2.1 Two-state spin systems

A spin system is symmetric if $A_{ij} = A_{ji}$ for all i, j. When $q = 2$ and the system is symmetric, we have states $\{0, 1\}$ and can normalize A and b so that the interaction between 0 and 1 and the contribution of 0 are 1, and $A = \begin{bmatrix} \beta & 1 \\ 1 & \gamma \end{bmatrix}$ and $b = (1, \lambda)$ for $\beta, \gamma \ge 0$, and $\lambda > 0$.

When $\beta = \gamma$ the system is an Ising model, and for $\beta = 1, \gamma = 0$ the system is a hard-core gas model. We call a system *anti-ferromagnetic* if disagreeing assignments of adjacent vertices are more heavily weighted, namely $\beta\gamma < 1$.

For a tree T rooted at v and a partial configuration σ_S we define the marginal ratio

$$R_T^{\sigma_S} := \frac{\mu_v^{\sigma_S}(1)}{\mu_v^{\sigma_S}(0)} = \frac{\mu_v^{\sigma_S}(1)}{1 - \mu_v^{\sigma_S}(1)},$$

or $R_T^{\sigma_S} := \infty$ if $\mu_v^{\sigma_S}(1) = 1$. These ratios satisfy a well-known recurrence relation:

$$R_T^{\sigma_S} = \lambda \prod_{i=1}^{d} \frac{\gamma R_{T_i}^{\sigma_S} + 1}{R_{T_i}^{\sigma_S} + \beta}, \tag{2}$$

where T_i is the ith subtree of T. Similarly, for a graph G we can define $R_{G,v}^{\sigma_S} = \mu_v^{\sigma_S}(1)/(1 - \mu_v^{\sigma_S}(1))$. While $R_{G,v}^{\sigma_S}$ does not exhibit a simple recursion, the self-avoiding walk (SAW) tree of G at v as constructed by Weitz [34] can be used to compute it.

▶ **Theorem 8** (Theorem 3.1 of [34]). *For any $G = (V, E)$, a configuration σ_S on $S \subset V$, and any $v \in V$, there exists a tree $T_{\mathrm{SAW}} = T_{\mathrm{SAW}}(G, v)$ such that*

$$R_{G,v}^{\sigma_S} = R_{T_{\mathrm{SAW}}}^{\sigma_S}.$$

The SAW tree is rooted at v. Each node corresponds to a self-avoiding walk starting from v. The length of the walk is the same as the distance between the node and the root v. When a walk is closed, the node is set to unoccupied or occupied according to if the penultimate vertex is before or after the starting vertex of the cycle in some pre-determined local ordering at the last vertex. For details, see [34].

The SAW tree can have depth up to n, so may be exponential in size. Marginals on the SAW tree are therefore difficult to compute, but using the recursion in Equation (2) we can approximate them by truncating the tree. This approximation is accurate when strong spatial mixing holds, and the time to compute the marginal is linear in the size of the truncated tree. To maintain a polynomial running time, Weitz [34] choose to truncate it at a suitable logarithmic depth.

3 Fast SSM regime for 2-spin systems

In this section we give a quadratic speedup of Weitz's Algorithm to estimate the marginal of a single vertex in 2-spin systems, albeit being randomised instead of deterministic. We use the hard-core model as our running example to illustrate the main ideas. The main result of the section is Theorem 1.

Let the hard-core model be described by $A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ and $b = (1, \lambda)$. The support of the Gibbs distribution is the set of independent sets of G. Let vertices assigned 0 not be in the independent set (unoccupied) and vertices assigned 1 be in the independent set (occupied). Our algorithm uses self-reduction [25] as follows. Since unoccupied vertices contribute 1 to the weight of a configuration, we can consider the all 0 configuration σ_0 where

$$\begin{aligned} \frac{1}{Z(G)} = \mu(\sigma_0) = \mu_{v_1}(0)\mu^{v_1 \leftarrow 0}_{V\setminus\{v_1\}}(\mathbf{0}) = \mu_{v_1}(0)\frac{1}{Z(G\setminus\{v_1\})} \\ = \mu_{G_1,v_1}(0)\mu_{G_2,v_2}(0)\cdots\mu_{G_n,v_n}(0), \end{aligned} \tag{3}$$

and where $G_i = G \setminus \{v_1, \ldots, v_{i-1}\}$ for all $i \in [n]$. This reduces the problem of computing $Z(G)$ to computing μ_{v_1} and recursively $Z(G \setminus \{v_1\})$. As the G_i's are subgraphs of G, they have the same degree bound and still exhibit SSM. The crux of our algorithm is to design a random variable that estimates μ_v in time $\widetilde{O}(n^{1/(2k)})$.

Another ingredient we need is the lazy single-site sampler by Anand and Jerrum [1], which allows us to rapidly sample a partial configuration vertex by vertex. The original setting of [1] requires sub-exponential neighbourhood growth in order to work up to the strong spatial mixing threshold, but in our parameter regime no sub-exponential growth is required. Moreover, only the expected running time is studied in [1], while we need a tail bound. A similar analysis is done in [16, Appendix B]. For completeness, we provide a proof specialised to our setting in Appendix A.

▶ **Lemma 9.** *Let $\Delta \geq 2$ be an integer and $\lambda < \frac{1}{\Delta-1}$. Let $G = (V, E)$ be a graph with maximum degree Δ. There exists an algorithm that, for any $v \in V$, draws a sample from μ_v and halts in time $O(\log \frac{1}{\varepsilon})$ with probability at least $1 - \varepsilon$.*

Our algorithm then combines the lazy sampler of Lemma 9 with the SAW tree of [34]. We expand the SAW tree, and then use Lemma 9 to sample a truncated boundary, from which we use the recursion in (2) to get our estimate. The depth of the truncation controls the variance of this estimator. In our algorithm, we only need to bound the variance from above by $1/n$. In contrast, Weitz's algorithm requires the error of the marginal incurred

by the truncation to be bounded from above by $O(1/n)$. As the variance of our estimator decays twice as fast as the marginal errors, our truncation depth is roughly half of that in Weitz's algorithm. Consequently, we achieved a quadratic speedup for estimating each term in (3).

▶ **Lemma 10.** *For a graph G with maximum degree Δ, if the hard-core model on G has strong spatial mixing with decay rate $C\Delta^{-k\ell}$ for some constant $C > 0$, there exists an algorithm that generates a random sample $\widetilde{p}_v$ and halts in time $O(n^{1/(2k)}(\log \frac{n}{\delta})^2)$ with probability at least $1 - \frac{\delta}{8}$. Furthermore, $\mathbb{E}[\widetilde{p}_v] = \mu_v(0)$ and $\mathrm{Var}\,(\widetilde{p}_v) \le 1/n$.*

Proof. Let T_{SAW} be the self-avoiding walk tree for G rooted at v as defined in Theorem 8, and let $S = \{u \in V | d_{T_{\mathrm{SAW}}}(v, u) = \ell\}$ where ℓ is a parameter we will fix later. We have

$$\mu_{T_{\mathrm{SAW}},v}(0) = \sum_{\sigma \in \{0,1\}^S} \mu_{T_{\mathrm{SAW}}}(\sigma)\mu^{\sigma}_{T_{\mathrm{SAW}},v}(0) = \mathbb{E}_{\sigma \sim \mu_{T_{\mathrm{SAW}},S}}[\mu^{\sigma}_{T_{\mathrm{SAW}},v}(0)].$$

We use Lemma 9 to sample σ. Fix an arbitrary order of $S = \{s_1, s_2, \ldots, s_{|S|}\}$. We sample first the marginal of s_1 with $\varepsilon := \frac{\delta}{8|S|}$. Then, conditioned on the result on s_1, we sample s_2 with the same ε, and so on and so forth. Note that whatever the result on s_1 is, it always reduces to a hard-core instance of a smaller graph. Thus, the condition of Lemma 9 is always satisfied until all of S are sampled. This gives a boundary condition σ_S in T_{SAW}.

As the full SAW tree may be exponential in size, a little care is required to implement the outline above. We first expand T_{SAW} up to level ℓ, denoted $T_{\mathrm{SAW},\ell}$. The algorithm in Lemma 9 (Algorithm 1 in Appendix A) is essentially an exploration process. When we apply it to sample the boundary condition σ_S, we expand the SAW tree below $T_{\mathrm{SAW},\ell}$ on the fly, only creating vertices that are explored by the algorithm. Note that the construction of the SAW tree imposes a boundary condition whenever a vertex in G is encountered again in a self-avoiding walk. We implement this pinning by remembering a list of all ancestors of a given node in the SAW tree and checking the next vertex to explore against this list. Since Lemma 9 halts in $O(\log \frac{1}{\varepsilon})$ time with probability at least $1 - \varepsilon$, this extra check incurs a multiplicative slowdown factor $O(\ell + \log \frac{1}{\varepsilon}) = O(\ell + \log \frac{|S|}{\delta})$ with probability at least $1 - \varepsilon$.

Given σ_S, we can compute $\mu^{\sigma_S}_v(0) = \widetilde{p}_v$ with the standard dynamic programming approach. By a union bound, the total running time of sampling the boundary is $O(|S| \log \frac{|S|}{\delta}(\ell + \log \frac{|S|}{\delta}))$ with probability at least $1 - \frac{\delta}{8}$, and the dynamic programming step uses time $O(|T_{\mathrm{SAW},\ell}|)$.

We choose $\ell := \left\lceil \frac{\log(n)/2 - \log C}{k \log(\Delta)} \right\rceil$ so that $C\Delta^{-k\ell} \le n^{-0.5}$ and $\Delta^{\ell} \le C' n^{1/(2k)}$ for some constant $C' > 0$. Note that $|S| \le (\Delta - 1)^{\ell}$ and $|T_{\mathrm{SAW},\ell}| \le \Delta^{\ell}$. Then the total runtime to draw a sample is $O(n^{1/(2k)}(\log \frac{n}{\delta})^2)$ with probability at least $1 - \frac{\delta}{8}$.

Finally, we analyze the variance. SSM implies that for any σ_S, $|\mu^{\sigma_S}_v(0) - \mu_v(0)| \le C\Delta^{-k\ell}$, so

$$\mathrm{Var}\,(\widetilde{p}_v) = \mathrm{Var}_{\sigma_S}\left(\mu^{\sigma_S}_v(0)\right) = \mathbb{E}_{\sigma_S \sim \mu_{T_{\mathrm{SAW}},S}}[|\mu^{\sigma_S}_v(0) - \mu_v(0)|^2] \le \left(C\Delta^{-k\ell}\right)^2 \le n^{-1}, \qquad (4)$$

which is what we desire. ◀

▶ **Lemma 11.** *For a graph G with maximum degree Δ, if $\lambda \le \frac{1}{\Delta^k(\Delta-1)}$ for some constant $k > 0$, the hard-core model on G exhibits strong spatial mixing with decay rate $C\Delta^{-k\ell}$.*

Proof. It is well-known that if $\lambda < \lambda_c(\Delta) = \frac{(\Delta-1)^{\Delta-1}}{(\Delta-2)^{\Delta}} \approx \frac{e}{\Delta}$, strong spatial mixing holds with exponential decay Cr^{ℓ} for some constant C and $r < 1$ [34]. Moreover, the decay rate r can be controlled by a quantity related to the recursion (2) [30]. For example, by [20,

Lemma 7.20], r is bounded by $r \le |f'(\widehat{x})|$, where $f(x) := \frac{\lambda}{(1+x)^{\Delta-1}}$ is the symmetric version of the recursion in (2) and $\widehat{x}$ is the unique positive fixed point of f. (Note that when the degree of G is at most Δ, all vertices but the root in T_{SAW} have branching number $\Delta - 1$.) Then we have

$$|f'(x)| = \left| -\frac{f(x)(\Delta-1)}{1+x} \right| < (\Delta-1)f(x).$$

As $\widehat{x} > 0$ and $\widehat{x}$ is a fixed point, it holds that $\widehat{x} < \widehat{x}(1+\widehat{x})^{\Delta-1} = \lambda$. Thus, as $\lambda \le \frac{1}{\Delta^k(\Delta-1)}$,

$$r \le |f'(\widehat{x})| < (\Delta-1)f(\widehat{x}) = (\Delta-1)\widehat{x} < \frac{1}{\Delta^k}. \qquad \blacktriangleleft$$

Now we are ready to prove Theorem 1.

Proof of Theorem 1. We will first give an algorithm whose running time has a tail bound. To have a fixed running time upper bound, we then truncate this algorithm.

Set $N := \lceil 8e^{(1+\lambda)^2}/\varepsilon_0^2 \rceil$ where $\varepsilon_0 = \varepsilon/2$. Let $X := \prod_{i=1}^{n} \widetilde{p}_{G_i,v_i}$ where $G_1 = G$ and $G_i = G_{i-1} \setminus \{v_{i-1}\}$. By Lemma 11, we can use Lemma 10 to draw N samples of X and take its average, where we set $\delta = \frac{1}{nN}$ in Lemma 10. Each $\widetilde{p}_{G_i,v_i}$ can be computed in time $O(n^{1/(2k)}(\log \frac{n}{\delta})^2)$ with probability at least $1 - \frac{\delta}{8}$, so computing one sample of X takes time $O(n^{1+1/(2k)}(\log \frac{n}{\delta})^2)$ time with probability at least $1 - \frac{n\delta}{8}$ by a union bound. By a union bound again, the overall running time of taking the average is $O(Nn^{1+1/(2k)}(\log \frac{n}{\delta})^2) = O\left(\frac{n^{1+1/(2k)}}{\varepsilon^2}(\log \frac{n}{\varepsilon})^2\right)$ with probability at least $1 - \frac{\delta}{8} \cdot nN = \frac{7}{8}$.

Since $\{\widetilde{p}_{G_i,v_i}\}$ are mutually independent, by Lemma 10,

$$\mathbb{E}[X] = \mathbb{E}\left[\prod_{i=1}^{n} \widetilde{p}_{G_i,v_i}\right] = \prod_{i=1}^{n} \mu_{G_i,v_i}(0) = \frac{1}{Z(G)}.$$

We bound $\mathrm{Var}\,(X)$ as follows

$$\begin{aligned}\frac{\mathrm{Var}\,(X)}{(\mathbb{E}[X])^2} &= \frac{\mathbb{E}[X^2]}{(\mathbb{E}[X])^2} - 1 = \frac{\prod_{i=1}^{n}\mathbb{E}[\widetilde{p}_{G_i,v_i}^2]}{\prod_{i=1}^{n}\mathbb{E}[\widetilde{p}_{G_i,v_i}]^2} - 1 = \prod_{i=1}^{n}\left(1 + \frac{\mathrm{Var}\,(\widetilde{p}_{G_i,v_i})}{(\mathbb{E}[\widetilde{p}_{G_i,v_i}])^2}\right) - 1 \\ &\le \left(1 + \frac{c}{n}\right)^n - 1 < e^c, \qquad \text{(by Lemma 10)}\end{aligned}$$

where $c = \max_i(1/\mu_{G_i,v_i}(0)^2)$. Note that as $\mu_{G_i,v_i}(0) \ge \frac{1}{1+\lambda}$, $c \le (1+\lambda)^2$ and is a constant.

Let $\widetilde{X}$ be the average of N samples of X. Then $\mathrm{Var}\left(\widetilde{X}\right) = \frac{\mathrm{Var}(X)}{N} \le \frac{e^c}{N \cdot Z(G)^2}$. By Chebyshev's inequality,

$$\Pr\left[\left|\widetilde{X} - \frac{1}{Z(G)}\right| \ge \frac{\varepsilon_0}{Z(G)}\right] \le \frac{\mathrm{Var}\left(\widetilde{X}\right)}{\frac{\varepsilon_0^2}{Z(G)^2}} \le \frac{e^c}{N \cdot Z(G)^2} \cdot \frac{Z(G)^2}{\varepsilon_0^2} \le \frac{1}{8}.$$

Thus, with probability at least $7/8$, we have that $\frac{1-\varepsilon_0}{Z(G)} \le \widetilde{X} \le \frac{1+\varepsilon_0}{Z(G)}$. Finally, we output $\widetilde{Z} = 1/\widetilde{X}$. To make sure that the algorithm runs within the desired time bound $O\left(\frac{n^{1+1/(2k)}}{\varepsilon^2}(\log \frac{n}{\varepsilon})^2\right)$, we truncate the algorithm if it runs overtime and output an arbitrary value in that case. This truncated version can be coupled with the untruncated algorithm with probability at least $7/8$, and its output $\widetilde{Z}$ satisfies $1 - \varepsilon \le \frac{\widetilde{Z}}{Z(G)} \le 1 + \varepsilon$ with probability at least $7/8 - 1/8 = 3/4$. $\blacktriangleleft$

Note that, Weitz's algorithm is faster if the correlation decay is faster, but in that case so is our algorithm. In Appendix B, Lemma 26 shows that the correlation decay cannot be much faster than the standard analysis in the parameter regimes of Theorem 1, and our speed-up, comparing to Weitz's algorithm, is always at least $\widetilde{O}(n^{1/2k-o(1/k^2)})$.

We also remark that Theorem 1 generalises to antiferromagnetic 2-spin systems. This is because all the key ingredients, namely correlation decay, Weitz's SAW tree, and the marginal sampler of Anand and Jerrum all generalise, except that the Anand-Jerrum algorithm would require the neighbourhood growth rate smaller than the decay rate (see Theorem 25). This is also the parameter regime where Weitz's algorithm is faster than $O(n^2)$. Thus, our speedup is still in the sub-quadratic regime. The self-reduction in (3) also generalises (as we will see in (5) in the next section). One needs to redo the calculations in Lemma 11 to get a precise statement, which we will omit here.

4 Speed-up on planar graphs

In this section we mainly consider (not necessarily two-state) spin systems on planar graphs. We show that for any planar graph with quadratic neighbourhood growth, when SSM holds with exponential decay, approximate counting can be done in sub-quadratic time. For example, this includes all subgraphs of the 2D lattice $\mathbb{Z}^2$. The circle-packing theorem asserts that any planar graph is the tangent graph of some circle packing. All planar graphs with bounded-radius circle packings have quadratic neighbourhood growth. Thus this is a substantial family of planar graphs. Moreover, in Section 4.2 we extend the result to (not necessarily planar) graphs with polynomial growth, but the speed up factor there is sub-polynomial yet faster than $(\log n)^k$ for any k.

▶ **Definition 12.** *A graph family $\mathbb{G}$ has* quadratic growth*, if there is a constant C_0 such that for any $G = (V, E) \in \mathbb{G}$, $v \in V$, and any integer $\ell > 0$, $|B_v(\ell)| \le C_0\ell^2$.*

Subgraphs of the 2D lattice $\mathbb{Z}^2$ satisfies Definition 12 with $C_0 = 5$. Note that by taking $\ell = 1$, Definition 12 implies that the maximum degree is no larger than C_0.

▶ **Theorem 13.** *Let $\mathbb{G}$ be a family of planar graphs with quadratic growth (assume the rate is $C_0\ell^2$). Let $\boldsymbol{A}$ and $\boldsymbol{b}$ specify a q-state spin system, which exhibits SSM with decay rate $Cr^{-\ell}$ on $\mathbb{G}$. Then there is a constant $c > 0$ such that there exists an FPRAS for the partition function of $G \in \mathbb{G}$ with n vertices with run-time $\widetilde{O}(n^{2-c}/\varepsilon^2)$. The constant c depends on C_0, q, and r.*

Theorem 13 is the detailed version of Theorem 3.

Essentially the idea is still to find an estimator for the marginal of an arbitrary vertex that can be evaluated very quickly. Let us first consider a $\sqrt{n}$-by-$\sqrt{n}$ grid. For any vertex v, we consider the sphere $S_v(\ell)$ of radius $\ell = O(\log n)$ centered at v, and a random configuration τ on $S_v(\ell)$. Let $B_v(\ell)$ be the ball of radius ℓ centered at v. Since any planar graph has linear local tree-width [12, 15], $B_v(\ell)$ has tree-width $O(\ell)$. Thus, given a configuration τ on S, the law of μ_v^τ can be computed in time $2^{O(\ell)}\mathrm{poly}(\ell)$ for a fixed τ (see, e.g. [35][5]). This step can be very efficient with a carefully chosen ℓ.

For a general bounded degree planar graph, $|S_v(\ell)|$ can be a polynomial in n, which makes the number of possible τ's exponential in n. However, for a $\sqrt{n}$-by-$\sqrt{n}$ grid, $|S_v(\ell)| \le 4\ell = O(\log n)$, and the number of possible τ's is much smaller and is a small polynomial in

[5] The algorithm in [35] uses the separator decomposition. Another possibility is to first find a constant approximation of the tree decomposition first [26], and then apply Courcelle's theorem.

n. Thus, it would be more efficient to first create a table to list all possibilities of τ, and then, instead of computing μ_v^τ each time, simply look up the answer from this table. We can do the same for any subgraph of $\mathbb{Z}^2$ by choosing a boundary based on distance in the original grid.

For a general $G \in \mathbb{G}$, we no longer have a linear bound on the size of the boundary. (In Appendix C of the arXiv version of this paper, we construct a subgraph of $\mathbb{Z}^2$ where the distance ℓ boundary has size $\Omega(\ell^2)$.) However, since $\mathbb{G}$ has quadratic growth, we know that $B_v(\ell) \le C_0\ell^2$ for some constant $C_0 > 0$. It implies that

$$\sum_{i=\ell/2}^{\ell} |S_v(\ell)| \le |B_v(\ell)| \le C_0\ell^2.$$

Thus, there must exist an $\ell' \in [\ell/2, \ell]$ such that $|S_v(\ell')| \le 2C_0\ell$. We will find this ℓ' and use $S_v(\ell')$ instead.

Once again, we use a self-reduction similar to (3). For q-spin systems, given a feasible configuration σ, we have the decomposition,

$$\frac{w(\sigma)}{Z_G} = \mu(\sigma) = \mu_{v_1}(\sigma_{v_1})\mu_{v_2}^{\sigma_{v_1}}(\sigma_{v_2})\mu_{v_3}^{\sigma_{v_1},\sigma_{v_2}}(\sigma_{v_3})\dots\mu_{v_n}^{\sigma_{v_1},\dots,\sigma_{v_{n-1}}}(\sigma_{v_n}). \tag{5}$$

When computing our table, we will have to condition on the already pinned vertices.

▶ **Lemma 14.** *Let $\boldsymbol{A}$, $\boldsymbol{b}$, q and G be as in Theorem 13. For $v \in V$, a partial configuration σ, and an integer ℓ, we can find an ℓ' such that $\ell' \in [\ell/2, \ell]$, and then construct a table of $\mu_v^{\sigma,\tau}$, indexed by every boundary configuration τ on unpinned vertices of $S_v(\ell')$. The total run-time is $2^{C_1\ell}$, where C_1 is a constant depending on C_0 and q.*

Proof. As discussed earlier, due to the quadratic growth of G, there must exist an ℓ such that $\ell' \in [\ell/2, \ell]$ and $|S_v(\ell')| \le 2C_0\ell$. To find this ℓ, we do a breadth-first-search to check $S_v(i)$ from $i = \ell/2$ to ℓ. The running time is at most $O(B_v(\ell)) = O(\ell^2)$.

Once ℓ' is found, $|S_v^\sigma(\ell')| \le |S_v(\ell')| \le 2C_0\ell$, and there are at most $q^{2C_0\ell}$ configurations τ in our table. As G is a planar graph, the tree-width of the ball $\mathrm{tw}(B_v(\ell')) = O(\ell') = O(\ell)$. Thus, using for example the algorithm of [35], each entry of the table can be computed in time $2^{O(\ell)}\mathrm{poly}\,(\ell)$. The total amount of time required is $O(\ell^2) + q^{2C_0\ell}2^{O(\ell)}\mathrm{poly}\,(\ell) \le 2^{C_1\ell}$, for some sufficiently large constant C_1. ◀

While we may construct this table very quickly, it is not clear how to compute or estimate the marginals of the boundary condition τ's rapidly. Instead, we sample a random one using the marginal sampler [1] that terminates in almost linear time with high probability. See Theorem 25.

▶ **Lemma 15.** *Let $\boldsymbol{A}$, $\boldsymbol{b}$, q and $G \in \mathbb{G}$ be as in Theorem 13. Let σ be a partial configuration. For any $v \in V$ not pinned under σ and any $k \in [q]$, there exists an algorithm that generates a random variable $\widetilde{Z}$ such that $\mathbb{E}[\widetilde{Z}] = \mu_v^\sigma(k)$ and $\mathrm{Var}\left(\widetilde{Z}\right) \le 1/n$. Moreover, its running time is $\widetilde{O}(n^{1-c})$ with high probability where c depends on C_0, q, and r.*

Proof. Let ℓ be a constant that we will choose later. Let $\ell' \in [\ell/2, \ell]$ be as in Lemma 14, and let τ be a boundary condition on the unpinned vertices of $S_v(\ell')$ under σ. Let $Z_v(\tau) = \mu_v^{\sigma,\tau}(k)$ so that $\mathbb{E}_\tau[Z_v(\tau)] = \mu_v^\sigma(k)$. Then, let

$$\widetilde{Z} := \frac{1}{m}\sum_{j=1}^{m} Z_v(\tau_j)$$

be the empirical mean over m random samples τ_j, where we will choose m later.

Since the spin system exhibits SSM with decay rate $Cr^{-\ell}$, similar to (4), we have $\mathrm{Var}\left(Z_v(\tau)\right) \le C^2 r^{-2\ell'} \le C^2 r^{-\ell}$. Then

$$\mathrm{Var}\left(\widetilde{Z}\right) = \mathrm{Var}\left(\frac{1}{m}\sum_{j=1}^{m} Z_v(\tau_j)\right) \le \frac{C^2}{mr^{\ell'}}.$$

Thus, we set $m = \lceil nC^2 r^{-\ell'} \rceil$ samples so that $\mathrm{Var}\left(\widetilde{Z}\right) \le 1/n$.

For the running time, we first construct the table as in Lemma 14. Then we take m samples of τ, each of which can be generated in time almost linear in $|S_v(\ell')|$ with high probability using Theorem 25. As $|S_v(\ell')| \le 2C_0\ell$, the runtime in total is at most $O(2^{C_1\ell} + n\ell r^{-\ell'} \log n)$ with high probability. We choose $\ell = \frac{1-c}{C_1}\log n$ for $c = \frac{\log r}{\log r + 2C_1} \in [0,1]$, so that $\ell' \ge \ell/2 = \frac{1-c}{2C_1}\log n$ and the total runtime is $O(n^{1-c} + n^{1-(1-c)\log r/(2C_1)}\log^2 n) = \widetilde{O}(n^{1-c})$. ◀

Now we are ready to prove Theorem 13.

Proof of Theorem 13. We are going to use (5) to do a self-reduction. First we construct the target configuration σ adaptively. Given σ on $v_1, \dots, v_{i-1}$, we want to choose σ_{v_i} to be $k \in [q]$ with the largest marginal. In other words, $\sigma_{v_i} = \mathrm{argmax}_{k\in[q]} \mu_{v_i}^{\sigma_i}(k)$ for each i, where σ_i is what has been constructed so far, namely $\sigma_{v_1}, \dots, \sigma_{v_{i-1}}$. Of course, this step cannot be done exactly. Instead, we may fix a constant $t = t(C, r, q)$ such that $Cr^{-t} \le \frac{1}{2q}$, fix an arbitrary boundary configuration τ on $S_{v_i}^{\sigma_i}(t)$ and then pick $k \in [q]$ that maximises $\mu_{v_i}^{\sigma_i,\tau}(k)$. SSM guarantees that $\mu_{v_i}^{\sigma_i}(\sigma_{v_i}) \ge 1/2q$, where $\sigma_{v_i} = k$. This step takes constant time as t is a constant.

The rest of the proof is very similar to that of Theorem 1. Set $N := \lceil 10e^{4q^2}/\varepsilon_0^2 \rceil$ where $\varepsilon_0 = \varepsilon/2$. We compute $X = \prod_{i=1}^n \widetilde{Z}_i$ where each $\widetilde{Z}_i$ is from Lemma 15 plugging in v_i and σ_i. Due to the decomposition (5) we have

$$\mathbb{E}[X] = \mathbb{E}\left[\prod_{i=1}^n \widetilde{Z}_i\right] = \prod_{i=1}^n \mathbb{E}\left[\widetilde{Z}_i\right] = \prod_{i=1}^n \mu_{v_i}^{\sigma_i}(\sigma_{v_i}) = \mu(\sigma).$$

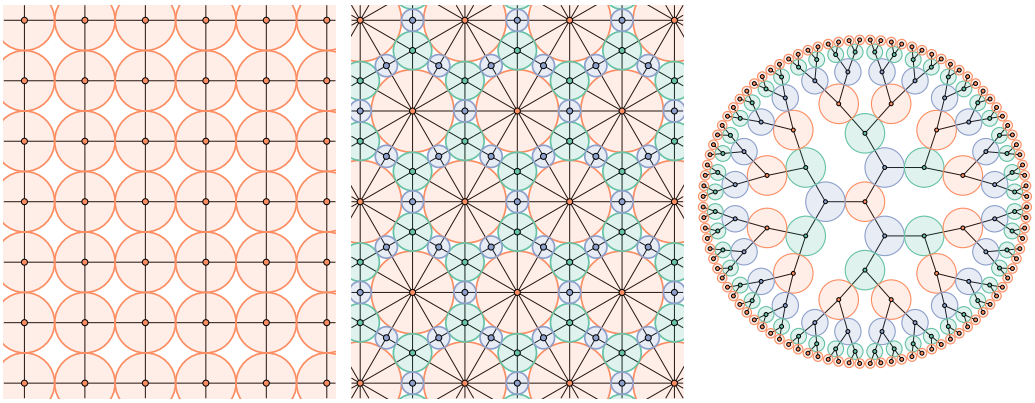
We also compute $w(\sigma)$ which can be done in $O(n)$ on a planar graph with quadratic growth. By Lemma 15, the time to generate one X is $O(n^{2-c}\mathrm{polylog}\,(n))$ with high probability. We bound $\mathrm{Var}\,(X)$ as follows

$$\begin{aligned}\frac{\mathrm{Var}\,(X)}{(\mathbb{E}[X])^2} = \frac{\mathbb{E}[X^2]}{(\mathbb{E}[X])^2} - 1 = \frac{\prod_{i=1}^n \mathbb{E}[\widetilde{Z}_i^2]}{\prod_{i=1}^n \mathbb{E}[\widetilde{Z}_i]^2} - 1 &= \prod_{i=1}^n \left(1 + \frac{\mathrm{Var}\left(\widetilde{Z}_i\right)}{(\mathbb{E}[\widetilde{Z}_i])^2}\right) - 1 \\ &\le \left(1 + \frac{4q^2}{n}\right)^n - 1 \le e^{4q^2},\end{aligned}$$

where we use $\mu_{v_i}^{\sigma_i}(\sigma_{v_i}) \ge 1/2q$ for any $i \in [n]$. Let $\widetilde{X}$ be the average of N samples of X. Then $\mathrm{Var}\left(\widetilde{X}\right) = \frac{\mathrm{Var}(X)}{N} \le \frac{e^{4q^2}}{N \cdot Z(G)^2}$. By Chebyshev's inequality,

$$\Pr\left[\left|\widetilde{X} - \frac{1}{Z(G)}\right| \ge \frac{\varepsilon_0}{Z(G)}\right] \le \frac{\mathrm{Var}\left(\widetilde{X}\right)}{\frac{\varepsilon_0^2}{Z(G)^2}} \le \frac{e^{4q^2}}{N \cdot Z(G)^2} \cdot \frac{Z(G)^2}{\varepsilon_0^2} \le \frac{1}{10}.$$

Thus, with probability at least 9/10, we have that $\frac{1-\varepsilon_0}{Z(G)} \le \widetilde{X} \le \frac{1+\varepsilon_0}{Z(G)}$. Finally, we output $\widetilde{Z} = w(\sigma)/\widetilde{X}$. Since Definition 12 implies a constant degree bound, the graph is sparse and $w(\sigma)$ can be computed in $O(n)$ time. To make sure that the algorithm runs within the



(a) (b) (c)

Figure 2 Circle packings of some lattices. (a): $\mathbb{Z}^2$ grid, $R = 1$. (b): Kisrhombille tiling, $R = 2 - \sqrt{3}$. (c): degree-3 Bethe lattice, $R = 0$.

desired time bound $O\left(\frac{n^{2-c}}{\varepsilon^2}\right)$, we truncate the algorithm if it runs overtime and output an arbitrary value in that case. This truncated version can be coupled with the untruncated algorithm with probability at least 7/8, and its output $\widetilde{Z}$ satisfies $1 - \varepsilon \leq \frac{\widetilde{Z}}{Z(G)} \leq 1 + \varepsilon$ with probability at least 3/4. ◀

4.1 Bounded-radius circle packing

Here we show that Theorem 13 applies to any planar graph with bounded-radius circle packings. We begin with the definition of a circle packing.

▶ **Definition 16.** *A* circle packing *is a collection $\mathcal{C}$ of interior-disjoint circles over the 2-dimensional plane. A* tangency graph *of a circle packing is a graph having a vertex for each circle, and an edge between two vertices if and only if the two corresponding circles are tangent.*

The Koebe-Andreev-Thurston *circle packing theorem* states the following.

▶ **Theorem 17.** *For every connected locally finite simple planar graph $\mathbb{G}$, there exists a circle packing whose tangency graph is (isomorphic to) $\mathbb{G}$.*

We are concerned with the radius of the circles used in the packing, especially the ratio between the smallest and largest ones.

▶ **Definition 18.** *A locally finite simple planar graph $\mathbb{G}$ is said to have an R-*bounded-radius circle packing *(R-BRCP) for some constant $R > 0$, if there exists a circle packing $\mathcal{C}$ whose tangency graph is (isomorphic to) $\mathbb{G}$ such that*

$$\frac{\inf\limits_{\odot \in \mathcal{C}} r_{\odot}}{\sup\limits_{\odot \in \mathcal{C}} r_{\odot}} \geq R$$

where $r_{\odot}$ denotes the radius of a circle $\odot$ in the packing.

Three examples are given in Figure 2. The $\mathbb{Z}^2$ grid can be naturally packed by unit disks, leading to $R = 1$. Such a graph is called a "penny graph". The 3, 6-kisrhombille tiling is a tiling of the 2-dimensional plane by $\pi/6$-$\pi/3$-$\pi/2$ triangles. This lattice can be packed by

circles of radii $1, 2\sqrt{3}-3, 2-\sqrt{3}$, so $R = 2-\sqrt{3}$. The degree-3 Bethe lattice, also known as the infinite 3-regular tree, can be drawn as a planar graph on the 2-dimensional plane. However, the neighbourhood growth is so fast that $R = 0$.

Fix the underlying graph $\mathbb{G}$ and its R-BRCP $\mathcal{C}$. Without loss of generality, we assume the diameter of the largest circle in $\mathcal{C}$ is 1. Thus, the radius of an arbitrary circle in $\mathcal{C}$ is between $R/2$ and $1/2$. Let G be a finite subgraph of $\mathbb{G}$. Here we need to distinguish the graph distance in G and the geometric distance (the Euclidean distance $\|\cdot\|_2$ between the center of their corresponding disks on the 2-dimensional plane). For two vertices u and v, we use $\mathrm{dist}_G(u,v)$ to denote their graph distance, and use $\|u-v\|_2$ to denote their geometric distance. Note that $\mathrm{dist}_{\mathbb{G}}(u,v) \geq \|u-v\|_2$ and $\mathrm{dist}_G(u,v) \geq \mathrm{dist}_{\mathbb{G}}(u,v)$.

For any vertex v and u in the ℓ-ball $B_v(\ell)$ in G, $\|u-v\|_2 \leq \mathrm{dist}_G(u,v) \leq \ell$. The disk $\odot_u$ corresponding to u must be contained completely in the circle centered at u with radius $\ell + 1/2$. By considering the area they cover,

$$|B_v(\ell)| \leq \frac{\pi(\ell+1/2)^2}{\pi(R/2)^2} = O(\ell^2/R^2).$$

Thus, any family of subgraphs of $\mathbb{G}$ has quadratic growth, where the growth constant depends on R. Together with Theorem 13, we have the following corollary.

▶ **Corollary 19.** *Let $\mathbb{G}$ be a locally finite simple planar graph, together with an R-BRCP where $R > 0$ is a constant. Let $\mathcal{G}$ be a family of subgraphs of $\mathbb{G}$, and $\boldsymbol{A}, \boldsymbol{b}$ specify a q-spin system that exhibits SSM with exponential decay on $\mathcal{G}$. Then there exists an FPRAS that takes a graph $G \in \mathcal{G}$ as an input and estimates the partition function of the spin system on G in time $\widetilde{O}(n^{2-c}/\varepsilon^2)$. Here, $n = |V(G)|$, and $c > 0$ is a constant depending on q, decay rate of SSM, and R.*

▶ Remark 20. The algorithm does not need to know the circle packing, as long as an R-BRCP exists.

On a separate note, although a good approximation of the circle packing of a finite planar graph can be found in near linear time [13], its output does not optimise the radius ratio. It is not clear how to generate a circle packing with a constant approximation of the optimal radius ratio. In the extreme, it is **NP**-hard to decide if a given graph G (without geometric positions) is a penny graph, namely admitting a circle packing using unit circles [14], even if G is restricted to be a tree [5].

4.2 Polynomial-growth graphs

Our method goes beyond planar graphs with quadratic growth rate. For any graph with a polynomial growth rate, we have a speed-up that is faster than any polylog factors.

▶ **Definition 21.** *A graph family $\mathbb{G}$ has* polynomial growth*, if there are constants C and d such that for any $G = (V, E) \in \mathbb{G}$, $v \in V$, and any integer $\ell > 0$, $|B_v(\ell)| \leq C_0 \ell^d$.*

Examples of graphs with polynomial growth include finite subgraphs of the d-dimensional integer lattice $\mathbb{Z}^d$. Again, by taking $\ell = 1$, Definition 21 implies that the maximum degree is no larger than C_0.

▶ **Theorem 22.** *Let $\mathbb{G}$ be a family of graphs with polynomial growth (assume the rate is $C_0\ell^d$). Let $\boldsymbol{A}$ and $\boldsymbol{b}$ specify a q-state spin system, which exhibits SSM with decay rate $Cr^{-\ell}$ on $\mathbb{G}$. Then there is a constant $c > 0$ such that there exists an FPRAS for the partition function of $G \in \mathbb{G}$ with n vertices with run-time $\widetilde{O}\left(\frac{n^2}{\varepsilon^2 2^{c(\log n)^{1/d}}}\right)$. The constant c depends on C_0, q, and r.*

Theorem 22 is the detailed version of Theorem 4.

In comparison to Theorem 13, the proof of Theorem 22 needs only a few small tweaks. Let ℓ be a parameter we will choose later, and our estimator is still set by using a random boundary condition on $S_v(\ell)$ to estimate the marginal at v. Note that we no longer need to find ℓ' for a smaller boundary. The main difference is in Lemma 14, where we no longer have linear local tree-width. Instead, we have to create the table by brute-force enumeration. There are $q^{C_0\ell^d}$ possible boundary conditions, and the overall time cost for creating the table is $O(q^{2C_0\ell^d})$.

We use the same estimator as in Lemma 15. To reduce the variance of our estimator to $1/n$, we need $nC^2r^{-2\ell}$ samples, each of which can be looked up quickly using the table. Let $\ell = \frac{0.99(\log n)^{1/d}}{2C_0 \log q}$. The overall time cost is

$$\widetilde{O}\left(\frac{n}{\varepsilon^2}\left(q^{2C\ell^d} + nr^{-2\ell}\right)\right) = \widetilde{O}\left(\frac{n}{\varepsilon^2}\left(n^{0.99} + \frac{n}{2^{c(\log n)^{1/d}}}\right)\right) = \widetilde{O}\left(\frac{n^2}{\varepsilon^2 2^{c(\log n)^{1/d}}}\right),$$

where $c = \frac{0.99 \log r}{C_0 \log q}$. This shows Theorem 22. Note that the factor $2^{c(\log n)^{1/d}}$ grows faster than $(\log n)^k$ for any $k > 0$.

References

1 Konrad Anand and Mark Jerrum. Perfect sampling in infinite spin systems via strong spatial mixing. *SIAM J. Comput.*, 51(4):1280–1295, 2022.

2 Nima Anari, Vishesh Jain, Frederic Koehler, Huy Tuan Pham, and Thuy-Duong Vuong. Entropic independence: optimal mixing of down-up random walks. In *STOC*, pages 1418–1430. ACM, 2022.

3 Alexander I. Barvinok. *Combinatorics and Complexity of Partition Functions*, volume 30 of *Algorithms and combinatorics*. Springer, 2016.

4 Amartya Shankha Biswas, Ronitt Rubinfeld, and Anak Yodpinyanee. Local access to huge random objects through partial sampling. In *ITCS*, volume 151 of *LIPIcs*, pages 27:1–27:65. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

5 Clinton Bowen, Stephane Durocher, Maarten Löffler, Anika Rounds, André Schulz, and Csaba D. Tóth. Realization of simply connected polygonal linkages and recognition of unit disk contact trees. In *GD*, volume 9411 of *Lecture Notes in Computer Science*, pages 447–459. Springer, 2015.

6 Jin-Yi Cai and Xi Chen. Complexity of counting CSP with complex weights. *J. ACM*, 64(3):19:1–19:39, 2017.

7 Ruoxu Cen, William He, Jason Li, and Debmalya Panigrahi. Beyond the quadratic time barrier for network unreliability. *CoRR*, 2023. `arXiv:2304.06552`.

8 Xiaoyu Chen, Weiming Feng, Yitong Yin, and Xinyuan Zhang. Optimal mixing for two-state anti-ferromagnetic spin systems. In *FOCS*, pages 588–599. IEEE, 2022.

9 Yuansi Chen and Ronen Eldan. Localization schemes: A framework for proving mixing bounds for markov chains (extended abstract). In *FOCS*, pages 110–122. IEEE, 2022.

10 Zongchen Chen, Kuikui Liu, and Eric Vigoda. Optimal mixing of Glauber dynamics: Entropy factorization via high-dimensional expansion. In *STOC*, pages 1537–1550. ACM, 2021.

11 Timothy Chu, Yu Gao, Richard Peng, Sushant Sachdeva, Saurabh Sawlani, and Junxing Wang. Graph sparsification, spectral sketches, and faster resistance computation, via short cycle decompositions. In *FOCS*, pages 361–372. IEEE Computer Society, 2018.

12 Erik D. Demaine and Mohammad Taghi Hajiaghayi. Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In *SODA*, pages 840–849. SIAM, 2004.

13 Sally Dong, Yin Tat Lee, and Kent Quanrud. Computing circle packing representations of planar graphs. In *SODA*, pages 2860–2875. SIAM, 2020.

14 Peter Eades and Sue Whitesides. The logic engine and the realization problem for nearest neighbor graphs. *Theor. Comput. Sci.*, 169(1):23–37, 1996.

15 David Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000.

16 Weiming Feng, Heng Guo, Chunyang Wang, Jiaheng Wang, and Yitong Yin. Towards derandomising Markov chain Monte Carlo. In *FOCS*, pages 1963–1990. IEEE, 2023.

17 Weiming Feng, Heng Guo, and Yitong Yin. Perfect sampling from spatial mixing. *Random Struct. Algorithms*, 61(4):678–709, 2022.

18 Andreas Galanis, Daniel Štefankovič, and Eric Vigoda. Inapproximability for antiferromagnetic spin systems in the tree nonuniqueness region. *J. ACM*, 62(6):50:1–50:60, 2015.

19 Andreas Galanis, Daniel Štefankovič, and Eric Vigoda. Inapproximability of the partition function for the antiferromagnetic Ising and hard-core models. *Combin. Probab. Comput.*, 25(4):500–559, 2016.

20 Heng Guo. *Complexity Classification of Exact and Approximate Counting Problems*. PhD thesis, University of Wisconsin - Madison, 2015.

21 Thomas P. Hayes and Alistair Sinclair. A general lower bound for mixing of single-site dynamics on graphs. *Ann. Appl. Probab.*, 17(3):931–952, 2007.

22 Mark Huber. Approximation algorithms for the normalizing constant of Gibbs distributions. *Ann. Appl. Probab.*, 25(2):974–985, 2015.

23 Mark Jerrum and Alistair Sinclair. Approximating the permanent. *SIAM J. Comput.*, 18(6):1149–1178, 1989.

24 Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM J. Comput.*, 22(5):1087–1116, 1993.

25 Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986.

26 Frank Kammer and Torsten Tholey. Approximate tree decompositions of planar graphs in linear time. *Theor. Comput. Sci.*, 645:60–90, 2016.

27 Vladimir Kolmogorov. A faster approximation algorithm for the Gibbs partition function. In *COLT*, volume 75 of *Proceedings of Machine Learning Research*, pages 228–249. PMLR, 2018.

28 Liang Li, Pinyan Lu, and Yitong Yin. Correlation decay up to uniqueness in spin systems. In *SODA*, pages 67–84. SIAM, 2013. Full version from arXiv at `arXiv:1111.7064`.

29 Viresh Patel and Guus Regts. Deterministic polynomial-time approximation algorithms for partition functions and graph polynomials. *SIAM J. Comput.*, 46(6):1893–1919, 2017.

30 Alistair Sinclair, Piyush Srivastava, and Marc Thurley. Approximation algorithms for two-state anti-ferromagnetic spin systems on bounded degree graphs. In *SODA*, pages 941–953. SIAM, 2012.

31 Allan Sly and Nike Sun. Counting in two-spin models on d-regular graphs. *Ann. Probab.*, 42(6):2383–2416, 2014.

32 Daniel Štefankovič, Santosh S. Vempala, and Eric Vigoda. Adaptive simulated annealing: A near-optimal connection between sampling and counting. *J. ACM*, 56(3):18:1–18:36, 2009.

33 Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

34 Dror Weitz. Counting independent sets up to the tree threshold. In *STOC*, pages 140–149. ACM, 2006.

35 Yitong Yin and Chihao Zhang. Approximate counting via correlation decay on planar graphs. In *SODA*, pages 47–66. SIAM, 2013.

A Lazy marginal samplers

Lemma 9 is proved in this subsection. The single-site Anand-Jerrum algorithm adapts to the hard-core model as in Algorithm 1.

Algorithm 1 HardcoreSampler$(G, \lambda, (\Sigma, \sigma), v)$.

Input: a Δ-degree graph G, fugacity λ, a set of vertices $\Sigma \subseteq V$ with a configuration $\sigma \in \Omega_\Sigma$, and vertex to sample $v \notin \Sigma$
Output: the partial configuration passed in with a spin at v: $(\Sigma, \sigma) \oplus (v, i)$ for some $i \in \{0, 1\}$.

Decrease the global timer $T \leftarrow T - 1$;
if *there exists* $u \in \Sigma \cap N(v)$ *such that* $\sigma(u) = 1$ **then**
  return $((\Sigma, \sigma) \oplus (v, 0))$;
Sample random $X \in \{\perp, 0\}$ with $\Pr[X = \perp] = \lambda/(1 + \lambda)$ and $\Pr[X = 0] = 1/(1 + \lambda)$;
if $X = \perp$ **then**
  $(\Sigma', \sigma') \leftarrow (\Sigma, \sigma)$;
  $Y \leftarrow 1$;
  forall $u \in N(v) \backslash \Sigma$ **do**
    $(\Sigma', \sigma') \leftarrow$ HardcoreSampler$(G, \lambda, (\Sigma', \sigma'), u)$;
    if $\sigma'(u) = 1$ **then** $Y \leftarrow 0$;
  return $((\Sigma, \sigma) \oplus (v, Y))$;
else
  return $((\Sigma, \sigma) \oplus (v, 0))$;

The correctness of the algorithm is summarised by the following theorem, adapted to our setting.

▶ **Theorem 23** ([1, Theorem 5.3]). *Suppose G is a graph with maximum degree bounded by Δ, and $\lambda < \lambda_c(\Delta)$. If the untruncated algorithm* HardcoreSampler$_{+\infty}(G, \lambda, (\Sigma, \sigma), v)$ *terminates with probability 1, then it generates a spin of v according to the correct marginal distribution upon termination, provided that the partial configuration (Σ, σ) is feasible.*

We remark that the correctness does not rely on the graph's neighbourhood growth being sub-exponential. However, the algorithm given here is a special case of that in [1], where they look at an ℓ-distance neighbourhood. Fixing $\ell = 1$ as we do here results in the regime of fugacity λ being worse than the critical λ_c, as we will see very soon. The saving grace of [1] is that other ℓ's might be chosen in order to get to the critical regime, but this is at the cost of limiting the neighbourhood growth. Our main algorithm does not work up to the critical λ_c, so only the 1-hop neighbourhood is considered.

In [1], the expected running time is studied and turns out to be a constant depending on the parameters of the model. However, we further need an exponential tail bound of the algorithm. This is done by the same idea of [16, Section B.3], though we do not truncate this algorithm as is done there. As soon as an exponential tail bound of running time is established, the algorithm then terminates with probability 1 and hence is correct.

We treat the algorithm as a branching process. Each time the algorithm recurses into its neighbourhood, it creates at most $\Delta - 1$ new copies of the routine HardcoreSampler. Such branching happens with probability $p := \lambda/(1 + \lambda)$. This leads us to study the following Markovian process that stochastically dominates the actual branching process. Let $(X_t)_{t \in \mathbb{Z}_{\geq 0}}$ be a discrete Markov chain where $X_t \in \mathbb{Z}_{\geq 0}$ with initial state $X_0 = 1$. This chain has an absorbing barrier at 0, and for any other $X_t > 0$, the transition probability is given by

$$X_{t+1} \leftarrow \begin{cases} X_t + \Delta - 1 & \text{with probability } p; \\ X_t - 1 & \text{with probability } 1 - p. \end{cases} \tag{6}$$

In the general case, the tail bound of this process is proved in [16, Lemma B.12], and this requires $\lambda \le \frac{1}{2e\Delta-1}$ when specialised to the hard-core model. Here we provide a stronger analysis to remove the constant.

▶ **Lemma 24.** *Suppose $\lambda < \frac{1}{\Delta-1}$. For any $0 < \varepsilon < 1$, let $T = \frac{2\Delta^2}{\left(\frac{\lambda}{1+\lambda}\Delta-1\right)^2}\log\frac{1}{\varepsilon}$. Then with probability at most ε, the process (X_t) defined by (6) does not terminate in T rounds.*

Proof. Given $\{X_t\}_{t\in\mathbb{Z}_{\ge 0}}$, define an auxillary process $\{Y_t\}_{t\in\mathbb{Z}_{\ge 0}}$ in the following way. Let $Y_0 = 1$, and the transition probability is given by

$$Y_{t+1} \leftarrow \begin{cases} Y_t + \frac{1}{1+\lambda}\Delta & \text{with probability } \frac{\lambda}{1+\lambda}; \\ Y_t - \frac{\lambda}{1+\lambda}\Delta & \text{with probability } \frac{1}{1+\lambda}. \end{cases} \tag{7}$$

Then couple X_t with Y_t perfectly that, if X_t increases then so does Y_t, and vice versa, till X_t reaches the absorbing barrier. After this point, Y_t just performs the above transition independently.

Clearly, $\{Y_t\}$ is a martingale, and if $X_t > 0$ is not absorbed then $Y_t = X_t + \left(\frac{\lambda}{1+\lambda}\Delta - 1\right)t$. Also note that the regime on λ ensures $\frac{\lambda}{1+\lambda}\Delta - 1 > 0$. This allows us to bound the probability of $\{X_t\}$ not terminating after T rounds by by applying Azuma–Hoeffding inequality:

$$\Pr[X_T > 0] = \Pr[X_T \ge X_0] = \Pr\left[Y_T - Y_0 \ge T\left(\frac{\lambda}{1+\lambda}\Delta - 1\right)\right]$$

$$\le \exp\left\{-\frac{T^2\left(\frac{\lambda}{1+\lambda}\Delta - 1\right)^2}{2\Delta^2 T}\right\} = \varepsilon. \qquad \blacktriangleleft$$

Lemma 9 then follows by exactly the same argument as in [16, Proof of Lemma B.10], by noticing that the branching process (X_t) stochastically dominates the number of "active" instances of HardcoreSampler, and using Lemma 24.

If we want to cover the whole strong spatial mixing regime but only work on amenable graphs, then we can invoke the original Anand-Jerrum algorithm, allowing us to do recursion at farther vertices rather than one-hop neighbours. Its running time tail bound is shown in [16, Lemma B.10].

▶ **Theorem 25** ([16, Lemma B.10])**.** *Suppose a q-spin system $\mathcal{S} = (G, [q], \boldsymbol{b}, \boldsymbol{A})$ exhibits strong spatial mixing with decay rate $f(\ell)$, and there is a function $s(\ell)$ such that the neighbourhood growth of G satisfies $|\{u \mid \mathrm{dist}_G(u,v) = \ell\}| \le s(\ell)$ for all v. If there is some $r \in \mathbb{Z}_{\ge 1}$ such that $2eq(1+s(r))f(r) \le 1$, then for any feasible boundary configuration (Σ, σ), the Anand-Jerrum algorithm, on input $(\mathcal{S}, (\Sigma, \sigma), v, r)$, generates a sample of v subject to the correct marginal distribution, and halts in time $O(s(r)\log\frac{1}{\varepsilon})$ with probability at least $1-\varepsilon$.*

B A lower bound for Weitz's algorithm

In this section, we prove a lower bound for the running time of the standard implementation of Weitz's algorithm. Consider the hard-core model on $G = (V, E)$ with parameter λ. Suppose we want to estimate the partition function Z within a *constant* approximation error. Let $V = \{v_1, \ldots, v_n\}$ and $G_i = G \setminus \{v_1, \ldots, v_{i-1}\}$. Weitz's algorithm solves this task by estimating each $\mu_{G_i, v_i}(0)$ within an approximation error $O(\frac{1}{n})$. It first constructs the SAW tree of G_i rooted at v_i, then truncates the tree at level ℓ and applies dynamic programming on the truncated tree to estimate $\mu_{G_i, v_i}(0)$. The standard implementation

of Weitz's algorithm [34, 28] ensures that for any tree with maximum degree Δ, any two configurations σ, τ at level ℓ, $d_{\mathrm{TV}}(\mu_v^\sigma, \mu_v^\tau) = O(\frac{1}{n})$. Standard analysis bounds the total running time from above by $T_{\mathrm{Weitz}} = \Theta(n\Delta^\ell)$.

By the same correlation decay analysis as in Lemma 11, when the algorithm in Theorem 1 has running time $\widetilde{O}(n^{1+1/2k})$, we need to choose ℓ so that $T_{\mathrm{Weitz}} = O(n^{1+1/k})$. This analysis only gives an upper bound on the correlation decay rate. If the decay rate is faster, then Weitz's algorithm is faster, and so is the algorithm in Theorem 1. The speedup will depend on how much faster the decay rate becomes. Nevertheless, the next lemma shows that the analysis in Lemma 11 is almost sharp in the worst case. The speedup in Theorem 1 is at least $\widetilde{\Omega}\left(n^{\frac{1}{2k} - O(\frac{1}{k^2 \log \Delta})}\right)$.

▶ **Lemma 26.** *Let the real number $k > 0$ and the integer $\Delta \geq 2$ be two constants satisfying $\Delta^k \geq 4$. Let $\lambda = \frac{2}{(\Delta-1)\Delta^k}$. Let T be an infinite Δ-regular tree with root v. For any $\ell \geq 2$, let σ_0 and σ_1 be all-0 and all-1 configurations at level ℓ of T respectively. The Gibbs distribution μ of the hard-core model on T with parameter λ satisfies*

$$d_{\mathrm{TV}}(\mu_v^{\sigma_0}, \mu_v^{\sigma_1}) \geq \frac{1}{2}\left(\frac{1}{\Delta^k}\right)^\ell.$$

Let the parameters k, Δ, and λ be as in Lemma 26. Consider a family of hard-core instances where the graphs are indeed Δ-regular trees. In Weitz's algorithm, in order to ensure an $O(\frac{1}{n})$ truncation error, Lemma 26 implies that ℓ must satisfy $\frac{1}{2}\left(\frac{1}{\Delta^k}\right)^\ell = O(\frac{1}{n})$, namely,

$$\Delta^\ell = \Omega(n^{\frac{1}{k}}).$$

This makes the overall running time $T_{\mathrm{Weitz}} = \Omega(n^{1+\frac{1}{k}})$. In comparison, for these parameters, the algorithm in Theorem 1 has a running time upper bound $\widetilde{O}\left(n^{1+\frac{1}{2k}+O(\frac{1}{k^2 \log \Delta})}\right)$, which is faster by a factor of roughly $\widetilde{\Omega}(n^{1/2k})$.

Proof of Lemma 26. Let w be an arbitrary vertex at level $0 \leq t \leq \ell$. Let π denote the Gibbs distribution on the subtree rooted T_w at w. Recall that σ_0, σ_1 are pinnings on $T(\ell)$, where $T(\ell)$ is level ℓ of T. Let $p_t^0(c) = \pi_w^{\sigma_0}(c)$ and $p_t^1(c) = \pi_w^{\sigma_1}(c)$ for $c \in \{0, 1\}$, where we use σ_0 and σ_1 to denote all-0 and all-1 pinnings on $T_w \cap T(\ell)$. By symmetry, $p_t^0(\cdot)$ and $p_t^1(\cdot)$ depend only on t but not on w. In particular, $p_0^0 = \mu_v^{\sigma_0}$ and $p_0^1 = \mu_v^{\sigma_1}$ for the root v. For any $0 \leq t \leq \ell$, define

$$R_t^0 := \frac{p_t^0(1)}{p_t^0(0)}, \quad R_t^1 := \frac{p_t^1(1)}{p_t^1(0)}.$$

We next prove the following result holds for all $1 \leq t \leq \ell - 1$:

$$\left|R_t^0 - R_t^1\right| \geq \frac{1}{2}\left(\frac{1}{\Delta^k}\right)^{\ell-t-1}. \tag{8}$$

We need the following bound to prove (8). By considering the worst pinning on the neighbourhood, we have the following bound on both ratios R_s^0 and R_s^1

$$\forall 0 \leq s \leq \ell - 1, \quad R_s^0, R_s^1 \leq \lambda = \frac{2}{(\Delta-1)\Delta^k}. \tag{9}$$

We prove (8) by induction on t from $\ell-1$ to 1. The base case is $t=\ell-1$. Note that $\Delta^k \geq 4$. A straightforward calculation shows that

$$\left|R^0_{\ell-1} - R^1_{\ell-1}\right| = |1-\lambda| = 1 - \frac{2}{(\Delta-1)\Delta^k} \geq \frac{1}{2}.$$

For the induction step, fix $1 \leq t \leq \ell-2$. The recursion function in $(\Delta-1)$-ary tree is

$$f(x) = \lambda\left(\frac{1}{1+x}\right)^{\Delta-1}.$$

Note that $R^0_t = f(R^0_{t+1})$ and $R^1_t = f(R^1_{t+1})$. By the mean value theorem, there exists $\min(R^0_{t+1}, R^1_{t+1}) < \theta < \max(R^0_{t+1}, R^1_{t+1})$ such that

$$\left|R^0_t - R^1_t\right| = |f'(\theta)| \cdot \left|R^0_{t+1} - R^1_{t+1}\right|.$$

By (9) and the fact $\Delta^k \geq 4$, we have

$$\begin{aligned}|f'(\theta)| &= \lambda(\Delta-1)\left(\frac{1}{1+\theta}\right)^{\Delta} \geq \lambda(\Delta-1)\left(\frac{1}{1+\lambda}\right)^{\Delta} \geq \lambda(\Delta-1)\exp(-\lambda\Delta)\\ &= \frac{2}{\Delta^k}\exp\left(-\frac{2\Delta}{(\Delta-1)\Delta^k}\right) \geq \frac{2}{\Delta^k}\exp\left(-\frac{4}{\Delta^k}\right) \geq \frac{1}{\Delta^k}.\end{aligned}$$

By the induction hypothesis that $\left|R^0_{t+1} - R^1_{t+1}\right| \geq \frac{1}{2}(\frac{1}{\Delta^k})^{\ell-t-2}$, we can prove (8) for t. This finishes the induction step for $1 \leq t \leq \ell-3$.

Finally, we use (8) to bound $|R^0_0 - R^1_0|$. The proof is similar to the proof in the induction step. The only difference is that the recursion for root v becomes $g(x) = \lambda\left(\frac{1}{1+x}\right)^{\Delta}$. By a similar calculation, there exists $\min(R^0_1, R^1_1) < \theta < \max(R^0_1, R^1_1)$ such that

$$\begin{aligned}|R^0_0 - R^1_0| = |g'(\theta)| \cdot \left|R^0_1 - R^1_1\right| &\geq \Delta\lambda\left(\frac{1}{1+\theta}\right)^{\Delta+1} \cdot \frac{1}{2}\left(\frac{1}{\Delta^k}\right)^{\ell-2}\\ &= \frac{\Delta}{(\Delta-1)(1+\theta)} \cdot \lambda(\Delta-1)\left(\frac{1}{1+\theta}\right)^{\Delta} \cdot \frac{1}{2}\left(\frac{1}{\Delta^k}\right)^{\ell-2}\\ &\geq \frac{\Delta}{(\Delta-1)(1+\lambda)} \cdot \frac{1}{2}\left(\frac{1}{\Delta^k}\right)^{\ell-1} \geq \frac{1}{3}\left(\frac{1}{\Delta^k}\right)^{\ell-1}.\end{aligned}$$

By the definitions of R^0_0 and R^1_0 and the fact $\Delta^k \geq 4$, we have

$$\begin{aligned}d_{\mathrm{TV}}(\mu^{\sigma_0}_v, \mu^{\sigma_1}_v) = |\mu^{\sigma_0}_v(1) - \mu^{\sigma_1}_v(1)| &= \mu^{\sigma_0}_v(0)\mu^{\sigma_1}_v(0)|R^0_0 - R^1_0|\\ &\geq \left(\frac{1}{1+\lambda}\right)^2 \cdot \frac{1}{3}\left(\frac{1}{\Delta^k}\right)^{\ell-1} \geq \frac{4}{27}\left(\frac{1}{\Delta^k}\right)^{\ell-1} \geq \frac{1}{2}\left(\frac{1}{\Delta^k}\right)^{\ell}.\end{aligned}$$

◀

From Proof Complexity to Circuit Complexity via Interactive Protocols

Noel Arteche
Lund University, Sweden
University of Copenhagen, Denmark

Erfan Khaniki
Institute of Mathematics of the Czech Academy of Sciences, Prague, Czech Republic

Ján Pich
University of Oxford, UK

Rahul Santhanam
University of Oxford, UK

Abstract

Folklore in complexity theory suspects that circuit lower bounds against $\mathbf{NC}^1$ or $\mathbf{P}/\text{poly}$, currently out of reach, are a necessary step towards proving strong proof complexity lower bounds for systems like Frege or Extended Frege. Establishing such a connection formally, however, is already daunting, as it would imply the breakthrough separation $\mathbf{NEXP} \not\subseteq \mathbf{P}/\text{poly}$, as recently observed by Pich and Santhanam [58].

We show such a connection conditionally for the Implicit Extended Frege proof system (iEF) introduced by Krajíček [45], capable of formalizing most of contemporary complexity theory. In particular, we show that if iEF proves efficiently the standard derandomization assumption that a concrete Boolean function is hard on average for subexponential-size circuits, then any superpolynomial lower bound on the length of iEF proofs implies $\#\mathbf{P} \not\subseteq \mathbf{FP}/\text{poly}$ (which would in turn imply, for example, $\mathbf{PSPACE} \not\subseteq \mathbf{P}/\text{poly}$). Our proof exploits the formalization inside iEF of the soundness of the sum-check protocol of Lund, Fortnow, Karloff, and Nisan [54]. This has consequences for the self-provability of circuit upper bounds in iEF. Interestingly, further improving our result seems to require progress in constructing interactive proof systems with more efficient provers.

2012 ACM Subject Classification Theory of computation → Proof complexity; Theory of computation → Circuit complexity; Theory of computation → Complexity theory and logic

Keywords and phrases proof complexity, circuit complexity, interactive protocols

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.12

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2405.02232

Funding *Noel Arteche*: This work was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.
Erfan Khaniki: This work was supported by the Institute of Mathematics of the Czech Academy of Sciences (RVO 67985840) and GAČR grant 19-27871X.
Ján Pich: This work received support from the Royal Society University Research Fellowship URF\R1\211106 "Proof complexity and circuit complexity: a unified approach".

Acknowledgements Independently, Albert Atserias suggested to us to consider using interactive proof systems to derive this type of connections. We thank Pavel Pudlák for useful comments, and the anonymous reviewers for relevant comments and references. This work was done in part while the first author was visiting the University of Oxford and the Institute of Mathematics of the Czech Academy of Sciences. For the purpose of Open Access, the authors have applied a CC BY public copyright license to any Author Accepted Manuscript version arising from this submission.

1 Introduction

At a high level, both circuit complexity and proof complexity can be thought of as an approach towards the **P** versus **NP** question. The circuit complexity program, which met with considerable success in the 1980s, tries to prove lower bounds against gradually larger circuit classes, hoping to eventually show $\mathbf{NP} \not\subseteq \mathbf{P}/\text{poly}$. Proof complexity, often identified with the so-called Cook-Reckhow program, intends to show $\mathbf{NP} \neq \mathbf{coNP}$ and, in turn, $\mathbf{P} \neq \mathbf{NP}$, by proving lower bounds against gradually more powerful proof systems for propositional logic.

While both enterprises share the motivation to study *concrete* computational models of increasing power hoping to build up techniques to attack the long-sought separations, there exist notable differences. Circuit complexity looks at deterministic models of computation, while proof complexity deals with proof systems, which are inherently non-deterministic. Furthermore, while circuit complexity has a clear end-goal (lower bounds against general Boolean circuits), it remains wide open whether the Cook-Reckhow program can be realized even in principle. It is not known whether lower bounds against strong systems like Extended Frege can imply lower bounds for every other system and, as such, one could potentially keep proving lower bounds for ever-stronger systems without ever settling whether $\mathbf{NP} \neq \mathbf{coNP}$.

The parallels between circuit complexity and proof complexity are made clearer by Frege systems. For each circuit complexity class $\mathcal{C}$, one can define the proof system $\mathcal{C}$-Frege, in which proof lines are restricted to be circuits from $\mathcal{C}$. In this setting strong systems like Frege and Extended Frege correspond to $\mathbf{NC}^1$-Frege and $\mathbf{P}/\text{poly}$-Frege, respectively, and thus the natural question arises: Can we turn explicit lower bounds for $\mathcal{C}$ circuits into lower bounds for $\mathcal{C}$-Frege systems, and vice versa?

While the question is essentially open, work on weaker systems and circuit classes has proven successful. In one direction, the method of feasible interpolation [43, 65, 50] (see [51, §17.9.1] for the history of the method) has been extensively applied to obtain proof complexity lower bounds. The framework of feasible interpolation formalizes the idea of extracting computational content from proofs: given short proofs in a given system, one can extract a small Boolean circuit in some restricted classes for a related interpolant function. Contrapositively, circuit lower bounds for such functions (often coming from unconditional results such as lower bounds against monotone circuits [64, 4, 3]), turn into lower bounds for proofs systems like Resolution [50] or Cutting Planes [61] (and conditionally for other systems, such as Polynomial Calculus or Sum-of-Squares [32]). Unfortunately, this connection breaks for stronger proof systems: already $\mathbf{AC}^0$-Frege and $\mathbf{TC}^0$-Frege are known to lack feasible interpolation properties[1] under standard cryptographic hardness assumptions [52, 13, 12], and this holds even if we allow feasible interpolation by quantum circuits [6].

In the other direction (circuit complexity from proof complexity), the theory of lifting has unveiled deep connections between proofs, circuits and communication protocols. Here, so-called query-to-communication lifting theorems translate query complexity lower bounds (corresponding to weak systems, like Resolution) into communication complexity lower bounds (e.g. [63, 53]). The latter provide restricted circuit lower bounds, such as for monotone circuits (see e.g. [27, 24, 25] and references therein). It is, however, not known how to derive non-monotone lower bounds for unrestricted Boolean circuits by lifting proof complexity lower bounds.

[1] Some of these systems are known to admit some form of interpolation by stronger computational models, see e.g. [62, 23], but we are interested in Boolean circuits.

For proper Frege systems, the connection has worked mostly in one direction, from circuits to proofs, particularly at the level of techniques. The method of random restrictions and the celebrated switching lemmas used to show constant-depth circuit lower bounds in the 1980s [26, 1, 33] were successfully transferred into $\mathbf{AC}^0$-Frege lower bounds shortly after [2, 10, 43, 8, 59, 49]. This suggests that understanding what makes proof lines large might be necessary to understand why proofs are long. Intriguingly, understanding the proof lines alone does not seem to suffice: the $\mathbf{AC}^0[p]$ lower bounds of Razborov and Smolensky [66, 68] are yet to be successfuly translated to proof complexity, with lower bounds for $\mathbf{AC}^0[2]$-Frege being one of the prominent frontier problems in the field.

The current situation seems to suggest that in order to make progress towards proof complexity lower bounds, it is *necessary* (though seemingly not sufficient) to first obtain strong enough circuit lower bounds. In particular, under this folklore belief, circuit lower bounds against $\mathbf{NC}^1$ or $\mathbf{P}/\text{poly}$, currently out or reach, would be a necessary step towards proving strong proof complexity lower bounds for systems like Frege or Extended Frege. However, the suspicion remains unproven, and no generic way of deriving explicit circuit lower bounds for unrestricted Boolean circuits from proof complexity lower bounds for concrete propositional proof systems has been discovered[2].

The first result giving such a connection under relatively conventional assumptions which are presumably weaker than the conclusion of the connection itself was presented recently by Pich and Santhanam [58]. Specifically, they showed that any superpolynomial lower bound on the length of tautologies in the Extended Frege system EF implies $\mathbf{NP} \not\subseteq \mathbf{P}/\text{poly}$ assuming hypotheses (1) and (2) below:

(1) (*Provable circuit lower bound.*) EF proves efficiently that a concrete Boolean function in $\mathbf{E}$ is average-case hard for subexponential-size circuits.

(2) (*Provable reduction of OWFs to* $\mathbf{P} \neq \mathbf{NP}$.) EF proves efficiently that a polynomial-time function transforms circuits breaking one-way functions into circuits solving SAT.

We remark that Hypothesis (1) above presupposes $\mathbf{E} \not\subseteq \mathbf{P}/\text{poly}$, which is however believed to be a significantly weaker statement than $\mathbf{NP} \not\subseteq \mathbf{P}/\text{poly}$. Alternatively, Hypotheses (1) and (2) can be replaced by a single assumption on the feasible provability of the existence of antickeckers in EF. These results remain valid even if we replace EF by an essentially arbitrary proof system simulating EF.

Crucially, improving this and related results by dropping the hypotheses is surprisingly daunting. As noted by Pich and Santhanam [58, Prop. 1], if one unconditionally establishes the implication "if S is not polynomially bounded, then $\mathbf{NP} \not\subseteq \mathbf{P}/\text{poly}$" for a concrete proof system S, then the breakthrough separation $\mathbf{NP} \not\subseteq \mathbf{SIZE}[n^k]$, for every fixed k (and $\mathbf{NEXP} \not\subseteq \mathbf{P}/\text{poly}$) follows!

In short, proving a formal connection between proof complexity and circuit complexity provably requires breakthrough circuit lower bounds! Despite this setback, one can still hope to get evidence that points at these connections, possibly by shifting some of the components of the ingredients. Namely, one may try to (a) adopt some hardness assumption, in the style of [58]; (b) conclude lower bounds weaker than $\mathbf{NP} \not\subseteq \mathbf{P}/\text{poly}$; or (c) look at non-Cook-Reckhow proof systems (such as MA proof systems or proof systems for languages beyond $\mathbf{coNP}$).

[2] We note that the issue lies in establishing such a connection for a *concrete* system. Of course, the statement "there is a proof system S such that if S is not polynomially bounded, then $\mathbf{P} \neq \mathbf{NP}$" is true: if $\mathbf{NP} = \mathbf{coNP}$ the implication is vacuously true by taking a polynomially bounded proof system; if $\mathbf{NP} \neq \mathbf{coNP}$, then $\mathbf{P} \neq \mathbf{NP}$ and thus the statement holds for any proof system. It would be dramatically different to obtain such a connection for a concrete system.

In this style, Grochow and Pitassi [31] showed that the Ideal Proof System (IPS) does satisfy such a connection, to *algebraic* circuit complexity. Indeed, any superpolynomial lower bound in the length of proofs in $\mathsf{IPS}_{\mathbb{F}}$ implies $\mathbf{VP}_{\mathbb{F}} \neq \mathbf{VNP}_{\mathbb{F}}$. Grochow and Pitassi avoid the Pich-Santhanam barrier by means of (b) and (c) above: first, IPS is not known to be a Cook-Reckhow system, since proofs are verified by randomized machines via polynomial identity testing; second, the lower bounds are algebraic and not Boolean. Recall that while separating **VP** and **VNP** is a necessary step[3] towards $\mathbf{NP} \not\subseteq \mathbf{P}/\mathrm{poly}$ [14], the converse is not known.

Another interesting connection has been established in the realm of quantified Boolean formulas, where the connection can be made essentially optimal. Beyersdorff, Bonacina, Chew, and Pich [11] showed that for every circuit class $\mathcal{C}$, the quantified system $\mathcal{C}$-Frege + $\forall$red is not polynomially bounded if and only if either $\mathbf{PSPACE} \not\subseteq \mathcal{C}$ or $\mathcal{C}$-Frege is not polynomially bounded. Here, $\mathcal{C}$-Frege + $\forall$red stands for the natural quantified system obtained by extending $\mathcal{C}$-Frege with a universal reduction rule, which takes care of universal quantifiers by instantiating concrete values for its variables in the hope of refuting the formula. The reason this avoids the Pich-Santhanam barrier is the disjunct in the conclusion. That is, in the context of QBF the conclusion of the Pich-Santhanam barrier becomes that that either $\mathbf{NEXP} \not\subseteq \mathbf{P}/\mathrm{poly}$ or $\mathcal{C}$-Frege is not polynomially bounded. But this disjunction is no breakthrough, since it follows directly by a diagonalization argument anyway: if a propositional system is polynomially bounded, then **NEXP** is hard for $\mathbf{P}/\mathrm{poly}$ [44].

Contributions

We prove a new conditional connection between proof complexity and circuit complexity, giving further evidence that strong proof complexity lower bounds require circuit lower bounds. This constitutes the first example of a natural proof system that is conditionally Cook-Reckhow and whose lower bounds imply Boolean circuit lower bounds.

The system in question is (an extension of) the Implicit Extended Frege (iEF) proof system of Krajíček [45], capable of formalizing most of contemporary complexity theory. Our result can be informally stated as follows, where $\mathsf{iEF}^{\mathrm{tt}(h)}$ stands for the proof system extending iEF by axioms $\mathrm{tt}^{\mathrm{avg}}_{1/4}(h_n, 2^{n/4})$ claiming there are no circuits of size $2^{n/4}$ approximating a concrete function h on more than a $(1/2 + 1/2^{n/4})$-fraction of the inputs.[4]

▶ **Theorem 1** (Main theorem, informal). *Suppose there is a Boolean function $h \in \mathbf{NE} \cap \mathbf{coNE}$ that is hard on average for subexponential-size circuits. If the Cook-Reckhow proof system $\mathsf{iEF}^{\mathrm{tt}(h)}$ is not polynomially bounded, then $\#\mathbf{P} \not\subseteq \mathbf{FP}/\mathrm{poly}$.*

In the theorem above one could instead consider the system $\mathsf{iEF}^{\mathrm{tt}(h)}$ for some unconditionally hard function family h that is guaranteed to exist. The only problem in this case is that we might need non-uniform advice to verify the proofs, and so the system would not be Cook-Reckhow (we refer to Cook and Krajíček [19] for a systematic treatment of non-uniform proof systems).

One can interpret our theorem as improving on the connection of Pich and Santhanam [58] from proof complexity to circuit complexity. Our result improves that of Pich and Santhanam by completely dropping their second assumption (the one about EF proving the existence of one-way functions under $\mathbf{P} \neq \mathbf{NP}$). The price to pay for these changes is two-fold:

[3] Unconditionally over finite fields, and assuming the Generalized Riemann Hypothesis for infinite fields.
[4] For technical reasons, we define $\mathsf{iEF}^{\mathrm{tt}(h)}$ using a system which is polynomially equivalent to iEF instead of iEF itself, see Definition 17.

1. we need to replace EF by the seemingly stronger Implicit Extended Frege system (iEF). Informally, iEF extends EF with an extra rule allowing us to derive a formula φ after we have derived that a truth table of a given circuit encodes an EF-proof of φ. Such a circuit is called an *implicit* proof;
2. we can conclude only $\#\mathbf{P} \not\subseteq \mathbf{FP}/\text{poly}$ from iEF lower bounds, instead of $\mathbf{NP} \not\subseteq \mathbf{P}/\text{poly}$.

One may also compare our result to that of Grochow and Pitassi [31], who showed $\mathbf{VP} \neq \mathbf{VNP}$ (and hence hardness of computing the permanent) would follow from IPS lower bounds. Like our result, the IPS proof system is only conditionally Cook-Reckhow. Indeed, IPS is a Merlin-Arthur proof system which can be derandomized[5] under standard assumptions, like $\mathbf{E}$ being hard to approximate by subexponential-size circuits. Our result is in some sense stronger in that the lower bounds obtained are Boolean rather than algebraic. However, we seem to be getting to lower bounds for the same problem as Grochow and Pitassi, since computing the permanent is both $\mathbf{VNP}$-complete and $\#\mathbf{P}$-complete.

We note that the requirement that $h \in \mathbf{NE} \cap \mathbf{coNE}$ is not strictly needed and, in fact, one can phrase the result in a more general style (as we do in the technical part) in which the connection holds for any extension of iEF by truth table formulas for any hard function. Observe, however, that iEF is a very strong proof system, with its bounded arithmetic counterpart being the theory V^1_2 (or $\mathsf{S}^1_2 + \text{1-EXP}$, in the first-order setting), and so it is plausible that iEF already proves such a circuit lower bound. For example, already EF can prove efficiently the PCP theorem [57], $\mathbf{AC}^0$, $\mathbf{AC}^0[2]$ and monotone circuit lower bounds [67, 55], or the hardness amplification producing average-case hard functions in $\mathbf{E}$ from worst-case hard functions in $\mathbf{E}$ [36]. Furthermore, iEF proves efficiently the correctness of Zhuk's algorithm from a CSP dichotomy [28, 29]. Hence, it is plausible to imagine that if circuit lower bounds are at all provable, they may well be provable already in iEF. If that turned out to be the case, then the concrete proof system in our main theorem becomes iEF itself.

▶ **Corollary 2** (Main theorem, restated). *Assume that* iEF *proves efficiently* $\mathrm{tt}^{\mathrm{avg}}_{1/4}(h_n, 2^{n/4})$ *for some function family* h *and each sufficiently big* n*. Then, if* iEF *is not polynomially bounded,* $\#\mathbf{P} \not\subseteq \mathbf{FP}/\text{poly}$.

Let us note that one cannot make big improvements to this result without hitting the Pich-Santhanam barrier that implies $\mathbf{NEXP} \not\subseteq \mathbf{P}/\text{poly}$ unconditionally: if we managed to prove Theorem 1 for a Cook-Reckhow proof system, then $\mathbf{NEXP} \not\subseteq \mathbf{P}/\text{poly}$ would follow unconditionally. On the other hand, if our final goal is to prove $\mathbf{FP} \neq \#\mathbf{P}$, then the assumption of Theorem 1 is given to us for free even for some hard $h \in \mathbf{E}$, as otherwise, if $\mathbf{E}$ can be computed by subexponential-size circuits, it is not hard to show that $\mathbf{P} \neq \mathbf{NP}$ [44].

Consequences for self-provability of circuit upper bounds

Our result has consequences for the self-provability of circuit upper bounds. Suppose that $\#\mathbf{P} \subseteq \mathbf{FP}/\text{poly}$. Then, there is a sequence of polynomial-size circuits $\{C_n\}_{n\in\mathbb{N}}$ that on input a formula φ of size n, outputs a satisfying assignment if one exists. This means that the propositional formula $\mathrm{SAT}_n(\varphi, \alpha) \to \mathrm{SAT}_n(\varphi, C_n(\varphi))$ claiming the correctness of C_n as a SAT solver is tautological (where SAT_n is the satisfiability predicate, taking a formula φ and

[5] In fact, derandomizing IPS at all by simulating it by a Cook-Reckhow system implies a non-trivial derandomization of polynomial identity testing to $\mathbf{NP}$ [30]; this, in turn, implies some circuit lower bounds, as shown by Kabanets and Impagliazzo [38].

an assignment α and evaluating the formula). But by Theorem 1, $\mathsf{iEF}^{\mathrm{tt}}$ is now polynomially bounded, and so the proof system is able to efficiently argue for the correctness of the circuits. Namely, the mere validity of the upper bound $\#\mathbf{P} \subseteq \mathbf{FP}/\mathrm{poly}$ would imply the efficient propositional provability of $\mathrm{SAT} \in \mathbf{P}/\mathrm{poly}$.

Outline of the proof

Our main result follows from a derandomization of the known fact that $\mathbf{coNP} \not\subseteq \mathbf{MA}$ implies $\#\mathbf{P} \not\subseteq \mathbf{FP}/\mathrm{poly}$ (see, for example, [5, Thm. 8.22]), together with a formalization of the underlying MA system in a suitable theory of bounded arithmetic. The implication holds, actually, for the MA system given by the sum-check protocol of Lund, Fortnow, Karloff, and Nisan [54] in which proofs consist of a circuit simulating the moves of the Prover in the protocol, so that given such a circuit, the Verifier can simulate the entire protocol on their own with the aid of randomness. If $\#\mathbf{P} \subseteq \mathbf{FP}/\mathrm{poly}$, then the $\#\mathbf{P}$-powerful Prover in the sum-check protocol can be replaced by a polynomial-size circuit and thus the system is a polynomially bounded Merlin-Arthur system. Clearly, lower bounds on the length of proofs in this system are exactly circuit lower bounds against $\#\mathbf{P}$.

Since $\mathbf{MA}$ can be derandomized under standard hardness assumptions, assuming, for example, that $\mathbf{E}$ is hard for subexponential-size circuits, the proof system R based on the sum-check protocol above becomes a Cook-Reckhow system such that if R is not polynomially bounded, then $\#\mathbf{P} \not\subseteq \mathbf{FP}/\mathrm{poly}$. This is almost our goal. Our task now is to replace this system by a different more standard Cook-Reckhow system S. This can be achieved by proving efficiently the reflection principle of the system R in S, which essentially amounts to proving the soundness of the sum-check protocol in S. Here, we employ a recent work of Khaniki [40], in which the soundness of the sum-check protocol was formalized in $\mathsf{S}^1_2 + \mathsf{1\text{-}EXP}$.

In order to translate the formalization inside $\mathsf{S}^1_2 + \mathsf{1\text{-}EXP}$ into propositional logic, we need to express the soundness of the sum-check protocol by propositional formulas. This is achieved using the machinery of approximate counting of Jeřábek [37], which exploits Nisan-Wigderson generators based on a hard Boolean function.

Open problems

Improving our result seems to require significant conceptual work. Of course, simultaneously dropping the circuit lower bound assumption as well as getting the stronger separation $\mathbf{NP} \not\subseteq \mathbf{P}/\mathrm{poly}$ would already imply $\mathbf{NEXP} \not\subseteq \mathbf{P}/\mathrm{poly}$, but one may hope to improve the existing connection by improving on one of the two fronts only. Interestingly, this seems to require progress in some of the central open questions in the theory of interactive proof systems or in hardness magnification.

The power of the prover

Is it possible to strengthen the conclusion of the main theorem all the way down to $\mathbf{NP} \not\subseteq \mathbf{P}/\mathrm{poly}$? This would follow, for example, if we managed to design an interactive protocol for TAUT with a prover solving only $\mathbf{NP}$ problems and prove its correctness in iEF (unlike the current situation, where the prover is required to compute a $\#\mathbf{P}$-complete function). The general question of constructing a protocol for a language L where the prover's power is limited to $\mathbf{P}^L$ is a well-known open problem in the theory of interactive proof systems (see, for example, [5, §8.4]).

Note, of course, that the existence of such a protocol does not suffice, since its soundness must be provable inside iEF. In fact, the reason why we require iEF (or $\mathsf{S}^1_2 + \mathsf{1\text{-}EXP}$) to carry out the formalization of the existing sum-check protocol is that one cannot feasibly talk about #SAT directly in EF or S^1_2 (unless $\mathbf{FP} = \#\mathbf{P}$).

Hardness magnification

Is it possible to replace iEF in the main theorem by Gentzen's system G, or even by Extended Frege? One option would be to carry out the existing formalization inside EF, as mentioned above. The caveat would be, however, that we would then have to make the assumption on truth table tautologies for EF. Whether EF can prove general circuit lower bounds at all seems much less believable than for iEF, and so the plausibility of our hypotheses seems affected.

Instead, one may choose to keep everything in iEF and obtain the connection indirectly for EF via hardness magnification. Is there a natural class of formulas over which EF simulates iEF (and which are believably hard for EF)? If so, assuming hardness of these formulas for EF would imply iEF lower bounds. By our main theorem, $\#\mathbf{P} \not\subseteq \mathbf{P}/\mathrm{poly}$ would follow. To the best of our knowledge, no such type of hardness magnification is known for strong proof systems.

2 Preliminaries

We assume familiarity with the central concepts of computational complexity theory, propositional proof complexity and mathematical logic. Below we review the central concepts needed in this paper and fix some notation.

2.1 Proof complexity

Following Cook and Reckhow [22], a *propositional proof system* S for the language $\textsc{Taut}$ of propositional tautologies is a polynomial-time surjective function $S : \{0,1\}^* \to \textsc{Taut}$ taking as input a proof $\pi \in \{0,1\}^*$ and outputting $S(\pi) = \varphi$, the theorem that π proves. Soundness follows from the fact that the range is exactly $\textsc{Taut}$, and implicational completeness is guaranteed by the fact that S is surjective. We sometimes drop the term *proof* in *proof system* and use the term *system* alone to refer to a function S that is not guaranteed to be a Cook-Reckhow proof system (perhaps because it is unsound, or not deterministically computable).

We denote by $\mathsf{size}_S(\varphi)$ the *size* of the smallest S-proof of φ plus the size of φ. A proof system S is *polynomially-bounded* if for every $\varphi \in \textsc{Taut}$, $\mathsf{size}_S(\varphi) \leq |\varphi|^{O(1)}$. We say that a proof system S *polynomially simulates* a system Q, written $S \geq Q$, if for every $\varphi \in \textsc{Taut}$, $\mathsf{size}_S(\varphi) \leq \mathsf{size}_Q(\varphi)^{O(1)}$. Note that the notion of size and the definition of simulation do not exploit the soundness requirement of Cook-Reckhow systems. In particular, an unsound system can be polynomially bounded and simulate every other system. In some cases simulations hold only for some set T of tautologies, such as the set of tautologies written as 3DNFs, and not for all formulas, and then we say that S polynomially simulates Q over T. Given a family $\{\varphi_n\}_{n\in\mathbb{N}}$ of propositional tautologies, we write $S \vdash \varphi_n$ whenever $\mathsf{size}_S(\varphi_n) \leq |\varphi_n|^{O(1)}$.

2.1.1 Frege systems

Proof complexity studies a wide variety of proof systems. The most important ones for us are *Frege systems*. A Frege system is a finite set of axiom schemas and inference rules that are sound and implicationally complete for the language of propositional tautologies built from the Boolean connectives negation ($\neg$), conjunction ($\wedge$), and disjunction ($\vee$). A Frege proof is a sequence of formulas where each formula is obtained by either substitution of an axiom schema or by application of an inference rule on previously derived formulas. The specific choice of rules does not affect proof size up to polynomial factors, as long as there are only finitely many rules and these are sound and implicationally complete. Indeed, Frege systems polynomially simulate each other [51, Thm. 4.4.13]. Alternatively, one may choose to think of Frege systems as some variant of Natural Deduction or the Sequent Calculus for classical propositional logic.

Particularly important for us is the Extended Frege (EF) system, in which proof lines can be Boolean circuits and not just formulas, which would allow in principle for more succinct proofs. We shall often consider extensions of Extended Frege by sets of additional axioms. For a set $A \subseteq \textsc{Taut}$ of tautologies recognizable in polynomial time, the system $\mathsf{EF} + A$ refers to Extended Frege extended with substitution instances of any formula in A. Note that if A were to contain contingent formulas, then $\mathsf{EF} + A$ would not be sound; in particular, it would not be a Cook-Reckhow system, though it would be polynomially bounded.

A useful property of EF is the fact that $\mathsf{EF} + \mathrm{Ref}_S \geq S$ for every propositional system S [47]. Here Ref_S is the sequence of tautologies encoding the *reflection principle for* S, which states that S is sound. Namely, $\mathrm{Ref}_S \coloneqq \{\mathrm{Ref}_{S,n,m}\}_{n,m\in\mathbb{N}}$ where the formulas $\mathrm{Ref}_{S,n,m} \coloneqq \mathrm{Prf}_{S,n,m}(\pi,\varphi) \to \mathrm{Sat}_{n,m}(\varphi,\alpha)$ encode the soundness of S, and φ is a formula of size n, π is a purported S-proof of size m and α is an assignment to the variables in φ, which are all encoded by free variables. The formula $\mathrm{Prf}_{S,n,m}$ encodes that π is a correct S-proof of φ, and $\mathrm{Sat}_{n,m}(\varphi,\alpha)$ encodes the standard satisfaction relation for propositional formulas. Alternatively, one may exploit the same relation with respect to the *consistency* of S, $\mathrm{Con}_S \coloneqq \{\mathrm{Con}_{S,m}\}_{m\in\mathbb{N}}$, where $\mathrm{Con}_{S,m} \coloneqq \neg\,\mathrm{Prf}_{S,1,m}(\pi,\bot)$ and π encodes a purported proof of size m.

2.1.2 Quantified propositional systems

It is often convenient to operate on systems capable of reasoning with *quantified* Boolean formulas, where the quantification ranges over $\{0,1\}$. We denote by Σ^q_i (respectively, Π^q_i) the class of quantified Boolean formulas with i alternations between existential and universal quantifiers, starting with an existential (respectively, universal) one.

We are particularly interested in Gentzen's Sequent Calculus for quantified propositional logic. The system extends the usual propositional Sequent Calculus by four new rules to handle quantifiers (see [51, Def. 4.1.2] for a formal definition of the rules). We denote this system by G, and by G^* its tree-like counterpart. The system G_i, for $i \in \mathbb{N}$, corresponds to G where the quantified formulas appearing in the sequents can only be in the class $\Sigma^q_i \cup \Pi^q_i$. The tree-like counterpart of G_i is naturally denoted G^*_i. It is useful to know that EF and G^*_1 are polynomially equivalent with respect to Π^q_1 formulas [51, Thm. 4.1.3].

2.1.3 Implicit proof systems

Implicit proof systems constitute a systematic way of obtaining, for every proof system S, a potentially stronger system S', and were introduced by Krajíček [45]. The essential idea is to encode a given proof in the system S as a multi-output Boolean circuit taking

as input a number i in binary and outputting the i-th step of the proof. More formally, given propositional proof systems S and Q, a proof of a tautology φ in the *implicit system* $[S, Q]$ is a pair (π, C) consisting of a proof and a circuit, such that the truth table of C encodes a valid Q-proof of φ (the *implicit* proof), while π is an *explicit* S-proof of the formula $\mathrm{Correct}_Q(\varphi, C)$, which is the formula stating that the truth table of C is a correct Q-proof of φ. If S and Q are Cook-Reckhow proof systems, then so is $[S, Q]$.

For a system S, the implicit system $[S, S]$ is denoted by $\mathsf{i}S$. In particular, we shall work with the *Implicit Extended Frege* proof system, $\mathsf{iEF} \coloneqq [\mathsf{EF}, \mathsf{EF}]$. The system iEF is particularly strong, and it can in fact simulate all of Gentzen's G with respect to propositional tautologies [45, Cor. 2.4].

2.2 Bounded arithmetic

Our proofs exploit the connections between propositional proof complexity and theories of bounded arithmetic. Below we cover the essential preliminaries, which should be accessible to any reader with basic knowledge of first-order logic.

2.2.1 The theories S^1_2 and $\mathsf{S}^1_2 + \text{1-EXP}$

Theories of bounded arithmetic capture various levels of feasible reasoning and act as a uniform counterpart of propositional systems. Intuitively, feasibility is achieved by restricting the complexity of formulas over which one can apply general reasoning schemes like induction.

The central theory for us is Buss's S^1_2, which we think of as corresponding to polynomial-time reasoning. In this context, we work over the first-order language of bounded arithmetic, $\mathcal{L}_{\mathrm{BA}} \coloneqq \{0, S, +, \cdot, <, |x|, \lfloor x/2 \rfloor, x\#y\}$, which extends the language of Peano Arithmetic by the symbols $|x|$, $\lfloor x/2 \rfloor$ and $x\#y$. The standard interpretation of $\lfloor x/2 \rfloor$ is clear. The notation $|x|$ denotes the length of the binary encoding of the number x, $\lceil \log(x+1) \rceil$, while the *smash symbol* $x\#y$ stands for $2^{|x| \cdot |y|}$.

The definition of *bounded formulas* is analogous to the bounded quantification one encounters in the polynomial hierarchy. For a quantifier $Q \in \{\exists, \forall\}$ and a term t in the language of bounded arithmetic, a formula of the form $Qx < t.\varphi(x)$ stands for either $\forall x.(x < t \to \varphi(x))$ or $\exists x.(x < t \wedge \varphi(x))$. These are called *bounded quantifiers*. Whenever the bounded quantifier is of the form $Q < |s|$ for some term s, we talk about *sharply bounded quantifiers*. The hierarchy of *bounded formulas* consists of the classes Σ^b_n and Π^b_n, for $n \geq 1$, which are defined by counting the alternations of bounded quantifiers ignoring the sharply bounded ones, starting with an existential (respectively, universal) one. The class Δ^b_n consists of all formulas that admit an equivalent definition in both Σ^b_n and Π^b_n. In particular, the class Δ^b_0 stands for all formulas with sharply bounded quantifiers only.

The theory S^1_2 of Buss [15] extends Robinson's arithmetic Q by some basic axioms for the new function symbols and the polynomial induction scheme (PInd) for Σ^b_1-formulas: for every $\varphi \in \Sigma^b_1$, the theory contains the axiom

$$\varphi(0) \wedge \forall x(\varphi(\lfloor x/2 \rfloor) \to \varphi(x)) \to \forall x \varphi(x). \tag{PInd}$$

An alternative system intended to capture polynomial-time reasoning is Cook's equational theory PV [21]. In the formalism of PV one has some basic function symbols and introduces new ones recursively by composition and limited recursion on notation, in the style of Cobham's functional definition of **FP** [17]. In this way, the function symbols obtained in PV are precisely those of all polynomial-time functions over the naturals. The first-order version of PV is PV_1 [48, 16, 18]. Without loss of generality, we shall work in the theory $\mathsf{S}^1_2(\mathsf{PV})$,

which is the theory S^1_2 in the language of bounded arithmetic extended by all PV function symbols, meaning that we have a fresh symbol for each function in **FP**, and induction is now available for all $\Sigma^b_1(\mathsf{PV})$ formulas. We abuse notation and refer to this directly as S^1_2.

While S^1_2 is able to formalize a significant amount of complexity theory and some mathematics, it suffers from the drawback of being unable to even state the existence of exponentially large objects. For certain more elaborate arguments we shall work instead inside $\mathsf{S}^1_2 + \mathsf{1\text{-}EXP}$, which patches this issue. We follow here the definition of the theory given by Krajíček [45, Cor. 2.2]: we write $\mathsf{S}^1_2 + \mathsf{1\text{-}EXP} \vdash \forall x \varphi(x)$ for some arithmetic formula φ if there exists a term t such that

$$\mathsf{S}^1_2 \vdash \forall x \forall y (t(x) \leq |y| \to \varphi(x)).$$

The definition is somewhat indirect and may be hard to grasp at first glance. Intuitively, it allows one to derive properties about x under the assumption that $y = 2^x$ exists.

The theory S^1_2 corresponds to polynomial-time computations in the sense that the provably total relations in S^1_2 are precisely the polynomial-time-computable ones. The same relation holds for $\mathsf{S}^1_2 + \mathsf{1\text{-}EXP}$ and the complexity class **EXP**.

2.2.2 Approximate counting

Many of the formalizations carried out in bounded arithmetic require the ability to count. In some cases, small sets can be counted *exactly*, but one often requires more sophisticated machinery for *approximate counting*, needed to formalize many probabilistic arguments.

For $a \in \mathbb{N}$, a *bounded definable set* is a set of naturals $X = \{x < a \mid \varphi(x)\} \subseteq [0, a)$, where $\varphi \in \Sigma^b_\infty$ is some arithmetic formula. For $X \subseteq a$ and $Y \subseteq b$, we define $X \times Y \coloneqq \{bx + y \mid x \in X, y \in Y\} \subseteq ab$ and $X \dot\cup Y \coloneqq X \cup \{y + a \mid y \in Y\} \subseteq a + b$. Rational numbers are assumed to be represented by pairs of integers in the natural way. We also use the unfortunate but standard *Log-notation* widespread in bounded arithmetic, by which $n \in \mathrm{Log}$ stands for the formula $\exists x (n = |x|)$ and $n \in \mathrm{LogLog}$ stands for $\exists x (n = ||x||)$.

Intuitively, from the point of view of the theory, numbers in Log are "small" numbers. For a circuit $C : 2^k \to 2$, where we adopt the set-theoretic custom of identifying $\{0, 1\}$ with the number 2, we can consider the bounded definable set $X_C \coloneqq \{x < 2^k \mid C(x) = 1\}$, and ask about the task of counting the size of X_C.

There exists a PV-function $\mathrm{Count}(C, y) = |X_C \cap |y||$. This means that if $2^k \in \mathrm{Log}$, then one can do *exact counting* of $|X_C|$ efficiently. We use the notation $\Pr_{x<|y|}[C(x) = 1] \leq z/w$ for the PV-relation $w \cdot \mathrm{Count}(C, y) \leq |y| \cdot z$.

If $2^k \notin \mathrm{Log}$, exact counting becomes problematic. To avoid this, Jeřábek [36, 37] systematically developed the theory APC_1 capturing probabilistic polynomial-time reasoning by means of approximate counting. The theory APC_1 is defined as $\mathsf{PV}_1 + \mathrm{dWPHP}(\mathsf{PV})$ where $\mathrm{dWPHP}(\mathsf{PV})$ stands for the *dual (surjective) pigeonhole principle* for all PV-functions. That is, the set of all formulas

$$x > 0 \to \exists v < x(|y| + 1). \forall u < x|y|.\ f(u) \neq v, \tag{dWPHP}$$

where f is a PV-function which might involve other parameters not explicitly shown.

We write $C : X \twoheadrightarrow Y$ if C is a surjective mapping from X to Y. Let $X, Y \subseteq 2^n$ be definable sets, and $\epsilon \leq 1$. The size of X is *approximately less than the size of Y with error ϵ*, written as $X \preceq_\epsilon Y$, if there exists a circuit C, and $v \neq 0$ such that

$$C : v \times (Y \dot\cup \epsilon 2^n) \twoheadrightarrow v \times X.$$

In this context, the notation $X \approx_\epsilon Y$ stands for $X \preceq_\epsilon Y$ and $Y \preceq_\epsilon X$. As with exact counting, the notation $\Pr_{x<y}[C(x) = 1] \circ_\epsilon z/w$ stands for $w \cdot (X_C \cap y) \circ_\epsilon y \cdot z$, for $\circ \in \{\preceq, \approx\}$. Since a number s is identified with the interval $[0, s)$, $X \preceq_\epsilon s$ means that the size of X is at most s with error ϵ.

The definition of $X \preceq_\epsilon Y$ is an unbounded $\exists\Pi^b_2$ formula even if X and Y are defined by circuits, so it cannot be used freely in bounded induction. This problem can be solved by working in $\mathsf{sHARD}^{\mathrm{A}}$, defined as the relativized theory $\mathsf{S}^1_2(\alpha)$ extended with axioms postulating that $\alpha(x)$ is a truth table of a function on $||x||$ variables hard on average for circuits of size $2^{||x||/4}$. In $\mathsf{sHARD}^{\mathrm{A}}$ there is a $\mathsf{PV}(\alpha)$ function Size approximating the size of any set $X \subseteq 2^n$ defined by a circuit C so that $X \approx_\epsilon \mathrm{Size}(\alpha, C, 2^n, 2^{\epsilon^{-1}})$ for $\epsilon^{-1} \in \mathrm{Log}$ (by combination of [37, Lemma 2.14] and [35, Cor. 3.6]).

The following key definition allows us to express that a function is indeed hard on average.

▶ **Definition 3** ($\mathrm{Hard}^{\mathrm{A}}_\epsilon(f)$, in PV_1 [37]). *Let $f : 2^k \to 2$ be a truth table of a Boolean function with k inputs (with f encoded as a string of 2^k bits, and hence with $k \in \mathrm{LogLog}$). We say that f is* average-case ϵ-hard*, written as $\mathrm{Hard}^A_\epsilon(f)$, if for every circuit C of size at most $2^{\epsilon k}$,*

$$|\{u < 2^k \mid C(u) = f(u)\}| < (1/2 + 2^{-\epsilon k})2^k.$$

Note that $\mathrm{Hard}^A_\epsilon(f)$ is Π^b_1-definable in PV_1.

We write $\mathrm{tt}^{\mathrm{avg}}_\epsilon(f_k, 2^{\epsilon k}) := ||\,\mathrm{Hard}^{\mathrm{A}}_\epsilon(f)||_m$ for the propositional translation (see Section 2.2.3) of the formula $\mathrm{Hard}^{\mathrm{A}}_\epsilon(f)$ above and an appropriately chosen parameter m depending on k and ϵ. We also consider the polynomial-time function $\mathrm{CorrectFracTT}^\delta(s, n, C, f)$, that checks whether f is a string of length 2^n, C encodes a circuit of size at most s, and finally verifies whether the fraction of accepted inputs is larger than $(1/2 + 2^{-\delta n})2^n$.

The theory APC_1 is strong enough to show that hard-on-average functions do exist.

▶ **Proposition 4** (Jeřábek [35]). *For every rational constant $\epsilon < 1/3$, there exists a constant c such that APC_1 proves that for every $k \in \mathrm{LogLog}$ such that $k \geq c$, there exist a function $f : 2^k \to 2$ that is average-case ϵ-hard.*

The theory S^1_2 can be relativized to $\mathsf{S}^1_2(\alpha)$. This means, in particular, that the language of $\mathsf{S}^1_2(\alpha)$, denoted also $\mathsf{S}^1_2(\alpha)$, contains symbols for all polynomial-time machines with access to the oracle α.

▶ **Definition 5** ($\mathsf{sHARD}^{\mathrm{A}}$ [35]). *The theory sHARD^A is an extension of the theory $\mathsf{S}^1_2(\alpha)$ by the axioms stating*

1. *the number $\alpha(x)$ encodes the truth table of a Boolean function in $||x||$ variables;*
2. *$x \geq c \to \mathrm{Hard}^A_{1/4}(\alpha(x))$, where c is the constant from the previous proposition;*
3. *$||x|| = ||y|| \to \alpha(x) = \alpha(y)$.*

The key technical tool from the framework of approximate counting is the following theorem by Jeřábek.

▶ **Theorem 6** (Jeřábek [37]). *There is a $\mathsf{PV}(\alpha)$-function* Size *such that sHARD^A proves that if $X \subseteq 2^n$ is definable by a circuit C, then $X \approx_\epsilon \mathrm{Size}(\alpha, C, 2^n, e)$, where $\epsilon = |e|^{-1}$.*

For a circuit $C : 2^n \to 2$, we introduce the notation

$$\Pr_{x<y}[C(x) = 1] \preceq^f_\epsilon \frac{z}{w}$$

to mean $w \cdot \mathrm{Size}(f, C, 2^n, e) \leq y \cdot z$, where $\epsilon = |e|^{-1}$.

2.2.3 Correspondences and propositional translations

While our formalizations are comfortably carried out in the first-order theories presented above, we are able to transfer our results back to propositional logic thanks to the existence of *propositional translations*. Following Krajíček [51], we say that a theory T *corresponds* to a propositional proof system S if (i) T can prove the soundness of S and (ii) every universal consequence $\forall x\varphi(x)$ of T, where φ is quantifier-free, admits polynomial-size proofs in S when grounded into a sequence of propositional formulas. Pudlák alternatively says that S is the *weak system* of the theory T [62]. More formally, for such a universal formula φ, we denote by $||\varphi||_n$ the propositional translation for models of size n. Sometimes we abuse the notation and write $||\varphi||$ dropping the subscript n. We refer the reader to standard texts like those of Krajíček [51] or Cook and Nguyen [20] for formal definitions of the translation.

The key fact for us is that universal theorems of S^1_2 admit short propositional proofs in Extended Frege. More importantly, $\mathsf{S}^1_2 + \text{1-EXP}$ corresponds to Implicit Extended Frege.

▶ **Theorem 7** (Correspondence of $\mathsf{S}^1_2 + \text{1-EXP}$ and iEF [45, Thm. 2.1])**.** *The proof system* iEF *corresponds to* $\mathsf{S}^1_2 + \text{1-EXP}$*. That is,*

- **(i)** *the theory* $\mathsf{S}^1_2 + \text{1-EXP}$ *proves the soundness of* iEF*;*
- **(ii)** *whenever a* $\forall\Pi^b_1$*-sentence* $\forall x\varphi(x)$ *is provable in* $\mathsf{S}^1_2 + \text{1-EXP}$*, there are polynomial-size* iEF*-proofs of the sequence of tautologies* $\{||\varphi||_n\}_{n\in\mathbb{N}}$*;*
- **(iii)** *if* $\mathsf{S}^1_2 + \text{1-EXP}$ *proves the soundness of some propositional system* S*, then* $\text{iEF} \geq S$*.*

The translation also works for formulas beyond $\forall\Pi^b_1$ as long as we translate into a quantified propositional system. The definition of the translation is straightforward, and we note that Σ^b_1-consequences of S^1_2 translated as Σ^q_1 formulas admit polynomial-size proofs in G^*_1.

▶ **Theorem 8** (Correspondence of S^1_2 and G^*_1 [47])**.** *Whenever a* $\forall\Sigma^b_1$*-sentence* $\forall x\exists y \leq t.\varphi(x,y)$ *is provable in* S^1_2*, there are polynomial-size proofs of the sequence of* Σ^q_1*-formulas obtained by the translation,* $\{||\exists x\varphi(x,y)||_n\}_{n\in\mathbb{N}}$*, in* G^*_1*.*

2.3 Interactive proof systems and the sum-sheck protocol

While our focus is on propositional proof systems in the sense of Cook and Reckhow, our work exploits relations to more lax notions of provability. Following Babai [7], an *Merlin-Arthur proof system* or *Merlin-Arthur protocol* for a language $L \subseteq \{0,1\}^*$ is a polynomial-time function S together with some constant c such that the two following properties are satisfied for every $x \in \{0,1\}^*$. Namely,

1. if $x \in L$, then there exists some $\pi \in \{0,1\}^*$ such that $\Pr_{r\in\{0,1\}^{(|x|+|\pi|)^c}}[S(x,\pi,r) = 1] = 1$;
2. if $x \notin L$, then for every $\pi \in \{0,1\}^*$, $\Pr_{r\in\{0,1\}^{(|x|+|\pi|)^c}}[S(x,\pi,r) = 1] < 1/3$.

The first condition formalizes *completeness*, while the second corresponds to *soundness*. The complexity class **MA** contains all languages that admit a polynomially-bounded Merlin-Arthur protocol, meaning that there exists a constant d such that the completeness guarantee is strengthened to proofs $\pi \in \{0,1\}^{|x|^d}$. One should think of MA proof systems as Cook-Reckhow systems where the verifier is randomized and may thus accept some incorrect proofs with small probability.

We recall that, under the standard derandomization assumption that there exists a Boolean function family in **E** that is wort-case hard for subexponential-size circuits, every Merlin-Arthur system derandomizes into a Cook-Reckhow system and, in particular, **MA** = **NP** [56, 34].

Our proofs rely on a particular interactive protocol, the *Sum-Check Protocol* of Lund, Fortnow, Karloff, and Nisan [54] for the language of unsatisfiable 3CNFs. Unlike Merlin-Arthur protocols, this is an interactive protocol running for multiple rounds between a Prover and a Verifier, before the Verifier makes a decision. We now recall the details of the protocol.

The Sum-Check Protocol [54]

The protocol considers a 3CNF $\varphi(x_1, \dots, x_n)$ over m clauses, known to both the Verifier and the Prover.

1. The Prover generates a prime number[6] $p \in (2^{2n^3+n}, 2^{(2n^3+n)^{c_p}}]$ together with a Pratt certificate[7] on the primality of p and sends them to the Verifier, who checks for correctness of the certificate, and aborts if incorrect.
2. The Prover and the Verifier arithmetize φ into a polynomial $P_\varphi(x_1, \dots, x_n)$ of degree at most $3m$ over $\mathbb{F}_p$ in the usual way: a clause like $(x \vee \neg y \vee z)$ is turned into $1-(1-x)y(1-z)$, and one then takes the product of all such arithmetized clauses. In this way, for all $x \in \{0,1\}^n$, $\varphi(x) = 1$ if and only if $P_\varphi(x) = 1$.
3. The Verifier sets $(a_1, \dots, a_n) := (0, \dots, 0)$, $Q_0(a_0) := 0$ and for $i \in \{1, \dots, n\}$, the following interaction is carried out:
   a. Leaving x_i free, the Prover computes the coefficients of the following univariate polynomial over $\mathbb{F}_p$, $Q_i(x_i) := \sum_{x_{i+1} \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} P_\varphi(a_1, \dots, a_{i-1}, x_i, x_{i+1}, \dots, x_n)$ and sends the $O(m)$ coefficients of Q_i to the Verifier.
   b. The Verifier checks whether $Q_i(0) + Q_i(1) = Q_{i-1}(a_{i-1})$. If the check fails, the Verifier rejects. Otherwise, it samples a random $a_i \in \mathbb{F}_p$ and sends it to the Prover.
   c. In the final round, instead of sending a_n to the Prover, the Verifier checks whether $P_\varphi(a_1, \dots, a_n) = Q_n(a_n)$ and accepts or rejects based on this.

3 Main result

Our proof exploits the known fact that if $\#\mathbf{P} \subseteq \mathbf{FP}/\text{poly}$, then $\mathbf{coNP} \subseteq \mathbf{MA}$. Indeed, if $\#\mathbf{P}$ has small circuits one can provide polynomial-size circuits that simulate the Prover's movements in the Sum-Check protocol for Unsat, since one can consider the MA proof system in which Arthur receives from Merlin a circuit claiming to be the circuit that the Prover used to carry out their strategy, and with the aid of randomness, Arthur can execute this on his own and decide based on the outcome of this simulation.

Let us make this formal.

▶ **Definition 9** (The SC proof system). *Let $V(p, u, \varphi, C, r)$ be the polynomial-time function carrying out the simulation of the Sum-Check protocol. Namely, p is intended to be a prime in $(2^{2n^3+n}, 2^{(2n^3+n)^{c_p}}]$, u a Pratt certificate for p, φ a 3CNF over n variables, r a string of random bits, and C a multi-output circuit providing the Prover's responses in the interactions with the Verifier in the Sum-Check protocol.*

The Sum-Check Proof System*, denoted by* SC*, is a Merlin-Arthur proof system for proving 3DNF tautologies. An* SC *proof of φ is a tuple $\langle p, u, C \rangle$ such that p is indeed a prime in the interval above, correctly certified by the Pratt certificate u, and such that* $\Pr_{r \in \mathbb{F}_p^n}\left[V(p, u, \neg\varphi, C, r) = 1\right] = 1$.

[6] The constant c_p in the exponent comes from the formalization of the soundness of the sum-check protocol inside $\mathsf{S}_2^1 + \text{1-EXP}$ in a recent work of Khaniki [40]; while we do not need such details in our proofs, we leave it here to be faithful to the formalization.

[7] A *Pratt certificate* is a succinct witness for primality checkable in polynomial time [60]. The details are not relevant for our results, but it is important that the Verifier can be convinced of p being a prime.

The following is just a rephrasing of the fact that $\#\mathbf{P} \subseteq \mathbf{FP}/\mathrm{poly}$ implies $\mathbf{coNP} \subseteq \mathbf{MA}$, in terms of the Merlin-Arthur system SC. We omit the proof, which can be be found in standard texts (see e.g. [5, Thm. 8.22]).

▶ **Lemma 10.** *If $\#\mathbf{P} \subseteq \mathbf{FP}/\mathrm{poly}$, then SC is polynomially bounded over 3DNF tautologies.*

Our goal is to extend the previous lemma from SC to the concrete and natural Cook-Reckhow system Implicit Extended Frege. The idea again is that iEF (or rather its first-order counterpart, $\mathsf{S}^1_2 + \text{1-EXP}$) can prove the soundness of this system and thus simulate it. We shall then derandomize the SC protocol inside iEF. Fortunately for us, the soundness of the Sum-Check protocol was recently proven by Khaniki in the right theory of bounded arithmetic.

▶ **Theorem 11** (Soundness of the sum-check protocol [39, Thm. 15.3])**.** *There are constants $c, k \in \mathbb{N}$ such that S^1_2 proves the following sentence: for every n, φ, a, p, u, C, if it holds that (i) φ is a 3CNF in n variables where $n \geq c$, and (ii) $\varphi(a) = 1$ and, (iii) $2^{2n^3+n} < p \leq 2^{(2n^3+n)^{c_p}}$ and, (iv) $n^k \in \mathrm{Log}\,\mathrm{Log}$, then*

$$\Pr_{r \in \mathbb{F}_p^n}\left[V(p, u, \varphi, C, r) = 1\right] \leq \frac{n\binom{2n}{3}}{p}.$$

We can now formalize the soundness of the SC proof system from Definition 9. The arguments that follow are a concrete application of more sophisticated techniques employed by Khaniki [40, 39], who has studied interactive protocols in the context of defining new jump operators in proof complexity.

▶ **Definition 12** (The $\mathrm{Sound}_c(\mathsf{SC})$ formula)**.** *We denote by $\mathrm{Sound}_c(\mathsf{SC})$ the following $\forall\Sigma^b_1$ sentence, claiming the soundness of SC: for all φ, a, p, u, C, f, where $|\varphi| > c$, there is a circuit D of size $\leq \lceil |f|^{1/4} \rceil$ such that if*

$$\neg\left(\Pr_{r \in \mathbb{F}_p^n}[V(p, u, \neg\varphi, C, r) = 1] \preceq^f_\epsilon \frac{3}{8}\right)$$

holds, then at least one of the following conditions holds:

- **(i)** $|f| \neq |C|^{k_a} + k'_a$ *or,*
- **(ii)** $\mathrm{CorrectFracTT}^{1/4}(\lceil |f|^{1/4} \rceil, ||f||, D, f) = 1$ *or,*
- **(iii)** $p \notin (2^{2n^3+n}, 2^{(2n^3+n)^{c_p}}]$ *or,*
- **(iv)** $\varphi(a) = 1$*,*

where k_a, k'_a are the constants from Theorem 6 making sure that Size *function works properly, $\epsilon = 1/16$ and n is the number of variables of φ. In the definition of the displayed probability, we assume that $y = p^n$ and that the circuit defining the set of strings accepted by V rejects all $r \geq p^n$.*

Note that even if V accepts with probability 1 on a given input, the approximating probability from Definition 12 can be significantly smaller because of the difference between p^n and the input-size of the circuit in the input of the Size function. Another relevant point is that for each $C, 2^n, e$, the function $\mathrm{Size}(\alpha, C, 2^n, e)$ calls α only once. In fact, it calls $\alpha(x)$ on an input x which depends only on $|C|, n, |e|$. This is needed for the formula $\mathrm{Sound}_c(\mathsf{SC})$ to be well-defined.

It now suffices to verify that the encoding of the soundness of SC is indeed provable in $\mathsf{S}^1_2 + \text{1-EXP}$.

▶ **Proposition 13** (Soundness of SC inside $\mathsf{S}^1_2 + \text{1-EXP}$). *There is a constant $c \in \mathbb{N}$ such that* $\mathsf{S}^1_2 + \text{1-EXP} \vdash \text{Sound}_c(\mathsf{SC})$.

Proof. Let $c \in \mathbb{N}$ be a big enough constant that can be computed from the rest of the argument and

$$\text{Sound}_c(\mathsf{SC}) := \forall \varphi, a, p, u, C, f \exists D \Phi(\varphi, a, p, u, C, f, D)$$

the soundness formula in Definition 12 above. Let φ be a 3DNF in n variables such that $|\varphi| > c$, and consider a, p, u, C, f. Then the following cases can happen:

(a) If $|f| \neq |C|^{k_a} + k'_a$ or $p \notin (2^{2n^3+n}, 2^{(2n^3+n)^{c_p}}]$, then $\Phi(\varphi, a, p, u, C, f, 0)$ is trivially true.

(b) If there is a circuit D of size $\leq \lceil |f|^{1/4} \rceil$ such that $\text{CorrectFracTT}^{1/4}(\lceil |f|^{1/4} \rceil, ||f||, D, f) = 1$, then $\Phi(\varphi, a, p, u, C, f, D)$ is trivially true.

(c) If the previous cases do not happen and moreover

$$\neg \left(\Pr_{r \in \mathbb{F}_p^n} [V(p, u, \neg\varphi, C, r) = 1] \preceq^f_\epsilon \frac{3}{8} \right)$$

holds, then we have that $8 \cdot \text{Size}(f, C^*, 2^m, e) > 3p^n$, where m is the smallest integer such that $2^m \geq p^n$, $\epsilon := |e|^{-1}$ and $C^*(r) := V(p, u, \neg\varphi, C, r)$. By the assumption $\text{Hard}^{\text{A}}_{1/4}(f)$ holds and by the fact that we are over S^1_2 and we can use f as a parameter in polynomial induction for Σ^b_1 formulas, we can do approximate counting using Theorem 6. Hence there is a circuit G and some $v \leq \text{poly}(m\epsilon^{-1}|C^*|)$ such that

$$G : v \times (X_{C^*} \dot\cup\ \epsilon 2^m) \twoheadrightarrow v \times \text{Size}(f, C^*, 2^m, e).$$

As we work in $\mathsf{S}^1_2 + \text{1-EXP}$ and G is surjective, we can find a subset $A \subseteq v \times (X_{C^*} \dot\cup\ \epsilon 2^m)$ such that G restricted to A is a one-to-one function from A to $v \times \text{Size}(f, C^*, 2^m, e)$. Now we can apply exact counting (as we have 1-EXP) and show that

$$\text{Size}(f, C^*, 2^m, e) \leq |X_{C^*}| + \epsilon 2^m.$$

By the fact that $8 \cdot \text{Size}(f, C^*, 2^m, e) > 3p^n > 3 \cdot 2^m/2$, we have $2^m/8 < |X_{C^*}|$. Now if $\varphi(a) = 0$, by Theorem 11 we get

$$\Pr_{r \in \mathbb{F}_p^n} \left[V(p, u, \neg\varphi, \pi, r) = 1 \right] \leq \frac{n\binom{2n}{3}}{p}.$$

Note that $|\varphi| > c$ which implies that n is big enough and as $p > 2^{2n^3+n}$ we get that $n\binom{2n}{3}/p \leq 1/8$, which implies

$$\Pr_{r \in \mathbb{F}_p^n} \left[V(p, u, \neg\varphi, \pi, r) = 1 \right] \leq \frac{1}{8}.$$

As C^* rejects all $r \geq p^n$, this implies that $|X_{C^*}| \leq 2^m/8$ which leads to a contradiction, so $\varphi(a) = 1$. ◀

The main technical issue now is that $\text{Sound}_c(\mathsf{SC})$ is a $\forall\Sigma^b_1$ sentence that does not translate into a propositional formula that iEF can reason about. Instead, we shall work on a quantified propositional system. For this to make sense we need to know the quantified propositional proof system associated with $\mathsf{S}^1_2 + \text{1-EXP}$.

We invoke the following known **TFNP** characterization of the Σ^b_1 consequences of $\mathsf{S}^1_2 + \text{1-EXP}$, which identifies a "complete" Σ^b_1 sentence Ψ such that any other Σ^b_1 consequence of $\mathsf{S}^1_2 + \text{1-EXP}$ reduces to it in G^*_1.

▶ **Theorem 14** ([42, 41, 46, 9]). *There is a $\forall\Sigma^b_1$ sentence $\Psi := \forall x \exists y \psi(x,y)$ (the bound on y is implicit in ψ) such that the following statements are true:*

(i) $\mathsf{S}^1_2 + \text{1-EXP} \vdash \forall x \exists y \psi(x,y)$*;*

(ii) *for any $\forall\Sigma^b_1$ sentence $\forall x \exists y \alpha(x,y)$ such that $\mathsf{S}^1_2 + \text{1-EXP} \vdash \forall x \exists y \alpha(x,y)$, there are* PV *functions f and g such that $\mathsf{S}^1_2 \vdash \forall x, y(\psi(f(x), y) \to \alpha(x, g(x, y)))$.*

In what follows, we shall work with Gentzen's system G extended with the propositional translation of the sentence Ψ in the theorem above. We denote this system by $\mathsf{G}_{\mathsf{EXP}} := \mathsf{G}^*_1 + ||\Psi||$ and use the following key properties about it. The full version of the paper contains explicit proofs of each of these items, but we shall omit these here.

▶ **Corollary 15.** *The following statements about $\mathsf{G}_{\mathsf{EXP}}$ hold:*

(i) $\mathsf{S}^1_2 + \text{1-EXP} \vdash \Sigma^q_1\text{-Ref}(\mathsf{G}_{\mathsf{EXP}})$*, i.e. the reflection principle for $\mathsf{G}_{\mathsf{EXP}}$ and Σ^q_1 formulas is provable in $\mathsf{S}^1_2 + \text{1-EXP}$;*

(ii) *for every $\forall\Sigma^b_1$-sentence $\forall x \exists y \alpha(x,y)$, if $\mathsf{S}^1_2 + \text{1-EXP} \vdash \forall x \exists y \alpha(x,y)$, then there are polynomial-size $\mathsf{G}_{\mathsf{EXP}}$-proofs of the sequence of Σ^q_1-tautologies $\{||\exists y \alpha(y)||_n\}_{n\in\mathbb{N}}$;*

(iii) *if $\mathsf{S}^1_2 + \text{1-EXP}$ proves the soundness of a propositional proof system S, then $\mathsf{G}_{\mathsf{EXP}} \geq S$.*

Let us observe that $\mathsf{G}_{\mathsf{EXP}}$ is in fact equivalent to iEF.

▶ **Lemma 16.** *The proof systems $\mathsf{iEF}, \mathsf{EF} + \text{Ref}_{\mathsf{iEF}}$ and $\mathsf{G}_{\mathsf{EXP}}$ are polynomially equivalent over propositional tautologies.*

Proof. By item (iii) of Corollary 15 and item (iii) Theorem 7, iEF and $\mathsf{G}_{\mathsf{EXP}}$ polynomially simulate each other. As mentioned in Section 2.1.1, $\mathsf{EF} + \text{Ref}_{\mathsf{iEF}} \geq \mathsf{iEF}$. It is also easy to see that $\mathsf{S}^1_2 + \text{1-EXP}$ proves the soundness of $\mathsf{EF} + \text{Ref}_{\mathsf{iEF}}$, which by item (iii) of Theorem 7 gives us $\mathsf{iEF} \geq \mathsf{EF} + \text{Ref}_{\mathsf{iEF}}$. ◀

We are now ready to define the extension of iEF for which our main theorem holds. Recall that the propositional formulas $\text{tt}^{\text{avg}}_{1/4}(h_n, 2^{n/4})$ were defined in Section 2.2.2 and state the average-case hardness of a Boolean function h_n represented as a truth table.

▶ **Definition 17** (The systems iEF^{tt}). *Let $h = \{h_n\}_{n\in\mathbb{N}}$ be some family of Boolean functions, and let $n_0 \in \mathbb{N}$. We denote by $\mathsf{iEF}^{\text{tt}(h,n_0)} := \mathsf{G}_{\mathsf{EXP}} + \{\text{tt}^{avg}_{1/4}(h_n, 2^{n/4})\}_{n\geq n_0}$ the system that extends $\mathsf{G}_{\mathsf{EXP}}$ by the axioms claiming the hardness of h_n, for $n \geq n_0$.*

Note that $\mathsf{iEF}^{\text{tt}(h,n_0)}$ is a family of proof systems, parameterized by a Boolean function family h and some threshold parameter n_0. Observe that depending on the choice of h and n_0, the system $\mathsf{iEF}^{\text{tt}(h,n_0)}$ may not be a Cook-Reckhow system: if h is not a hard function, or n_0 is not large enough, we will be adding axioms which are not tautologies; and even if h is hard and n_0 is large enough, the system may require advice to verify the proofs. As we shall see, however, these degenerate instantiations of $\mathsf{iEF}^{\text{tt}(h,n_0)}$ are not a problem.

What is more important, the systems $\mathsf{iEF}^{\text{tt}(h,n_0)}$, regardless of their consistency, always simulate SC.

▶ **Lemma 18.** *Let h be family of Boolean functions and let $n_0 \in \mathbb{N}$. The system $\mathsf{iEF}^{\text{tt}(h,n_0)}$ polynomially simulates* SC *over 3DNF tautologies.*

Proof. If the system $\mathsf{iEF}^{\text{tt}(h,n_0)}$ is unsound because the added axioms are not tautologies, then the system is polynomially bounded and simulates every other proof system. So suppose the added axioms are indeed tautologies, meaning that the function h is indeed hard on average.

Let φ_1 be a 3DNF in n_1 variables and $\langle p_1, u_1, C_1\rangle$ be a SC-proof of φ_1. This means

$$2^{2n_1^3+n_1} < p_1 \leq 2^{(2n_1^3+n_1)^{c_p}} \wedge \Pr_{r\in\mathbb{F}_{p_1}^{n_1}}\left[V(p_1, u_1, \neg\varphi_1, C_1, r) = 1\right] = 1.$$

Note that by Theorem 11 and Corollary 15, there are PV functions l, g such that

$$\mathsf{S}_2^1 \vdash \forall\varphi, a, p, u, C, f\,(\psi(l(\varphi, a, p, u, C, f), y) \to \Phi(\varphi, a, p, u, C, f, g(\varphi, a, p, u, C, f, y)))\,,$$

where $\mathrm{Sound}_c(\mathsf{SC}) \coloneqq \forall\varphi, a, p, u, C, f\exists D\Phi(\varphi, a, p, u, C, f, D)$. Let $s \coloneqq |\,\langle p, u, C\rangle\,|$. Then by Theorem 8 there is a $s^{O(1)}$-size G_1^*-proof of

$$||\forall\varphi, a, p, u, C, f\,(\psi(l(\varphi, a, p, u, C, f), y) \to \Phi(\varphi, a, p, u, C, f, g(\varphi, a, p, u, C, f, y)))\,||_{s'},$$

where $s' \coloneqq \mathrm{poly}(s)$. Let us rewrite the previous quantified propositional formula as $||\Psi'|| \to ||\Phi'||$ with the right range of parameters such that p_1, u_1, φ_1, C_1 are substituted in the formula in their corresponding places. Now we take the substitution instance $\mathrm{tt}_{1/4}^{\mathrm{avg}}(h_{n'}, 2^{n'/4})$ where $|h_{n'}| \coloneqq |C_1|^{k_a} + k'_a$ and we substitute $h_{n'}$ to the variables corresponding to f and therefore the disjunct which corresponds to CorrectFracTT disappears from $||\Phi'||$ when we apply the rules of G_1^*. Moreover, it is not hard to verify that after the substitutions every other disjunct which corresponds to subformulas of $\mathrm{Sound}_c(\mathsf{SC})$ from Definition 12 disappears except φ_1. So what we have is G_1^*-proof of $||\Psi''||(\bar{x}, \bar{y}) \to \varphi_1(\bar{x})$ ($\bar{x}$ and $\bar{y}$ are disjoint variables) where $||\Psi''||$ is a substitution instance of $||\Psi'||$. Since we are working in $\mathsf{G}_{\mathsf{EXP}}$, we have the substitution instance $\exists\bar{y}||\Psi''||(\bar{x}, \bar{y})$ and therefore using the rules of G_1^* we get a short $\mathsf{G}_{\mathsf{EXP}}$-proof of $\varphi_1(\bar{x})$. ◀

Our main theorem now easily follows.

▶ **Theorem 19** (Main theorem). *Let h be a family of Boolean functions and let $n_0 \in \mathbb{N}$. If the system $\mathsf{iEF}^{\mathrm{tt}(h,n_0)}$ is not polynomially bounded, then $\#\mathbf{P} \not\subseteq \mathbf{FP}/\mathrm{poly}$.*

Proof. By Lemma 18 above, for every choice of h and n_0, the system $\mathsf{iEF}^{\mathrm{tt}(h,n_0)}$ polynomially simulates SC, so if $\mathsf{iEF}^{\mathrm{tt}(h,n_0)}$ is not polynomially bounded, then SC is not either. Then, by the contrapositive of Lemma 10, $\#\mathbf{P} \not\subseteq \mathbf{FP}/\mathrm{poly}$. ◀

As discussed, depending on the choice of h and n_0, the system $\mathsf{iEF}^{\mathrm{tt}(h,n_0)}$ may not be sound and thus possibly not Cook-Reckhow. However, for any fixed choice of a uniform candidate hard function, the system is concrete and exhibits the desired connection that proof complexity lower bounds for it imply strong circuit lower bounds. In particular, if there exist functions in $\mathbf{NE} \cap \mathbf{coNE}$ average-case hard for subexponential-size circuits, then we recover the version of the theorem presented in the introduction (Theorem 1).

We note that there is the possibility that iEF, given its strength, already proves such strong circuit lower bounds for some Boolean function. It is thus worth to mention the following corollary.

▶ **Corollary 20.** *Suppose there exists a sequence of Boolean functions $\{h_n\}_{n\in\mathbb{N}}$ for which iEF has polynomial-size proofs of the formula family $\{\mathrm{tt}_{1/4}^{\mathrm{avg}}(h_n, 2^{n/4})\}_{n\geq n_0}$ for some sufficiently large $n_0 \in \mathbb{N}$. If iEF is not polynomially bounded, then $\#\mathbf{P} \not\subseteq \mathbf{FP}/\mathrm{poly}$.*

Proof. If there is such a function h and threshold n_0, then $\mathsf{iEF}^{\mathrm{tt}(h,n_0)}$ is polynomially equivalent to iEF itself, so by Theorem 19 the corollary follows. ◀

References

1 Miklós Ajtai. Σ_1^1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24(1):1–48, 1983.

2 Miklós Ajtai. The complexity of the pigeonhole principle. *Combinatorica*, 14:417–433, 1994.

3 Noga Alon and Ravi B Boppana. The monotone circuit complexity of Boolean functions. *Combinatorica*, 7:1–22, 1987.

4 Aleksandr Egorovich Andreev. A method for obtaining lower bounds on the complexity of individual monotone functions. In *Doklady Akademii Nauk*, volume 282(5), pages 1033–1037. Russian Academy of Sciences, 1985.

5 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

6 Noel Arteche, Gaia Carenini, and Matthew Gray. Quantum automating $\mathbf{TC}^0$-Frege is LWE-hard, 2024. `arXiv:2402.10351`.

7 László Babai. Trading group theory for randomness. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 421–429, 1985.

8 Paul Beame, Russell Impagliazzo, Jan Krajíček, Toniann Pitassi, Pavel Pudlák, and Alan Woods. Exponential lower bounds for the pigeonhole principle. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, pages 200–220, 1992.

9 Arnold Beckmann and Sam Buss. The **NP** search problems of Frege and Extended Frege proofs. *ACM Transactions on Computational Logic (TOCL)*, 18(2):1–19, 2017.

10 Stephen Bellantoni, Toniann Pitassi, and Alasdair Urquhart. Approximation and small-depth Frege proofs. *SIAM Journal on Computing*, 21(6):1161–1179, 1992.

11 Olaf Beyersdorff, Ilario Bonacina, Leroy Chew, and Jan Pich. Frege systems for quantified boolean logic. *Journal of the ACM (JACM)*, 67(2):1–36, 2020.

12 Maria Luisa Bonet, Carlos Domingo, Ricard Gavalda, Alexis Maciel, and Toniann Pitassi. Non-automatizability of bounded-depth Frege proofs. *computational complexity*, 13:47–68, 2004.

13 Maria Luisa Bonet, Toniann Pitassi, and Ran Raz. On interpolation and automatization for Frege systems. *SIAM Journal on Computing*, 29(6):1939–1967, 2000.

14 Peter Bürgisser. Completeness and reduction in algebraic complexity theory. *Algorithms and Computation in Mathematics*, 2000.

15 Samuel R Buss. *Bounded arithmetic*. Princeton University, 1985.

16 Samuel R Buss. Relating the bounded arithmetic and polynomial time hierarchies. *Annals of Pure and Applied Logic*, 75(1-2):67–77, 1995.

17 A Cobham. The intrinsic computational difficulty of functions. In *Proc. 1964 Congress for Logic, Methodology, and the Philosophy of Science*, pages 24–30. North-Holland, 1964.

18 Stephen Cook. Relating the provable collapse of **P** to $\mathbf{NC}^1$ and the power of logical theories. In *Proof Complexity and Feasible Arithmetics*, pages 73–91, 1996.

19 Stephen Cook and Jan Krajíček. Consequences of the provability of $\mathbf{NP} \subseteq \mathbf{P}/\mathrm{poly}$. *The Journal of Symbolic Logic*, 72(4):1353–1371, 2007.

20 Stephen Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. Cambridge University Press, 2010.

21 Stephen A. Cook. Feasibly constructive proofs and the propositional calculus. In *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*, pages 83–97, 1975.

22 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Logic, Automata, and Computational Complexity*, 1979.

23 Ben Davis and Robert Robere. Colourful TFNP and Propositional Proofs. In *38th Computational Complexity Conference (CCC 2023)*, volume 264 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:21, 2023. `doi:10.4230/LIPIcs.CCC.2023.36`.

24 Susanna De Rezende, Or Meir, Jakob Nordström, Toniann Pitassi, Robert Robere, and Marc Vinyals. Lifting with simple gadgets and applications to circuit and proof complexity. In *61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 24–30, 2020.

25 Susanna F de Rezende, Mika Göös, and Robert Robere. Proofs, circuits, and communication. *ACM SIGACT News*, 53(1):59–82, 2022.

26 Merrick Furst, James B Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

27 Ankit Garg, Mika Göös, Pritish Kamath, and Dmitry Sokolov. Monotone circuit lower bounds from resolution. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 902–911, 2018.

28 Azza Gaysin. Proof complexity of CSP. *arXiv preprint*, 2022. `arXiv:2201.00913`.

29 Azza Gaysin. Proof complexity of universal algebra in a CSP dichotomy proof. *arXiv preprint*, 2024. `arXiv:2403.06704`.

30 Joshua A Grochow. Polynomial identity testing and the Ideal proof system: PIT is in **NP** if and only if IPS can be p-simulated by a Cook-Reckhow proof system. *arXiv preprint*, 2023. `arXiv:2306.02184`.

31 Joshua A Grochow and Toniann Pitassi. Circuit complexity, proof complexity, and polynomial identity testing: The ideal proof system. *Journal of the ACM (JACM)*, 65(6):1–59, 2018.

32 Tuomas Hakoniemi. Feasible interpolation for Polynomial Calculus and Sums-of-Squares. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, 2020.

33 John Håstad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 6–20, 1986.

34 Russell Impagliazzo and Avi Wigderson. **P** = **BPP** if **E** requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of computing*, pages 220–229, 1997.

35 Emil Jeřábek. Dual weak pigeonhole principle, Boolean complexity, and derandomization. *Annals of Pure and Applied Logic*, 129(1-3):1–37, 2004.

36 Emil Jeřábek. *Weak pigeonhole principle, and randomized computation*. PhD thesis, Faculty of Mathematics and Physics, Charles University, Prague, 2005.

37 Emil Jeřábek. Approximate counting in bounded arithmetic. *The Journal of Symbolic Logic*, 72(3):959–993, 2007.

38 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1/2):1–46, 2004.

39 Erfan Khaniki. *(Im)possibilty results in Proof Complexity and Arithmetic*. PhD thesis, Faculty of Mathematics and Physics, Charles University, Prague, 2023. URL: `https://dspace.cuni.cz/handle/20.500.11956/187614`.

40 Erfan Khaniki. Jump operators, interactive proofs, and proof complexity generators, 2023. Unpublished preprint.

41 Leszek Aleksander Kołodziejczyk, Phuong Nguyen, and Neil Thapen. The provably total **NP** search problems of weak second order bounded arithmetic. *Annals of Pure and Applied Logic*, 162(6):419–446, 2011.

42 Jan Krajíček. Exponentiation and second-order bounded arithmetic. *Annals of Pure and Applied Logic*, 48(3):261–276, 1990.

43 Jan Krajíček. Lower bounds to the size of constant-depth propositional proofs. *The Journal of Symbolic Logic*, 59(1):73–86, 1994.

44 Jan Krajíček. Diagonalization in proof complexity. *Fundamenta Mathematicae*, 182:181–192, 2004.

45 Jan Krajíček. Implicit proofs. *The Journal of Symbolic Logic*, 69(2):387–397, 2004.

46 Jan Krajíček. Consistency of circuit evaluation, Extended Resolution and total **NP** search problems. In *Forum of Mathematics, Sigma*, volume 4, page e15. Cambridge University Press, 2016.

47 Jan Krajíček and Pavel Pudlák. Quantified propositional calculi and fragments of bounded arithmetic. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 36(1):29–46, 1990.

48 Jan Krajíček, Pavel Pudlák, and Gaisi Takeuti. Bounded arithmetic and the polynomial hierarchy. *Annals of Pure and Applied Logic*, 52(1-2), 1991.

49 Jan Krajíček, Pavel Pudlák, and Alan Woods. An exponential lower bound to the size of bounded depth Frege proofs of the pigeonhole principle. *Random Structures & Algorithms*, 7(1):15–39, 1995.

50 Jan Krajíček. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *The Journal of Symbolic Logic*, 62(2):457–486, 1997.

51 Jan Krajíček. *Proof Complexity*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2019. doi:10.1017/9781108242066.

52 Jan Krajíček and Pavel Pudlák. Some consequences of cryptographical conjectures for S_2^1 and EF . *Information and Computation*, 140(1):82–94, 1998.

53 Shachar Lovett, Raghu Meka, Ian Mertz, Toniann Pitassi, and Jiapeng Zhang. Lifting with sunflowers. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, 2022.

54 Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)*, 39(4):859–868, 1992.

55 Moritz Müller and Ján Pich. Feasibly constructive proofs of succinct weak circuit lower bounds. *Annals of Pure and Applied Logic*, 171(2):102735, 2020.

56 Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

57 Ján Pich. Logical strength of complexity theory and a formalization of the PCP theorem in bounded arithmetic. *Logical Methods in Computer Science*, 11, 2015.

58 Jan Pich and Rahul Santhanam. Towards $\mathbf{P} \neq \mathbf{NP}$ from Extended Frege lower bounds. *arXiv preprint*, 2023. arXiv:2312.08163.

59 Toniann Pitassi, Paul Beame, and Russell Impagliazzo. Exponential lower bounds for the pigeonhole principle. *Computational complexity*, 3:97–140, 1993.

60 Vaughan R Pratt. Every prime has a succinct certificate. *SIAM Journal on Computing*, 4(3):214–220, 1975.

61 Pavel Pudlák. Lower bounds for resolution and cutting planes proofs and monotone computations. *The Journal of Symbolic Logic*, 62(3):981–998, 1997.

62 Pavel Pudlák. Reflection principles, propositional proof systems, and theories, 2020. arXiv:2007.14835.

63 Ran Raz and Pierre McKenzie. Separation of the monotone **NC** hierarchy. In *Proceedings 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 234–243. IEEE, 1997.

64 Alexander Razborov. Lower bounds on the monotone complexity of some Boolean function. In *Soviet Math. Dokl.*, volume 31, pages 354–357, 1985.

65 Alexander Razborov. Unprovability of lower bounds on circuit size in certain fragments of bounded arithmetic. *Izvestiya: mathematics*, 59(1):205, 1995.

66 Alexander A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

67 Alexander A Razborov. Bounded arithmetic and lower bounds in boolean complexity. In *Feasible Mathematics II*, pages 344–386. Springer, 1995.

68 Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the Nineteenth Annual ACM Symposium on the Theory of Computing*, pages 77–82, 1987.

Learning Low-Degree Quantum Objects

Srinivasan Arunachalam
IBM Quantum, Thomas J Watson Research Center, Yorktown Heights, NY, USA

Arkopal Dutt
IBM Quantum, IBM Research Cambridge, MA, USA

Francisco Escudero Gutiérrez
CWI & QuSoft, Amsterdam, The Netherlands

Carlos Palazuelos
Dpto. Análisis Matemático y Matemática Aplicada, Fac. Ciencias Matemáticas,
Universidad Complutense de Madrid, Spain
Instituto de Ciencias Matemáticas, Madrid, Spain

Abstract

We consider the problem of learning low-degree quantum objects up to ε-error in ℓ_2-distance. We show the following results: (i) unknown n-qubit degree-d (in the Pauli basis) quantum channels and unitaries can be learned using $O(1/\varepsilon^d)$ queries (which is independent of n), (ii) polynomials $p : \{-1,1\}^n \to [-1,1]$ arising from d-query quantum algorithms can be learned from $O((1/\varepsilon)^d \cdot \log n)$ many random examples $(x, p(x))$ (which implies learnability even for $d = O(\log n)$), and (iii) degree-d polynomials $p : \{-1,1\}^n \to [-1,1]$ can be learned through $O(1/\varepsilon^d)$ queries to a quantum unitary U_p that block-encodes p. Our main technical contributions are new Bohnenblust-Hille inequalities for quantum channels and completely bounded polynomials.

2012 ACM Subject Classification Theory of computation → Quantum complexity theory

Keywords and phrases Tomography

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.13

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2405.10933` [6]

Funding This research was supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no. 945045, and by the NWO Gravitation project NETWORKS under grant no. 024.002.003.

Acknowledgements We thank the referees of ICALP'24 and TQC'24 for useful comments. We thank Jop Briët and Antonio Pérez Hernández for useful discussions. We thank Alexandros Eskenazis, Paata Ivanisvili, Alexander Volberg and Haonan Zhang for useful comments. A.D. and S.A. thank the Institute for Pure and Applied Mathematics (IPAM) for its hospitality throughout the long program "Mathematical and Computational Challenges in Quantum Computing" in Fall 2023 during which part of this work was initiated.

1 Introduction

Computational learning theory refers to the mathematical framework for understanding machine learning models and quantifying their complexity. The seminal result of Leslie Valiant [54] (who introduced the Probably Approximately Correct (PAC) model) gives a complexity-theoretic definition of what it means for a class of functions $f : \{0,1\}^n \to \{0,1\}$ to be learnable information-theoretically and computationally.

One of the foundational results in computational learning theory is the one of Linial, Mansour and Nisan [39] who showed that AC_0, or constant-depth n-bit classical circuits consisting of AND, OR and NOT gates, can be learned in quasi-polynomial time. A crucial aspect of their proof is the following structural theorem: if a Boolean function f is computable by AC_0 then f can be approximated by a *low-degree polynomial*. Using this structural property, the learning algorithm approximates the coefficients of all the low-degree monomials of f in the PAC model and hence approximately learns the unknown AC_0 function. Since their work, the notion of Boolean functions being low-degree or being well-approximated by low-degree polynomials has been a central technique [16] in obtaining new learning algorithms. Furthermore, low-degree approximations have played a significant role in theoretical computer science topics such as quantum computing, circuit complexity, learning theory and cryptography.

In the last few years, there have been several works in quantum learning theory where the goal has been to learn an unknown object on a quantum computer under various access models. Motivated by classical learning theory, in this work our primary focus will be on learning objects that have the additional structure of being *low-degree*. Since we consider different objects, when presenting our results we will make the definition of being low-degree clear, but the main motivation of this work can be summarized by the following question:

Can we learn low-degree n-qubit quantum objects information-theoretically with complexity that scales only polynomial (or better polylogarithmic) in n?

We give a positive answer to this question for low-degree channels, unitaries, quantum query algorithms, polynomials, and states. The organization of the paper is as follows. In Section 2 we present our main technical contribution, then in Section 3 we describe our applications to learning, in Section 4 we prove the result for channels, and in Section 5 we prove the result for quantum query algorithms. Due to page restrictions, we defer the preliminaries of our paper as well as the remaining proofs to the extended version [6].

2 Technical contribution: New Bohnenblust-Hille inequalities

In 1931, Bohnenblust and Hille [13] (generalizing the classic theorem of Littlewood [40]) gave a solution to the famous *Bohr strip problem* of Dirichlet series [14]. To do that, they showed the following: let $T : ([-1,1]^n)^d \to [-1,1]$ be a d-tensor specifed by the coefficients $T = (\widehat{T}_{i_1,\ldots,i_d})_{i_1,\ldots,i_d \in [n]}$, then

$$\Big(\sum_{i_1,\ldots,i_d=1}^{n} |\widehat{T}_{i_1,\ldots,i_d}|^{2d/(d+1)} \Big)^{(d+1)/2d} \leq C(d), \tag{1}$$

where $C(d)$ is a universal constant independent of n.[1] Their work marked the birth of the Bohnenblust-Hille (BH) inequality, which has became a fundamental tool in functional analysis. Despite being studied over a century, the best known upper bound on $C(d)$ scales polynomially with d, while the best lower bound is a constant. Closing this gap has been an active area of research in mathematics. In 2011 Defant et. al [22] refined the BH inequality and found a striking application of the BH inequality: they determined the precise asymptotic behavior of the n-dimensional Bohr radius using the BH inequality. Since then,

[1] We remark that the inequality above is a simplified version of the original Bohnenblust-Hille inequality and we discuss this in more detail in the preliminaries.

there has been renewed interest in the BH inequality and has found several applications in theoretical computer science such as Fourier-Entropy influence conjecture [4], classical learning theory [26], non-local games [42], and quantum computing [33, 57].

Of particular relevance to our work, the BH inequality recently captured the attention of the computer science community when Eskenazis and Ivanisvili [26] used a version of it to prove a major improvement in the problem of classical learning bounded low-degree multilinear polynomials (which we discuss in detail below). The key insight of Eskenazis and Ivanisvili is that BH inequalities imply that for bounded operators the small coefficients have a low total contribution, so one does not have to learn them. Multiple extensions of the BH have been proved since then and applied to learning quantum objects [33, 50, 51, 57, 36]. In this work, we extend the BH inequality in two ways:

1. We consider a variant of the BH inequality, that can be regarded as a hybrid between the BH inequality and the celebrated Grothendieck inequality [30]. We show that degree-d completely bounded tensors $\widehat{T}$ (which are known to be the the output of d-query quantum algorithms [3]) satisfy

$$\Big(\sum_{i_1,\ldots,i_d=1}^{n} |\widehat{T}_{i_1,\ldots,i_d}|^{2d/d+1}\Big)^{(d+1)/2d} \leq 1.$$

   In other words, we improve the BH constant for completely bounded tensors from poly(d) to 1. See Theorem 15 for a precise statement.
2. More recently, the works of [33, 57] considered non-commutative variants of the BH inequality. They showed that the Pauli coefficients of n-qubit degree-d (i.e., d-local) observables that are bounded in operator norm, can be bounded in a similar fashion to Eq. (1), but with exp(d) instead of poly(d). Here, we prove another non-commutative version of the BH inequality for quantum *channels* (we in fact prove a stronger BH inequality for *maps* that are bounded in $S_1 \to S_1$ norm, and refer the reader to Theorem 11). In particular, if Φ is a quantum channel defined as $\Phi(\rho) := \sum_{x,y\in\{0,1,2,3\}^n} \widehat{\Phi}(x,y)\sigma_x\rho\sigma_y$ (where $\sigma_x = \otimes_i \sigma_{x_i}$ and $\sigma_0 = I, \sigma_1 = X, \sigma_2 = Y, \sigma_3 = Z$ are the usual single-qubit Pauli operators) where $\widehat{\Phi}(x,y) = 0$ if $|x|, |y| > d$, then the Pauli coefficients $\widehat{\Phi}(x,y)$ satisfy

$$\Big(\sum_{x,y} |\widehat{\Phi}(x,y)|^{2d/(d+1/2)}\Big)^{(d+1/2)/2d} \leq \exp(d). \tag{2}$$

   While exp(d) in Equation (2) might seem much higher than the factor poly(d) of Equation (1), this is not a fair comparison. Equation (1) corresponds to tensors, which are a very structured class of polynomials, while Equation (2) is a non-commutative analogue of the BH inequality for general polynomials, for which the best known upper bounds [23] are superpolynomial in d.

These BH inequalities might be of independent interest both for mathematicians and quantum computing; in this work we crucially use them for our learning algorithms.

3 Applications and main results

3.1 Result 1: Learning channels

Learning a quantum process is a fundamental task in quantum computing and this can be modelled as learning an unknown *quantum channel*, also referred to as *quantum process tomography*. On an experimental level, the dynamics of closed quantum systems can be modeled as a unitary transformation from the initial state to the final state. However, in

practice, quantum systems interact with the environment and must be treated as an open quantum system. To learn the behavior of these open quantum systems, it is convenient to model this map as a quantum channel [45]. Learning an n-qubit quantum channel is however challenging and is known to require $\Theta(4^n)$ queries to the channel [31]. This exponential sample complexity can be drastically improved when prior information on the structure of the channel is available. For example, a recent work of Bao and Yao [10] considered k-junta quantum channels, i.e., n-qubit channels that act non-trivially only on at most k of the n (unknown) qubits leaving rest of qubits unchanged. These channels were shown to be learnable using $\widetilde{\Theta}(4^k)$ queries to the channel [10]. In [21], it was shown that quantum channels that can be efficiently generated, can be learned efficiently, albeit in the PAC learning model. In this work, we consider learning n-qubit quantum channels which have a Pauli decomposition only involving low-degree Pauli operators.

General quantum channels. To describe our result, we first describe Pauli analysis for quantum channels. An n-qubit to n qubit quantum channel Φ can be expressed as

$$\Phi(\rho) = \sum_{x,y\in\{0,1,2,3\}^n} \widehat{\Phi}(x,y)\cdot\sigma_x\rho\sigma_y, \tag{3}$$

where $\sigma_x = \otimes_{i\in[n]}\sigma_{x_i}$ and σ_i for $i\in\{0,1,2,3\}$ are the Pauli matrices

$$\sigma_0 = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix},\ \sigma_1 = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix},\ \sigma_2 = \begin{pmatrix}0 & -i\\ i & 0\end{pmatrix},\ \sigma_3 = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix};$$

and $\widehat{\Phi}(x,y)$ are the Pauli coefficients of the channel. Given $x\in\{0,1,2,3\}^n$, $|x|$ is the number of non-zero entries of x. The degree of a channel Φ is the minimum integer d such that $\widehat{\Phi}(x,y)=0$ if $|x|>d$ or $|y|>d$. Our first result is an efficient learning algorithm for low-degree channels. The learning model we consider is the same as the recent work of Bao and Yao [10]. Given a channel Φ, a learning algorithm is allowed to make *queries to* Φ as follows: it can apply Φ to an arbitrary ρ (or subsystem of ρ) of its choice and measure the resulting state in in any basis. From the measurement outcomes, the learner should output a classical description of a superoperator $\widetilde{\Phi}$ that is close to Φ in the ℓ_2-distance defined by the usual inner product for superoperators, i.e., $\langle\Phi,\widetilde{\Phi}\rangle = \mathrm{Tr}[J(\Phi),J(\widetilde{\Phi})]/4^n$, where $J(\Phi)$ is the Choi-Jaminkowski (CJ) representation of Φ.[2]

▶ **Theorem 1.** *Let Φ be a n-qubit degree-d quantum channel. There is an algorithm that (ε,δ)-learns Φ (in ℓ_2-distance) using $\exp\left(\widetilde{O}(d^2+d\log(1/\varepsilon))\right)\cdot\log(1/\delta)$ queries to Φ.*

We remark that the sample complexity of learning general quantum channels requires $\Omega(4^n)$ queries, but if we are promised the channel is *low-degree*, then our algorithm is much faster than the general algorithm. Additionally, observe that the sample complexity of our learning algorithm is independent of n, in contrast to the results [33, 57, 50, 51, 36] on quantum learning observables (which also are based on the BH inequality) that have a logarithmic dependence on n.

To prove the theorem, we first note the fact that the matrix $\widehat{\Phi}$ (whose entries are given by $\widehat{\Phi}_{x,y} = \widehat{\Phi}(x,y)$) is the density matrix of a state that is unitarily equivalent to the CJ state of the channel Φ [10, Lemma 8]. Hence, $\widehat{\Phi}$ can be prepared by applying Φ to the first n-qubits of n EPR pairs defined over $2n$ qubits, and $\{\widehat{\Phi}(x,x)\}_x$ is a probability distribution. The high-level idea behind the learning algorithm is the following.

[2] In this paper we say an algorithm (ε,δ)-learns a quantum object if it succeeds with probability $\geq 1-\delta$ and outputs an ε-approximation to the unknown quantum object in a metric that will be made clear.

1. Prepare T few copies of $\widehat{\Phi}$ and measure them in the computational basis, allowing the learner to sample from the distribution $\{\widehat{\Phi}(x,x)\}_x$.
2. Using a well-known result in distribution learning theory, we observe that $O(1/\alpha^2)$ many samples from $\{\widehat{\Phi}(x,x)\}_x$ suffices to obtain the xs such that $\Phi(x,x)$ is α-large.
3. Approximate all the large Pauli coefficients in the step above using a SWAP test for mixed states.
4. Output $\widetilde{\Phi}$ with coefficients that were estimated above and the remaining coefficients set to 0.

At this point, we use the BH inequality for quantum channels and show that, as long as $T \sim \exp(d^2/\varepsilon^d)$, then the output $\widetilde{\Phi}$ is ε-close to the target Φ in ℓ_2-distance.

Remark on learning Pauli channels. One can also consider a special case of quantum channels called *Pauli channels* Φ, motivated by the fact that Pauli channels are often the dominant noise on quantum devices and a practical noise model for analyzing fault-tolerance [52]. For these channels, the only non-zero terms in the Fourier expansion (3) are $\widehat{\Phi}(x,x)$ (i.e., $\widehat{\Phi}(x,y) = 0$ when $x \neq y$) and these coefficients are often called *error rates*. Since learning Pauli channels is an important task on near-term quantum devices for error mitigation [55] and analyzing error correction, it is desirable to avoid using entangled copies of a state and access to ancillary qubits as part of the learning algorithm. With this requirement, it was shown that, in order to ε-learn (in diamond norm) an unknown n-qubit Pauli channels using unentangled measurements, one needs to use the channel $\Omega(4^n/\varepsilon^2)$ many times [28]. In this work, we show that the subclass of low-degree Pauli channels are efficiently learnable.

Noise on current large-scale quantum devices is modeled as Pauli channels containing a sparse set of local Paulis [55], which is a subclass of low-degree Pauli channels. Such models are available from device physics and experiments used to characterize noise on quantum hardware. When non-local interactions but only over few qubits are included in the Pauli channel [53], the corresponding Paulis are still low-degree which fits into the class of Pauli channels considered. Our learning result is as follows.

▶ **Fact 2.** *Let Φ be an n-qubit degree-d Pauli channel. There is an algorithm that (ε,δ)-learns Φ (under the diamond norm) using $O\left(n^{2d}/\varepsilon^2 \cdot \log(n/\delta)\right)$ queries to Φ. The learning algorithm only requires preparation of product states and measurements in the Pauli basis.*

We remark that the dependence on n in our sample complexity matches the algorithm in [29] for learning low-degree Pauli channels under the diamond norm but our analysis differs from theirs; our result is obtained using a Fourier-analytic approach in contrast to the the result of [29] which uses ideas from population recovery.

3.2 Result 2: Learning unitaries

Apart from learning channels, in this paper we also consider the task of learning unknown n-qubit unitaries. Similar to the case for channels, it is well-known that learning an unstructured n-qubit unitary requires $\widetilde{\Theta}(4^n)$ applications of U [31]. This complexity can be significantly improved if structural information is available. For example, it has been show efficient learning is possible if the unitary corresponds to Clifford circuits [41], corresponds to Clifford circuits with few non-Clifford gates [38], or are the diagonal unitaries of the Clifford hierarchy [2].

In a recent work, Chen et al. [20] considered the task of learning unitaries U that are k-juntas and showed that this class can be learned querying the unitary $\widetilde{\Theta}(4^k)$ many times. In this work, we consider the scenario where the unitary is degree-d and show such an exponential saving (in comparison to naive tomography) is possible. This structured class of low-degree unitaries occur in many instances. For example, in nature, the dynamics of many physical

systems are governed by local Hamiltonians, whose unitary time evolution operators for short time evolution are close to being low-degree unitaries. Moreover, through the application of Lieb-Robinson bounds to structured many-body Hamiltonians, the corresponding unitary evolution operator can be seen to only have support only on low-degree Paulis [19]. In addition, the quantum unitaries corresponding to quantum circuits producing degree-d phase states containing all commuting diagonal quantum gates [2], are low-degree. The learning algorithm that we will present is thus applicable for learning and verifying such circuits.

Consider the Pauli decomposition of an n-qubit unitary as follows:

$$U = \sum_{x \in \{0,1,2,3\}^n} \widehat{U}(x)\sigma_x,$$

where $\widehat{U}(x)$ are its Pauli coefficients. The degree of U is the minimum integer d such that $|x| > d$ implies $\widehat{U}(x) = 0$. Our learning model is the one by Chen et al. [20]. Given an unitary U, a learning algorithm is allowed to make *queries to* U (and to control-U) as follows: it can choose a state ρ, apply U to the state to obtain $U\rho U^*$, and measure $U\rho U^*$ in a chosen basis. From the measurement outcomes, the learner should output a classical description of an operator $\widetilde{U}$ that is close to U in the ℓ_2-distance determined by the usual inner product for operators defined as $\langle U, V\rangle = \mathrm{Tr}[U^*V]/2^n$.

▶ **Theorem 3.** *Let U be a n-qubit degree-d unitary. There is an algorithm that (ε, δ)-learns U (in ℓ_2-distance) using the unitary U $\exp\left(\widetilde{O}(d^2 + d\log(1/\varepsilon))\right) \cdot \log(1/\delta)$ many times.*

The proof of Theorem 3 follows the same structure as that of Theorem 1, but now we learn the Pauli coefficients via an extension of the algorithm of Montanaro and Osborne [43], and the control on the contribution of the small coefficients relies on the non-commutative BH inequality of Volberg and Zhang [57].

The BH inequality of Volberg and Zhang [57] works for matrices with bounded operator norm, of which unitaries are a very special case. As argued by Montanaro and Osborne [43], matrices bounded on the operator norm are the quantum analogue of bounded functions $f : \{-1,1\}^n \to [-1,1]$, while unitaries are the analogue of Boolean functions[3] $f : \{-1,1\}^n \to \{-1,1\}$. These two families of classical functions differ a lot with respect to the BH inequalities: the best upper bound [23] for the BH constant for bounded functions is $\exp(d^{1/2})$, while for Boolean functions of degree d one can even prove that the *usually much bigger* quantity $\sum_s |\widehat{f}(s)|$ is at most 2^{d-1} [46, Exercise 1.11]. An open question is if this fact can be generalized to the quantum setting.

▶ **Question 4.** *Is there a constant $C(d)$ such that $\sum_{x\in\{0,1,2,3\}^n} |\widehat{U}(x)| \leq C(d)$ for every n-qubit degree-d unitary U?*

If Question 4 was answered positively, then one could improve the ε dependence of Theorem 3 to $(1/\varepsilon)^2$. Some evidence in favor of an affirmative answer to Question 4 is that if a conjecture of Montanaro and Osborne was true [43, Conjecture 4], then every degree-d hermitian unitary would be a 2^d-junta, which would imply an affirmative answer to Question 4 for the hermitian case.

Remark on learning low-degree quantum states. Learning quantum states has been an active line of research given its fundamental importance and applications to quantum system characterization, assessing quality of quantum gates, verification of quantum circuits and

[3] To be precise, they argue that unitary Hermitian matrices are the analogue of Boolean functions.

validating performance of quantum algorithms. Breakthrough results of Haah et al. [32] and O'Donnell and Wright [47] showed that the sample complexity of learning an unknown n-qubit state, up to trace distance ε is $\Theta(4^n/\varepsilon^2)$. A natural consideration is the task of learning *low-degree* quantum states. To describe this, we first write down the Pauli expansion of an n-qubit state ρ as

$$\rho = \sum_{x\in\{0,1,2,3\}^n} \widehat{\rho}(x)\sigma_x.$$

Then, we say that ρ has degree at most d if $\widehat{\rho}(x) = 0$ for all $|x|> d$. It is not too hard to see that one can use the formalism of classical shadows [34] to obtain a learning (in trace norm) algorithm that has a sample complexity of $\widetilde{O}(n^d/\varepsilon^2 \cdot \log(n/\delta))$. A similar result (with a different norm and with a bit more structure than just being low-degree) was noted in a recent work of Nadimpalli et al. [44], where they used the result to give efficient algorithms to learn QAC^0 circuits.

3.3 Result 3: Learning quantum query algorithms

Eskenazis and Ivanisvili [26] established a surprising connection between the BH inequality and learning theory. They considered the following question: suppose $f : \{-1,1\}^n \to [-1,1]$ is a bounded degree-d function, and a learner is given uniformly random x and $f(x)$, then how many $(x, f(x))$ suffices to learn f up to error ε in ℓ_2^2 error? The seminal low-degree algorithm of Linial, Mansour and Nisan uses[4] $O_{d,\varepsilon}(n^d)$ many such samples [39]. This was not improved until recently, when Iyer et al. [35] reduced this complexity to $O_{d,\varepsilon}(n^{d-1})$. In a surprising work, [26] showed that one can learn f in sample complexity $O_{d,\varepsilon}(\log n)$. For the particular case of bounded d-linear tensors $T : (\{-1,1\}^n)^d \to [-1,1]$ they showed that it suffices to use

$$(1/\varepsilon)^d \cdot \Big(\sum_{i_1,\ldots,i_d=1}^{n} |\widehat{T}_{i_1,\ldots,i_d}|^{2d/d+1} \Big)^{(d+1)/2d} \cdot \log n \tag{4}$$

samples $(x, T(x))$, where x is uniform from $(\{-1,1\}^n)^d$ and $\widehat{T}$ is the tensor of coefficients of T, i.e., $T(x) = \sum_{i_1,\ldots,i_d} \widehat{T}_{i_1,\ldots,i_d} x_1(i_1)\ldots x_d(i_d)$. Combining that with the upper bound of the BH constant for multilinear tensors [11], it yields

$$(d/\varepsilon)^{O(d)} \cdot O(\log n) \tag{5}$$

uniformly random samples are enough to learn T. Although this result is surprising since the complexity only scales polylogarithmic with n, observe that if $d = \omega(\log n)$, then the sample complexity is superpolynomial in n, motivating the natural question, are there classes of polynomials that can be learned using $\mathrm{poly}(n)$ samples for any $d = \omega(\log n)$? Below we show that the class of polynomials that arise from quantum query algorithms answers this question in the positive.

Quantum polynomials. The result of Equation (5) can be applied to learn the amplitudes of quantum algorithms that query different blocks of variables every time (see Figure 1), as they are multilinear tensors bounded on the supremum norm [12]. To be precise, we consider quantum query algorithms such that they prepare a state

$$|\psi_x\rangle = U_d(O_{x_d} \otimes \mathrm{Id}_m)U_{d-1}\ldots U_1(O_{x_1} \otimes \mathrm{Id}_m)U_0|u\rangle,$$

[4] Here and below, we use $O_{d,\varepsilon}$ to hide the factors that depend on $d, 1/\varepsilon$ and independent of n.

where m is an integer, x stands for $(x_1, \ldots, x_d)$, O_y is the n-dimensional unitary that maps $|i\rangle$ to $y_i|i\rangle$, $U_0, \ldots, U_d$ are $(n+m)$-dimensional unitaries and $|u\rangle$ is a $n+m$-dimensional unit vector. The algorithm succeeds according to a projective measurement that measures the projection of the final state onto some fixed $n+m$ dimensional unit vector $|v\rangle$. Hence, the amplitude of $|v\rangle$ is $T(x) = \langle v|\Psi_x\rangle$, so $|T(x)|^2$ is the acceptance probability of the algorithm. These quantum algorithms have been considered in the quantum computing literature. For

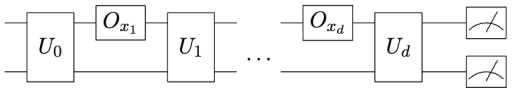


Figure 1 Quantum query algorithms considered in Theorem 5.

example, k-forrelation, that witnesses the biggest possible quantum-classical separation, has this structure [1, 8]. Also, for these algorithms the Aaronson and Ambainis conjecture is known to be true, so they can be classically and efficiently simulated almost everywhere [9, 25]. In addition, Arunachalam et al. showed that those amplitudes are not only bounded, but also completely bounded [3]. Our main contribution regarding these algorithms is showing that for d-linear tensors T that are *completely bounded*, we can improve the BH inequality to

$$\Big(\sum_{i_1,\ldots,i_d=1}^{n} |\widehat{T}_{i_1,\ldots,i_d}|^{2d/d+1} \Big)^{(d+1)/2d} \leq 1.$$

Using this upper bound, we show the following.

▶ **Theorem 5.** *For a quantum algorithm that makes d-queries as in Figure 1, its amplitudes can be learned up to error ε in ℓ_2^2 accuracy using $O\left((1/\varepsilon)^d \cdot \log n\right)$ uniformly random samples.*

This exponentially improves the complexity of [26], as stated in Eq. (5) for the natural class of polynomials arising from quantum query algorithms. In particular, for $d = \omega(\log n)$ and contant ε, one can learn this class of polynomials with sample complexity that is polynomial in n.

3.4 Result 4: Quantum learning of classical polynomials

3.4.1 Boolean functions

Quantum learning not only concerns quantum objects, but also classical ones. For instance, Boolean functions $f : \{-1,1\}^n \to \{-1,1\}$ can be accessed using a *quantum example*, given by

$$|\psi_f\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{-1,1\}^n} |x, f(x)\rangle.$$

This data access model has been vastly studied in the literature, where several notable quantum speedups have been proven [15, 7, 5]. Many of these speedups are analyzed trough the Fourier transform, that allows to identify every function via $f : \{-1,1\}^n \to \mathbb{R}$ with a multilinear polynomial $f = \sum_{s \in \{0,1\}^n} \widehat{f}(s)\chi_s$, where $\widehat{f}(s) \in \mathbb{R}$ are the Fourier coefficients and χ_s are the character functions $\chi_s(x) = \prod_{i \in [n]} x_i^{s_i}$. The degree for these functions is the minimum integer d such that if $|s| > d$ then $\widehat{f}(s) = 0$. It is a well-known fact that Boolean functions $f : \{-1,1\}^n \to \{-1,1\}$ of degree d are 2^{1-d}-granular, meaning that their Fourier coefficients lie in $2^{1-d}\mathbb{Z}$. This has immediate consequences for both learning theory and BH inequalities.

▶ **Fact 6.** *Let $f : \{-1,1\}^n \to \{-1,1\}$ be a degree-d function. There is a quantum algorithm that $(0,\delta)$ learns f using $O\left(4^d d \log(1/\delta)\right)$ quantum examples. However, a classical algorithm can learn it using $O\left(4^d d \log(n/\delta)\right)$ uniform examples and requires $\Omega(2^d + \log n)$ examples.*

Despite the simplicity of the proof of Fact 6, we state it for completeness and because it seems not to be well-known (see for instance [44, Corollary 34], which proposes a quantum algorithm for the same problem that requires $O(n^d)$ samples, or [27, Corollary 4] that proposes a classical algorithm that requires $O(2^{d^2} \log n)$ samples). Furthermore, we observe a BH-type inequality for Boolean functions.

▶ **Fact 7.** *Let $f : \{-1,1\}^n \to \{-1,1\}$ of degree at most d. Then,*

$$\Big(\sum_{s\in\{0,1\}^n} |\widehat{f}(s)|^{\frac{2d}{d+1}} \Big)^{\frac{d+1}{2d}} \leq 2^{\frac{d-1}{d}}.$$

The equality it is witnessed by the address function.

Fact 7 might be of interest in functional analysis for two reasons: (i) it is conjectured that the value of the BH constant for d-linear tensors is $2^{\frac{d-1}{d}}$ [48], so this fact proves the conjecture for particular case of d-linear Boolean tensors, (ii) the address function[5], that saturates the inequality, is a d-linear form that gives a lower bound for the BH constant for multilinear tensors of $2^{\frac{d}{d-1}}$, which matches the best lower bound known so far [24].

3.4.2 Real-valued polynomials

For bounded functions $f : \{-1,1\}^n \to [-1,1]$ the definition of quantum uniform examples $|\psi_f\rangle$ is unclear. Given that bounded polynomials have received attention in a few works [39, 35, 26], we propose a way learning them quantumly by accessing these polynomials through a block encoding [18], i.e., a learning algorithm has access to a block encoding of the 2^n-dimensional diagonal matrix whose diagonal entries all equal f. To this end, we prove the following theorem.

▶ **Proposition 8.** *Let $f : \{-1,1\}^n \to [-1,1]$ be a degree-d polynomial. There is an algorithm that (ε,δ)-learns f (in ℓ_2-distance) using $\exp(\widetilde{O}(\sqrt{d^3} + d\log(1/\varepsilon))\log(1/\delta))$ copies of a block-encoding of f.*

The proof of Proposition 8 is a combination of the ideas of Eskenazis and Ivanisvili [26] with the Fourier sampling and block-encoding quantum primitives. We remind the reader that the merit of Eskenazis and Ivanisvili [26] was to bring down the classical complexity of the problem from $O_{d,\varepsilon}(n^d)$ to $O_{d,\varepsilon}(\log n)$. Proposition 8 shows that the quantum complexity this could be even reduced to $O_{d,\varepsilon}(1)$. Proposition 8 also implies a quantum speedup (with respect to n), as the lower bound of $\Omega(2^d + \log n)$ also holds for membership queries[6], which are the classical analogue of accessing a unitary block-encoding of f.

[5] For $d \in \mathbb{N}$, the address function $f : (\{-1,1\}^2)^{d-1} \times \{-1,1\}^{2^{d-1}} \to \{-1,1\}$ is defined as $f(x,y) = \sum_{a\in\{-1,1\}^{d-1}} g_a(x)y(a)$, where we identify $\{-1,1\}^{d-1}$ with $[2^{d-1}]$ and $g_a(x)$ is 0 unless $x_i(1) = a_i x_i(2)$ for every $i \in [n]$, in which case it takes the value $\prod_{i\in[n]} x_i(1)$.

[6] Making a membership query to f consists on accessing one pair $(x, f(x))$ where x is chosen by the learner, not necessarily uniformly at random. If f is a Boolean function and U_f is the unitary defined by $U_f|x\rangle = f(x)|x\rangle$, then a membership query $(x, f(x))$ can be simulated by applying $(H\otimes \mathrm{Id}_n)CU_f(H\otimes \mathrm{Id}_n)$ to $|0\rangle|x\rangle$ and measuring the first qubit in the computational basis. Note that U_f can be regarded as a block-encoding of f.

Remarkably, we highlight that although our results are about sample complexity, the time complexity of our quantum algorithm of Proposition 8 scales as $\mathrm{poly}(n, \exp(d^{1.5}))$, while the current state-of-the-art classical algorithm [26] approximates the $\binom{n}{d}$ Fourier coefficients, so it requires $\mathrm{poly}(n^d)$ time. In particular, for $d = (\log n)^{2/3}$, constant ε, δ, our quantum algorithm has a time complexity of $\mathrm{poly}(n)$ time, while the state-of-the-art classical algorithm runs in time $n^{\mathrm{polylog} n}$.

4 Learning algorithm for low-degree quantum channels

In this section we give our learning algorithm for quantum channels, i.e., prove Theorem 1. To do that, we first need to prove a new BH inequality for quantum channels.

4.1 Bohnenblust-Hille inequality for quantum channels

In this section, we prove a Bohnenblust-Hille inequality for n-qubit quantum channels. In fact, it is a result for superoperators which are bounded in the S_1 to S_1 norm (defined below), of which quantum channels are a particular example. Hence, we will treat Φ as a linear map from $\mathcal{M}_N$ to $\mathcal{M}_N$, the space of N-dimensional matrices with $N = 2^n$. In particular, we will evaluate Φ on matrices that are not states. The S_1 to S_1 norm of superoperator is defined by

$$\|\Phi\|_{S_1 \to S_1} = \sup_{M \neq 0} \frac{\|\Phi(M)\|_{S_1}}{\|M\|_{S_1}},$$

where $\|M\|_{S_1}$ is the Schatten 1-norm of M, i.e., the sum of the singular values of M.

To prove our theorem will reduce to the classical case of functions $f : \{-1,1\}^n \to \mathbb{R}$.

▶ **Theorem 9** ([23]). *Let $p : \{-1,1\}^n \to \mathbb{R}$ of degree at most d. Then,*

$$\|\widehat{p}\|_{\frac{2d}{d+1}} \leq C^{\sqrt{d \log d}} \|p\|_\infty,$$

where $C > 0$ is a constant.

To achieve this reduction, for every superoperator $\Phi : \mathcal{M}_M \to \mathcal{M}_N$, we assign it a function $f_\Phi : \{-1,1\}^{3n} \times \{-1,1\}^{3n} \to \mathbb{C}$ defined as follows. For $a = (a^1, a^2, a^3)$, $b = (b^1, b^2, b^3) \in \{-1,1\}^n \times \{-1,1\}^n \times \{-1,1\}^n$ and $s, t \in \{1,2,3\}^n$, define the following matrices (which are not necessarily states)

$$|a^s\rangle\langle b^t| = \bigotimes_{i \in [n]} |\chi^{s(i)}_{a^s(i)}\rangle\langle\chi^{t(i)}_{b^t(i)}|,$$

where $|\chi^s_{\pm 1}\rangle$ are the ± 1 eigenstates of the single-qubit Pauli operators σ_s. The function $f_\Phi : \{-1,1\}^{3n} \times \{-1,1\}^{3n} \to \mathbb{C}$ is then given by

$$f_\Phi(a,b) = \frac{1}{9^n} \sum_{s,t \in \{1,2,3\}^n} \mathrm{Tr}[\Phi\left(|a^s\rangle\langle b^t|\right) |b^t\rangle\langle a^s|],$$

where f_Φ has the following properties, allowing us to reduce to the classical BH inequality.

▶ **Lemma 10.** *Let Φ be a degree-d superoperator. Then, $|f_\Phi(a,b)| \leq \|\Phi\|_{S_1 \to S_1}$ for all a, b and $\left\|\widehat{\Phi}\right\|_p \leq 9^d \left\|\widehat{f_\Phi}\right\|_p$. The degree of f_Φ as a multilinear polynomials is $2d$.*

Proof. We first show the bound on $|f_\Phi|$. Given that $(|a^s\rangle\langle b^t|)(|a^s\rangle\langle b^t|)^* = |a^s\rangle\langle a^s|$, we have that

$$\left\||a^s\rangle\langle b^t|\right\|_{S_1} = \left\||a^s\rangle\langle b^t|\right\|_{S_\infty} = 1. \tag{6}$$

Thus, we have that f_Φ is bounded:

$$\begin{aligned}
|f_\Phi(a,b)| &\leq \frac{1}{9^n} \sum_{s,t\in\{1,2,3\}^n} |\mathrm{Tr}[\Phi\left(|a^s\rangle\langle b^t|\right)|b^t\rangle\langle a^s|]| \\
&\leq \frac{1}{9^n} \sum_{s,t\in\{1,2,3\}^n} \left\|\Phi\left(|a^s\rangle\langle b^t|\right)\right\|_{S_1} \left\||b^t\rangle\langle a^s|\right\|_{S_\infty} \\
&\leq \frac{1}{9^n} \sum_{s,t\in\{1,2,3\}^n} \|\Phi\|_{S_1\to S_1} \left\||a^s\rangle\langle b^t|\right\|_{S_1} \left\||b^t\rangle\langle a^s|\right\|_{S_\infty} \\
&\leq \frac{1}{9^n} \sum_{s,t\in\{1,2,3\}^n} \|\Phi\|_{S_1\to S_1} = \|\Phi\|_{S_1\to S_1},
\end{aligned}$$

where in the first inequality we have used the triangle inequality, in the second inequality Riesz theorem, in the third the definition of $S_1 \to S_1$ norm and in the fourth Equation (6). We now prove that $\left\|\widehat{\Phi}\right\|_p \leq 9^{-d}\left\|\widehat{f_\Phi}\right\|_p$ and that the degree of f_Φ is $2d$. It suffices to show that

$$f_\Phi(a,b) = \sum_{x,y\in\{0,1,2,3\}^n} \frac{\widehat{\Phi}(x,y)}{3^{|x|+|y|}} \prod_{i\in\mathrm{supp}(x)} \prod_{j\in\mathrm{supp}(y)} a_i^{x(i)} b_j^{y(j)}, \tag{7}$$

where $\mathrm{supp}(x) = \{i \in [n] : x_i \neq 0\}$ and $|x|$ is the size of $\mathrm{supp}(x)$. To prove Equation (7) the key is observing that for every $s,t \in \{1,2,3\}$, $x,y \in \{0,1,2,3\}$ and $a,b \in \{-1,1\}$ we have that

$$\mathrm{Tr}[\sigma_x|\chi_a^s\rangle\langle\chi_b^t|\sigma_y|\chi_b^t\rangle\langle\chi_a^s|] = \begin{cases} 0 & \text{if } (s\neq x \text{ and } x\neq 0) \text{ or } (t\neq y \text{ and } y\neq 0), \\ 1 & \text{if } x=0 \text{ and } y=0, \\ a & \text{if } s=x \text{ and } y=0, \\ b & \text{if } x=0 \text{ and } t=y, \\ ab & \text{if } s=x \text{ and } y=t. \end{cases}$$

After taking tensor products, we observe that for every $s,t \in \{1,2,3\}^n$, $x,y \in \{0,1,2,3\}^n$ and $a=(a^1,a^2,a^3)$, $b=(b^1,b^2,b^3) \in \{-1,1\}^n \times \{-1,1\}^n \times \{-1,1\}^n$, it holds that

$$\mathrm{Tr}[\sigma_x|a^s\rangle\langle b^t|\sigma_y|b^t\rangle\langle a^s|] = \prod_{i\in\mathrm{supp}\,x} \prod_{j\in\mathrm{supp}\,y} a_i^{x(i)} b_j^{y(j)} \delta_{x(i),s(i)} \delta_{y(j),t(j)},$$

where $\delta_{x,y}$ is the delta function taking value of 1 when $x=y$, and 0 otherwise. In particular, from this follows that

$$f_{\Phi_{x,y}}(a,b) = \frac{1}{9^n} \sum_{s,t\in\{1,2,3\}^n} \mathrm{Tr}[\sigma_x|a^s\rangle\langle b^t|\sigma_y|b^t\rangle\langle a^s|] = \frac{1}{9^n} \prod_{i\in\mathrm{supp}\,x} \prod_{j\in\mathrm{supp}\,y} a_i^{x(i)} b_j^{y(j)} \sum_{s\in\mathcal{X},t\in\mathcal{Y}} 1,$$

where $\mathcal{X} = \{s \in \{1,2,3\}^n : s(i) = x(i)\ \forall\ i \in \mathrm{supp}(x)\}$. Hence, as $|\mathcal{X}| = 3^{n-|x|}$, Equation (7) follows for the case of $\Phi_{x,y}$. By linearity, Equation (7) follows for every superoperator. ◀

▶ **Theorem 11** (Bohnenblust-Hille inequality for $S_1 \to S_1$ maps). *Let Φ be a super-operator of degree at most d. Then there exists a constant C such that*

$$\left\|\widehat{\Phi}\right\|_{\frac{2d}{d+1/2}} \leq C^d \|\Phi\|_{S_1\to S_1}.$$

In particular, if Φ is a quantum channel, then there exists a constant C such that

$$\left\|\widehat{\Phi}\right\|_{\frac{2d}{d+1/2}} \leq C^d.$$

Proof. Let $\Re f_\Phi : \{-1,1\}^n \to \mathbb{R}$ be defined as taking the real part of f_Φ i.e., $(\Re f_\phi)(x) = \Re(f_\Phi(x))$ and $\Im f_\Phi : \{-1,1\}^n \to \mathbb{R}$ as taking the imaginary part of f_Φ i.e., $(\Im f_\phi)(x) = \Im(f_\Phi(x))$. Note that we have that $\widehat{f}_\phi = \widehat{\Re f}_\Phi + i\widehat{\Im f}_\Phi$. By Lemma 10, $|(\Re f_\phi)(x)|, |(\Im f_\phi)(x)| \leq |f_\Phi(x)| \leq \|\Phi\|_{S_1\to S_1}$ and that the degree of both the real and imaginary part is at most $2d$. Hence, by the triangle inequality and Theorem 9 we have

$$\left\|\widehat{f}_\Phi\right\|_{\frac{4d}{2d+1}} \leq \left\|\widehat{\Re f}_\Phi\right\|_{\frac{4d}{2d+1}} + \left\|\widehat{\Im f}_\Phi\right\|_{\frac{4d}{2d+1}} \leq C\sqrt{d\log d}\,\|\Phi\|_{S_1\to S_1}\,.$$

Thus, using that $\left\|\widehat{\Phi}\right\|_{2d/(d+1)} \leq 9^d \left\|\widehat{f}_\Phi\right\|_{2d/(d+1)}$ it follows that $\left\|\widehat{\Phi}\right\|_{2d/(d+1)} \leq C^d \|\Phi\|_{S_1\to S_1}$. This proves the first part of the statement.

For the second we just have to show that if Φ is a quantum channel, then $\|\Phi\|_{S_1\to S_1}$ is bounded by a constant. Indeed, if M is self-adjoint, we can write it as $M = M^+ - M^-$, with M^+ and M^- being positive semidefinite, so

$$\frac{\|\Phi(M)\|_{S_1}}{\|M\|_{S_1}} \leq \frac{\mathrm{Tr}[\Phi(M^+)] + \mathrm{Tr}[\Phi(M^-)]}{\mathrm{Tr}[M^+] + \mathrm{Tr}[M^-]} = 1, \tag{8}$$

where in the first equality we have used that Φ is positive and in the second that is trace perserving. Finally, any matrix M can be written as $M = \Re M + i\Im M$, where $\Re M = (M + M^*)/2$ and $\Im M = (M - M^*)/2$ are self-adjoint. Hence,

$$\frac{\|\Phi(M)\|_{S_1}}{\|M\|_{S_1}} \leq \frac{\|\Phi(\Re M)\|_{S_1} + \|\Phi(\Im M)\|_{S_1}}{\|M\|_{S_1}} \leq \frac{\|\Re M\|_{S_1} + \|\Im M\|_{S_1}}{\|M\|_{S_1}} \leq \frac{\|M\|_{S_1} + \|M\|_{S_1}}{\|M\|_{S_1}} = 2,$$

where in the first inequality we have used the triangle inequality, in the second inequality we have used Equation (8) and in the third inequality that $\|\Re M\|_{S_1}, \|\Im M\|_{S_1} \leq \|M\|_{S_1}$. ◀

4.2 Learning low-degree quantum channels

Before we prove the main theorem of the section, we show that for given $x, y \in \{0,1,2,3\}^n$, the corresponding Fourier coefficient $\widehat{\Phi}(x,y)$ can be efficiently learned. This is accomplished through the combination of a few SWAP tests.

▶ **Fact 12** (SWAP test for mixed states [37]). *Let ρ, ρ' be two states. Then, one can estimate $\mathrm{Tr}[\rho\rho']$ up to error ε with probability $1-\delta$ using $O((1/\varepsilon)^2 \log(1/\delta))$ copies of ρ and ρ'.*

▶ **Lemma 13** (Pauli coefficient estimation for channels). *Let $x, y \in \{0,1,2,3\}^n$. Then, $\widehat{\Phi}(x,y)$ can be estimated with error ε and probability $1-\delta$ using $O((1/\varepsilon)^2 \log(1/\delta))$ queries to Φ.*

Proof. If $x = y$, we just have to prepare $\widehat{\Phi}$ (which can be done by preparing the Choi state $J(\Phi)$ following by a unitary transformation) and apply Fact 12 to $\widehat{\Phi}$ and the state $\rho = |x\rangle\langle x|$. If $x \neq y$, one first learns $\widehat{\Phi}(x,x)$ and $\widehat{\Phi}(y,y)$ with error ε as before. One the one hand, one can learn $\widehat{\Phi}(x,x) + \widehat{\Phi}(x,x) + 2\Re\widehat{\Phi}(x,y)$, with error ε by applying Fact 12 to $\widehat{\Phi}$ and $1/2\sum_{z,t\in\{x,y\}} |z\rangle\langle t|$. Hence, one learns $\Re\widehat{\Phi}(x,y)$ with error $3\varepsilon/2$. On the other hand, one can learn $\widehat{\Phi}(x,x) + \widehat{\Phi}(y,y) + 2\Im\widehat{\Phi}(x,y)$, with error ε by applying Fact 12 to $\widehat{\Phi}$ and $1/2(|x\rangle\langle x| + i|x\rangle\langle y| - i|y\rangle\langle x| + |y\rangle\langle y|)$, and thus one can learn $\Im\widehat{\Phi}(x,y)$ with error $3\varepsilon/2$. ◀

We will also need the following well-known result on learning discrete probability distributions. See [17, Theorem 9] for a proof.

▶ **Lemma 14.** *Let $p = \{p(x)\}_x$ be a probability distribution over some set $\mathcal{X}$. Let $p' = (p'(x))_x$ the empirical probability distribution obtained after sampling T times from p. Then, for $T = O((1/\varepsilon)^2 \log(1/\delta))$ with probability $1-\delta$ we have that $|p(x) - p'(x)| \leq \varepsilon$ for every $x \in \mathcal{X}$.*

Now, we are ready to prove Theorem 1, which we restate for the convenience of the reader.

▶ **Theorem 1.** *Let Φ be a n-qubit degree-d quantum channel. There is an algorithm that (ε, δ)-learns Φ (in ℓ_2-distance) using $\exp\left(\widetilde{O}(d^2 + d\log(1/\varepsilon))\right) \cdot \log(1/\delta)$ queries to Φ.*

Proof. We first state the algorithm.

Algorithm 1 Learning low-degree channels via BH inequality.

Input: A quantum channel Φ of degree at most d, and error ε and a failure probability δ

1: Let $c = \varepsilon^{2d+1} C^{-d^2}$
2: Prepare $T_1 = O((1/c)^2 \log(1/\delta))$ copies of $\widehat{\Phi}$ to sample from $(\widehat{\Phi}(x,x))_x$. Let $(\widehat{\Phi}'(x,x))_x$ be the associated empirical distribution
3: **for** $x, y \in \mathcal{X}_c = \{x : |\widehat{\Phi}'(x,x)| \geq c\}$ **do**
4: Prepare $O((1/c)^2 (1/\varepsilon)^2 \log((1/c)^2 (1/\delta)))$ copies of $\widehat{\Phi}$ and use them to approximate $\widehat{\Phi}(x,y)$ with $\widehat{\Phi}''(x,y)$ using Lemma 13.
5: **end for**

Output: $\sum_{x,y \in \mathcal{X}_c} \widehat{\Phi}''(x,y) \Phi_{x,y}$

Let $c > 0$ to be determined later. In the first part of the algorithm we prepare $\widehat{\Phi}$ and measure, i.e., we sample from $(\widehat{\Phi}(x,x))_{x \in \{0,1,2,3\}^n}$. Let $(\widehat{\Phi}'(x,x))_{x \in \{0,1,2,3\}^n}$ be the empirical distribution one obtains after T_1 samples. $\mathcal{E} = \{|\widehat{\Phi}(x,x) - \widehat{\Phi}'(x,x)| \leq c \ \forall x \in \{0,1,2,3\}^n\}$. By Lemma 14, taking T_1 to be $O((1/c)^2 \log(1/\delta))$ ensures that

$$\Pr[\mathcal{E}] \geq 1 - \delta.$$

Let $\mathcal{X}_c = \{x : \ |\widehat{\Phi}'(x,x)| \geq c\}$. Note that, as $\sum_{x \in \mathcal{X}_c} \Phi(x,x) \leq 1$,

$$|\mathcal{X}_c| \leq c^{-1}, \tag{9}$$

and in the event of $\mathcal{E}$ we have that

$$x \notin \mathcal{X}_c \implies |\widehat{\Phi}(x,x)| \leq |\widehat{\Phi}'(x,x)| + ||\widehat{\Phi}(x,x)| - |\widehat{\Phi}'(x,x)|| \leq 2c. \tag{10}$$

In particular, it follows that

$$x \notin \mathcal{X}_c \implies |\widehat{\Phi}(x,y)| \leq \sqrt{|\widehat{\Phi}(x,x)||\widehat{\Phi}(y,y)|} \leq \sqrt{2c} \quad \forall\, y \in \{0,1,2,3\}^n, \tag{11}$$

where in the first inequality we have used that $\widehat{\Phi}$ is positive semidefinite and in the second inequality Equation (10) and that $\widehat{\Phi}(y,y) \leq 1$. We assume that the first part of the algorithm succeeds, meaning that $\mathcal{E}$ happens. In the second part of the algorithm we approximate all the Pauli coefficients of $\mathcal{X}_c \times \mathcal{X}_c$ with error $c\varepsilon$ and probability $1 - \delta$ querying Φ just

$$T_2 = O((1/c)^4 (1/\varepsilon)^2 \log((1/c)^2 (1/\delta)))$$

times from Lemma 13. Note that $T_2 > T_1$, so this complexity dominates the one of the first part of the algorithm. Let $\widehat{\Phi}''(x,y)$ be these approximations and let $\Phi_c = \sum_{x,y \in \mathcal{X}_c} \widehat{\Phi}''(x,y) \sigma_x \cdot \sigma_y$. Now, we have that

$$\begin{aligned}\|\Phi - \Phi_c\|_2^2 &= \sum_{x,y\in\mathcal{X}_c} |\widehat{\Phi}(x,y) - \widehat{\Phi}''(x,y)|^2 + \sum_{x\vee y\notin\mathcal{X}_c} |\widehat{\Phi}(x,y)|^2 \\ &\leq \varepsilon^2 + \sum_{x\vee y\notin\mathcal{X}_c} |\widehat{\Phi}(x,y)|^{\frac{1}{d+1/2}} |\widehat{\Phi}(x,y)|^{\frac{2d}{d+1/2}} \\ &\leq \varepsilon^2 + (2c)^{\frac{1/2}{d+1/2}} \left\|\widehat{\Phi}\right\|_{\frac{2d}{d+1/2}}^{\frac{2d}{d+1/2}} \\ &\leq \varepsilon^2 + c^{\frac{1/2}{d+1/2}} C^d,\end{aligned}$$

where in the equality we have used Parseval's identity; in the first inequality we used Equation (9), the learning guarantees of the second part of the algorithm and that $2 = 1/(d+1/2) + 2d/(d+1/2)$; in the second inequality we have used Equation (11); and in the third inequality we used the Bohnenblust-Hille inequality for channels (Theorem 11). Hence, by choosing

$$c = \varepsilon^{4d+2} C^{-d^2}$$

we obtain the desired result. ◀

5 Learning quantum query algorithms

In this section, our goal will be to prove Theorem 5, restated below for the reader's convenience.

▶ **Theorem 5.** *For a quantum algorithm that makes d-queries as in Figure 1, its amplitudes can be learned up to error ε in ℓ_2^2 accuracy using $O\left((1/\varepsilon)^d \cdot \log n\right)$ uniformly random samples.*

Proof. Eskenazis and Ivanishvili [26] showed that a function $f : \{-1,1\}^n \to [-1,1]$ with degree at most d and $\left\|\widehat{f}\right\|_{\frac{2d}{d+1}} \leq C$, can be learned with success probablity $1-\delta$ and error ε in ℓ_2^2 accuracy using $O(\varepsilon^{-(d+1)} C^{2d} \log(n/\delta))$ samples $(x, f(x))$, where x is drawn uniformly at random from $x \in \{-1,1\}^n$. Arunachalam et al. [3] showed that the amplitudes of quantum algorithms that d queries as in Figure 1 are completely bounded d-tensors. Hence, using Theorem 15 (which we prove below) we can let $C = 1$, and obtain the desired result. ◀

5.1 The constant of the completely bounded BH inequality is 1

In this section we determine that the exact value of the constant of the completely bounded BH inequality is 1. Before that, we first define the completely bounded norm. For d-linear tensors, the completely bounded norm is defined as

$$\|T\|_{\text{cb}} = \sup \left\| \sum_{\mathbf{i}\in[n]^d} \widehat{T}_{\mathbf{i}} X_1(i_1) \dots X_t(i_t) \right\|_{\text{op}}, \tag{12}$$

where $X_s(i_s)$ are matrices of size $m \times m$ that have operator norm at most 1 and $m \in \mathbb{N}$.

▶ **Theorem 15.** *Let $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$. Let $T : (\mathbb{K}^n)^d \to \mathbb{K}$ be a d-linear form. Then,*

$$\left\|\widehat{T}\right\|_{\frac{2d}{d+1}} \leq \|T\|_{\text{cb}},$$

and the inequality can be saturated.

Theorem 15 establishes that the best constant for the completely bounded BH inequality is exactly 1. This sharply contrasts with the current knowledge about the BH constants, where only poly(d) upper bounds are known. We thus *close* one of the edges of comparison of the three norms that appear in Grothendieck and Bohnenblust-Hille inequalities (see Figure 2).

Figure 2 Triangles of norm comparisons. In the left triangle, we display the norm comparisons implied by the Littlewood [40] and Grothendieck inequalities [30] and our Theorem 15 for real bilinear maps. In the right triangle, we depict the best upper bound for the BH constant [11], the no extension of the Grothendieck inequality [49], and our Theorem 15 for d-linear tensors.

The main ingredient of the proof of Theorem 15 is a general lower bound for the completely bounded norm, Lemma 17. This technique is inspired by the idea of Varoupoulos to rule out a generalization of von Neumann's inequality [56] and was recently used by Escudero-Gutiérrez to show a particular case of the famous Aaronson and Ambainis conjecture of quantum query complexity [25]. Theorem 15 follows from combining Lemma 17 with Blei's inequality Lemma 16. See [11, Theorem 2.1] for a proof of Blei´s inequality.

▶ **Lemma 16** (Blei's inequality). *Let $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$ and let $\widehat{T} \in (\mathbb{K})^{n^d}$. Then,*

$$\Big(\prod_{s\in[d]}\sum_{i_s\in[n]}\sqrt{\sum_{i_1,\ldots,i_{s-1},i_{s+1},\ldots,i_d\in[n]}|\widehat{T}_{\mathbf{i}}|^2}\Big)^{\frac{1}{d}} \geq \left\|\widehat{T}\right\|_{\frac{2d}{d+1}}.$$

▶ **Lemma 17.** *Let $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$, let $T : (\mathbb{K}^n)^d \to \mathbb{K}$ be a d-linear form, and let $s \in [d]$. Then,*

$$\|T\|_{\mathrm{cb}} \geq \sum_{i_s\in[n]}\sqrt{\sum_{i_1,\ldots,i_{s-1},i_{s+1},\ldots,i_d\in[n]}|\widehat{T}_{\mathbf{i}}|^2}.$$

Proof. The proof involves evaluating Equation (12) on an explicit set of contractions (matrices with operator norm at most 1). Let $m = \sum_{r=0}^{s-1} n^r + \sum_{r=0}^{d-r} n^r$ and let $\{e_{\mathbf{i}}, f_{\mathbf{j}} : \mathbf{i} \in [n]^r, r \in \{0\} \cup [d-s], \mathbf{j} \in [n]^t, t \in \{0\} \cup [s-1]\}$ be an orthonormal basis of $\ell_2^m(\mathbb{K})$, where we identify $[n]^0$ with $\emptyset$. We define $m \times m$ matrices $X(i)$ for $i \in [n]$ by

$$X(i)e_{\mathbf{j}} = e_{(i,\mathbf{j})}, \text{ if } \mathbf{j} \in [n]^r,\ r \in \{0\} \cup [d-s-1],$$
$$X(i)e_{\mathbf{j}} = \frac{\sum_{\mathbf{k}\in[n]^{s-1}} \widehat{T}^*_{\mathbf{k}i\mathbf{j}} f_{\mathbf{k}}}{\sqrt{\sum_{k_1,\ldots,k_{s-1},k_{s+1},\ldots,k_d\in[n]}|\widehat{T}|^2_{(k_1,\ldots,k_{s-1},i,k_{s+1},\ldots,k_d)}}}, \text{ if } \mathbf{j} \in [n]^{d-s},$$
$$X(i)f_{\mathbf{j}} = \delta_{i,j_s} f_{(i_1,\ldots,i_{s-1})}, \text{ if } \mathbf{j} \in [n]^{d-s},\ r \in \{0\} \cup [d-1].$$

For some intuition of the behaviour of these matrices, one may interpret the first $d-s$ applications of the matrices $X(i)$ as *creation* operators and the last $s-1$ as *destruction* operators. Assume for the moment that $X(i)$ are contractions. Given that,

$$\langle f_\emptyset, X(i_1)\ldots X(i_d)e_\emptyset\rangle = \frac{\widehat{T}_{\mathbf{i}^*}}{\sqrt{\sum_{i_1,\ldots,i_{s-1},i_{s+1},\ldots,i_d\in[n]}|\widehat{T}_{\mathbf{i}}|^2}},$$

it would then follow that

$$\begin{aligned}\|T\|_{\mathrm{cb}} &\geq \left\| \sum_{\mathbf{i}\in[n]^d} \widehat{T}_{\mathbf{i}} X(i_1)\dots X(i_d) \right\|_{\mathrm{op}} \\ &\geq \sum_{\mathbf{i}\in[n]^d} \widehat{T}_{\mathbf{i}} \frac{\widehat{T}^*_{\mathbf{i}}}{\sqrt{\sum_{k_1,\dots,k_{s-1},k_{s+1},\dots,k_d\in[n]} |\widehat{T}|^2_{(k_1,\dots,k_{s-1},i_s,k_{s+1},\dots,k_d)}}} \\ &= \sum_{i_s\in[n]} \sqrt{\sum_{i_1,\dots,i_{s-1},i_{s+1},\dots,i_d\in[n]} |\widehat{T}_{\mathbf{i}}|^2},\end{aligned}$$

as desired. It then remains to prove that the matrices $X(i)$ are contractions. Given that $X(i)$ maps $\{e_{\mathbf{i}} : \mathbf{i}\in[n]^r,\, r\in\{0\}\cup[d-s-1]\}$, $\{e_{\mathbf{i}} : \mathbf{i}\in[n]^{d-s}\}$ and $\{f_{\mathbf{i}} : \mathbf{i}\in[n]^r,\, r\in\{0\}\cup[s]\}$ to orthogonal subspaces, it suffices to show that the $X(i)$ are contractions when restricted to those subspaces. For the first and third sets, this is true because $X(i)$ maps each basis vector of these sets to a different basis vector or to 0. For the second set, is also true because for every $\lambda\in\mathbb{K}^{n^{d-s}}$

$$\begin{aligned}\left\| X(i) \sum_{\mathbf{j}\in[n]^{d-s}} \lambda_{\mathbf{j}} e_{\mathbf{j}} \right\|_2 &= \left\| \frac{\sum_{\mathbf{k}\in[n]^{s-1}} (\sum_{\mathbf{j}\in[n]^{d-s}} \lambda_{\mathbf{j}} \widehat{T}^*_{\mathbf{k}i\mathbf{j}}) f_{\mathbf{k}}}{\sqrt{\sum_{k_1,\dots,k_{s-1},k_{s+1},\dots,k_d\in[n]} |\widehat{T}|^2_{(k_1,\dots,k_{s-1},i,k_{s+1},\dots,k_d)}}} \right\|_2 \\ &= \frac{\sqrt{\sum_{\mathbf{k}\in[n]^{s-1}} |\sum_{\mathbf{j}\in[n]^{d-s}} \lambda_{\mathbf{j}} \widehat{T}^*_{\mathbf{k}i\mathbf{j}}|^2}}{\sqrt{\sum_{k_1,\dots,k_{s-1},k_{s+1},\dots,k_d\in[n]} |\widehat{T}|^2_{(k_1,\dots,k_{s-1},i,k_{s+1},\dots,k_d)}}} \\ &\leq \frac{\sqrt{\sum_{\mathbf{k}\in[n]^{s-1}} \sum_{\mathbf{j}\in[n]^{d-s}} |\widehat{T}^*_{\mathbf{k}i\mathbf{j}}|^2} \sqrt{\sum_{\mathbf{j}\in[n]^{d-s}} |\lambda_{\mathbf{j}}|^2}}{\sqrt{\sum_{k_1,\dots,k_{s-1},k_{s+1},\dots,k_d\in[n]} |\widehat{T}|^2_{(k_1,\dots,k_{s-1},i,k_{s+1},\dots,k_d)}}} \\ &= \|\lambda\|_2 ,\end{aligned}$$

where in the inequality we have used Cauchy-Schwarz for the sum over $\mathbf{j}$. ◀

Proof of Theorem 15. The inequality $\|\widehat{T}\|_{\frac{2d}{d+1}} \leq \|T\|_{\mathrm{cb}}$ follows from Lemmas 16 and 17. The inequality is saturated by the form $T(x_1,\dots,x_d) = x_1(1)$. ◀

References

1 Scott Aaronson and Andris Ambainis. Forrelation: A problem that optimally separates quantum from classical computing. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 307–316, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2746539.2746547.

2 Srinivasan Arunachalam, Sergey Bravyi, Arkopal Dutt, and Theodore J. Yoder. Optimal Algorithms for Learning Quantum Phase States. In *18th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2023)*, volume 266 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:24, 2023.

3 Srinivasan Arunachalam, Jop Briët, and Carlos Palazuelos. Quantum query algorithms are completely bounded forms. *SIAM J. Comput*, 48(3):903–925, 2019. Preliminary version in ITCS'18.

4 Srinivasan Arunachalam, Sourav Chakraborty, Michal Koucký, Nitin Saurabh, and Ronald De Wolf. Improved bounds on Fourier entropy and min-entropy. *ACM Transactions on Computation Theory (TOCT)*, 13(4):1–40, 2021.

5 Srinivasan Arunachalam, Sourav Chakraborty, Troy Lee, Manaswi Paraashar, and Ronald De Wolf. Two new results about quantum exact learning. *Quantum*, 5:587, 2021.

6 Srinivasan Arunachalam, Arkopal Dutt, Francisco Escudero Gutierrez, and Carlos Palazuelos. Learling low-degree quantum objects. *arXiv preprint*, 2024.

7 Alp Atıcı and Rocco A Servedio. Quantum algorithms for learning and testing juntas. *Quantum Information Processing*, 6(5):323–348, 2007.

8 Nikhil Bansal and Makrand Sinha. k-forrelation optimally separates quantum and classical query complexity. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 1303–1316, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3406325.3451040.

9 Nikhil Bansal, Makrand Sinha, and Ronald de Wolf. Influence in Completely Bounded Block-Multilinear Forms and Classical Simulation of Quantum Algorithms. In *37th Computational Complexity Conference (CCC 2022)*, volume 234, pages 28:1–28:21, 2022. doi:10.4230/LIPIcs.CCC.2022.28.

10 Zongbo Bao and Penghui Yao. On Testing and Learning Quantum Junta Channels. In Gergely Neu and Lorenzo Rosasco, editors, *Proceedings of Thirty Sixth Conference on Learning Theory*, volume 195 of *Proceedings of Machine Learning Research*, pages 1064–1094. PMLR, 12–15 July 2023. URL: https://proceedings.mlr.press/v195/bao23b.html.

11 Frédéric Bayart, Daniel Pellegrino, and Juan B Seoane-Sepúlveda. The Bohr radius of the n-dimensional polydisk is equivalent to $(\log n)/n$. *Advances in Mathematics*, 264:726–746, 2014.

12 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *J. ACM*, 48(4):778–797, 2001.

13 Henri Frédéric Bohnenblust and Einar Hille. On the absolute convergence of Dirichlet series. *Annals of Mathematics*, pages 600–622, 1931.

14 Harald Bohr. Ueber die Bedeutung der Potenzreihen unendlich vieler Variablen in der Theorie der Dirichletschen Reihen. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, 1913:441–488, 1913.

15 Nader H. Bshouty and Jeffrey C. Jackson. Learning DNF over the uniform distribution using a quantum example oracle. In *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, COLT '95, pages 118–127, New York, NY, USA, 1995. Association for Computing Machinery. doi:10.1145/225298.225312.

16 Mark Bun and Justin Thaler. Guest column: Approximate degree in classical and quantum computing. *ACM SIGACT News*, 51(4):48–72, 2021.

17 Clément L Canonne. A short note on learning discrete distributions. *arXiv preprint*, 2020. arXiv:2002.11457.

18 Shantanav Chakraborty, András Gilyén, and Stacey Jeffery. The Power of Block-Encoded Matrix Powers: Improved Regression Techniques via Faster Hamiltonian Simulation. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2019.33.

19 Chi-Fang Chen, Andrew Lucas, and Chao Yin. Speed limits and locality in many-body quantum dynamics. *Reports on Progress in Physics*, 2023.

20 Thomas Chen, Shivam Nadimpalli, and Henry Yuen. Testing and Learning Quantum Juntas Nearly Optimally. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1163–1185. SIAM, 2023.

21 Kai-Min Chung and Han-Hsuan Lin. Sample efficient algorithms for learning quantum channels in pac model and the approximate state discrimination problem. *arXiv preprint*, 2018. arXiv:1810.10938.

22 Andreas Defant, Leonhard Frerick, Joaquim Ortega-Cerdà, Myriam Ounaïes, and Kristian Seip. The Bohnenblust–Hille inequality for homogeneous polynomials is hypercontractive. *Annals of mathematics*, pages 485–497, 2011.

23 Andreas Defant, Mieczysław Mastyło, and Antonio Pérez. On the Fourier spectrum of functions on boolean cubes. *Mathematische Annalen*, 374:653–680, 2019.

24 Diogo Diniz, Gustavo Muñoz-Fernández, Daniel Pellegrino, and J Seoane-Sepúlveda. Lower bounds for the constants in the Bohnenblust–Hille inequality: The case of real scalars. *Proceedings of the American Mathematical Society*, 142(2):575–580, 2014.

25 Francisco Escudero Gutiérrez. Influences of Fourier completely bounded polynomials and classical simulation of quantum algorithms. *arXiv:2304.06713*, 2023.

26 Alexandros Eskenazis and Paata Ivanisvili. Learning low-degree functions from a logarithmic number of random queries. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 203–207, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3519981.

27 Alexandros Eskenazis, Paata Ivanisvili, and Lauritz Streck. Low-degree learning and the metric entropy of polynomials. *arXiv preprint*, 2022. arXiv:2203.09659.

28 Omar Fawzi, Aadil Oufkir, and Daniel Stilck França. Lower bounds on learning Pauli channels. *arXiv preprint*, 2023. arXiv:2301.09192.

29 Steven T Flammia and Ryan O'Donnell. Pauli error estimation via population recovery. *Quantum*, 5:549, 2021.

30 Alexandre Grothendieck. *Résumé de la théorie métrique des produits tensoriels topologiques*. Soc. de Matemática de São Paulo, 1953.

31 Gus Gutoski and Nathaniel Johnston. Process tomography for unitary quantum channels. *Journal of Mathematical Physics*, 55(3), 2014.

32 Jeongwan Haah, Aram W Harrow, Zhengfeng Ji, Xiaodi Wu, and Nengkun Yu. Sample-optimal tomography of quantum states. *IEEE Transactions on Information Theory*, 63(9):5628–5641, 2017.

33 Hsin-Yuan Huang, Sitan Chen, and John Preskill. Learning to predict arbitrary quantum processes. *PRX Quantum*, 4:040337, December 2023. doi:10.1103/PRXQuantum.4.040337.

34 Hsin-Yuan Huang, Richard Kueng, and John Preskill. Predicting many properties of a quantum system from very few measurements. *Nature Physics*, 16(10):1050–1057, 2020.

35 Siddharth Iyer, Anup Rao, Victor Reis, Thomas Rothvoss, and Amir Yehudayoff. Tight bounds on the Fourier growth of bounded functions on the hypercube. *arXiv preprint*, 2021. arXiv:2107.06309.

36 Ohad Klein, Joseph Slote, Alexander Volberg, and Haonan Zhang. Quantum and Classical Low-Degree Learning via a Dimension-Free Remez Inequality. In *15th Innovations in Theoretical Computer Science Conference (ITCS 2024)*, volume 287 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 69:1–69:22, 2024.

37 Hirotada Kobayashi, Keiji Matsumoto, and Tomoyuki Yamakami. Quantum merlin-arthur proof systems: Are multiple merlins more helpful to arthur? In *Algorithms and Computation: 14th International Symposium, ISAAC 2003*, pages 189–198. Springer, 2003.

38 Ching-Yi Lai and Hao-Chung Cheng. Learning quantum circuits of T-depth one. In *2022 IEEE International Symposium on Information Theory (ISIT)*, pages 2213–2218, 2022. doi:10.1109/ISIT50566.2022.9834452.

39 Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, Fourier transform, and learnability. *Journal of the ACM (JACM)*, 40(3):607–620, 1993.

40 John E Littlewood. On bounded bilinear forms in an infinite number of variables. *The Quarterly Journal of Mathematics*, pages 164–174, 1930.

41 Richard A. Low. Learning and testing algorithms for the clifford group. *Phys. Rev. A*, 80:052314, November 2009.

42 Ashley Montanaro. Some applications of hypercontractive inequalities in quantum information theory. *Journal of Mathematical Physics*, 53(12), 2012.

43 Ashley Montanaro and Tobias J Osborne. Quantum Boolean functions. *arXiv preprint*, 2008. arXiv:0810.2435.

44 Shivam Nadimpalli, Natalie Parham, Francisca Vasconcelos, and Henry Yuen. On the Pauli spectrum of QAC0. *arXiv preprint*, 2023. arXiv:2311.09631.

45 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, Cambridge, UK, 2010. doi:10.1017/CBO9780511976667.

46 Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

47 Ryan O'Donnell and John Wright. Efficient quantum tomography. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 899–912, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2897518.2897544.

48 Daniel Pellegrino and Eduardo V Teixeira. Towards sharp Bohnenblust–Hille constants. *Communications in Contemporary Mathematics*, 20(03):1750029, 2018.

49 David Pérez-García, Michael M Wolf, Carlos Palazuelos, Ignacio Villanueva, and Marius Junge. Unbounded violation of tripartite Bell inequalities. *Communications in Mathematical Physics*, 279:455–486, 2008.

50 Joseph Slote, Alexander Volberg, and Haonan Zhang. Bohnenblust–Hille inequality for cyclic groups. *arXiv preprint*, 2023. arXiv:2305.10560.

51 Joseph Slote, Alexander Volberg, and Haonan Zhang. Noncommutative bohnenblust-hille inequality in the heisenberg-weyl and gell-mann bases with applications to fast learning. *arXiv preprint*, 2023. arXiv:2301.01438.

52 Barbara M. Terhal. Quantum error correction for quantum memories. *Rev. Mod. Phys.*, 87:307–346, April 2015.

53 Minh C Tran, Kunal Sharma, and Kristan Temme. Locality and error mitigation of quantum circuits. *arXiv preprint*, 2023. arXiv:2303.06496.

54 Leslie G. Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

55 Ewout Van Den Berg, Zlatko K Minev, Abhinav Kandala, and Kristan Temme. Probabilistic error cancellation with sparse Pauli–Lindblad models on noisy quantum processors. *Nature Physics*, pages 1–6, 2023.

56 N. Th. Varopoulos. On an inequality of von Neumann and an application of the metric theory of tensor products to operators theory. *J. Functional Analysis*, 16:83–100, 1974. doi:10.1016/0022-1236(74)90071-8.

57 Alexander Volberg and Haonan Zhang. Noncommutative Bohnenblust–Hille inequalities. *Mathematische Annalen*, pages 1–20, 2023.

A Multivariate to Bivariate Reduction for Noncommutative Rank and Related Results

Vikraman Arvind
The Institute of Mathematical Sciences (HBNI), Chennai, India
Chennai Mathematical Institute, Siruseri, Kelambakkam, India

Pushkar S. Joglekar
Vishwakarma Institute of Technology, Pune, India

Abstract

We study the *noncommutative rank* problem, ncRANK, of computing the rank of matrices with linear entries in n noncommuting variables and the problem of *noncommutative Rational Identity Testing*, RIT, which is to decide if a given rational formula in n noncommuting variables is zero on its domain of definition.

Motivated by the question whether these problems have *deterministic* NC algorithms, we revisit their interrelationship from a parallel complexity point of view. We show the following results:

1. Based on Cohn's embedding theorem [14, 15] we show deterministic NC reductions from multivariate ncRANK to bivariate ncRANK and from multivariate RIT to bivariate RIT.
2. We obtain a deterministic NC-Turing reduction from bivariate RIT to bivariate ncRANK, thereby proving that a deterministic NC algorithm for bivariate ncRANK would imply that both multivariate RIT and multivariate ncRANK are in deterministic NC.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases noncommutative rank, rational formulas, identity testing, parallel complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.14

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: http://arxiv.org/abs/2404.16382 [4]

Acknowledgements We are grateful to the anonymous referees for their valuable comments and suggestions.

1 Introduction

There are two main algorithmic problems of interest in this paper. These are the *noncommutative Rational Identity Testing problem* (RIT) and the *noncommutative rank* (ncRANK) problem for matrices with linear entries.

The RIT problem is a generalization of multivariate polynomial identity testing to identity testing of multivariate rational expressions. When the variables are commuting, rational identity testing and polynomial identity testing are equivalent problems. On the other hand, if the variables are all noncommuting, the RIT problem needs different algorithmic techniques as rational expressions in noncommuting variables are more complicated. Mathematically, rational expressions over noncommuting variables are quite well studied. They arise in the construction of the so-called free skew fields [15]. Hrubes and Wigderson [21] initiated the algorithmic study of RIT for *rational formulas* and gave a deterministic polynomial time reduction from RIT to ncRANK. Subsequently, deterministic polynomial-time algorithms were obtained independently by Ivanyos et al [23, 22] and by Garg et al [18, 19] for the RIT problem, in fact they obtain deterministic polynomial time algorithms for ncRANK, and

using Hrubes-Wigderson reduction from RIT to ncRANK get a polynomial time algorithm for RIT. The Ivanyos et al algorithm is algebraic and works for fields of all characteristics. The Garg et al algorithm has an analytic flavor and is for the characteristic zero case.

The Edmonds' Problem and ncRANK

The ncRANK problem is essentially the noncommutative version of the well-known Edmonds' problem: determine the rank of a matrix M whose entries are linear forms in commuting variables (see [23, 19, 7, 10] for more details). A special case of it is to determine if a square matrix M with linear entries in commuting variables is *singular*. This is also known as the symbolic determinant identity testing problem, SDIT. There is an easy randomized NC algorithm for it, based on the Polynomial Identity Lemma [5, 28, 30, 16], by randomly substituting scalar values for the variables from the field (or a suitable extension of it) and evaluating the determinant using a standard NC algorithm. However, a deterministic polynomial-time algorithm for SDIT is an outstanding open problem [7].

Recently, for the RIT problem the first deterministic quasi-NC algorithm has been obtained [2]. Another recent development – building on the connection between the noncommutative Edmonds' problem and identity testing for noncommutative algebraic branching programs [10] – is a generalization of the Edmond's problem to a partially commutative setting with application to the weighted k-tape automata equivalence problem [3].

1.1 This paper: overview of results and proofs

With this background, the natural algorithmic questions are whether RIT for noncommutative rational formulas and ncRANK have deterministic NC algorithms. We revisit the problems from this perspective and obtain the following new results.

1. We show that multivariate RIT for formulas is deterministic NC reducible to bivariate RIT for formulas. More precisely, given a rational formula $\Phi(x_1, x_2, \ldots, x_n)$, computing an element of the skew field $\mathbb{F}\langle\langle X\rangle\rangle$, where $X = \{x_1, x_2, \ldots, x_n\}$, the deterministic NC reduction replaces each x_i by a formula $\Phi_i(x, y)$ computing a polynomial in $\mathbb{F}\langle x, y\rangle$. Then the resulting rational formula

   $$\Psi(x, y) = \Phi(\Phi_1(x, y), \Phi_2(x, y), \ldots, \Phi_n(x, y))$$

   has the property that

   $$\Phi(x_1, x_2, \ldots, x_n) \neq 0 \text{ iff } \Psi(x, y) \neq 0.$$

2. We next show that multivariate ncRANK is deterministic NC reducible to bivariate ncRANK. More precisely, given a $d \times d$ linear matrix $A = A_0 + \sum_{i=1}^{n} A_i x_i$ in noncommuting variables $X = \{x_1, x_2, \ldots, x_n\}$, where the A_i are matrices over the scalar field $\mathbb{F}$, we first give a deterministic NC reduction that transforms A to a $d \times d$ matrix B whose entries are bivariate polynomials in $\mathbb{F}\langle x, y\rangle$, where x and y are two noncommuting variables, where its entries $B[i, j]$ are given by *polynomial size noncommutative formulas*, with the property that $\text{ncrk}(A) = \text{ncrk}(B)$. Then we examine the Higman linearization process [21] that transforms B into a matrix B' with linear entries in x and y such that the noncommutative rank $\text{ncrk}(B)$ of B can be easily recovered from $\text{ncrk}(B')$. We show that this process can be implemented in deterministic NC (the earlier works [21, 22, 23, 19] only consider its polynomial-time computability).
   Additionally, we consider the more general problem ncRANK_{poly} of computing the noncommutative rank of a matrix whose entries are noncommutative formulas computing polynomials. We show using our parallel Higman linearization algorithm that ncRANK_{poly} is also deterministic NC reducible to bivariate ncRANK.

Both the multivariate to bivariate reductions, stated above, are crucially based on a theorem of Cohn [14] (also see [15, Theorem 4.7.3]) which we will refer to as Cohn's embedding theorem and describe it later in the introduction.

3. Finally, obtaining a deterministic NC reduction from RIT to ncRANK turns out to be quite subtle. From the work of Hrubes and Wigderson [21], who initiated this line of research on RIT, we can only obtain a sequential deterministic polynomial-time reduction from RIT to ncRANK. However, for our result we require an NC reduction. If the given rational formula has logarithmic depth, then their result already implies an NC reduction. Now, in the same paper [21], Hrubes and Wigderson have also shown a *depth reduction* result for multivariate noncommutative rational formulas: every rational formulas of size s is equivalent to a logarithmic depth rational formula of size $\text{poly}(s)$. Their construction is based on Brent's depth reduction result for commutative arithmetic formulas. However, due to noncommutativity and the presence of inversion gates, the formula constructed in their proof needs to be different based on whether certain rational subformulas, arising in the construction procedure, are identically zero or not. To algorithmize such steps in the construction we need to use RIT as a subroutine. As RIT has a polynomial-time algorithm [23, 19], the depth-reduction in [21] also has a polynomial time algorithm.[1]
   As the third result of this paper, building on the Hrubes-Wigderson depth-reduction construction, we are able to show that, with oracle access to RIT, rational formula depth reduction can be done in deterministic NC. Using this we are able to obtain a deterministic NC-Turing reduction from RIT to ncRANK. Hence, if bivariate ncRANK is in deterministic NC we will obtain a deterministic NC algorithm also for RIT. We leave open the question whether depth reduction of noncommutative rational formulas is unconditionally in NC.

2 Preliminaries

In this section we recall the essential basic definitions and fix the notation.

Let $\mathbb{F}$ be a (commutative) field[2] and $X = \{x_1, x_2, \ldots, x_n\}$ be n free noncommuting variables. The free monoid X^* is the set of all monomials in the variables X. A *noncommutative polynomial* $f(X)$ is a finite $\mathbb{F}$-linear combination of monomials in X^*, and the *free noncommutative ring* $\mathbb{F}\langle X\rangle$ consists of all noncommutative polynomials.

Noncommutative Rational Formulas

An *arithmetic circuit* computing an element of $\mathbb{F}\langle X\rangle$ is a directed acyclic graph with each indegree 0 node labeled by either an input variable $x_i \in X$ or some scalar $c \in \mathbb{F}$. Each internal node g has indegree 2 and is either a $+$ gate or a $\times$ gate: it computes the sum (resp. left to right product) of its inputs. Thus, each gate of the circuit computes a polynomial in $\mathbb{F}\langle X\rangle$ and the polynomial computed by the circuit is the polynomial computed at the *output gate*. A *formula* is restricted to have fanout 1 or 0.

[1] In the commutative case, Brent's result is parallelizable to yield an NC algorithm. For noncommutative formulas without inversion gates we can obtain the depth-reduced formula in NC, as we will observe later in the paper.

[2] In this paper, $\mathbb{F}$ will either be the field of rationals or a finite field.

When we allow the formulas/circuits to have *inversion gates* we get *rational formulas and rational circuits*.

The Free Skew Field

We now briefly explain the free skew field construction. The elements of the free skew field are noncommutative rational functions which are more complicated than their commutative counterparts. Rational formulas in the commutative setting can be canonically expressed as ratios of two polynomials. There is no such canonical representation for noncommutative rational formulas.

Following Hrubes-Wigderson [21], we use Amitsur's approach [1] for formally defining skew fields.[3]

It involves defining appropriate notion of equivalence of formulas (intuitively, two formulas are equivalent if they agree on their *domain of definition*). The equivalence classes under this equivalence relation are the elements of the free skew field. We give the formal definitions below.

Let $\mathbb{M}_k(\mathbb{F})$ denote the ring of $k \times k$ matrices with entries from field $\mathbb{F}$. Note that a rational formula Φ defines a partial function

$$\hat{\Phi} : \mathbb{M}_k(\mathbb{F})^n \mapsto \mathbb{M}_k(\mathbb{F})$$

that on input $(a_1, a_2, \ldots, a_n) \in \mathbb{M}_k(\mathbb{F})^n$ evaluates Φ by substituting $x_i \leftarrow a_i$ for $i \in [n]$. $\hat{\Phi}(a_1, \ldots, a_n)$ is undefined if the input to some inversion gate in Φ is not invertible in $\mathbb{M}_k(\mathbb{F})$.

▶ **Definition 1.** *Let Φ be a rational formula in variables X. For each $k \in \mathbb{N}$, let $\mathcal{D}_{k,\Phi}$ be the set of all matrix tuples $(a_1, a_2, \ldots, a_n) \in \mathbb{M}_k(\mathbb{F})^n$ such that $\hat{\Phi}(a_1, a_2, \ldots, a_n)$ is defined. The* domain of definition *of Φ is the union $\mathcal{D}_\Phi = \bigcup_k \mathcal{D}_{k,\Phi}$.*

▶ **Definition 2** ([21])**.**

- *A rational formula Φ is called* correct *if for every gate u of Φ the subformula Φ_u has a nonempty domain of definition.*
- *Correct rational formulas Φ_1, Φ_2 are said to be* equivalent *(denoted $\Phi_1 \equiv \Phi_2$) if the intersection $\mathcal{D}_{\Phi_1} \cap \mathcal{D}_{\Phi_2}$ of their domains of definitions is nonempty and they agree on all the points in the intersection.*

We note that equivalent formulas need not have the same domain of definition. For example, $\Phi_1 = z_1 z_2 z_3$ and $\Phi_2 = (z_1 z_2 z_3 \cdot (z_2 z_3 - z_3 z_2)^{-1}) \cdot (z_2 z_3 - z_3 z_2)$ are equivalent. However, the domain of definition of Φ_1 includes all matrix tuples, whereas the domain of definition of Φ_2 contains only matrix tuples (Z_1, Z_2, Z_3) such that $\det(Z_2 Z_3 - Z_3 Z_2) \neq 0$.

The relation $\equiv$ as defined above is an equivalence relation on rational formulas and the equivalence classes, called *rational functions*, are the elements of the skew field $\mathbb{F}\langle\langle X \rangle\rangle$ [21, 1].

Noncommutative Rank

We now recall the notion of rank for matrices over the noncommutative ring $\mathbb{F}\langle X \rangle$.

▶ **Definition 3** (inner rank)**.** *Let M be a matrix over $\mathbb{F}\langle X \rangle$. Its* inner rank *is the least r such that M can be written as a matrix product $M = PQ$ where Q has r rows (and P has r columns).*

[3] There are other ways to defining free skew fields [1, 6, 27, 11, 12, 13, 15].

▶ **Definition 4** (full matrices). *An $n \times n$ square matrix M over $\mathbb{F}\langle X\rangle$ is* full *if it cannot be decomposed as a matrix product $M = PQ$ where P is $n \times r$ and Q is $r \times n$ for $r < n$. In other words, an $n \times n$ matrix is called full precisely when its inner rank is n.*

We can also define the rank of a matrix M to be the maximum r such that M contains an $r \times r$ full submatrix. For matrices over $\mathbb{F}\langle X\rangle$ these notions of *noncommutative rank* coincide as summarized below.[4]

▶ **Proposition 5** ([15]). *Let M be an $n \times n$ matrix over the ring $\mathbb{F}\langle X\rangle$. Then*

- *M is a full matrix (that is, M has inner rank n) iff it is invertible over the skew field $\mathbb{F}\langle\langle X\rangle\rangle$.*
- *More generally, M has inner rank r iff the largest full submatrix of M is $r \times r$.*

The Algorithmic Problems of Interest

At this point we formally define the problems of interest in this paper.

1. The multivariate RIT problem takes as input a rational formula Φ, computing a rational function $\hat{\Phi}$ in $\mathbb{F}\langle\langle X\rangle\rangle$, and the problem is to check if Φ is equivalent to 0? In the bivariate RIT problem Φ computes a rational function in $\mathbb{F}\langle\langle x, y\rangle\rangle$.
2. The multivariate ncRANK problem takes as input a matrix M with affine linear form entries over X and the problem is to determine its noncommutative rank $\mathrm{ncrk}(M)$. Bivariate ncRANK is similarly defined.
3. A more general version of ncRANK is ncRANK_{poly} in which the matrix entries are allowed to be polynomials in $\mathbb{F}\langle X\rangle$ computed by noncommutative formulas. A closely related problem is SINGULAR where the problem is to test if a square matrix M over $\mathbb{F}\langle X\rangle$ with entries computed by formulas is singular or not.

The complexity class NC and NC reductions

The class NC consists of decision problems that can be solved in $\mathrm{polylog}(n)$ time with $\mathrm{poly}(n)$ many processors.[5] For two decision problems A and B we say A is many-one NC reducible to B if there is a reduction from A to B that is NC computable. Similarly, A is NC-Turing reducible to B if there is an oracle NC algorithm for A that has oracle access to B.

It turns out that SINGULAR and ncRANK are equivalent even under deterministic NC reductions.[6]

Cohn's Embedding Theorem

We now give an outline of Cohn's embedding theorem and how it gives us the desired reduction from multivariate to bivariate RIT and also from multivariate to bivariate ncRANK. However, for multivariate to bivariate reduction for ncRANK we will require additional NC algorithms for formula depth reduction and Higman linearization.

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of n noncommuting variables, and let x, y be a pair of noncommuting variables. We first recall the following well-known fact, observed in the early papers on noncommutative polynomial identity testing [8, 26]: for noncommutative

[4] For a ring R in general, a full matrix R need not be invertible (see [21] for an example).
[5] This model is widely accepted as the right theoretical notion for efficient parallel algorithms.
[6] As for $M \in \mathbb{F}\langle X\rangle^{m\times n}$, $\mathrm{ncrk}(M) = r$ iff r is a size of a largest sized invertible minor of M, so to compute $\mathrm{ncrk}(M)$, it suffices to test singularity of matrix UMV, where U, V are generic $r \times m, n \times r$ matrices respectively with entries as fresh noncommuting variables for $r \leq \min(m, n)$. See e.g. [19, Lemma A.3] for details.

polynomials in $\mathbb{F}\langle X\rangle$, the problem of polynomial identity testing (PIT) is easily reducible to PIT for bivariate noncommutative polynomials in $\mathbb{F}\langle x, y\rangle$. Indeed, more formally, we have the following easy to check fact.

▶ **Proposition 6.** *The map*

$$x_i \mapsto x^{i-1}y, 1 \leq i \leq n$$

extends to an injective homomorphism (i.e. a homomorphic embedding*) from the ring* $\mathbb{F}\langle X\rangle$ *to the ring* $\mathbb{F}\langle x, y\rangle$*.*

However, in order to obtain our multivariate to bivariate reductions, we need a mapping $\beta : X \to \mathbb{F}\langle x, y\rangle$ which has the following properties:

- For each i, there is a small noncommutative arithmetic formula that computes $\beta(x_i)$.
- β extends to an injective homomorphism[7], not just from the ring $\mathbb{F}\langle X\rangle$ to $\mathbb{F}\langle x, y\rangle$, but also to an injective homomorphism from the skew field $\mathbb{F}\langle\langle X\rangle\rangle$ to the skew field $\mathbb{F}\langle\langle x, y\rangle\rangle$. This will guarantee that for two rational formulas Φ_1, Φ_2 computing inequivalent rational functions in $\mathbb{F}\langle\langle X\rangle\rangle$ their images $\beta(\Phi_1)$ and $\beta(\Phi_2)$ also compute inequivalent rational functions in $\mathbb{F}\langle x, y\rangle$.
- Furthermore, in order to get the multivariate to bivariate reduction for ncRANK, we will additionally require of the map β that for any matrix M over $\mathbb{F}\langle X\rangle$ its image $\beta(M)$, which is a matrix over $\mathbb{F}\langle x, y\rangle$ obtained by applying β to each entry of M, has the same rank as M. Such a homomorphic embedding is called an *honest embedding* [14]. Here we note that, full matrices over $\mathbb{F}\langle X\rangle$ are invertible over $\mathbb{F}\langle\langle X\rangle\rangle$ [15]. Consequently if one can lift embedding β to one between the corresponding free skew fields, it enforces β to be an honest embedding.

The mapping $x_i \mapsto x^{i-1}y$ actually *does not* extend to an honest embedding as observed in [14]. Indeed, the rank 2 matrix $\begin{pmatrix} x_1 & x_2 \\ x_3 & x_4 \end{pmatrix}$ has image

$$\begin{pmatrix} y & xy \\ x^2y & x^3y \end{pmatrix} = \begin{pmatrix} 1 \\ x^2 \end{pmatrix} \begin{pmatrix} y & xy \end{pmatrix}$$

which is rank 1. In general, a homomorphic embedding from a ring R to a ring R' is an *honest embedding* if it maps full matrices over R to full matrices over R'. We now state Cohn's embedding theorem.

For polynomials $f, g \in \mathbb{F}\langle x, y\rangle$ let $[f, g]$ denotes the commutator polynomial $fg - gf$. Cohn's embedding map $\beta : \mathbb{F}\langle X\rangle \to \mathbb{F}\langle x, y\rangle$ is defined as follows.

- Let $\beta(x_1) = y$. For $i \geq 2$, define $\beta(x_i) = [\beta(x_{i-1}), x]$.
- We can then naturally extend β to a homomorphism from $\mathbb{F}\langle X\rangle$ to $\mathbb{F}\langle y, x\rangle$, and it is easy to check that it is injective. In fact, we can even assume $|X|$ to be countably infinite.

▶ **Theorem 7** (Cohn's embedding theorem [15, Theorem 7.5.19])**.** *The embedding map* $\beta : \mathbb{F}\langle X\rangle \to \mathbb{F}\langle x, y\rangle$ *defined above extends to an embedding between the corresponding skew fields* $\beta : \mathbb{F}\langle\langle Z\rangle\rangle \to \mathbb{F}\langle\langle x, y\rangle\rangle$ *and hence is an honest embedding.*

Cohn's construction is based on skew polynomial rings, which explains the appearance of the iterated commutators $\beta(x_i) = [\beta(x_{i-1}), x]$. We briefly explain the underlying ideas in the arxiv version [4]. For more details see [14, 15].

[7] That is, a homomorphic embedding.

3 The Reduction from multivariate RIT to bivariate RIT

The reduction follows quite easily from Theorem 7. However, we present some complexity details in this section showing that it is actually a deterministic NC reduction. The following lemma is useful to describe the reduction.

▶ **Lemma 8.** *Recall the embedding map β defined above. $\beta(z_0) = y$ and $\beta(z_{i+1}) = [\beta(z_i), x]$ are polynomials in $\mathbb{F}\langle x, y\rangle$ for each $i \geq 0$. Then, for $n \geq 1$ we have*

$$\beta(z_n) = \sum_{i=0}^{n} (-1)^i \binom{n}{i} x^i y x^{n-i}.$$

As a consequence, there is a deterministic NC *algorithm that constructs a* poly(n)*-sized formula for $\beta(z_n)$.*

Proof. We will use induction on n. The base case follows from the fact that $\beta(z_1) = yx - xy$. Inductively assume the claim is true for all n. Now, $\beta(z_{n+1}) = [\beta(z_n), x]$

$$\begin{aligned}
&= \sum_{i=0}^{n} (-1)^i \binom{n}{i} x^i y x^{n-i+1} - \sum_{j=0}^{n} (-1)^j \binom{n}{j} x^{j+1} y x^{n-j} \text{ by induction hypothesis} \\
&= yx^{n+1} + \sum_{i=1}^{n} (-1)^i \binom{n}{i} x^i y x^{n-i+1} + \sum_{i=1}^{n+1} (-1)^i \binom{n+1-1}{i-1} x^i y x^{n+1-i} \\
&= yx^{n+1} + \sum_{i=1}^{n} (-1)^i \left[\binom{n+1-1}{i} + \binom{n+1-1}{i-1} \right] x^i y x^{n-i+1} + (-1)^{n+1} x^{n+1} y \\
&= yx^{n+1} + (-1)^{n+1} x^{n+1} y + \sum_{i=1}^{n} (-1)^i \binom{n+1}{i} x^i y x^{n+1-i} \text{ from Pascal's identity} \\
&= \sum_{i=0}^{n+1} (-1)^i \binom{n+1}{i} x^i y x^{n+1-i}
\end{aligned}$$

This completes the inductive proof.

As the binomial coefficients can be computed in NC using Pascal's identity, the expression for $\beta(z_n)$ obtained above immediately implies an NC algorithm for construction of a poly(n) sized formula for $\beta(z_n)$. ◀

▶ **Theorem 9.** *The multivariate RIT problem is deterministic* NC *(in fact, logspace) reducible to bivariate RIT. More precisely, given as input a rational formula Φ computing an element of $\mathbb{F}\langle\langle X\rangle\rangle$, $X = \{x_1, x_2, \ldots, x_n\}$ there is a deterministic* NC *algorithm that computes a rational formula Ψ computing an element of $\mathbb{F}\langle\langle x, y\rangle\rangle$ such that Φ is nonzero in its domain of definition iff Ψ is nonzero in its domain of definition.*

Proof. We can identify $\mathbb{F}\langle\langle X\rangle\rangle$ with $\mathbb{F}\langle\langle z_0, z_1, \ldots, z_{n-1}\rangle\rangle$. Let $\Phi_i(x, y)$ be the poly(i) size noncommutative formula computing the nested commutator $\beta(z_i)$ for each i. In the rational formula Φ, for each i we replace the input z_i to Φ by $\Phi_i(x, y)$. The new formula we obtain is

$$\Psi(x, y) = \Phi(\Phi_1(x, y), \Phi_2(x, y), \ldots, \Phi_{n-1}(x, y)).$$

By Theorem 7, $\Psi(x, y) \neq 0$ on its domain of definition iff $\Phi(z_0, z_1, \ldots, z_{n-1})$ is nonzero on its domain of definition. Furthermore, because β is an embedding, it is guaranteed that if Φ has a nontrivial domain of definition then Ψ also has a nontrivial domain of definition.

As computation of Ψ from Φ involves only replacing the z_i by Φ_i, the reduction is clearly logspace computable. ◀

4 Reduction from n-variate ncRANK_{poly} to 2-variate ncRANK

In this section we give a deterministic NC reduction from n-variate ncRANK_{poly} to bivariate ncRANK. The basic idea of the reduction is as follows. Given a polynomial matrix[8] $M \in \mathbb{F}\langle X\rangle^{m\times m}$ such that each entry of M is computed by formula of size at most s. We will use Cohn's embedding theorem 7 to get a matrix M_1 with *bivariate* polynomial entries such that each entry of M_1 is computed by a $\mathrm{poly}(n,s)$ size noncommutative formula and $\mathrm{ncrk}(M) = \mathrm{ncrk}(M_1)$. Notice that M_1 is an instance of bivariate ncRANK_{poly}. Next, we need to give an NC reduction transforming M_1 to an instance of bivariate ncRANK (which will be a matrix with linear entries in x and y).

In order to do this transformation in NC, we will first apply the depth-reduction algorithm of Lemma 10 to get matrix M_2 whose entries are $\mathrm{poly}(n,s)$ size log-depth formulas that compute the same polynomials as the corresponding entries of M_1. Then we apply Higman Linearization to M_2 to obtain a bivariate linear matrix M_3. For this we will use our parallel algorithm for Higman Linearization described in Theorem 13. From the properties of Higman linearization we can easily recover $\mathrm{ncrk}(M_2)$ from $\mathrm{ncrk}(M_3)$. In what follows, first we give a deterministic NC algorithm for the depth reduction of noncommutative formulas and Higman linearization process. We conclude the section by giving an NC reduction from multivariate to bivariate ncRANK using above NC algorithms combined with Cohn's embedding theorem 7.

4.1 Depth reduction for noncommutative formulas without divisions

In the commutative setting Brent [9] obtained a deterministic NC algorithm to transform a given *rational* formula (which may have division gates) to a log-depth rational formula. In the noncommutative setting, Hrubes and Wigderson [21] proved the *existence* of log-depth *rational* formula equivalent to any given rational formula. Their proof is based on [9]. However, it is not directly algorithmic as explained in the introduction. We will discuss it in more detail in Section 5. However, it turns out that, if the noncommutative formula doesn't have division gates then the depth reduction is quite easy and we obtain a simple deterministic NC algorithm for it that computes a log-depth noncommutative formula equivalent to the given noncommutative formula. The proof is based Brent's commutative version. We highlight the distinctive points arising in the noncommutative version.

We introducing some notation. Let Φ be a noncommutative arithmetic formula computing a polynomial in $\mathbb{F}\langle x_1, x_2, \ldots, x_n\rangle$. Let $\hat{\Phi}$ denote the polynomial computed by Φ. For a node $v \in \Phi$, let Φ_v denote the subformula of Φ rooted at node v, so $\hat{\Phi_v}$ is the polynomial computed by the subformula rooted at v. For a node $v \in \Phi$, let $\Phi_{v\leftarrow z}$ be a formula obtained from Φ by replacing the sub-formula Φ_v by single variable z. For a node $v \in \Phi$, let $wt(v) = |\Phi_v|$ denote the number of nodes in the subformula rooted at v. By size of formula Φ we refer to number of gates in Φ.

▶ **Lemma 10.** *Given a formula Φ of size s computing a noncommutative polynomial $f \in \mathbb{F}\langle x_1, x_2, \ldots, x_n\rangle$ there is an NC algorithm to obtain an equivalent formula Φ' for f with depth $O(\log s)$.*

Proof. First we describe a recursive construction to compute a formula Φ' equivalent to Φ and inductively prove that the depth of Φ' is $c\log_2 s$ for an absolute constant c. Then we analyze the parallel time complexity of the construction and prove that it can be implemented in NC.

[8] We can assume it is a square matrix without loss of generality.

Let A_Φ be $s \times s$ matrix such that for gates $u, v \in \Phi$, $(u,v)^{th}$ entry of A_Φ is 1 if gate v is a descendent of gate u. Using the well-known pointer doubling strategy (see e.g. [29, 25]) we can compute matrix A_Φ in NC. So by adding elements in each row of A_Φ, we can compute $wt(u)$ (that is the number of descendants of gate $u \in \Phi$) in NC. Let v be a gate in Φ such that $\frac{s}{3} \leq wt(v) < \frac{2s}{3}$. Such a gate always exists by a standard argument. Since we can compute the number of descendants of a gate in NC, we can also find such a gate v in NC, by simply having a processor associated to each gate to check the above inequalities. Now we are ready to describe recursive construction of Φ'.

1. In NC find a gate v in Φ such that $\frac{s}{3} \leq wt(v) < \frac{2s}{3}$.
2. Let $r = v_0$ be the root of Φ and $v_1, v_2, \ldots, v_{\ell-1}$ be gates on r to v path in Φ. Let $v = v_\ell$. For $1 \leq i \leq \ell$, let u_i denote a sibling of v_i. Let S_1 be collection of all indices j such that $1 \leq j \leq \ell$, v_j is a product gate and is a right child of its parent. Similarly let S_2 be collection of all indices j such that $1 \leq j \leq \ell$, v_j is a product gate and is a left child of its parent. Define formula $\Psi_1 = \prod_{j \in S_1} \Phi_{u_j}$. The product is computed using sequence of multiplication gates, starting with Φ_{u_j} for the first u_j (one with smallest index $j \in S_1$) each multiplication gate multiplies the product so far from right by Φ_{u_j} for the next gate u_j, $j \in S_1$, along the root to v path. Similarly define formula $\Psi_2 = \prod_{j \in S_2} \Phi_{u_j}$. Let Ψ_3 be a formula obtained from Φ by replacing subformula Φ_v by zero.
3. Recursively in parallel compute log-depth formulas $\Psi_1', \Psi_2', \Psi_3', \Phi_v'$ equivalent to Ψ_1, Ψ_2, Ψ_3 and Φ_v respectively.
4. Define formula Φ' as $(\Psi_1' \cdot \Phi_v') \cdot \Psi_2' + \Psi_3$.

From the definitions of Ψ_1, Ψ_2 and Ψ_3 it is clear that the polynomial computed by Φ equals $(\hat{\Psi}_1 \cdot \hat{\Phi}_v) \cdot \hat{\Psi}_2 + \hat{\Psi}_3$, where $\hat{\Psi}_1, \hat{\Psi}_1, \hat{\Psi}_1, \hat{\Phi}_v$ are the polynomials computed by Ψ_1, Ψ_2, Ψ_3 and Φ_v respectively. Hence, Φ', defined in Step 4, is equivalent to Φ.

Let $d(s)$ denote the upper bound on the depth of the formula output by the above procedure if size s formula is given to it as input. We use induction on the size s to prove that $d(s) \leq c \log_2 s$. As Ψ_1, Ψ_2 are disjoint subformulas of $\Phi_{v \leftarrow z}$, clearly we have $|\Psi_1| + |\Psi_2| \leq |\Phi_{v \leftarrow z}|$. Since $|\Phi_v| \geq \frac{s}{3}$, it implies $|\Psi_1|, |\Psi_2| \leq |\Phi_{v \leftarrow z}| \leq \frac{2s}{3}$. From the definition of Ψ_3, it is clear that $|\Psi_3| \leq |\Phi_{v \leftarrow z}| \leq \frac{2s}{3}$. So the size of each formula Ψ_1, Ψ_2, Ψ_3 and Φ_v is upper bounded by $\frac{2s}{3}$. Hence, inductively, for each of the formulas $\Psi_1', \Psi_2', \Psi_3', \Phi_v'$ the depth is upper bounded by $c \log_2 \frac{2s}{3}$. As Φ' is obtained from $\Psi_1', \Psi_2', \Psi_3', \Phi_v'$ using two multiplications and an addition as in Step 4, it follows that the depth of $\Phi' = d(s) \leq c \log_2 \frac{2s}{3} + 3$. Choosing $c \geq \frac{3}{(\log_2 3 - 1)}$ we get $d(s) \leq c \log_2 \frac{2s}{3} + 3 \leq c \log_2 s$. This completes the induction, proving that the depth of Φ' is at most $c \log_2 s$.

Let $t(s)$ denotes parallel time complexity of the above procedure. Since Steps 1, 2, 4 can be implemented in NC they together take $(\log s)^k$ parallel time for an absolute constant k. As all the recursive calls in Step 3 are processed in parallel, we have the recurrence $t(s) \leq t(2s/3) + (\log s)^k$. Hence, $t(s) \leq (\log s)^{(k+1)}$. This shows that the above procedure can be implemented in NC, completing the proof of the theorem. ◀

▶ Remark 11. In Lemma 10, we avoid using the depth-reduction approach for noncommutative rational formulas that is used in [21]. This is because it can introduce inversion gates even if the original formula has no inversion gates. Instead, we directly adapt ideas from Brent's construction for commutative formulas to obtain the NC algorithm. We note here that Nisan's seminal work [24] also briefly mentions noncommutative formula depth reduction (but not its parallel complexity or even in an algorithmic context).

Higman linearization which is sometimes called Higman's trick was first used by Higman in [20]. Cohn extensively used Higman Linearization in his factorization theory of free ideal rings. Given a matrix with noncommutative polynomials as its entries, Higman linearization

process transforms it into a matrix with linear entries. This transformation process has several nice properties such as: it preserves fullness of the matrix (that is the input polynomial matrix is full rank iff final linear matrix is full rank), it preserves irreducibility of the matrix, etc.

We first describe a single step of the linearization process applied to a single noncommutative polynomial, which easily generalizes to matrices with polynomial entries. Given an $m \times m$ matrix M over $\mathbb{F}\langle X\rangle$ such that $M[m,m] = f + g \times h$, apply the following:

- Expand M to an $(m+1) \times (m+1)$ matrix by adding a new last row and last column with diagonal entry 1 and remaining new entries zero:

$$\left[\begin{array}{c|c} M & 0 \\ \hline 0 & 1 \end{array}\right].$$

- Then the bottom right 2×2 submatrix is transformed as follows by elementary row and column operations

$$\begin{pmatrix} f+gh & 0 \\ 0 & 1 \end{pmatrix} \to \begin{pmatrix} f+gh & g \\ 0 & 1 \end{pmatrix} \to \begin{pmatrix} f & g \\ -h & 1 \end{pmatrix}$$

Given a polynomial $f \in \mathbb{F}\langle X\rangle$ by repeated application of the above step we will finally obtain a *linear matrix* $L = A_0 + \sum_{i=1}^{n} A_i x_i$, where each $A_i, 0 \le i \le n$ is an $\ell \times \ell$ over $\mathbb{F}$, for some ℓ. The following theorem summarizes its properties.

▶ **Theorem 12** (Higman Linearization [15]). *Given a polynomial $f \in \mathbb{F}\langle X\rangle$, there are matrices $P, Q \in \mathbb{F}\langle X\rangle^{\ell\times\ell}$ and a linear matrix $L \in \mathbb{F}\langle X\rangle^{\ell\times\ell}$ such that*

$$P \left(\begin{array}{c|c} f & 0 \\ \hline 0 & I_{\ell-1} \end{array}\right) Q = L \tag{1}$$

with P upper triangular, Q lower triangular, and the diagonal entries of both P and Q are all 1's. (Hence, P and Q are both invertible over $\mathbb{F}\langle\langle X\rangle\rangle$, moreover entries of P^{-1} and Q^{-1} are in $\mathbb{F}\langle X\rangle$).

Instead of a single f, we can apply Higman linearization to a matrix of polynomials $M \in \mathbb{F}\langle X\rangle^{m\times m}$ to obtain a linear matrix L such that $P(M \oplus I_k)Q = L$ for invertible matrices P, Q. Garg et al. [19] gave polynomial time algorithm to carry out Higman linearization for polynomial matrix whose entries are given by noncommutative formulas. We will give an NC algorithm to implement this transformation.

▶ **Theorem 13.** *Let $A \in \mathbb{F}\langle X\rangle^{n\times n}$ be a polynomial matrix such that each entry of A is given by a noncommutative formula of size at most s. Then there is a deterministic* NC *algorithm (with parallel time complexity* $\mathrm{poly}(\log s, \log n)$*) to compute invertible upper and lower triangular matrices $P, Q \in \mathbb{F}\langle X\rangle^{\ell\times\ell}$ with all diagonal entries 1 and a linear matrix $L \in \mathbb{F}\langle X\rangle^{\ell\times\ell}$ such that $P \left(\begin{array}{c|c} A & 0 \\ \hline 0 & I_k \end{array}\right) Q = L$, where $\ell = n + k$ and k is $O(n^2 \cdot s)$. All the entries of P, Q are computable by algebraic branching programs of size* $\mathrm{poly}(n, s)$*. Moreover $ncrk(A) + k = ncrk(L)$, hence the rank of A is easily computable from the rank of L.*

The proof of the above theorem can be found in the arXiv version [4].

Using Theorem 7, Lemma 10 and Theorem 13 we get a deterministic NC reduction from multivariate ncRANK to bivariate ncRANK.

▶ **Theorem 14.** *There is a deterministic* NC *reduction from the multivariate ncRANK problem to the bivariate ncRANK problem.*

5 NC Reduction from RIT to bivariate ncRANK

In this section we give an NC-Turing reduction from RIT to bivariate ncRANK. That is, we design an NC algorithm for RIT assuming we have an oracle for bivariate ncRANK. Hrubes and Wigderson in [21] give a polynomial time reduction from RIT to ncRANK problem [21, Theorem 2.6]. Also they show that for any given rational formula Φ there is a log-depth rational formula that is equivalent to Φ [21, Proposition 4.1].

Our key contribution here is to use Cohn's embedding theorem to transform RIT problem to the *bivariate* case. Then we parallelize the Hrubes-Wigderson reduction from RIT to ncRANK. In fact, if the input rational formula is already logarithmic depth then the Hrubes-Wigderson reduction from RIT to ncRANK can be implemented in NC. In this section we design an NC algorithm for depth reduction of rational formulas (possibly with division gates) assuming NC oracle for bivariate ncRANK.[9] Indeed, the construction of an equivalent log-depth rational formula, as described in [21], does not appear to be directly parallelizable, as its description crucially requires rational formula identity testing.[10]

We show that, indeed, the depth-reduction proof in [21] can be parallelized step by step. However, there are some key points where our algorithmic proof is different. Firstly, to solve the RIT instance arising in the depth-reduction proof, we need to recursively depth-reduce the corresponding subformula and then apply Hrubes-Wigderson reduction from RIT to ncRANK on the constructed log-depth subformula. Secondly, we need to handle an important case arising in the proof (namely, the case (2) in the description of the Normal-Form procedure in the proof of the Lemma 17) which was not significant for the existential argument in [21]. In fact to handle this case, we require an argument based on Brent's commutative formula depth reduction [9].

In the detailed proof of Lemma 17, we first sketch our NC algorithm for depth reduction of rational formula assuming oracle access to bivariate ncRANK. We highlight and elaborate the key steps where our proof differs from [21]. We need to reproduce some parts of their proof for completeness, for these parts we just sketch the argument and refer to [21] for the details. Using this depth reduction algorithm we give an NC Turing reduction from RIT to bivariate ncRANK, which is a main result of this section.

5.1 Depth reduction for noncommutative formulas with inversion gates

The following is a consequence of results in [21] and [17].

▶ **Theorem 15** ([21, 17]). *Let Φ be a rational formula of size s computing a* non-zero *rational function in $\mathbb{F}\langle\langle X\rangle\rangle$. If the field $\mathbb{F}$ is sufficiently large and $k > 2s$ then, at a matrix tuple $(M_1, M_2, \ldots, M_n)$ chosen uniformly at random from $M_{k\times k}(\mathbb{F})^n$, the matrix $\hat{\Phi}(M_1, \ldots, M_n)$ is nonsingular with "high" probability.*

If $\mathbb{F}$ is small then we can pick the random matrices over a suitable extension field, and by "high" probability we mean, say, $1 - 2^{-\Omega(s+n)}$.

[9] It is an interesting problem to devise an NC algorithm for rational formula depth reduction without oracle access to singularity test.

[10] The overall proof in [21] is based on the Brent's depth-reduction of commutative rational formulas [9]. In the commutative case addressed by Brent, it turns out that the construction procedure does not require oracle access to identity testing and he obtains an NC algorithm for obtaining the depth-reduced formula.

Let Φ_1 and Φ_2 be correct rational formulas of size at most s computing rational functions in $\mathbb{F}\langle\langle X\rangle\rangle$. By Theorem 15 and a union bound argument, for a random matrix substitution $(M_1, M_2, \ldots, M_n)$ from $M_{k\times k}(\mathbb{F})^n$, inputs to all the inversion gates in Φ_1 and Φ_2 simultaneously evaluate to non-singular matrices with "high" probability. Hence, for k sufficiently large we have $\mathcal{D}_{k,\Phi_1} \cap \mathcal{D}_{k,\Phi_2} \neq \emptyset$. It follows that a random matrix tuple is in $\mathcal{D}_{k,\Phi_1} \cap \mathcal{D}_{k,\Phi_2}$ with high probability.

By Theorem 15 and the definition of correct rational formulas (Definition 2), it follows that if Φ_1 and Φ_2 are size s correct formulas that are *not* equivalent then for a random matrix substitution of dimension $k > 2s$ both Φ_1 and Φ_2 are defined and they disagree with high probability. As noted in Section 2, equivalent formulas need not have the same domain of definition.

Lemma 17 is the main technical result of this section. It describes an NC algorithm for depth-reduction of correct formulas assuming an oracle for bivariate ncRANK. The next lemma is useful for establishing equivalences of formulas arising in the proof of Lemma 17. Suppose Φ is a rational formula computing the rational function $\hat{\Phi}$. For a gate v in formula Φ, Φ_v denotes the subformula rooted at v. The formula $\Phi_{v\leftarrow z} \in \mathbb{F}\langle\langle X \cup \{z\}\rangle\rangle$ is obtained from Φ by replacing subformula Φ_v with fresh variable z.

▶ **Lemma 16** (local surgery). *Let Φ be a correct rational formula and v be a gate in Φ. Suppose Ψ is a correct rational formula equivalent to Φ_v. Let $\Psi' = (A\cdot z+B)\cdot(C\cdot z+D)^{-1}$ is a formula equivalent to $\Phi_{v\leftarrow z}$ such that A, B, C, D are correct formulas which do not depend upon z and $\hat{C}\cdot\hat{\Delta}+\hat{D} \neq 0$ for any formula Δ such that $\Phi_{v\leftarrow\Delta}$ is correct. Let Φ' denote the rational formula obtained by replacing z in Ψ' by Ψ. Then Φ' is correct and it is equivalent to Φ.*

Proof. From the definitions of Φ_v and $\Phi_{v\leftarrow z}$ it follows that $\Phi = \Phi_{v\leftarrow\Phi_v}$. As Φ is correct, from the properties of formulas C, D as stated in the lemma it follows that $\hat{C}\hat{\Phi}_v + \hat{D} \neq 0$. Which implies $\hat{C}\hat{\Psi} + \hat{D} \neq 0$ as $\Psi \equiv \Phi_v$. This shows that the formula $\Phi' = (A\cdot\Psi+B)\cdot(C\cdot\Psi+D)^{-1}$ is correct. Now let $\tau = (M_1, \ldots, M_n)$ is a matrix tuple in $\mathcal{D}_\Phi \cap \mathcal{D}_{\Phi'}$, the intersection of domains of definition of Φ and Φ'. This implies $\tau \in \mathcal{D}_{\Phi_v}$ as $\mathcal{D}_\Phi \subseteq \mathcal{D}_{\Phi_v}$, Φ_v being subformula of Φ. Similarly, $\tau \in \mathcal{D}_\Psi$ as $\mathcal{D}_{\Phi'} \subseteq \mathcal{D}_\Psi$, Ψ being a subformula of Φ'. So $\tau \in \mathcal{D}_{\Phi_v} \cap \mathcal{D}_\Psi$. As $\Phi_v \equiv \Psi$, it follows that $\Phi_v(\tau) = \Psi(\tau)$. As $\tau \in \mathcal{D}_\Phi$. It implies that $(M_1, M_2, \ldots, M_n, \Phi_v(\tau)) = (\tau, \Phi_v(\tau)) \in \mathcal{D}_{\Phi_{v\leftarrow z}}$. Similarly, as $\tau \in \mathcal{D}_{\Phi'}$, it follows that $(\tau, \Psi(\tau)) \in \mathcal{D}_{\Psi'}$. As $\Phi_v(\tau) = \Psi(\tau)$, it implies

$$(\tau, \Phi_v(\tau)) = (\tau, \Psi(\tau)) \in \mathcal{D}_{\Phi_{v\leftarrow z}} \cap \mathcal{D}_{\Psi'}$$

As $\Phi_{v\leftarrow z} \equiv \Psi'$, this implies $\Phi_{v\leftarrow z}(\tau, \Phi_v(\tau)) = \Psi'(\tau, \Psi(\tau))$. But $\Phi_{v\leftarrow z}(\tau, \Phi_v(\tau)) = \Phi(\tau)$ and $\Psi'(\tau, \Psi(\tau)) = \Phi'(\tau)$. So we get $\Phi(\tau) = \Phi'(\tau)$ for any $\tau \in \mathcal{D}_\Phi \cap \mathcal{D}_{\Phi'}$. Thus proving $\Phi \equiv \Phi'$. ◀

▶ **Lemma 17.** *Given a* correct *formula Φ of size s computing a rational function $f \in \mathbb{F}\langle\langle X\rangle\rangle$, for sufficiently large s and absolute constants c, b*

1. *we give an* NC *algorithm, with oracle access to bivariate ncRANK, that outputs a correct formula Φ' of depth at most $c\log_2 s$ which is equivalent to Φ.*
2. *If a variable z occurs in Φ at most once then we give an* NC *algorithm, with bivariate ncRANK as oracle, that constructs* correct *rational formulas A, B, C, D which do not depend on z with depth at most $c\log_2 s + b$ and the formula $\Phi' = (A\cdot z+B)\cdot(C\cdot z+D)^{-1}$ is equivalent to Φ. Moreover, the rational function $\hat{C}\hat{\Psi} + \hat{D} \neq 0$ for any formula Ψ such that $\Phi_{z\leftarrow\Psi}$ is correct.*

Proof. We give a recursive construction for both the parts and prove the correctness by induction on s, the size of the formula Φ.

Depth-reduce(Φ).
Input: A correct formula Φ of size s computing a rational function in $\mathbb{F}\langle\langle X\rangle\rangle$.
Output: A correct formula Φ' of depth at most $c\log_2 s$ such that $\Phi \equiv \Phi'$.

1. Find a gate $v \in \Phi$ such that $\frac{s}{3} < wt(v) \leq \frac{2s}{3}$.
2. In Parallel construct formulas Ψ, Δ such that Ψ = Depth-Reduce(Φ_v) and Δ = Normal-Form($\Phi_{v\leftarrow z}, z$).
3. Obtain formula Φ' from Δ by replacing z in Δ by Ψ.
4. Output Φ'.

Normal-Form(Φ, z).
Input: A correct formula Φ of size s computing a rational function in $\mathbb{F}\langle\langle X\rangle\rangle$ and a variable $z \in \Phi$ which appears at most once in Φ
Output: A correct formula Φ' which is of the form

$$\Phi' = (Az + B)(Cz + D)^{-1}$$

where A, B, C and D are correct rational formulas which do not depend on z with depth at most $c\log_2 s + b$. Moreover, the rational function $\hat{C}\hat{\Psi} + \hat{D} \neq 0$ for any formula Ψ such that $\Phi_{z\leftarrow\Psi}$ is correct.

Let $v_1, v_2, \ldots, v_\ell = z$ be gates on the path from root r of Φ to the leaf gate z, such that v_j is not an inverse gate. So parent of each v_j has two children, and let u_i denote the sibling of v_i. Use pointer doubling based parallel algorithm (as mentioned in the proof of Lemma 10) to compute $wt(u_i)$ and $wt(v_i)$ for all $i \in [\ell]$. We call gate v_i for $i \in [\ell]$ as a *balanced* gate if $wt(\Phi_{v_i}), wt(\Phi_{v_i\leftarrow z'}) \leq \frac{5s}{6}$. Now we consider two cases.

1. There exist a balanced gate v_i:
   a. Let $v = v_i$. In parallel compute formulas Ψ_1, Ψ_2 such that
   $$\Psi_1 = (A_1z' + B_1)(C_1z' + D_1)^{-1} = \text{Normal-Form}(\Phi_{v\leftarrow z'}, z')$$
   $$\Psi_2 = (A_2z + B_2)(C_2z + D_2)^{-1} = \text{Normal-Form}(\Phi_v, z)$$
   b. Define formulas A, B, C, D as $A_1 \cdot A_2 + B_1 \cdot C_2$, $A_1 \cdot B_2 + B_1 \cdot D_2$, $C_1 \cdot A_2 + D_1 \cdot C_2$ and $C_1 \cdot B_2 + D_1 \cdot D_2$ respectively.
   c. Let $\Phi' = (A \cdot z + B) \cdot (C \cdot z + D)^{-1}$
   d. Output Φ' and halt
2. There does not exist a balanced gate:
   In this case, we can prove that there is a unique $i \in [\ell]$ such that $wt(u_i) > \frac{s}{6}$.
   a. Let v be parent of the gate v_i.
   b. In Parallel find formulae $\Psi_1, \Psi_2, \Psi_3, \Psi_4$ such that
   $$\Psi_1 = (A_1z' + B_1)(C_1z' + D_1)^{-1} = \text{Normal-Form}(\Phi_{v\leftarrow z'}, z')$$
   $$\Psi_2 = (A_2z + B_2)(C_2z + D_2)^{-1} = \text{Normal-Form}(\Phi_{v_i}, z)$$
   $$\Psi_3 = \text{Depth-Reduce}(\Phi_{u_i})$$
   $$\Psi_4 = (Az + B)(Cz + D)^{-1} = \text{Normal-Form}(\Phi'', z)$$
   where Φ'' is obtained by replacing sub-tree rooted at u_i by 0 in Φ.
   c. Using the algorithm of Theorem 19 check if $\hat{\Psi}_3 \equiv 0$ in NC with oracle access to bivariate ncRANK. If $\hat{\Psi}_3 \equiv 0$ then output Ψ_4 and halt.

d. If $\hat{\Psi_3} \not\equiv 0$ then let $\Phi' = (A \cdot z + B) \cdot (C \cdot z + D)^{-1}$, where

$$A = \begin{cases} A_1A_2 + B_1C_2 + A_1\hat{\Psi_3}C_2 & \text{if } v_i \text{ is a } +\text{-gate} \\ A_1\hat{\Psi_3}A_2 + B_1C_2 & \text{if } v_i \text{ is a } \times\text{-gate and is a right child of } v \\ A_1A_2 + B_1\hat{\Psi_3}^{-1}C_2 & \text{if } v_i \text{ is a } \times\text{-gate and is a left child of } v \end{cases}$$

$$B = \begin{cases} A_1B_2 + B_1D_2 + A_1\hat{\Psi_3}D_2 & \text{if } v_i \text{ is a } +\text{-gate} \\ A_1\hat{\Psi_3}B_2 + B_1D_2 & \text{if } v_i \text{ is a } \times\text{-gate and is a right child of } v \\ A_1B_2 + B_1\hat{\Psi_3}^{-1}D_2 & \text{if } v_i \text{ is a } \times\text{-gate and is a left child of } v \end{cases}$$

$$C = \begin{cases} C_1A_2 + D_1C_2 + C_1\hat{\Psi_3}C_2 & \text{if } v_i \text{ is a } +\text{-gate} \\ C_1\hat{\Psi_3}A_2 + D_1C_2 & \text{if } v_i \text{ is a } \times\text{-gate and is a right child of } v \\ C_1A_2 + D_1\hat{\Psi_3}^{-1}C_2 & \text{if } v_i \text{ is a } \times\text{-gate and is a left child of } v \end{cases}$$

$$D = \begin{cases} C_1B_2 + D_1D_2 + C_1\hat{\Psi_3}D_2 & \text{if } v_i \text{ is a } +\text{-gate} \\ C_1\hat{\Psi_3}B_2 + D_1D_2 & \text{if } v_i \text{ is a } \times\text{-gate and is a right child of } v \\ C_1B_2 + D_1\hat{\Psi_3}^{-1}D_2 & \text{if } v_i \text{ is a } \times\text{-gate and is a left child of } v \end{cases}$$

e. Output Φ' and halt.

Case 2 above is not explicitly dealt with in [21] as their focus is on the *existence* of a log-depth formula equivalent to Φ. In contrast to that, in our case we want to *algorithmically construct* log-depth formula Φ' equivalent to Φ. This makes the details of Case 2 crucial as the construction of Φ' in Case 2 depends on whether $\hat{\Phi_{u_i}} \equiv 0$ or not. To solve this RIT instance we need to recursively compute log-depth formula Ψ_3 equivalent to Φ_{u_i} and then invoke algorithm of theorem 19 to carry out RIT test in NC with oracle access to bivariate ncRANK, as in Step (c).

We first prove the correctness of both the algorithms described above using induction on s, then we analyze the parallel complexity of both the algorithms. We will choose appropriate constants c, b during the proof.

Correctness of the algorithm Depth-Reduce. We know that there exists a gate $v \in \Phi$ such that $\frac{s}{3} < wt(v) \leq \frac{2s}{3}$. As in proof of Lemma 10 we can find such a gate v as required by Step 1 of the Depth-Reduce algorithm. Clearly, the formulas Φ_v and $\Phi_{v \leftarrow z}$ are of size at most $2s/3$. Using inductive assumption, we can construct a correct formula Ψ such that depth of Ψ is at most $c \log_2 \frac{2s}{3}$ and $\Psi \equiv \Phi_v$. Again using inductive assumption we can construct correct formulas A, B, C, D (which do not depend on z) of depth at most $c \log_2 \frac{2s}{3} + b$ such that the formula $\Delta = (A \cdot z + B) \cdot (C \cdot z + D)^{-1}$ is equivalent to $\Phi_{v \leftarrow z}$. Since Φ is equal to the formula obtained from $\Phi_{v \leftarrow z}$ by replacing z by Φ_v and Φ is correct so from inductive assumption it follows that $\hat{C}\hat{\Phi_v} + \hat{D} \neq 0$.

As $\Psi \equiv \Phi_v$, $\Delta \equiv \Phi_{v \leftarrow z}$ and $\hat{C}\hat{\Phi_v} + \hat{D} \neq 0$ from Lemma 16 it follows that Φ' is correct and $\Phi' \equiv \Phi$. Since $\Phi' = (A \cdot \Psi + B) \cdot (C \cdot \Psi + D)^{-1}$ we get that depth of Φ' is at most $c \log_2 \frac{2s}{3} + b + 4$. By choosing constant $c \geq \frac{b+4}{(\log_2 3 - 1)}$, we get that the depth of Φ' is at most $c \log_2 s$. Completing the inductive argument for the correctness proof of the algorithm Depth-Reduce.

Correctness of the algorithm Normal-Form. In case (1) we know that there exists a balanced gate $v = v_i$. We have $wt(\Phi_v), wt(\Phi_{v\leftarrow z'}) \leq \frac{5s}{6}$. So by inductive assumption we know that the formulas A_j, B_j, C_j, D_j for $j \in \{1,2\}$ are correct, and their depths are at most $c\log_2 \frac{5s}{6} + b$. Now using compositionality of the z-normal forms as in Proposition 4.1 of [21] it follows that the formula $\Phi' = (A\cdot z + B)\cdot(C\cdot z + D)^{-1}$ is equivalent to Φ where A, B, C, D are $A_1\cdot A_2 + B_1\cdot C_2$, $A_1\cdot B_2 + B_1\cdot D_2$, $C_1\cdot A_2 + D_1\cdot C_2$ and $C_1\cdot B_2 + D_1\cdot D_2$ respectively. Also, clearly the depth of A, B, C, D is at most $c\log_2 \frac{5s}{6} + b + 2$. By choosing $c \geq \frac{2}{(\log_2 6 - \log_2 5)}$ it follows that the depths of formulas A, B, C, D are at most $c\log_2 s + b$. To complete the inductive proof we need to prove that $\hat{C}\hat{\pi} + \hat{D} \neq 0$ for any formula π such that $\Phi_{z\leftarrow\pi}$ is correct. Let π be such that $\Phi_{z\leftarrow\pi}$ is correct. For simplicity lets denote formulas Φ_v and $\Phi_{v\leftarrow z'}$ by α and β respectively. Since $\Phi_{z\leftarrow\pi}$ is correct, it implies $\alpha_{z\leftarrow\pi}$ is correct as α is a subformula of Φ. So by inductive assumption $\hat{C}_2\hat{\pi} + \hat{D}_2 \neq 0$. Since $\Phi_{z\leftarrow\pi}$ is correct, β being a subformula of Φ it also implies $\beta_{z'\leftarrow\gamma}$ is correct where $\gamma = (A_2\cdot\pi + B_2)\cdot(C_2\cdot\pi + D_2)^{-1}$. By inductive assumption we get

$$\begin{aligned}
\hat{C}_1\hat{\gamma} + D_1 &\neq 0 \text{ which implies} \\
\hat{C}_1[(\hat{A}_2\cdot\hat{\pi} + \hat{B}_2)\cdot(\hat{C}_2\cdot\hat{\pi} + \hat{D}_2)^{-1}] + D_1 &\neq 0 \text{ which implies} \\
\hat{C}_1(\hat{A}_2\cdot\hat{\pi} + \hat{B}_2) + \hat{D}_1(\hat{C}_2\cdot\hat{\pi} + \hat{D}_2) &\neq 0 \text{ as } \hat{C}_2\hat{\pi} + \hat{D}_2 \neq 0 \\
\text{So } (\hat{C}_1\hat{A}_2 + \hat{D}_1\hat{C}_2)\hat{\pi} + (\hat{C}_1\hat{B}_2 + \hat{D}_1\hat{D}_2) &\neq 0 \text{ which implies} \\
\hat{C}\hat{\pi} + \hat{D} &\neq 0
\end{aligned}$$

This completes the proof for case 1.

Next we argue that case 1 and case 2 together cover all the possibilities. To see this, we will argue that if there does not exist a unique u_i with $wt(u_i) \geq \frac{s}{6}$ then there must exist a balanced gate. There are two possibilities: either for every gate u_i, $wt(u_i) \leq \frac{s}{6}$ or there are two or more gates u_i's with $wt(u_i) > \frac{s}{6}$ If for every $i \in [\ell]$, $wt(u_i) \leq \frac{s}{6}$, we find smallest j such that $\sum_{i=1}^{j} wt(u_i) > \frac{s}{6}$. Clearly $\sum_{i=1}^{j} wt(u_i) \leq \frac{2s}{6}$, which implies $wt(\Phi_v), wt(\Phi_{v\leftarrow z'}) \leq \frac{5s}{6}$ for $v = v_{i+1}$. So v is balanced. If there are two or more u_i's such that $wt(u_i) > \frac{s}{6}$ then v be parent of gate u_i such that i is a largest index with $wt(u_i) > \frac{s}{6}$. Clearly v is a balanced gate. This proves that case 1, 2 together cover all possibilities.

Assume that there is a unique $i \in [\ell]$ such that $wt(u_i) > \frac{s}{6}$. We first apply Depth-Reduce on formula Φ_{u_i} and get a log-depth formula Ψ_3 equivalent to Φ_{u_i}, we carry out this depth reduction to efficiently test if $\Phi_{u_i} \equiv 0$?. We will give details on this later when we figure out the parallel time complexity of the algorithm. Now when $\Phi_{u_i} \equiv \Psi_3 \equiv 0$, clearly formula $\Phi \equiv \Phi''$ where Φ'' is a formula obtained from Φ by replacing sub-formula rooted at u_i by 0. As $wt(u_i) > \frac{s}{6}$, we have $|\Phi''| \leq \frac{5s}{6}$. So by inductive assumption, the correct sub-formulas A, B, C, D of Ψ_4 obtained by recursive call Normal-Form(Φ'', z) have depth at most $c\log_2 \frac{5s}{6} + b \leq c\log_2 s + b$ and $\Psi_4 \equiv \Phi'' \equiv \Phi$. So it follows, $\Phi_{z\leftarrow\pi} \equiv \Phi''_{z\leftarrow\pi}$. Consequently $\Phi_{z\leftarrow\pi}$ is correct iff $\Phi''_{z\leftarrow\pi}$ is correct. So from inductive hypothesis it follows that $\hat{C}\hat{\pi} + \hat{D} \neq 0$ for any formula π such that $\Phi_{z\leftarrow\pi}$ is correct. This proves the correctness of Normal-form procedure when $\Phi_{u_i} \equiv 0$.

Now let $\Phi_{u_i} \not\equiv 0$. Let v be the parent of u_i. Below we discuss the case when v is $\times$-gate and u_i is a right child of v.

We have $\Psi_2 \equiv \Phi_{v_i} \equiv (A_2\cdot z + B_2)\cdot(C_2\cdot z + D_2)^{-1}$. Let $h_1 = A_2\cdot z + B_2$ and $h_2 = C_2\cdot z + D_2$). So $\Phi_{v_i} \equiv h_1\cdot h_2^{-1}$. Now as v is $\times$-gate and v_i, u_i are left and right children of v respectively. So we get $\Phi_v \equiv h_1h_2^{-1}\Phi_{u_i} \equiv h_1h_2^{-1}\Psi_3$. We have $\Psi_1 \equiv \Phi_{v\leftarrow z'} = (A_1z' + B_1)(C_1z' + D_1)^{-1}$. So we get

$$\begin{aligned}
\Phi &\equiv (\ A_1 \cdot (h_1 h_2^{-1}\Psi_3) + B_1) \cdot (\ C_1 \cdot (h_1 h_2^{-1}\Psi_3) + D_1)^{-1} \\
&\equiv (A_1 \cdot h_1 + B_1 \cdot \Psi_3^{-1} h_2) \cdot h_2^{-1}\Psi_3 \cdot [\ (C_1 \cdot h_1 + D_1 \cdot \Psi_3^{-1} h_2) \cdot h_2^{-1}\Psi_3\ ]^{-1} \\
&\equiv (A_1 \cdot h_1 + B_1 \cdot \Psi_3^{-1} h_2) \cdot h_2^{-1}\Psi_3 \cdot \Psi_3^{-1} h_2\ \cdot (C_1 \cdot h_1 + D_1 \cdot \Psi_3^{-1} h_2)^{-1} \\
&\equiv (A_1 \cdot h_1 + B_1 \cdot \Psi_3^{-1} h_2) \cdot (C_1 \cdot h_1 + D_1 \cdot \Psi_3^{-1} h_2)^{-1}
\end{aligned}$$

By substituting values of h_1, h_2 and simplifying we get that $\Phi \equiv (A \cdot z + B) \cdot (C \cdot z + D)^{-1}$ where A, B, C, D are $A_1 \cdot A_2 + B_1 \cdot \Psi_3^{-1} \cdot C_2$, $A_1 \cdot B_2 + B_1 \cdot \Psi_3^{-1} \cdot D_2$, $C_1 \cdot A_2 + D_1 \cdot \Psi_3^{-1} \cdot C_2$ and $C_1 \cdot B_2 + D_1 \cdot \Psi_3^{-1} \cdot D_2$ respectively as defined in Step 2(d).

As $wt(u_i) > \frac{s}{6}$, clearly $|\Phi_{v_i}|, |\Phi_{v \leftarrow z'}| \leq \frac{5s}{6}$. Since

$$\begin{aligned}
\Psi_1 &= (A_1 z' + B_1)(C_1 z' + D_1)^{-1} = \text{Normal-Form}(\Phi_{v \leftarrow z'}, z') \\
\Psi_2 &= (A_2 z + B_2)(C_2 z + D_2)^{-1} = \text{Normal-Form}(\Phi_{v_i}, z)
\end{aligned}$$

So by inductive assumptions the sub-formulas A_1, B_1, C_1, D_1 of Ψ_1 and the sub-formulas A_2, B_2, C_2, D_2 of Ψ_2 are correct and have depths at most $c \log_2 \frac{5s}{6} + b$. As $\Psi_3 = \text{Depth-Reduce}(\Phi_{u_i})$ and $|\Phi_{u_i}| < s$ by inductive assumption we get that the depth of Ψ_3 is at most $c \log_2 s$. Clearly $c \log_2 \frac{5s}{6} + b \leq c \log_2 s$ for $c \geq \frac{b}{\log_2 6 - \log_2 5)}$. So from expressions for A, B, C, D it follows that the depth of A, B, C, D is at most depth of Ψ_3 plus 4. Which implies that the depth of A, B, C, D is at most $c \log_2 s + 4 \leq c \log_2 s + b$ if the constant $b \geq 4$. This gives us the desired bound on the depth of A, B, C, D. To summarize if we choose constant $b \geq 4$ and choose constant c such that it satisfies all the lower bounds required in different steps of the above proof, we will get the desired bound on the depth of A, B, C, D. We can show that $\hat{C}\hat{\pi} + \hat{D} \neq 0$ for any formula π such that $\Phi_{z \leftarrow \pi}$ is correct. The proof is similar to one for Case (1), we additionally need to take into account $\times$-gate at v while composing z-Normal forms Ψ_1 and Ψ_2. We skip the details.

When v is a v is a $\times$-gate and u_i is a left child of v, the composition of z-normal forms is easy as we do not need an oracle access for RIT as in the case discussed above when u_i is the right child. In the commutative case we can by default assume that u_i is the left child. Precisely for this reason Brent's construction [9] is independent of whether $\Phi_{u_i} \equiv 0$ or not. We skip the details of cases when v is a $+$-gate or v is $\times$-gate and u_i is the left child which can be handled similar to case (1). This proves the correctness of the procedure Normal-Form.

Next we analyze the parallel time complexity of both the procedures. Let $t_1(s), t_2(s)$ denote the parallel time complexities of the procedures Depth-Reduce and Normal-Form respectively. The step (1) of the Depth reduce procedure can be implemented in NC so it has parallel time complexity $(\log_2 s)^k$ for some absolute constant k. As both the formulas Φ_v and $\Phi_{v \leftarrow z}$ have sizes at most $2s/3$, the parallel time complexity of step (2) is at most $\max(t_1(2s/3), t_2(2s/3))$. So we get the following recurrence for $t_1(s)$.

$$t_1(s) \leq (\log_2 s)^a + \max\left(\ t_1\left(\frac{2s}{3}\right)\ ,\ t_2\left(\frac{2s}{3}\ \right)\right)$$

where a is an absolute constant. Now we obtain recurrence for $t_2(s)$. In Case (1) of Normal-Form when there exist a balanced gate v_i, we can find such a gate in NC. Both the formulas Φ_v and $\Phi_{v \leftarrow z'}$ have the sizes at most $5s/6$ so step 1(a) takes parallel time $\max(t_1(5s/6), t_2(5s/6))$. In case (2) the formulas $\Phi_{v \leftarrow z'}$, Φ_{v_i} and Φ'' all have sizes at most $5s/6$ and formula Φ_{u_i} has size at most $s - 1$. So collectively we get the following recurrence for $t_2(s)$

$$\begin{aligned} t_2(s) &\leq (\log_2 s)^b + \max\left(t_1\left(\frac{5s}{6}\right) \ , \ t_2\left(\frac{5s}{6}\right) \ , \ t_1(s-1)\right) \\ &\leq (\log_2 s)^b + \max\left(t_2\left(\frac{5s}{6}\right) \ , \ t_1(s)\right) \end{aligned}$$

for an absolute constant b. The upper bound $t_1(s), t_2(s) \leq (\log_2 s)^c$ for sufficiently large constant c follows from an easy induction. Hence both the procedures can be implemented in deterministic NC. ◀

Hrubes-Wigderson reduction from RIT to ncRANK

Now we recall the polynomial-time reduction from RIT to ncRANK from [21, Theorem 2.6]. Given a rational formula Φ their reduction outputs an invertible linear matrix M in the variables X.[11] Their reduction ensures that the top right entry of M^{-1} is $\hat{\Phi}$, the rational function computed by the formula Φ. It turns out that if Φ is already of logarithmic depth then their reduction can be implemented in NC.

▶ **Theorem 18** ([21]). *Let Φ be a rational formula of size s and depth $O(\log s)$ computing a rational expression in $\mathbb{F}\langle\langle X\rangle\rangle$ there is an* NC *algorithm to construct an invertible linear matrix M_Φ such that the top right entry of M_Φ^{-1} is $\hat{\Phi}$.*

Proof. We only briefly sketch the NC algorithm. Their reduction recursively constructs the matrix M_Φ, using the formula structure of Φ.

Given a formula Φ we can compute the sizes of all its subformulas in NC using a standard pointer doubling algorithm. This allows us to estimate the dimensions of matrices M_{Φ_v} for subformulas Φ_v for each gate v of Φ. We can also compute in NC the precise location for placement of the sub-matrices M_{Φ_v} inside the matrix M_Φ following their construction. Assuming that Φ is already of logarithmic depth, there are only $O(\log s)$ nested recursive calls for this recursive procedure. This ensures that the overall process can be implemented in NC. ◀

After constructing linear matrix M_Φ such that the top right entry of M_Φ^{-1} is $\hat{\Phi}$, define matrix M' as $M' = \begin{pmatrix} v^T & M_\Phi \\ 0 & -u \end{pmatrix}$

where u, v are $1 \times k$ vectors, such that $u = (1, 0, \ldots, 0)$ and $v = (0, 0, \ldots, 0, 1)$ where k is the dimension of the matrix M_Φ. It follows that $\hat{\Phi} \neq 0$ iff M' is invertible in the skew field $\mathbb{F}\langle\langle X\rangle\rangle$ (see e.g. [18, Proposition 3.29]). So we have the following theorem.

▶ **Theorem 19** ([21]). *Let Φ be a rational formula of size s and depth $O(\log s)$ computing a rational expression in $\mathbb{F}\langle\langle X\rangle\rangle$ then there is an* NC *algorithm to construct a linear matrix M such that $\hat{\Phi} \neq 0$ iff M is invertible in the skew field $\mathbb{F}\langle\langle X\rangle\rangle$.*

Now, from Lemma 17, Theorem 19, and Theorem 9, we obtain an NC Turing reduction from multivariate RIT to bivariate ncRANK.

▶ **Theorem 20.** *There is a deterministic* NC *Turing reduction from RIT problem to ncRANK problem for bivariate linear matrices.*

[11] Notice that the entries of M^{-1} are elements of the skew field $\mathbb{F}\langle\langle X\rangle\rangle$

Concluding Remarks

Motivated by the question whether RIT and ncRANK have deterministic NC algorithms, we show that multivariate RIT is NC-reducible to bivariate RIT and multivariate ncRANK is NC-reducible to bivariate ncRANK. RIT is known to be polynomial-time reducible to ncRANK, and indeed that is how the polynomial-time algorithm for RIT works, by reducing to ncRANK and solving ncRANK. We show that RIT is deterministic NC-Turing reducible to ncRANK. We prove this by showing that noncommutative rational formula depth reduction is NC-Turing reducible to ncRANK. The main open problem is to obtain deterministic NC algorithms for bivariate ncRANK and bivariate RIT. We also leave open finding an unconditional NC algorithm for depth-reduction of noncommutative rational formulas.

References

1 S. A. Amitsur. Rational identities and applications to algebra and geometry. *Journal of Algebra*, 3:304–359, 1966.

2 V. Arvind, Abhranil Chatterjee, and Partha Mukhopadhyay. Black-box identity testing of noncommutative rational formulas in deterministic quasipolynomial time. *CoRR*, abs/2309.1564, 2023. In STOC 2024, to appear. arXiv:2309.15647.

3 V. Arvind, Abhranil Chatterjee, and Partha Mukhopadhyay. Trading determinism for noncommutativity in Edmonds' problem. *CoRR*, abs/2404.07986, 2024. arXiv:2404.07986.

4 V. Arvind and Pushkar Joglekar. A multivariate to bivariate reduction for noncommutative rank and related results. *CoRR*, abs/2404.16382, 2024. arXiv:2404.16382.

5 Vikraman Arvind, Pushkar S. Joglekar, Partha Mukhopadhyay, and S. Raja. Randomized polynomial-time identity testing for noncommutative circuits. *Theory Comput.*, 15:1–36, 2019. doi:10.4086/TOC.2019.V015A007.

6 G. M. Bergman. Skew fields of noncommutative rational functions, after Amitsur. *Sé Schü–Lentin–Nivat, Paris*, 1970.

7 Markus Bläser, Gorav Jindal, and Anurag Pandey. A deterministic PTAS for the commutative rank of matrix spaces. *Theory Comput.*, 14(1):1–21, 2018. doi:10.4086/TOC.2018.V014A003.

8 Andrej Bogdanov and Hoeteck Wee. More on noncommutative polynomial identity testing. In *20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA*, pages 92–99, 2005. doi:10.1109/CCC.2005.13.

9 Richard P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, 1974. doi:10.1145/321812.321815.

10 Abhranil Chatterjee and Partha Mukhopadhyay. The noncommutative edmonds' problem re-visited. *CoRR*, abs/2305.09984, 2023. doi:10.48550/arXiv.2305.09984.

11 P. M. Cohn. The embedding of fir in skew fields. *Proceedings of the London Mathematical Society*, 23:193–213, 1971.

12 P. M. Cohn. Universal skew fields of fractions. *Sympos. Math.*, 8:135–148, 1972.

13 P. M. Cohn. *Free Rings and their Relations*. London Mathematical Society Monographs. Academic Press, 1985.

14 P. M. Cohn. An embedding theorem for free associative algebras. *Mathematica Pannonica*, 1(1):49–56, 1990.

15 P. M. Cohn. *Free Ideal Rings and Localization in General Rings*. New Mathematical Monographs. Cambridge University Press, 2006. doi:10.1017/CBO9780511542794.

16 Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978. doi:10.1016/0020-0190(78)90067-4.

17 Harm Derksen and Visu Makam. Polynomial degree bounds for matrix semi-invariants. *CoRR*, abs/1512.03393, 2015. arXiv:1512.03393.

18 Ankit Garg, Leonid Gurvits, Rafael Mendes de Oliveira, and Avi Wigderson. A deterministic polynomial time algorithm for non-commutative rational identity testing. *CoRR*, abs/1511.03730, 2015. arXiv:1511.03730.

19 Ankit Garg, Leonid Gurvits, Rafael Mendes de Oliveira, and Avi Wigderson. Operator scaling: Theory and applications. *Found. Comput. Math.*, 20(2):223–290, 2020. doi:10.1007/s10208-019-09417-z.

20 Graham Higman. The units of group-rings. *Proceedings of the London Mathematical Society*, s2-46(1):231–248, 1940. doi:10.1112/plms/s2-46.1.231.

21 Pavel Hrubes and Avi Wigderson. Non-commutative arithmetic circuits with division. *Theory Comput.*, 11:357–393, 2015. doi:10.4086/TOC.2015.V011A014.

22 Gábor Ivanyos, Youming Qiao, and K. V. Subrahmanyam. Non-commutative edmonds' problem and matrix semi-invariants. *Comput. Complex.*, 26(3):717–763, 2017. doi:10.1007/s00037-016-0143-x.

23 Gábor Ivanyos, Youming Qiao, and K. V. Subrahmanyam. Constructive non-commutative rank computation is in deterministic polynomial time. *Comput. Complex.*, 27(4):561–593, 2018. doi:10.1007/s00037-018-0165-7.

24 Noam Nisan. Lower bounds for non-commutative computation (extended abstract). In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 410–418. ACM, 1991. doi:10.1145/103418.103462.

25 Sanguthevar Rajasekaran and John H. Reif, editors. *Handbook of Parallel Computing - Models, Algorithms and Applications*. Chapman and Hall/CRC, 2007. doi:10.1201/9781420011296.

26 Ran Raz and Amir Shpilka. Deterministic polynomial identity testing in non-commutative models. *Comput. Complex.*, 14(1):1–19, 2005. doi:10.1007/S00037-005-0188-8.

27 L. H. Rowen. *Polynomial identities in ring theory, Pure and Applied Mathematics*. Academic Press Inc., Harcourt Brace Jovanovich Publishers, New York, 1980.

28 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.

29 James C. Wyllie. *Complexity of Parallel Computation, PhD Thesis*. Cornell University, 1979.

30 Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation, Marseille, France, June 1979, Proceedings*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979. doi:10.1007/3-540-09519-5_73.

List Update with Delays or Time Windows

Yossi Azar
School of Computer Science, Tel Aviv University, Israel

Shahar Lewkowicz
School of Computer Science, Tel Aviv University, Israel

Danny Vainstein
School of Computer Science, Tel Aviv University, Israel
Google Research, Tel Aviv, Israel

Abstract

We address the problem of **List Update**, which is considered one of the fundamental problems in online algorithms and competitive analysis. In this context, we are presented with a list of elements and receive requests for these elements over time. Our objective is to fulfill these requests, incurring a cost proportional to their position in the list. Additionally, we can swap any two consecutive elements at a cost of 1. The renowned "Move to Front" algorithm, introduced by Sleator and Tarjan, immediately moves any requested element to the front of the list. They demonstrated that this algorithm achieves a competitive ratio of 2. While this bound is impressive, the actual cost of the algorithm's solution can be excessively high. For example, if we request the last half of the list, the resulting solution cost becomes quadratic in the list's length.

To address this issue, we consider a more generalized problem called **List Update with Time Windows**. In this variant, each request arrives with a specific deadline by which it must be served, rather than being served immediately. Moreover, we allow the algorithm to process multiple requests simultaneously, accessing the corresponding elements in a single pass. The cost incurred in this case is determined by the position of the furthest element accessed, leading to a significant reduction in the total solution cost. We introduce this problem to explore lower solution costs, but it necessitates the development of new algorithms. For instance, Move-to-Front fails when handling the simple scenario of requesting the last half of the list with overlapping time windows. In our work, we present a natural $O(1)$ competitive algorithm for this problem. While the algorithm itself is intuitive, its analysis is intricate, requiring the use of a novel potential function.

Additionally, we delve into a more general problem called **List Update with Delays**, where the fixed deadlines are replaced with arbitrary delay functions. In this case, the cost includes not only the access and swapping costs, but also penalties for the delays incurred until the requests are served. This problem encompasses a special case known as the prize collecting version, where a request may go unserved up to a given deadline, resulting in a specified penalty. For this more comprehensive problem, we establish an $O(1)$ competitive algorithm. However, the algorithm for the delay version is more complex, and its analysis involves significantly more intricate considerations.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online, List Update, Delay, Time Window, Deadline

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.15

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2304.06565 [12]

Funding *Yossi Azar*: Supported in part by the Israel Science Foundation (grant No. 2304/20).

1 Introduction

One of the fundamental problems in online algorithms is the **List Update** problem. In this problem we are given an ordered list of elements and requests for these elements that arrive over time. Upon the arrival of a request, the algorithm must serve it immediately by accessing the required element. The cost of accessing an element is equal to its position in

the list. Finally, any two consecutive elements in the list may be swapped at a cost of 1. The goal in this problem is to devise an algorithm so as to minimize the total cost of accesses and swaps. Note that it is an online algorithm and hence does not have any knowledge of future requests and must decide what elements to swap only based on requests that have already arrived.

Although the list update problem is a fundamental and simple problem, its solutions may be costly. Consider the following example. Assume that we are given requests to each of the elements in the farther half of the list. Serving these requests sequentially results in quadratic cost (quadratic in the length of the list). However, in many scenarios, while the requests arrive simultaneously, they do not have to be served immediately. Instead, they arrive with some deadline such that they must be served some time in the (maybe near) future. If this is the case, and the requests' deadlines are further in the future than their arrival, they may be jointly served; thereby incurring a linear (rather than quadratic) cost in the former example. This example motivates the following definition of List Update with Time Windows problem which may improve the algorithms' costs significantly.

The **List Update with Time Windows** problem is an extension of the classical List Update problem. Requests are once again defined as requests that arrive over time for elements in the list. However, in this problem they arrive with future deadlines. Requests must be served during their time window which is defined as the time between the corresponding request's arrival and deadline. This grants some flexibility, allowing an algorithm to serve multiple requests jointly at a point in time which lies in the intersection of their time windows. The cost of serving a set of requests is defined as the current position of the farthest of those elements (i.e. serving a request for the i-th item in the list causes all the other active requests for the first i elements in the list to be served as well in this access operation). In addition, as in the classical problem, swaps between any two consecutive elements may be performed at a cost of 1. Note that both accessing elements (or, serving requests) and swapping consecutive elements is done instantaneously (i.e., time does not advance during these actions). The goal is then to devise an online algorithm so as to minimize the total cost of serving requests and swapping elements. Also note that this problem encapsulates the original List Update problem. In particular, the List Update problem can be viewed as List Update with Time Windows where each time window consist of a unique single point.

We also consider a generalization of the time-window version – **List Update with Delays**. In this problem each request is associated with an arbitrary delay function, such that an algorithm accumulates delay cost while the request remains pending (i.e., unserved). The goal is to minimize the cost of serving the requests plus the total delay. This provides an incentive for the algorithm to serve the requests early.

Another interesting and related variant is the prize collecting variant, which has been heavily researched in other fields as well. The price collecting problem is a special case of List Update with Delay and a generalization of List Update with Time Windows. In the context of List Update, the prize collecting problem is defined such that a request must be either served until some deadline or incur some penalty. Note that List Update with Delays encapsulates this variant by defining a delay function that incurs 0 cost and thereafter (at the deadline) immediately jumps to the penalty cost. The prize collecting problem encapsulates List Update with Time Windows when the penalty is arbitrarily large.

While the flexibility introduced in the list update with time windows or delays problems allow for lower cost solutions, it also introduces complexity in the considered algorithms. In particular, the added lenience will force us to compare different algorithms (our online algorithm compared to an optimal algorithm, for instance) at different time points in the

input sequence. Since the problem definition allows for serving requests at different time points, this results in different sets of unserved requests when comparing the algorithms – this divergence will prove to be the crux of the problem and will result in significant added complexity compared to the classical List Update problem.

Originally, the List Update problem was defined to allow for free swaps to the accessed element: i.e., immediately after serving an element e, the algorithm may move e towards the head of the list - free of charge. All other swaps between consecutive elements still incur a cost of 1. In our work, it will be convenient for us to consider the version of the problem where these free swaps are not allowed and all swaps between two consecutive elements incur a cost of 1. We would like to stress that while these two settings may seem different, this is not the case. One may easily observe that the difference in costs between a given solution in the two models is at most a multiplicative factor of 2. This can be seen to be true since the cost of the free swaps may be attributed to the cost of accessing the corresponding element (that was swapped) which is always at least as large. Thus, our results extend easily to the model with free swaps to the accessed element (by losing a factor of 2 in the competitive ratio). In particular, an algorithm which is constant competitive for one of the models is also constant-competitive for the other.

Using the standard definitions an **offline algorithm** sees the entire sequence of requests in advance and thus may leverage this knowledge for better solutions. Conversely, an **online algorithm** only sees a request (i.e., its corresponding element and entire time window or a delay function) upon its arrival and thus must make decisions based only on requests that have already arrived[1]. To analyze the performance of our algorithms we use the classical notion of **competitive ratio**. An online algorithm is said to be c-competitive (for $c \geq 1$) if for every input, the cost of the online algorithm is at most c times the cost of the optimal offline algorithm[2].

1.1 Our Results

In this paper, we show the following results:

- For the List Update with Time Windows problem we provide a 24-competitive algorithm.
- For the List Update with Delays we provide a 336-competitive algorithm.

For the **time windows** version the algorithm is natural. Upon a deadline of a request for an element, the algorithm serves all requests up to twice the element's position and then moves that element to the beginning of the list. Note that the algorithm does not use the fact that the deadline is known when the request arrives. I.e. our result holds even if the deadline is unknown until it is reached (as in non-clairvoyant models). Also note that while the algorithm is deceptively straightforward - its resulting analysis is tremendously more involved.

In the **delay** version the algorithm is more sophisticated. (See the full version [12] for counter examples to some simpler algorithms). The algorithm maintains two types of counters: request counters and element counters. For every request, its request counter increases over time at a rate proportional to the delay cost the request incurred. The request

[1] In principle the time when a request arrives (i.e., is revealed to the online algorithm) need not be the same as the time when its time window or delay begins (i.e., when the algorithm may serve the request). Note however that the change makes no difference with respect to the offline algorithms but only allows for greater flexibility of the online algorithms. Therefore, any competitiveness results for our problem will transcend to instances with this change.

[2] We note that the lists of both the online algorithm and the optimum offline algorithm are identical at the beginning.

counter will be deleted at some point in time after the request has been served (it may not happen immediately after the request is served, but rather further in the future). Unlike the request counters, an element counter's scope is the entire time horizon. The element counter increases over time at a rate that is proportional to the sum of delay costs of unserved requests to that element. Once the requests are served, the element counter ceases to increase. There are two types of events that cause the algorithm to take action: prefix-request-counter events and element-counter events. A prefix-request-counter event takes place when the sum of the request counters of the first ℓ elements reaches a value of ℓ. This event causes the algorithm to access the first 2ℓ elements in the list and delete the request counters for requests to the first ℓ elements. The request counters of the elements in positions $\ell + 1$ up to 2ℓ remain undeleted but cease to increase (Note that this will also result in the first 2ℓ element counters to also cease to increase). An element-counter event takes place when an element counter's value reaches the element's position. Let ℓ be that position. This event causes the algorithm to access the first 2ℓ elements in the list. Thereafter, the algorithm deletes all request counters of requests to that element. Finally, the element's counter is zeroed and the algorithm moves the element to the front.

It is interesting to note that List Update with Delay in the clairvoyant case can be reduced to the special case of prize collecting List Update (which is a generalization of List Update with Time Windows) by creating multiple requests with appropriate penalties. However, neither our algorithm for Delay nor our proof are getting simplified for this case, therefore we present our algorithm and proof for the general case (i.e. for List Update with Delay). Moreover, the reduction from List Update with Delay to prize collecting holds only for the clairvoyant case while our algorithm works on the non-clairvoyant model as well. The full version of this paper contains all the omitted proofs, figures and additional counter examples, and appears in [12].

1.2 Previous Work

We begin by reviewing previous work relating to the classical List Update problem. Sleator and Tarjan [27] began this line of work by introducing the deterministic online algorithm Move to Front (i.e. MTF). Upon a request for an element e, this algorithm accesses e and then moves e to the beginning of the list. They proved that MTF is 2-competitive in a model where free swaps to the accessed element are allowed. The proof uses a potential function defined as the number of inversions between MTF's list and OPT's list. An inversion between two lists is two elements such that their order in the first list is opposite to their order in the second list. A simple lower bound of 2 for the competitive ratio of deterministic online algorithms is achieved when the adversary always requests the last element in the online algorithm's list and OPT orders the elements in its list according to the number of times they were requested in the sequence. Since the model with no free swaps differs in the cost by at most a factor of 2 this immediately yields that MTF is 4-competitive for the model with no free swaps. The simple 2 lower bound also holds for this model. Previous work regarding randomized upper bounds for the competitive ratio have been done by many others [25, 26, 2, 1, 5]. Currently, the best known competitiveness was given by Albers, Von Stengel, and Werchner [3], who presented a random online algorithm and proved it is 1.6 competitive. Previous work regarding lower bounds for this problem have also been made [28, 26, 5]. The highest of which was achieved by Ambühl, Gartner and Von Stengel [6], who proved a lower bound of 1.50084 on the competitive ratio for the classical problem. With regards to the offline classical problem: Ambühl proved this problem is NP-hard [4].

Problems with time windows have been considered for various online problems. Gupta, Kumar and Panigrahi [24] considered the problem of paging (caching) with time windows. Bienkowski et al. [16] considered the problem of online multilevel aggregation. Here, the problem is defined via a weighted rooted tree. Requests arrive on the tree's leaves with corresponding time windows. The requests must be served during their time window. Finally, the cost of serving a set of requests is defined as the weight of the subtree spanning the nodes that contain the requests. Bienkowski et al. [16] showed a $O(D^4 2^D)$ competitive algorithm where D denotes the depth of the tree. Buchbinder et al. [21] improved this to $O(D)$ competitiveness. Later, Azar and Touitou [14, 15] provided a framework for designing and analyzing algorithms for these types of metric optimization problems.

In addition, set cover with deadline [9] was also considered as well as online service in a metric space [20, 11]. To all these problems poly-logarithmic competitive algorithms were designed. It is interesting to note that in contrast to all these problems we show that for our list update problem constant competitive algorithms are achievable. We note that problems with deadline can be also extended to problems with delay where there is a monotone penalty function for each request that is increasing over time until the request is served (and is added to the original cost). Many of the results mentioned above can be extended to arbitrary penalty function. The main exception is matching with delays that can be efficiently solved (i.e. with poly-logarithmic competitive ratio) only for linear functions [22, 8, 7] as well as for concave functions [13]. For other problems that tackle deadlines and delays see: [17, 23, 10, 18, 19].

1.3 Our Techniques

While introducing delays or time windows introduces the option of serving multiple requests simultaneously thereby drastically improving the solution costs, this lenience requires the algorithms and their analyses to be much more intricate.

The "freedom" given to the algorithm compared with the classical List Update problem requires more decisions to be made: for example, in the time windows version assume there are currently two active requests: a request for an element e_1 which just reached its deadline and a request for a further element in the list, e_2 but its deadline has not been reached yet. Should the algorithm access only e_1, pay its position in the list and leave the request for e_2 to be served later or access both e_1 and e_2 together and pay the position of e_2 in the list? If no more requests arrive until the deadline of the second active request, the latter option is better. However, requests that might arrive before the deadline of the second active request might cause the former option to be better after all. In the delay version the decision is more complicated since it may be the case that there are various requests for elements, each request accumulated a small or medium delay but their total is large. Hence, we need to decide at what stage and to what extend serving these requests. Moreover it is more tricky to decide which element to move to the front of the list and at which point in time.

As for the analysis, we need to handle the fact that the online algorithm and the optimal algorithm serve requests at different times. Further, since both algorithms may serve different sets of requests at different times, we may encounter situations wherein a given request at a given time would have been served by the online algorithm and not the optimal algorithm (and vice versa). This, combined with the fact that the algorithms' lists may be ordered differently at any given time, will prove to be the crux of our problem and its analysis.

To overcome these problems, we introduce new potential functions (one for the time windows case and one for the delays case). We note that the original List Update problem was also solved using a potential function [27], however, due to the aforementioned issues,

the original function failed to capture the resulting intricacies and we had to introduce novel (and more involved) functions. Ultimately, this resulted in constant competitiveness for both settings.

List Update with Time Windows. Here, the potential function consists of three terms. The first accounts for the difference (i.e., number of inversions) between the online and optimal algorithms' lists at any given time (similar to that of Sleator and Tarjan [27]). The second term accounts for the difference in the set of served requests between the two algorithms. Specifically, whenever the optimal algorithm serves a request not yet served by the online algorithm, we add value to this term which will be subtracted once the online algorithm serves the request. The third term accounts for the movement costs made by the online algorithm incurred by requests that were already served by the optimal algorithm.

At any given time point, our proof considers separately elements that are positioned (significantly) further in the list in the online algorithm compared to the optimal algorithm, as opposed to all other elements (which we will refer to as "the closer" elements). To understand the flavor of our proofs, e.g., the incurred costs of "the further" elements is charged to the first term of the potential function. In contrast, the change in the first term is not be enough to cover the incurred costs of "the closer" elements (the term may even increase). Fortunately, the second term is indeed enough to cover both the incurred costs and the (possible) increase in the first term. Specifically, the added value is of the same order of magnitude as the access cost incurred by the optimal algorithm for serving the corresponding requests. This follows from (a) only requests for elements in ALG which are located at a position which is of the same order of magnitude as the location in OPT get "gifts" in the second term. (b) The fact that the number of trigger elements and their positions in ALGs list is bounded because upon a deadline of a trigger, ALG serves all the elements located up to twice the position of the trigger in its list. The definition of the term "trigger" appears in the beginning of Section 3.

Note however that the analysis above holds only as long as the optimal algorithm does not move an element further in the list between the time it serves it and the time the online algorithm serves it. In such a case, the third term will offset the costs.

List Update with Delays. Here, the potential function consists of five terms. The first term is similar to that of the time windows setting with the caveat that defining the distance between the online and optimal algorithms' lists should depend on the values of the element counters as well. Consider the following example. Assume that the ordering of i, j is reversed when comparing it between the online and optimal algorithms and assume it is ordered (i, j) in the online algorithm. As we defined our algorithm, once the element counter of j is filled, it is moved to the front and therefore the ordering will be reversed. Therefore, intuitively, if j element counter is almost filled we consider the distance between this pair smaller than the case where its element counter is completely empty. Therefore, we would like the contribution to the potential function to be smaller in the former case.

Note that the contribution of the inversion (i, j) depends on the element counter of j but not on the element counter of i (i.e. the contribution is asymmetric). Even if the element counter of j is very close to its position in the online algorithm's list, we still need a big contribution of the pair (i, j) in order to pay for the next element counter event on j. However, if the element counter of j is far from its position in the online algorithm's list, we will need even more contribution of the pair (i, j) to the potential function in order to also cover future delay penalty which the algorithm may suffer on the element j that will not cause an element counter event on j to occur in the short term.

The second part of the potential function consists of the delay cost that both the online and optimal algorithms incurred for requests which were active in both algorithms. This term is used to cover the next element counter events for the elements required in these requests. The third part of the potential function offsets the requests which have been served by the optimal algorithm but not by the online algorithm. This part is very similar to the gifts in the second term of the potential function in time windows and the ideas behind it are similar. Again, the gifts are only given to requests which are located by the online algorithm at a position which is of the same order of magnitude as the location in the optimal algorithm. The gift is of the same order of magnitude as the total delay the online algorithm pays for the request (including the delay it will pay in the future). This is used in order to offset the next element counter event in the online algorithm on the element. However, this gift also decreases as the online algorithm suffers more delay for the request because we want this term in the potential function to also cover the future delay penalty the online algorithm will pay for the request.

The fourth and fifth terms in the potential function are very similar to the third term in the potential function of time windows but each one of them has its own purposes: The fourth term should cover the next element counter event on the element while the fifth term should cover the scenario in which the optimal algorithm served a request and then moved the element further in its list but the online algorithm will suffer more delay penalty for this request in the future. The fifth term should cover this delay cost that the online algorithm pays and thus it is proportional to the fraction between the future delay the online algorithm pays for the request and the position of the element in the online algorithm's list.

2 The Model for Time Windows and Delays

Given an input σ and algorithm ALG we denote by $ALG(\sigma)$ the cost of its solution. Recall that in the **time windows** setting $ALG(\sigma)$ is defined as the sum of (1) the algorithm's access cost: the algorithm may serve multiple requests at a single time point and then the access cost is defined as the position of the farthest element in this set of requests. $ALG(\sigma)$ also accounts for (2) the total number of element swaps performed by ALG. In total, $ALG(\sigma)$ is equal to the sum of access costs and swaps. In the **delay** setting $ALG(\sigma)$ accounts (1) and (2) as above in addition to (3) the sum of the delay incurred by all requests. The delay is defined via a delay function that is associated with each request. The delay functions may be different per request and are each a monotone non-decreasing non-negative function. In total, $ALG(\sigma)$ is equal to the sum of access costs, swaps and delay costs. As is traditional when analysing online algorithms, we denote by $OPT(\sigma)$ the cost of the optimal solution to input σ. Furthermore, we say that ALG is c-competitive (for $c \geq 1$) if for every input σ, $ALG(\sigma) \leq c \cdot OPT(\sigma)$. Throughout our work, when clear from context, we use $ALG(\sigma)$ to denote both the cost of the solution and the solution itself. Our algorithms work also in the non-clairvoyant case: In the time windows version we only know the deadline of a request upon its deadline (and not upon its arrival). In the delay version we know the various delay functions of the requests only up to the current time. Next we introduce several notations that will aid us in our proofs.

▶ **Definition 1.** *Let $\mathbb{E}$ be the set of the elements.*

- *Let n denote the number of elements in our list ($|\mathbb{E}| = n$) and m the number of requests.*
- *Let r_k denote the kth request and e_k the requested element by r_k.*
- *Let $y_k \in [n]$ denote the position of e_k in OPTs list at the time OPT served r_k. Let $x_k \in [n]$ denote the position of e_k in ALGs list at the time OPT (and not ALG) served r_k*[3].

[3] In the delay version, x_k and y_k are defined only in case OPT indeed served the request r_k at some time.

Throughout our work, given an element in the list, we oftentimes consider its neighboring elements in the list. We therefore introduce the following conventions to avoid confusion. Given an element in the list we refer to its **previous** element as its neighbor which is closer to the head of the list and its **next** element as its neighbor which is further from the head of the list.

3 The Algorithm for Time Windows

Prior to defining our algorithm, we need the following definitions.

▶ **Definition 2.** *We define the* ***triggering element****, when a deadline of a request is reached, as the farthest element in the list such that there exists an active request for it which just reached its deadline. We define the* ***triggering request*** *as one of the active requests for the triggering element that just reached its deadline - arbitrary.*

When clear from context we will use the term "trigger" instead of "triggering request" or "triggering element". Next, we define the algorithm.

Algorithm 1 Algorithm for Time Windows (i.e. Deadlines).

```
1 Upon deadline of a request do:
2     i ← triggering element's position
3     Serve the set of requests in the first 2i − 1 elements in the list
4     Move-to-front the triggering element
```

We prove the following theorem for the above algorithm in Appendix A.

▶ **Theorem 3.** *For each sequence of requests σ, we have that*

$$ALG(\sigma) \leq 24 \cdot OPT(\sigma).$$

4 The Algorithm for Delays

Our algorithm maintains two types of counters in order to process the input: requests counters and element counters. We begin by defining the **request counters**. The algorithm maintains a separate request counter for every incoming request. For a given request r_k we denote its corresponding counter as RC_k. The counter is initialized to 0 the moment the request arrives and increases at the same rate that the request incurs delay. Once the request is served, the counter ceases to increase. Finally, our algorithm deletes the request counters - it will do so at some point in the future after the request is served (but not necessarily immediately when the request is served).

Next we define the **element counters**. Unlike the request counters, element counters exist throughout the entire input (i.e., they are initialized at the start of the input and do not get deleted). We define an element counter EC_e for every element $e \in \mathbb{E}$. These counters are initialized to 0 and increase at a rate equal to the total delay incurred by requests to the specific element.

We define two types of events that cause the algorithm to act: prefix-request-counter events and element-counter events. A **prefix request counters event on ℓ** for $\ell \in [n]$ occurs when the sum of all the request counters of requests for the first ℓ elements in the list reaches the value of ℓ. When this type of event takes place, the algorithm performs the

following two actions. First, it serves the requests of the first 2ℓ elements. Second, it deletes the request counters that belong to the first ℓ elements. Note that these are the request elements that contributed to this event and are therefore deleted. Also note that the request counters of the elements $\ell+1$ to 2ℓ and the element counters of the first 2ℓ elements cease to increase since their requests have been served.

An **element counter event on e** for $e \in \mathbb{E}$ occurs when EC_e reaches the value of ℓ, where $\ell \in [n]$ is the position of the element e in the list, currently. When this type of event takes place, the algorithm performs the following three actions. First, it serves the requests on the first 2ℓ elements. Second, it deletes all request counters of requests to the element e. Third, it sets EC_e to 0 and perform move-to-front to e.

Note that the increase in an element counter equals to the sum of the increase of all the request counters to this element. In particular, the value of the element counter is at least the sum of the non-deleted request counters for the element (It may be larger since request counters may be deleted in request counters events while the element counter maintains its value). Hence when we zero an element counter, we also delete the request counters of requests for this element in order to maintain this invariant.

Next, we present the algorithm.

Algorithm 2 Algorithm for Delay.

Initialization:
  For each $e \in \mathbb{E}$ **do**:
    $EC_e \leftarrow 0$
Upon arrival of a new request r_k **do**:
  $RC_k \leftarrow 0$
Upon prefix-request-counters event on $\ell \in [n]$ **do**:
  Serve the set of requests in the first 2ℓ elements in the list
  Delete the request counters for the first ℓ elements in the list
Upon element-counter event on e (let ℓ denote e's current position) **do**:
  Serve the set of requests in the first 2ℓ elements in the list
  Delete all the request counters of requests for the element e
  $EC_e \leftarrow 0$
  Move-to-front the element e

We prove the following theorem for the above algorithm in the full version.

▶ **Theorem 4.** *For each sequence of requests σ, we have that*

$$ALG(\sigma) \leq 336 \cdot OPT(\sigma).$$

5 Potential Functions for Time Windows and Delay

Our proofs use potential functions. In particular we prove for each possible event that

$$\Delta ALG + \Delta\Phi \leq c \cdot \Delta OPT$$

where Φ is the potential and c is the competitive ratio. In this section we describe the potential functions. The detailed proofs that use these potential functions appear in the full version.

5.1 Time Windows

As mentioned earlier, our potential function used for the time windows setting consists of three terms. We will define them separately. We begin with the first term that aims to capture the difference between ALG and OPT's lists at any given moment.

▶ **Definition 5.** *Let $\phi(t)$ denote the number of **inversions** between ALG's and OPT's lists at time t. Specifically,*

$$\phi(t) = |\{(i,j) \in \mathbb{E}^2 | \text{ At time } t, \text{ } i \text{ is before } j \text{ in ALG's list and after } j \text{ in OPT's list}\}|.$$

The second term accounts for the difference in the set of served requests between the two algorithms. Specifically, whenever the optimal algorithm serves a request not yet served by the online algorithm, we add value to this term which will be subtracted once the online algorithm serves the request. Before defining this term, we need the following definition.

▶ **Definition 6.** *For each time t, let $\lambda(t) \subseteq [m]$ be the set of all the request indices k such that the request r_k arrived and was served by OPT but was not served by ALG at time t.*

Recall that for request r_k we denote by y_k the position of e_k in OPT's list at the time that OPT served r_k. Furthermore, we denote by x_k the position of e_k in ALG's list at the time OPT (and not ALG) served r_k. We are now ready to define the second term in our potential function.

▶ **Definition 7.** *For $k \in \lambda(t)$ we define $\psi(x_k, y_k) \geq 0$ as*

$$\psi(x,y) = \begin{cases} 7x & \text{if } 1 \leq x \leq y \\ 8y - x & \text{if } y \leq x \leq 8y \\ 0 & \text{if } 8y \leq x \end{cases}$$

Next, we define the third term of our potential function.

▶ **Definition 8.** *We define $\mu_k(t)$ as the number of swaps OPT performed between e_k and its next element in the list from the time OPT served the request r_k until time t.*

Finally, we combine the terms and define our potential function.

▶ **Definition 9.** *We define our potential function for Time Windows as*

$$\Phi(t) = 4 \cdot \phi(t) + \sum_{k \in \lambda(t)} \psi(x_k, y_k) + 4 \cdot \sum_{k \in \lambda(t)} \mu_k(t).$$

5.2 Delay

In the delays setting, we define a different potential function that is comprised of five terms. We will define the terms separately first and thereafter use them to compose our potential function. We begin with the first term.

As mentioned in Our Techniques, the first term also aim to capture the distance between ALG's and OPT's lists. In the time windows setting, we defined this term as the number of element inversions. In the delays case this does not suffice; we have to take into the account the elements' counters as well. To gain some intuition as to why this addition is needed, consider the following example. Assume that elements i, j are ordered (i, j) in ALG and reversed in OPT. Recall that ALG is defined such that when j' element counter is filled,

then we move it to the front (thereby changing the ALG's ordering to (j,i)). Therefore, if it is the case that j's element counter is nearly filled, intuitively we may say that i,j's ordering in ALG and OPT are closer to each other than if j's element counter would have been empty. Therefore, we would like the contribution to the potential function to be smaller in the former case.

Note that the contribution of the inversion (i,j) depends on the element counter of j but not on the element counter of i (i.e. the contribution is asymmetric). Even if the element counter of j is very close to its position in the online algorithm's list, we still need a big contribution of the pair (i,j) in order to pay for the next element counter event on j. However, if the element counter of j is far from its position in the online algorithm's list, we need even more contribution of the pair (i,j) to the potential function in order to also cover future delay penalty which the algorithm may suffer on the element j that does not cause an element counter event on j to occur in the short term. Before formally defining this term, we define the following.

▶ **Definition 10.** *For a time t and an element $e \in \mathbb{E}$ we define:*
- *EC_e^t to be the value of the element counter EC_e at time t.*
- *$x_e^t \in [n]$ ($y_e^t \in [n]$ resp) to be the position of e in ALGs (OPTs resp) list at time t.*
- *$I_e^t = \{i \in \mathbb{E} | i$ is before e in ALGs list and after e in OPTs list at time $t\}$.*

▶ **Definition 11.** *For element $e \in \mathbb{E}$ we define $\rho_e(t) = |I_e^t| \cdot (28 - 8 \cdot \frac{EC_e^t}{x_e^t})$*

Observe that each $i \in I_e^t$ contributes $20 + 8 \cdot (1 - \frac{EC_e^t}{x_e^t})$ to $\rho_e(t)$. The additive term of 20 is used in order to cover the next element counter event for e while the second term is used to cover the delay penalty ALG will pay in the future for requests for e. Note that the term $1 - \frac{EC_e^t}{x_e^t}$ is the fraction of EC_e which is not "filled" yet. If this term is very low, ALG is very close to have an element counter event on e, which causes the order of i and e in ALGs list and OPTs list to be the same, thus it makes sense that the contribution of i to $\rho_e(t)$ is lower compared with the case where $1 - \frac{EC_e^t}{x_e^t}$ would be higher.

Next, we consider the second term. First, we denote the total incurred delay by a request as $d_k(t)$. Formally, this is defined as follows.

▶ **Definition 12.** *For a given request r_k and time t let $d_k(t)$ denote the total delay incurred by the request by ALG up to time t. (Note that it is defined as 0 before the request arrived and remains unchanged after the request is served). Let $d_k = \sup_t d_k(t)$. Note that this is a supremum and not maximum for the case that r_k is never served. Note that $d_k \leq n$ because ALG always serves r_k before $d_k > n$.*

Our second term is a sum of incurred delay costs of specific elements.

▶ **Definition 13.** *For each $k \in [m]$, the request r_k is considered:*
- ***active** in ALG (resp. OPT) from the time it arrives until it is served by ALG (resp. OPT).*
- ***frozen** from the time it is served by ALG until EC_{e_k} is zeroed in an e_k element counter event.*

▶ **Definition 14.** *For time t we define $\lambda(t) \subseteq [m]$ as the set of requests (request indices) which are either active **or** frozen in ALG at time t. We define $\lambda_1(t) \subseteq \lambda(t)$ as the set of requests that are also active in OPT at time t and $\lambda_2(t) \subseteq \lambda(t)$ as the set of requests that are also not active in OPT at time t.*

Finally, we define our second term.

▶ **Definition 15.** *We define the second term of the Delays potential function as* $\sum_{k\in\lambda_1(t)} d_k(t)$.

The third term is defined as follows (we use x_k and y_k as previously defined).

▶ **Definition 16.** *We define the third term as* $\sum_{k\in\lambda_2(t)}(42d_k - 6d_k(t)) \cdot 1[x_k \leq 4y_k]$.

Note that $42d_k - 6d_k(t) = 36d_k + 6 \cdot (d_k - d_k(t))$. Therefore each request index $k \in \lambda_2(t)$ contributes two terms to Φ: $36d_k$ is used to cover the next element counter on e_k while the second term is 6 times the delay ALG will pay for r_k in the future, which will be used to cover this exact delay penalty that ALG will pay in the future for r_k.

The fourth term is defined to cover the next element counter event on a given element as follows.

▶ **Definition 17.** *Let $\mu_e(t)$, for $e \in \mathbb{E}$, be the number of swaps OPT performed between e and its next element in its list ever since the last element counter event before time t on e by ALG (or the beginning of the time horizon if there was not such an event).*

Finally, we define the fifth term. The fifth term should cover the scenario in which the optimal algorithm served a request and then moved the element further in its list but the online algorithm will suffer more delay penalty for this request in the future. It will also cover the delay cost that the online algorithm will pay and thus it is proportional to the fraction between the future delay the online algorithm will pay for the request and the position of the element in the online algorithm's list.

▶ **Definition 18.** *Let $\mu_k(t)$, for $k \in \lambda_2(t)$, be the number of swaps OPT performed between e_k and its next element in its list ever since OPT served the request r_k (by accessing e_k).*

▶ **Definition 19.** *We define the fifth term of the Delays potential function as*

$$8 \cdot \sum_{k\in\lambda_2(t)} \frac{d_k - d_k(t)}{x^t_{e_k}} \cdot \mu_k(t).$$

We are now ready to define our potential function.

▶ **Definition 20.** *We define our potential function for the delays setting as*

$$\Phi(t) = \sum_{e\in\mathbb{E}} \rho_e(t) + 36 \cdot \sum_{k\in\lambda_1(t)} d_k(t) + \sum_{k\in\lambda_2(t)} (42d_k - 6d_k(t)) \cdot 1[x_k \leq 4y_k] +$$
$$+ 48 \cdot \sum_{e\in\mathbb{E}} \mu_e(t) + 8 \cdot \sum_{k\in\lambda_2(t)} \frac{d_k - d_k(t)}{x^t_{e_k}} \cdot \mu_k(t)$$

6 Conclusion and Open Problems

In this paper, we presented the List Update with Time Windows and Delay, which generalize the classical List Update problem.

- We presented a 24-competitive ratio algorithm for the List Update with Time Windows problem.
- We presented a 336-competitive ratio algorithm for the List Update with Delays problem.
- Open problems: The main issue left unsolved is the gap between the upper and lower bounds. Currently, the best lower bound for both problems considered is 2. Note that this is the same lower bound given to the original List Update problem. An interesting followup would be to improve upon this result and show a better lower bound. On the other hand, one may improve the upper bound - our algorithms are non-clairvoyant in the

sense that our proofs and algorithms hold even when the deadlines/delays are unknown. It would be interesting to understand whether clairvoyance may improve the upper bound. Another interesting direction would be to consider randomization as a way of improving our bounds.

References

1. Susanne Albers. Improved randomized on-line algorithms for the list update problem. *SIAM Journal on Computing*, 27(3):682–693, 1998.
2. Susanne Albers and Michael Mitzenmacher. Revisiting the counter algorithms for list update. *Information processing letters*, 64(3):155–160, 1997.
3. Susanne Albers, Bernhard Von Stengel, and Ralph Werchner. A combined bit and timestamp algorithm for the list update problem. *Information Processing Letters*, 56(3):135–139, 1995.
4. Christoph Ambühl. Offline list update is np-hard. In *European Symposium on Algorithms*, pages 42–51. Springer, 2000.
5. Christoph Ambühl, Bernd Gärtner, and Bernhard von Stengel. Optimal projective algorithms for the list update problem. In *International Colloquium on Automata, Languages, and Programming*, pages 305–316. Springer, 2000.
6. Christoph Ambühl, Bernd Gärtner, and Bernhard Von Stengel. A new lower bound for the list update problem in the partial cost model. *Theoretical Computer Science*, 268(1):3–16, 2001.
7. Itai Ashlagi, Yossi Azar, Moses Charikar, Ashish Chiplunkar, Ofir Geri, Haim Kaplan, Rahul M. Makhijani, Yuyi Wang, and Roger Wattenhofer. Min-cost bipartite perfect matching with delays. In *APPROX/RANDOM*, pages 1:1–1:20, 2017.
8. Yossi Azar, Ashish Chiplunkar, and Haim Kaplan. Polylogarithmic bounds on the competitiveness of min-cost perfect matching with delays. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1051–1061, 2017.
9. Yossi Azar, Ashish Chiplunkar, Shay Kutten, and Noam Touitou. Set cover with delay – clairvoyance is not required. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 8:1–8:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.8.
10. Yossi Azar, Yuval Emek, Rob van Stee, and Danny Vainstein. The price of clustering in bin-packing with applications to bin-packingwith delays. In *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 1–10, 2019.
11. Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. In *STOC*, pages 551–563, 2017.
12. Yossi Azar, Shahar Lewkowicz, and Danny Vainstein. List update with delays or time windows, 2023. arXiv:2304.06565.
13. Yossi Azar, Runtian Ren, and Danny Vainstein. The min-cost matching with concave delays problem. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 301–320. SIAM, 2021. doi:10.1137/1.9781611976465.20.
14. Yossi Azar and Noam Touitou. General framework for metric optimization problems with delay or with deadlines. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 60–71. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00013.
15. Yossi Azar and Noam Touitou. Beyond tree embeddings – A deterministic framework for network design with deadlines or delay. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1368–1379. IEEE, 2020. doi:10.1109/FOCS46700.2020.00129.

16 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Lukasz Jez, Jiri Sgall, Kim Thang Nguyen, and Pavel Veselý. Online algorithms for multi-level aggregation. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 12:1–12:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.12.

17 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, and Jan Marcinkowski. Online facility location with linear delay. In Amit Chakrabarti and Chaitanya Swamy, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, September 19-21, 2022, University of Illinois, Urbana-Champaign, USA (Virtual Conference)*, volume 245 of *LIPIcs*, pages 45:1–45:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.APPROX/RANDOM.2022.45.

18 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Neil B. Dobbs, Tomasz Nowicki, Maxim Sviridenko, Grzegorz Swirszcz, and Neal E. Young. Approximation algorithms for the joint replenishment problem with deadlines. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming – 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 135–147. Springer, 2013. doi:10.1007/978-3-642-39206-1_12.

19 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Lukasz Jez, Dorian Nogneng, and Jirí Sgall. Better approximation bounds for the joint replenishment problem. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 42–54. SIAM, 2014. doi:10.1137/1.9781611973402.4.

20 Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. Online service with delay on a line. In *SIROCCO*, 2018.

21 Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Ohad Talmon. O(depth)-competitive algorithm for online multi-level aggregation. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1235–1244. SIAM, 2017. doi:10.1137/1.9781611974782.80.

22 Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 333–344, 2016.

23 Leah Epstein. On bin packing with clustering and bin packing with delays. *CoRR*, abs/1908.06727, 2019. arXiv:1908.06727.

24 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. Caching with time windows and delays. *SIAM J. Comput.*, 51(4):975–1017, 2022. doi:10.1137/20m1346286.

25 Sandy Irani. Two results on the list update problem. *Information Processing Letters*, 38(6):301–306, 1991.

26 Nick Reingold, Jeffery Westbrook, and Daniel D Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, 1994.

27 Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

28 Boris Teia. A lower bound for randomized list update algorithms. *Information Processing Letters*, 47(1):5–9, 1993.

A The Analysis for the Algorithm for Time Windows

In this section we will prove Theorem 3. Throughout we will denote Algorithm 1 as *ALG*.

▶ **Definition 21.** *For each $k \in [m]$ we use a_k and q_k to denote the arrival time and deadline of the request r_k.*

As a first step towards proving Theorem 3 we prove in Lemma 22 that it is enough to consider inputs that only contain triggering requests. The proof appears in the full version [12].

▶ **Lemma 22.** *Let σ be a sequence of requests and let σ' be σ after omitting all the non-triggering requests (with respect to ALG). Then*

$$\frac{ALG(\sigma)}{OPT(\sigma)} \leq \frac{ALG(\sigma')}{OPT(\sigma')}.$$

▶ **Corollary 23.** *We may assume w.l.o.g. that the input σ only contains triggering requests (with respect to ALG).*

The following lemma is simple but will be very useful later. Recall that k refers to the index of the k'th request in the input σ and that e_k denotes its requested element.

▶ **Lemma 24.** *For every $k \in [m]$, the position of e_k in ALG's list remains unchanged throughout the time interval $[a_k, q_k)$. Hence x_k denotes the location of e_k in ALGs list during the time interval $[a_k, q_k)$.*

Proof. During the time interval between a_k and q_k, *ALG* did not access the element e_k and did not access any element located after e_k in its list, because otherwise it would be a contradiction to our assumption that r_k is a trigger request. Therefore, during the time interval mentioned above, *ALG* only accessed (served) elements which were before e_k in its list and performed move-to-fronts on them. But these move-to-fronts did not change the position of e_k in *ALG*'s list. ◀

▶ **Definition 25.** *Let z_k denote the position of the farthest element OPT accesses at the time it served r_k.*

Note that z_k defines the cost *OPT* pays for serving the set of requests that contain r_k.

Recall that *ALG* serves all requests separately (since all requests are triggering requests - Corollary 23). *OPT*, on the other hand, may serve multiple requests simultaneously. Note that at the time *OPT* serves the request r_k, it pays access cost of z_k (and it is guaranteed that $z_k \geq y_k$). The strict inequality $z_k > y_k$ occurs in case *OPT* serves a request for an element located further than e_k in its list and by accessing this far element, *OPT* also accesses e_k, thus serving r_k.

▶ **Lemma 26.** *The cost of ALG is bounded by*

$$ALG(\sigma) \leq 3 \cdot \sum_{k=1}^{m} x_k$$

Proof. For each $k \in [m]$, e_k is located at position x_k at the time when *ALG* serves r_k. *ALG* pays an access cost of at most $2x_k - 1$ when it serves the request r_k (Observe that *ALG* may pay an access cost of less than $2x_k - 1$ in case $n < 2x_k - 1$). *ALG* also pays a cost of $x_k - 1$ for performing move-to-front on e_k. Therefore, *ALG* suffers a total cost of at most $(2x_k - 1) + (x_k - 1) \leq 3x_k$ for serving this request. If we sum for all the requests, we get that $ALG(\sigma) \leq 3 \cdot \sum_{k=1}^{m} x_k$. ◀

▶ **Lemma 27.** *Let t be a time when the active request indices in ALG are $R = \{k_1, k_2, ..., k_d\}$ where $1 \leq x_{k_1} < x_{k_2} < ... < x_{k_d} \leq n$. We have:*

1. *For each $\ell \in [d-1]$, we have $q_{k_\ell} \leq q_{k_{\ell+1}}$, i.e., ALG serves the request r_{k_ℓ} before it serves $r_{k_{\ell+1}}$.*

2. *For each* $\ell \in [d-1]$ *we have that* $2x_{k_\ell} \leq x_{k_{\ell+1}}$.
3. $d \leq \log n + 1$*, i.e., at any time, there are at most* $\log n + 1$ *active requests in ALG.*

Proof. If ALG serves $r_{k_{\ell+1}}$ before it serves r_{k_ℓ}, it serves also r_{k_ℓ} by passing through e_{k_ℓ} when it accesses $e_{k_{\ell+1}}$, contradicting our assumption that r_{k_ℓ} is a trigger request (Observation 23).

If we have $x_{k_{\ell+1}} \leq 2x_{k_\ell} - 1$, then when ALG serves r_{k_ℓ}, it will also access $e_{k_{\ell+1}}$, thus serving $r_{k_{\ell+1}}$, contradicting our assumption that $r_{k_{\ell+1}}$ is a trigger request (Observation 23).

Using what we have already proved, a simple induction can be used in order to prove that for each $\ell \in [|R_t^{OPT}|]$, we have that $2^{\ell-1}x_{k_1^t} \leq x_{k_\ell^t}$. Therefore, we have that $2^{|R_t^{OPT}|-1}x_{k_1^t} \leq x_{k^t_{|R_t^{OPT}|}}$. We also have that $1 \leq x_{k_1^t}$ and $x_{k^t_{|R_t^{OPT}|}} \leq n$. These three inequalities yield to $|R_t^{OPT}| \leq \log n + 1$. ◀

Next we consider OPT's solution.

▶ **Definition 28.** *Let* T_{OPT} *be the set of times when* OPT *served requests. We then define:*

- *For each time* $t \in T_{OPT}$*, let* $R_t^{OPT} = \{k_1^t, k_2^t, ..., k^t_{|R_t^{OPT}|}\}$ *be the non-empty set of request indices that* OPT *served at time* t *where* $1 \leq x_{k_1^t} < x_{k_2^t} < ... < x_{k^t_{|R_t^{OPT}|}} \leq n$.
- *Let* $J(t) = \arg\max_{k \in R_t^{OPT}} \{y_k\}$.

By definition for each $t \in T_{OPT}$, we have

$$1 \leq y_{J(t)} = z_{k_1^t} = z_{k_2^t} = ... = z_{k^t_{|R_t^{OPT}|-1}} = z_{k^t_{|R_t^{OPT}|}} \leq n$$

Observe that at time $t \in T_{OPT}$, OPT serves the requests R_t^{OPT} together by accessing the $y_{J(t)}$'s element in its list. Therefore, OPT pays an access cost of $y_{J(t)}$ at time t.

▶ **Observation 29.** *For any* $t \in T_{OPT}$*, the total cost* OPT *pays for accessing elements at time* t *is* $y_{J(t)}$.

▶ **Lemma 30.** *Let* $t \in T_{OPT}$*. We have:*

1. *For each* $\ell \in [|R_t^{OPT}| - 1]$ *we have that* $2x_{k_\ell^t} \leq x_{k_{\ell+1}^t}$.
2. $|R_t^{OPT}| \leq \log n + 1$*, i.e,* OPT *serves at most* $\log n + 1$ *triggers at the same time.*

Proof. Since OPT served the requests R_t^{OPT} at time t, all these requests arrived at time t or before it. Therefore, from Observation 32 we get that all the requests R_t^{OPT} were active in ALG at time t. Therefore, we get that this lemma holds due to Lemma 27. Note that there may be additional requests which were active in ALG at time t but OPT did not serve at time t (meaning they were not in R_t^{OPT}), but this does not contradict the conclusion. ◀

The following lemma allows us to consider from now on only algorithms such that if they serve requests at time t, then at least one of these requests has a deadline at t. In particular, we can assume that OPT has this property. Observe that ALG also has this property.

▶ **Lemma 31.** *For every algorithm* A*, there exists an algorithm* B *such that for each sequence of request* σ *we are guaranteed that:*

1. B *only serves requests upon some deadline.*
2. $B(\sigma) \leq A(\sigma)$.

For convenience, we assume that when both ALG and OPT are serving σ, in case both OPT and ALG perform access or swapping operations at the same time - we first let OPT perform its operations and only then ALG will perform its operations.

On the other hand, for elements which are not served at the same time by OPT and ALG, by combining the fact that ALG serves requests at their deadline (see Corollary 23) with the fact that OPT must serve requests before the deadline, we get that again OPT serves the request before ALG. Combining the two cases yields Observation 32.

▶ **Observation 32.** *For each $k \in [m]$, OPT serves the request r_k before ALG serves r_k.*

▶ **Definition 33.** *We define the set of **events** P which contains the following 3 types of events:*

1. *ALG serves the request r_k at time q_k.*
2. *OPT serves the requests R_t^{OPT} at time t.*
3. *OPT swaps two elements.*

Recall that the potential function Φ is defined in Section 5.1 as follows:

$$\Phi(t) = 4 \cdot \phi(t) + \sum_{k \in \lambda(t)} \psi(x_k, y_k) + 4 \cdot \sum_{k \in \lambda(t)} \mu_k(t)$$

where the terms ϕ, λ, ψ and μ_k are also defined in that section.

▶ **Definition 34.** *For each event $p \in P$, we define:*

- *ALG^p (OPT^p) to be the cost ALG (OPT) pays during p.*
- *For any parameter z, Δz^p to be the value of z after p minus the value of z before p.*

Clearly, we have $ALG(\sigma) = \sum_{p \in P} ALG^p$ and $OPT(\sigma) = \sum_{p \in P} OPT^p$. Observe that Φ starts with 0 (since at the beginning, the lists of ALG and OPT are identical) and is always non-negative. Therefore, if we prove that for each event $p \in P$, we have

$$ALG^p + \Delta\Phi^p \leq 24 \cdot OPT^p$$

then, by summing it up for over all the events, we will be able to prove Theorem 3. Note that we do not care about the actual value $\Phi(t)$ by itself, for any time t. We will only measure the change of Φ as a result of each type of event in order to prove that the inequality mentioned above indeed holds. The three types of events that we will discuss are:

1. The event where ALG serves the request r_k at time q_k (event type 1) is analyzed in Lemma 35.
2. The event where OPT serves the requests R_t^{OPT} at time t (event type 2) is analyzed in Lemma 39.
3. The event where OPT swaps two elements (event type 3) is analyzed in Lemma 40.

We begin by analyzing the event where ALG served a request.

▶ **Lemma 35.** *Let $p \in P$ be the event where ALG served the request r_k (where $k \in [m]$) at time q_k. We have*

$$ALG^p + \Delta\Phi^p \leq 0 \ (= OPT^p)$$

In order to prove Lemma 35, we separate the movement of ALG versus the movement of OPT. The final proof is the superposition of the two movements. Firstly we assume that OPT did not increase the position of e_k in its list ever since it served the request r_k until ALG served it, then we remove this assumption.

▶ **Lemma 36.** *Let $p \in P$ be the event where ALG served the request r_k (where $k \in [m]$) at time q_k. Assume that ever since OPT served r_k until ALG served r_k, OPT did not increase the position of e_k in its list. We have that*

$$3x_k + 4 \cdot \Delta\phi^p - \psi(x_k, y_k) \leq 0$$

Proof. The assumption means that at time q_k, the position of e_k in OPT's list is at most y_k (it may be even lower, due to movements which may be performed by OPT to e_k towards the beginning of its list, after OPT served r_k). Recall that after ALG serves r_k, the position of e_k in ALG's list changes from x_k to 1, as a result of the move-to-front ALG performs on e_k. In order to prove the required inequality, we consider the following cases, depending on the value of y_k:

- The case $1 \leq x_k \leq y_k$. We have $\psi(x_k, y_k) = 7x_k$.
  Therefore, observe that it is sufficient to prove that $\Delta\phi^p \leq x_k$.
  This is indeed the case, because moving e_k from position x_k to position 1 in ALG's list required ALG to perform $x_k - 1$ swaps, each one of those caused ϕ to either increase by 1 or decrease by 1. Therefore, all these $x_k - 1$ swaps cause ϕ to increase by at most $x_k - 1$.
- The case $y_k \leq x_k \leq n$.
  On one hand, there are at least $x_k - y_k$ elements which were before e_k in ALG's list and after e_k in OPT's list before the move-to-front ALG performed on e_k, but they will be after e_k in ALG's list after this move-to-front. This causes ϕ to decrease by at least $x_k - y_k$. On the other hand, there are at most $y_k - 1$ elements which were before e_k in both OPT's list and ALG's list before the move-to-front ALG performed on e_k, but they will be after e_k in ALG's list after this move-to-front. This causes ϕ to increase by at most $y_k - 1$. Therefore, we have that

  $$\Delta\phi^p \leq -(x_k - y_k) + (y_k - 1) = 2y_k - x_k - 1$$

  Hence,

  $$3x_k + 4 \cdot \Delta\phi^p \leq 3x_k + 4 \cdot (2y_k - x_k - 1) \leq 8y_k - x_k$$

  Now we distinguish between these two following cases, depending on the value of y_k:
  - The case $y_k \leq x_k \leq 8y_k$. We have that $\psi(x_k, y_k) = 8y_k - x_k$. Hence

    $$3x_k + 4 \cdot \Delta\phi^p - \psi(x_k, y_k) \leq 8y_k - x_k - \psi(x_k, y_k) = 8y_k - x_k - (8y_k - x_k) = 0$$

  - The case $8y_k \leq x_k \leq n$. We have that $\psi(x_k, y_k) = 0$. Hence

    $$3x_k + 4 \cdot \Delta\phi^p - \psi(x_k, y_k) \leq 8y_k - x_k - \psi(x_k, y_k) = 8y_k - x_k \leq 8y_k - 8y_k = 0$$

◀

Now we are ready to complete the proof of Lemma 35.

Proof of Lemma 35. Since OPT has already served this request, we have $OPT^p = 0$. As explained in the proof of Lemma 26, we have $ALG^p \leq 3x_k$. Therefore, we are left with the task to prove that

$$3x_k + \Delta\Phi^p \leq 0$$

Observe that $\psi(x_k, y_k)$ and $\mu_k(t)$ are dropped (and thus are subtracted) from Φ as a result of ALG serving r_k. Therefore, we are left with the task to prove that

$$3x_k + 4 \cdot \Delta\phi^p - \psi(x_k, y_k) - 4 \cdot \mu_k(t) \leq 0$$

We first assume that ever since OPT served r_k until ALG served r_k, OPT did not increase the position of e_k in its list (later we remove this assumption). This assumption means that $\mu_k(t) = 0$. Therefore, due to Lemma 36, we have that the above inequality holds. We are left with the task to prove that the above inequality continues to hold even without this assumption.

Assume that ever since OPT served r_k until ALG served r_k, OPT performed a swap between e_k and another element where e_k's position has been increased as a result of this swap. We shall prove that the above inequality continues to hold nonetheless.

On one hand, this swap causes either an increase of 1 or a decrease of 1 to $\Delta\phi^p$. Therefore, the left term of the inequality will be increased by at most 4. On the other hand, the left term of the inequality will certainty be decreased by 4 as a result of this swap, because μ_k will certainty be increased by 1. To conclude, a decrease of at least $4 - 4 = 0$ will be applied to the left term of the inequality, thus the inequality will continue to hold after this swap as well.

By using the argument above for each swap of the type mentioned above, we get that the above inequality continues to hold even without the assumption that OPT did not increase the position of e_k in its list since it served r_k until ALG served r_k, thus the lemma has been proven. ◀

Now that we analyzed the event when ALG serves a request, the next target is to analyze the event where OPT serves multiple request together. The following observation contains useful properties of ψ that will be used later on. The reader may prove them algebraically.

▶ **Observation 37.** *For each $x, x', y, y' \in [1, \infty)$ such that $x \leq x'$ and $y \leq y'$, the function ψ satisfies the following claims:*

1. $0 \leq \psi(x, y) \leq 7x$.
2. *If $y \leq x \leq x'$ then $\psi(x', y) \leq \psi(x, y)$.*
3. $\psi(x, y) \leq \psi(x, y')$.

The target now is to analyze the event when OPT serves the requests R_t^{OPT} together at time t. Recall that when OPT serves a request r_k, the value $\psi(x_k, y_k)$ is added to Φ. The following lemma will be needed in order to analyze this event.

▶ **Lemma 38.** *Let $a > 0$ and let $f : [0, \infty) \to [0, \infty)$ be the function defined as follows:*

$$f(x) = \begin{cases} 7x & \text{if } 0 \leq x \leq a \\ 8a - x & \text{if } a \leq x \leq 8a \\ 0 & \text{if } 8a \leq x \end{cases}$$

Consider the optimization problem Q of choosing a (possibly infinite) subset $U \subseteq (0, 8a)$ that will maximize $\sum_{x \in U} f(x)$ with the requirement $\forall x, y \in U : x < y \implies 2x \leq y$. Then the optimal value of Q is $24a$.

Now we can use Lemma 38 in order to analyze the event when OPT serves multiple requests together. The proofs of Lemmas 38 and 39 appear in the full version [12].

▶ **Lemma 39.** *Let $p \in P$ be the event where OPT served the requests R_t^{OPT} at time t. We have*

$$ALG^p + \Delta\Phi^p \leq 24 \cdot OPT^p$$

We analyzed the event where ALG serves a request and the event when OPT serves multiple requests together. The only event which is left to be analyzed is the event when OPT performs a swap. We analyze it in the lemma below. The proof is in the full version [12].

▶ **Lemma 40.** *Let $p \in P$ be the event where OPT performed a swap at time t. We have*

$$ALG^p + \Delta\Phi^p \leq 8 \cdot OPT^p$$

We are now ready to prove Theorem 3.

Proof of Theorem 3. Due to Lemma 35, Lemma 39 and Lemma 40, we have for each event $p \in P$ that

$$ALG^p + \Delta\Phi^p \leq 24 \cdot OPT^p$$

The theorem follows by summing it up for over all events and use the fact that Φ starts with 0 and is always non-negative. ◀

NP-Hardness of Testing Equivalence to Sparse Polynomials and to Constant-Support Polynomials

Omkar Baraskar
University of Waterloo, Canada

Agrim Dewan
Indian Institute of Science, Bengaluru, India

Chandan Saha
Indian Institute of Science, Bengaluru, India

Pulkit Sinha
University of Waterloo, Canada

Abstract

An s-sparse polynomial has at most s monomials with nonzero coefficients. The Equivalence Testing problem for sparse polynomials (ETsparse) asks to decide if a given polynomial f is equivalent to (i.e., in the orbit of) some s-sparse polynomial. In other words, given $f \in \mathbb{F}[\mathbf{x}]$ and $s \in \mathbb{N}$, ETsparse asks to check if there exist $A \in \mathrm{GL}(|\mathbf{x}|, \mathbb{F})$ and $\mathbf{b} \in \mathbb{F}^{|\mathbf{x}|}$ such that $f(A\mathbf{x} + \mathbf{b})$ is s-sparse. We show that ETsparse is NP-hard over any field $\mathbb{F}$, if f is given in the sparse representation, i.e., as a list of nonzero coefficients and exponent vectors. This answers a question posed by Gupta, Saha and Thankey (SODA 2023) and also, more explicitly, by Baraskar, Dewan and Saha (STACS 2024). The result implies that the Minimum Circuit Size Problem (MCSP) is NP-hard for a *dense* subclass of depth-3 arithmetic circuits if the input is given in sparse representation. We also show that approximating the smallest s_0 such that a given s-sparse polynomial f is in the orbit of some s_0-sparse polynomial to within a factor of $s^{\frac{1}{3}-\epsilon}$ is NP-hard for any $\epsilon > 0$; observe that s-factor approximation is trivial as the input is s-sparse. Finally, we show that for any constant $\sigma \geq 6$, checking if a polynomial (given in sparse representation) is in the orbit of some support-σ polynomial is NP-hard. Support of a polynomial f is the maximum number of variables present in any monomial of f. These results are obtained via direct reductions from the 3-SAT problem.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases Equivalence testing, MCSP, sparse polynomials, 3SAT

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.16

Category Track A: Algorithms, Complexity and Games

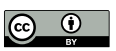
Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2024/104`

Funding *Chandan Saha*: Partially supported by a MATRICS grant of the Science and Engineering Research Board, DST, India.
Pulkit Sinha: Partially supported by a Mike and Ophelia Lazardis Fellowship, and partially supported by NSERC Canada.

Acknowledgements We thank the anonymous reviewers for their detailed and constructive feedback, which has helped us improve the presentation of this work. In particular, we thank one of the reviewers for pointing out some inaccuracies in the original proofs of Lemmas 50 and 51; simpler proofs for both lemmas came up in the process of fixing these inaccuracies.

1 Introduction

The Polynomial Equivalence (PE) problem asks to decide if two polynomials, given as lists of coefficients, are equivalent. Polynomials $f, g \in \mathbb{F}[\mathbf{x}]$ are *equivalent*, denoted as $f \sim g$, if there is an $A \in \mathrm{GL}(|\mathbf{x}|, \mathbb{F})$ and a $\mathbf{b} \in \mathbb{F}^{|\mathbf{x}|}$ such that $f = g(A\mathbf{x} + \mathbf{b})$. Equivalent polynomials



represent the same function up to a change of the coordinate system.[1] The PE problem is thus regarded as the algebraic analog of the graph isomorphism (GI) problem. PE is at least as hard as GI [2, 36], but we do not know if it is much harder than GI. There is, in fact, a cryptographic authentication scheme based on the presumed average-case hardness of PE [46]. Is PE NP-hard? Over finite fields, PE is not NP-hard unless the polynomial hierarchy collapses [51, 55]. In contrast, PE is not even known to be decidable over $\mathbb{Q}$. With the aim of gaining more insight into the complexity of testing polynomial equivalence, a natural variant of PE has been studied in the literature. This variant is known as *equivalence testing*.

In the following discussion, whenever we write "circuit(s)" and "formula(s)", we mean arithmetic circuit(s) and arithmetic formula(s), respectively, unless mentioned otherwise.[2]

Equivalence testing. Equivalence testing (ET) comes in two flavors – ET for polynomial families and ET for circuit classes. ET for a polynomial family $\mathcal{F}$ is defined as follows: given a *single* polynomial f, check if it is equivalent to some $g \in \mathcal{F}$. This variant of PE was introduced in [37, 36], wherein randomized polynomial-time ET algorithms were provided for the permanent, determinant, and elementary and power symmetric polynomial families. Subsequently, efficient ET algorithms were given for various other important polynomial families, such as the iterated matrix multiplication (IMM) family [39] (see the Related Works section in the full version). These algorithms are efficient even if f is provided as a circuit or a black-box.[3] ET for a circuit class $\mathcal{C}$ (a.k.a testing equivalence to $\mathcal{C}$) is defined similarly: given a polynomial f, decide if it is equivalent to some polynomial g that is computable by a circuit in $\mathcal{C}$. Recently, efficient ET algorithms have been given for read-once formulas [24] and a special subclass of sparse polynomials, namely t-design polynomials for constant t [8]. Sparse polynomials are depth-2 circuits.[4] It is natural to ask whether or not ET can be solved efficiently for *general* sparse polynomials. This question was posed in [24] and also, more explicitly, in [8].

Before proceeding to discuss ET for sparse polynomials, we point out a subtle difference between ET for polynomial families and that for circuit classes. The polynomial families for which ET has been studied so far are such that if f is equivalent to some g in the family, then g is unique and it can be readily identified from f. For example, if f is equivalent to some determinant polynomial[5], then we know which one simply from the number of variables of f. Moreover, polynomials in most of these families admit well-known polynomial-size circuits. So, a circuit for g can be derived once it is identified. Thus, if f is also given as a circuit, then ET for such a family reduces to PE with the input polynomials given as circuits. Over finite fields, this version of PE is in $\mathsf{AM} \cap \mathsf{coAM}$ and hence unlikely to be NP-hard. On the other hand, in the case of ET for a circuit class, if f is equivalent to some circuit C in the class, then C need *not* be unique, and further, C may not be easily deducible from f. This leaves us with the prospect of proving that ET is hard for some natural circuit class. Do sparse polynomials form such a class?

---

[1] Over $\mathbb{R}$, an invertible map $\mathbf{x} \mapsto A\mathbf{x} + \mathbf{b}$ is simply a combination of rotation, reflection, scaling, and translation.

[2] An *arithmetic circuit* is like a Boolean circuit but with AND and OR replaced by $\times$ and $+$ gates, and with edges labelled by $\mathbb{F}$-elements. It computes a polynomial over $\mathbb{F}$. A *formula* is a circuit whose underlying graph is a tree.

[3] Black-box access to f means oracle access to f, we get $f(\mathbf{a})$ from a query point $\mathbf{a}$ in unit time. It is as if f is given as a "hidden" circuit, and the only operation we are allowed is to evaluate the circuit at chosen points.

[4] We assume that a depth-2 circuit has a $+$ gate on top and a bottom layer of $\times$ gates. If the top gate is a $\times$ gate, then ET can be solved efficiently using polynomial factorization algorithms [35].

[5] The n^2-variate determinant polynomial is the determinant of the matrix $(x_{i,j})_{i,j\in[n]}$ of formal variables.

ET for sparse polynomials. An n-variate, degree-d polynomial is *s-sparse* if it has at most s monomials with nonzero coefficients. An s-sparse polynomial is computable by a depth-2 circuit having top fan-in s. Sparse polynomials have been extensively studied in algebraic complexity, particularly with regard to identity testing [41, 43], interpolation [10, 21, 41, 12], and factorization [56, 11] (see the tutorial [49] and the references therein for more algorithms involving sparse polynomials). ET provides yet another avenue to understand these "basic" polynomials better. ET for sparse polynomials asks to check if a given polynomial is sparse in some coordinate system. More formally, given a polynomial f as an arithmetic circuit and an $s \in \mathbb{N}$, decide if there is an s-sparse polynomial g such that $f \sim g$. This problem was studied in [20] over $\mathbb{Q}$, wherein an exponential in n^4 time algorithm was provided. There has not been any significant progress on this problem since that work. The lack of improvements in the complexity for over three decades makes one wonder:

Is ET for sparse polynomials NP-hard?

In this work, we answer this question in the affirmative over *any* field (see the first part of Theorem 2) even if the input f is provided as a depth-2 circuit. The result answers the question posed in [24, 8]. To our knowledge, the theorem gives the first example of a natural circuit class for which ET is provably hard.

Although ET for sparse polynomials (ETsparse) is a fairly natural problem, there is a deeper reason to study ETsparse originating from the expressive power of affine projections of sparse polynomials and the *Minimum Circuit Size Problem* (MCSP) for depth-3 circuits. We discuss this reason below to motivate ETsparse when the input is a *homogeneous* polynomial.

1.1 ETsparse and MCSP for depth-3 circuits

First, we need a few definitions: A polynomial g is an *affine projection* of f if $g = f(A\mathbf{x} + \mathbf{b})$ for some $A \in \mathbb{F}^{|\mathbf{x}|\times|\mathbf{x}|}$ and $\mathbf{b} \in \mathbb{F}^{|\mathbf{x}|}$. If $\mathbf{b} = 0$, we say g is a *linear projection* of f; additionally, if $A \in \mathrm{GL}(|\mathbf{x}|)$, we say g is in the *orbit* of f, denoted as $\mathrm{orb}(f)$. Depth-3 circuits form a highly expressive class [23, 54]. A depth-3 ($\Sigma\Pi\Sigma$) circuit is a circuit with a $+$ gate on top, a middle layer of $\times$ gates, and a bottom layer of $+$ gates. A depth-3 circuit with a top fan-in of s is an affine projection of an s-sparse polynomial. Thus, the problem of deciding if a given f is an affine projection of an s-sparse polynomial is closely related to MCSP for depth-3 circuits. We say "closely related to" instead of "the same as" because the size of a depth-3 circuit is determined by not only its top fan-in but also its formal degree.

MCSP. The complexity of MCSP for Boolean circuits has baffled researchers for over six decades. MCSP for a Boolean circuit class $\mathcal{C}$ ($\mathcal{C}$-MCSP) takes input the truth table of an n-variate Boolean function f and a parameter $s \in \mathbb{N}$ and asks to check if f is computable by a circuit in $\mathcal{C}$ of size at most s. There are intriguing connections between MCSP and several other areas such as cryptography [34, 3], learning theory [14], average-case complexity [27], and proof complexity [47]. Whether or not MCSP for general Boolean circuits is NP-hard is a long-standing open question. It is known that MCSP is NP-hard for DNF [44, 4] and DNF $\circ$ XOR formulas [29]. But no NP-hardness result is known (under deterministic polynomial-time reductions) for more general circuit models such as AC^0 circuits.[6] This is not too surprising

[6] However, strong hardness results are known for several powerful circuit models under randomized or quasi-polynomial time or subexponential time reductions [30, 32, 31, 28].

as [34] showed that NP-hardness of $\mathcal{C}$-MCSP under *natural*[7] deterministic polynomial-time reductions implies a $2^{\Omega(n)}$ lower bound for $\mathcal{C}$, unless NP $\subseteq$ SUBEXP. Unfortunately, such strong lower bounds are not known even for depth-3 Boolean circuits. However, a $2^{\Omega(n)}$ lower bound is known for XOR $\circ$ AND $\circ$ XOR formulas [48], which are depth-3 arithmetic circuits over $\mathbb{F}_2$ and are like DNF $\circ$ XOR formulas but with the top OR gate replaced by an XOR gate. In fact, a $2^{\Omega(n)}$ lower bound is known for depth-3 arithmetic circuits over any fixed finite field [22]. This raises hope that we will be able to prove the hardness of MCSP for depth-3 arithmetic circuits over finite fields. But how is the input given in the case of MCSP for arithmetic circuits? And what about depth-3 circuits over fields of characteristic 0?

MCSP for arithmetic circuits: Input representation and model of computation. In the Boolean setting of MCSP, one of the main reasons to assume the input to be a truth table is that the assumption puts MCSP in NP. Analogously, in the algebraic setting, we could assume that the polynomial is given in the dense representation as a list of $\binom{n+d}{n}$ coefficients. But observe that even if the input is given as an arithmetic circuit, MCSP is in the complexity class MA over finite fields. This is because verifying if two circuits compute the same polynomial is the polynomial identity testing problem, which admits a randomized polynomial-time algorithm [16, 57, 52]. Furthermore, class MA equals NP, assuming a widely believed circuit lower bound [33]. A succinct input representation also opens up the possibility of proving NP-hardness of MCSP for models, such as depth-3 circuits over fields of characteristic 0, for which strong exponential lower bounds are unknown (the MCSP hardness to lower bound implication in [34] needs the input in the dense format). The current best lower bound for depth-3 circuits over fields of characteristic 0 is quasi-polynomial in n [42, 5].

It is, therefore, reasonable to assume that the input polynomial is given succinctly as a circuit which should only facilitate our efforts in proving NP-hardness of MCSP for arithmetic circuit classes. For example, there is an instance in the Boolean setting wherein succinct representation of the input helped prove NP-hardness of MCSP long before such a hardness result was shown with respect to the dense representation – it is the case of the *partial* MCSP problem [25, 28]. In this work, we assume that the input is given as a depth-2 circuit, i.e., as a list of nonzero coefficients, and exponent vectors in unary – this is the *sparse representation*.[8]

A few remarks are in order concerning the model of computation. Over finite fields, we assume the Turing machine model. However, over arbitrary fields of characteristic 0, it is natural to consider an arithmetic model of computation (similar to the Blum-Shub-Smale machine model [13]) that allows us to store a field element in unit space and perform an arithmetic operation in unit time. Over $\mathbb{Q}$, it is not clear if MCSP for arithmetic circuits is even decidable in the Turing machine model. But, if we confine our search to size-s circuits whose field constants are $s^{O(1)}$ bit rational numbers, then we can work with the Turing machine model.

MCSP for homogeneous depth-3 circuits. The size of a $\Sigma\Pi\Sigma$ circuit is primarily determined by its formal degree and its top fan-in, whereas the size of a homogeneous depth-3 (hom-$\Sigma\Pi\Sigma$) circuit is mainly decided by its top fan-in (the formal degree of a $\Sigma\Pi\Sigma$ circuit is the maximum

[7] i.e., the size of the output of the reduction and the output parameter s depend only on the size of the input instance. Almost all reductions that show NP-hardness of problems are natural.

[8] Sparse representations of polynomials are also used in computer algebra systems wherein the exponent vector is given in binary. As the degree is $n^{O(1)}$ in this work (except on one occasion; see the remark following Theorem 16), whether the exponent vector is given in unary or binary makes little difference.

fan-in of the middle layer of $\times$ gates). MCSP for $\Sigma\Pi\Sigma$ circuits can be defined as follows: given f and $D, s \in \mathbb{N}$, decide if there is a $\Sigma\Pi\Sigma$ circuit with formal degree bounded by D and top fan-in bounded by s that computes f. Similarly, MCSP for hom-$\Sigma\Pi\Sigma$ circuits is defined as: given a homogeneous f and $s \in \mathbb{N}$, check if there is a hom-$\Sigma\Pi\Sigma$ circuit with top fan-in at most s that computes f. In order to prove NP-hardness of $\Sigma\Pi\Sigma$-MCSP, it is *necessary* to prove NP-hardness of hom-$\Sigma\Pi\Sigma$-MCSP. The reason is: a polynomial $f(x_1, x_2, \ldots, x_n)$ has a $\Sigma\Pi\Sigma$ circuit with formal degree bounded by D and top fan-in bounded by s if and only if the homogeneous polynomial $z^D f(x_1 z^{-1}, x_2 z^{-1}, \ldots, x_n z^{-1})$ has a hom-$\Sigma\Pi\Sigma$ circuit with top fan-in bounded by s. Moreover, if the reduction in a hypothetical proof of NP-hardness of hom-$\Sigma\Pi\Sigma$-MCSP has a certain simple feature, then it would imply NP-hardness of $\Sigma\Pi\Sigma$-MCSP (see the second remark following Proposition 39). Hence, it is natural to study the hardness of hom-$\Sigma\Pi\Sigma$-MCSP first.

NP-hardness of MCSP is known for two interesting subclasses of hom-$\Sigma\Pi\Sigma$ circuits, namely depth-3 powering circuits [53] and set-multilinear $\Sigma\Pi\Sigma$ circuits [26]; the top fan-in's of circuits in these two classes correspond to Waring rank and tensor rank, respectively. Perhaps an appealing evidence in favor of NP-hardness of hom-$\Sigma\Pi\Sigma$-MCSP is a proof of NP-hardness of MCSP for a "dense" subclass of hom-$\Sigma\Pi\Sigma$ circuits. Intuitively, $\mathcal{C}$ is a *dense* subclass of hom-$\Sigma\Pi\Sigma$ circuits if every hom-$\Sigma\Pi\Sigma$ circuit can be approximated "infinitesimally closely" by circuits in $\mathcal{C}$.[9] Unfortunately, depth-3 powering circuits and set-multilinear $\Sigma\Pi\Sigma$ circuits are *not* dense inside hom-$\Sigma\Pi\Sigma$ circuits.[10] On the other hand, *orbits of homogeneous sparse polynomials* form a dense subclass of hom-$\Sigma\Pi\Sigma$ circuits.[11] It is natural to ask:

Is MCSP for orbits of homogeneous sparse polynomials NP-hard?

MCSP for orbits of homogeneous sparse polynomials is exactly the ETsparse problem on inputs that are homogeneous polynomials. The second part of Theorem 2 answers the question positively over fields of characteristic 0.

Approximating the sparse-orbit complexity. Call the smallest s_0 such that f is in the orbit of an s_0-sparse polynomial, the *sparse-orbit complexity* of f. Theorem 2 shows that sparse-orbit complexity is hard to compute in the worst case.

Is sparse-orbit complexity easy to approximate?

In Theorem 8, we show that it is NP-hard to approximate the sparse-orbit complexity of a given s-sparse polynomial (homogeneous or not) to within a $s^{\frac{1}{3}-\epsilon}$ factor for any $\epsilon \in (0, 1/3)$. For s-sparse inputs, a within s factor approximation of the sparse-orbit complexity is trivial.

[9] Formally, a subclass $\mathcal{C}$ of hom-$\Sigma\Pi\Sigma$ circuits is *dense* if there are polynomial functions $p, q : \mathbb{N} \to \mathbb{N}$ such that the following holds: For $n, d, s \in \mathbb{N}$, the coefficient vector of every n-variate degree-d polynomial computable by a size-s hom-$\Sigma\Pi\Sigma$ circuit is in the *Zariski closure* of the set of coefficient vectors of $p(nds)$-variate degree-d polynomials computable by size-$q(nds)$ circuits in $\mathcal{C}$. Here, "size" means "top fan-in".

[10] Circuits of these two classes have small read-once algebraic branching programs (ROABPs), and the class ROABP is closed under Zariski closure [18]. So, the closures of these two classes are also contained inside ROABPs. But, there are explicit $O(n)$ size hom-$\Sigma\Pi\Sigma$ circuits that require $2^{\Omega(n)}$ size ROABPs [50, 38].

[11] Every n-variate degree-d size-s hom-$\Sigma\Pi\Sigma$ circuit is a linear projection of an s-sparse degree-d homogeneous polynomial in at most sd variables. It is well known that linear projections of f are contained in the Zariski closure of the orbit of f over fields of characteristic 0 (see [50] for a proof of this fact).

1.2 ET for constant-support polynomials

ET is efficiently solvable for two special sparse polynomial families, namely the power symmetric polynomial $\mathsf{PSym} := x_1^d + \ldots + x_n^d$ [36] and the sum-product polynomial $\mathsf{SP} := \sum_{i\in[s]} \prod_{j\in[d]} x_{i,j}$ [45, 36]. What makes ET easy for these sparse polynomials? Explanations were provided in [24, 8]: SP is a read-once formula; it is also a 1-design polynomial. PSym is a 1-design polynomial, but it is also a support-1 polynomial.

Is ET easy for constant-support polynomials?

In Theorem 16, we show that checking if a given f is in the orbit of a support-6 polynomial is NP-hard; this answers the question in the negative.

1.3 Our results

We now state our results formally. The ETsparse problem is defined as follows.

▶ **Problem 1** (ETsparse)**.** *Given a polynomial $f \in \mathbb{F}[\mathbf{x}]$ in its sparse representation and an $s \in \mathbb{Z}$, check if there exists an $A \in \mathrm{GL}(|\mathbf{x}|, \mathbb{F})$ and a $\mathbf{b} \in \mathbb{F}^{|\mathbf{x}|}$ such that $f(A\mathbf{x}+\mathbf{b})$ is s-sparse.*

Our first result, Theorem 2, shows the NP-hardness of ETsparse over any field.

▶ **Theorem 2** (ETsparse is NP-hard)**.**
1. *Let $\mathbb{F}$ be any field. There exists a deterministic polynomial-time many-one reduction from* 3-SAT *to* ETsparse *over $\mathbb{F}$.*
2. *Let* $\mathrm{char}(\mathbb{F}) = 0$*. There exists a deterministic polynomial-time many-one reduction from* 3-SAT *to* ETsparse *over $\mathbb{F}$ where the input polynomial in* ETsparse *is homogeneous.*

▶ Remark 3. The reduction is *natural*[12] and has the feature that a satisfying assignment can be mapped to a sparsifying invertible $A \in \{-1, 0, 1\}^{|\mathbf{x}|\times|\mathbf{x}|}$ and vice versa. So, ETsparse is NP-hard even when A is restricted to having only $\{-1, 0, 1\}$ entries.

▶ Remark 4. The authors of [15] showed the undecidability over $\mathbb{Z}$ of testing if a given f is shift equivalent to some sparse polynomial (f is shift equivalent to a polynomial g, if there exists a $\mathbf{b} \in \mathbb{F}^{|\mathbf{x}|}$ s.t $f = g(\mathbf{x}+\mathbf{b})$). However, their result does not imply the intractability of ETsparse as testing shift equivalence to a sparse polynomial is a special case of ETsparse when A is the identity map.

▶ Remark 5. Depth-3 power circuits, set-multilinear depth-3 circuits, and shifted sparse polynomials are all contained inside ROABPs. So, these models admit polynomial-time (improper) learning algorithms [9, 40] and quasi-polynomial-time hitting sets [1, 19]. Orbits of sparse polynomials require exponential size ROABPs [50]; we cannot expect to improperly learn them via ROABPs. Theorem 2 suggests that proper learning orbits of sparse polynomials is likely hard. Nonetheless, there is a quasi-polynomial time hitting set for orbits of sparse polynomials [45, 50].

▶ Remark 6. We believe that with some more effort, the second part of Theorem 2 can be proven over fields of finite characteristics as well. See the last remark in Section 3.4.
We prove Theorem 2 in Section 3. Next, we define the gap version of ETsparse.

▶ **Problem 7** (α-gap-ETsparse)**.** *Let $\alpha > 1$ be a parameter. Given a polynomial $f \in \mathbb{F}[\mathbf{x}]$ in its sparse representation and an integer s_0, output:*
- *YES, if there exist an $A \in \mathrm{GL}(|\mathbf{x}|, \mathbb{F})$ and $\mathbf{b} \in \mathbb{F}$ such that $f(A\mathbf{x}+\mathbf{b})$ is s_0-sparse.*
- *NO, if for all $A \in \mathrm{GL}(|\mathbf{x}|, \mathbb{F})$ and $\mathbf{b} \in \mathbb{F}$, $f(A\mathbf{x}+\mathbf{b})$ has sparsity at least αs_0.*

[12] unless $\mathrm{char}(\mathbb{F}) = 2$. See the full version for details.

Our second result, Theorem 8, shows that α-gap-ETsparse is NP-hard for $\alpha = s^{\frac{1}{3}-\epsilon}$, where s is the sparsity of the input polynomial f and $\epsilon \in (0, \frac{1}{3})$ is an arbitrary constant. Theorem 8 is proven in Section 4. From Theorem 8, we get Corollary 11 which states that $s^{\frac{1}{3}-\epsilon}$ factor approximation of the sparse-orbit complexity of an s-sparse polynomial is NP-hard.

▶ **Theorem 8** ($s^{\frac{1}{3}}$-gap-ETsparse is NP-hard). *Let $\epsilon \in (0, \frac{1}{3})$ be an arbitrary constant.*

1. *Let $\mathbb{F}$ be any field. There exists a deterministic polynomial-time many-one reduction from* 3-SAT *to* $s^{\frac{1}{3}-\epsilon}$-gap-ETsparse *over $\mathbb{F}$ where the input polynomial in* $s^{\frac{1}{3}-\epsilon}$-gap-ETsparse *is s-sparse.*
2. *Let* $\mathrm{char}(\mathbb{F}) = 0$. *There exists a deterministic polynomial-time many-one reduction from* 3-SAT *to* $s^{\frac{1}{3}-\epsilon}$-gap-ETsparse *over $\mathbb{F}$ where the input polynomial in* $s^{\frac{1}{3}-\epsilon}$-gap-ETsparse *is homogeneous and s-sparse.*

▶ Remark 9. With a more careful analysis, the constant $\frac{1}{3}$ in $s^{\frac{1}{3}-\epsilon}$ may be improved.

▶ Remark 10. Interestingly, the above results are obtained without invoking the celebrated PCP theorem [7, 6, 17].

▶ **Corollary 11.** *Let $0 < \epsilon < \frac{1}{3}$ be an arbitrary constant.*

1. *Let $\mathbb{F}$ be any field. It is NP-hard to compute $s^{\frac{1}{3}-\epsilon}$ factor approximation of the sparse-orbit complexity when the input is an s-sparse polynomial over $\mathbb{F}$.*
2. *Let* $\mathrm{char}(\mathbb{F}) = 0$. *It is NP-hard to compute $s^{\frac{1}{3}-\epsilon}$ factor approximation of the sparse-orbit complexity when the input is an s-sparse homogeneous polynomial over $\mathbb{F}$.*

Now, we formally define the support of a polynomial.

▶ **Definition 12** (Support of a polynomial). *For a monomial $\mathbf{x}^{\boldsymbol{\alpha}}$, where $\boldsymbol{\alpha}$ is the exponent vector, the support of $\mathbf{x}^{\boldsymbol{\alpha}}$,* $\mathrm{Supp}(\mathbf{x}^{\boldsymbol{\alpha}})$*, is the number of variables with non-zero exponent. The support of a polynomial f,* $\mathrm{Supp}(f)$*, is the maximum support size over all the monomials of f.*

Thus, a polynomial has support σ if there exists a monomial with support σ and no other monomial has support $> \sigma$. The ET problem for constant-support polynomials and a stronger version of it are defined next (henceforth, σ is assumed to be a constant).

▶ **Problem 13** (ETsupport). *Given a polynomial $f \in \mathbb{F}[\mathbf{x}]$ in its sparse representation and an integer σ, check if there exists an $A \in \mathrm{GL}(|\mathbf{x}|, \mathbb{F})$ such that* $\mathrm{Supp}(f(A\mathbf{x})) \leq \sigma$.

▶ **Problem 14** ($(\sigma+1)$-to-σ ETsupport). *Given a polynomial $f \in \mathbb{F}[\mathbf{x}]$ with support $\sigma+1$ in its sparse representation, check if there exists an $A \in \mathrm{GL}(|\mathbf{x}|, \mathbb{F})$ such that* $\mathrm{Supp}(f(A\mathbf{x})) \leq \sigma$.

▶ Remark 15. Unlike ETsparse, checking if f is in the *orbit* of a constant-support polynomial is the same as checking if f is equivalent to a constant-support polynomial. This follows from the observation that $\mathrm{Supp}(f(\mathbf{x})) = \mathrm{Supp}(f(\mathbf{x}+\mathbf{b}))$ for any $\mathbf{b} \in \mathbb{F}^{|\mathbf{x}|}$.

Our third and last result, Theorem 16, shows that ETsupport and $(\sigma+1)$-to-σ ETsupport are NP-hard. We prove Theorem 16 in Section 5.

▶ **Theorem 16** (ETsupport is NP-hard). *Let $\sigma \geq 6$ be a constant and $\mathbb{F}$ be a field with* $\mathrm{char}(\mathbb{F}) = 0$ *or* $> \sigma + 1$. *There is a deterministic polynomial-time many-one reduction from* 3-SAT *to* ETsupport *over $\mathbb{F}$. In particular,* 3-SAT *reduces to* $(\sigma+1)$-to-σ ETsupport *in deterministic polynomial time.*

▶ Remark 17. Over fields of finite characteristic, it is assumed that the exponent vectors corresponding to the monomials of the input polynomial are given in binary.

We prove Theorems 2, 8 and 16 by direct reductions from 3-SAT, and at the beginning of Sections 3, 4 and 5, we give proof sketches of the respective reductions.

2 Preliminaries

2.1 Definitions and notations

For $n, a, b \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2 \ldots, n\}$ and $[a, b]$ denotes the integers from a to b, both inclusive. A polynomial is *homogeneous* if all its monomials have the same total degree. The set of invertible linear transforms in n variables over a field $\mathbb{F}$ is denoted by $\mathrm{GL}(n, \mathbb{F})$. For a polynomial $f \in \mathbb{F}[\mathbf{x}]$, the action of a linear transform $A \in \mathbb{F}^{|\mathbf{x}|\times|\mathbf{x}|}$ on its variables is denoted by $f(A\mathbf{x})$ as well as by $A(f)$. The sparsity of a polynomial f, denoted as $\mathcal{S}(f)$, is the number of monomials in f with non-zero coefficients. For a polynomial f, $\mathrm{var}(f)$ denotes the set of variables that occur in at least one monomial of f. We have used the notation $f \sim g$ earlier to denote $f = g(A\mathbf{x} + \mathbf{b})$. Henceforth, we will ignore the translation vector $\mathbf{b}$ in the main body of the discussion for simplicity but mention the necessary changes in the proofs or point to appropriate sections when translations are involved. Thus, for polynomials f and g, $f \sim g$ will mean $f(\mathbf{x}) = g(A\mathbf{x})$ where $A \in \mathrm{GL}(|\mathbf{x}|, \mathbb{F})$.[13] Similarly, the orbit of a polynomial f will now denote the set $\{f(A\mathbf{x}), A \in \mathrm{GL}(|\mathbf{x}|, \mathbb{F})\}$.

▶ **Definition 18** (Degree separated polynomials)**.** *Polynomials f and g are **degree separated** if no monomial of f has the same degree as a monomial of g. Similarly, f and g are **degree separated with respect to a variable** x if no monomial of f has the same x-degree as a monomial of g.*[14]

2.2 Algebraic preliminaries

The proofs of the observations and claims in this section can be found in the full version.

▶ **Observation 19.** *Let f and g be polynomials such that $f \sim g$. Then, f and g have the same set of degrees*[15] *for the monomials. Thus, if f and g are degree separated, then $f \not\sim g$.*

▶ **Observation 20.** *If f and g are degree separated (or degree separated with respect to some variable), then $\mathcal{S}(f + g) = \mathcal{S}(f) + \mathcal{S}(g)$.*

▶ **Observation 21.** *If f and g are degree separated, $f_1 \sim f$ and $g_1 \sim g$, then $\mathcal{S}(f_1 + g_1) = \mathcal{S}(f_1) + \mathcal{S}(g_1)$.*

Observation 22 analyzes the sparsity of powers of linear forms. Observation 23 is a special case of Observation 22 and is stated separately because it is simpler and is invoked many times. Observation 24 analyzes the sparsity of powers of affine forms.

▶ **Observation 22.** *Let ℓ be a m-variate linear form*[16] *and $d \in \mathbb{N}$. If $\mathrm{char}(\mathbb{F}) = 0$, $\mathcal{S}(\ell^d) = \binom{d+m-1}{m-1}$ and if $\mathrm{char}(\mathbb{F}) = p$, $\mathcal{S}(\ell^d) = \prod_{i=0}^{k} \binom{e_i+m-1}{m-1}$, where $d = \sum_{i=0}^{k} e_i p^i$, $e_i \in [0, p-1]$.*

▶ **Observation 23.** *If $\mathrm{char}(\mathbb{F}) = 0$ and ℓ is a linear form in exactly two variables, then $\mathcal{S}(\ell^d) = d+1$. The result holds for $\mathrm{char}(\mathbb{F}) = p$ fields if $p > d$ or if $d = p^k - 1$ for some $k \in \mathbb{N}$. Further, if ℓ is a linear form in at least two variables and d is as before, then $\mathcal{S}(\ell^d) \geq d + 1$.*

---

[13] Note $\sim$ is an equivalence relation under this definition.

[14] The degree of a monomial means its total degree and the degree of a polynomial f is the maximum degree amongst all monomials in f. The x-degree of a monomial is the degree of the variable x in the monomial.

[15] The set of degrees is the set of distinct degrees of all the monomials in the polynomial. For example, the set of degrees of $f(x_1, x_2) = x_1^2 + x_1x_2 + 4x_2$ is $\{2, 1\}$.

[16] A linear form is a homogeneous degree one polynomial. An affine form is a degree one polynomial.

▶ **Observation 24.** *Let $h = \ell + c_0$, where ℓ is a linear form in at least one variable and $c_0 \in \mathbb{F}\setminus\{0\}$, then $\mathcal{S}(h^d) \geq \mathcal{S}(\ell^d) + 1$. More precisely, $\mathcal{S}(h^d) \geq d + 1$ holds if $\mathrm{char}(\mathbb{F}) = 0$ or if $\mathrm{char}(\mathbb{F}) = p$ and $p > d$ or $d = p^k - 1$ for some $k \in \mathbb{N}$.*

Claim 25 analyzes the sparsity of polynomials divisible by a power of some linear form in at least two variables and is used to prove part two of Theorems 2 and 8. Claim 26 analyzes the support of monomials under invertible linear transforms and is used to prove Theorem 16.

▷ Claim 25. Let $\mathrm{char}(\mathbb{F}) = 0$. If $f \in \mathbb{F}[\mathbf{x}]$ is a non-zero polynomial divisible by ℓ^d for some linear form ℓ in at least two variables, then $\mathcal{S}(f) \geq d + 1$.

▷ Claim 26. Let $\sigma, d, n \in \mathbb{N}$, $d \geq \sigma$, $f = (x_1 \cdots x_n)^d$, and $\ell_1, \ldots, \ell_n$ be linearly independent linear forms in $x_1, \ldots, x_n$. If $|\cup_{i=1}^n \mathrm{var}(\ell_i)| \geq \sigma$ and $g := f(\ell_1 \cdots \ell_n)$, then $\mathrm{Supp}(g) \geq \sigma$. The claim holds if $\mathrm{char}(\mathbb{F}) = 0$, or $\mathrm{char}(\mathbb{F}) = p$ with $p > d$, or $p > \sigma$ and $d = p^k - 1$ for some $k \in \mathbb{N}$.

3 NP-hardness of ETsparse

In this section, we prove Theorem 2 over $\mathrm{char}(\mathbb{F}) = 0$ fields without translations for easy understanding.[17] The full version contains the proofs of the lemmas and the observations in this section, and the reduction over $\mathrm{char}(\mathbb{F}) > 0$ fields and also when translations are allowed.

Proof sketch. The reduction maps each variable and clause of a 3-CNF[18] ψ to distinct degree separated polynomials which, summed together, give the polynomial f. As the summands are degree separated, the sparsity of f under invertible transforms can be analyzed by doing so for individual polynomials. The degrees are chosen such that f is equivalent to an s-sparse polynomial (for a suitable sparsity parameter s) if and only if $\psi \in$ 3-SAT.

3.1 Constructing f and s

Let ψ be a 3-CNF in variables $\mathbf{x} := \{x_1, x_2 \ldots x_n\}$ and m clauses:

$$\psi = \wedge_{k=1}^{m} \vee_{j \in C_k} (x_j \oplus a_{k,j}),$$

where C_k denotes the set of indices of the variables in the k^{th} clause and $a_{k,j} \in \{0, 1\}$. Let $\mathbf{y} := \{y_1, y_2 \ldots y_n\}$, x_0 be a new variable and $\mathbf{z} := \{x_0\} \sqcup \mathbf{x} \sqcup \mathbf{y}$. For $d_1, d_2, d_3, d_4 \in \mathbb{N}$, consider the following polynomials:

- Corresponding to variable x_i, where $i \in [n]$, define $Q_i(\mathbf{z})$ as:

$$Q_i(\mathbf{z}) := Q_{i,1}(\mathbf{z}) + Q_{i,2}(\mathbf{z}) + Q_{i,3}(\mathbf{z}), \text{ where}$$
$$Q_{i,1}(\mathbf{z}) := x_0^{(3i-2)d_1} x_i^{d_2},\ Q_{i,2}(\mathbf{z}) := x_0^{(3i-1)d_1}(y_i + x_i)^{d_3} \text{ and } Q_{i,3}(\mathbf{z}) := x_0^{3id_1}(y_i - x_i)^{d_3}.$$

- For the k^{th} clause, $k \in [m]$, define $R_k(\mathbf{z}) := x_0^{(3n+k)d_1} \prod_{j \in C_k} (y_j + (-1)^{a_{k,j}} x_j)^{d_4}$.

[17] Note that for f and g two homogeneous polynomials, $f(\mathbf{x}) = g(A\mathbf{x} + \mathbf{b})$ implies $f(\mathbf{x}) = g(A\mathbf{x})$, where $A \in \mathrm{GL}(|\mathbf{x}|, \mathbb{F})$ and $\mathbf{b} \in \mathbb{F}^{|\mathbf{x}|}$. Hence, it suffices to prove part 2 of Theorem 2 without translations.

[18] We assume, without loss of generality, that each clause of a 3-CNF has 3 distinct variables. This can be achieved by introducing extra variables for clauses with < 3 variables.

Define $s := 1 + n(3 + d_3) + m(d_4 + 1)^2$ and the polynomial f as:

$$f(\mathbf{z}) := x_0^{d_1} + \sum_{i=1}^{n} Q_i(\mathbf{z}) + \sum_{k=1}^{m} R_k(\mathbf{z}). \tag{1}$$

The following conditions are imposed on the d_i's:

$$d_1 \geq \max(s, d_2 + 1),\ d_2 \geq 2d_3,\ d_3 \geq m(d_4 + 1)^2 + 1,\ \text{and } d_4 \geq m. \tag{2}$$

Set $d_4 := m$, $d_3 := m(m+1)^2 + 1 = O(m^3)$, $d_2 = 2m(m+1)^2 + 2 = O(m^3)$ and $d_1 = 1 + n(4 + m(m+1)^2) + m(m+1)^2 = O(nm^3)$. Then $s = O(nm^3)$ and the d_i's satisfy the conditions of (2), under which the following observations hold.

▶ **Observation 27.** *For $i \in [n]$, $k \in [m]$, the polynomials $x_0^{d_1}$, $Q_{i,1}(\mathbf{z})$, $Q_{i,2}(\mathbf{z})$, $Q_{i,3}(\mathbf{z})$ and $R_k(\mathbf{z})$ are degree separated from one another. Also, $Q_i(\mathbf{z})$ is degree separated from $Q_j(\mathbf{z})$, for $i, j \in [n]$ and $i \neq j$. Similarly, $R_k(\mathbf{z})$ is degree separated from $R_l(\mathbf{z})$ for $k, l \in [m]$ and $k \neq l$.*

▶ **Observation 28.** *The degree of f is $(3n + m)d_1 + 3d_4 = (mn)^{O(1)}$, $\mathcal{S}(f(\mathbf{z})) = 1 + n(2d_3 + 3) + m(d_4 + 1)^3$ and $\mathrm{Supp}(f) = 7$.*

3.2 The forward direction

Proposition 29 shows how a satisfiable ψ implies the existence of an invertible A, such that $\mathcal{S}(f(A\mathbf{z})) \leq s$ by constructing A from a satisfying assignment $\mathbf{u} \in \{0,1\}^n$ of ψ.

▶ **Proposition 29.** *Let $\mathbf{u} \in \{0,1\}^n$ be such that $\psi(\mathbf{u}) = 1$. Then $\mathcal{S}(f(A\mathbf{z})) \leq s$ for A as:*

$$A : x_0 \mapsto x_0, x_i \mapsto x_i, y_i \mapsto y_i + (-1)^{u_i} x_i, \quad \forall i \in [n]. \tag{3}$$

Proof. It follows from the definition of f in (1), Observations 27 and 21 that

$$\mathcal{S}(f(A\mathbf{z})) = \mathcal{S}(A(x_0^{d_1})) + \sum_{i=1}^{n} \mathcal{S}(Q_i(A\mathbf{z})) + \sum_{k=1}^{m} \mathcal{S}(R_k(A\mathbf{z})).$$

Thus, it suffices to analyze the sparsity of $A(x_0^{d_1}), Q_i(A\mathbf{z})$'s and $R_k(A\mathbf{z})$'s. Now, $\mathcal{S}(A(x_0^{d_1})) = 1$ as $A(x_0^{d_1}) = x_0^{d_1}$. We now analyze $\mathcal{S}(Q_i(A\mathbf{z}))$ for $i \in [n]$. If $u_i = 0$, then

$$Q_{i,1}(A\mathbf{z}) = x_0^{(3i-2)d_1} x_i^{d_2},\ Q_{i,2}(A\mathbf{z}) = x_0^{(3i-1)d_1}(y_i + 2x_i)^{d_3} \text{ and } Q_{i,3}(A\mathbf{z}) = x_0^{3id_1} y_i^{d_3}.$$

If $u_i = 1$, then

$$Q_{i,1}(A\mathbf{z}) = x_0^{(3i-2)d_1} x_i^{d_2},\ Q_{i,2}(A\mathbf{z}) = x_0^{(3i-1)d_1} y_i^{d_3} \text{ and } Q_{i,3}(A\mathbf{z}) = x_0^{3id_1}(y_i - 2x_i)^{d_3}.$$

By Observation 23 (for linear forms in two variables over $\mathrm{char}(\mathbb{F}) = 0$ fields), if $u_i = 0$ then $\mathcal{S}(Q_{i,2}(A\mathbf{z})) = d_3 + 1$ and $\mathcal{S}(Q_{i,3}(A\mathbf{z})) = 1$ and, if $u_i = 1$ then $\mathcal{S}(Q_{i,2}(A\mathbf{z})) = 1$ and $\mathcal{S}(Q_{i,3}(A\mathbf{z})) = d_3 + 1$. In either case, by Observations 27 and 21,

$$\mathcal{S}(Q_i(A\mathbf{z})) = \mathcal{S}(Q_{i,1}(A\mathbf{z})) + \mathcal{S}(Q_{i,2}(A\mathbf{z})) + \mathcal{S}(Q_{i,3}(A\mathbf{z})) = d_3 + 3.$$

For the k^{th} clause, $k \in [m]$, the action of A on the corresponding polynomial R_k is:

$$R_k(A\mathbf{z}) = x_0^{(3n+k)d_1} \prod_{j \in C_k} (y_j + ((-1)^{a_{k,j}} + (-1)^{u_j}) x_j)^{d_4}.$$

As the multiplicands in $R_k(A\mathbf{z})$ do not share any variables, $\mathcal{S}(R_k(A\mathbf{z}))$ is the product of the sparsity of the multiplicands. Since $\psi(\mathbf{u}) = 1$, therefore in the k^{th} clause there exists $j \in C_k$ such that $a_{k,j} \neq u_j$. For that j, $(y_j + ((-1)^{a_{k,j}} + (-1)^{u_j})x_j)^{d_4} = y_j^{d_4}$. As at least one literal is true in every clause under $\mathbf{u}$, $\mathcal{S}(R_k(A\mathbf{z})) \leq (d_4+1)^2$ using Observation 23. Thus,

$$\begin{aligned}\mathcal{S}(f(A\mathbf{z})) &= \mathcal{S}(A(x_0^{d_1})) + \sum_{i=1}^{n} \mathcal{S}(Q_i(A\mathbf{z})) + \sum_{k=1}^{m} \mathcal{S}(R_k(A\mathbf{z})) \\ &\leq 1 + n(d_3+3) + m(d_4+1)^2 = s.\end{aligned}$$

◀

3.3 The reverse direction

Now, we show that $(f, s) \in$ ETsparse implies $\psi \in$ 3-SAT by showing that the permuted and scaled versions of the transform of (3) form all the viable sparsifying invertible linear transforms. This is where the constraints on the d_i's are used. So, let $A \in \mathrm{GL}(|\mathbf{z}|, \mathbb{F})$ be such that $\mathcal{S}(f(A\mathbf{z})) \leq s$. Lemma 30 shows that $A(x_0)$ is just a variable by leveraging $d_1 \geq s$.

▶ **Lemma 30.** *Without loss of generality,* $A(x_0) = x_0$.

The proof of Lemma 31 uses $d_2 \geq 2d_3$ while that of Lemma 32 uses $d_3 \geq m(d_4+1)^2 + 1$.

▶ **Lemma 31.** *For any invertible* A *and* $i \in [n]$*:*

$$\mathcal{S}(Q_i(A\mathbf{z})) = \mathcal{S}(Q_{i,1}(A\mathbf{z})) + \mathcal{S}(Q_{i,2}(A\mathbf{z})) + \mathcal{S}(Q_{i,3}(A\mathbf{z})) \geq d_3 + 3,$$

where Q_i, $Q_{i,1}$, $Q_{i,2}$ *and* $Q_{i,3}$ *are as defined in Section 3.1. Equality holds if and only if* $A(x_i) = X_i$ *and* $A(y_i) = Y_i + (-1)^{u_i} X_i$ *for some scaled variables* $X_i, Y_i \in \mathbf{z}$ *and* $u_i \in \{0,1\}$*. Further, if* $\mathcal{S}(Q_i(A\mathbf{z})) \neq d_3 + 3$*, then* $\mathcal{S}(Q_i(A\mathbf{z})) \geq 2d_3 + 3$.

▶ **Lemma 32.** *Under the given* A, $\mathcal{S}(Q_i(A\mathbf{z})) = d_3 + 3$ *holds for all* $i \in [n]$.

Lemmas 30, 31 and 32 together show that A is a permuted scaled version of the transform of (3). We can assume A to be as described in (3) without loss of generality as permutation and non-zero scaling of variables do not affect the sparsity of a polynomial. Proposition 33 shows how a satisfying assignment can be formed from A using $d_4 \geq m$.

▶ **Proposition 33.** *For* A *as given in* (3), $\mathbf{u} = (u_1, \ldots, u_n)$ *is a satisfying assignment for* ψ.

Proof. Suppose not; then there exists $k \in [m]$ such that the k^{th} clause, $\vee_{j \in C_k}(x_j \oplus a_{k,j})$, in ψ is unsatisfied. Since this clause is unsatisfied, $u_j = a_{k,j}$ for all $j \in C_k$. Thus, $R_k(A\mathbf{z}) = x_0^{(3n+k)d_1} \prod_{j \in C_k}(y_j \pm 2x_j)^{d_4}$, where R_k is as defined in Section 3.1, and $\mathcal{S}(R_k(A\mathbf{z})) = (d_4+1)^3 \geq (m+1)(d_4+1)^2$ by Observation 23, the fact that $R_k(A\mathbf{z})$ is a product of linear forms not sharing variables, and the condition $d_4 \geq m$. By the definition of f and s in Section 3.1, Observations 27 and 21, it holds that

$$\begin{aligned}\mathcal{S}(f(A\mathbf{z})) &\geq \mathcal{S}(A(x_0)^{d_1}) + \sum_{i=1}^{n} \mathcal{S}(Q_i(A\mathbf{z})) + \mathcal{S}(R_k(A\mathbf{z})) \\ &\geq 1 + n(3+d_3) + m(d_4+1)^2 + (d_4+1)^2 = s + (d_4+1)^2 > s,\end{aligned}$$

a contradiction. Thus, $\mathbf{u}$ is a satisfying assignment for ψ. ◀

3.4 Homogeneous case: Proof of Part 2 of Theorem 2

We show a modification of the construction in Section 3.1 which, along with arguments similar to those in Sections 3.2 and 3.3, can be used to prove Theorem 2 for homogeneous polynomials over $\text{char}(\mathbb{F}) = 0$ fields. Because the polynomials are homogeneous, we cannot use degree separation like in the non-homogeneous case. Instead, we introduce a new variable y_0 and redefine $Q_i(\mathbf{z})$ and $R_k(\mathbf{z})$ of Section 3.1 so that:

1. Each polynomial is homogeneous with the same degree and is divisible by $x_0^{d_1}$ and $y_0^{d_1}$.
2. Each polynomial has distinct y_0-degree and if P_1 and P_2 are polynomials where P_1 has a higher y_0-degree than P_2, then the y_0-degree of P_1 is greater than the degree of any variable (except possibly x_0) in P_2.

The divisibility condition ensures that both x_0 and y_0 map to scaled variables (see Lemma 37 and its proof), and the second condition induces a degree separation of the polynomials with respect to y_0 (see Observation 34 and Lemma 38). Formally, let x_0, $\mathbf{x}$ and $\mathbf{y}$ be as defined in Section 3.1 and y_0 be a new variable. Define $\mathbf{z} := \mathbf{x} \sqcup \mathbf{y} \sqcup \{x_0\} \sqcup \{y_0\}$. Let $d_1, d_2, d_3, d_4 \in \mathbb{N}$. Consider the following polynomials:

1. For each variable x_i, $i \in [n]$, let $Q_i(\mathbf{z}) := Q_{i,1}(\mathbf{z}) + Q_{i,2}(\mathbf{z}) + Q_{i,3}(\mathbf{z})$, where

$$Q_{i,1}(\mathbf{z}) := x_0^{d_1(3n-3i+6)-d_2} y_0^{d_1(3i+m-2)} x_i^{d_2},\ Q_{i,2}(\mathbf{z}) := x_0^{d_1(3n-3i+5)-d_3} y_0^{d_1(3i+m-1)} (y_i + x_i)^{d_3}$$
$$Q_{i,3}(\mathbf{z}) := x_0^{d_1(3n-3i+4)-d_3} y_0^{d_1(3i+m)} (y_i - x_i)^{d_3}.$$

2. For the k^{th} clause, $k \in [m]$, let $R_k(\mathbf{z}) := x_0^{d_1(3n+m+4-k)-3d_4} y_0^{d_1 k} \prod_{j \in C_k} (y_j + (-1)^{a_{k,j}} x_j)^{d_4}$.

Define $s := 1 + n(d_3 + 3) + m(d_4 + 1)^2$ as before and impose the conditions of (2) on the d_i's. Using the conditions on the d_i's, it is easy to verify that the individual degrees of x_0 and y_0 in every polynomial defined above is at least d_1. Define f as:

$$f(\mathbf{z}) := x_0^{3d_1} y_0^{d_1(3n+m+1)} + \sum_{i=1}^{n} Q_i(\mathbf{z}) + \sum_{k=1}^{m} R_k(\mathbf{z}). \tag{4}$$

Clearly, f is a homogeneous polynomial of degree $(3n + m + 4)d_1$ and is divisible by $x_0^{d_1}$ and $y_0^{d_1}$. Further, we have the following observations under the constraints of (2).

▶ **Observation 34.** *For all $i \in [n]$, $k \in [m]$, the polynomials $x_0^{3d_1} y_0^{d_1(3n+m+1)}$, $Q_{i,1}(\mathbf{z})$, $Q_{i,2}(\mathbf{z})$, $Q_{i,3}(\mathbf{z})$ and $R_k(\mathbf{z})$ are degree separated with respect to y_0 from one another. Also, $Q_i(\mathbf{z})$ is degree separated with respect to y_0 from other $Q_j(\mathbf{z})$'s, for $i, j \in [n]$ and $i \neq j$. Similarly, $R_k(\mathbf{z})$ is degree separated with respect to y_0 from $R_l(\mathbf{z})$ for $k, l \in [m]$ and $k \neq l$.*

▶ **Observation 35.** $\mathcal{S}(f(\mathbf{z})) = 1 + n(2d_3 + 3) + m(d_4 + 1)^3$ *and* $\text{Supp}(f) = 8$.

The forward direction. Let $\mathbf{u} \in \{0,1\}^n$ be such that $\psi(\mathbf{u}) = 1$ and f, as described in (4), be the polynomial corresponding to ψ. Proposition 36 shows how to construct a sparsifying transform from $\mathbf{u}$. The proof of Proposition 36 is very similar to that of Proposition 29.

▶ **Proposition 36.** $\mathcal{S}(f(A\mathbf{z})) \leq s$ *where* $A \in \text{GL}(|\mathbf{z}|, \mathbb{F})$ *is as follows:*

$$A : y_0 \mapsto y_0,\ x_0 \mapsto x_0,\ x_i \mapsto x_i,\ y_i \mapsto y_i + (-1)^{u_i} x_i \quad i \in [n]. \tag{5}$$

The reverse direction. Let $\mathcal{S}(f(A\mathbf{z})) \leq s$ for some $A \in \text{GL}(|\mathbf{z}|, \mathbb{F})$. Lemma 37, the proof of which requires Claim 25, shows that $A(x_0)$ and $A(y_0)$ have only one variable. With this, Lemma 38 shows that the summands of $f(A\mathbf{z})$ must be degree separated with respect to y_0.

▶ **Lemma 37.** *Without loss of generality,* $A(x_0) = x_0$ *and* $A(y_0) = y_0$.

▶ **Lemma 38.** *For all* $i \in [n]$, $k \in [m]$, *the polynomials* $x_0^{3d_1} y_0^{d_1(3n+m+1)}$, $Q_{i,1}(A\mathbf{z})$, $Q_{i,2}(A\mathbf{z})$, $Q_{i,3}(A\mathbf{z})$ *and* $R_k(A\mathbf{z})$ *are degree separated from one another with respect to* y_0. *Also,* $Q_i(A\mathbf{z})$ *is degree separated with respect to* y_0 *from other* $Q_j(A\mathbf{z})$*'s, for* $i, j \in [n]$ *and* $i \neq j$. *Similarly,* $R_k(A\mathbf{z})$ *is degree separated with respect to* y_0 *from* $R_l(A\mathbf{z})$ *for* $k, l \in [m]$ *and* $k \neq l$.

$$\therefore \quad \mathcal{S}(f(A\mathbf{z})) = \mathcal{S}(x_0^{3d_1} y_0^{d_1(3n+m+1)}) + \sum_{i=1}^{n} \mathcal{S}(Q_i(A\mathbf{z})) + \sum_{k=1}^{m} \mathcal{S}(R_k(A\mathbf{z})), \text{ by Lemma 38.}$$

Lemmas 31 and 32 then hold with slight modification to their proofs, which, along with Lemma 37, show that A is a permuted scaled version of the transform of (5). Proposition 39 then holds and can be proven similarly to Proposition 33.

▶ **Proposition 39.** *For* A *as given in* (5), $\mathbf{u} = (u_1, \ldots, u_n)$ *is a satisfying assignment for* ψ.

▶ Remark 40. In the definition of f in Section 3.1 and this section, an extra summand is present besides Q_i's and R_k's. If $\text{char}(\mathbb{F}) = 0$, we can drop the summand by using Claim 25 and suitably modifying f, the current parameters and arguments to make the reduction work. We preserve the extra summand here for two reasons: One, for ease of understanding, because the definition of f in (1) is similar to that in (4). Two, in the non-homogeneous case, the extra summand proves useful in showing the reduction over finite characteristic fields, where Claim 25 does not hold, and thus it may also prove useful in showing the reduction for homogeneous polynomials over such fields.

▶ Remark 41. A feature of the reduction is that we can assume that the output polynomial is of the form $w^D f(\mathbf{z})$, where $w \notin \mathbf{z}$. This can be achieved by multiplying the output polynomial f of the current reduction by w^D, where D is greater than the sparsity parameter s in the reduction. If a proof of NP-hardness of hom-ΣΠΣ-MCSP has this feature, then it would imply NP-hardness of ΣΠΣ-MCSP (via a homogenization trick).

▶ Remark 42. We believe Claim 25 (used to prove Lemma 37) can be modified for finite characteristic fields, using which the argument in this section can be extended to such fields.

4 NP-hardness of α-gap-ETsparse

In this section, we prove parts 1 and 2 of Theorem 8 over $\text{char}(\mathbb{F}) = 0$ fields when no translations are involved. The full version contains the proofs of the lemmas in this section and that of the finite characteristic case with translations allowed for part 1 of Theorem 8.

Proof sketch. For a 3-CNF ψ, we carefully analyze the sparsity of the corresponding polynomial f, as defined in (1) with the constraints of (2) and (6), under all $A \in \text{GL}(|\mathbf{z}|, \mathbb{F})$. For $\psi \in \overline{\text{3-SAT}}$, Lemma 43 shows lower bounds on $\mathcal{S}(f(A\mathbf{z}))$ for any $A \in \text{GL}(|\mathbf{z}|, \mathbb{F})$. For $\psi \in$ 3-SAT, by Proposition 29, there exists $A \in \text{GL}(|\mathbf{z}|, \mathbb{F})$ such that $\mathcal{S}(f(A\mathbf{z})) \leq s_0 := 1 + n(d_3 + 3) + m(d_4 + 1)^2$. Proposition 44 compares the sparsities for satisfiable and unsatisfiable ψ's and shows α-gap-ETsparse is NP-hard using Lemma 43 and the conditions in (6).

$$d_1 = d_2 + 1, \; d_2 = d_3^2 + 1, \; d_3 = m(d_4 + 1)^2 + 1, \; d_4 \geq 4mn. \tag{6}$$

▶ **Lemma 43.** *Let $\psi \in \overline{\text{3-SAT}}$, f be as defined in (1) corresponding to ψ and $A \in \text{GL}(|\mathbf{z}|, \mathbb{F})$.*
1. *If $A(x_0)$ is a linear form in at least 2 variables, $\mathcal{S}(f(A\mathbf{z})) \geq d_1 + 1$.*
2. *If A is not as in item 1 and $A(x_j)$ is a linear form in at least 2 variables for some $j \in [n]$, then $\mathcal{S}(f(A\mathbf{z})) \geq d_2 + 1$.*
3. *If A is not as in items 1 and 2 and $\mathcal{S}(A(y_j + x_j)) \geq 3$ or $\mathcal{S}(A(y_j - x_j)) \geq 3$ for some $j \in [n]$, then $\mathcal{S}(f(A\mathbf{z})) \geq \frac{d_3^2+3d_3+2}{2}$.*
4. *If A is not of the form described in the previous three cases, then $\mathcal{S}(f(A\mathbf{z})) \geq (d_4+1)^3$.*

▶ **Proposition 44.** *α-gap-ETsparse is NP-hard for s-sparse polynomial inputs over $\mathbb{F}$ and $\alpha = s^{1/3-\epsilon}$, where $\epsilon \in (0, 1/3)$ is an arbitrary constant.*

Proof. If $\psi \in$ 3-SAT, then $\mathcal{S}(f(A\mathbf{z})) \leq s_0$ where A is as described in (3). If $\psi \in \overline{\text{3-SAT}}$, then it follows from Lemma 43 that for any $A \in \text{GL}(|\mathbf{z}|, \mathbb{F})$:

$$\mathcal{S}(f(A\mathbf{z})) \geq \min\left(d_1 + 1, d_2 + 1, \frac{d_3^2 + 3d_3 + 2}{2}, (d_4+1)^3\right).$$

The constraints imposed in (6) ensure that $(d_4+1)^3$ is the minimum. As $d_3 = m(d_4+1)^2+1$, therefore $s_0 = 1 + n(d_3+3) + m(d_4+1)^2 \leq 3nd_3 = 3mn(d_4+1)^2 + 3n \leq 4mn(d_4+1)^2$. Thus, the gap in the sparsities of the YES instances and the NO instances is

$$\frac{(d_4+1)^3}{s_0} \geq \frac{(d_4+1)^3}{4mn(d_4+1)^2} = \frac{d_4+1}{4mn}.$$

Also, as $d_4 \geq 4mn$, $\mathcal{S}(f) = s \leq 2m(d_4+1)^3 \implies d_4 + 1 \geq (\frac{s}{2m})^{1/3}$. Then, the gap is

$$\frac{(d_4+1)^3}{s_0} \geq \frac{d_4+1}{4mn} \geq \frac{s^{1/3}}{2^{1/3}4m^{4/3}n}.$$

Let $\epsilon \in (0, 1/3)$ be an arbitrary constant. The parameter d_4, which determines s, can be chosen a sufficiently large polynomial function in m and n such that $2^{1/3}4m^{4/3}n \leq s^{\epsilon}$. Hence, the gap is at least $s^{1/3-\epsilon}$. Thus, 3-SAT reduces to α-gap-ETsparse for $\alpha = s^{1/3-\epsilon}$. ◀

Homogeneous polynomials over $\text{char}(\mathbb{F}) = 0$ fields. We now consider the polynomial f as defined in (4) for ψ with the constraints of (6). For $\psi \in$ 3-SAT, $\mathcal{S}(f(A\mathbf{z})) \leq s_0$ where A is as described in (5) and s_0 is as defined earlier. For $\psi \in \overline{\text{3-SAT}}$, Lemma 45, proved using Claim 25, shows lower bounds on $\mathcal{S}(f(A\mathbf{z}))$, for all $A \in \text{GL}(|\mathbf{z}|, \mathbb{F})$. Proposition 46 proves α-gap-ETsparse is NP-hard using Lemma 45 and has the same proof as Proposition 44.

▶ **Lemma 45.** *Let $\psi \in \overline{\text{3-SAT}}$, f be as defined in (4) corresponding to ψ and $A \in \text{GL}(|\mathbf{z}|, \mathbb{F})$.*
1. *If $A(x_0)$ or $A(y_0)$ is a linear form in at least 2 variables, then $\mathcal{S}(f(A\mathbf{z})) \geq d_1 + 1$.*
2. *If A is not as in item 1 and if $A(x_j)$ is a linear form in at least 2 variables for some $j \in [n]$, then $\mathcal{S}(f(A\mathbf{z})) \geq d_2 + 1$.*
3. *If A is not as in items 1 and 2 and $\mathcal{S}(A(y_j + x_j)) \geq 3$ or $\mathcal{S}(A(y_j - x_j)) \geq 3$ for some $j \in [n]$, then $\mathcal{S}(f(A\mathbf{z})) \geq \frac{d_3^2+3d_3+2}{2}$.*
4. *If A is not of the form described in the previous three cases, then $\mathcal{S}(f(A\mathbf{z})) \geq (d_4+1)^3$.*

▶ **Proposition 46.** *Let $\text{char}(\mathbb{F}) = 0$. Then, α-gap-ETsparse is NP-hard for s-sparse homogeneous polynomial inputs over $\mathbb{F}$ and $\alpha = s^{\frac{1}{3}-\epsilon}$, where $\epsilon \in (0, \frac{1}{3})$ is an arbitrary constant.*

▶ Remark 47. The proof of Lemma 45 uses Claim 25, which works over $\text{char}(\mathbb{F}) = 0$ fields. We believe that Claim 25 can be modified for finite characteristic fields, using which the argument in this section can be extended over such fields.

5 NP-hardness of ETsupport

In this section, we prove Theorem 16 for characteristic 0 fields. The full version contains the proofs of the lemmas and the reduction for the finite characteristic case.

Proof sketch. We map ψ, a 3-CNF, to a polynomial f, which is the sum of degree separated polynomials with at least one polynomial of support $\sigma+1$ (σ is a constant integer) and the rest of support σ. As the summands are degree separated, $\text{Supp}(f) = \sigma+1$ and for any invertible linear transform A, $\text{Supp}(A(f))$ is the maximum support size over all the transformed summands. Claim 26 is used to show $\psi \in$ 3-SAT iff there exists an invertible linear transform A, such that $\text{Supp}(A(f)) \leq \sigma$. Thus, the reduction also holds for $(\sigma+1)$-to-σ ETsupport.

5.1 Construction of f and σ

Let $\sigma \geq 6$ be an even integer constant and ψ be as denoted in Section 3.1. Assume $n \geq \sigma+4$ and that in the first clause of ψ all the variables are complemented.[19] Let $\mathbf{x} := \{x_1, \ldots, x_n\}$, $\mathbf{y} := \{y_1, \ldots, y_n\}$ and $\mathbf{z} := \{z_1, \ldots, z_{\sigma-5}\}$ and $\mathbf{w} := \mathbf{x} \sqcup \mathbf{y} \sqcup \mathbf{z}$. Consider the polynomials:

- First, introduce $\binom{n+\sigma-5}{\sigma}$ many monomials defined by the set

$$P := \{(w_{i_1} \cdots w_{i_\sigma})^\star \mid w_{i_1}, \ldots, w_{i_\sigma} \in \mathbf{z} \sqcup \mathbf{x} \text{ and are pairwise distinct}\}.$$

- Then, introduce $\binom{n}{\frac{\sigma}{2}}$ many monomials defined by the set

$$Q := \{((x_{i_1} y_{i_1}) \cdots (x_{i_{\frac{\sigma}{2}}} y_{i_{\frac{\sigma}{2}}}))^\star \mid i_1, \ldots, i_{\frac{\sigma}{2}} \in [n] \text{ and are pairwise distinct}\}.$$

- Let $R := \{R_k(\mathbf{w}) \mid k \in [m]\}$, where $R_k(\mathbf{w})$ is defined corresponding to the k^{th} clause as:

$$R_k(\mathbf{w}) := (\prod_{j \in C_k} (y_j - a_{k,j} x_j))^2 (z_1 \cdots z_{\sigma-5})^\star.$$

Define $f(\mathbf{w}) := \sum_{g \in P} g(\mathbf{w}) + \sum_{h \in Q} h(\mathbf{w}) + \sum_{k=1}^{m} R_k(\mathbf{w})$. The degrees, denoted by $\star$, are of form $\sigma + i$ where $i \in [N]$ and $N = \binom{n+\sigma-5}{\sigma} + \binom{n}{\sigma/2} + m$ to ensure all polynomials in $P \sqcup Q \sqcup R$ are degree separated with degrees $\geq \sigma+1$. Based on this, Observation 48 holds.

▶ **Observation 48.** *$\mathcal{S}(f(\mathbf{w})) = O(n^\sigma + m)$ and $\text{Supp}(f(\mathbf{w})) = \sigma+1$.*

5.2 The forward direction

Proposition 49 shows how a satisfying assignment for ψ implies the existence of an invertible A, such that $\text{Supp}(f(A\mathbf{w})) = \sigma$ by constructing A from the satisfying assignment.

▶ **Proposition 49.** *Let $\psi \in$ 3-SAT with $(u_1, \ldots, u_n) \in \{0,1\}^n$ a satisfying assignment. Then, $\text{Supp}(f(A\mathbf{w})) = \sigma$, where the transform A is defined as*

$$A : z_j \mapsto z_j,\ x_i \mapsto x_i,\ y_i \mapsto y_i + (1-u_i)x_i \quad i \in [n], j \in [\sigma-5]. \tag{7}$$

[19] To achieve $n \geq \sigma+4$, add fresh variables and clauses in these variables to ψ. To ensure that the first clause contains only complemented variables, every uncomplemented variable x in the first clause is replaced by $\neg x$ followed by complementing each occurrence of x in the remaining clauses of ψ.

Proof. As all polynomials in $P \sqcup Q \sqcup R$ are degree separated, analysing the action of A on individual polynomials suffices. For $g \in P$, $g(A\mathbf{w}) = g$. On each monomial of Q, A acts as:

$$A : ((x_{i_1}y_{i_1}) \cdots (x_{i_{\frac{s}{2}}}y_{i_{\frac{\sigma}{2}}}))^\star \mapsto ((x_{i_1})(y_{i_1} + (1-u_{i_1})x_{i_1}) \cdots (x_{i_{\frac{\sigma}{2}}})(y_{i_{\frac{\sigma}{2}}} + (1-u_{i_{\frac{s}{2}}})x_{i_{\frac{\sigma}{2}}}))^\star.$$

Under A, each monomial of Q has support σ by Claim 26. For $k \in [m]$, A acts on $R_k(\mathbf{w})$ as:

$$A : (\prod_{j \in C_k}(y_j - a_{k,j}x_j))^2 \cdot (z_1 \cdots z_{\sigma-5})^\star \mapsto (\prod_{j \in C_k}(y_j + (1-a_{k,j}-u_j)x_j)^2 \cdot (z_1 \cdots z_{\sigma-5})^\star.$$

If $a_{k,j} \neq u_j$, then $a_{k,j} = 1 - u_j$. Since ψ is satisfiable, therefore for all $k \in [m]$, $a_{k,j} \neq u_j$ for some $j \in C_k$. Hence, $\mathrm{Supp}(R_k(A\mathbf{w})) \leq (\sigma - 5) + 5 = \sigma$ for all $k \in [m]$. ◀

5.3 The reverse direction

Now, we show that if $\mathrm{Supp}(f(A\mathbf{w})) \leq \sigma$ for $A \in \mathrm{GL}(|\mathbf{w}|, \mathbb{F})$, then a satisfying assignment can be obtained for ψ. Lemmas 50 and 51, proved using Claim 26, together show that A is as:

$$A : z_j \mapsto z_j,\ x_i \mapsto x_i,\ y_i \mapsto y_i + c_i x_i \quad c_i \in \mathbb{F},\ j \in [\sigma - 5], i \in [n]$$

without loss of generality.[20] Proposition 52 constructs a satisfying assignment for ψ from A.

▶ **Lemma 50.** *If* $\mathrm{Supp}(f(A\mathbf{w})) \leq \sigma$*, then* $\forall w \in \mathbf{z} \sqcup \mathbf{x}$*,* $A(w) = W$*, for scaled variable* $W \in \mathbf{w}$*.*

▶ **Lemma 51.** *If* $\mathrm{Supp}(f(A\mathbf{w})) \leq \sigma$*, then* $A(x_i) = X_i$ *and* $A(y_i) = Y_i + c_i X_i$*, for scaled variables* $Y_i, X_i \in \mathbf{w}$*.*

▶ **Proposition 52.** *A satisfying assignment* $\mathbf{u}$ *for* ψ *can be constructed from* A*.*

Proof. The action of A on R_k, where $k \in [m]$, is:

$$(\prod_{j \in C_k}(y_j - a_{k,j}x_j))^2 \cdot (z_1 \cdots z_{\sigma-5})^\star \mapsto (\prod_{j \in C_k}(y_j + (c_j - a_{k,j})x_j))^2 \cdot (z_1 \cdots z_{\sigma-5})^\star.$$

Thus, $\mathrm{Supp}(R_k(A\mathbf{w})) \leq \sigma$ iff for some $j \in C_k$, $c_j = a_{k,j}$. By assumption $\mathrm{Supp}(R_k(A\mathbf{w})) \leq \sigma$ for all $k \in [m]$. Hence, for each $R_k(\mathbf{w})$, there exists $j \in C_k$ such that $c_j \in \{0, 1\}$. Construct $\mathbf{u} \in \{0,1\}^n$ by setting $u_j := 1 - c_j$, for appropriate $j \in C_k$ and the remaining u_i's to arbitrary values in $\{0, 1\}$. From the definition of $\mathbf{u}$, it follows that for all $k \in [m]$, there exists $j \in C_k$ such that $u_j \neq a_{k,j}$. As k is arbitrary, all clauses are satisfied. ◀

6 Conclusion

In this work, we show that ET for sparse polynomials is NP-hard. Particularly, we show the NP-hardness of MCSP for orbits of homogeneous sparse polynomials (a dense subclass of hom-ΣΠΣ circuits) over characteristic 0 fields. We also define a gap version of ET for sparse polynomials and show it is NP-hard, which implies the NP-hardness of $s^{\frac{1}{3}-\epsilon}$-factor approximation of the sparse-orbit complexity of s-sparse polynomials. Lastly, we show that ET for constant-support polynomials is NP-hard. In all three cases, we reduce 3-SAT to the respective problems. We end by listing some problems whose solutions we do not know:

[20] as permutation and non-zero scaling of variables do not affect the support.

1. **Hardness of ETsparse for constant degree polynomials:** In the reduction of Theorem 2, can the degree of the output polynomial be made constant? Currently, the degree is polynomial in the number of clauses and variables.
2. **Improving the gap in Theorem 8:** Can α-gap-ETsparse be shown NP-hard for $\alpha = s^{1-\epsilon}$, where the input polynomial has sparsity s and $\epsilon > 0$ is an arbitrary constant?
3. **Hardness of ETsupport for $\sigma = 2$:** Is checking if a given polynomial is in the orbit of a support-2 polynomial NP-hard? Theorem 16 shows ETsupport is NP-hard for $\sigma \geq 6$.
4. **Hardness of MCSP for hom-$\Sigma\Pi\Sigma$ circuits:** Is MCSP for hom-$\Sigma\Pi\Sigma$ circuits NP-hard?

References

1 Manindra Agrawal, Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Hitting-Sets for ROABP and Sum of Set-Multilinear Circuits. *SIAM J. Comput.*, 44(3):669–697, 2015. doi:10.1137/140975103.

2 Manindra Agrawal and Nitin Saxena. Automorphisms of finite rings and applications to complexity of problems. In *Proceedings of the 22nd Annual Conference on Theoretical Aspects of Computer Science*, STACS'05, pages 1–17, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/978-3-540-31856-9_1.

3 Eric Allender and Bireswar Das. Zero knowledge and circuit minimization. *Inf. Comput.*, 256:2–8, 2017. Conference version appeared in the proceedings of MFCS 2014. doi:10.1016/J.IC.2017.04.004.

4 Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael E. Saks. Minimizing DNF Formulas and AC^0_d Circuits Given a Truth Table. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 237–251. IEEE Computer Society, 2006. doi:10.1109/CCC.2006.27.

5 Prashanth Amireddy, Ankit Garg, Neeraj Kayal, Chandan Saha, and Bhargav Thankey. Low-depth arithmetic circuit lower bounds: Bypassing set-multilinearization. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 12:1–12:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.12.

6 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998. Conference version appeared in the proceedings of FOCS 1992. doi:10.1145/278298.278306.

7 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998. Conference version appeared in the proceedings of FOCS 1992. doi:10.1145/273865.273901.

8 Omkar Baraskar, Agrim Dewan, and Chandan Saha. Testing equivalence to design polynomials. In Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov, editors, *41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024, March 12-14, 2024, Clermont-Ferrand, France*, volume 289 of *LIPIcs*, pages 9:1–9:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.STACS.2024.9.

9 Amos Beimel, Francesco Bergadano, Nader H. Bshouty, Eyal Kushilevitz, and Stefano Varricchio. Learning functions represented as multiplicity automata. *J. ACM*, 47(3):506–530, 2000. Conference version appeared in the proceedings of FOCS 1996. doi:10.1145/337244.337257.

10 Michael Ben-Or and Prasoon Tiwari. A Deterministic Algorithm for Sparse Multivariate Polynominal Interpolation (Extended Abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 301–309. ACM, 1988. doi:10.1145/62212.62241.

11 Vishwas Bhargava, Shubhangi Saraf, and Ilya Volkovich. Deterministic factorization of sparse polynomials with bounded individual degree. *J. ACM*, 67(2):8:1–8:28, 2020. Conference version appeared in the proceedings of FOCS 2018. doi:10.1145/3365667.

12 Markus Bläser and Gorav Jindal. A new deterministic algorithm for sparse multivariate polynomial interpolation. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 51–58. ACM, 2014. doi:10.1145/2608628.2608648.

13 Lenore Blum, Mike Shub, and Steve Smale. On a Theory of Computation and Complexity over the Real Numbers: NP-completeness, Recursive Functions and Universal Machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, 1989. doi:10.1090/S0273-0979-1989-15750-9.

14 Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning Algorithms from Natural Proofs. In Ran Raz, editor, *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, volume 50 of *LIPIcs*, pages 10:1–10:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.CCC.2016.10.

15 Suryajith Chillara, Coral Grichener, and Amir Shpilka. On hardness of testing equivalence to sparse polynomials under shifts. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 22:1–22:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.STACS.2023.22.

16 Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978. doi:10.1016/0020-0190(78)90067-4.

17 Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):12, 2007. Conference version appeared in the proceedings of STOC 2006. doi:10.1145/1236457.1236459.

18 Michael A. Forbes. Some concrete questions on the border complexity of polynomials, 2016. URL: https://www.youtube.com/watch?v=1HMogQIHT6Q.

19 Michael A. Forbes and Amir Shpilka. Quasipolynomial-Time Identity Testing of Noncommutative and Read-Once Oblivious Algebraic Branching Programs. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 243–252. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.34.

20 Dima Grigoriev and Marek Karpinski. A Zero-Test and an Interpolation Algorithm for the Shifted Sparse Polynominals. In Gérard D. Cohen, Teo Mora, and Oscar Moreno, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 10th International Symposium, AAECC-10, San Juan de Puerto Rico, Puerto Rico, May 10-14, 1993, Proceedings*, volume 673 of *Lecture Notes in Computer Science*, pages 162–169. Springer, 1993. doi:10.1007/3-540-56686-4_41.

21 Dima Grigoriev, Marek Karpinski, and Michael F. Singer. Fast Parallel Algorithms for Sparse Multivariate Polynomial Interpolation over Finite Fields. *SIAM J. Comput.*, 19(6):1059–1063, 1990. doi:10.1137/0219073.

22 Dima Grigoriev and Alexander A. Razborov. Exponential Complexity Lower Bounds for Depth 3 Arithmetic Circuits in Algebras of Functions Over Finite Fields. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 269–278. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743456.

23 Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Arithmetic circuits: A chasm at depth 3. *SIAM J. Comput.*, 45(3):1064–1079, 2016. Conference version appeared in the proceedings of FOCS 2013. doi:10.1137/140957123.

24 Nikhil Gupta, Chandan Saha, and Bhargav Thankey. Equivalence Test for Read-Once Arithmetic Formulas. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 4205–4272. SIAM, 2023. doi:10.1137/1.9781611977554.ch162.

25 Thomas R. Hancock, Tao Jiang, Ming Li, and John Tromp. Lower Bounds on Learning Decision Lists and Trees. *Inf. Comput.*, 126(2):114–122, 1996. doi:10.1006/INCO.1996.0040.

26 Johan Håstad. Tensor Rank is NP-Complete. *J. Algorithms*, 11(4):644–654, 1990. Conference version appeared in the proceedings of ICALP 1989. doi:10.1016/0196-6774(90)90014-6.

27 Shuichi Hirahara. Non-Black-Box Worst-Case to Average-Case Reductions within NP. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 247–258. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00032.

28 Shuichi Hirahara. NP-Hardness of Learning Programs and Partial MCSP. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 – November 3, 2022*, pages 968–979. IEEE, 2022. doi:10.1109/FOCS54457.2022.00095.

29 Shuichi Hirahara, Igor C. Oliveira, and Rahul Santhanam. NP-hardness of Minimum Circuit Size Problem for OR-AND-MOD Circuits. In Rocco A. Servedio, editor, *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, volume 102 of *LIPIcs*, pages 5:1–5:31. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.CCC.2018.5.

30 Rahul Ilango. Constant Depth Formula and Partial Function Versions of MCSP are Hard. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 424–433. IEEE, 2020. doi:10.1109/FOCS46700.2020.00047.

31 Rahul Ilango. The Minimum Formula Size Problem is (ETH) Hard. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 427–432. IEEE, 2021. doi:10.1109/FOCS52979.2021.00050.

32 Rahul Ilango, Bruno Loff, and Igor C. Oliveira. NP-Hardness of Circuit Minimization for Multi-Output Functions. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 22:1–22:36. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.CCC.2020.22.

33 Russell Impagliazzo and Avi Wigderson. $P = BPP$ if E Requires Exponential Circuits: Derandomizing the XOR Lemma. In Frank Thomson Leighton and Peter W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 220–229. ACM, 1997. doi:10.1145/258533.258590.

34 Valentine Kabanets and Jin-yi Cai. Circuit minimization problem. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 73–79. ACM, 2000. doi:10.1145/335305.335314.

35 Erich L. Kaltofen and Barry M. Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *J. Symb. Comput.*, 9(3):301–320, 1990. Conference version appeared in the proceedings of FOCS 1988. doi:10.1016/S0747-7171(08)80015-6.

36 Neeraj Kayal. Efficient algorithms for some special cases of the polynomial equivalence problem. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1409–1421. SIAM, 2011. doi:10.1137/1.9781611973082.108.

37 Neeraj Kayal. Affine projections of polynomials: extended abstract. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19–22, 2012*, pages 643–662. ACM, 2012. doi:10.1145/2213977.2214036.

38 Neeraj Kayal, Vineet Nair, and Chandan Saha. Separation between read-once oblivious algebraic branching programs (roabps) and multilinear depth-three circuits. *ACM Trans. Comput. Theory*, 12(1):2:1–2:27, 2020. Conference version appeared in the proceedings of STACS 2016. doi:10.1145/3369928.

39 Neeraj Kayal, Vineet Nair, Chandan Saha, and Sébastien Tavenas. Reconstruction of full rank algebraic branching programs. *ACM Trans. Comput. Theory*, 11(1):2:1–2:56, 2019. Conference version appeared in the proceedings of CCC 2017. doi:10.1145/3282427.

40 Adam R. Klivans and Amir Shpilka. Learning restricted models of arithmetic circuits. *Theory Comput.*, 2(10):185–206, 2006. Conference version appeared in the proceedings of COLT 2003. doi:10.4086/TOC.2006.V002A010.

41 Adam R. Klivans and Daniel A. Spielman. Randomness efficient identity testing of multivariate polynomials. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 216–223. ACM, 2001. doi:10.1145/380752.380801.

42 Nutan Limaye, Srikanth Srinivasan, and Sébastien Tavenas. Superpolynomial Lower Bounds Against Low-Depth Algebraic Circuits. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 804–814. IEEE, 2021. doi:10.1109/FOCS52979.2021.00083.

43 Richard J. Lipton and Nisheeth K. Vishnoi. Deterministic identity testing for multivariate polynomials. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 756–760. ACM/SIAM, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644233.

44 W. J. Masek. Some NP-complete set covering problems. *Unpublished Manuscript*, 1979.

45 Dori Medini and Amir Shpilka. Hitting sets and reconstruction for dense orbits in vp_{e} and ΣΠΣ circuits. In Valentine Kabanets, editor, *36th Computational Complexity Conference, CCC 2021, July 20-23, 2021, Toronto, Ontario, Canada (Virtual Conference)*, volume 200 of *LIPIcs*, pages 19:1–19:27. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CCC.2021.19.

46 Jacques Patarin. Hidden fields equations (HFE) and isomorphisms of polynomials (IP): two new families of asymmetric algorithms. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, volume 1070 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 1996. doi:10.1007/3-540-68339-9_4.

47 Ján Pich and Rahul Santhanam. Why are Proof Complexity Lower Bounds Hard? In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1305–1324. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00080.

48 Alexander A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Math. Notes*, 41:333–338, 1987. doi:10.1007/BF01137685.

49 Daniel S. Roche. What Can (and Can't) we Do with Sparse Polynomials? In Manuel Kauers, Alexey Ovchinnikov, and Éric Schost, editors, *Proceedings of the 2018 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2018, New York, NY, USA, July 16-19, 2018*, pages 25–30. ACM, 2018. doi:10.1145/3208976.3209027.

50 Chandan Saha and Bhargav Thankey. Hitting sets for orbits of circuit classes and polynomial families. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPIcs*, pages 50:1–50:26. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.APPROX/RANDOM.2021.50.

51 Nitin Saxena. *Morphisms of rings and applications to complexity*. PhD thesis, Indian Institute of Technology, Kanpur, 2006. URL: https://www.cse.iitk.ac.in/users/manindra/Students/thesis_saxena.pdf.

52 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.

53 Yaroslav Shitov. How hard is the tensor rank?, 2016. arXiv:1611.01559.

54 Sébastien Tavenas. Improved bounds for reduction to depth 4 and depth 3. *Inf. Comput.*, 240:2–11, 2015. Conference version appeared in the proceedings of MFCS 2013. doi:10.1016/J.IC.2014.09.004.

55 Thomas Thierauf. The isomorphism problem for read-once branching programs and arithmetic circuits. *Chic. J. Theor. Comput. Sci.*, 1998, 1998. URL: http://cjtcs.cs.uchicago.edu/articles/1998/1/contents.html.

56 Joachim von zur Gathen and Erich L. Kaltofen. Factoring Sparse Multivariate Polynomials. *J. Comput. Syst. Sci.*, 31(2):265–287, 1985. doi:10.1016/0022-0000(85)90044-3.

57 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation, Marseille, France, June 1979, Proceedings*, pages 216–226, 1979. doi:10.1007/3-540-09519-5_73.

Vital Edges for (s,t)-Mincut: Efficient Algorithms, Compact Structures, & Optimal Sensitivity Oracles

Surender Baswana
Department of Computer Science & Engineering, IIT Kanpur, India

Koustav Bhanja
Department of Computer Science & Engineering, IIT Kanpur, India

Abstract

Let G be a directed weighted graph on n vertices and m edges with designated source and sink vertices s and t. An edge in G is vital if its removal reduces the capacity of (s,t)-mincut. Since the seminal work of Ford and Fulkerson [CJM 1956], a long line of work has been done on computing the *most vital edge* and *all vital edges* of G. However, even after 60 years, the existing results are for either undirected or unweighted graphs. We present the following result for *directed weighted graphs* that also solves an open problem by Ausiello, Franciosa, Lari, and Ribichini [NETWORKS 2019].

1. **Algorithmic Results:** There is an algorithm that computes all vital edges as well as the most vital edge of G using $\mathcal{O}(n)$ maximum (s,t)-flow computations.

   Vital edges play a crucial role in the design of *sensitivity oracle* for (s,t)-mincut – a compact data structure for reporting (s,t)-mincut after insertion/failure of any edge. For directed graphs, the only existing sensitivity oracle is for unweighted graphs by Picard and Queyranne [MPS 1982]. We present the first and optimal sensitivity oracle for *directed weighted graphs* as follows.
2. **Sensitivity Oracles:**
   - **(a)** There is an optimal $\mathcal{O}(n^2)$ space data structure that can report an (s,t)-mincut C in $\mathcal{O}(|C|)$ time after the failure/insertion of any edge.
   - **(b)** There is an $\mathcal{O}(n)$ space data structure that can report the capacity of (s,t)-mincut after failure or insertion of any edge e in $\mathcal{O}(1)$ time if the capacity of edge e is known.

   A *mincut for a vital edge* e is an (s,t)-cut of the least capacity in which edge e is outgoing. For unweighted graphs, in a classical work, Picard and Queyranne [MPS 1982] designed an $\mathcal{O}(m)$ space directed acyclic graph (DAG) that stores and characterizes all mincuts for all vital edges. Conversely, there is a set containing at most $n-1$ (s,t)-cuts such that at least one mincut for every vital edge belongs to the set. We generalize these results for *directed weighted graphs* as follows.
3. **Structural & Combinatorial Results:**
   - **(a)** There is a set $\mathcal{M}$ containing at most $n-1$ (s,t)-cuts such that at least one mincut for every vital edge belongs to the set. This bound is tight as well. We also show that set $\mathcal{M}$ can be computed using $\mathcal{O}(n)$ maximum (s,t)-flow computations.
   - **(b)** We design two compact structures for storing and characterizing all mincuts for all vital edges – (i) an $\mathcal{O}(m)$ space DAG for *partial* and (ii) an $\mathcal{O}(mn)$ space structure for *complete* characterization.

To arrive at our results, we develop new techniques, especially a generalization of *maxflow-mincut* Theorem by Ford and Fulkerson [CJM 1956], which might be of independent interest.

2012 ACM Subject Classification Theory of computation → Network flows; Theory of computation → Data structures design and analysis; Mathematics of computing → Graph algorithms

Keywords and phrases maxflow, vital edges, graph algorithms, structures, st-cuts, sensitivity oracle

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.17

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2310.12096

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 17; pp. 17:1–17:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding The research work of Surender Baswana was partially supported by Tapas Mishra Memorial Chair at Indian Institute of Technology Kanpur.

Acknowledgements We thank Keerti Choudhary for her valuable feedback on this article.

1 Introduction

For any graph problem, there are edges whose removal affects the solution of the given problem. These edges are called *vital* edges for the problem. There has been extensive research on the vital edges for various fundamental problems – shortest paths/distance [32, 34, 23, 28, 19], minimum spanning trees [15, 26, 33], strongly connected components (SCC) [16]. The concept of vital edge for (s,t)-mincut has existed ever since the seminal work of Ford and Fulkerson [14] on maximum (s,t)-flow. In this article, for directed weighted graphs, we present the following two main results – (1) an efficient *algorithm* for computing all vital edges, and (2) optimal *sensitivity oracles* for (s,t)-mincut. The algorithm in (1) is the first nontrivial algorithm for computing all vital edges in directed weighted graphs, which also answers an open question in [3]. Our optimal sensitivity oracle in (2) is the first sensitivity oracle for directed weighted graphs in the area of minimum cuts. In order to arrive at these results, we present interesting *structural* & optimal *combinatorial* results on mincuts for vital edges. These results provide a generalization of the classical work of Picard and Queyranne [30] and the recent work of Baswana, Bhanja, and Pandey [4].

Let $G=(V,E)$ be a directed graph on $n=|V|$ vertices and $m=|E|$ edges. Each edge $e \in E$ has a capacity, denoted by $w(e)$, which is a positive real number. Let s be a designated source vertex and t be a designated sink vertex in G.

A set $C \subset V$ is said to be a cut if $C \neq \emptyset$. The outgoing edges from C are called *contributing* edges of C. The *capacity* of cut C, denoted by $c(C)$, is defined as the sum of the capacities of all contributing edges of C. A cut C is said to be an (s,t)-cut if $s \in C$ and $t \in \overline{C} = V \setminus C$. An (s,t)-cut C with the least capacity is called an (s,t)-*mincut*. We denote the capacity of (s,t)-mincut by f^*. The (s,t)-mincut of a graph is a fundamental concept in graph theory and is used to design efficient algorithms for numerous real-world problems [1]. Now we formally define the vital edges for (s,t)-micuts.

▶ **Definition 1** (vital edge). *An edge $e \in E$ is said to be a vital edge if the removal of e decreases the capacity of (s,t)-mincut in G. E_{vit} denotes the set of all vital edges in G.*

Observe that each edge that contributes to an (s,t)-mincut is definitely a vital edge. However, a vital edge might not necessarily contribute to any (s,t)-mincut (e.g., edge (v_1,v_4) in Figure 1)(i). An edge that is not vital is called a *nonvital* edge. At first glance, it may appear that we may remove all nonvital edges from the graph without affecting the (s,t)-mincut. But it is not true, as stated in the following note.

▶ Note 2. Although the removal of any single nonvital edge does not decrease the capacity of (s,t)-mincut, the removal of a set of nonvital edges might lead to the reduction in the capacity of (s,t)-mincut (e.g., edges (v_2,v_6) and (v_3,v_5) in Figure 1(i)).

The design of efficient algorithms for various problems [31, 35, 27, 29] related to vital edges started just a few years after the seminal work of Ford and Fulkerson [14]. The *vitality of an edge* e is the reduction in the capacity of (s,t)-mincut after the removal of edge e. Observe that one maximum (s,t)-flow computation is sufficient for computing the vitality of any edge. So, for computing all vital edges and their vitality, there is a trivial algorithm that requires $\mathcal{O}(m)$ maximum (s,t)-flow computations. An edge having the maximum vitality is

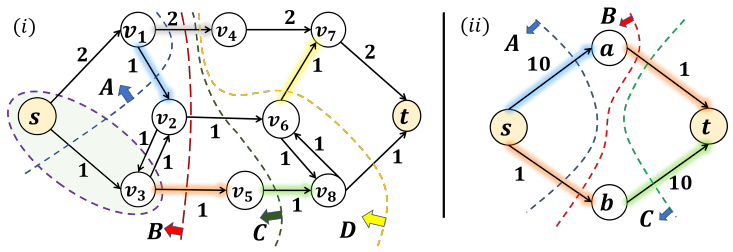


Figure 1 (i) Each mincut for vital edge (v_1, v_4) contains a nonvital edge (shown in the same color). (ii) A mincut cover fails to include mincut B for edges (a, t) and (s, b) if property $\mathcal{P}$ is not ensured in the construction. An edge and the mincut for the edge are shown in the same color.

said to be the *most vital edge*. Aneja, Chandrasekaran, and Nair [2] designed an algorithm that performs $\mathcal{O}(n)$ maximum (s,t)-flow computations to compute the most vital edge in an undirected graph. Ausiello, Franciosa, Lari, and Ribichini [3] showed that $\mathcal{O}(n)$ maximum (s,t)-flow computations are sufficient even for computing all vital edges and their vitality in an undirected graph. Unfortunately, even after 60 years, the following question has remained unanswered, which is also posed as an open problem in [3].

▶ **Question 1.** *For directed weighted graph G, does there exist an algorithm that can compute all vital edges along with their vitality using $\mathcal{O}(n)$ maximum (s,t)-flow computations?*

We can generalize the notion of (s,t)-mincut to *"mincut for an edge"* as follows.

▶ **Definition 3** (mincut for an edge). *Let e be a contributing edge of an (s,t)-cut C. C is a mincut for edge e if $c(C) \leq c(C')$ for each (s,t)-cut C' in which edge e appears as a contributing edge.*

Not only the study of mincuts for vital edges is important from a graph theoretic perspective but it also plays a crucial role in designing sensitivity oracle for (s,t)-mincuts – a compact data structure that efficiently reports an (s,t)-mincut after the failure/insertion of any edge.

The minimum cuts in a graph can be quite large in number – $\Omega(n^2)$ global mincuts [11], $\Omega(2^n)$ (s,t)-mincuts [30]. Interestingly, compact structures have been invented that compactly store and *characterize* various types of minimum cuts and cuts of capacity near minimum [13, 12, 30, 11, 4]. A compact structure G' is said to characterize a set $\mathcal{C}$ of cuts using a property $\mathcal{P}$ if the following holds. A cut C belongs to $\mathcal{C}$ if and only if cut C satisfies property $\mathcal{P}$ in graph G'. Specifically for (s,t)-mincuts in any directed weighted graph G, Picard and Queyranne [30] showed that there exists a directed acyclic graph (DAG), denoted by $\mathcal{D}_{PQ}(G)$, occupying $\mathcal{O}(m)$ space that compactly stores all (s,t)-mincuts. In addition, it provides the following characterization for each (s,t)-mincut.

An (s,t)-cut C is an (s,t)-mincut in G if and only if C is a 1-transversal cut in $\mathcal{D}_{PQ}(G)$.

An (s,t)-cut is said to be 1-transversal if its edges intersect any simple path at most *once*; e.g., cut A is 1-transversal but cut $\{s, v_3\}$ is not 1-transversal in Figure 1(i). The 1-transversality property also plays a crucial role in designing sensitivity oracles for (s,t)-mincuts in unweighted graphs [30, 4]. Observe that $\mathcal{D}_{PQ}(G)$ stores all mincuts for all edges that contribute to (s,t)-mincuts. However, edges contributing to (s,t)-mincuts may constitute a small subset of the set of all vital edges. Therefore, $\mathcal{D}_{PQ}(G)$ may fail to preserve all mincuts for all vital edges. This raises the following question.

▶ **Question 2.** *Does there exist a compact graph structure that stores and characterizes all mincuts for all vital edges in directed weighted graph G?*

Constructing the smallest set storing a minimum cut for every edge or every pair of vertices has been addressed extensively [22, 11, 9, 18, 20, 21] since the remarkable work of Gomory and Hu [18]. The smallest set of (s,t)-cuts that has at least one mincut for every edge is called a *mincut cover*. For directed unweighted graphs, Baswana, Bhanja, and Pandey [4], exploiting the DAG structure in [30], showed that there is a mincut cover of cardinality at most $n-1$ for all vital edges. For undirected weighted graphs, it is shown in [3] that there is a mincut cover of cardinality at most $n-1$ for all edges, and hence for all vital edges. Unfortunately, for directed weighted graphs, we establish that it is not possible to have a mincut cover of cardinality $o(n^2)$ for all edges (Theorem 37). Therefore, the following question naturally arises.

▶ **Question 3.** *Does there exist a mincut cover of cardinality $o(n^2)$ for all vital edges in directed weighted graph G?*

Design of sensitivity oracles have been studied quite extensively for various fundamental problems in both unweighted and weighted graphs – shortest paths/distances [8, 19], reachability [24, 10], strongly connected components [5, 16], all-pairs mincuts [6]. In weighted graphs, the concept of failures/insertions of edges is, interestingly, more generic. Here, the aim is to report the solution to the given problem given that the capacities of a *small* set of edges are decreased/increased by an amount $\Delta > 0$.

For (s,t)-mincuts in directed graphs, the existing sensitivity oracles are only for unweighted graphs and completely based on DAG $\mathcal{D}_{PQ}(G)$ [30], which dates back to 1982. Firstly, there is an $\mathcal{O}(m)$ space sensitivity oracle given in [30]. After the failure/insertion of an edge, the oracle takes $\mathcal{O}(1)$ time for reporting the capacity and $\mathcal{O}(m)$ time for reporting the corresponding (s,t)-mincut. Assuming the edge of the query exists in the graph, an $\mathcal{O}(n)$ space data structure can be designed [4]. This data structure can report the capacity of (s,t)-mincut in $\mathcal{O}(1)$ time and an (s,t)-mincut C in $\mathcal{O}(|C|)$ time after the failure/insertion of an edge. For various problems related to sensitivity analysis, it is also important to efficiently report a compact structure that stores and characterizes *all* (s,t)-mincuts after the failure of any edge, as shown in [30]. For unweighted graphs, DAG $\mathcal{D}_{PQ}$ for the resulting graph can be reported in $\mathcal{O}(m)$ time [30]. All these results crucially exploit the property that, in unweighted graphs, a mincut for a vital edge is also an (s,t)-mincut; hence, every contributing edge is also vital. However, for weighted graphs, this property no longer holds. In fact, nonvital edges may contribute to all mincuts for a vital edge (e.g., all mincuts $\{A,B,C,D\}$ for edge (v_1,v_4) in Figure 1(i)). So, the following question arises.

▶ **Question 4.** *For directed weighted graph G, does there exist*
1. *an $\mathcal{O}(n)$ space data structure that can report the capacity of (s,t)-mincut in $\mathcal{O}(1)$ time,*
2. *a compact data structure that can report an (s,t)-mincut C in $\mathcal{O}(|C|)$ time, and*
3. *a compact data structure that can report DAG $\mathcal{D}_{PQ}$ in $\mathcal{O}(m)$ time for the resulting graph after increasing/decreasing the capacity of any given edge?*

▶ Note 4. It is a simple exercise to design an $\mathcal{O}(n^2)$ space data structure and an $\mathcal{O}(n)$ space data structure that, after increasing the capacity of any edge, can report an (s,t)-mincut C in $\mathcal{O}(|C|)$ time and its capacity in $\mathcal{O}(1)$ time, respectively. Henceforth, while addressing sensitivity oracles, we focus only on handling the decrease in the capacity of an edge.

1.1 Our contribution and the organization of the results

We provide an affirmative answer to all the four questions raised above. We present basic notations and terminologies in the following section. Thereafter, we present a generalization of *maxflow-mincut* Theorem by Ford and Fulkerson [14] in Section 3. This property, which

might be of independent interest, plays a key role in establishing various results in this article. Mincut cover for all vital edges and sensitivity oracles for (s,t)-mincut are presented in Section 4 and 5 respectively. Algorithm for computing all vital edges is presented in Section 6. Compact structures for storing and characterizing all mincuts for all vital edges are described in Section 7. We present various lower bounds in Section 8.

2 Preliminaries

For any directed weighted graph H with a designated source vertex s and a designated sink vertex t, we define the following notations to be used throughout the article.

- $H \setminus \{e\}$: Graph after the failure of an edge e in H.
- value(f,H): value of an (s,t)-flow f in graph H. We denote value(f,G) by f^*.
- $f(e,H)$: value of (s,t)-flow f along edge e in H.
- $f_{in}(C,H)$: For any (s,t)-flow f and an (s,t)-cut C in H, $f_{in}(C,H)$ is the sum of the flow through all edges that leave $\overline{C}$ and enter C.
- $f_{out}(C,H)$: For any (s,t)-flow f and an (s,t)-cut C in H, $f_{out}(C,H)$ is the sum of the flow through all edges that leave C and enter $\overline{C}$.

Without causing any ambiguity, we use $f_{in}(C)$ and $f_{out}(C)$ to denote $f_{in}(C,G)$ and $f_{out}(C,G)$ respectively.

The following lemma can be proved easily using the conservation of an (s,t)-flow.

▶ **Lemma 5.** *For any (s,t)-cut C in H, $f_{out}(C,H) - f_{in}(C,H) = value(f,H)$.*

For a set $U \subseteq V$, $E_{vit}(U)$ denotes the set of vital edges whose both endpoints belong to U. We now introduce a notation that quantitatively captures the *vitality* of an edge.

- $w_{min}(e)$: the reduction in the capacity of (s,t)-mincut in G after the failure of edge e.

The following observation is immediate from Definition 1.

▶ **Observation 6.** *For any vital edge e, $w_{min}(e) > 0$.*

▶ **Definition 7** (A mincut cover for a set of edges)**.** *A set $\mathcal{A}$ consisting of (s,t)-cuts is said to be a mincut cover for a set of edges E' if, for each edge $e \in E'$, at least one mincut for e is present in $\mathcal{A}$, and $|\mathcal{A}|$ is the smallest.*

The following lemma provides a maximum (s,t)-flow based characterization for a vital edge.

▶ **Lemma 8.** *An edge e is vital if and only if $f(e) > 0$ in every maximum (s,t)-flow f in G.*

Lemma 8 can be proved easily using the strong duality between maximum (s,t)-flow and (s,t)-mincut [14].

▶ **Definition 9** (Edge-set of a cut)**.** *A cut C is said to* separate *a pair of vertices $u,v \in V$ if either $u \in C$ and $v \in \overline{C}$ or $v \in C$ and $u \in \overline{C}$. The edge-set of C is the set of all those edges whose endpoints are separated by C.*

For an undirected graph, the set of contributing edges of a cut is the edge-set of the cut. In a seminal work, Gomory and Hu [17] established the following two results for an undirected graph. (1) There is a set of $n-1$ cuts such that the cut of the least capacity separating each pair of vertices is present in this set. (2) Each of these $n-1$ cuts appears as a cut in a spanning tree on the vertex set V. This tree is widely-known as Gomory-Hu tree.

The construction of Gomory-Hu tree crucially exploits that the capacity of each cut in graph is defined as the sum of the capacities of all contributing edges of the cut. Suppose capacity of each cut is defined by any arbitrary real-valued function F. Even for this generic

setting, Cheng and Hu [9] showed that there exists a set consisting of $n-1$ cuts such that a cut of the least capacity (F-value) separating any pair of vertices is present in the set. Furthermore, Cheng and Hu [9] showed that all these cuts can be stored in a suitably augmented rooted full binary tree, called *ancestor tree* as follows. Every vertex of the given graph is mapped to a unique leaf node in the ancestor tree, and the cut of the least capacity separating any pair of vertices is stored at their lowest common ancestor (LCA) in the tree. The ancestor tree occupies $\mathcal{O}(n^2)$ space and, Cheng and Hu [9] also designed an efficient algorithm to construct the tree – It performs only $\mathcal{O}(n)$ calls to an algorithm that, given any pair of vertices, can report a cut of the least capacity separating them.

Ausiello, Franciosa, Lari, and Ribichini [3] showed that the ancestor tree of Cheng and Hu [9] can be constructed for (s,t)-cuts as well if function F is defined as follows.

$$\text{For a set } C \subset V,\ F(C) = \begin{cases} c(C), & \text{if } s \in C \text{ and } t \in \overline{C} \\ \infty, & \text{otherwise.} \end{cases} \tag{1}$$

This insight played a crucial role in the computation of all vital edges in an undirected graph using $\mathcal{O}(n)$ maximum (s,t)-flow computations [3].

3 A Generalization of FlowCut Property

Let $E_{min} \subseteq E$ be the set of all edges contributing to any (s,t)-mincut. Ford and Fulkerson [14] established a strong duality between (s,t)-mincut and maximum (s,t)-flow. Exploiting this, the following property provides a maximum (s,t)-flow based characterization for mincut for any edge $e \in E_{min}$.

FlowCut: *Let C be an (s,t)-cut and $e \in E_{min}$ is a contributing edge of C. C is a mincut for edge e if and only if for any maximum (s,t)-flow, each contributing edge of C is fully saturated and each incoming edge of C carries no flow.*

For directed weighted graphs, we know that E_{min} maybe only a proper subset of the set of all vital edges. We now present a generalization of FlowCut property that provides a maximum (s,t)-flow based characterization for each mincut $C(e)$, $\forall e \in E_{vit}$.

▶ **Theorem 10** (GenFlowCut). *Let C be an (s,t)-cut and a vital edge $e=(u,v)$ contributes to C in G. C is a mincut for e if and only if there is a maximum (s,t)-flow f such that*

1. *$f_{in}(C) = 0$.*
2. *e carries exactly $w_{min}(e)$ amount of (s,t)-flow and every other contributing edge e' in C is fully saturated, that is, $f(e') = w(e')$.*

Proof. Suppose C is a mincut for vital edge e in G. After the removal of edge e, the capacity of (s,t)-cut C is related to f^* as follows.

$$\text{In graph } G \setminus \{e\},\ c(C) = f^* - w_{min}(e) \tag{2}$$

Let G' be the graph obtained from G after reducing the capacity of edge e from $w(e)$ to $w_{min}(e)$. Therefore, using Equation 2, the capacity of C in G' is $(f^* - w_{min}(e)) + w_{min}(e) = f^*$. Let $\mathcal{C}_{u,v}$ be the set of all (s,t)-cuts that keep u on the side of s and v on the side of t. The capacity of each cut in $\mathcal{C}_{u,v}$ gets reduced by the same amount due to reduction in the capacity of e. Therefore, C is a mincut for edge e in G' as well. Hence the capacity of every cut belonging to $\mathcal{C}_{u,v}$ in G' is at least f^*. Capacity of any (s,t)-cut in G', that does not belong to $\mathcal{C}_{u,v}$, is at least f^* since it remains unaffected by the reduction in the capacity of

edge e. These facts imply that f^* is the capacity of (s,t)-mincut in G'. Thus, using the strong duality between maximum (s,t)-flow and (s,t)-mincut, it follows that there exists a maximum (s,t)-flow f in G' such that value$(f,G') = f^*$. Using Lemma 5 for C in G', we get the following equality.

$$f_{out}(C,G') - f_{in}(C,G') = f^* \tag{3}$$

It follows from the capacity constraint that $f_{out}(C,G') \leq c(C)$. So, Equation 3 implies that $f_{in}(C,G') \leq c(C) - f^*$. Since $c(C) = f^*$ in G' as shown above, we arrive at the following.

$$f_{in}(C,G') = 0 \qquad \text{and} \qquad f_{out}(C,G') = f^* \tag{4}$$

This implies that, in G', each outgoing edge of C is fully saturated and each incoming edge does not carry any amount of flow. f is also a valid maximum (s,t)-flow for graph G because f^* is the value of maximum (s,t)-flow in G and $f((u,v)) \leq w((u,v))$ in G. Therefore, it follows from Equation 4 that in graph G, $f_{in}(C) = 0$, each outgoing edge of C is fully saturated, and edge e carries exactly $w_{min}(e)$ amount of (s,t)-flow.

We now prove the converse part. Suppose there is a maximum (s,t)-flow in G such that each outgoing edge of C, except e, is fully saturated and each incoming edge carries no flow. Therefore, $c(C)$ in G is $f^* - w_{min}(e) + w(e)$. So, in graph $G \setminus \{e\}$, capacity of C is $f^* - w_{min}(e)$. Since e is a vital edge, it follows from definition that $f^* - w_{min}(e)$ is the capacity of (s,t)-mincut in $G \setminus \{e\}$. Therefore, C is an (s,t)-mincut in $G \setminus \{e\}$. Hence, each (s,t)-cut in G that keeps u on the side of s and v on the side of t has a capacity at least $f^* - w_{min}(e) + w(e)$. This implies that C is a mincut for edge e. ◀

▶ Remark 11. Theorem 10 crucially exploits Equation 2 which holds for vital edges only. Note that $w_{min}(e) = 0$ for a nonvital edge e. So, if C is a mincut for e, C might have capacity $> f^*$ even after removing nonvital edge e from G.

4 A Mincut Cover for All Vital Edges

For any subset $\mathcal{E}$ of vital edges, let $V(\mathcal{E})$ denote the smallest set of vertices such that for each edge $(u,v) \in \mathcal{E}$, both u and v belong to $V(\mathcal{E})$. We establish an upper bound on the cardinality of the mincut cover of $\mathcal{E}$ in terms of $|V(\mathcal{E})|$.

Let e be a vital edge, and let C be a mincut for e. In undirected graphs, recall that each edge belonging to the edge-set of C is a contributing edge of C. However, in directed graphs, there may exist edges incoming to C. Any such incoming edge, say e', is not a contributing edge of C, and certainly, any mincut for edge e' is different from C. Can e' be a vital edge? The following lemma answers this question in negation. It crucially exploits the properties of (s,t)-maxflow across C as stated in GenFlowCut Property (Theorem 10).

▶ **Lemma 12.** *If C is a mincut for a vital edge, each incoming edge of C must be a nonvital edge.*

Proof. Let e' be an incoming edge of C. It follows from Theorem 10(1) that there is a maximum (s,t)-flow f such that $f_{in}(C) = 0$. So e' does not carry any flow in maximum (s,t)-flow f. Hence, it follows from Lemma 8 that e' is not a vital edge in G. ◀

The following lemma, which can be seen as a corollary of Lemma 12, establishes that a mincut for a vital edge partitions all the vital edges into three sets.

▶ **Lemma 13.** *Let $G = (V, E)$ be a directed weighted graph with a designated source vertex s and a designated sink vertex t. Let $\mathcal{E}$ be a subset of vital edges. For any edge $e \in \mathcal{E}$, let C be a mincut for e. C partitions the entire set $\mathcal{E}$ into three subsets – (i) $\mathcal{E}_C$: edges that are contributing to C, (ii) $\mathcal{E}_L$: edges whose both endpoints belong to C, and (iii) $\mathcal{E}_R$: edges whose both endpoints belong to $\overline{C}$.*

Let $\mathcal{M}(\mathcal{E})$ denote the mincut cover for a subset $\mathcal{E}$ of vital edges. Suppose Lemma 13 additionally guarantees the following property.

$\mathcal{P}$: *For each vital edge $e' \in \mathcal{E}$ that contributes to C, C is a mincut for e' as well.*

Property $\mathcal{P}$ ensures that C is a mincut for all vital edges that belong to $\mathcal{E}_C$. This implies that $\mathcal{M}(\mathcal{E}_L) \cup \mathcal{M}(\mathcal{E}_R) \cup \{C\}$ is a mincut cover for $\mathcal{E}$ as well. Therefore, the cardinality of mincut cover for set $\mathcal{E}$ can be bounded as follows.

$$|\mathcal{M}(\mathcal{E})| \leq |\mathcal{M}(\mathcal{E}_L)| + |\mathcal{M}(\mathcal{E}_R)| + 1 \tag{5}$$

Let $\mathcal{N}(\mu)$ denote the cardinality of a mincut cover for any set $\mathcal{E}$ of vital edges with $\mu = |V(\mathcal{E})|$. Note that $\mathcal{E} = \emptyset$ implies $\mu = 0$, and $\mathcal{E} \neq \emptyset$ implies $\mu \geq 2$. Equation 5, Lemma 13, and Property $\mathcal{P}$ lead to the following recurrence for $\mathcal{N}(\mu)$.

$$\text{Base case:} \qquad \mathcal{N}(0) = 0.$$

$$\text{For any } \mu \geq 2, \qquad \mathcal{N}(\mu) \leq 1 + \mathcal{N}(\mu_1) + \mathcal{N}(\mu_2), \quad \text{where } \mu_1 = |V(\mathcal{E}_L)|,\ \mu_2 = |V(\mathcal{E}_R)| \tag{6}$$

Note that $\mu_1, \mu_2 < \mu$, and $\mu_1 + \mu_2 \leq \mu$. Using induction on μ, it is a simple exercise to show that $\mathcal{N}(\mu) \leq \mu - 1$ for all $\mu \geq 2$.

Though property $\mathcal{P}$ is crucially used in establishing an upper bound on the mincut cover of a set of vital edges, Lemma 13, in its current form, does not guarantee it. However, we can enforce $\mathcal{P}$ easily as follows. In Lemma 13, the mincut for edge e is of the least capacity among mincuts for all edges in $\mathcal{E}$.

Using Recurrence 6 for the entire set of vital edges, we get an upper bound of $n-1$ on the cardinality of the mincut cover for all vital edges in G. This leads to the following theorem that answers Question 3 in the affirmative.

▶ **Theorem 14** (Mincut Cover)**.** *For any directed weighted graph G on n vertices with a designated source vertex s and a designated sink vertex t, there exists a set $\mathcal{C}_{min}$ containing at most $n-1$ (s,t)-cuts such that, for any vital edge e in G, at least one mincut for edge e is present in set $\mathcal{C}_{min}$.*

The following note emphasizes the need of property $\mathcal{P}$ in establishing the upper bound derived above.

▶ Note 15. Selecting any arbitrary edge $e \in \mathcal{E}$ might skip some mincuts for vital edges. Refer to Figure 1(ii) on Page 3. The set of vital edges, denoted by $\mathcal{E}$, contains four vital edges (s,a), (s,b), (a,t), and (b,t). Suppose we first select vital edge (s,a) and mincut $A = \{s\}$ for edge (s,a). It follows from Lemma 13 that $\mathcal{E}$ is partitioned into $\mathcal{E}_L$, $\mathcal{E}_R$ and $\mathcal{E}_A$. Observe that $\mathcal{E}_L$ does not contain any vital edge, vital edges (a,t) and (b,t) belong to $\mathcal{E}_R$, and (s,b) belongs to $\mathcal{E}_A$. We now recurse on set $\mathcal{E} = \mathcal{E}_R$ and select the vital edge (b,t). Mincut $C = \{s,a,b\}$ partitions $\mathcal{E}$ into $\mathcal{E}_L$, $\mathcal{E}_R$, and $\mathcal{E}_C$. Observe that both $\mathcal{E}_L$ and $\mathcal{E}_R$ do not contain any vital edge, and vital edge (a,t) belongs to $\mathcal{E}_C$. The process terminates as $V(\mathcal{E}_L)$ and $V(\mathcal{E}_R)$ are empty. We get a pair of (s,t)-cuts A and C as mincut cover for $\mathcal{E}$. Unfortunately, it does not contain mincut $B = \{s,a\}$ for edges (a,t) and (s,b).

5 Optimal Sensitivity Oracles for (s,t)-mincut

Let $e = (u, v)$ be an edge in G, and we wish to determine the impact of the failure of e on (s, t)-mincut. We begin with a brief overview of $\mathcal{O}(n)$ space sensitivity oracle [4] for (s, t)-mincuts in unweighted graphs. By construction of $\mathcal{D}_{PQ}(G)$ [30], there is a mapping from the vertices of G to the nodes of $\mathcal{D}_{PQ}(G)$ such that the following assertion holds – u and v are mapped to the same node of $\mathcal{D}_{PQ}(G)$ if and only if there is no (s, t)-mincut in G that separates u and v. In $\mathcal{D}_{PQ}(G)$, each 1-transversal cut is an (s, t)-mincut in G. Exploiting this property, it is shown in [4] that there is an (s, t)-mincut to which edge (u, v) contributes if and only if the node containing u succeeds the node containing v in any topological ordering of $\mathcal{D}_{PQ}(G)$. If u and v are mapped to the same node of $\mathcal{D}_{PQ}(G)$, it follows that the mincut for edge (u, v) has a capacity strictly larger than that of the (s, t)-mincut. Therefore, if G is unweighted, the failure of edge (u, v) does not reduce the capacity of the (s, t)-mincut; so (u, v) is not a vital edge. Thus, the mapping from V to the nodes of $\mathcal{D}_{PQ}(G)$ along with its topological ordering serves as an $\mathcal{O}(n)$ space sensitivity oracle in unweighted graphs.

If G is weighted, it is still possible that though u and v mapped to the same node of $\mathcal{D}_{PQ}(G)$, yet the failure of edge (u, v) reduces (s, t)-mincut. In other words, if G is weighted, there may be many vital edges internal to the nodes of $\mathcal{D}_{PQ}(G)$ (refer to Figure 2). Thus $\mathcal{D}_{PQ}(G)$ fails to serve as sensitivity oracle for a weighted graph G; hence we need to explore the *structure* of cuts *internal* to a node of $\mathcal{D}_{PQ}(G)$.

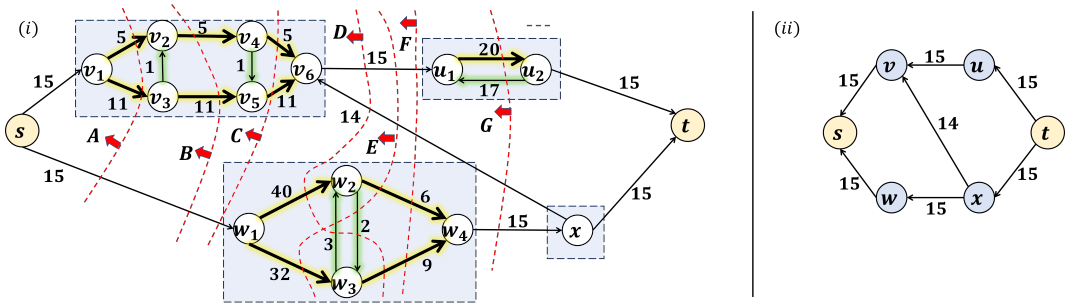


Figure 2 (i) A graph H and (ii) $\mathcal{D}_{PQ}(H)$. Thick edges in (i) represent the vital edges of H that are internal to the nodes of $\mathcal{D}_{PQ}(H)$. A mincut for them is represented by dashed curves.

Let $\textsc{cap}(e, \Delta)$ denote the query for reporting the capacity of (s, t)-mincut after reducing the capacity of $e \in E$ by Δ such that $0 \leq \Delta \leq w(e)$. We now address the design of a compact data structure that can efficiently answer query $\textsc{cap}(e, \Delta)$ for G.

Let us first consider only the set of vital edges. There may exist $\Omega(n^2)$ vital edges in a graph (refer to Figure 4(ii) on Page 17). Therefore, explicitly storing the capacity of a mincut for each vital edge would occupy $\Omega(n^2)$ space. To design an $\mathcal{O}(n)$ space data structure, we use the following lemma for vital edges. This lemma follows from Lemma 13 and the discussion we used for bounding the size of mincut cover for vital edges.

▶ **Lemma 16.** *Let $U \subseteq V$ and an edge $e \in E_{vit}(U)$. If mincut for e, say C, has the least capacity among the mincuts for all vital edges from $E_{vit}(U)$, then, for any edge $e' \in E_{vit}(U)$, either C is a mincut for e' or both endpoints of e' belong to $U \cap C$ or $U \cap \overline{C}$.*

Using Lemma 16, we design a divide and conquer based algorithm to build a data structure that compactly stores the capacity of mincut for each vital edge as follows.

Let e^* be an edge in $E_{vit}(V)$ such that the capacity of mincut for e^* is less than or equal to the capacity of mincut for each vital edge in G. Let C be a mincut for e^*. C serves as the mincut for each vital edge contributing to C. For the mincuts of the remaining vital edges, we recursively process $E_{vit}(V \cap C)$ and $E_{vit}(V \cap \overline{C})$. Refer to Algorithm 1 for the pseudocode of this recursive algorithm.

It can be observed that the data structure resulting from Algorithm 1 is a rooted binary tree. We refer to it as $\mathcal{T}_{vit}(G)$ henceforth. Its leaves are associated with disjoint subsets of the entire vertex set V – for each vertex $v \in V$, $\mathcal{L}(v)$ stores the pointer to the leaf to which v is mapped. Each internal node ν in $\mathcal{T}_{vit}(G)$ has the following three fields.

- $\nu.cap$ is the capacity of mincut for the vital edge selected during the recursive call in which node ν is created.
- $\nu.left$ is the left child of ν and $\nu.right$ is the right child of ν.

Algorithm 1 : TreeConstruction(V) computes $\mathcal{T}_{vit}(G)$.

```
1:  procedure TreeConstruction(U)
2:      Create a node ν;
3:      if E_vit(U) = ∅ then
4:          for each x ∈ U do L(x) ← ν;
5:          end for
6:      else
7:          Let C(e) denote a mincut for edge e;
8:          Select an edge e* ∈ E_vit(U) such that c(C(e*)) ≤ c(C(e')) ∀e' ∈ E_vit(U);
9:          Assign ν.cap ← c(C(e*));
10:         ν.left ← TreeConstruction(U ∩ C(e*));
11:         ν.right ← TreeConstruction(U ∩ C̄(e*));
12:     end if
13:     return ν;
14: end procedure
```

The following observation is immediate from the construction of $\mathcal{T}_{vit}(G)$.

▶ **Observation 17.** *Let ν be an internal node in $\mathcal{T}_{vit}(G)$ and C be the (s,t)-cut associated with node ν (chosen in Step 8 of Algorithm 1). For a vertex u, $\mathcal{L}(u)$ belongs to the subtree rooted at $\nu.left$ if and only if $u \in C$.*

Observe that $\mathcal{T}_{vit}(G)$ is a full binary tree with at most n leaves. So the number of internal nodes is at most $n-1$. Moreover, every internal node of $\mathcal{T}_{vit}(G)$ stores $\mathcal{O}(1)$ information. Hence, we can state the following theorem.

▶ **Theorem 18.** *Let $G = (V, E)$ be a directed weighted graph on $n = |V|$ vertices with designated source vertex s and designated sink vertex t. There is a rooted full binary tree occupying $\mathcal{O}(n)$ space that stores the capacity of mincut for each vital edge.*

Let $e = (x, y)$ be a query edge. If e is a vital edge, it follows from the construction of tree $\mathcal{T}_{vit}(G)$ in Theorem 18 that the LCA of $\mathcal{L}(x)$ and $\mathcal{L}(y)$ stores the capacity of mincut for (x, y). So, using the efficient data structure for LCA [7], it takes $\mathcal{O}(1)$ time to answer CAP(e, Δ) for any vital edge e. However, what if the query edge is a nonvital edge? Although there is no impact on (s,t)-mincut capacity due to the reduction in the capacity of a nonvital edge, tree $\mathcal{T}_{vit}(G)$ stores no information explicitly about nonvital edges. Therefore, in order to answer CAP(e, Δ) for any edge, it seems a classification of all edges as vital or nonvital is required. Unfortunately, any such explicit classification would require $\Omega(m)$ space, which is also trivial for answering query CAP as discussed above.

Interestingly, $\mathcal{T}_{vit}(G)$ can answer CAP(e, Δ) for any edge e without any such classification. This is because, as shown in the following lemma, it is already capable of classifying an edge to be vital or nonvital.

▶ **Lemma 19.** *Let $e = (x, y)$ be any edge in G such that $\mathcal{L}(x) \neq \mathcal{L}(y)$. Let ν be the* LCA *of $\mathcal{L}(x)$ and $\mathcal{L}(y)$ in $\mathcal{T}_{vit}(G)$, and let C be the (s,t)-cut associated with node ν in $\mathcal{T}_{vit}(G)$. Edge (x, y) is a nonvital edge if and only if exactly one of the following two conditions is satisfied.*

1. *$\mathcal{L}(x) \in \nu.right$ and $\mathcal{L}(y) \in \nu.left$.*
2. *$\mathcal{L}(x) \in \nu.left$, $\mathcal{L}(y) \in \nu.right$, and $c(C) - w(e) \geq f^*$.*

Proof. Suppose edge (x, y) is a nonvital edge. Observe that $\mathcal{L}(x)$ and $\mathcal{L}(y)$ belong to the subtree rooted at ν in $\mathcal{T}_{vit}(G)$ because ν is the LCA of $\mathcal{L}(x)$ and $\mathcal{L}(y)$. It follows from the construction of $\mathcal{T}_{vit}(G)$ (Algorithm 1) that edge (x, y) either contributes to C or is an incoming edge to C. Suppose (x, y) is a contributing edge to C. So using Observation 17, $\mathcal{L}(x) \in \nu.left$ and $\mathcal{L}(y) \in \nu.right$. If $c(C)$ falls below f^* upon removal of edge (x, y), (x, y) would be a vital edge by Definition 1. Hence, $c(C) - w(e) \geq f^*$. So condition (2) is satisfied. Suppose (x, y) is an incoming edge to C, that is, $x \notin C$ and $y \in C$. It follows from Observation 17 that $\mathcal{L}(x)$ must belong to the right subtree and $\mathcal{L}(y)$ must belong to the left subtree of ν. So condition (1) is satisfied.

We now prove the converse part. Suppose condition (1) is satisfied, that is, $\mathcal{L}(x) \in \nu.right$ and $\mathcal{L}(y) \in \nu.left$. Observation 17 implies that $x \in \overline{C}$ and $y \in C$. Hence, edge (x, y) is an incoming edge of C. It follows from the construction of $\mathcal{T}_{vit}(G)$ that C is a mincut for some vital edge. So, by Theorem 10(1), there exists a maximum (s,t)-flow f such that $f_{in}(C) = 0$. Hence $f(x, y) = 0$. So it follows from Lemma 8 that (x, y) is a nonvital edge. Suppose condition (2) is satisfied. Hence (x, y) is an outgoing edge of C. Assume to the contrary that e is a vital edge. It follows from the construction of $\mathcal{T}_{vit}(G)$ that C is a mincut for e. Since e is a vital edge, by Definition 1, $c(C) - w(e) < f^*$, a contradiction. ◀

For any edge $e \in E$ and $0 \leq \Delta \leq w(e)$, Algorithm 2 verifies conditions of Lemma 19 to answer query CAP(e, Δ) in $\mathcal{O}(1)$ time using $\mathcal{T}_{vit}(G)$. So we can state the following Theorem.

Algorithm 2 Reporting the capacity of (s,t)-mincut after decreasing the capacity of an edge.

```
1:  procedure CAP((x, y), Δ)
2:      if L(x)==L(y) then
3:          return f*
4:      else
5:          ν ← LCA of L(x) and L(y) in T_vit(G).
6:      end if
7:      NewCapacity ← ν.cap − Δ
8:      if (L(x) ∈ ν.left ∧ L(y) ∈ ν.right ) ∧ NewCapacity < f* then
9:          return NewCapacity.
10:     else
11:         return f*.
12:     end if
13: end procedure
```

▶ **Theorem 20** (Sensitivity Oracle for Reporting Capacity)**.** *Let G be a directed weighted graph on n vertices with a designated source vertex s and a designated sink vertex t. There is an $\mathcal{O}(n)$ space data structure that, given any edge $e \in E$ and a value Δ satisfying $0 \leq \Delta \leq w(e)$, can report in $\mathcal{O}(1)$ time the capacity of (s,t)-mincut after reducing the capacity of e by Δ.*

Reporting an (s,t)-mincut. An internal node, say ν, of $\mathcal{T}_{vit}(G)$ stores only the capacity of the mincut for a vital edge. We augment ν with a mincut for the edge that was picked by Algorithm 1 in Step 8. The size of the resulting data structure is $\mathcal{O}(n^2)$. Observe that the condition $\Delta \leq w(e)$ can be verified for any edge e in $\mathcal{O}(1)$ time if we store the capacity of all edges of G. This will require only additional $\mathcal{O}(m) = \mathcal{O}(n^2)$ space. So we can state the following theorem.

▶ **Theorem 21** (Sensitivity Oracle)**.** *Let G be a directed weighted graph on n vertices with a designated source vertex s and a designated sink vertex t. There is an $\mathcal{O}(n^2)$ space data structure that, given any edge e and any value $\Delta \geq 0$, can report*
1. *the capacity of (s,t)-mincut in $\mathcal{O}(1)$ time, and*
2. *an (s,t)-mincut C in $\mathcal{O}(|C|)$ time for the resulting graph*
3. *the DAG $\mathcal{D}_{PQ}$ occupying $\mathcal{O}(m)$ space in $\mathcal{O}(m)$ time (proof is in the full version)*

after reducing the capacity of the edge e by Δ.

▶ Remark 22. It is a simple exercise to show that our data structure in Theorem 21(1) and (2) can be extended to handle a vertex failure using the standard technique of vertex splitting.

Lower Bound. We complement the result in Theorem 21 by a matching lower bound, which holds even for undirected graphs as follows (refer to Section 8 for the proof).

▶ **Theorem 23.** *Any data structure for reporting the capacity of (s,t)-mincut after the failure of an edge in an (un)directed graph on n vertices with positive edge capacities must require $\Omega(n^2 \log\ n)$ bits of space in the worst case, irrespective of the query time.*

▶ Remark 24. The $\mathcal{O}(n)$ space upper bound in Theorem 20 does not violate the $\Omega(n^2)$ space lower bound in Theorem 23. This is because Theorem 20 assumes that the query edge e belongs to E and the reduction in capacity of e is at most $w(e)$. However, this assumption seems practically justified since, in real world, the capacity of an edge can decrease only if the edge *actually* exists, and furthermore, it can decrease by an amount at most the capacity of the edge.

Labeling Scheme. Let f be any function defined on the vertex/edge set of the graph. A labeling scheme of f assigns small labels to each vertex/edge of the graph in such a way that, for any given subset A of edges/vertices, $f(A)$ can be computed only by using the labels of vertices/edges in A. A labeling scheme of $\mathcal{O}(\log^2\ n + \log\ n \log\ W)$ bits can be designed for reporting capacity of (s,t)-mincut after decreasing capacity of any edge in G using Theorem 20 and a labeling scheme for LCA query [25]. Here W is the maximum capacity of any edge in G. The proof is in the full version.

6 An Algorithm for Computing All Vital Edges

The existing algorithms [2, 3] that compute the most vital edge or all vital edges in undirected graphs take *cut-based* approaches as follows. For computing the most vital edge in an undirected graph, Aneja, Chandrasekaran, and Nair [2] establish that there exists an (s,t)-cut such that the most vital edge is the maximum capacity edge among all the edges that contribute to the cut. Ausiello, Franciosa, Lari, and Ribichini [3] define function F on cuts as in Equation 1 in order to use the ancestor tree of Cheng and Hu [9]. To compute all vital edges efficiently in a directed weighted graph, we take a *flow-based* approach that analyzes flow along a set of edges in a maximum (s,t)-flow. We first classify all vital edges into two types based on the maximum flow that an edge can carry in any maximum (s,t)-flow.

A classification of vital edges. In graph G, there may exist multiple (s,t)-flows attaining value f^*. The flow along an edge may be different in these multiple maximum (s,t)-flows. A vital edge is a *tight* edge if it carries flow equal to its capacity in at least one maximum (s,t)-flow; otherwise, it is a *loose* edge.

Our algorithm for computing all vital edges makes use of the classification of vital edges, and employs two results that were derived for solving two seemingly *unrelated* problems.

To compute all the loose edges, we crucially exploit the following result on maximum (s,t)-flow, which has been used for solving *mincost maximum* (s,t)*-flow problem.*

▶ **Lemma 25** (Theorem 11.1 and Theorem 11.2 in [1]). *For any directed weighted graph G, there exists a maximum (s,t)-flow $f^{\#}$ such that the edges that carry nonzero flow but not fully saturated are at most $n-1$.*

For any maximum (s,t)-flow, by definition, the set of all loose edges is a subset of all edges that carry nonzero flow but are not fully saturated. So, by Lemma 25, the number of loose edges can be at most $n-1$. Moreover, there is an algorithm that, given any maximum (s,t)-flow, can compute $f^{\#}$ of Lemma 25 in $\mathcal{O}(mn)$ time [1]. As a result, all the loose edges of G can be computed using $\mathcal{O}(n)$ maximum (s,t)-flow computations.

Observe that the efficient computation of the set of loose edges crucially exploits the fact that there can be at most $n-1$ loose edges. However, there can be $\Omega(n^2)$ tight edges in a graph (refer to Figure 4(*ii*) on page 17). Hence, to compute all the vital edges, the main challenge arises in computing all the tight edges of G. We now state a property of tight edges that plays a crucial role in their efficient computation.

A property satisfied by tight edges

Let E_{min} be the set of all edges that contribute to (s,t)-mincuts. Set E_{min} is a subset of all the tight edges since all edges in E_{min} are fully saturated in every maximum (s,t)-flow. Exploiting the strong duality between maximum (s,t)-flow & (s,t)-mincut and Lemma 5, it can be observed that each edge $(u,v) \in E_{min}$ satisfies the following property. If C is any (s,t)-cut such that $v \in C$ and $u \in \overline{C}$, then $c(C)$ is strictly greater than f^*. The following lemma shows that this property is satisfied by each tight edge as well.

▶ **Lemma 26.** *Let $e=(u,v)$ be a tight edge. Let C be a mincut for edge e and $C_{v,u}$ be an (s,t)-cut of the least capacity that keeps v on the side of s and u on the side of t. Then, $c(C) < c(C_{v,u})$.*

Proof. Edge (u,v) is a tight edge, so by definition, there is a maximum (s,t)-flow f such that $f(e)$ is equal to $w(e)$. This would imply that $f_{in}(C_{v,u})$ is at least $w(e)$ since edge (u,v) is an incoming edge for cut $C_{v,u}$. It follows from Lemma 5 that $f_{out}(C_{v,u}) - f_{in}(C_{v,u}) = f^*$. Therefore, $f_{out}(C_{v,u}) - w(e) \geq f^*$. Since the capacity of an (s,t)-cut is an upper bound on the value of any outgoing flow of the (s,t)-cut, therefore, $f_{out}(C_{v,u}) \leq c(C_{v,u})$. Hence we arrive at the following inequality.

$$c(C_{v,u}) \geq f^* + w(e) \tag{7}$$

It follows from Theorem 10(2) that $w_{min}(e)$ is the minimum amount of flow that must pass through edge (u,v) to get a maximum flow of value f^*. Let f' be the value of maximum (s,t)-flow in graph $G \setminus \{e\}$. Thus we arrive at the following equality.

$$f^* = f^{'} + w_{min}(e) \tag{8}$$

We now add $(w(e) - w_{min}(e))$ on both sides of Equation 8 and get the following equation.

$$f^* + w(e) - w_{min}(e) = f^{'} + w(e) \tag{9}$$

It follows from Theorem 10(2) that $f^{'}$ is also equal to the sum of capacities of all edges contributing to C except edge (u, v). Therefore, $f^{'} + w(e)$ is equal to the capacity of C in G. So it follows from Equation 9 that $f^* + w(e) - w_{min}(e) = c(C)$. Since (u, v) is a vital edge, therefore, $w_{min}(e) > 0$ using Observation 6. Hence,

$$f^* + w(e) > c(C) \tag{10}$$

It follows from Inequality 7 and Inequality 10 that $c(C_{v,u}) > c(C)$. ◀

For any tight edge (u, v), let C be an (s, t)-cut of the least capacity that separates vertex u and vertex v. It follows from Lemma 26 that C is also a mincut for edge (u, v). So we can state the following theorem.

▶ **Theorem 27.** *Let G be a directed weighted graph with a designated source vertex s and a designated sink vertex t. For any tight edge $e = (u, v)$ in G, a mincut for edge e is identical to an (s, t)-cut of the least capacity that separates vertex u and vertex v.*

Algorithm for computing tight edges

For the given directed weighted graph G, we define function F as in Equation 1 and then compute the Ancestor tree of Cheng and Hu [9] for (s, t)-cuts. For each pair of vertices, $\mathcal{T}_{(s,t)}$ stores an (s, t)-cut of the least capacity separating them at their LCA. This tree, denoted by $\mathcal{T}_{(s,t)}$, can be built using $\mathcal{O}(n)$ maximum (s, t)-flow computations [9]. We augment $\mathcal{T}_{(s,t)}$ with a data structure for LCA queries [7].

To compute all tight edges, we process $\mathcal{T}_{(s,t)}$ and the edges of G as follows. Let (u, v) be an edge in G. We perform LCA query on $\mathcal{T}_{(s,t)}$ for u and v to get the (s, t)-cut, say C, of the least capacity that separates u and v. We determine whether C satisfies the following two conditions.

1. Edge (u, v) contributes to C.
2. $c(C) - w((u, v)) < f^*$

If both conditions are satisfied, by Definition 1, (u, v) is a vital edge. Observe that there may be vital edges that do not satisfy Condition (1); refer to Figure 3(i). However, it follows from Theorem 27 that each tight edge does satisfy these two conditions. After processing all edges of G, let S be the resulting set of vital edges that satisfy both these conditions. We eliminate from S all the loose edges to get all the tight edges.

▶ Remark 28. In an undirected graph, observe that a mincut for an edge (u, v) is always an (s, t)-cut of the least capacity that separates u and v. Therefore, computing all vital edges in an undirected graph amounts to just verifying condition (2) for each edge as shown by Ausiello, Lari, Franciosa, and Ribichini [3].

Thus we can state the following theorem.

▶ **Theorem 29.** *For any directed weighted graph G on n vertices with a designated source vertex s and a designated sink vertex t, there is an algorithm that computes all tight edges of G using $\mathcal{O}(n)$ maximum (s, t)-flow computations.*

Theorem 29 and the discussion on computing the loose edges lead to Theorem 30.

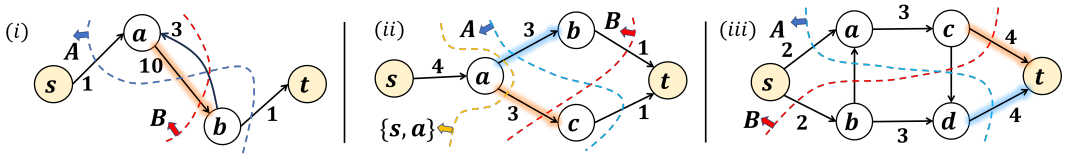


Figure 3 (i) A is the (s,t)-cut of least capacity that separates a and b. Edge (a,b) is a vital edge, and it is incoming to A. (ii) $\mathcal{D}_{vit}(G)$=G. $\{s,a\}$ is a 1-transversal cut, but not a mincut for any vital edge. (iii) mincuts A and B for vital edges (c,t) and (d,t), respectively, are not closed under union. Moreover, edge (b,a) (likewise edge (c,d)) contributes to A (likewise to B) and is incoming to B (likewise to A).

▶ **Theorem 30** (Computing All Vital Edges). *For any directed weighted graph on n vertices with a designated source vertex s and designated sink vertex t, there is an algorithm that computes all vital edges and their vitality using $\mathcal{O}(n)$ maximum (s,t)-flow computations.*

▶ Note 31. We present the following applications of our algorithm in Theorem 30 (refer to the full version).

1. Observe that data structure $\mathcal{T}_{(s,t)}$ also stores a mincut for every tight edge. Therefore, an (s,t)-mincut in graph $G\setminus\{e\}$ for every tight edge e can be computed using $\mathcal{O}(n)$ maximum (s,t)-flow computations. Moreover, since loose edges are at most $n-1$, therefore, using $\mathcal{O}(n)$ maximum (s,t)-flow computations, we can compute an (s,t)-mincut in graph $G\setminus\{e\}$ for every vital edge e of G.
2. Given a mincut for all vital edges, using Algorithm 1, the process of constructing $\mathcal{T}_{vit}(G)$ and the data structure in Theorem 21(2) requires $\mathcal{O}(m \log\ n + n^2)$ time. As a byproduct, data structure in Theorem 21(2) provides a mincut cover for all vital edges.
3. As stated in (2), we can construct $\mathcal{T}_{vit}(G)$ using $\mathcal{O}(n)$ maximum (s,t)-flow computations. Given any $k \in [m]$, using $T_{vit}(G)$ and a maxheap data structure, the k most vital edges can be reported in $\mathcal{O}(m + k \log\ k)$ time.

▶ Note 32. Given any edge e and its capacity $w(e)$, the data structure in Theorem 20 can be used to report the vitality of e in $\mathcal{O}(1)$ time by assigning $\Delta = w(e)$.

7 Compact Structures for all Mincuts for all Vital Edges

The set containing all (s,t)-mincuts satisfies two important properties – (i) closed under both intersection and union and (ii) an edge contributing to an (s,t)-mincut can never be an incoming edge to another (s,t)-mincut. These two properties are exploited crucially in the design of DAG structures for compactly storing and characterizing all (s,t)-mincuts [30, 4] using 1-transversal cuts. Unfortunately, none of these properties holds for the set of all mincuts for all vital edges (refer to Figure 3(iii)). This makes the problem of designing compact structures for mincuts for all vital edges challenging. We present two structures for storing and characterizing all mincuts for all vital edges – one is a single DAG that provides a *partial* characterization and the other consists of a set of $\mathcal{O}(n)$ DAGs that provides a complete characterization.

(a) An $\mathcal{O}(m)$ Space DAG for Partial Characterization. We first show that the approaches taken in [30, 4] are not sufficient for designing a compact structure for all mincuts for all vital edges. In particular, the resulting graph $Q'(G)$, obtained from G after applying the techniques from [30, 4], is still not acyclic. Moreover, a mincut for a vital edge can have

unbounded transversality (refer to the full version) in $Q'(G)$. It turns out that the source of this problem is the set of all edges that are contributing to a mincut for a vital edge as well as incoming to a mincut for another vital edge. The set of such edges is denoted by Γ-*edges*. Exploiting GenFlowCut property crucially, we show that all edges in Γ-edges are nonvital. Thereafter, counter intuitive to Note 2, we show that the graph obtained after the removal of all Γ-edges from G still preserves all (s,t)-mincuts of G, and provides a partial characterization for all mincuts for all vital edges as follows.

▶ **Theorem 33** (Partial Characterization). *For a directed weighted graph G on m edges with a designated source vertex s and a designated sink vertex t, there is an $\mathcal{O}(m)$ space directed acyclic graph $\mathcal{D}_{vit}(G)$ that preserves the capacity of (s,t)-mincut, and for each vital edge e in G, (1) Every mincut for edge e in G is a 1-transversal cut in $\mathcal{D}_{vit}(G)$ and (2) A mincut C for e in G appears as a relevant cut in $\mathcal{D}_{vit}(G)$, that is, $c(C) - w(e) < f^*$ in $\mathcal{D}_{vit}(G)$.*

▶ Note 34. Existing DAG structures for (s,t)-mincuts [30, 4] turn out to be just a special case of DAG $\mathcal{D}_{vit}(G)$ in Theorem 33. Moreover, $\mathcal{D}_{vit}(G)$ additionally guarantees the property (2) in Theorem 33, which does not hold in the existing DAGs [30, 4] (refer to full version).

(b) An $\mathcal{O}(mn)$ Space Structure for Complete Characterization. Although each mincut for every vital edge is a 1-transversal cut in $\mathcal{D}_{vit}(G)$, it is not necessary that each 1-transversal cut in $\mathcal{D}_{vit}(G)$ is a mincut for a vital edge (refer to Figure 3(ii)). Hence, $\mathcal{D}_{vit}(G)$ could provide only a partial characterization. Let $e = (u,v)$ be any vital edge, and H_e denote the graph obtained by adding two infinite weight edges, (s,u) and (v,t), in graph G. Observe that all mincuts for (u,v) are compactly stored in the DAG for (s,t)-mincuts [30] built on graph H_e. We denote this DAG by $\mathcal{D}_{PQ}(H_e)$. It can be seen that keeping $\mathcal{D}_{PQ}(H_e)$ for each vital edge e serves as a compact structure for storing all mincuts for all vital edges and characterizing them in terms of 1-transversal cuts. However, this structure may take $\mathcal{O}(m^2)$ space in the worst case. We present an $\mathcal{O}(mn)$ space structure $S_{vit}(G)$. This structure consists of $\mathcal{O}(n)$ DAGs for storing and characterizing all mincuts for all vital edges in terms of 1-transversal cuts. To build the structure $\mathcal{S}_{vit}(G)$, the main challenge turns out to be designing a compact structure for storing and characterizing all mincuts for all tight edges. It follows from Theorem 27 that any tight edge $e = (u,v)$ contributes to the (s,t)-cut of the least capacity separating u and v. Therefore, a solution to the following problem would suffice to overcome the challenge.

▶ **Problem 1.** *Given a directed weighted graph G with a designated source s and a designated sink t, build a compact structure that, for each pair of vertices a and b in G, stores and characterizes all the (s,t)-cuts of the least capacity that separate a and b in G.*

To solve Problem 1, we address the problem of designing a compact structure for storing and characterizing all Steiner (s,t)-mincuts for a given Steiner set, which is of independent interest.

Compact Structure for all Steiner (s,t)-mincuts. Let $S \subseteq V$ be a Steiner set. An (s,t)-cut C is a *Steiner (s,t)-cut* if $C \cap S$ and $\overline{C} \cap S$ are nonempty. A Steiner (s,t)-cut with the least capacity is a *Steiner (s,t)-mincut*. The main hurdle in designing a compact structure for Steiner (s,t)-mincuts is that, unlike (s,t)-mincuts, they are not closed under intersection/union. We overcome this hurdle by generalizing the *covering* technique from [4] to directed weighted graphs and obtain a structure occupying $\mathcal{O}(|S|m)$ space.

Finally, we design a hierarchy tree, which is a generalization of the hierarchy tree of [25], and augment it with the structure for Steiner (s,t)-mincuts. It leads to the following result.

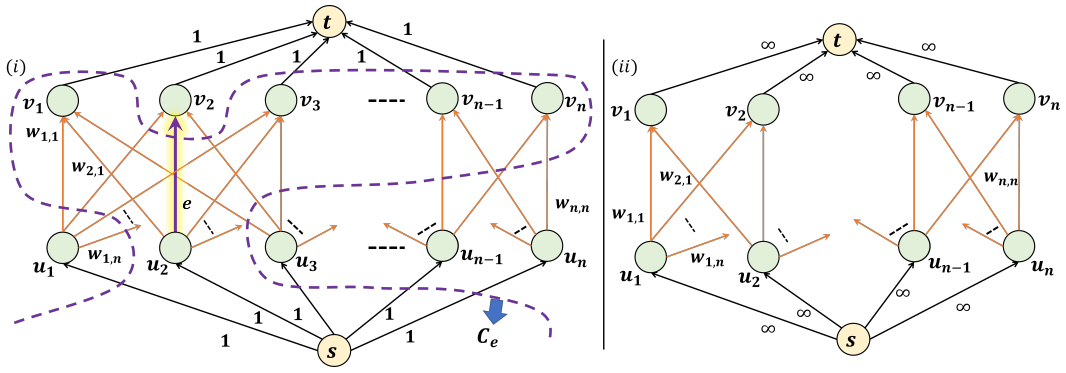


Figure 4 (i) Graph $\mathcal{G}$. The dashed curve represents the mincut for edge (u_2, v_2). (ii) Graph $G(M)$. Each edge (u_i, v_j), $1 \leq i, j \leq n$, is a vital as well as tight edge.

▶ **Theorem 35** (Complete Characterization). *For any directed weighted graph G on n vertices and m edges with a designated source vertex s and a designated sink vertex t, there is an $\mathcal{O}(mn)$ space structure $\mathcal{S}_{vit}(G)$ consisting of $\mathcal{O}(n)$ DAGs such that for any vital edge e,*
1. There exists at most two DAGs, $\mathcal{D}^s(e)$ and $\mathcal{D}^t(e)$, that store all mincuts for edge e.
2. An (s,t)-cut C in G is a mincut for edge e if and only if C is a 1-transversal cut in either $\mathcal{D}^s(e)$ or $\mathcal{D}^t(e)$ and e is a contributing edge of C.

▶ Note 36. Problem 1 is addressed in [4] for unweighted graphs and only for those pair of vertices for which the capacity of (s,t)-cut of the least capacity separating them is minimum+1. An $\mathcal{O}(mn)$ space structure is designed in [4] for this special case of Problem 1. Hence, Theorem 35 provides a generalization of the result in [4] for weighted graphs while matching the space bound. An important application of structure $\mathcal{S}_{vit}(G)$ is Theorem 21(3) (refer to the full version).

8 Lower Bounds

In this section, we first provide a lower bound on the worst case size of mincut cover for all edges. Later, we give a lower bound on the space for any data structure that can report the capacity of (s,t)-mincut after the failure of any edge in both undirected and directed graphs.

8.1 Mincut Cover for All Edges

In this section, we establish the following theorem.

▶ **Theorem 37.** *There exists a directed weighted graph $G = (V, E)$ on $n = |V|$ vertices such that the number of mincuts for all edges in G is $\Omega(n^2)$.*

In order to prove Theorem 37, we construct a graph $\mathcal{G}$ on $2n + 2$ vertices with a set of $\Omega(n^2)$ edges E' that has the following property. For each pair of edges $e, e' \in E'$, the capacity of mincut for edge e is different from the capacity of mincut for edge e' (refer to Figure 4(i)).

Construction of $\mathcal{G}$. The vertex set consists of two disjoint sets $A = \{u_1, u_2, \ldots, u_n\}$ and $B = \{u_1, u_2, \ldots, u_n\}$ of n vertices each, along with a source vertex s and a sink vertex t. The edges are defined as follows. There is an edge with unit capacity from vertex s to each vertex in A. Likewise, there is a unit capacity edge from each vertex in B to sink t. For each $u \in A$ and $v \in B$, there is an edge (u, v). Let E' be the set of all edges from A to B. Each edge in E' is assigned a unique capacity from $[n^2, 2n^2]$. It can be observed that $w(e_1) \neq w(e_2)$ for every pair of edges $e_1, e_2 \in E'$.

▶ **Lemma 38.** *For every pair of edges e_1 and e_2 from E', the capacity of mincut for e_1 is different from the capacity of mincut for e_2.*

Proof. Let $e = (u, v)$ be an edge from E'. Let us define an (s, t)-cut C_e in which edge e contributes. C_e keeps each vertex $x \in A \setminus \{u\}$ on the side of t. Similarly, C_e keeps each vertex $x \in B \setminus \{v\}$ on the side of s. Therefore, the contributing edges of (s, t)-cut C_e are edge e, the edges from s to each vertex $x \in A \setminus \{u\}$, and the edges from each vertex $x \in B \setminus \{v\}$ to t. Hence the capacity of C_e is $w(e) + (n-1) + (n-1) = w(e) + 2n - 2$. Any other (s, t)-cut in which edge e contributes must have at least one more contributing edge e' from E'. Since $w(e')$ is at least n^2, therefore, C_e is the mincut for edge e. For any other edge e'' from E', in a similar way, the capacity of the mincut for e'' is $w(e'') + 2n - 2$, which is different from the capacity of C_e since $w(e) \neq w(e'')$. This completes the proof. ◀

Since $|E'| = \Omega(n^2)$, it follows from Lemma 38 that graph $\mathcal{G}$ has $\Omega(n^2)$ different capacities of mincuts for edges in E'. This completes the proof of Theorem 37.

8.2 Fault-tolerant (s,t)-mincut

In this section, we provide a lower bound on the space for any data structure that can report the capacity of (s, t)-mincut after the failure of any edge. Let M be a $n \times n$ matrix where for any $i, j \in [n]$, $M[i, j]$ stores an integer in the range $[1, n^c]$ for some constant $c > 0$. Given any instance of the matrix M, we construct the following undirected graph $G(M)$ (refer to Figure 4(*ii*)).

Description of $G(M)$. The vertex set is defined as follows. There is a source vertex s and a sink vertex t. For all the n rows of matrix M, there is a set R of n vertices $\{u_1, u_2, \ldots, u_n\}$. Similarly, For all the n columns of matrix M, there is a set C of n vertices $\{v_1, v_2, \ldots, v_n\}$. For each $i, j \in [n]$, there is an edge e_{ij} of capacity $w_{i,j} = M[i, j]$ between vertex u_i and v_j. For each $i \in [n]$, there is an edge of infinite capacity between source s and vertex u_i and there is an edge of infinite capacity between vertex v_i and sink t. Let λ be the capacity of (s, t)-mincut in $G(M)$.

We now establish a relation between the graph $G(M)$ and matrix M in the following lemma.

▶ **Lemma 39.** *For any $i, j \in [n]$, the capacity of (s, t)-mincut in $G(M)$ after the failure of edge (u_i, v_j) is $\lambda - M[i, j]$.*

Proof. In graph $G(M)$, $C = \{s\} \cup R$ is the only (s, t)-cut of finite capacity. Therefore, C is the (s, t)-mincut in graph $G(M)$. Observe that only edges (u_i, v_j) for any $0 < i, j \leq n$ contribute to C. Hence $\lambda = \sum_{i,j \in [n]} w((u_i, v_j))$. Therefore, upon failure of edge $e_{ij} = (u_i, v_j)$, the reduction in the capacity of (s, t)-mincut is the same as $w(e_{ij})$, which is $M[i, j]$ as follows from the construction of $G(M)$ described above. ◀

Let $F(G(M))$ be any data structure that can report the (s, t)-mincut after the failure of an edge in the graph $G(M)$. We use the data structure $F(G(M))$ and Lemma 39 to report $M[i, j]$ for any $0 < i, j \leq n$ as follows.

return $(\lambda - \lambda')$ where λ' is the value returned by $F(G(M))$ after failure of edge (u_i, v_j).

Note that there are $2^{n^2 c \log\ n}$ different instances of the matrix M. It follows from Lemma 39 that for any pair of distinct instances of matrix M, the encoding of the corresponding data structures must differ. Therefore, there is an instance of matrix M for which the data structure $F(G(M))$ must occupy $\Omega(n^2 \log\ n)$ bits of space.

For directed graphs, a lower bound of $\Omega(n^2 \log\ n)$ bits of space can be achieved by orienting each undirected edge of $G(M)$ as follows. For each $i, j \in [n]$, orient edge (u_i, v_j) from vertex u_i to vertex v_j. For each $i \in [n]$, orient edge (s, u_i) from source s to vertex u_i. Similarly, for each $j \in [n]$, orient edge (v_j, t) from vertex v_j to sink t. The rest of the proof is the same as the case of undirected graphs. This completes the proof of Theorem 23.

By exploiting Theorem 37, the following data structure lower bound can also be established along similar lines to the proof of Theorem 23.

▶ **Theorem 40.** *Any data structure that, given a pair of vertices $\{u, v\}$, can report the capacity of an (s,t)-cut C of the least capacity such that $u \in C$ and $v \in \overline{C}$ in a directed weighted graph on n vertices must occupy $\Omega(n^2 \log\ n)$ bits of space.*

References

1. Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows - theory, algorithms and applications*. Prentice Hall, 1993.
2. Yash P. Aneja, R. Chandrasekaran, and Kunhiraman Nair. Maximizing residual flow under an arc destruction. *Networks*, 38(4):194–198, 2001. doi:10.1002/net.10001.
3. Giorgio Ausiello, Paolo Giulio Franciosa, Isabella Lari, and Andrea Ribichini. Max flow vitality in general and st-planar graphs. *Networks*, 74(1):70–78, 2019. doi:10.1002/net.21878.
4. Surender Baswana, Koustav Bhanja, and Abhyuday Pandey. Minimum+1 (s, t)-cuts and dual-edge sensitivity oracle. *ACM Trans. Algorithms*, 19(4), October 2023. doi:10.1145/3623271.
5. Surender Baswana, Keerti Choudhary, and Liam Roditty. An efficient strongly connected components algorithm in the fault tolerant model. *Algorithmica*, 81:967–985, 2019.
6. Surender Baswana and Abhyuday Pandey. Sensitivity oracles for all-pairs mincuts. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 581–609. SIAM, 2022.
7. Michael A Bender, Martin Farach-Colton, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms*, 57(2):75–94, 2005.
8. Davide Bilò, Shiri Chechik, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, Simon Krogmann, and Martin Schirneck. Approximate distance sensitivity oracles in subquadratic space. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1396–1409, 2023.
9. Chung-Kuan Cheng and T. C. Hu. Ancestor tree for arbitrary multi-terminal cut functions. *Ann. Oper. Res.*, 33(3):199–213, 1991. doi:10.1007/BF02115755.
10. Keerti Choudhary. An optimal dual fault tolerant reachability oracle. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2016.
11. Efim A Dinitz, Alexander V Karzanov, and Michael V Lomonosov. On the structure of the system of minimum edge cuts of a graph. *Studies in discrete optimization*, pages 290–306, 1976.
12. Yefim Dinitz and Zeev Nutov. A 2-level cactus model for the system of minimum and minimum+ 1 edge-cuts in a graph and its incremental maintenance. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 509–518, 1995.
13. Yefim Dinitz and Alek Vainshtein. The general structure of edge-connectivity of a vertex subset in a graph and its incremental maintenance. odd case. *SIAM Journal on Computing*, 30(3):753–808, 2000.
14. L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. doi:10.4153/CJM-1956-045-5.
15. Greg N Frederickson and Roberto Solis-Oba. Increasing the weight of minimum spanning trees. *Journal of Algorithms*, 33(2):244–266, 1999.

16 Loukas Georgiadis, Giuseppe F Italiano, and Nikos Parotsidis. Strong connectivity in directed graphs under failures, with applications. *SIAM Journal on Computing*, 49(5):865–926, 2020.

17 R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961. URL: http://www.jstor.org/stable/2098881.

18 Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

19 Fabrizio Grandoni and Virginia Vassilevska Williams. Faster replacement paths and distance sensitivity oracles. *ACM Trans. Algorithms*, 16(1):15:1–15:25, 2020. doi:10.1145/3365835.

20 Frieda Granot and Refael Hassin. Multi-terminal maximum flows in node-capacitated networks. *Discret. Appl. Math.*, 13(2-3):157–163, 1986. doi:10.1016/0166-218X(86)90079-X.

21 Dan Gusfield and Dalit Naor. Efficient algorithms for generalized cut-trees. *Networks*, 21(5):505–520, 1991.

22 Refael Hassin and Asaf Levin. Flow trees for vertex-capacitated networks. *Discrete applied mathematics*, 155(4):572–578, 2007.

23 John Hershberger and Subhash Suri. Erratum to "Vickrey pricing and shortest paths: What is an edge worth?". In *Annual Symposium on Foundation of Computer Science*, volume 43, pages 809–810. IEEE Computer Society Press, 2002.

24 Giuseppe F Italiano, Adam Karczmarz, and Nikos Parotsidis. Planar reachability under single vertex or edge failures. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2739–2758. SIAM, 2021.

25 Michal Katz, Nir A. Katz, Amos Korman, and David Peleg. Labeling schemes for flow and connectivity. *SIAM J. Comput.*, 34(1):23–40, 2004. doi:10.1137/S0097539703433912.

26 Kao-Chêng Lin and Maw-Sheng Chern. The most vital edges in the minimum spanning tree problem. *Inf. Process. Lett.*, 45(1):25–31, 1993. doi:10.1016/0020-0190(93)90247-7.

27 Stephen H Lubore, HD Ratliff, and GT Sicilia. Determining the most vital link in a flow network. *Naval Research Logistics Quarterly*, 18(4):497–502, 1971.

28 Enrico Nardelli, Guido Proietti, and Peter Widmayer. A faster computation of the most vital edge of a shortest path. *Information Processing Letters*, 79(2):81–85, 2001.

29 Cynthia A Phillips. The network inhibition problem. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 776–785, 1993.

30 Jean-Claude Picard and Maurice Queyranne. On the structure of all minimum cuts in a network and applications. *In Rayward-Smith V.J. (eds) Combinatorial Optimization II. Mathematical Programming Studies*, 13(1):8–16, 1980. doi:10.1007/BFb0120902.

31 H Donald Ratliff, G Thomas Sicilia, and SH Lubore. Finding the n most vital links in flow networks. *Management Science*, 21(5):531–539, 1975.

32 Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. *ACM Transactions on Algorithms (TALG)*, 8(4):1–11, 2012.

33 Fu-Shang P Tsen, Ting-Yi Sung, Men-Yang Lin, Lih-Hsing Hsu, and Wendy Myrvold. Finding the most vital edges with respect to the number of spanning trees. *IEEE Transactions on Reliability*, 43(4):600–603, 1994.

34 Virginia Vassilevska Williams, Eyob Woldeghebriel, and Yinzhan Xu. Algorithms and lower bounds for replacement paths under multiple edge failure. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 907–918. IEEE, 2022.

35 Richard D Wollmer. *Some methods for determining the most vital link in a railway network*. Rand Corporation, 1963.

It's Hard to HAC Average Linkage!

MohammadHossein Bateni
Google Research, New York, NY, USA

Laxman Dhulipala
University of Maryland, College Park, MD, USA

Kishen N. Gowda
University of Maryland, College Park, MD, USA

D. Ellis Hershkowitz
Brown University, Providence, RI, USA

Rajesh Jayaram
Google Research, New York, NY, USA

Jakub Łącki
Google Research, New York, NY, USA

Abstract

Average linkage Hierarchical Agglomerative Clustering (HAC) is an extensively studied and applied method for hierarchical clustering. Recent applications to massive datasets have driven significant interest in near-linear-time and efficient parallel algorithms for average linkage HAC.

We provide hardness results that rule out such algorithms. On the sequential side, we establish a runtime lower bound of $n^{3/2-\epsilon}$ on n node graphs for sequential combinatorial algorithms under standard fine-grained complexity assumptions. This essentially matches the best-known running time for average linkage HAC. On the parallel side, we prove that average linkage HAC likely cannot be parallelized even on simple graphs by showing that it is CC-hard on trees of diameter 4. On the possibility side, we demonstrate that average linkage HAC can be efficiently parallelized (i.e., it is in NC) on paths and can be solved in near-linear time when the height of the output cluster hierarchy is small.

2012 ACM Subject Classification Theory of computation → Parallel algorithms; Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases Clustering, Hierarchical Graph Clustering, HAC, Fine-Grained Complexity, Parallel Algorithms, CC

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.18

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2404.14730` [6]

Funding LD and KNG were supported by NSF grant CNS-2317194.

Acknowledgements We thank the anonymous reviewers for their useful comments.

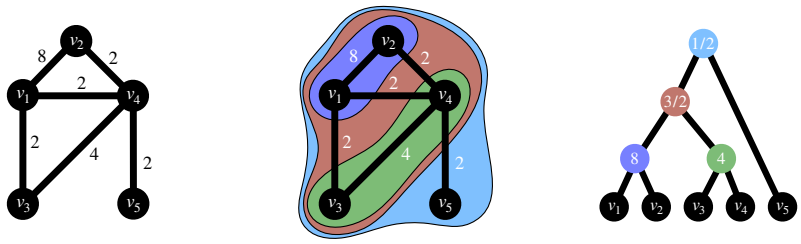
1 Introduction

Hierarchical clustering is a fundamental method for data analysis which organizes data points into a hierarchical structure so that similar points appear closer in the hierarchy. Unlike other common clustering methods, such as k-means, hierarchical clustering does not require the the number of clusters to be fixed ahead of time. This allows it to capture structures that are inherently hierarchical – such as phylogenies [18] and brain structure [11]. One of the most widely used and studied methods for hierarchical clustering is Hierarchical Agglomerative

Clustering (HAC) [22, 24, 37]. HAC produces a hierarchy by first placing each point in its own cluster and then iteratively merging the two *most similar* clusters until all points are aggregated into a single cluster. The similarity of two clusters is given by a *linkage function*. HAC is included in many popular scientific computing libraries such as scikit-learn [35], SciPy [40], ALGLIB [36], Julia, R, MATLAB, Mathematica and many more [33, 34]. This cluster hierarchy is often equivalently understood as a binary tree – a.k.a. dendrogram – whose internal nodes correspond to cluster merges.

The proliferation of massive datasets with billions of points has driven the need for more efficient HAC algorithms that can overcome the inherent $\Theta(n^2)$ complexity required to read all pairwise distances [16, 17, 29]. Finer-grained running time bounds for HAC were recently obtained by assuming that only $m = o(n^2)$ pairs of points have nonzero similarity, and analyzing the running time as a function of both n and m. This is a natural assumption in practice, as in large datasets of billions of datapoints, typically a small fraction of pairs exhibit nonnegligible similarity. In this case, the input to HAC is an edge-weighted graph, where each vertex represents an input point and each edge weight specifies the similarity between its endpoints. This approach is convenient for large-scale applications since (1) very large clustering instances can be compactly represented as sparse weighted graphs and (2) the running time of HAC can be decoupled from the running time of nearest-neighbor search.

A particularly common linkage function for HAC is *average linkage*, which both optimizes reasonable global objectives [31] and exhibits good empirical performance [5, 12, 20, 23, 27, 29, 30, 32, 44]. Here, the similarity of two clusters is the average edge weight between them (non-present edges are treated as having weight 0). In other words, average linkage HAC repeatedly merges the two clusters with the highest average edge weight between them (see Figure 1 for an example).



(a) Input G. **(b)** HAC Output. **(c)** Dendrogram.

Figure 1 An example of average linkage HAC run on an input graph G. Edges labeled with weights. 1a gives G. 1b gives the cluster hierarchy output by HAC. 1c gives the corresponding dendrogram with internal nodes labeled with the weight of their corresponding merge.

A natural algorithmic question then is how quickly can we solve average linkage HAC on n node and m edge graphs? Recent work has provided a partial answer to this basic question in sequential and parallel models of computation. In particular, [15] showed that average linkage HAC can be solved in $\tilde{O}(n\sqrt{m})$ time, thus providing a sub-quadratic time algorithm for sufficiently sparse graphs. A follow-up paper studied average linkage HAC in the parallel setting and showed that the problem is P-complete and so likely does not admit NC algorithms [16]. However, the P-completeness result of [16] holds for worst case graphs whereas typical applications of HAC are on highly structured graphs – namely those which are meant to capture relevant properties of an underlying metric – and so there is still hope for parallelizing average linkage HAC on more structured instances.

In fact, such structured instances of average linkage HAC are known to admit much faster algorithms in the sequential setting: the sequential algorithm of [15] implies that if the input graph is planar (or, more generally, minor-free) average linkage HAC can be solved in time $\tilde{O}(m)$. More generally, if each graph obtained by contracting all clusters at each step of average linkage HAC has $O(1)$ arboricity[1], then it is possible to solve average linkage HAC in time $\tilde{O}(m)$; it follows that average linkage HAC can be solved in sequential time $\tilde{O}(m)$ on trees or planar graphs. In light of these improved sequential results for highly structured graphs, it becomes natural to hope for efficient parallel algorithms on structured graphs such as low arboricity graphs or, even, just trees.

1.1 Our Contributions

In this work, we continue the line of work which studied the computational complexity of different variants of HAC [1, 15, 16, 19, 39] and perform a careful investigation into the complexity of average linkage HAC. In particular, we study HAC on n node and m edge graphs and investigate whether near-linear time algorithms, or more efficient parallel algorithms are possible, namely:

1. **Near-Linear Time Algorithms:** Can we improve over the best known $\tilde{O}(n\sqrt{m})$ upper bound for average linkage HAC and obtain near-linear time sequential algorithms?
2. **NC Algorithms:** are there $\mathsf{polylog}(n)$ depth parallel algorithms for average linkage HAC with $\mathsf{poly}(n)$ work for highly structured instances, e.g., trees, or minor-closed graphs?

We give both new lower bounds which (conditionally) rule out near-linear time and NC algorithms, and provide conditions under which these impossibility results can be bypassed.

First, we demonstrate that near-linear time algorithms are impossible under standard fine-grained complexity assumptions.

▶ **Theorem 1.** *If average linkage HAC can be solved by a combinatorial algorithm in $O(n^{3/2-\epsilon})$ time for any $\epsilon > 0$, then the Combinatorial Boolean Matrix Multiplication (Combinatorial BMM) Conjecture is false.*

Our reduction also implies a second (weaker) conditional lower bound that also holds for non-combinatorial algorithms (e.g., algebraic algorithms) based on the running time of matrix multiplication. In particular, for two $n \times n$ binary matrices, it is well known that matrix multiplication can be solved in time $O(n^\omega)$ where $2 \leq \omega < 2.3716$ [43]. In this setting, we obtain the following result:

▶ **Theorem 2.** *If average linkage HAC can be solved by an algorithm in $O(n^{\omega/2-\epsilon})$ time for some $\epsilon > 0$, then boolean matrix multiplication can be solved in $O(n^{\omega-\epsilon'})$ time for some $\epsilon' > 0$.*

Notably, Theorem 1 shows that the prior running time of $\tilde{O}(n\sqrt{m})$ of [15] is *optimal* up to logarithmic factors under standard fine-grained complexity assumptions, at least for graphs consisting of $O(n)$ many edges. We obtain this conditional lower bound by showing that a carefully constructed instance of HAC can be used to solve the triangle detection problem, which is sub-cubically equivalent to Boolean Matrix Multiplication [42]. We obtain a bound of (essentially) $\Omega(n^{3/2})$ since our reduction incurs a quadratic time and space blowup when transforming an input triangle detection instance to an instance of average linkage HAC.

[1] A graph has arboricity at most α if all of its edges can be covered by at most α trees.

We next turn to the parallel setting. Here, we show that HAC – *even on trees* – is unlikely to admit efficient parallel algorithms. More formally, we show that average linkage HAC on low diameter trees is as hard as any problem in the complexity class Comparator Circuit (CC) [13, 38]. It is believed that CC is incomparable with NC and that CC-hardness is evidence that a problem is not parallelizable [13, 28].

▶ **Theorem 3.** *Average linkage HAC is CC-hard, even on trees of diameter* 4*.*

We note that it is known that CC $\subseteq$ P and so the P-hardness of [16] already suggests the impossibility of efficient parallel algorithms on *general graphs*. However, our result suggests the impossibility of efficient parallel algorithms *even on very simple graphs* (trees of diameter 4). We obtain this result by reducing from the lexicographically first maximal matching (LFM Matching) problem and an intermediate problem which we call Adaptive Minimum, which captures some of what makes HAC intrinsically difficult to parallelize.

On the positive side, we demonstrate that average linkage HAC on path graphs is in NC, under the mild assumption that the aspect ratio is polynomial. While the class of path graphs is restrictive, even on paths average linkage is highly non-trivial and naively running HAC requires resolving chains of $\Omega(n)$ sequential dependencies. For example, consider a path of vertices $(v_1, v_2, \ldots, v_n)$ where the edge $\{v_i, v_{i+1}\}$ has weight $1 + i \cdot \epsilon$ for some small $\epsilon > 0$ and initially each vertex is in its own cluster. Initially, v_n's most similar neighbor is v_{n-1} and so v_n would like to merge with v_{n-1} but v_{n-1}'s most similar neighbor is v_{n-2} and so on. Thus, whether or not v_n gets to merge with v_{n-1} depends on the merge behavior of $\Theta(n)$ other clusters and so it is not at all clear that NC algorithms should be possible for this setting. Nonetheless, we show the following.

▶ **Theorem 4.** *Average linkage HAC on paths is in NC. In particular, there is an algorithm for average linkage HAC that runs in $O(\log^2 n \log\log n)$ depth with $O(n \log n \log\log n)$ work.*

The above algorithm leverages the fact that in average linkage HAC the maximum edge similarity monotonically decreases. In particular, it works in $O(\log n)$ phases where each phase consists of merges of equal similarity up to constants. The goal then becomes to efficiently perform merges until every edge is no longer within a constant of the starting maximum similarity of the phase. The starting point of the algorithm is to observe that $\Omega(n)$ sequential dependencies of clusters of equal size can be resolved efficiently in parallel in a phase by noting that in this phase only the odd-indexed edges merge in the chain. Thus, each edge can decide if it is odd-indexed in parallel by, e.g., using prefix-sum, which is well known to be solvable in linear work in NC.

For chains with clusters of general weights, we decompose dependency chains into short ($(O(\log n)$-length) subchains where resolving dependencies within the subchain must be done sequentially but in the current phase each subchain's merge behavior only depends on whether or not its closest neighboring subchains merges into it or not. Thus, each subchain can compute its merge behavior for these two cases and then, similar to the equal weights setting, we propagate merge behavior across subchains efficiently in parallel.

To complement our sequential lower bound with a positive result, we demonstrate that it is possible to achieve near-linear running time, provided the dendrogram has low height. Thus, if the output dendrogram is a relatively balanced tree, then near-linear time algorithms are possible.

▶ **Theorem 5.** *There is an implementation of the nearest-neighbor chain algorithm for average linkage HAC that runs in $O(m \cdot h \log n)$ time where h is the height of the output dendrogram.*

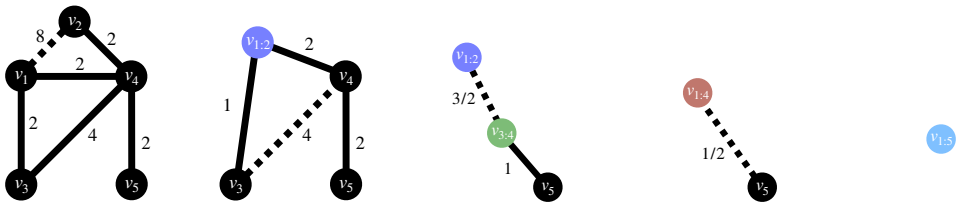
The above result is in fact obtained by a relatively simple (but to the best of our knowledge, new) analysis of *existing* classic HAC algorithms. In particular, we show that the nearest-neighbor chain [7, 21] and heap-based algorithms [27] for HAC, which were developed over 40 years ago achieve this bound. Due to space constraints, we prove Theorem 5 in the full version [6].

2 Preliminaries

The input to the HAC algorithm is an undirected weighted graph $G = (V, E, w)$, where $w : V \times V \to \mathbb{R}_+ \cup \{0\}$ is a function assigning nonnegative weights to the edges. For convenience we assume $w(x, y) = 0$ when $xy \notin E$. The vanilla version of average linkage HAC is given as Algorithm 1. It starts by putting each vertex in a cluster of size 1 and then repeats the following step. While there is a pair of clusters of positive similarity, find two most similar clusters and merge them together, that is, replace them by their union. The similarity between two clusters is the total edge weight between them divided by the product of the cluster sizes. We refer to this version as the *static graph* version, since the graph is not changed throughout the run of the algorithm.

Throughout the paper we usually work with a different (equivalent) way of presenting the same algorithm which is given as Algorithm 2. In this version we maintain a graph G whose vertices are clusters. The *size* of the vertex is the size of the cluster it represents. The *normalized* weight of an edge xy in G is $w(x, y)$ divided by the product of the sizes of x and y.

Whenever two clusters merge, their corresponding vertices are merged into one, i.e., the edge between them is contracted and the size of the new vertex is the sum of the sizes of the vertices that merged. In the following we sometimes say that a vertex x merges into vertex y. In this case we simply assume that the name of the resulting vertex is y and the size of y is increased by the size of x. See Figure 2.



(a) Input G. **(b)** 1 merge. **(c)** 2 merges. **(d)** 3 merges. **(e)** 4 merges.

Figure 2 An example of average linkage HAC run on an input graph G where we imagine we contract merged clusters. Intermediate vertices labeled with the vertices of G their corresponding cluster contains. Edges labeled with their weight and next merged edge is dashed.

The output of HAC is a *dendrogram* – a rooted binary tree representing the cluster merges performed by the algorithm. Every node of the dendrogram is a cluster built by the algorithm. There are exactly $|V|$ leaves corresponding to the single-element clusters that are formed in the beginning of the algorithm. Whenever two clusters C_1 and C_2 are merged, we add to the dendrogram a new node $C_1 \cup C_2$ whose children are C_1 and C_2. See Figure 1c for the dendrogram of Figure 2.

We use the classic multithreaded model [4, 9, 10] (formally, the MP-RAM [9]) to analyze the parallel algorithms. We assume a set of threads that share the memory. Each thread acts like a sequential RAM plus a `fork` instruction that forks two new child threads. When a

Algorithm 1 Average linkage HAC – static graph version.

Input: $G = (V, E, w)$
Function `Similarity`(C_1, C_2, w)**:**
  return $\sum_{x \in C_1, y \in C_2} w(x, y)/(|C_1| \cdot |C_2|)$
Function `HAC`(G)**:**
  $\mathcal{C} \leftarrow$ clustering where each vertex of G is in a separate cluster
  while $\exists_{C_1, C_2 \in \mathcal{C}}$ *s.t.* $C_1 \neq C_2$ *and* `Similarity`$(C_1, C_2, w) > 0$ **do**
    $(C_1, C_2) = \arg\max_{(C_1, C_2) \in \mathcal{C} \times \mathcal{C}}$ `Similarity`(C_1, C_2, w)
    $\mathcal{C} := (\mathcal{C} \setminus \{C_1, C_2\}) \cup \{C_1 \cup C_2\}$.

Algorithm 2 Average linkage HAC – graph contraction version.

Input: $G = (V, E, w)$
Function `Similarity`(x, y, w, S)**:**
  return $w(x, y)/(S(x) \cdot S(y))$
Function `HAC`(G)**:**
  $S :=$ a function mapping each element of V to 1
  while $\exists_{xy \in E}$ *s.t.* `Similarity`$(x, y, w, S) > 0$ **do**
    $xy = \arg\max_{xy \in E}$ `Similarity`(x, y, w, S)
    Contract x with y in G creating a vertex z. The parallel edges that are created are merged into a single edge whose weight is the sum of the merged edge weights. Any resulting self-loops are removed.
    Set $S(z) := S(x) + S(y)$

thread performs a fork, the two child threads can both start by running their next instructions, and the original thread is suspended until both children terminate. A computation starts with a single root thread and finishes when that root thread finishes. A parallel for-loop can be viewed as executing `fork`s for a logarithmic number of levels. A computation can thus be viewed as a DAG (directed acyclic graph). We say the *work* is the total number of operations in this DAG and *span (depth)* is equal to the longest path in the DAG. We note that computations in this model can be cross-simulated in standard variants of the PRAM model in the same work (asymptotically), and losing at most a single logarithmic factor in the depth [9].

3 An $\Omega(n^{3/2-\epsilon})$ Conditional Lower Bound for Average Linkage HAC

In this section, we show an $\Omega(n^{3/2-\epsilon})$ conditional lower bound on the time required to solve average linkage HAC on general weighted graphs. Specifically, we show this lower bound assuming the Combinatorial Boolean Matrix Multiplication (BMM) conjecture, a central conjecture in fine-grained complexity about the time required to multiply two $n \times n$ boolean matrices [2, 42].

▶ **Conjecture 6** (Combinatorial BMM). *Combinatorial algorithms cannot solve Boolean Matrix Multiplication in time $O(n^{3-\epsilon})$ for $\epsilon > 0$.*

We refer to [2] for an in-depth discussion of the somewhat informal notion of "combinatorial" algorithms and more on Conjecture 6 and its history.

In this work we will make use of an equivalent characterization of the BMM conjecture due to [42]. Specifically, [42] shows that the BMM problem is sub-cubically equivalent to the Triangle Detection problem: the problem of deciding whether or not an input graph G contains a triangle (i.e., cycle of 3 vertices). The following summarizes this result.

▶ **Theorem 7** (Theorem 1.3 of [42]). *Combinatorial algorithms cannot solve Triangle Detection in time $O(n^{3-\epsilon})$ for $\epsilon > 0$ unless Conjecture 6 is false.*

Thus, we give a reduction from Triangle Detection to average linkage HAC. Our reduction will quadratically increase the number of vertices of the input Triangle Detection instance, and therefore give an $\Omega(n^{3/2-\epsilon})$ lower bound for average linkage HAC. In the rest of this section, we show the following quadratic-blowup reduction from Triangle Detection to average linkage HAC.

▶ **Theorem 8.** *Given a Triangle Detection instance on graph G with t vertices and m edges, there is a reduction that runs in $O(t^2)$ time and constructs an instance of average linkage HAC on graph G' with $t + t^2$ vertices and $t^2 + m$ edges. Furthermore, given the sequence of merges performed by average linkage HAC on G', we can solve Triangle Detection on G in time $O(t^2)$.*

As a corollary of this reduction and Theorem 7, we obtain the following conditional lower-bound on the running time of HAC.

▶ **Theorem 1.** *If average linkage HAC can be solved by a combinatorial algorithm in $O(n^{3/2-\epsilon})$ time for any $\epsilon > 0$, then the Combinatorial Boolean Matrix Multiplication (Combinatorial BMM) Conjecture is false.*

As a second corollary, we obtain a conditional lower-bound in terms of the optimal running time of matrix multiplication for two $n \times n$ binary matrices. Matrix multiplication can be solved in time $O(n^\omega)$ where $2 \leq \omega < 2.3716$ [43]. An extensive line of research on matrix multiplication over the past thirty years has only improved ω from 2.376 to 2.3716, with the current state-of-the-art being due to a very recent result of Williams et al. [43] (for a subset of the historical advances in this area see, e.g., [3, 14, 26, 41]). The fastest known algorithm for triangle detection works by simply reducing the problem to matrix multiplication and therefore runs in $O(n^\omega)$ time. Surprisingly, despite triangle detection only returning a single bit (whether a triangle exists or not in G), the problem can be used to give a sub-cubic reduction for boolean matrix multiplication (where the output is n^2 bits). In particular, an algorithm for triangle detection running in time $O(n^{3-\delta})$ for some $\delta > 0$ yields an algorithm for matrix multiplication in time $O(n^{3-\delta/3})$ [42]. Using this fact, we can derive a conditional lower bound based on the value of ω.

▶ **Theorem 2.** *If average linkage HAC can be solved by an algorithm in $O(n^{\omega/2-\epsilon})$ time for some $\epsilon > 0$, then boolean matrix multiplication can be solved in $O(n^{\omega-\epsilon'})$ time for some $\epsilon' > 0$.*

An interesting open question is whether there are faster non-combinatorial algorithms that can leverage fast matrix multiplication or Strassen-like techniques and improve over the $\Omega(n^{3/2-\epsilon})$ barrier for combinatorial algorithms for average linkage HAC.

3.1 Reduction

We now prove Theorem 8 by giving a quadratic-time reduction from triangle detection to average linkage HAC. The reduction is loosely inspired by a recent lower-bound result for multidimensional range queries [25]. The input to the reduction is an unweighted graph G

on t vertices with m edges; the problem is to detect whether G has a triangle. To do this, we will construct a HAC instance on an edge-weighted graph G' with $t + t^2$ vertices and $t^2 + m$ edges. We will show that the specific way in which an exact HAC algorithm merges the edges in this instance reveals whether or not G has a triangle.

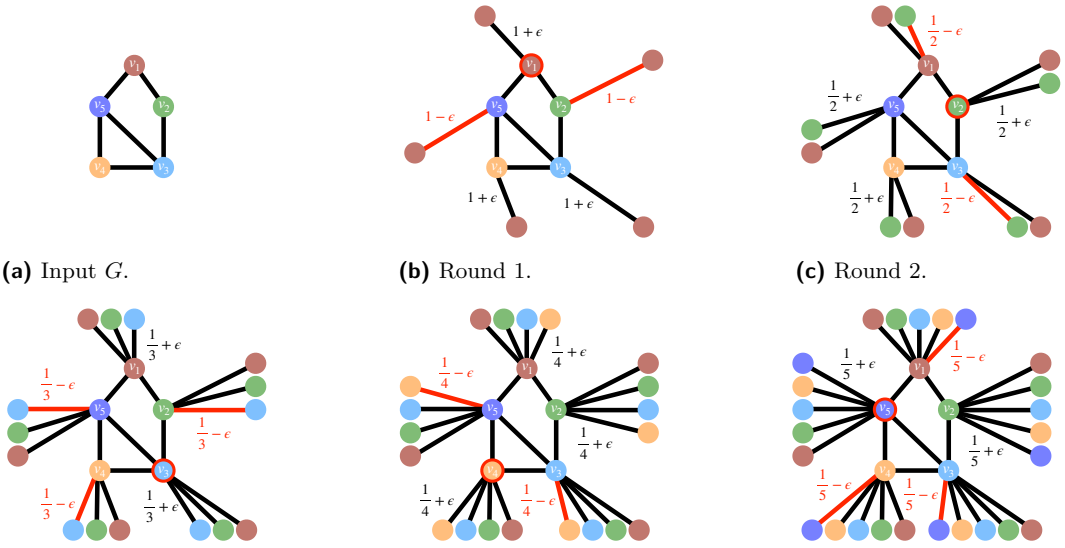
Constructing G'

Let $N_G(v)$ denote the neighbors of a vertex $v \in G$ (note that $v \notin N_G(v)$). We define G' as follows. We start by adding all vertices and edges from G, that is the t vertices $v_1, \ldots, v_t$ from G, including all of their incident edges $N_G(v_i)$. We call these the *core* vertices. The initail weight of the edges between any two core vertices is set to 1.

In addition to the core vertices, we add an additional t^2 *leaf* vertices that we connect to the core vertices with specific edge weights. We add the t^2 leaf vertices over a sequence of t *rounds* where the i-th round connects one new leaf vertex to every core vertex. The weights to the newly added leaves depend on the neighbors of the node v_i in the original graph G, and are set as follows:

(1) A core vertex v_j is connected to its new leaf with an edge of weight $(1/i) - \epsilon$ if $v_j \in N_G(v_i)$.
(2) A core vertex v_j is connected to its new leaf with an edge of weight $(1/i) + \epsilon$ if $v_j \notin N_G(v_i)$.

See Figure 3 for an illustration of our reduction.



(a) Input G. **(b)** Round 1. **(c)** Round 2.

(d) Round 3. **(e)** Round 4. **(f)** Round 5.

Figure 3 Our triangle detection reduction where we compute G' from G by adding $t = 5$ nodes over t rounds. 3f gives G'. Each node labeled according to its round and corresponding vertex in G. Edges labelled with their weight in the round they are added (edges of G have weight 1). For the ith round we highlight in red v_i and the edges added with weight $1/i - \epsilon$.

Running HAC on G'

Having defined G', let us consider the merges that the exact HAC algorithm will make on this instance. In this section, for succinctness we use *weight* to refer to the normalized (i.e., average linkage) weight. HAC will begin by merging the maximum weight edges. The maximum weight initially depends on the structure of $N_G(v_1)$. First, all core vertices that

are not in $N_G(v_1)$ will merge with their round 1 leaves (since these edges have weight $1+\epsilon$) increasing their cluster size to 2. This leaves all core vertices in $N_G(v_1)$. Any edge in G' between two such core vertices will have weight 1, and will be merged next. Crucially, any merge in this round with a weight of 1 indicates a triangle incident on v_1 since the two core vertex endpoints of the edge must be contained in $N_G(v_1)$, hence connected by edges to v_1. If no edges of weight 1 merge, the remaining leaves that we added to core vertices in $N_G(v_1)$ merge into their neighboring core vertex.

Assuming we did not merge any weight 1 edges in the first round, at this point the cluster size of each core vertex is 2, and so the weight of any edge originally in G (between two core vertices) will be $1/4$. The edge weights to leaves in round 2 will be $((1/2)\pm\epsilon)/2 = 1/4 \pm (\epsilon/2)$, which is larger than $1/4$ for edges of Type 2. Therefore, the same argument for how edges merge in round 1 can be inductively applied to the next round. The edge weights in round i for edges between core vertices will be $1/i^2$, and by the same argument as before, the Type 2 (Type 1) edges will be larger (smaller) by ϵ/i. As a result of how an exact average linkage HAC will merge the edges of G', we obtain the following lemma:

▶ **Lemma 9.** *Consider the sequence of merges performed by the HAC algorithm on G'. If the merge sequence consists of t^2 merges, which first merge all leaf vertices, and only then makes merges between core vertices, then G does not contain any triangles. If the merge sequence merges any edge between two core vertices in the first t^2 merges, then G contains a triangle.*

Completing the Reduction

We will now complete the proof of Theorem 8. Suppose we are given an instance of Triangle Detection on n vertices. Conjecture 6 implies that this instance cannot be solved by combinatorial algorithms in $O(n^{3-\epsilon})$ time for any $\epsilon > 0$.

Let the time complexity of HAC on a graph G with n vertices and m edges be $T_{\mathsf{HAC}}(n, m)$. Suppose HAC can be solved combinatorially in $O(n^{3/2-\epsilon})$ time. Given a Triangle Detection instance on n vertices we create a graph G' with $O(n^2)$ vertices and $O(n^2)$ edges, and run HAC on G'. The running time of the reduction is $O(n^2)$, and the running time of HAC on G' is $O((n^{2\cdot(3/2-\epsilon)}) = O(n^{3-2\epsilon})$, which will falsify Conjecture 6 by Theorem 7. Thus, conditional on Conjecture 6, there is no algorithm for HAC running in time $T_{\mathsf{HAC}}(n, m) = O(n^{3/2-\epsilon})$ for any constant $\epsilon > 0$, completing the proof of Theorems 8 and 1. The same argument, under the assumption that triangle detection cannot be solved in $O(n^{\omega-\epsilon})$ time for any constant $\epsilon > 0$ implies Theorem 2.

4 Average Linkage HAC is Hard to Parallelize Even on Trees

In this section we prove that average linkage HAC is likely hard to parallelize by showing it is CC-hard even on low depth trees. We begin with some preliminaries. The formal definition of CC-hardness we will use is as follows.

▶ **Definition 10** (CC-Hard). *A problem is CC-hard if all problems of CC are logspace-reducible to it.*

For our purposes we will not need to define the class CC. Rather, we only need the above definition of CC-hardness and a single CC-hard problem, LFM Matching. Recall that a matching of a graph $G = (V, E)$ is a subset of edges $M \subseteq E$ if each vertex is incident to at most one edge of M. A matching is said to be maximal if each $e = \{u, v\} \notin M$ satisfies the property that either u or v is incident to an edge of M. The greedy algorithm for maximal matching initializes M as $\emptyset$ and then simply iterates over the edges of E in some order and adds the current edge e to M if the result of doing so is a matching.

▶ **Problem 1** (LFM Matching). *An instance of lexicographically first maximal matching (LFM Matching) consists of a bipartite graph $G = (V = L \sqcup R, E)$ with vertices ordered as $L = (l_0, l_1, \ldots, l_{n-1})$ and $R = (r_0, r_1, \ldots, r_{n-1})$. The lexicographically first maximal matching is the matching obtained by running the greedy algorithm for maximal matching on edges ordered first by their endpoint in L and then by their endpoint in R. That is, in this ordering $e = \{l_i, r_j\}$ precedes $e' = \{l_{i'}, r_{j'}\}$ iff (1) $i < i'$ or (2) $i = i'$ and $j < j'$. Our goal is to decide if a designated input edge is in the LFM Matching.*

The following summarizes known hardness of LFM Matching.

▶ **Theorem 11** ([28, 38]). *LFM Matching is CC-hard.*

Next, we introduce the search variant of the HAC problem whose CC-hardness we will prove.

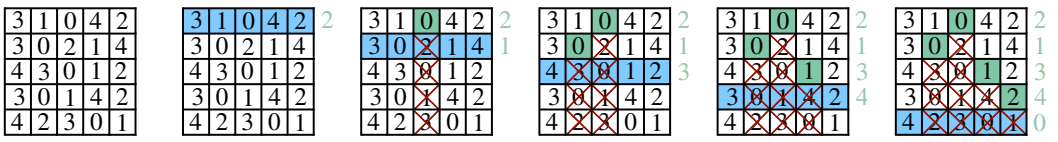
▶ **Problem 2** (Average Linkage HAC). *An instance of Average Linkage HAC consists an undirected graph $G = (V, E)$, along with edge weights $w : E \to \mathbb{R}_{\geq 0}$. Consider the sequence $(C_1, C_1'), (C_2, C_2'), \ldots$ of cluster merges produced by the procedure* `HAC(G)` *from Algorithm 1. Given any pair of vertices $u, v \in G$, the goal of the Average Linkage HAC problem is to output the index i such that u, v first merge together at step i, namely $u \in C_i$ and $v \in C_i'$ (or $u \in C_i'$ and $v \in C_i$).*

To prove the hardness of Average Linkage HAC, we will first prove the hardness of an intermediate problem, called Adaptive Minimum. The construction of the Adaptive Minimum problem will be more amenable to our reductions, and therefore simplify the following exposition. See Figure 4 for an illustration of Adaptive Minimum.

▶ **Problem 3** (Adaptive Minimum). *An instance of Adaptive Minimum consists of a (0-based indexed) $n \times n$ matrix A where each row contains a permutation of $\{0, \ldots, n-1\}$ and some index $x \in [0, n)$. The goal is to simulate the following algorithm. Start with $I = \{0, \ldots, n-1\}$ and execute the following steps for $i = 0, \ldots, x$:*

1. *Let $k_i = \arg\min_{j \in I} A[i, j]$.*
2. *Set $I := I \setminus \{k_i\}$.*

Our goal is to compute k_x.



(a) Input A. **(b)** Step 1. **(c)** Step 2. **(d)** Step 3. **(e)** Step 4. **(f)** Step 5.

Figure 4 Adaptive Minimum on matrix A. The row considered in each step is shown in blue. k_i for the i-th row written to the right of A in green with witnessing entry of A also in green. Indices removed from I in relevant rows crossed out in red.

Observe that both problems 2 and 3 are defined as having an algorithm output an index $i \in \{1, 2, \ldots\}$. Thus, these can be considered search problems instead of decision problems. We choose to work with the search versions of these problems for simplicity of our reductions, however, our reduction naturally extends to the decision variants (e.g., where the algorithm is given $u, v \in V$ and an index i and asked if u, v merge on step i).

We first prove the CC-hardness of this intermediate problem. See Figure 5 for an illustration of our reduction from LFM Matching to Adaptive Minimum.

▶ **Lemma 12.** *Adaptive Minimum is CC-hard.*

Proof. By Theorem 11 and the definition of CC-hardness (Definition 10), it suffices to argue that LFM Matching (Problem 1) is logspace reducible to Adaptive Minimum (Problem 3). We begin by describing our reduction and then observe that it only requires logarithmic space. The basic idea of the reduction is to associate with each vertex on the left side of our instance of LFM Matching a row of the matrix of Adaptive Minimum and each vertex on the right side of LFM Matching a column of the matrix of Adaptive Minimum.

More formally, consider an instance of LFM Matching on graph $G = (V = L \sqcup R, E)$ where our goal is to decide if a given edge e is in the LFM matching. We consider the following instance of Adaptive Minimum to solve this on a $2n \times 2n$ size matrix A. We will refer to an index i as a *dummy index* if $i \geq n - 1$.

Consider a vertex $l_i \in L$ connected to vertices $R_i \subseteq R$ in G, for some $i \leq n - 1$. We will construct the ith row of A to correspond to a permutation π_i that first gives the indices of all neighbors of l_i in R sorted according to the ordering of R then gives all dummy indices then gives the indices of non-neighbors of l_i in R. Specifically, the first $|R_i|$ indices of π_i will be the indices of R_i (sorted by their order in R), the next n indices will be dummy indices $n, n+1, \ldots 2n-1$ and the remaining $n - |R_i|$ indices will be the indices of vertices in $R \setminus R_i$ (sorted, say, by their order in R). For $i \geq n$ we can construct our permutation arbitrarily. Lastly, let x (the index for which we would like to compute k_x in our instance of Adaptive Minimum) be the index of the endpoint of e in L. Once Adaptive Minimum computes k_x, we verify whether or not it corresponds to the endpoint of e in R to determine the final output of the LFM Matching instance. Again, see Figure 5.

We now argue correctness of the reduction. A straightforward proof by induction on i demonstrates that at the beginning of the ith round of the Adaptive Minimum algorithm we have that I consists of at least $n - i$ dummy indices and $j < n$ is not in I only if r_j is in the LFM Matching and is matched to some $l_{i'}$ for $i' < i$. It follows that $e = (l_i, r_j)$ is in the LFM Matching iff $k_i = j$, showing correctness of our reduction.

It remains to show that the above reduction can be done with logspace. In order to do so, we must argue that $A[i, j]$ can be computed with logspace for every i and j. Doing so is trivial if i is a dummy index, so consider $i < n$.

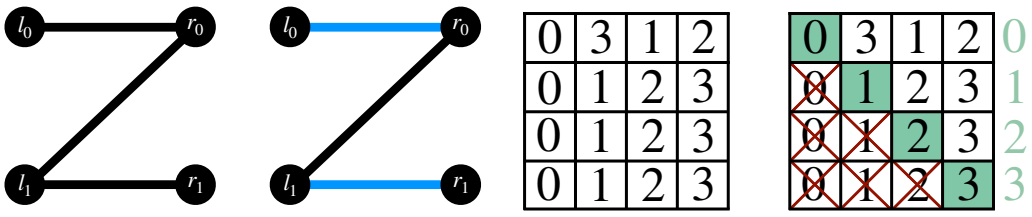
- If $j \leq |R_i|$ then $A[i, j]$ is just the index of the jth vertex of R_i (i.e., neighbor of l_i) in the ordering given by R.
- If $j \in [|R_i|, |R_i| + n]$ then $A[i, j]$ just is $j - |R_j| + n$.
- If $j > |R_i| + n$ then $A[i, j]$ is the index of the $(j - |R_i| - n)$th vertex in $R \setminus R_i$ (i.e., non-neighbors of l_i) when vertices of $R \setminus R_i$ are sorted according to the ordering on R.

All three of the above quantities can easily be computed in logspace. ◀

Concluding, we use the CC-hardness of Adaptive Minimum to prove the CC-hardness of Average Linkage HAC.

▶ **Theorem 3.** *Average linkage HAC is CC-hard, even on trees of diameter* 4.

Proof. Our reduction shows how to reduce an instance of Adaptive Minimum (Problem 3) of size n to an instance of average linkage HAC on a tree. We build a rooted tree, in which each root-to-leaf path has length 2 (i.e., the tree has depth 2). We call the neighbors of the root *internal* nodes. Observe that each node is either the root, an internal node or a leaf. The fact that the tree is rooted is only for the convenience of the description. In the construction, we will begin by assigning each node an initial size (see the definition of *size* in Section 2) which is possibly larger than 1 (but at most $\mathsf{poly}(n)$). We will later show how to remove these variable sizes, and reduce to the case where all nodes have initial size 1 (as in the original definition of HAC).



(a) LFM Matching. **(b)** Solution. **(c)** Adaptive Minimum. **(d)** Solution.

Figure 5 Reduction from LFM Matching to Adaptive Minimum. 5a gives the LFM Matching instance and 5b its solution. 5c gives the Adaptive Minimum instance from the reduction and 5d its solution.

The basic idea of our construction is as follows. Our HAC instance will consist of a rooted tree where each child of the root corresponds to a column of A in our Adaptive Minimum instance. HAC merges will then happen in phases where each phase corresponds to a row of A. In a given phase, exactly one internal node will merge with the root which will correspond to this internal node's column being minimum for the corresponding row of A. In order to guarantee this, each child of the root will have its own carefully selected children such that merging with these children guarantees the desired behavior in every phase.

More formally, the tree is constructed as follows. The root r of the tree has initial size n^8. It has n children, each of initial size n^4 – we denote them by $v_0, \ldots, v_{n-1}$. The root is connected to its children using edges of weight 1, i.e., $w(r, v_i) = 1$ for all $i = 0, 1, \ldots, n-1$ (thus, the normalized weight of the edges $\{rv_i\}_{i=0}^{n-1}$ are each $\frac{1}{n^{12}}$ at the start). Each internal node has $n(n+1)$ children (leaf nodes) grouped into n groups of $n+1$ leaves each. We write $C_{i,j} = \{v_{i,j,0}, v_{i,j,1}, \ldots, v_{i,j,n}\}$ to denote the j-th group of children of the i-th internal vertex. All the leaves $v_{i,j,k}$ have initial size 1. Thus, the vertices in the full graph in our construction consists of the root r, internal nodes $\{v_0, v_1, \ldots, v_{n-1}\}$ and leaves $\cup_{i=0}^{n-1} \cup_{j=0}^{n-1} \cup_{k=0}^{n} \{v_{i,j,k}\}$

Let $r_i = n^8 + i \cdot n^4$ (for $0 \le i < n$). For each pair of an internal node and each of its groups there are only two distinct edge weights for edges between the internal node and the leaves in the group. Specifically, for an internal node v_j and group $C_{j,i}$ of its children we have $A[i,j]+1$ edges of weight $\frac{1}{r_i - 1}$ and $n - A[i,j]$ edges of weight $\frac{1}{r_i + i \cdot n^3}$. Specifically, we set

$$w(v_j, v_{j,i,k}) = \begin{cases} \frac{1}{r_i - 1} & \text{if } 0 \le k \le A[i,j] \\ \frac{1}{r_i + i \cdot n^3} & \text{if } A[i,j] < k \le n \end{cases}$$

We call the two weights *high-weight* and *low-weight* edges, respectively. Note that their normalized weights are $\frac{1}{n^4(r_i - 1)}$ and $\frac{1}{n^4(r_i + i \cdot n^3)}$ respectively. Observe that the setting of *high-weight* and *low-weight* edges is independent of the internal node v_j, although the number of high versus low-weight edges depends on $A[i,j]$. Moreover, note that even low-weight edges of any group $C_{j,i}$ have higher weights than high-weight edges of group $C_{j,i+1}$:

$$\begin{aligned} \frac{1}{n^4(r_i + i \cdot n^3)} &> \frac{1}{n^4(r_{i+1} + (i+1) \cdot n^3)} \iff \\ \frac{1}{n^8 + i \cdot n^4 + i \cdot n^3} &> \frac{1}{n^8 + (i+1)n^4 - 1} \iff \\ i \cdot n^3 &< n^4 - 1. \end{aligned}$$

which follows from the fact that $i \leq n-1$. We now demonstrate that the average linkage HAC on this instance works in n phases numbered from 0 to $n-1$, where in each phase i,

- $n-1$ internal nodes contract all of their incident group i edges, and
- one internal node contracts all of its high-weight edges to group i, after which it merges with the root.

Because of the internal node merging with the root, the root has incident leaves, but they are connected with edges of (normalized) edge weights $\leq \frac{1}{n^{15}}$ and so they will be irrelevant until all phases have been completed. We show that if we denote by k_i the index of the internal node which merges with the root in phase i, the sequence $k_0, \ldots, k_{n-1}$ is a correct solution to the Adaptive Minimum problem.

In order to analyze the algorithm, we prove the following claim. For convenience, let us define $w_i = n^4 + i \cdot (n+1)$.

▷ Claim 13. In the beginning of phase i, the graph is as follows:

1. The size of the root node is $r_i + i^2 \cdot \Delta_i$ for some $\Delta_i \in [0, n+1]$.
2. Exactly i internal nodes have been merged with the root, and the corresponding values $k_0, \ldots, k_{i-1}$ have been computed correctly.
3. The size of each of the $n-i$ remaining internal nodes is w_i.
4. For all remaining internal nodes, *all* leaves in groups $0, \ldots, i-1$ have been merged into their parents, and no leaves in groups $i, \ldots, n-1$ have been merged.
5. The root may have incident leaves (resulting from internal nodes contracting into it) connected to the root with edges of normalized weights $\leq \frac{1}{n^{15}}$.

Proof. We prove the above claim using induction on i. The base case of $i = 0$ follows directly from how the tree is constructed.

We now simulate a single phase. The edges between the root and the internal nodes have (normalized) weights $\frac{1}{w_i(r_i + i^2 \cdot \Delta_i)} > \frac{1}{n^{15}}$. Hence, the additional leaf nodes incident to the root (see Item 5 of the Claim) are irrelevant. Thus, the highest weight edge in the graph is surely incident to one of the internal nodes. Observe that the relative order of edge weights between an internal node v and its children does not change as the leaves are merged into v. Therefore, given that groups $0, \ldots, i-1$ do not exist anymore, among edges between the internal nodes and leaves, the edges of group i have the highest weights. In the beginning of a phase the high-weight edges in that group have normalized weights $\frac{1}{w_i(r_i - 1)}$ and the low-weight edges have weight $\frac{1}{w_i(r_i + i \cdot n^2)}$.

Hence, we have that if we sort the edges by their normalized weights, the top 3 classes of edges are, starting from the highest weight:

1. High-weight edges between internal nodes and leaves of group i.
2. Edges between the root and the internal nodes.
3. Low-weight edges between internal nodes and leaves of group i.

We will show that the phase consists of the following sub-phases.

1. First, there is some number of subphases, where each of $n-i$ internal nodes contract one incident high-weight edge.
2. Then, there is exactly one subphase, where $n-i-1$ nodes contract an incident high-weight edge and one internal node merges with the root.
3. Then, the remaining $n-i-1$ internal nodes merge with all of their group i leaves (we do not analyze the order in this subphase, as it is irrelevant).

Assume that each internal node has at least one high-weight edge in group i. Then, the algorithm will execute a Type 1 subphase: the first $n-i$ steps of the algorithm would merge exactly one high-weight edge incident to each internal node. Note that when an edge incident to an internal node v merges, the weight of v increases, and so the incident edge weights decrease. This guarantees that in the considered $n-i$ steps exactly one merge per internal node happens.

Type 1 subphases of $n-i$ steps continue as long as each each internal node has at least one high-weight group i edge in the beginning of the subphase. Each Type 1 subphase also causes the weight of each internal node to increase by 1. Clearly, since nodes are being merged into internal nodes, the ordering of edge weights incident to any internal node does not change.

At some point, in the beginning of a subphase there is an internal node that does not have any incident high-weight edge in group i. Assume that this happened after p Type 1 subphases have completed. Thus, the size of each internal node is $w_i + p$. Since by the construction each internal node had a different number of high-weight edges in group i, there is exactly one node v with no high-weight incident edges and that node merges with the root. This is when Type 2 subphase happens. First, $n-i-1$ internal nodes contract with a high-weight incident group i edge. At this point the edge weights are as follows. The weight of an edge between v and the root is $\frac{1}{(w_i+p)(r_i+i^2\cdot\Delta_i)}$ and the weight of a high-weight edge in group i is $\frac{1}{(w_i+p+1)(r_i-1)}$. We have that the former is larger since

$$\begin{aligned}
(w_i+p)(r_i+i^2\cdot\Delta_i) &< (w_i+p+1)(r_i-1) \iff \\
(w_i+p)\cdot i^2\cdot\Delta_i &< r_i - 1 \iff \\
(n^4+i\cdot(n+1)+p)\cdot i^2\cdot\Delta_i &< n^8+i\cdot n^4-1 \Leftarrow \\
(n^4+n\cdot(n+1)+n)\cdot n\cdot n^2 &< n^8+i\cdot n^3-1.
\end{aligned}$$

Thus, the internal node with no incident high-weight edges in group i merges with the root. Observe that this is exactly the internal node which had the lowest number of high-weight edges in group i among all remaining internal nodes. This immediately implies that k_i is computed correctly, proving Item 2.

We now show that in the remaining part of the phase the $n-i-1$ remaining internal nodes contract their incident group i edges. First, observe that the new size of the root node is

$$r_i + w_i + p + i^2\cdot\Delta_i = \left(n^8 + i\cdot n^4 + n^4\right) + \left(i\cdot(n+1) + i^2\cdot\Delta_i + p\right) = r_{i+1} + (i+1)^2\Delta_{i+1}$$

for some $\Delta_{i+1} \in [0, n+1]$. Note that we use the fact that both Δ_i and p are upper bounded by $n+1$, which implies $i\cdot(n+1) + i^2\cdot\Delta_i + p \le (i+1)^2(n+1)$. This proves Item 1. Thus for an internal node of weight w, the weight of its edge to the root is

$$\frac{1}{w\cdot(r_{i+1}+(i+1)^2\Delta_{i+1})} \le \frac{1}{w\cdot r_{i+1}} = \frac{1}{w\cdot(n^8+(i+1)\cdot n^4)}.$$

On the other hand, its low-weight edges to group i leaves have weight

$$\frac{1}{w(n^8+i(n^4+n^2))}.$$

As a result, in the remaining part of the current phase all internal nodes will contract all their incident group i edges. This implies Item 4. Thus, the size of each internal node within the phase increases to $w_i + (n+1) = n^4 + i\cdot(n+1) + n + 1 = n^4 + (i+1)(n+1)$, as required. This proves Item 3 and completes the proof. ◁

Finally, we now claim that, given an instance of Adaptive Minimum with input index $x \in [0, n)$ and an algorithm which can compute the solutions to Problem 2, we can compute the solution k_x to Problem 3 in logspace. To see this, note that it suffices to determine the value k_x as defined above given an algorithm for Average Linkage HAC. To see this, note that for any internal node i, we can query the Average Linkage HAC algorithm to determine which time step t_i it merged with the root. This does not directly tell us which phase i merged with the root, but for a given i we can determine if it merged in phase x by comparing t_i with t_j for all $j \in \{0, 1, \ldots, n-1\} \setminus \{i\}$, and checking if there are exactly $x - 1$ values of t_j smaller than t_i. This clearly can be verified in log-space. Repeating for all $i \in \{0, 1, \ldots, n-1\}$ allows us to correctly determine the identity of the internal node that merged with the root in phase x, and therefore the value of k_x, in logspace as required.

To complete the proof of the Lemma it remains to show how to drop the assumption on the node sizes being initially not all equal to 1. In order to obtain a node of size w it suffices to create a node of weight 1 and initially connect it to $w - 1$ auxiliary nodes using very high weight edges. This will force the algorithm to merge all these auxiliary nodes and increase the size of that node to w. Since the auxiliary leaves are connected only to the root and internal nodes (the leaves in our construction have weight 1), the diameter of the tree does not increase. ◀

5 Average Linkage HAC on Paths in NC

In this section, we present an $\tilde{O}(n)$ work and $O(\mathsf{polylog}(n))$ depth algorithm for solving average linkage HAC on path graphs, provided that the aspect ratio of the input instance is bounded by $\mathsf{poly}(n)$. The *aspect ratio* is defined as $\mathcal{A} = W_{\max}/W_{\min}$, where $W_{\max} = \arg\max_{e \in E} w(e)$ and $W_{\min} = \arg\min_{e \in E} w(e)$ (note that this definition excludes all non-edges, which implicitly have a weight of 0).

In average linkage HAC, the weight (i.e., similarity) of edges monotonically decreases over time. Thus, our idea is to partition the edges into *buckets* where the edges in any bucket have the same similarity, up to constant factors. Next, we process these buckets in *phases*, from the highest similarity bucket to the lowest. In each phase, we perform a modified version of the classic nearest-neighbor chain algorithm, wherein we compute the nearest-neighbor chains for the graph induced on the edges in that bucket, and process each chain independently. We note that when we use the terminology *nearest neighbor* of a vertex in what follows, we refer to the neighbor along the *highest weight edge* incident to the vertex.

Initially, each cluster is a singleton, and we might end up with $\Omega(n)$ sequential dependencies to resolve. However, we observe that in this special case when the size of every cluster is equal, starting with the reciprocal pair, every *alternate* edge in this chain can be merged independently, and the rest of the edges will be moved to a later bucket. We can compute the edges that will be merged easily via a simple prefix-sum routine [8]. However, when the cluster sizes are arbitrary, this observation no longer holds. Nonetheless, we show that we can partition each chain further into $O(\log n)$-sized subchains such that, even though the dependencies within a subchain must be resolved sequentially, the dependencies across subchains can be resolved in parallel using a similar application of prefix-sum.

We state the following theorems showing (1) that our parallel algorithm is highly efficient (it runs near-linear time in the number of nodes) and runs in poly-logarithmic depth and (2) that our algorithm implies that the dendrogram height of a path input with polynomial aspect ratio is always poly-logarithmic. Due to space constraints, we defer the proofs and the details of the algorithm to the full version [6].

▶ **Theorem 4.** *Average linkage HAC on paths is in NC. In particular, there is an algorithm for average linkage HAC that runs in* $O(\log^2 n \log\log n)$ *depth with* $O(n \log n \log\log n)$ *work.*

▶ **Theorem 14.** *Average linkage HAC on path graphs with* $\mathsf{poly}(n)$ *aspect-ratio returns a dendrogram with height at most* $O(\log^2 n)$.

6 Conclusion

In this paper, we studied the parallel and sequential complexity of hierarchical graph clustering. We gave new classic and fine-grained reductions for Hierarchical Agglomerative Clustering (HAC) under the average linkage measure that likely rule out efficient algorithms for *exact* average linkage, parallel or otherwise. We also showed that such impossibility results can be circumvented if the output dendrogram has low height or is a path. An interesting question is whether such structure can be leveraged for other variants of interest of average linkage HAC: for example, can we can obtain dynamic algorithms for HAC that are also parameterized by the height?

References

1 Amir Abboud, Vincent Cohen-Addad, and Hussein Houdrouge. Subquadratic high-dimensional hierarchical clustering. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Annual Conference on Neural Information Processing Systems (NeurIPS)*, volume 32. Curran Associates, Inc., 2019.

2 Amir Abboud, Nick Fischer, and Yarin Shechter. Faster combinatorial k-clique algorithms. *arXiv preprint*, 2024. `arXiv:2401.13502`.

3 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

4 N. S. Arora, R. D. Blumofe, and C. G. Plaxton. Thread scheduling for multiprogrammed multiprocessors. *ACM Transactions on Computer Systems*, 34(2), April 2001.

5 MohammadHossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Raimondas Kiveris, Silvio Lattanzi, and Vahab Mirrokni. Affinity clustering: Hierarchical clustering at scale. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 6864–6874, 2017.

6 MohammadHossein Bateni, Laxman Dhulipala, Kishen N Gowda, D Ellis Hershkowitz, Rajesh Jayaram, and Jakub Łącki. It's hard to hac with average linkage!, 2024. `arXiv:2404.14730`.

7 J-P Benzécri. Construction d'une classification ascendante hiérarchique par la recherche en chaîne des voisins réciproques. *Cahiers de l'analyse des données*, 7(2):209–218, 1982.

8 Guy E Blelloch. Scans as primitive parallel operations. *IEEE Transactions on computers*, 38(11):1526–1538, 1989.

9 Guy E. Blelloch, Jeremy T. Fineman, Yan Gu, and Yihan Sun. Optimal parallel algorithms in the binary-forking model. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2020.

10 Robert D. Blumofe and Charles E. Leiserson. Space-efficient scheduling of multithreaded computations. *SIAM J. on Computing*, 27(1), 1998.

11 Mélanie Boly, Vincent Perlbarg, Guillaume Marrelec, Manuel Schabus, Steven Laureys, Julien Doyon, Mélanie Pélégrini-Issac, Pierre Maquet, and Habib Benali. Hierarchical clustering of brain activity during human nonrapid eye movement sleep. *Proceedings of the National Academy of Sciences*, 109(15):5856–5861, 2012.

12 Vincent Cohen-Addad, Varun Kanade, Frederik Mallmann-Trenn, and Claire Mathieu. Hierarchical clustering: Objective functions and algorithms. *Journal of the ACM (JACM)*, 66(4), 2019.

13 Stephen A Cook, Yuval Filmus, and Dai Tri Man Le. The complexity of the comparator circuit value problem. *ACM Transactions on Computation Theory (TOCT)*, 6(4):1–44, 2014.

14 Don Coppersmith and Shmuel Winograd. On the asymptotic complexity of matrix multiplication. *SIAM Journal on Computing*, 11(3):472–492, 1982.

15 Laxman Dhulipala, David Eisenstat, Jakub Lacki, Vahab Mirrokni, and Jessica Shi. Hierarchical agglomerative graph clustering in nearly-linear time. In *International Conference on Machine Learning (ICML)*, pages 2676–2686, 2021.

16 Laxman Dhulipala, David Eisenstat, Jakub Lacki, Vahab Mirrokni, and Jessica Shi. Hierarchical agglomerative graph clustering in poly-logarithmic depth. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 35:22925–22940, 2022.

17 Laxman Dhulipala, Jakub Łącki, Jason Lee, and Vahab Mirrokni. Terahac: Hierarchical agglomerative clustering of trillion-edge graphs. *Proceedings of the ACM on Management of Data*, 1(3):1–27, 2023.

18 Michael B Eisen, Paul T Spellman, Patrick O Brown, and David Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences*, 95(25):14863–14868, 1998.

19 Raymond Greenlaw and Sanpawat Kantabutra. On the parallel complexity of hierarchical clustering and cc-complete problems. *Complexity*, 14(2):18–28, 2008.

20 Guan-Jie Hua, Che-Lun Hung, Chun-Yuan Lin, Fu-Che Wu, Yu-Wei Chan, and Chuan Yi Tang. MGUPGMA: a fast UPGMA algorithm with multiple graphics processing units using NCCL. *Evolutionary Bioinformatics*, 13:1176934317734220, 2017.

21 J Juan. Programme de classification hiérarchique par l'algorithme de la recherche en chaîne des voisins réciproques. *Cahiers de l'analyse des données*, 7(2):219–225, 1982.

22 Benjamin King. Step-wise clustering procedures. *Journal of the American Statistical Association*, 62(317):86–101, 1967.

23 Ari Kobren, Nicholas Monath, Akshay Krishnamurthy, and Andrew McCallum. A hierarchical algorithm for extreme clustering. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 255–264, 2017.

24 Godfrey N Lance and William Thomas Williams. A general theory of classificatory sorting strategies: 1. hierarchical systems. *The computer journal*, 9(4):373–380, 1967.

25 Joshua Lau and Angus Ritossa. Algorithms and hardness for multidimensional range updates and queries. In *Innovations in Theoretical Computer Science Conference (ITCS)*, 2021.

26 François Le Gall. Faster algorithms for rectangular matrix multiplication. In *Symposium on Foundations of Computer Science (FOCS)*, pages 514–523. IEEE, 2012.

27 Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

28 Ernst W Mayr and Ashok Subramanian. The complexity of circuit value and network stability. *Journal of Computer and System Sciences*, 44(2):302–323, 1992.

29 Nicholas Monath, Kumar Avinava Dubey, Guru Guruganesh, Manzil Zaheer, Amr Ahmed, Andrew McCallum, Gokhan Mergen, Marc Najork, Mert Terzihan, Bryon Tjanaka, et al. Scalable hierarchical agglomerative clustering. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1245–1255, 2021.

30 Benjamin Moseley and Joshua R. Wang. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 3094–3103, 2017.

31 Benjamin Moseley and Joshua R Wang. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. *Journal of Machine Learning Research*, 24(1):1–36, 2023.

32 Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint*, 2011. `arXiv:1109.2378`.

33 Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

34 Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview, ii. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(6):e1219, 2017.

35 Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

36 JM Shearer and Michael A Wolfe. Alglib, a simple symbol-manipulation package. *Communications of the ACM*, 28(8):820–825, 1985.

37 Peter Henry Andrews Sneath. The principles and practice of numerical classification. *Numerical taxonomy*, 573, 1973.

38 Ashok Subramanian. *A new approach to stable matching problems*. Stanford University, 1989.

39 Tom Tseng, Laxman Dhulipala, and Julian Shun. Parallel batch-dynamic minimum spanning forest and the efficiency of dynamic agglomerative graph clustering. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 233–245, 2022.

40 Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.

41 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 887–898, 2012.

42 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Symposium on Foundations of Computer Science (FOCS)*, pages 645–654, 2010.

43 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3792–3835. SIAM, 2024.

44 Ying Zhao and George Karypis. Evaluation of hierarchical clustering algorithms for document datasets. In *Conference on Information and Knowledge Management (CIKM)*, pages 515–524, 2002.

Sublinear Algorithms for TSP via Path Covers

Soheil Behnezhad
Northeastern University, Boston, MA, USA

Mohammad Roghani
Stanford University, CA, USA

Aviad Rubinstein
Stanford University, CA, USA

Amin Saberi
Stanford University, CA, USA

Abstract

We study sublinear time algorithms for the *traveling salesman problem* (TSP). First, we focus on the closely related *maximum path cover* problem, which asks for a collection of vertex disjoint paths that include the maximum number of edges. We show that for any fixed $\varepsilon > 0$, there is an algorithm that $(1/2 - \varepsilon)$-approximates the maximum path cover size of an n-vertex graph in $\widetilde{O}(n)$ time. This improves upon a $(3/8 - \varepsilon)$-approximate $\widetilde{O}(n\sqrt{n})$-time algorithm of Chen, Kannan, and Khanna [ICALP'20].

Equipped with our path cover algorithm, we give an $\widetilde{O}(n)$ time algorithm that estimates the cost of $(1, 2)$-TSP within a factor of $(1.5 + \varepsilon)$ which is an improvement over a folklore $(1.75 + \varepsilon)$-approximate $\widetilde{O}(n)$-time algorithm, as well as a $(1.625 + \varepsilon)$-approximate $\widetilde{O}(n\sqrt{n})$-time algorithm of [CHK ICALP'20]. For graphic TSP, we present an $\widetilde{O}(n)$ algorithm that estimates the cost of graphic TSP within a factor of 1.83 which is an improvement over a 1.92-approximate $\widetilde{O}(n)$ time algorithm due to [CHK ICALP'20, Behnezhad FOCS'21]. We show that the approximation can be further improved to 1.66 using $n^{2-\Omega(1)}$ time.

All of our $\widetilde{O}(n)$ time algorithms are information-theoretically time-optimal up to $\operatorname{poly}\log n$ factors. Additionally, we show that our approximation guarantees for path cover and $(1, 2)$-TSP hit a natural barrier: We show better approximations require better sublinear time algorithms for the well-studied maximum matching problem.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Sublinear Algorithms, Traveling Salesman Problem, Approximation Algorithm, (1, 2)-TSP, Graphic TSP

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.19

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/pdf/2301.05350`

Funding Mohammad Roghani and Amin Saberi were supported by NSF award 1812919 and ONR award 141912550. Soheil Behnezhad and Aviad Rubinstein were supported by NSF CCF-1954927, and a David and Lucile Packard Fellowship. Soheil Behnezhad was additionally supported by NSF Awards 1942123, 1812919 and by Moses Charikar's Simons Investigator Award.

1 Introduction

The *traveling salesman problem* (TSP) is a central problem in combinatorial optimization. Given a set V of n vertices and their pairwise distances, it asks for a Hamiltonian cycle of the minimum cost. In this paper, we study *sublinear time* algorithms for TSP. The algorithm is given query access to the distance pairs, and the goal is to estimate the solution cost in time sublinear in the input size (which is $\Theta(n^2)$).

TSP is NP-hard to approximate within a polynomial factor for an arbitrary distance function. As such, much of the work in the literature has been on more specific distance functions. Some notable examples include *graphic TSP* [15, 22, 23, 25, 10] where the distances are the shortest paths over an arbitrary unweighted undirected graph, $(1,2)$-*TSP* [1, 10, 7, 18, 21] where the distances are 1 or 2, and more generally *metric TSP* [17, 14, 12, 26] where the distances satsify triangle inequality.

In 2003, Czumaj and Sohler [13, 14] showed that for any fixed $\varepsilon > 0$, a $(1+\varepsilon)$-approximation of the cost of metric minimum spanning tree (MST) and thus a $(2+\varepsilon)$-approximation of the cost of metric TSP can be found in $\widetilde{O}(n)$ time. Twenty years later, it still remains a major open problem to either break two-approximation in $n^{2-\Omega(1)}$ time or prove a lower bound.[1] However, better bounds are known for both graphic TSP and $(1,2)$-TSP. In this paper, we present improved algorithms for these two well-studied variants of TSP. Our main tool to achieve this is an improved algorithm for the closely related *maximum path cover* problem which might be of independent interest.

Maximum Path Cover. The maximum path cover in a graph is a collection of vertex disjoint paths with the maximum number of edges in it. The (almost) 1/2-approximate maximum matching size estimator of Behnezhad [2] immediately implies an (almost) 1/4-approximation for the maximum path cover problem in $\widetilde{O}(n)$ time.[2] This can be improved to an (almost) $(3/8 = .375)$-approximation using the *matching-pair* idea of Chen, Kannan, and Khanna [10] in $\widetilde{O}(n\sqrt{n})$-time.[3] Our first main contribution is an improvement over both of these results:

▶ **Result 1** (Formally as Theorem 17). *For any $\varepsilon > 0$, there is a randomized algorithm that w.h.p. $(1/2-\varepsilon)$-approximates the size of maximum path cover in $\widetilde{O}(n \cdot \text{poly}(1/\varepsilon))$ time.*

Besides quantitavely improving prior work both in the running time and the approximation ratio, Result 1 reaches a qualitatively important milestone as well. First, the running time of Result 1 is information-theoretically optimal up to $\text{poly}\log n$ factors (the lower bound holds for any constant approximation – see the arXiv version of the paper). Second, its approximation ratio hits a rather important barrier. We give a non-trivial reduction that shows a $(1/2+\Omega(1))$-approximation in $\widetilde{O}(n)$ time for maximum path cover would imply the same bound for maximum matching in bipartite graphs. Such a result has remained elusive for matching, which is one of the most extensively studied problems in the literature of sublinear time algorithms. See the arXiv version of the paper for the lower bound details.

It is also worth noting that in bounding the running time of our algorithm in Result 1, we use connections to parallel algorithms. Such a connection was previously only used for matchings [2].

$(1,2)$-TSP. The $(1,2)$-TSP problem has been studied extensively in the classical setting. In his landmark paper, Karp [18] showed that $(1,2)$-TSP is NP-hard. Papadimitriou and Yannakakis [24] then proved its APX-hardness. Since then there has been a significant amount of work on $(1,2)$-TSP in the classical setting. The current best known inapproximability bound for $(1,2)$-TSP is 535/534 [19]. After a series of works, the best known polynomial

[1] See e.g. Open Problem 71 on sublinear.info [16].

[2] The application of sublinear time maximum matching algorithms for approximating maximum path cover was first proposed by Gupta and Onak. See [16].

[3] We note that even though a subsequent result of Behnezhad [2] improved the running time for maximal matchings and graphic TSP from $O(n\sqrt{n})$ in [10] to $\widetilde{O}(n)$, it is not immediately clear whether the same holds for path cover and $(1,2)$-TSP as they rely on a different notion of a matching pair.

Table 1 Comparison of running time and approximation ratio of our TSP algorithms and lower bounds with prior work.

Running Time	Approximation Ratio	Metric	Reference
$\widetilde{O}(n)$	$1.75+\varepsilon$	(1,2)	Folklore
$\widetilde{O}(n\sqrt{n})$	$1.625+\varepsilon$	(1,2)	Chen, Kannan, and Khanna [10]
$\widetilde{O}(n)$	$1.5+\varepsilon$	(1,2)	**This work** (Result 2)
$\widetilde{O}(n)$	1.929	Graphic	Chen, Kannan, and Khanna [10]
$\widetilde{O}(n)$	1.834	Graphic	**This work** (Result 3)
$n^{2-\Omega(1)}$	1.667	Graphic	**This work** (Result 4)
$\Omega(n^2)$	$1+\varepsilon$	(1,2) & Graphic	Chen, Kannan, and Khanna [10]
$n^{1+\Omega(1)}$ (Conditional)	$1.5-\varepsilon$	(1,2) & Graphic	**This work** (arXiv version)

time approximation is 8/7 [7] which can be implemented in $O(n^3)$ time [1]. For sublinear time algorithms, an $\widetilde{O}(n)$-time (almost) 1.75-approximation is folklore [16]. Chen, Kannan, and Khanna [10] improved the approximation to (almost) 1.625 in $\widetilde{O}(n\sqrt{n})$ time.

It is not hard to see that up to a small additive error of 1, $(1,2)$-TSP is equivalent to finding a maximum path cover on the weight-1 edges and then connecting their endpoints via weight-2 edges. A simple calculation shows that any α-approximation for the maximum path cover problem leads to a $(2-\alpha)$-approximation for $(1,2)$-TSP. Our path cover algorithm of Result 1 immediately implies the following result as a corollary:

▶ **Result 2.** *For any $\varepsilon > 0$, there is a randomized algorithm that w.h.p. $(1.5+\varepsilon)$-approximates the cost of $(1,2)$-TSP in $\widetilde{O}(n \cdot \text{poly}(1/\varepsilon))$ time.*

Similar to Result 1, the running time of Result 2 is information-theoretically optimal up to $\text{poly}\log n$ factors, and its approximation ratio hits a natural barrier due to a connection to sublinear time matching that we establish in this work.

Graphic TSP. The graphic TSP problem is equivalent to finding a tour of the minimium size that visits all the vertices. This is an important instance of TSP that has received a lot of attention over the years. For polynomial time algorithms, a 1.5-approximation of Christofides [12] (which also works more generally for metric TSP) had remained the best known until a series of works over the last decade improved it to $(1.5-\varepsilon_0)$ [15], 1.461 [22], 1.444 [23], and finally to 1.4 [25]. For sublinear time algorithms, Chen, Kannan, and Khanna [10] showed that an (almost) $(27/14 \approx 1.928)$-approximation of graphic TSP can be obtained in $\widetilde{O}(n\sqrt{n})$ time. The running time was subsequently improved to $\widetilde{O}(n)$ by Behnezhad [2].

We first show that plugging Result 1 into the framework of [10] immediately improves their approximation from 1.928 to (almost) 1.9 while keeping the running time $\widetilde{O}(n)$. We then give a more fine tuned algorithm that obtains a much improved approximation ratio of $(11/6 \approx 1.833)$.

▶ **Result 3.** *For any $\varepsilon > 0$, there is a randomized algorithm that w.h.p. $(1+\varepsilon)(\frac{11}{6} \approx 1.833)$-approximates the cost of graphic TSP in $\widetilde{O}(n \cdot \text{poly}(1/\varepsilon))$ time.*

Over the past few years, significant advancements have been made in the development of sublinear matching algorithms and lower bounds. Several recent results [3, 4, 5, 6, 8, 9] have led to the creation of a $(1, \varepsilon n)$-approximation algorithm for maximum matching, with running time of $n^{2-\Omega_\varepsilon(n)}$. Leveraging these sublinear algorithms, we have devised a slightly subquadratic algorithm that provides a more accurate estimation of the size of graphic TSP.

▶ **Result 4.** *For any $\varepsilon > 0$, there is a randomized algorithm that w.h.p. $(1+\varepsilon)(\frac{5}{3} \approx 1.666)$-approximates the cost of graphic TSP in $n^{2-\Omega_\varepsilon(1)}$ time.*

We contrast our results with prior sublinear TSP algorithms in Table 1.

Further related work. Finally, we note that in a recent paper, Chen, Khanna, and Tan [11] show that assuming that the metric has a spanning tree supported on weight 1 edges, one can obtain a $(2 - \varepsilon_0)$-approximation with $\widetilde{O}(n\sqrt{n})$ queries for some small unspecified constant $\varepsilon_0 > 0$. While this is a more general metric than graphic TSP and (1,2)-TSP that we study in this paper, we note that the two papers are orthogonal and their results are incomparable. In particular, the techniques developed in this paper are specifically designed to improve the approximation to much below 2, whereas [11] focuses on generalizing the distance function while beating 2.

2 Technical Overview

In this section, we give an overview of our algorithms, especially our sublinear time maximum path cover algorithm of Result 1 which is the key to the other results as well.

Let us start with using matchings to approximate maximum path cover. Consider a graph that has a Hamiltonian path. Here, the optimal maximum path cover has size $n - 1$. On the other hand, any maximum matching can have at most $n/2$ edges, which is by a factor 2 smaller than our optimal path cover. On top of this, we only know close to 1/2 approximations for maximum matching if we restrict the running to be close to linear in n [2, 6], thus can only achieve an approximation close to 1/4.

Instead of a single matching, Chen, Kannan, and Khanna [10] showed how to estimate the number of edges in a *maximal matching pair* in $\widetilde{O}(n\sqrt{n})$ time, where a matching pair is simply two edge disjoint matchings. It is not hard to see that the number of edges in a maximal matching pair is at least half the number of edges in a maximum path cover. The problem, however, is that a maximal matching pair is not a collection of paths! In particular, the two matchings can form cycles of length as small as four. Therefore, one may only be able to use 3/4 fraction of the edges of a matching pair in a path cover. This is precisely why the algorithm of [10] only obtains a $\frac{1}{2} \times \frac{3}{4} = \frac{3}{8}$ approximation for path cover, and a $2 - \frac{3}{8} = 1.625$ approximation for $(1, 2)$-TSP.

If we could modify the matching pair algorithm of [10], and avoid cycles by manually excluding edges whose endpoints are the endpoints of a path in the current matching pair, then we could avoid the 3/4 factor loss discussed above and achieve a 1/2-approximation. Unfortunately, checking whether the endpoints of an edge are endpoints of a path requires knowledge about whether a series of other edges belong to the solution, which seems hard to implement in sublinear time.

Instead of checking for cycles manually, we introduce the following Algorithm 1 which avoids cycles more naturally. While our final algorithm is a modified variant of Algorithm 1 described below, we start with Algorithm 1 as we believe it provides the right intuition.

Algorithm 1 A new algorithm for path cover.

1. Initialize $P \leftarrow \emptyset$.
2. Each vertex v has two *ports* that we denote by v^0 and v^1. Each of these ports throughout the algorithm will be either *free* or *occupied*. Initially, all ports are free.
3. Iterate over the edges in some ordering π. Upon visiting an edge $e = (u, v)$:
   - If v^0 and u^0 are free, add e to P, mark both as occupied, and skip to the next edge.
   - If v^1 and u^0 are free, add e to P, mark both as occupied, and skip to the next edge.
   - If v^0 and u^1 are free, add e to P, mark both as occupied, and skip to the next edge.
4. Return P.

Two properties of Algorithm 1 are crucial. First, it prioritizes occupying (u^0, v^0) (compared to (u^1, v^0) or (u^0, v^1)) which in particular implies that any component in P must have a (u^0, v^0) edge. Second, it never occupies (u^1, v^1) with an edge (u, v). While it is easy to see that the output of Algorithm 1 has maximum degree 2, and is thus a collection of paths or cycles, the two properties above actually guarantee that it never includes any cycle. See Figure 1. We provide the formal proof of this later in Section 4. Additionally, we show that the output of Algorithm 1 must be at least half the size of a maximum path cover, as we prove next. Hence, if we manage to estimate the size of the output P of Algorithm 1, then we have proved Result 1.

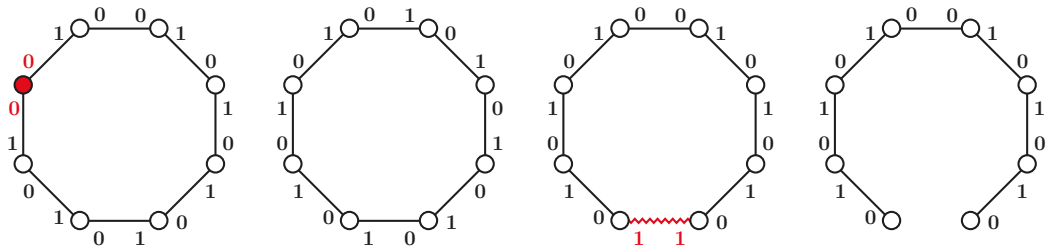


Figure 1 Examples of why the output of Algorithm 1 will not have cycles.

Our final algorithm is slightly different from Algorithm 1 discussed above. In particular, we slightly relax it – see Algorithm 2 – so that it can be solved via a randomized greedy maximal independent set (RGMIS), for which we have a rich toolkit of sublinear time estimators. Existing approaches (particularly the algorithm of Yoshida, Yamamoto, and Ito [27] and its two-step implementation by Chen, Kannan, and Khanna [10]) can be employed to estimate the value of this modified Algorithm 2 in $\widetilde{O}(n\sqrt{n})$ time. We achieve the improved, and near tight, $\widetilde{O}(n)$ time bound guarantee of Result 1 by building on the analysis of Behnezhad [2] for maximal independent set on the line graphs (i.e., maximal matchings). Though we note that several new ideas are needed, because the MIS graph in our case will not be exactly a line graph. We defer more discussions about this to Sections 4 and 5.

Implications for TSP. By having an α-approximate maximum path cover algorithm, we immediately obtain a $(2-\alpha)$-approximation for $(1,2)$-TSP. Therefore, the algorithm above immediately proves Result 2 that we can (almost) 1.5-approximate $(1,2)$-TSP in $\widetilde{O}(n)$ time. For our Result 3 for graphic TSP, we first observe that our improved path cover algorithm can be employed to provide a better lower bound for the optimal TSP solution. This improves the 1.92-approximation of [10] as black-box to 1.9-approximation (see the arXiv version of the paper). However, the final improvement to 1.83 requires more ideas, in particular, on how to better estimate the number of certain *bridges* in the graph. See Section 8 for more details about this.

3 Preliminaries

Problem Definition. In the sublinear TSP problem, we have a set V of n vertices and a distance function $d : V \times V \to \mathbb{R}_+$. The algorithm has query access to this distance function. Namely, for any pair (u,v) of the vertices of its choice, the algorithm may query the value of $d(u,v)$. The goal is to design an algorithm that runs in sublinear time in the input size, which is $\Theta(n^2)$ (all the distance pairs), and produces an estimate of the size of the optimal TSP solution. Denoting the optimal TSP value by $\tau(V)$, we say an estimate $\widetilde{\tau}(V)$ provides an α-approximation for $\alpha \geq 1$ if

$$\tau(V) \leq \widetilde{\tau}(V) \leq \alpha \cdot \tau(V).$$

We focus specifically on *graph TSP* and $(1,2)$-*TSP* problems. In graphic TSP, the distance function d is the shortest path metric on an unweighted undirected graph G that is unknown to the algorithm. Note, however, that the distance queries essentially provide *adjacency matrix* access to this graph G. In $(1,2)$-TSP, the assumption is that $d(u,v) \in \{1,2\}$ for every pair u,v. In the case of $(1,2)$-TSP we may use G to refer to the subgraph induced on the pairs with distance 1.

Defining graph G as above, we use n to denote the number of its vertices, m to denote the number of its edges, Δ to denote its maximum degree, $\mu(G)$ to denote its maximum matching size, $\nu(G)$ to denote its minimum vertex cover size, and $\bar{d}$ to denote its average degree.

Path Cover Definitions. Given an unweighted graph G, a path cover in G is a collection of vertex disjoint paths in G. A maximum path cover is a path cover of G with the maximum number of edges in it (note that we are not counting the number of paths, but rather the total number of edges in them). We use $\rho(G)$ to denote the size of the maximum path cover in G. We say an estimate $\widetilde{\rho}(G)$ for $\rho(G)$ provides an (α,ε)-approximation for $\alpha,\varepsilon \in [0,1]$ if

$$\alpha \cdot \rho(G) - \varepsilon n \leq \widetilde{\rho}(G) \leq \rho(G).$$

We may also use α-approximation instead of $(\alpha,0)$-approximation.

Graph Theory Definitions/Tools. A *bridge (cut edge)* in a graph is an edge whose deletion increases the number of connected components. Similarly, a *cut vertex* is a vertex whose deletion (along with its edges) increases the number of connected components. A *biconnected graph* is a connected graph with no cut vertex. Also, a *biconnected component (block)* of a graph is a maximal biconnected subgraph of the original graph. A non-trivial biconnected component is a block that is not a bridge. We say a graph is *2-edge-connected* if there is no bridge in the graph. A *2-edge-connected component* of a graph is maximal 2-edge-connected

subgraph of the original graph. The *bridge-block tree* of a graph is a tree obtained by contracting the 2-edge-connected components; note that the edge set of a bridge-block tree correspond to the bridges in the original graph.

We use the following classic theorem of König [20] that the size of the minimum vertex cover is equal to the size of maximum matching in bipartite graphs. Namely:

▶ **Proposition 1** (König's Theorem). *In any bipartite graph G, $\mu(G) = \nu(G)$.*

Probabilistic Tools. In our proofs, we use the following standard concentration inequalities.

▶ **Proposition 2** (Chernoff Bound). *Let $X_1, X_2, \ldots, X_n$ be independent Bernoulli random variables. Let $X = \sum_{i=1}^{n} X_i$. For any $t > 0$, $\Pr[|X - \mathbf{E}[X]| \geq t] \leq 2\exp\left(-\frac{t^2}{3\,\mathbf{E}[X]}\right)$.*

▶ **Proposition 3** (Hoeffding's Inequality). *Let $X_1, X_2, \ldots, X_n$ be independent random variables such that $a \leq X_i \leq b$. Let $\bar{X} = (\sum_{i=1}^{n} X_i)/n$. For any $t > 0$, $\Pr[|\bar{X} - \mathbf{E}[X]| \geq t] \leq 2\exp\left(-\frac{2nt}{(b-a)^2}\right)$.*

4 New Meta Algorithms for Maximum Path Cover

In this section, we present a new meta algorithm for maximum path cover that obtains a 1/2-approximation. The algorithm, as we will state it in this section, will not be particularly in the sublinear time model. We discuss its sublinear time implementation later in Sections 5 and 6.

Our starting point is the Algorithm 1 described in Section 2. Let us first formally prove that it obtains a 1/2-approximation, and that no component in it is a cycle.

▷ Claim 4. The output of Algorithm 1 is a collection of disjoint paths.

Proof. Since P has maximum degree two, it suffices to show none of its connected components are cycles. Property (i) above implies that at any point during the algorithm, any degree one vertex v has its port v^0 occupied. Now take an edge $e = (u, v)$ that forms a cycle if added to P. Both u and v must have degree one and so u^0 and v^0 are occupied. Since by property (ii) edge e does not occupy both v^1 and u^1, the algorithm does not add e to P thus not completing a cycle. ◁

▷ Claim 5. Let $P^\star$ be *any* path cover using weight one edges. Then the output of Algorithm 1 has size at least $\frac{1}{2}|P^\star|$.

Proof. For any edge $e = (u, v) \in P^\star$ define $\phi(e) = \frac{1}{4}(\deg_P(u) + \deg_P(v))$. We first claim that for every edge $e = (u, v)$ in G, we have $\phi(e) \geq 1/2$ (or, equivalently, $\deg_P(u) + \deg_P(v) \geq 2$). This is clear for edges $e \in P$ due to the contribution of e itself to its endpoints' degrees, so fix $e \notin P$. Consider the time that we process $e = (u, v)$ in the algorithm and decide not to add it to P. We claim that out of v^0, v^1, u^0, u^1 at least two ports must be occupied. Suppose w.l.o.g. and for contradiction that only v^x is occupied for $x \in \{0, 1\}$. Then (u, v) can occupy v^{1-x} and u^x and be added to P. This contradicts (u, v) not being added to P and proves our claim that $\phi(e) \geq 1/2$.

From the discussion above, we get that

$$\sum_{e \in P^\star} \phi(e) \geq \sum_{e \in P^\star} 1/2 = |P^\star|/2.$$

Moreover, because every vertex has degree at most two in $P^\star$, we get

$$\sum_{e \in P^\star} \phi(e) = \frac{1}{4} \sum_{(u,v) \in P^\star} \deg_P(u) + \deg_P(v) \leq \frac{1}{4} \cdot 2 \sum_{v \in V} \deg_P(v) = |P|.$$

The two inequalities above combined imply that $|P| \geq |P^\star|/2$. ◁

As discussed, our final algorithm is different from Algorithm 1 discussed above. One problem with Algorithm 1 is that it cannot be cast as an instance of the randomized greedy maximal independent set (RGMIS) algorithm for which there is a rich toolkit of sublinear time estimators. To remedy this, we present a modified variant of Algorithm 1 whose output is (almost) as good, but in addition can be modeled as an instance of RGMIS. We denote the output of RGMIS on a graph G with a permutation π on its vertices by $\text{RGMIS}(G, \pi)$.

The algorithm is stated below as Algorithm 2. Similar to the output of Algorithm 1, the output of Algorithm 2 can be verified to have maximum degree two. Thus, it is a collection of paths and cycles. But unlike Algorithm 1, the output of Algorithm 2 can have cycles. This happens since, unlike Algorithm 1, each connected component of the output of Algorithm 2 is not guaranteed to have an edge (u, v) occupying both u^0 and v^0. Nonetheless, we are able to show that this bad event only happens for a small fraction of connected components of the output of Algorithm 2 in expectation, and so once we remove one edge of each of these cycles, the resulting collection of disjoint paths has almost the same size.

Algorithm 2 A modification of Algorithm 1 that uses RGMIS.

1. **Parameter:** K (think of it as a large constant integer).
2. Let $G = (V, E)$ be the subgraph of weight one edges. We construct a graph $H = (V_H, E_H)$ from G on which we run RGMIS.
3. Each vertex in H corresponds to an edge e in G and two *ports* (as in Algorithm 1) of the endpoints of e that it occupies. Formally, for any $(u, v) \in E$ we have $K + 2$ vertices in H:
   - One vertex that corresponds to occuping u^0 and v^1.
   - One vertex that corresponds to occuping u^1 and v^0.
   - K vertices that each corresponds to occuping u^0 and v^0.
4. Consider two distinct vertices a and b in H corresponding to edges e_a and e_b in G:
   - If $e_a = e_b$ then we add an edge between a and b in H.
   - If e_a and e_b share exactly one endpoint v and both a and b occupy the same port of v, we add an edge between a and b in H.
5. Find a randomized greedy maximal independent set I of H.
6. Let P be the set of edges in G corresponding to the vertices in I.
7. Return P.

▶ **Observation 6.** *Let C be a connected component in the output of Algorithm 2. If C is a cycle, then every edge in C occupies one 0-port and one 1-port (that is, no edge occupies two 0-ports).*

Proof. Suppose that C has n' vertices. Since each vertex in a cycle has degree two, both ports of each vertex in C must be occupied. Hence, n' 0-ports and n' 1-ports of C are occupied in total. Given that any edge occupies at least one 0-port by the algorithm, we cannot have an edge that occupies two 0-ports, or else we should occupy more 0-ports than 1-ports of C, which is a contradiction. ◀

Next, we show that up to a factor of $(1+2/k)$ which is negligible for K in the order $1/\varepsilon$, the output of Algorithm 2 is an (almost) 1/2-approximation of the maximum path cover value.

▶ **Observation 7.** *Let C be a connected component in the output of Algorithm 2. If C is a path, then it contains at most one edge that occupies two 0-ports.*

Proof. Let C be the path $(v_1, v_2, \ldots, v_r)$. Since the degree of any vertex v_i for $1 < i < r$ is two in the path, both ports of v_i must be occupied. For v_1 and v_r, on the other hand, only one port is occupied. Hence, the total number of 0-ports that are occupied by C minus the number of 1-ports occupied by it is at most two. This means that there is at most one edge that occupies two 0-ports since all other types of edges occupy exactly one 0-port and one 1-port. ◀

▶ **Lemma 8.** *let P be the output of Algorithm 2 on graph G. Then*

$$\frac{1}{2}\rho(G) \leq \mathbf{E}\,|P| \leq \left(1+\frac{2}{K}\right)\rho(G),$$

where the expectation is taken over the randomization of computing RGMIS in Algorithm 2.

Proof. Let P^* be a maximum path cover. For any edge $e = (u, v) \in P^\star$ define $\phi(e) = \frac{1}{4}(\deg_P(u) + \deg_P(v))$. With the exact same argument as in the proof of Claim 5, we get that $\phi(e) \geq 1/2$, which implies

$$\sum_{e\in P^\star} \phi(e) \geq \sum_{e\in P^\star} 1/2 = \rho(G)/2.$$

Since the degree of each vertex in P is at most two, we get

$$\sum_{e\in P^\star} \phi(e) = \frac{1}{4}\sum_{(u,v)\in P^\star} \deg_P(u) + \deg_P(v) \leq \frac{1}{4}\cdot 2\sum_{v\in V}\deg_P(v) = |P|.$$

By combining above inequalities we get $\frac{1}{2}\rho(G) \leq |P|$. Note that we do not need the randomization for the proof of the lower bound.

By construction of P, every vertex has degree at most two in P. Hence, all connected components of P are cycles and paths. We claim that at most $\frac{2}{K+2}$ fraction of connected components are cycles in expectation. Since the expected number of connected components is at most $\mathbf{E}\,|P|$, from this we get that the expected number of cycles is at most $2\,\mathbf{E}\,|P|/(K+2)$. By removing one edge from each cycle, we obtain a valid solution for maximum path cover problem. Thus,

$$\mathbf{E}\,|P| - \frac{2\,\mathbf{E}\,|P|}{K+2} = \frac{K}{K+2}\,\mathbf{E}\,|P| \leq \rho(G) \qquad \Rightarrow \qquad \mathbf{E}\,|P| \leq \left(1+\frac{2}{K}\right)\cdot\rho(G).$$

So it remains to show that at most $\frac{2}{K+2}$ fraction of connected components are cycles in expectation. As we process edges one by one according to the ordering of RGMIS, let A be the set of edges that none of their incident edges are added to the solution of Algorithm 2. By definition of A, if one copy of edge (u, v) is in A, then all other copies of (u, v) are also in A. Therefore, at any point during running RGMIS, if a new component is added to the solution, the edge (u, v) that gets added to the solution occupies (u^0, v^0) with probability at least $\frac{K}{K+2}$ since K copies out of the $K+2$ copies are for (u^0, v^0). Let C_0 be the number of times that the newly added component is an edge occupying two 0-ports, and C_1 be the number of times that the newly added component is an edge occupying one 0-port and one 1-port. By the above argument, we have

$$\frac{\mathbf{E}[C_0]}{\mathbf{E}[C_0]+\mathbf{E}[C_1]} = \frac{K}{K+2}. \tag{1}$$

Note that after running Algorithm 2, it is possible that the number of connected components is actually smaller than $C_0 + C_1$, since some of the components may merge as the algorithm proceeds. However, by Observation 7, two components that their first edge occupies two 0-ports will not merge together. Also, by Observation 6, none of the cycle components have an edge that occupies two 0-ports. Therefore, in the end, there exists at most $\mathbf{E}[C_0] + \mathbf{E}[C_1]$ connected components and at least $\mathbf{E}[C_0]$ of them will not be cycles. This completes the proof. ◀

5 A Local Query Process for Algorithm 2 and its Complexity

In this section, we define a query process to estimate the size of the output of Algorithm 2.

In graph H of Algorithm 2, each vertex corresponds to an edge in the original graph. More precisely, we make $K+2$ copies of each edge (u, v) such that one of the copies corresponds to an edge occupying (u^0, v^1), one for (u^1, v^0), and K for (u^0, v^0). We use $G' = (V, E')$ to show the new graph with these parallel edges. During the course of Algorithm 2, two different edges that share the same endpoint and port cannot appear in the solution together. We use the following definition to formalize this notion.

▶ **Definition 9** (Conflicting Pair of Edges). *Two edges $e, e' \in E'$ that share an endpoint v are* conflicting *if both e and e' correspond to same port v^i for $i \in \{0, 1\}$. We call (e, e') a conflicting pair of edges.*

In order to estimate the size of the output of Algorithm 2, we define a vertex oracle that given a vertex v and a permutation π on E', returns the degree of vertex v in the output of Algorithm 2. These are akin to the query processes used before in the works of [2, 27], but are specific to our Algorithm 2.

Algorithm 3 "vertex oracle" $\text{VO}(u, \pi)$ to determine the degree of vertex u in $\text{RGMIS}(G', \pi)$.

```
1 Let e_1 = (u, v_1), ..., e_r = (u, v_r) be the edges incident to u with π(e_1) < ... < π(e_r).
2 d ← 0
3 for i in 1...r do
4     if EO(e_i, v_i, π) = TRUE then d ← d + 1;
5 return d
```

Note that in Line 2 of the Algorithm 4 we only recursively call the function on edges that their label, conflict with edge e since if other edges appear in the RMGIS subgraph, we can still have e in the RGMIS subgraph. Before analyzing the query complexity of the vertex oracle, we prove the correctness of the vertex oracle.

▷ Claim 10. For any edge $e = (u, z) \in E'$ that is occupying ports u^i and z^j, if $\text{EO}(e, u, \pi)$ is called while computing $\text{VO}(v, \pi)$, then $\text{EO}(e, u, \pi) = \text{TRUE}$ iff $e \in \text{RGMIS}(G', \pi)$.

Proof. We prove the claim using induction on ranking of edge e. Assume that the claim is true for all edges with ranking smaller than $\pi(e)$. If $\text{EO}(e, u, \pi)$ is called by $\text{EO}(e' = (w, z), z, \pi)$ or directly by $\text{VO}(v, \pi)$, then by definition of Algorithm 4 and Algorithm 3, all edges $e'' = (w', z)$ with $\pi(e'') < \pi(e')$ that are occupying z^j are queried before e' which means that

Algorithm 4 "edge oracle" $\text{EO}(e, u, \pi)$ to determine an edge e is in $\text{RGMIS}(G', \pi)$. Also, u must be an endpoint of e.

```
if EO(e, u, π) computed before then return the computed result.;
Let e_1 = (u, v_1), ..., e_r = (u, v_r) be the edges incident to e such that
  π(e_1) < ... < π(e_r) < π(e). Also, (e, e_i) is a conflicting pair for all 1 ≤ i ≤ r.
for i in 1...r do
    if EO(e_i, v_i, π) = True then return False;
return True
```

none of them return True. Hence, by induction hypothesis, none of the edges incident to z that are occupying z^j with lower rank are in the $\text{RGMIS}(G', \pi)$. Moreover, $\text{EO}(e, u, \pi)$ calls all incident edges to u with lower rank that are occupying u^i and return Trueif none of them are in the $\text{RGMIS}(G', \pi)$ by induction hypothesis. Therefore, $\text{EO}(e, u, \pi) = \text{True}$ iff $e \in \text{RGMIS}(G', \pi)$. ◁

▷ Claim 11. Let $v \in V$ and d be the output of $\text{VO}(v, \pi)$. Then d is equal to the degree of vertex v in the subgraph outputted by $\text{RGMIS}(G', \pi)$.

Proof. The observation follows by combining the fact that the vertex oracle queries edges in increasing order and Claim 10. ◁

Let $T(v, \pi)$ denote the number of recursive calls to the edge oracle during the execution of $\text{VO}(v, \pi)$.

▶ **Theorem 12.** *For a randomly chosen vertex v and permutation π on E', we have that*

$$\mathbf{E}_{v,\pi}[T(v, \pi)] = O(\bar{d} \cdot \log^2 n)$$

where $\bar{d}$ is the average degree of the graph G.

Let $Q(e, v, \pi)$ be the number of $\text{EO}(e, \cdot, \pi)$ calls during the execution of $\text{VO}(v, \pi)$. Moreover, let $Q(e, \pi)$ be the number of $\text{EO}(e, \cdot, \pi)$ calls starting from any vertex. In other words, we have that $Q(e, \pi) = \sum_{v \in V} Q(e, v, \pi)$.

▶ **Observation 13.** *For every edge e and permutation π, $Q(e, \pi) \leq O(n^2)$.*

Proof. Let $e = \{x, y\}$. For a fixed vertex u, either the vertex oracle $\text{VO}(u, \pi)$ queries the edge oracle for e directly, or through some incident edge e'. Hence, the edge oracle of e is called through at most $(K+2)(\deg(x)-1)+(K+2)(\deg(y)-1)$ of its incident edges ($K+2$ appears since each edge has $K+2$ copies), which implies that $Q(e, u, \pi) \leq (2K+4)(n-1)+1$. Therefore,

$$Q(e, \pi) \leq \sum_{u \in V} Q(e, u, \pi) \leq n\left((2K+4)(n-1)+1\right) \leq O(n^2).$$ ◀

The main contribution of this section is to show that the expected number of $\text{EO}(e, \pi)$ calls over all permutations π is $O(\log^2 n)$, which is formalized in the following lemma.

▶ **Lemma 14.** *For any edge $e \in E'$, we have $\mathbf{E}_{\pi}[Q(e, \cdot, \pi)] = O(\log^2 n)$.*

Assuming the correctness of Lemma 14, we can complete the proof of Theorem 12.

Proof of Theorem 12.

$$\begin{aligned}
\mathbf{E}_{v,\pi}[T(v,\pi)] = \frac{1}{n}\,\mathbf{E}_{\pi}\Big[\sum_{v\in V} T(v,\pi)\Big] &= \frac{1}{n}\,\mathbf{E}_{\pi}\Big[\sum_{v\in V}\sum_{e\in E'} Q(e,v,\pi)\Big] \\
&= \frac{1}{n}\,\mathbf{E}_{\pi}\Big[\sum_{e\in E'}\sum_{v\in V} Q(e,v,\pi)\Big] = \frac{1}{n}\,\mathbf{E}_{\pi}\Big[\sum_{e\in E'} Q(e,\pi)\Big] \\
&= \frac{1}{n}\sum_{e\in E'}\mathbf{E}_{\pi}[Q(e,\pi)] = \frac{1}{n}\sum_{e\in E'} O(\log^2 n) \\
&= \frac{1}{n}O(|E'|\cdot\log^2 n) = O(\bar{d}\cdot\log^2 n).
\end{aligned}$$

◀

We defer the proof of Lemma 14 to the arXiv version of the paper due to the page limit.

6 Our Estimator for Maximum Path Cover

In this section, we use the oracle of the previous section to estimate the number of edges in the output of Algorithm 2. In Section 5, we provide a lower bound on the number of recursive calls to our local query process. Note that this bound does not necessarily imply the same running time algorithm. For example, if we generate the whole permutation over all copies of edges before running the algorithm, it takes $\Theta(m)$ which is no longer sublinear. Using by now standard ideas of the literature, we show how we can implement the query process in almost the same running time (multiplied by a polylogarithmic factor) which is formalized in the following lemma (see the arXiv version of the paper).

▶ **Lemma 15.** *There exists a data structure that given a graph G in the adjacency list format, (implicitly) fixes a random permutation π over its edges. Then for any vertex v, the data structure returns the degree of vertex v in the subgraph P produced by Algorithm 2 according to a random permutation π. Each query v to the data structure is answered in $\tilde{O}(T(v,\pi))$ time w.h.p. where $T(v,\pi)$ is as defined in Section 5.*

Note that in our local query process, we need access to the adjacency list of weight-one edges. So the challenge that arises here is how to estimate the size of the output of Algorithm 2 in the adjacency matrix model. We present a reduction from adjacency matrix to adjacency list that appeared in the literature [2]. In this reduction, each query to the adjacency list can be implemented with $O(1)$ queries to the adjacency matrix and still we are able to estimate the maximum path cover with some additive error.

Let $\gamma = 16Kn$. We construct a graph $\hat{G} = (V_{\hat{G}}, E_{\hat{G}})$ for weight-one edges of graph G as follows:

- $V_{\hat{G}}$ is the union of V_1, V_2 and $U_1, U_2, \ldots, U_n$ such that:
  - V_1 and V_2 are two copies of the vertex set of the original graph G.
  - U_i is a vertex set of size γ for each $i \in [n]$.
- We define the edge set such that degree of each vertex is in $\{1, n, n+\gamma\}$:
  - Degree of each vertex $v \in V_1$ is n. The i-th neighbor of v is the i-th vertex in V_1 if $(v,i) \in E$, otherwise its i-th neighbor is the i-th vertex in V_2 for $i \le n$. Note that graph $(V_1, E_H \cap (V_1 \times V_1))$ is isomorphic to G.
  - Degree of each vertex $v \in V_2$ is $n+\gamma$. The i-th neighbor of v is the i-th vertex in V_2 if $(v,i) \in E$, otherwise, its i-th neighbor is the i-th vertex in V_1 for $i \le n$. For all $n < i \le n+\gamma$, the i-th neighbor of v is i-th vertex in U_v.
  - Degree of each vertex $u \in U_i$ is one for $i \in [n]$. The only neighbor of u is the i-th vertex of V_2.

By the construction of $\hat{G}$, the only neighbor of $v \in \bigcup_{i=1}^{n} U_i$ can be determined without any query to the adjacency matrix. Also, the i-th neighbor of each vertex in $V_1 \cup V_2$ can be determined with one query.

▶ **Observation 16.** *For each vertex $v \in V_{\hat{G}}$ and $i \in [\deg_{\hat{G}}(v)]$, the i-th neighbor of vertex v can be determined using at most one query to the adjacency matrix.*

Fix a vertex $v \in V_2$. When we run Algorithm 2, intuitively with high probability the first edge that is incident to v and occupies port v^0 is between v and $u \in U_v$. Furthermore, with high probability, the first two edges that are incident to v and occupies port v^1 are between v and $u \in U_v$. A vertex $v \in V_2$ is an *abnormal* vertex if the above properties do not hold for v. Let $R \in V_2$ be the set of abnormal vertices. We can show that there are few abnormal vertices exist in V_2, which implies that most of the incident edges to vertices of V_1 in the output of Algorithm 2 are in $\hat{G}[V_1]$ (only those between V_1 and R violate this property). Therefore, a natural way to estimate the number of edges in the output of Algorithm 2 on G, is to estimate the number of edges in $\hat{G}[V_1]$ in the output of Algorithm 2 on $\hat{G}$.

Algorithm 5 Final algorithm for maximum path cover.

1 Let $\hat{G} = (V_{\hat{G}}, E_{\hat{G}})$ as described above.
2 $r \leftarrow 192 \cdot K^2 \cdot \log n$.
3 Sample r vertices $u_1, u_2, \ldots, u_r$ uniformly at random from V_1 with replacement.
4 Sample r ports $p_1, p_2, \ldots, p_r$ uniformly at random from $\{0, 1\}$.
5 Run vertex oracle for each u_i and let X_i be the indicator if port $u_i^{p_i}$ is occupied with an edge in $\hat{G}[V_1]$ in output of Algorithm 2.
6 Let $X = \sum_{i \in [r]} X_i$ and $f = X/r$.
7 Let $\tilde{\rho} = \frac{K}{2(K+2)} \cdot (f \cdot n - \frac{n}{4K})$.
8 **return** $\tilde{\rho}$

▶ **Theorem 17.** *Given an adjacency matrix access for input graph G, there exists a randomized algorithm that w.h.p. runs in $\widetilde{O}(n)$ time and produces an estimate $\tilde{\rho}$, such that*

$$\left(\frac{1}{2} - \varepsilon\right) \cdot \rho(G) - \varepsilon n \leq \tilde{\rho} \leq \rho(G).$$

You can find the full version of this section with formal proofs in the arXiv version of the paper.

7 Our Estimator for (1,2)-TSP

In this section, we use the algorithm for estimating the size of maximum path cover as a black box to estimate the size of (1,2)-TSP. First, note that if there is no Hamiltonian cycle with weight one edges of the graph, then the set of weight-one edges of the graph (1,2)-TSP is a solution for maximum path cover of graph G. Also, in the case that there exists a Hamiltonian cycle, then the size of maximum path cover is $n - 1$. Moreover, if P^* is the maximum path cover of a graph G, then it is possible to create a TSP by connecting these paths using edges with weight two. This intuition helps to formalize the following observation.

▶ **Observation 18.** *Let $\tau(V)$ be the cost of (1,2)-TSP of graph $G = (V, E)$. Then, we have*

$$2n - \rho(G) - 1 \leq \tau(V) \leq 2n - \rho(G).$$

Now we are ready to present the final algorithm for estimating (1,2)-TSP.

Algorithm 6 Final algorithm for (1,2)-TSP.

1 Construct $\hat{G} = (V_{\hat{G}}, E_{\hat{G}})$ implicitly as desribed in Section 6.
2 Let $\tilde{\rho}$ be the output of Algorithm 5 on $\hat{G}$.
3 $\tilde{\tau} = 2n - \tilde{\rho}$
4 **return** $\tilde{\tau}$

▶ **Theorem 19.** *Let $\tau(V)$ be the cost of (1,2)-TSP of graph $G = (V, E)$. For any $\varepsilon > 0$, there exists an algorithm that estimate the cost of (1,2)-TSP, $\tilde{\tau}$, such that*

$$\tau(V) \leq \tilde{\tau} \leq (\frac{3}{2} + \varepsilon) \cdot \tau(V),$$

w.h.p in $\tilde{O}(n)$ running time.

You can find the full version of this section with formal proofs in the arXiv version of the paper.

8 Our Estimator for Graphic TSP

In this section, we use our algorithm for estimating the size of maximum path cover to estimate the size of graphic TSP. In a recent work, Chen et al. [10] showed that it is possible to obtain a (27/14)-approximate algorithm for graphic TSP by estimating the matching size and the number of biconnected components in the graph. Since the size of graphic TSP is at most $2n$ (the cost of MST is $n - 1$), they proved that if a graph has large matching and a few biconnected components, the cost of graphic TSP is significantly lower than $2n$. Since estimating the number of biconnected components is not an easy task in sublinear time, they use a proxy quantity that can be estimated in sublinear time.

In the full version of the paper, we show that if we use our estimator for maximum path cover as a black-box instead of matching estimator in algorithm of [10], we can improve the approximation ratio to 19/10. Moreover, we show that we can estimate the number of bridges in $\tilde{O}(n)$. We exploit this estimation for further improvement to get a 11/6-approximate algorithm for graphic TSP. We use the following algorithm for estimating the size of graphic TSP.

Algorithm 7 Improved algorithm for graphic TSP.

1 Construct $\hat{G} = (V_{\hat{G}}, E_{\hat{G}})$ implicitly as described in Section 6.
2 Let $\tilde{\rho}$ be the output of Algorithm 5 on $\hat{G}$.
3 Let $\tilde{B}$ be the estimate of the number of bridges.
4 $\tilde{\tau} = 2n - \frac{1}{3}(\tilde{\rho} - \tilde{B})$
5 **return** $\tilde{\tau}$

▶ **Theorem 20.** *Let $\tau(V)$ be the cost of graphic TSP of graph $G = (V, E)$. For any $\varepsilon > 0$, there exists an algorithm that estimate the cost of graphic TSP, $\tilde{\tau}$, such that*

$$\tau(V) \leq \tilde{\tau} \leq (\frac{11}{6} + \varepsilon) \cdot \tau(V),$$

w.h.p in $\tilde{O}(n)$ running time.

You can find the full version of this section with formal proofs in the arXiv version of the paper.

References

1. Anna Adamaszek, Matthias Mnich, and Katarzyna Paluch. New approximation algorithms for (1, 2)-tsp. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 9:1–9:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.9.
2. Soheil Behnezhad. Time-Optimal Sublinear Algorithms for Matching and Vertex Cover. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 873–884. IEEE, 2021. doi:10.1109/FOCS52979.2021.00089.
3. Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Local computation algorithms for maximum matching: New lower bounds. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 2322–2335. IEEE, 2023. doi:10.1109/FOCS57990.2023.00143.
4. Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Sublinear time algorithms and complexity of approximate maximum matching. *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 267–280, 2023. doi:10.1145/3564246.3585231.
5. Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Approximating maximum matching requires almost quadratic time. *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, Canada, To Appear*, 2024.
6. Soheil Behnezhad, Mohammad Roghani, Aviad Rubinstein, and Amin Saberi. Beating greedy matching in sublinear time. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3900–3945. SIAM, 2023. doi:10.1137/1.9781611977554.CH151.
7. Piotr Berman and Marek Karpinski. 8/7-approximation algorithm for (1, 2)-tsp. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 641–648. ACM Press, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109627.
8. Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic (1+ε)-approximate matching size in truly sublinear update time. *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 1563–1588, 2023. doi:10.1109/FOCS57990.2023.00095.
9. Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Sublinear algorithms for (1.5+ε)-approximate matching. *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 254–266, 2023. doi:10.1145/3564246.3585252.
10. Yu Chen, Sampath Kannan, and Sanjeev Khanna. Sublinear Algorithms and Lower Bounds for Metric TSP Cost Estimation. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:19, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
11. Yu Chen, Sanjeev Khanna, and Zihan Tan. Sublinear algorithms and lower bounds for estimating MST and TSP cost in general metrics. *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, 261:37:1–37:16, 2023. doi:10.4230/LIPICS.ICALP.2023.37.
12. Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

13 Artur Czumaj and Christian Sohler. Estimating the weight of metric minimum spanning trees in sublinear-time. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 175–183, 2004. doi:10.1145/1007352.1007386.

14 Artur Czumaj and Christian Sohler. Estimating the weight of metric minimum spanning trees in sublinear time. *SIAM J. Comput.*, 39(3):904–922, 2009. doi:10.1137/060672121.

15 Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 550–559. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.80.

16 Anupam Gupta and Krzysztof Onak. Sublinear.info Open Problem 71: Metric TSP Cost Approximation, 2016. Available at https://sublinear.info/index.php?title=Open_Problems:71.

17 Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric TSP. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 32–45. ACM, 2021. doi:10.1145/3406325.3451009.

18 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

19 Marek Karpinski and Richard Schmied. On approximation lower bounds for TSP with bounded metrics. *Electron. Colloquium Comput. Complex.*, page 8, 2012. arXiv:TR12-008.

20 D. König. Über graphen und ihre anwendung auf determinantentheorie und mengenlehre. *Mathematische Annalen*, 77:453–465, 1916. URL: http://eudml.org/doc/158740.

21 Matthias Mnich and Tobias Mömke. Improved integrality gap upper bounds for traveling salesperson problems with distances one and two. *Eur. J. Oper. Res.*, 266(2):436–457, 2018. doi:10.1016/J.EJOR.2017.09.036.

22 Tobias Mömke and Ola Svensson. Approximating graphic TSP by matchings. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 560–569. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.56.

23 Marcin Mucha. 13/9-approximation for graphic TSP. In *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 30–41. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPICS.STACS.2012.30.

24 Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Math. Oper. Res.*, 18(1):1–11, 1993. doi:10.1287/MOOR.18.1.1.

25 András Sebö and Jens Vygen. Shorter tours by nicer ears: 7/5-approximation for the graph-tsp, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. *Comb.*, 34(5):597–629, 2014. doi:10.1007/S00493-014-2960-3.

26 Anatoliy I Serdyukov. O nekotorykh ekstremal'nykh obkhodakh v grafakh. *Upravlyayemyye sistemy*, 17:76–79, 1978.

27 Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. An improved constant-time approximation algorithm for maximum matchings. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 225–234. ACM, 2009. doi:10.1145/1536414.1536447.

Better Space-Time-Robustness Trade-Offs for Set Reconciliation

Djamal Belazzougui
CAPA, DTISI, Centre de Recherche sur l'Information Scientifique et Technique, Algiers, Algeria

Gregory Kucherov
LIGM, CNRS, Université Gustave Eiffel, Marne-la-Vallée, France

Stefan Walzer
Karlsruhe Institute of Technology, Germany

Abstract

We consider the problem of reconstructing the symmetric difference between similar sets from their representations (sketches) of size linear in the number of differences. Exact solutions to this problem are based on error-correcting coding techniques and suffer from a large decoding time. Existing probabilistic solutions based on Invertible Bloom Lookup Tables (IBLTs) are time-efficient but offer insufficient success guarantees for many applications. Here we propose a tunable trade-off between the two approaches combining the efficiency of IBLTs with exponentially decreasing failure probability. The proof relies on a refined analysis of IBLTs proposed in (Bæk Tejs Houen et al. SOSA 2023) which has an independent interest. We also propose a modification of our algorithm that enables telling apart the elements of each set in the symmetric difference.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Randomness, geometry and discrete structures; Theory of computation → Sketching and sampling; Theory of computation → Error-correcting codes

Keywords and phrases data structures, hashing, set reconciliation, invertible Bloom lookup tables, random hypergraphs, BCH codes

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.20

Category Track A: Algorithms, Complexity and Games

Related Version *Preliminary version*: `https://arxiv.org/abs/2404.09607`

1 Introduction

The problem of *set reconciliation* (or *database reconciliation* [12]) lies at the intersection of several lines of algorithmic research. One intuitive way to define it is in terms of communication protocols: assume that two parties Alice and Bob hold similar sets S and T, respectively, that they seek to reconcile (i.e. identify differences) using low communication overhead. Rather than exchanging representations of entire sets, they exchange their small *sketches* holding enough information to identify differences of S and T provided that their number does not exceed a pre-defined parameter D.

In this paper, we study the version of the problem where the differences $S \triangle T := (S \setminus T) \cup (T \setminus S)$ should be recovered from the sketches of S and T rather than from the sketch of one of them and the entire other set. This opens a way to the setting, relevant to some applications, when only sketches of sets are stored in a database, and for each pair of sets, items proper to one of them can be retrieved from the corresponding sketches. As an example, consider a vast database of highly similar genomic sequences, such as those of SARS-Cov-2 genomes[1], that are stored in the form of sketches, rather than raw sequences, still supporting retrieval of differences between any two of them. Beyond this example, set reconciliation

[1] More than 16M SARS-Cov-2 are available in GISAID database at the time of writing, which typically differ one from another by just a few characters, over about 30,000 characters of length.

problem occurs in many other scenarios in distributed systems, where information needs to be synchronized between computational units. Those include blockchain systems, P2P systems, ad-hoc networks, synchronizing information across devices or data centers, and others.

The set reconciliation problem is also raised in the streaming framework. It is a common and natural requirement that the sketch can be efficiently updated when a new key is added to or deleted from the set. For example, [6] considers the *straggler identification* problem defined on a stream of insertions and deletions of keys modeling a stream of people entering and leaving a building. In this model, the sketch summarizes information about people currently in the building and if their number does not exceed D, is capable of listing them.

1.1 Prior work

Solutions to set reconciliation can be categorized into probabilistic and exact ones (although there exists an interplay between these two classes of algorithms) [25]. Probabilistic algorithms are based on Invertible Bloom Lookup Tables (IBLTs), also known as Invertible Bloom Filters, based on random hash functions. Originally proposed in [6] where they were applied to the problem of *subset reconciliation* (set reconciliation assuming $S \subseteq T$), they (in a slightly different version) were more systematically studied in [12]. Earlier a related data structure, called *k-set data structure*, was proposed in [10]. Set reconciliation by computing a difference of IBLTs (called *difference digest*) was studied in [7]. Work [21] further applies IBLTs to *multi-party* set reconciliation. The idea of "subtracting" IBLTs has also been applied in [9], where the emphasis is to store in an IBLT carefully defined hash values obtained using cryptography techniques. Very recent work [1] studies a simplified version of IBLTs that also applies to set reconciliation. We rely on this construction in the present paper.

An exact solution of the set reconciliation problem uses algebraic techniques, in particular *error-correcting codes* [7, 3, 4]. Work [20] proposes an exact solution based on characteristic polynomials to both subset and set reconciliation problems using $(D+1)\log U$ bits of space (i.e. essentially the space needed to store D differences). [11] studies the straggler identification problem with multiplicities and proposes an $\mathcal{O}(D\log(Um))$-bit solution based on polynomials over finite field, where each key occurs at most m times. A somewhat similar solution to subset reconciliation was proposed in [6], based on Newton polynomials. [3] surveys code-theoretic techniques for space recovery many of which apply to set reconciliation as well. Set reconciliation with BCH codes has been implemented in Minsketch software [19], leveraging an efficient syndrome decoding algorithm for BCH codes [5]. [4] proposes a method for sparse recovery based on expander codes; the construction of [4] can be interpreted as a table somewhat similar to IBLT but with a different decoding mechanism. Recent papers [23, 2] apply algebraic techniques to construct specific "hash functions" that guarantee successful decoding for IBLTs. The general downside of exact solutions is a larger decoding time, typically growing at least quadratically in D and relying on finite field arithmetic. Note also that the relationship between codes and set reconciliation is two-way: [22] proposes a construction of codes based on set reconciliation with IBLTs.

1.2 Our contribution: overview

The IBLT solution to set reconciliation is very efficient but provides a poor success guarantee. For example, if we have to compute similarity joins by performing all pairwise reconciliations of the objects in our database, then a significant fraction of comparisons may fail. On the other hand, techniques based on error-correcting codes are exact but have a high cost of

decoding. Here we propose a family of solutions to set reconciliation offering a trade-off between these two approaches, that is having significantly smaller error rate compared to IBLT at the price of an asymptotically vanishing increase of space and time bounds.

Our solution is based on the space-efficient IBLT from [1] complemented by an additional *stash* data structure supporting recovery in case of failure of the main IBLT. The analysis of [1] provides no guarantee in the case when the IBLT decoding fails. Therefore, here we enhance the analysis of [1] and prove a probability bound on the number of missing and extraneous elements reported by the data structure, in the event that the correct decoding fails. This is a key argument of our construction and its proof constitutes the main technical contribution of this paper.

The algorithm we obtain is parametrized and provides a tunable trade-off between the failure probability and space and time consumption. In the table below, we provide an overview of how our solution relates to the existing approaches surveyed in Section 1.1.

Table 1 Comparison of main techniques of set reconciliation to the results of the present work. r is a parameter assumed to verify $r = \min\left(\mathcal{O}(D/\log^2 U), \mathcal{O}(\log U)\right)$. For our algorithm, decoding time given is expected.

Method	Sketch size (bits)	Insertion time	Decoding time	Failure probability
IBLT [1]	$(c_3+\epsilon)D\log U$	$\mathcal{O}(1)$	$\mathcal{O}(D)$	$\mathcal{O}(D^{-1})$
IBLT with t-bit hashsum field	$(c_3+\epsilon)D(\log U + t)$	$\mathcal{O}(1+t/\log U)$	$\mathcal{O}(D(1+t/\log U))$	$\min(\mathcal{O}(D^{-1}), \mathcal{O}(D/2^t))$
BCH	$D\log U$	$\mathcal{O}(D\log U)$	$\mathcal{O}(D^2\log^2 U)$	0
expander code [4]	$\mathcal{O}(D\log^2 U)$	$\mathcal{O}(\log U)$	$\mathcal{O}(D\log U)$	0
this paper	$(c_3+\epsilon)D\log U + r\log U$	$\mathcal{O}(r\log U)$	$\mathcal{O}(D)$	$2^{-\Omega(r)}$

The paper is organized as follows. In Section 2, we introduce our version of IBLT, which is a slightly modified set sketch from [1]. This data structure is very space-efficient, however its compactness has a price: the decoding process can "go wrong" triggering undesirable *anomalous* steps [1]. In Section 3.2, we provide a refined analysis of anomalies and prove the following fundamental property: if the sketch stores a set S, the decoding produces a set S_{dec} with $|S \triangle S_{dec}| \leq r$, with a failure probability of $2^{-\Omega(r)}$. Section 4 introduces a solution to set reconciliation based on BCH error-correcting codes. Relying on that result, in Section 5, we extend the IBLT with a backup data structure (stash) to obtain an efficient solution to set reconciliation. The algorithm resorts to the stash when the decoding with the main IBLT is not completed, leaving out a small number of keys. The stash is defined using error-correcting codes. In Section 6, we discuss how to extend our algorithm in order to identify the original set of the keys of the set difference.

2 Set reconciliation with Invertible Bloom Lookup Tables

Definition of IBLT. An Invertible Bloom Lookup Table (IBLT) is an array $A[1:n]$ equipped with k random hash functions $h_1, \ldots, h_k : \mathcal{U} \to [n]$, from the key universe $\mathcal{U}$ to $[n] = \{1, \ldots, n\}$. We assume h_i's to be fully random and denote $h(x) = \{h_1(x), \ldots, h_k(x)\}$.

Several variants of IBLTs have been considered, which differ in how entries $A[i]$ are configured. In all of them, $A[i]$ contains a keysum field that holds either bitwise XOR [12, 7, 21, 1] or arithmetic sum [6, 12] of all current keys hashed to i. XOR provides a more elegant and space-efficient option whereas arithmetic sum becomes necessary when multi-sets are considered. In this work, the keysum field is defined with XOR and is set to contain $\log U$ bits ($U = |\mathcal{U}|$).

Algorithm `initialise`**:**
$A[1:n] = (0, \dots, 0)$

Algorithm `toggle`(x)**:**
for $i \in h(x)$ **do**
$A[i] \leftarrow A[i] \oplus x$

Algorithm `merge`$(A[1:n], A'[1:n])$**:**
for $i \in [n]$ **do**
$B[i] \leftarrow A[i] \oplus A'[i]$
return B

Algorithm `looksPure`$(i \in [n])$**:**
return $A[i] \neq 0 \wedge i \in h(A[i])$

Algorithm `decode`**:**
$S_{\text{dec}} \leftarrow \varnothing$
$Q \leftarrow \{i \in [n] \mid \mathsf{looksPure}(i)\}$
$(t, t_{\max}) \leftarrow (0, 2n)$ // time limit $2n$
while $Q \neq \varnothing$ **do**
$Q_{\text{next}} \leftarrow \varnothing$
for $i \in Q$ **if** $\mathsf{looksPure}(i)$ **do**
$x \leftarrow A[i]$ // *detected* key x
$\mathsf{toggle}(x)$
$S_{\text{dec}} \leftarrow S_{\text{dec}} \triangle \{x\}$
$Q_{\text{next}} \leftarrow Q_{\text{next}}$
$\cup \{i \in h(x) \mid \mathsf{looksPure}(i)\}$
$t \leftarrow t + 1$
if $t \geq t_{\max}$ **then**
return S_{dec}
$Q \leftarrow Q_{\text{next}}$
return S_{dec}

Figure 1 IBLT implementation from [1] with added time limit in decode.

Besides the keysum, $A[i]$ can include a hashsum field that holds a hash sum of the keys in $A[i]$ and/or a counter field that tracks the current number of keys in $A[i]$. These are used for enforcing proper functioning and integrity of the data structure at the price of using additional space. In this work, we use the most compact IBLT configuration with three hash functions ($k = 3$) and $A[i]$ storing the keysum field only. This variant was introduced and analysed in recent paper [1] that we rely on in this work. Later in Section 6 we consider an extension introducing a restricted counter field.

Our IBLT implementation is defined in Figure 1. Initially, all entries $A[i]$ are set to zero with `initialise`. Both insertion and deletion of a key x is done by performing `toggle`(x).

Decoding the keys stored in an IBLT resembles the *peeling* process of the k-hypergraph, where n IBLT entries correspond to hypergraph vertices and each stored key corresponds to a hyperedge defined as the set of entries the key is hashed to. Peeling a hypergraph iterates the following operation: for any vertex of degree 1, delete the incident hyperedge. If the peeling results in the empty graph with no hyperedges, the input graph is called *peelable*; otherwise the process yields the largest subhypergraph with every vertex of degree at least 2, which is called the 2-*core* (hereafter simply *core*). It is known that peelability of random k-hypergraphs ($k \geq 3$) with n vertices and m hyperedges undergoes a phase transition: if $n > c_k m$, then the hypergraph is peelable with high probability (hereafter, *whp*), whereas if $n < c_k m$, the hypergraph is not peelable whp. Here constant c_k is the peelability threshold, in particular $c_3 = 1.22179\dots$ is the smallest of c_k for $k \geq 3$. The following results is shown in [12].

▶ **Theorem 1.** *Whenever $n > c_k m$, a random k-hypergraph is peelable except with probability $\mathcal{O}(1/n^{k-2})$.*

Set reconciliation using difference IBLT. Consider two sets S and T stored in IBLTs $A_S[1:n]$ and $A_T[1:n]$ respectively, using the same hash functions. If S and T have a bounded symmetric difference, that is $|S \Delta T| \leq D$, and $n > c_k D$, then the keys of $S \Delta T$ can

be recovered whp from the *difference IBLT* $A_{ST}[1:n] = \texttt{merge}(A_S, A_T)$. It is immediate to see that common keys of S and T are canceled out and A_{ST} stores exactly the keys $S \Delta T$. Thus, the difference IBLT can solve set reconciliation except with probability $\mathcal{O}(1/D)$.

Note that A_{ST} does not allow distinguishing keys of $S \setminus T$ and $T \setminus S$; below in Section 6 we will extend the construction to make this possible. Note also that for the reconciliation to be possible, the size n of IBLTs only depends on the size of the symmetric difference and does not depend on the sizes of S and T.

3 Improved Guarantees for IBLTs

Consider the IBLT implementation as given in Figure 1. It uses only the keysum field in its buckets and was studied, modulo slight changes stated below, in [1] under the name "simple set sketch". In this section we enhance the analysis given in [1] (reproduced as (i) in Theorem 2 below) by also probabilistically bounding the magnitude of the error of decode in its failure cases (given as (ii)). Note that we restrict our attention to three hash functions ($k = 3$) as this yields the best threshold value.[2]

▶ **Theorem 2.** *Let $\varepsilon > 0$ and $n > (c_3 + \varepsilon)|S|$. Let S_{dec} be the set returned by decode.*
(i) $\Pr[S_{\text{dec}} = S] = 1 - \widetilde{\mathcal{O}}(1/n)$.
(ii) *For any $r = o(n)$ we have* $\Pr[|S \triangle S_{\text{dec}}| > r] = 2^{-\Omega(r)}$.

Changes made to decode. Our version of decode differs from that in [1] in two minor ways. First, we have introduced a *time limit* of $2n$ to the number of peeling steps that are performed and return the current state of S_{dec} when the time limit is reached[3]. The result from [1] still applies with this change, since the authors proved that decode terminates in $n + \text{poly}\log(n)$ peeling steps with $S_{\text{dec}} = S$ with probability $1 - \mathcal{O}(1/n)$. Second, we insist that Q and Q_{next} are implemented as set data structures, e.g. using hash tables, whereas [1] pointed out that Q and Q_{next} can be implemented as multiset data structures, e.g. using FIFO queues.[4]

It should be clear that our version of decode can be implemented such that it has running time $\mathcal{O}(n)$, both in expectation and with high probability.

3.1 The issue of anomalies

Decode checks if the ith *bucket* of A contains a single key using the function $\texttt{looksPure}(i)$ that checks if $h_j(A[i]) = i$ for some $j \in \{1, 2, 3\}$. Importantly, $\texttt{looksPure}(i)$ may produce a false positive result when $A[i] = x_1 \oplus x_2 \oplus \ldots \oplus x_{\ell-1}$ is the sum of several keys hashing to i and $y = A[i]$ also happens to hash to i, i.e. $h_\sigma(y) = i$ for some $\sigma \in \{1, 2, 3\}$. Such an occurence $\{x_1, \ldots, x_{\ell-1}, y\}$ is called an *anomaly* (of size ℓ) in [1] and causes the key y to be toggled and thus inserted into the data structure. Anomalies corrupt the decoding process and may cause decode to fail or even to run in an infinite loop if no time limit is set [1].

[2] In [1] any $k \geq 3$ was considered and our analysis here could likewise be extended.
[3] In practice a dynamic time limit could be considered. For instance: Abort if a round of decode only toggles keys that have already been toggled in a previous round.
[4] At no point did the analysis from [1] actually *rely* on Q or Q_{next} being multisets. The motivation was merely that FIFO queues are the more lightweight data structure. We revert this simplification here because an argument from [1] showing that the impact of duplicated elements is negligible no longer applies. In fact, with a FIFO implementation of Q and Q_{next} we could see 2^R copies of the same element in Q in the Rth iteration of the while loop with small but non-negligible probability $n^{-\mathcal{O}(1)}$.

Dealing with anomalies is a challenging issue. In [1], it was proved that certain kinds of anomalies and certain interactions of several anomalies do not occur except with probability $\tilde{O}(1/n)$. Since $\mathcal{O}(1/n)$ is the probability for the existence of a non-empty core (see Theorem 1 for $k = 3$), which also causes decoding to fail, anomalies do not increase the failure probability significantly.

In this section we will go further, proving that *even in the unlikely cases where decoding produces an incorrect result*, e.g. due to problematic anomalies, the magnitude of the error is likely to be small.

3.2 Proof Outline

We quantify the effect of anomalies by considering two sets. First, the set F of all *foreign keys*, which are keys from $\mathcal{U} \setminus S$ that are added to the sketch at least once during the decoding process. Second, the set N of all centres of certain anomalies we call *internal anomalies* (defined in Section 3.4). Our proof of Theorem 2 rests on the following arguments.

- In Section 3.3 we bound the size of F. Intuitively, the heuristic looksPure is used $\mathcal{O}(n)$ times and fails each time with probability $\mathcal{O}(1/n)$. Each failure can cause one foreign key to be added, suggesting $\mathbb{E}[|F|] = \mathcal{O}(1)$. The precise argument, including a concentration bound for $|F|$, is more subtle since failures do not happen independently.
- In Section 3.4 we import a tail bound on $|N|$ from [1].
- In Section 3.5 we reframe the decoding process as the well-understood *peeling process* on a random hypergraph. Since peeling follows a local rule, the corruption due to anomalies spreads locally as well. This already implies that the majority of keys are correctly decoded if anomalies are rare.
- In Section 3.6 we prove a claim regarding the maximum hyperedge density of any subhypergraph of a given random hypergraph. Its intuitive role is to show that the size of the queue Q in late rounds of decode would always be, in the absense of anomalies, at least linear in the number of remaining keys. This implies that the anomalies only have the potential to stall progress in a significant way if they affect a number of buckets that is linear in the number of remaining keys.
- In Section 3.7 we stitch everything together: The effect of anomalies as quantified by $|F|$ and $|N|$ is probabilistically bounded. Peeling is likely to proceed largely unhindered in its early rounds. In its later rounds it proceeds until the number of remaining keys is linear in $|F| + |N|$.

3.3 Bounding Foreign Keys

Let $y \in \mathcal{U} \setminus S$ be any foreign key. In order for y to be added to the IBLT, y must occur as the sum of several keys in one of the buckets $h_1(y), h_2(y), h_3(y)$. Since $h_1(y), h_2(y), h_3(y)$ are uniformly random in $[n]$ and independent of the initial state of the data structure, decode would have to effectively "guess" a hash of y. We argue that decode is, in this sense, a player in a corresponding guessing game that, even if played optimally, rarely allows a player to guess a lot of hashes correctly.

A guessing game. Let $(X_{j,\sigma})_{j\in\mathbb{N},\sigma\in[3]}$ be a family of i.i.d. random variables with uniformly random values in $[n]$. Consider a game in which the player does not know the random variables. She may issue queries of the form $(i, j) \in [n] \times \mathbb{N}$ and is told whether or not $i \in \{X_{j,\sigma} \mid \sigma \in [3]\}$. If she gets a positive answer, we say she has *solved* group j and she is told the values $X_{j,1}$,$X_{j,2}$ and $X_{j,3}$.

▶ **Lemma 3.** *Assume a player is given* $6n$ *queries. She solves more than* r *groups with probability* $\mathcal{O}(2^{-r})$.

Proof. Consider a query $(i, j) \in [n] \times \mathbb{N}$ that is the kth query of the form $(\cdot, j)$ where j is an unsolved group. We distinguish two cases.

Case 1: $k \geq n/2$. We generously grant group j as solved. This affects at most $6n/(n/2) = 12$ groups.

Case 2: $k \leq n/2$. Up to $k-1$ values are already excluded for the variables $X_{j,1}$, $X_{j,2}$ and $X_{j,3}$. All other values are equally likely. The probability of solving group j with this query is at most $1 - (1 - \frac{1}{n-k+1})^3 < 1 - (1 - \frac{2}{n})^3 \leq \frac{6}{n}$.

Let Y be the number of successes due to Case 2. A simple coupling yields a random variable Z with $Y \leq Z$ and $Z \sim \mathrm{Bin}(6n, \frac{6}{n})$. We can think of Z as the sum of the outcomes of $6n$ Bernoulli trials. In order for Z to attain a value of at least $\ell = r - 12$ there must exist a set $T \subseteq [6n]$ of size ℓ such that the corresponding Bernoulli trials are all successes. A union bound over all choices of T yields:

$$\begin{aligned}\Pr[Y \geq \ell] \leq \Pr[Z \geq \ell] &\leq \binom{6n}{\ell}\left(\frac{6}{n}\right)^{\ell} \leq \left(\frac{6ne}{\ell}\right)^{\ell}\left(\frac{6}{n}\right)^{\ell} \\ &\leq \left(\frac{36e}{\ell}\right)^{\ell} = \left(\frac{36e}{r-12}\right)^{r-12} \leq 2^{-r+12},\end{aligned}$$

where the last step assumes $r \geq 72e + 12$.

Overall, if $r \geq 72e + 12$ then the number of successes is bounded by r, except with probability $\mathcal{O}(2^{-r})$. ◀

How this relates to the foreign keys. Let $F \subseteq \mathcal{U} \setminus S$ be the set of all foreign keys that are added to S_{dec} at least once during the execution of decode.

▶ **Lemma 4.** *For any $r \in \mathbb{N}$ we have $\Pr[|F| > r] = \mathcal{O}(2^{-r})$.*

Proof. Before decoding begins, the family $(h_\sigma(y))_{y \in \mathcal{U} \setminus S, \sigma \in [3]}$ of hashes of foreign keys are independent and uniformly random in $[n]$ just as is required in the guessing game above. We interpret decode as a player. Whenever decode executes the looksPure operation for a bucket i containing $A[i] = y \in \mathcal{U} \setminus S$, then we interpret this as the player issuing the query (y, i). He learns whether $i \in \{h_1(y), h_2(y), h_3(y)\}$ and, on a positive answer, he also learns the values $h_1(y), h_2(y), h_3(y)$. This is sufficient to continue the execution of decode, in particular decode amounts to a valid player respecting the access limitation to the relevant random variables.

The crucial observation is that any $y \in F$ is solved by decode. Indeed, if $y \in F$ then y was added to S_{dec} at some point during the execution of decode and this addition was immediately preceeded by a corresponding successful looksPure check.

The number q of queries issued by decode is by definition the number of calls to looksPure. Every time a bucket i is considered to be added to Q or Q_{next} there is one such call, and, if i is added, there can be another call when i is removed from Q. Due to the time limit we have $q \leq 2(n + t_{\max}) \leq 6n$. Therefore Lemma 3 implies our claim. ◀

3.4 Bounding Centres of Internal Anomalies

It is worthwhile to clarify the notion of anomalies by giving a precise definition of the set $\mathcal{A}$ of all anomalies.

$$\mathcal{A} = \left\{ \varnothing \neq A \subseteq \mathcal{U} \,\middle|\, \bigoplus_{x \in A} x = 0, \bigcap_{x \in A} h(x) \neq \varnothing \right\}$$

If $A = \{x_1, \ldots, x_\ell\}$ is an anomaly then the presence of any $\ell - 1$ keys from A can be mistaken as the presence of the missing ℓth key (because of $x_1 \oplus \ldots \oplus x_{\ell-1} = x_\ell$). A value $i \in \bigcap_{x \in A} h(x)$ is a centre of the anomaly A (in principle a single anomaly could have multiple centres). The set $\mathcal{A}$ makes no mention of S and contains many anomalies that are unlikely to become relevant during decoding. Immediately threatening is, however, the set of *native anomalies* $\mathcal{A}_{\text{nat}}$ define as

$$\mathcal{A}_{\text{nat}} = \{A \in \mathcal{A} \mid |A \setminus S| \leq 1\}.$$

For a native anomaly A, all but at most 1 key from A are present in the beginning, enough to cause trouble right away. In what follows, a subset of the native anomalies, namely the *internal* anomalies $\mathcal{A}_{\text{int}}$ and the set N of its centres play a role. They are defined as

$$\mathcal{A}_{\text{int}} = \{A \in \mathcal{A} \mid A \subseteq S\} \text{ and } N = \bigcup_{A \in \mathcal{A}_{\text{int}}} \bigcap_{x \in A} h(x).$$

▶ **Lemma 5.** *For any $r \in \mathbb{N}$ we have $\Pr[|N| > r] = \mathcal{O}(2^{-r})$.*

Proof. The authors of [1] show in their Lemma 2.2 that the set $N_{\text{nat}} = \bigcup_{A \in \mathcal{A}_{\text{nat}}} \bigcap_{x \in A} h(x)$ of centres of native anomalies satisfies $\Pr[N' \geq r] \leq \left(\frac{3e^4}{r}\right)^r$. For $r \geq 6e^4$ this is at most 2^{-r}. Since $N_{\text{nat}} \supseteq N$ the claim follows. ◀

3.5 Early Rounds of the Peeling Process

We will use a result by Molloy who analysed cores of random hypergraphs [24]. The hypergraph H arising in our setting has vertex set is $[n]$ and the m hyperedges $\{\{h_\sigma(x) \mid \sigma \in [k]\} \mid x \in S\}$. By the *parallel peeling* process we mean an algorithm on a hypergraph. In each round, all vertices of degree 0 or 1 are determined (simultaneously) and then all of these vertices are removed, including all incident hyperedges. When no vertex of degree 0 or 1 remains, the core of the hypergraph is reached, which may be empty. Note that decode implements this peeling process, except that anomalies may occur.

Early Peeling Without Anomalies. A lemma by Molloy is the following.

▶ **Lemma 6** ([24, Lemma 3])**.** *For any $\varepsilon, \delta > 0$, there exists $R \in \mathbb{N}$ such that after peeling a k-uniform hypergraph with $\frac{n}{m} > c_3 + \varepsilon$ for R rounds then at most δn vertices remain in expectation.*[5]

In the following we assume that constants $\varepsilon, \delta > 0$ are given and $R \in \mathbb{N}$ is the corresponding constant from Lemma 6.

By the R-neighbourhood of a vertex v or hyperedge e we mean the subhypergraph of H induced by all vertices that can be reached from v or e by traversing at most R hyperedges.

It is a well-known fact (and a consequence of the Poisson limit theorem) that the degree distribution of H converges to a Poisson distribution with parameter $3/c_3$ for $n \to \infty$. More generally, the distribution of the R-neighbourhoods of the vertices in H converges (see e.g. [15, 16]). This implies that we can choose a constant $D = D(\varepsilon, \delta, R)$ large enough such the probability that a given vertex has a vertex of degree at least D in its R-neighbourhood is at most δ.

[5] Molloy's Lemma is slightly stronger in that it claims that at most δn vertices remain with probability $1 - o(1)$. Since we need a much stronger concentration bound, we will redo a corresponding step ourselves.

In this context call a vertex v *good* if

(a) v is removed by the R-round peeling process

(b) the R-neighbourhood of v does not contain a vertex of degree at least D.

Since the first condition is met by at least $(1-\delta)n$ vertices in expectation by Lemma 6 and the second condition by at least $(1-\delta)n$ vertices in expectation by choice of D, the set G of good vertices has expectation at least $(1-2\delta)n$.

We will now use the bounded difference inequality by McDiarmid [18].

▶ **Lemma 7** (McDiarmid's inequality [18]). *Let $\mathcal{X}$ be some set, $d \in \mathbb{N}$ and $f : \mathcal{X}^n \to \mathbb{R}$ a function such that changing any one argument of f can affect the function value by at most d. Let $X_1, \dots, X_n$ be i.i.d. random variables with values in $\mathcal{X}$. Then*

$$\Pr[|f(X_1,\dots,X_n) - \mathbb{E}[f(X_1,\dots,X_n)]| > \varepsilon] \leq \exp(-\varepsilon^2/(nd^2)).$$

This gives us a concentration bound on the number of good vertices.

▶ **Lemma 8.** *In the above context there exists a constant $\gamma = \gamma(\varepsilon, \delta, R, D) > 0$ such that*

$$\Pr[|G| < (1-3\delta)n] \leq \exp(-\gamma n).$$

Proof. Note that whether or not a vertex is good is a *local* property in the sense that it only depends on its R-neighbourhood. This gives us a bounded difference property of $|G|$ as follows.

The set G (and therefore $|G|$) is a function of the $3m$ hash values defining the m hyperedges. Assume we change a single hash value from v to $v' \neq v$. A previously non-good vertex might become good only if it was in the vicinity of v and a previously good vertex might become non-good only if it ends up in the vicinity of v'. More precisely, a vertex w might be affected if prior to the change, v or v' (or both) were reachable from w via a path of length at most R that only traverses vertices of degree at most D (vertices reachable through vertices of higher degree are non-good anyway). To bound the number of such paths, note that in every step we can first choose among $\leq D$ incident hyperedges and then among $3-1$ endpoints of the chosen hyperedge, or we can choose to end the path at the current vertex. In particular, changing a single incidence affects the number of good vertices by at most $2(2D+1)^R$. We can therefore apply McDiarmid's inequality (Lemma 7) to conclude that

$$\begin{aligned}\Pr[n - |G| > 3\delta n] &\leq \Pr\left[(n-|G|) - \mathbb{E}[n-|G|] > \delta n\right] \\ &\leq \exp(-(\delta n)^2/(n \cdot (4(2D-1)^{2R})) = \exp(-\gamma n)\end{aligned}$$

when defining $\gamma = \delta^2/(4(2D+1)^{2R})$. ◀

Early Peeling With Anomalies. We now get back to the analysis of decode. Let S^R_{dec} be the state of S_{dec} after R rounds of the while-loop are executed (ignoring the time limit). Let $S^R := S^R_{\text{dec}} \triangle S$ be the set of elements stored in the IBLT at that time and $I^R := \bigcup_{x \in S^R} h(x)$ the set of buckets touched by S^R.

▶ **Lemma 9.** *For any $\varepsilon > 0$ and $\delta > 0$ there exist constants R and β such that*

$$\Pr[|I^R| > 7\delta n] \leq \mathcal{O}(2^{-\beta n}).$$

Proof. Given ε and δ, let $R = R(\varepsilon, \delta)$, $D = D(\varepsilon, \delta, R)$ and $\gamma = \gamma(\varepsilon, \delta, R, D)$, be the corresponding constants from the discussion of the discussion above. Recall from Lemmas 4, 5, and 8 that F is the set of foreign keys added to S at least once, N is the set of centres of internal anomalies and G is the set of good vertices. We know:

$$
\begin{array}{rll}
|F| \leq \frac{\delta n}{(2D+1)^R} & \text{except with probability } \mathcal{O}\left(2^{-\frac{\delta n}{(2D+1)^R}}\right) & \text{by Lemma 4} \\
|N| \leq \frac{\delta n}{(2D+1)^R} & \text{except with probability } \mathcal{O}\left(2^{-\frac{\delta n}{(2D+1)^R}}\right) & \text{by Lemma 5} \\
|G| \geq (1-3\delta)n & \text{except with probability } \exp(-\gamma n) & \text{by Lemma 8}
\end{array}
$$

The sum of the error probabilities is $\mathcal{O}(2^{-\beta n})$ where $\beta = \delta/(2D+1)^R + \gamma \log_2(e)$. Thus we may assume that all three stated events occur.

We now upper bound $|I^R|$. We pessimistically assume that each $i \in [n]$ that is not good is in I^R, which accounts for at most $3\delta n$ vertices. Any good $i \in [n]$ would be removed by the proper peeling process, so if $i \in I^R$ then an anomaly must have interfered. Therefore, the R-neighbourhood of i must include the centre of an internal anomaly or a vertex incident to a key from F. In other words, the R-neighbourhood of i must include a vertex from $|N \cup h(F)|$. At most $|N \cup h(F)| \cdot (2D+1)^R$ good vertices can be affected in this way, namely those reachable from a vertex in $N \cup h(F)$ via a path of length at most R traversing only vertices of degree at most D. Taken together the size of I_R is bounded by

$$
|I_R| \leq 3\delta n + |N \cup h(F)| \cdot (2D+1)^R \leq 3\delta n + \frac{4\delta}{(2D+1)^R} n \cdot (2D+1)^R \leq 7\delta n. \qquad \blacktriangleleft
$$

3.6 Late Rounds of the Peeling Process

▶ **Lemma 10.** *There exists a constant δ such that for any $1 \leq r \leq n$ there is an event E with probability $\mathcal{O}(2^{-r})$ such that, when E occurs, then*

$$
\forall I \subseteq [n] \text{ with } 5r \leq |I| \leq \delta n : |S[I]| \leq \tfrac{3}{5}|I|.
$$

Proof. We use a union bound over all possible sizes i of I and all possible sets of size i. Let p_i be the probability that a violating set of size i exists and $j := \frac{3}{5}i$. Multiplying the number $\binom{n}{i}$ of choices for I, the number $\binom{m}{j}$ of choices for $T \subseteq S$ of size j and the probability for $T \subseteq S[I]$ (i.e. the probability that all $x \in T$ satisfy $h(x) \subseteq I$) gives

$$
\begin{aligned}
p_i &\leq \binom{n}{i}\binom{m}{j}\left(\frac{i}{n}\right)^{3j} \leq \left(\frac{ne}{i}\right)^i \left(\frac{ne}{j}\right)^j \left(\frac{i}{n}\right)^{3j} \\
&= e^{i+j}\left(\frac{i}{j}\right)^j \left(\frac{i}{n}\right)^{-i+2j} \leq e^{3i}\left(\frac{i}{n}\right)^{i/5} = \left(\frac{e^{15}i}{n}\right)^{i/5}.
\end{aligned}
$$

Setting $\delta = 1/(2e^{15})$ we can bound the overall failure probability p as

$$
\begin{aligned}
p &\leq \sum_{i=5r}^{\delta n} p_i \leq \sum_{i=5r}^{\delta n} \left(\frac{e^{15}i}{n}\right)^{i/5} \leq \sum_{i=5r}^{\delta n} \left(\frac{e^{15}\cdot \delta n}{n}\right)^{i/5} \leq \sum_{i=5r}^{\delta n} (\tfrac{1}{2})^{i/5} \\
&= \sum_{i=5r}^{\delta n} \left((\tfrac{1}{2})^{1/5}\right)^i \leq 2^{-r} \cdot \sum_{i=0}^{\infty} \left((\tfrac{1}{2})^{1/5}\right)^i = 2^{-r} \cdot \Theta(1) = \mathcal{O}(2^{-r}).
\end{aligned}
\qquad \blacktriangleleft
$$

3.7 Proof of Theorem 2

We now assemble the previous Lemmas into a proof of Theorem 2. Note that we only claimed an error probability of $2^{-\Omega(r)}$, not $\mathcal{O}(2^{-r})$. By a change of parameters it therefore suffices that we show $\Pr[|S_{\text{dec}} \triangle S| > 185r] = \mathcal{O}(2^{-r})$.

The parameter $\varepsilon > 0$ and $r = o(n)$ are given. We use the absolute constant $\delta > 0$ from Lemma 10. We apply Lemma 9 for ε and $\delta' = \delta/7$ which gives us $R \in \mathbb{N}$ and $\beta > 0$. For convenience, let us summarise the statements used in the following. Since each holds with probability $1 - \mathcal{O}(2^{-r})$ we need only show that they imply $|S_{\text{dec}} \triangle S| \leq 185r$.

$$|I^R| < \delta n \quad \text{by Lemma 9} \tag{1}$$
$$\forall I \subseteq [n] \text{ with } 5r \le |I| \le \delta n : |S[I]| \le \tfrac{3}{5}|I| \quad \text{by Lemma 10} \tag{2}$$
$$|N| \le r \text{ and } |F| \le r \quad \text{by Lemmas 4 and 5} \tag{3}$$

Recall that *anomalous decoding steps* are those that add a key rather than removing a key.

▷ Claim 11. There are at most $|N| + 3|F| \le 4r$ anomalous decoding steps per round.

Proof. There are at most $3|F|$ buckets in which a key from F might be placed. We may conservatively assume that there is an anomalous decoding step in each round for each of these buckets. Anomalous decoding steps where no foreign key is involved can arise, by definition, only in centres of internal anomalies, hence there can be at most $|N|$ per round. ◁

We are most interested in the first $\hat{\rho}$ rounds of decode where $\hat{\rho} := R + \log_{\frac{20}{19}}(\frac{n}{300r}) = \Theta(\log \frac{n}{r})$. We now make sure that we are referring to well-defined rounds. Execution might end due to $Q = \varnothing$, or due to the time limit. If execution ends prior to round $\hat{\rho}$ due to $Q = \varnothing$, just pretend the loop is executed for a suitable number of additional rounds. Note that an additional round does not perform any steps when $Q = \varnothing$. Let us now deal with the time limit.

▷ Claim 12. The limit $t_{\max} = 2n$ on steps does not take effect within the first $\hat{\rho}$ rounds.

Proof. We use a potential argument. There are n keys in the beginning. Each non-anomalous decoding step removes a key and each anomalous decoding step adds a key. Hence, if a denotes the number of anomalous decoding steps in the first $\hat{\rho}$ rounds then the total number of decoding steps is at most $n + 2a$. The previous claim gives $a \le 4r\hat{\rho}$. We can therefore bound the number of decoding steps in the first $\hat{\rho}$ rounds by

$$n + 2a \le n + 8r\hat{\rho} = n(1 + \Theta(\tfrac{r}{n} \log \tfrac{n}{r})) = n(1 + o(1)) \le 2n.$$

The last step uses $r = o(n)$ and $\lim_{x \to 0} x \log \frac{1}{x} = 0$. ◁

We can now safely refer to each round $\rho \in \{1, \dots, \hat{\rho}\}$. Let S^ρ be the set of keys stored in the IBLT after ρ rounds, $I^\rho = h(S^\rho)$ the set of buckets touched by these keys and $S_0^\rho = S^\rho \setminus F$.

▷ Claim 13. If $300r \le |I^\rho| \le \delta n$ then $|I^{\rho+1}| \le \frac{19}{20}|I^\rho|$.

Proof. Since $S_0^\rho \subseteq S[I^\rho]$ we know $|S_0^\rho| \le \frac{3}{5}|I^\rho|$ from Equation (2). The number of incidences due to the keys from S_0^ρ is $3|S_0^\rho| \le \frac{9}{5}|I^\rho| = \frac{9}{10} \cdot 2|I^\rho|$. Thus at most a $\frac{9}{10}$-fraction of the buckets in I^ρ have two or more incidences to keys from S_0^ρ. This leaves at least $\frac{1}{10}|I^\rho|$ buckets with at most one incidence to S_0^ρ. If i is such a bucket, we would normally expect $i \notin I^{\rho+1}$.

There are only two exceptions. A foreign key might be stored in i at some point during round $\rho + 1$ or a key from S is added back into bucket i at some point during round $\rho + 1$. Each foreign key and each anomalous decoding step affects at most 3 buckets. Hence, by an earlier claim, at most $3(|F| + 4r) \le 15r$ buckets are exceptional in this sense. Together we get

$$|I^{\rho+1}| \le |I^\rho| - \tfrac{1}{10}|I^\rho| + 15r \le \tfrac{9}{10}|I^\rho| + 15r \le \tfrac{19}{20}|I^\rho|.$$

where the last step uses $|I^\rho| \ge 300r$. ◁

▷ Claim 14. If $|I^\rho| \leq 300r$ then $|I^{\rho+1}| \leq 300r$.

Proof. Assume w.l.o.g. that $|I^\rho| \geq 5r$ and modify the last step of the previous claim:

$$\tfrac{9}{10}|I^\rho| + 15r \leq \tfrac{9}{10} \cdot 300r + 15r = 285r < 300r. \qquad \triangleleft$$

Let now $\rho_{\max}$ be the last round that is fully executed.

▷ Claim 15. $|I^{\rho_{\max}}| \leq 300r$.

Proof. We have $|I^R| < \delta n$ by Equation (1). In subsequent rounds $\rho \in \{R+1, R+2, \dots\}$, the size of I^ρ decreases by at least a factor of $\frac{19}{20}$ until being at most $300r$ by Claim 13. By choice of $\hat{\rho}$ we have $I^\rho \leq 300r$ for some $\rho \leq \hat{\rho}$. By Claim 14 $|I^\rho|$ will subsequently not rise above $300r$ for larger values of ρ. $\triangleleft$

By the last claim and Equation (2) we have

$$|S_0^{\rho_{\max}}| \leq S[|I^{\rho_{\max}}|] \leq \tfrac{3}{5} \cdot (\max(5r, |I^{\rho_{\max}}|)) \leq \tfrac{3}{5} \cdot 300r = 180r.$$

The set $S^{\rho_{\max}}$ of all keys after round $\rho_{\max}$ might additionally include up to $|F| \leq r$ foreign keys. Since an additional round might be started but cut short due to the time limit, another $4r$ keys might be added due to anomalous decoding steps. Overall the set $S \triangle S_{\text{dec}}$ of keys returned in the end has size at most $180r + r + 4r = 185r$. This concludes the proof.

4 Set reconciliation with error-correcting codes

The set reconciliation problem can be modeled by encoding a set as a binary string of length U which is then sent through a channel inflicting up to D errors (bitflips). Recovering the errors is then equivalent to reporting the keys inserted to or deleted from the set. This reduction immediately implies that the problem can be solved by an appropriate application of error-correcting codes, in particular a linear code such as BCH, Reed-Solomon or expander code (see [14]). This reduction, in turn, highlights the relationship of set reconciliation to the problem of *sparse recovery*: reconstructing a sparse vector from a set of linear measurements (see e.g. [4]).

We will use a BCH code over the field $GF(2^w)$ with $U = 2^w - 1$. It is known (see e.g. [17]) that if up to D errors have to be corrected, the binary parity-check matrix $\mathcal{H}$ for BCH has dimensions $Dw \times U$. Let each set S be conceptually represented by a binary vector of size U. Denote $\mathcal{H}(S)$ the image of S by $\mathcal{H}$. Then for two sets S, T, $\mathcal{H}(S) \oplus \mathcal{H}(T) = \mathcal{H}(S \triangle T)$. That is, if $|S \triangle T| \leq D$, then $\mathcal{H}(S \triangle T)$, called *syndrome*, encodes the differences between S and T and can be reconstructed by a syndrome decoding algorithm. This implies that $\mathcal{H}(S)$ can be defined as a sketch of S of Dw bits so that the differences between two sets can be recovered from the XOR of their sketches, provided that there are at most D of them.

From the algorithmic viewpoint, inserting a key amounts to XORing the sketch with the corresponding column of $\mathcal{H}$ composed of D blocks of w bits (elements of $GF(2^w)$). Computing each block amounts to D multiplications in $GF(2^w)$[6]. A multiplication in $GF(2^w)$ can be done with $\mathcal{O}(w \log w)$ bit operations [13], but only with $\mathcal{O}(w)$ operations in the RAM model (see Appendix). Thus, an insertion of a key takes time $\mathcal{O}(Dw) = \mathcal{O}(D \log U)$.

[6] We refer to [5] for details on how keys of $\mathcal{U}$ are represented as elements of $GF(2^w)$.

Algorithm `insert`$(x, \mathcal{S})$**:**
- `toggle`(x) in $\mathcal{S}.A$
- $\mathcal{S}.H \leftarrow \mathcal{S}.H \oplus h(x)$
- `toggle_BCH`$(x, \mathcal{S}.C)$

Algorithm `diff`$(\mathcal{S}_1, \mathcal{S}_2)$**:**
- $\hat{\mathcal{S}}.A \leftarrow$ `merge`$(\mathcal{S}_1.A, \mathcal{S}_2.A)$
- $\hat{\mathcal{S}}.H \leftarrow \mathcal{S}_1.H \oplus \mathcal{S}_2.H$
- $\hat{\mathcal{S}}.C \leftarrow$ `merge_BCH`$(\mathcal{S}_1.C, \mathcal{S}_2.C)$
- **return** $\hat{\mathcal{S}}$

Algorithm `report`$(\mathcal{S})$**:**
- $S_{\text{dec}} \leftarrow$ `decode`$(\mathcal{S}.A)$
- **for** $x \in S_{\text{dec}}$ **do**
  - $\mathcal{S}.H \leftarrow \mathcal{S}.H \oplus h(x)$
- **if** $\mathcal{S}.H \neq 0$ **then**
  - **for** $x \in S_{\text{dec}}$ **do**
    - `toggle_BCH`$(x, \mathcal{S}.C)$ // delete x from $\mathcal{S}.C$
  - $S'_{\text{dec}} \leftarrow$ `decode_BCH`$(\mathcal{S}.C)$
- **return** $S_{\text{dec}} \triangle S'_{\text{dec}}$

Figure 2 Sketch operations. `insert`$(x, \mathcal{S})$ inserts key x to sketch $\mathcal{S}$, `diff`$(\mathcal{S}_1, \mathcal{S}_2)$ computes the difference of sketches $\mathcal{S}_1$ and $\mathcal{S}_2$, `report`$(\mathcal{S})$ reports keys stored in $\mathcal{S}$.

Decoding is a more complex operation. The efficient syndrome decoding algorithm of [5] requires $\mathcal{O}(D^2 w)$ multiplications in $GF(2^w)$ resulting in $\mathcal{O}(D^2 w^2) = \mathcal{O}(D^2 \log^2 U)$ time for decoding. Note that if the number $d < D$ of errors is known, then by properties of syndromes, it is sufficient to decode the first dw bits of the sketch, and the decoding complexity becomes $\mathcal{O}(d^2 \log^2 U)$.

Denote `toggle_BCH`(x, C) and `decode_BCH`(C) the operation of inserting/deleting a key x to/from a BCH sketch C and decoding a BCH sketch C, respectively. Complexity of BCH sketches is summarized in the following theorem.

▶ **Theorem 16.** *Consider a set S containing no more than D keys from U. Let S be stored in a BCH sketch C of $D \log(U+1)$ bits of space. Then*
- `toggle_BCH`(x, C) *takes $\mathcal{O}(D \log U)$ time,*
- `decode_BCH`(C) *decodes keys of S with no error in $\mathcal{O}(D^2 \log^2 U)$ time.*

Reconciliation of two sets is done by simply XORing their sketches C_1 and C_2, which can be trivially sped up by packing bits in words. We denote this operation by `merge_BCH`(C_1, C_2). A software implementation of a BCH sketch for set reconciliation is available [19].

5 IBLT with stash

Our goal is to enhance the IBLT in order to solve set reconciliation with a much stronger success guarantee than that provided by IBLT alone ([1], Table 1) while keeping small space and fast decoding time provided by the IBLT approach. Given an upper bound D on the size of set differences, our sketch $\mathcal{S}$ consists of three components depending on a parameter r:
- an IBLT $\mathcal{S}.A$ of size $n = (c_3 + \varepsilon)D$ with three hash functions, as defined in Section 2,
- a control checksum $\mathcal{S}.H$ of r bits,
- a *stash* data structure $\mathcal{S}.C$ defined as a BCH syndrome of $r \log U$ bits (see Section 4).

The control checksum $\mathcal{S}.H$ is used to check if the decoding of the main IBLT $\mathcal{S}.A$ succeeded. Before the decoding, $\mathcal{S}.H = \bigoplus_{x \in S} h(x)$ where $h : \mathcal{U} \to 2^r$ is a random hash function and S is the currently stored key set.

The stash $\mathcal{S}.C$ is configured so that it can decode up to r keys with no error, as described in Section 4. The entire sketch takes $(c_3 + \varepsilon)D \log U + r(1 + \log U) = (c_3 + \varepsilon)D \log U + o(D)$ bits.

Figure 2 shows Algorithms for inserting a key to a sketch, computing a sketch difference, and reporting keys.

A run of `report`($\mathcal{S}$) (Figure 2) can follow two scenarios. If the number of keys in the sketched set is no more than D, the main IBLT recovers those keys with probability $1 - \tilde{O}(1/D)$ (Theorem 2). With probability $\tilde{O}(1/D)$, however, the recovery may fail by "getting stuck", by resulting in a "falsely empty" IBLT (see [1]), or by being aborted after $2n$ steps. Checking the checksum ensures that an incorrect decoding will be recognized except with probability $\mathcal{O}(1/2^r)$. By Theorem 2, with probability $2^{-\Omega(r)}$, the output S_{dec} differs from S by no more than r missing or foreign keys. The algorithm resorts then to the stash in order to correct the output, i.e. to recover the missing/superfluous keys, under assumption that there are at most r of them. In this case, the stash reports those keys with no error.

The execution time of `report` depends on whether the correction step is triggered or not. If the decoding of the main IBLT is deemed to be successful by the checksum, the time taken will be $\mathcal{O}(D(1 + \frac{r}{w}))$, where w is the wordsize. Here we assume that updating the checksum is done by bit packing and takes $\mathcal{O}(\frac{r}{w})$ time. Otherwise the algorithm will spend additional $\mathcal{O}(r \log U)$ time on performing `toggle_BCH` on each of $\mathcal{O}(D)$ key in S_{dec}, followed by the decoding taking additional $\mathcal{O}(r^2 \log^2 U)$ time. Since the stash is activated with probability $\tilde{O}(1/D)$, the expected time is $\mathcal{O}(D(1 + \frac{r}{w}) + r \log U + \frac{r^2 \log^2 U}{D})$. Assuming $r = \min\left(\mathcal{O}(D/\log^2 U), \mathcal{O}(\log U)\right)$, the time of `report` is $\mathcal{O}(D)$.

We summarize the properties of `report` in the following theorem.

▶ **Theorem 17.** *Let $\mathcal{S}$ be a sketch built for a key set S and $r = \min\left(\mathcal{O}(D/\log^2 U), \mathcal{O}(\log U)\right)$. Then*

- *$\mathcal{S}$ takes $(c_3 + \varepsilon)D \log U + r(1 + \log U) = (c_3 + \varepsilon)D \log U + o(D)$ bits,*
- *inserting an element to $\mathcal{S}$ takes time $\mathcal{O}(r \log U)$,*
- *when $|S| \leq D$, `report`($\mathcal{S}$) correctly recovers S with probability $1 - 2^{-\Omega(r)}$,*
- *`report`($\mathcal{S}$) takes $\mathcal{O}(D)$ expected time.*

Note that if $r = \omega(\log D)$, the failure probability for `report` to recover a set of up to D keys is $o(1/D)$, as opposed to $\tilde{O}(1/D)$ for recovery without stash [1].

For two sets S and T represented by their respective sketches $\mathcal{S}_S$ and $\mathcal{S}_T$, `diff`$(\mathcal{S}_S, \mathcal{S}_T)$ is the sketch of symmetric difference $S \Delta T$. Therefore, Theorem 17 applies directly to the set reconciliation problem. In particular, if $|S \Delta T| \leq D$, then $S \Delta T$ can be recovered with guarantees stated in Theorem 17.

6 Distinguishing $S \setminus T$ and $T \setminus S$

The algorithm we presented computes symmetric difference $S \Delta T$ but is not capable of distinguishing elements of S and T in the output. However, the latter is desirable for many applications. Here we outline how our sketch and set reconciliation protocol (Figure 2) can be modified in order to make this possible. The general idea is to assign a "sign" to keys depending on whether they come from S or T and to keep track of it in a consistent manner. Formal proofs are left to the full version of the paper.

Modified IBLT. We modify our IBLT (Section 2) as follows. IBLT entries will now be *ternary* strings of $\{0, 1, 2\}^\nu$ seen as elements of group $\mathbb{Z}_3^\nu$ (additive group of $GF(3^\nu)$). The first *trit* encodes a *sign* and the other $\nu - 1$ trits encode a keysum. We use an appropriate encoding of $\mathcal{U}$ into strings $\{0, 1, 2\}^{\nu-1}$ and denote by $\widetilde{x}$ the encoding of x.

Toggling a key x is replaced by two operations: insertion and deletion. Inserting x into an IBLT is done by adding (in $\mathbb{Z}_3^\nu$) $1\widetilde{x}$ to each of the three entries at $h(x)$, and deleting x is done by subtracting $1\widetilde{x}$ from each of the three entries at $h(x)$ or, equivalently, adding

$2(-\widetilde{x}) \equiv (-1)(-\widetilde{x})$ where $-\widetilde{x}$ is the inverse of $\widetilde{x}$ in $\mathbb{Z}_3^{\nu-1}$. Observe that insertion and deletion of x cancel each other out, and that inserting twice the same key x is equivalent to deleting x and vice versa.

IBLT operations (Figure 1) are modified as follows. Operation `looksPure` checks if the first trit equals 1 or $2 \equiv -1$. If it is 2, the key assumed to sit in this entry is the inverse of the retrieved key value. `toggle`(x) deletes or inserts x from/to each of the three entries at $h(x)$ depending on whether x has been identified as *pure* with the first trit 1 or 2, respectively. Merging two IBLTs `merge`(A, A') now becomes non-commutative and is done by subtracting (in $\mathbb{Z}_3^{\nu}$) A' from A entry-wise.

Consider a sketch resulting from `merge`(A, A'). When reporting a set difference (modified Algorithm `report` in Figure 2) the algorithm now outputs *signed elements*, where elements annotated with 1 (i.e. those with the first trit equal to 1 at the moment of toggling) are interpreted as belonging to the first set, while those annotated with 2 are interpreted as belonging to the second. The symmetric difference operation Δ in both `decode` and `report` is also modified as follows. When computing $S_1 \Delta S_2$, an element is canceled only if it occurs in S_1 and S_2 with opposite signs. If an element occurs in S_1 and S_2 with the same sign, it is reported in the resulting set with the opposite sign.

Observe that the modified structure does not prevent anomalies to occur but only makes them less likely: an anomaly can still occur if the first trit is 1 or 2 but the entry actually contains more than one key. On the other hand, our analysis of Section 3 carries over to the signed case. The difference in S_{dec} compared to the original decoding is due to repeated reportings of the same key with different sign combinations, which does not affect the proof ideas of the main Theorem 2.

Modified control hashsum. Here the hashsum uses arithmetic summation instead of XOR. We assume that overflow is implemented in a consistent way, that is $(x + y) - y = x$ even if $x + y$ results in overflow. Toggling x entails subtracting $h(x)$ from the hashsum if x is annotated with 1, and adding if it is annotated -1. The hashsum of `diff`$(\mathcal{S}_1, \mathcal{S}_2)$ is now defined as $\mathcal{S}_1.H - \mathcal{S}_2.H$.

Modified BCH sketch. In order to deal with signed elements, we use BCH code over field $GF(3^{\nu-1})$. The BCH sketch of a set is still defined as the syndrome vector under assumption that each set element is encoded by value 1 in the error vector. The `merge_BCH` operation will now subtract the syndromes (in $GF(3)$) instead of XORing them. By linearity of the code, the "positive" and "negative" elements will correspond to positions with 1 and $2 \equiv -1$ respectively in the recovered error vector.

Given a BCH sketch resulting from the difference of sketches of input sets S and T, algorithm `report` will subtract the syndrome corresponding to the set S_{dec} of signed elements decoded by the IBLT. Naturally, positive and negative elements are encoded respectively by values 1 and 2 as well. Again, by linearity, the resulting syndrome will encode exactly the elements S'_{dec} so that $S_{\text{dec}} \Delta S'_{\text{dec}} = S \Delta T$, where positive (resp. negative) elements are those belonging to S (resp. T) only.

Observe that, in general, S_{dec} will contain a subset of $S \Delta T$ as well as possibly some foreign keys, however our modified definition of Δ will ensure a correct recovery of the original set. As an example, assume $x \in S \setminus T$ and assume x has been reported by the IBLT as positive, followed by another reporting of x (produced by an anomaly) as positive as well. From our definition of Δ, x will become negative in S_{dec} and will be added (rather than subtracted) to the BCH sketch. This will result in reporting x as negative again by the BCH sketch, and, by our definition of Δ, will be eventually correctly reported as positive in the final output.

The syndrome of the BCH code over $GF(3^{\nu-1})$ consists of at most $2D$ elements of $GF(3^{\nu-1})$ i.e. at most $4D \log_3 U$ bits by a straightforward encoding. Using ternary representation of keys and arithmetic operations in $GF(3^{\nu})$ instead of $GF(2^w)$ introduces only a constant factor change in time complexities of both insertion and decoding (see Appendix). In conclusion, time and robustness guarantees of Theorem 17 remain valid.

7 Concluding remarks

The presented solution to set reconciliation uses asymptotically negligible additional space and additional time for decoding compared to the IBLT-only solution, but provides a drastic improvement in robustness. The decoding time of our algorithm is small in expectation, however it becomes substantial in worst case. More precisely, when the BCH correction is activated, the decoding time becomes $O(D\, r\, \mathrm{polylog}(U))$ which can approach $O(D^2)$. We believe however this can be overcome by implementing the stash with expander codes which have much smaller insertion and decoding times. We leave details for future work.

Very recently, we learned about paper [8] that proposes a modified construction of IBLT and applies hash functions of restricted independence, rather than assuming them fully random. As a result, the construction takes a smaller space (compared to the original IBLT with the same error guarantee) and requires less randomness for hash functions. However, space is measured in terms of the number of IBLT buckets rather than in bits, which leads to a much larger multiplicative constant compared to our result. The decoding time of [8] appears to be larger than ours as well (not specified in the paper). On the other hand, an emphasis of [8] is to using hash functions of restricted randomness, the issue that we don't deal with in this work.

References

1 Jakob Bæk Tejs Houen, Rasmus Pagh, and Stefan Walzer. Simple set sketching. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 228–241. SIAM, 2023.

2 Daniella Bar-Lev, Avi Mizrahi, Tuvi Etzion, Ori Rottenstreich, and Eitan Yaakobi. Coding for IBLTs with listing guarantees. In *IEEE International Symposium on Information Theory, ISIT 2023, Taipei, Taiwan, June 25-30, 2023*, pages 1657–1662. IEEE, 2023. doi:10.1109/ISIT54713.2023.10206563.

3 Mahdi Cheraghchi. Coding-theoretic methods for sparse recovery. In *49th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2011, Allerton Park & Retreat Center, Monticello, IL, USA, 28-30 September, 2011*, pages 909–916. IEEE, 2011. doi:10.1109/Allerton.2011.6120263.

4 Mahdi Cheraghchi and João Ribeiro. Simple codes and sparse recovery with fast decoding. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 156–160. IEEE, 2019.

5 Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008. doi:10.1137/060651380.

6 David Eppstein and Michael T Goodrich. Straggler identification in round-trip data streams via Newton's identities and invertible Bloom filters. *IEEE Transactions on Knowledge and Data Engineering*, 23(2):297–306, 2011.

7 David Eppstein, Michael T. Goodrich, Frank Uyeda, and George Varghese. What's the difference? Efficient set reconciliation without prior context. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 218–229, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/2018436.2018462.

8 Nils Fleischhacker, Kasper Green Larsen, Maciej Obremski, and Mark Simkin. Invertible Bloom lookup tables with less memory and randomness. *CoRR*, abs/2306.07583, 2023. arXiv:2306.07583, doi:10.48550/arXiv.2306.07583.

9 Nils Fleischhacker, Kasper Green Larsen, and Mark Simkin. Property-preserving hash functions for Hamming distance from standard assumptions. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022*, pages 764–781, Cham, 2022. Springer International Publishing.

10 Sumit Ganguly. Counting distinct items over update streams. *Theoretical Computer Science*, 378(3):211–222, 2007. Algorithms and Computation. doi:10.1016/j.tcs.2007.02.031.

11 Sumit Ganguly and Anirban Majumder. Deterministic k-set structure. *Information Processing Letters*, 109(1):27–31, 2008. doi:10.1016/j.ipl.2008.08.010.

12 Michael T Goodrich and Michael Mitzenmacher. Invertible Bloom lookup tables. In *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 792–799. IEEE, 2011.

13 David Harvey and Joris van der Hoeven. Polynomial multiplication over finite fields in time $\mathcal{O}(n \log n)$. *J. ACM*, 69(2):12:1–12:40, 2022. doi:10.1145/3505584.

14 Mark G Karpovsky, Lev B Levitin, and Ari Trachtenberg. Data verification and reconciliation with generalized error-control codes. *IEEE Transactions on Information Theory*, 49(7):1788–1793, 2003.

15 Jeong Han Kim. Poisson cloning model for random graphs. In *Proc. ICM, Vol. III*, pages 873–898, 2006. URL: https://www.mathunion.org/fileadmin/ICM/Proceedings/ICM2006.3/ICM2006.3.ocr.pdf.

16 M. Leconte, M. Lelarge, and L. Massoulié. Convergence of multivariate belief propagation, with applications to cuckoo hashing and load balancing. In *Proc. 24th SODA*, pages 35–46, 2013. URL: http://dl.acm.org/citation.cfm?id=2627817.2627820.

17 F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-holland Publishing Company, 2nd edition, 1978.

18 Colin McDiarmid. *On the method of bounded differences*, pages 148–188. London Mathematical Society Lecture Note Series. Cambridge University Press, 1989. doi:10.1017/CBO9781107359949.008.

19 Minisketch: an optimized library for BCH-based set reconciliation. https://github.com/sipa/minisketch/. Accessed: 2023-03-28.

20 Yaron Minsky, Ari Trachtenberg, and Richard Zippel. Set reconciliation with nearly optimal communication complexity. *IEEE Transactions on Information Theory*, 49(9):2213–2218, 2003. doi:10.1109/TIT.2003.815784.

21 Michael Mitzenmacher and Rasmus Pagh. Simple multi-party set reconciliation. *Distributed Computing*, 31:441–453, 2018.

22 Michael Mitzenmacher and George Varghese. Biff (Bloom filter) codes: Fast error correction for large data sets. In *2012 IEEE International Symposium on Information Theory Proceedings*, pages 483–487. IEEE, 2012.

23 Avi Mizrahi, Daniella Bar-Lev, Eitan Yaakobi, and Ori Rottenstreich. Invertible Bloom lookup tables with listing guarantees. *CoRR*, abs/2212.13812, 2022. arXiv:2212.13812, doi:10.48550/arXiv.2212.13812.

24 Michael Molloy. Cores in random hypergraphs and boolean formulas. *Random Structures & Algorithms*, 27(1):124–135, 2005.

25 Thomas Morgan. *An exploration of two-party reconciliation problems*. PhD thesis, Harvard University, Cambridge, MA, May 2018. URL: https://dash.harvard.edu/handle/1/39947174.

A Appendix

Below we describe how to perform multiplication in $GF(2^w)$ and $GF(3^\nu)$ in RAM model in times $O(w)$ and $O(\nu)$ respectively.

A.1 Multiplications in $GF(2^w)$ in RAM model in $O(w)$ time

The algorithm we describe is folklore, but since we did not find a reference we briefly sketch it here. The multiplication of two elements x and y in $GF(2^w)$ proceeds in two phases. In the first phase, one computes the product $z = x \cdot y$, where $\cdot$ refers to a carry-less multiplication of the binary representation of x and y. Thus, z will be a bitstring of length $2w - 1$. This phase can be easily implemented in $O(w)$ time using $O(w)$ XOR and shift operation (the result is the XOR of copies of y right-shifted by bit-positions of x that are equal to one) [7]. The second phase is the polynomial euclidean division of z by an irreducible polynomial P of degree w. The remainder of the division will be the final result of the multiplication and can also be easily implemented in $O(w)$ time using $O(w)$ XOR and shift operations: successively for decreasing positions i of z starting from position $2w - 1$ check whether the bit at that position is equal to one and if so, do $z \leftarrow (z \oplus (P \ll (i - w))$.

A.2 Multiplications in $GF(3^\nu)$ in RAM model in $O(\nu)$ time

Likewise, multiplication in $GF(3^\nu)$ can be implemented in $O(\nu)$ time. For that we can use a representation of an element using 2ν bits, where each element from the base field $GF(3)$ is represented using 2 bits. Note that any integer in the range $[0..3^\nu - 1]$ can be converted to this representation in $O(\nu)$ time, by doing successive euclidean divisions by 3. Notice that a division by 3 can be simulated using multiplications and other elementary operations. For a technical reason that will become clear later, we will instead use 4 bits to represent each element of the base field, resulting in a representation that uses 4ν bits. Let G be the function that transforms integers of $[0..3^\nu - 1]$ into binary representation of 4ν bits. Let G^{-1} be the inverse of G. Now assume that given two elements x and y represented as integers, we want to compute $z = x \cdot y$. We will first convert x and y into their binary representations $x' = G(x)$ and $y' = G(y)$, then compute the product $z' = x' \cdot y'$, and finally convert z' back into an integer z using function G^{-1}. We now describe how we do multiplication in the 4ν-bit representation. For that, we successively extract the quadbits (4-bit units) of x', multiply each of them with y', and aggregate all of them. The reason we use 4 bits instead of 2 to represent each digit is to accomodate the temporary result of the multiplication of two digits (numbers in $[0..2]$) which will be in the range $[0..4]$ (before doing Modulo 3 operation). Finally, the remainder of euclidean division by an irreducible polynomial can also be done in time $O(\nu)$ using $O(\nu)$ operations. Figure 3 shows the pseudocode of all steps of the algorithm.

[7] Note that in practice, most modern processors natively support carry-less multiplication in constant time using much less complex hardware implementation than standard multiplication, but is not part of the standard RAM operations.

Algorithm $G(x)$**:**
- $x' \leftarrow 0$
- $j \leftarrow 0$
- **for** $i \leftarrow 1$ **to** ν **do**
  - $x' \leftarrow x' + ((x \mod 3) \ll j)$
  - $x \leftarrow x/3$
  - $j \leftarrow j + 4$
- **return** x'

Algorithm $G^{-1}(x')$**:**
- $x \leftarrow 0$
- $j \leftarrow 4 \cdot (w - 1)$
- **for** $i \leftarrow 1$ **to** ν **do**
  - $x \leftarrow x \cdot 3 + ((x' \gg j) \wedge (0011)_2)$
  - $j \leftarrow j - 4$
- **return** x

Algorithm `MOD3`(x)**:**
- //reduce each quadbit modulo 3
- //works only if quadbit in range [0..5]
- $A \leftarrow (x + 1) \wedge (0100)_2^\nu$
- $x \leftarrow x - ((A \gg 2) + (A \gg 1))$
- **return** x

Algorithm `reduce`(x, P)**:**
- $//P$: irreducible polynomial represented in quadbits
- $P' \leftarrow$ `MOD3`$(2 \cdot P)$
- $j \leftarrow 4 \cdot (2 \cdot \nu - 1)$
- **for** $i \leftarrow 1$ **to** ν **do**
  - **if** $(x \gg j) > 0$ **then**
    - $x \leftarrow x + (0011)_2^\nu$
    - **if** $(x \gg j) = (P \gg (\nu - 1))$ **then**
      - $x \leftarrow x - P$
    - **else**
      - $x \leftarrow x - P'$
    - $x \leftarrow$ `MOD3`(x)
  - $j \leftarrow j - 4$
- **return** x

Algorithm `mult`(x', y')**:**
- $z' \leftarrow 0$
- $j \leftarrow 4 \cdot (\nu - 1)$
- **for** $i \leftarrow 1$ **to** ν **do**
  - $t \leftarrow (x' \gg j) \wedge (0011)_2$
  - $t \leftarrow t \cdot y'$
  - $t \leftarrow$ `MOD3`(t)
  - $z' \leftarrow (z' \ll 4) + t$
  - $z' \leftarrow$ `MOD3`(z')
  - $j \leftarrow j - 4$
- $z' \leftarrow$ `reduce`(z', P)
- **return** z'

Algorithm `MULT`(x, y)**:**
- $x' \leftarrow G(x)$
- $y' \leftarrow G(y)$
- $z' \leftarrow$ `mult`(x, y)
- $z \leftarrow G^{-1}(z)$
- **return** z

Figure 3 Multiplication algorithm $GF(3^\nu)$ modulo an irreuducible polynomial.

Oracle Separation of QMA and QCMA with Bounded Adaptivity

Shalev Ben-David
Institute for Quantum Computing, University of Waterloo, Canada

Srijita Kundu
Institute for Quantum Computing, University of Waterloo, Canada

Abstract

We give an oracle separation between QMA and QCMA for quantum algorithms that have bounded adaptivity in their oracle queries; that is, the number of rounds of oracle calls is small, though each round may involve polynomially many queries in parallel. Our oracle construction is a simplified version of the construction used recently by Li, Liu, Pelecanos, and Yamakawa (2023), who showed an oracle separation between QMA and QCMA when the quantum algorithms are only allowed to access the oracle classically. To prove our results, we introduce a property of relations called *slipperiness*, which may be useful for getting a fully general classical oracle separation between QMA and QCMA.

2012 ACM Subject Classification Theory of computation → Quantum complexity theory

Keywords and phrases Quantum computing, computational complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.21

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2402.00298

Funding *Shalev Ben-David*: Supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), DGECR-2019-00027 and RGPIN-2019-04804[1].
Srijita Kundu: Supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grants Program, and Fujitsu Labs America.

1 Introduction

It is a long-standing open problem in quantum complexity theory whether the two possible quantum analogs of the complexity class NP are equivalent. QMA is defined as the class of decision problems that are solvable by a polynomial-time quantum algorithm that has access to a polynomial-sized *quantum* witness, whereas QCMA is the class of decision problems that are solvable by a polynomial-time quantum algorithm that only has access to the polynomial-sized *classical* witness. In other words, the question asks: are quantum proofs more powerful than classical proofs?

While the inclusion QCMA $\subseteq$ QMA is easy to see, the question of whether these two classes are actually equal, which was first posed by Aharonov and Naveh [3], remains unanswered. Indeed, an unconditional separation between these classes is beyond currently known techniques.

An easier, but still unsolved, problem is to show an oracle separation between QMA and QCMA. This is because oracle separations in the Turing machine model can be shown by means of separations in the much simpler model of *query complexity*, where similar

[1] Cette recherche a été financée par le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG), DGECR-2019-00027 et RGPIN-2019-04804.

separations between complexity classes are routinely shown (for example, a recent oracle separation between BQP and PH was provided in [12]). The problem of finding an oracle separation between QMA and QCMA has been a longstanding focus of the quantum computing community; it boils down to asking whether quantum proofs are more powerful than classical proofs in the query model.

1.1 Previous work

The first progress on the question of an oracle separation of QMA and QCMA was made by Aaronson and Kuperberg [2], who showed that there is a quantum oracle, i.e., a blackbox unitary, relative to which QMA $\neq$ QCMA. Later, Fefferman and Kimmel [7] showed that the separation also holds under what they called an "in-place permutation oracle", which is still inherently quantum. Ideally, we would like to get these separations in the standard model of classical oracles: classical functions that a quantum algorithm may query in superposition. [4] showed separations between QMA and QCMA in other non-standard oracle models.

Very recently, there has been some progress on this question, with two different variations of the standard classical oracle model. Natarajan and Nirkhe [11] showed an oracle separation relative to a "distributional oracle". This essentially means that the classical oracle is drawn from a distribution, which the prover knows, but the specific instance drawn is not known to the prover. Therefore, the witness only depends on the distribution over the oracles, which makes it easier to show QCMA lower bounds. Following this, [9] showed a separation with a classical oracle that is fully known to the prover, but assuming the verifier is only allowed to access this classical oracle classically, i.e., the verifier is not allowed to make superposition queries (this makes the class similar to MA in terms of its query power and witness type). This model is also simpler to analyze because whatever information the verifier gets from the oracle by classically querying it, could also have been provided as the classical QCMA witness. [9] also gave an alternate construction of a distributional oracle separation, with a simpler proof than [11]. Their constructions are based on the relational problem used by Yamakawa and Zhandry [14], in their result on quantum advantage without structure.

Closely related to the QMA vs QCMA question is the $\mathsf{BQP}_{/\mathrm{qpoly}}$ vs $\mathsf{BQP}_{/\mathrm{poly}}$ question. $\mathsf{BQP}_{/\mathrm{qpoly}}$ is the class of decision problems that are solvable by a polynomial-time quantum algorithm with access to polynomial-sized quantum *advice*, which depends non-uniformly on the length of inputs, but nothing else. $\mathsf{BQP}_{/\mathrm{qpoly}}$ is the class of decision problems solvable by a polynomial-time quantum algorithm with access to polynomial-sized classical advice. Most works which have found oracle separations for QMA vs QCMA in various oracle models, such as [2, 11, 9], have also found oracle separations between $\mathsf{BQP}_{/\mathrm{qpoly}}$ and $\mathsf{BQP}_{/\mathrm{poly}}$ with related constructions in the same oracle models.

The question of the relative power of classical vs quantum advice was recently resolved unconditionally (without oracles) for relational problems by Aaronson, Buhrman and Kretschmer [1], who showed an unconditional separation between $\mathsf{FBQP}_{/\mathrm{qpoly}}$ and $\mathsf{FBQP}_{/\mathrm{poly}}$. $\mathsf{FBQP}_{/\mathrm{qpoly}}$ and $\mathsf{FBQP}_{/\mathrm{poly}}$ are the classes of relational problems analogous to $\mathsf{BQP}_{/\mathrm{qpoly}}$ and $\mathsf{BQP}_{/\mathrm{poly}}$ respectively. Their result was based on observing that separations between quantum and classical one-way communication complexity can be used to show separations between classical and quantum advice. The reason their result only works for the relation classes is that a separation in one-way communication complexity which satisfies the necessary conditions can only hold for relational problems. The specific relational problem used in [1] is known as the Hidden Matching problem. But as was observed in [9], the Yamakawa-Zhandry problem [14] also achieves the required communication separation, and could have been used instead. In light of this, the constructions in [14] can viewed as a way to convert relational

separations in one-way communication complexity, which correspond to relational separations for quantum vs classical advice, to separations for decision QMA vs QCMA, and $\mathsf{BQP}_{/\mathrm{qpoly}}$ vs $\mathsf{BQP}_{/\mathrm{poly}}$, relative to classically accessible oracles. The construction is not blackbox – it does not work if the Hidden Matching Problem is used instead of the Yamakawa-Zhandry problem, though it plausibly might work with a parallel repetition of the former.

1.2 Our results

Unlike previous work, we prove an oracle separation between QMA and QCMA relative to a *bona fide* regular oracle with regular (quantum) queries. Our catch is, instead, that we only allow the algorithms *bounded adaptivity*.

Bounded adaptivity means that the number of rounds of queries made by the algorithms is small, although there can be polynomially many queries in each round. Although our result is not formally stronger than those of [11] and [9], we feel our result is intuitively closer to a full QMA-QCMA separation, as it allows the full power of classical proofs and some of the power of quantum queries. Our main result is formally stated below.

▶ **Theorem 1.** *There is an oracle* $\mathcal{O}\colon \{0,1\}^* \to \{0,1\}$ *such that* $\mathsf{QCMA}^{\mathcal{O},r} \neq \mathsf{QMA}^{\mathcal{O},r}$*, for* $r = o(\log n/\log\log n)$.

In the above statement, $\mathsf{QMA}^{\mathcal{O},r}$ is the class of decision problems solvable by QMA algorithms that have oracle access to $\mathcal{O}$, and make at most r batches of parallel queries to $\mathcal{O}$; $\mathsf{QCMA}^{\mathcal{O},r}$ is defined analogously. The parameter n is the efficiency parameter (so the number of queries is $\mathrm{poly}(n)$).

▶ **Theorem 2.** *There is a function family* $F = \{F_N\}_{N\in I}$ *which is efficiently computable in 1-round query* QMA*, but with the property that the growth rate of* $\mathrm{QCMA}^r(F_N)$ *for* $r = o(\log\log N/\log\log\log N)$ *as* $N \to \infty$ *is not in* $O(\mathrm{polylog}(N))$.

We shall formally define the query versions of QMA and QCMA, and the r-round QCMA query complexity QCMA^r, which are used in this theorem statement, in Section 2.1.

Our construction for the query complexity separation is a somewhat simplified version of the construction in [9], which is based on the Yamakawa-Zhandry problem. [14] and [10] showed that there exists a relational problem R_f, indexed by functions $f : [n] \times \{0,1\}^m \to \{0,1\}$, for $m = \Theta(n)$, such that given oracle access to a quantum advice $|z_f\rangle$, a quantum algorithm on any input $x \in \{0,1\}^n$, and on average over f, can find a u such that $(x,u) \in R_f$[2]. On the other hand, no quantum algorithm can find such an u for most x, when given only a classical advice z_f, and classical query access to f. Using this relation R_f, for a subset $E \subseteq \{0,1\}^n$, we construct the following oracle:

$$O[f,E](x,u) = \begin{cases} 1 & \text{if } (x,u) \in R_f \wedge x \notin E \\ 0 & \text{otherwise.} \end{cases}$$

The 1-instances of the problem F_N that will separate QMA and QCMA in the query complexity model will be $O[f,\emptyset]$, and the 0-instances will be $O[f,E]$ for $|E| \geq \frac{2}{3}\cdot 2^n$, for a large subset of all functions f. This is essentially the same construction that is used in [9], except they also use an additional oracle G for a random function from $\{0,1\}^n$ to $\{0,1\}^n$, which O also depends on.

[2] The Yamakawa-Zhandry relation is a TFNP relation, which means that the u-s are of $\mathrm{poly}(n)$ length, and a u such that $(x,u) \in R_f$ exists for every x.

Note that the query complexity lower bound we obtain for QCMA is of a different nature than the one obtained in [9]: we need to lower bound (bounded-round) quantum query algorithms instead of only classical query algorithms, and we focus on the worst-case rather than average-case setting. In order to get an oracle separation for Turing machines from a separation in query complexity, one needs to use a diagonalization argument; because our result is set up a bit differently than in previous work, we reprove the diagonalization argument for our setting in. This can be found in Appendix A of the full version of this paper on arXiv.

Finally, we emphasize that the bounded adaptivity limitation of our result is because we allow the full power of classical proofs and also quantum queries. If one were to drop the power of classical proofs (resulting in the class BQP), or if one were to drop the power of quantum queries (resulting in, essentially, MA), it would follow from [9] that close variants of F_N cannot be solved even without the bounded-round restriction. We conjecture their lower bounds apply to F_N as well.

1.3 Our techniques

We briefly describe the techniques used to obtain the query complexity result. We start by observing that the oracle $O[f, \emptyset]$ is essentially just a verification oracle for the Yamakawa-Zhandry relation. Therefore, there is a quantum witness and a quantum algorithm that can distinguish $O[f, \emptyset]$ and $O[f, E]$ by using this witness, with only one query, with probability $1 - 2^{-\Omega(n)}$ over f. The witness for the yes instance $O[f, \emptyset]$ is simply the quantum advice for the Yamakawa-Zhandry problem, which finds a u for any x with probability $1 - 2^{-\Omega(n)}$ over f. The quantum algorithm finds a u for a random x using the witness, and queries the oracle. Since the no instances return 0 on any (x, u) for most x, this algorithm can distinguish $O[f, \emptyset]$ and $O[f, E]$ for $1 - 2^{-\Omega(n)}$ fraction of the f-s.

We now consider the uniform distribution over these good f-s (for which we can distinguish $O[f, \emptyset]$ and $O[f, E]$ with quantum advice), which has $2^{\Omega(n)}$ min-entropy. If there was a classical witness function depending on f, of size k, that made a quantum algorithm accept $O[f, \emptyset]$ for these f-s, then there would exist a fixed witness string w that would make $O[f, \emptyset]$ accept for 2^{-k} fraction of f-s. The quantum algorithm depends on the witness, but if we fix the witness string w, the algorithm is fixed, and we can then ignore the dependence of the algorithm on the witness.

We now attempt to remove rounds of the quantum query algorithm, starting with the first round, while keeping the behavior of the algorithm the same on as many oracles as possible. Every time we remove a round, we restrict our attention to a smaller set of oracles, all of which are consistent with a growing partial assignment we assume is given to us. At the end, the quantum algorithm will have no rounds left, and hence will make no queries; we want the set of oracles $O[f, \emptyset]$ on which the behavior is preserved to be non-empty, because then we can conclude that the algorithm cannot distinguish $O[f, \emptyset]$ and $O[f, E]$ for some large erased set E (since it now makes no queries).

To remove the first round of the query algorithm, we start by considering the the uniform distribution over the 2^{-k} fraction of good f-s such that $O[f, \emptyset]$ is accepted by w. This distribution has $2^{\Omega(n)} - k$ min-entropy, and therefore, by a result of [8, 6], it can be written as a convex combination of finitely many dense distributions. Dense distributions are a concept that was first introduced in the context of communication complexity: in a dense distribution, some coordinates are fixed, and the rest of the coordinates have high min-entropy in every subset. In fact we will not need the full convex combination of dense distributions – we restrict our attention to one such distribution in the convex combination, and try to preserve the behavior of the quantum algorithm only within a subset of its support.

Since some coordinates are fixed in our dense distribution, the probability over this distribution of the event $(x, u) \in R_f$ non-negligible, for some (x, u) pairs (this probability is exactly 2^{-n} over uniform f). The quantum algorithm can potentially learn a lot about f by querying the oracle $O[f, \emptyset]$ for these pairs. Therefore, we shall fix the coordinates of f that are fixed by (x, u) being in R_f. Here is where we use the fact that the Yamakawa-Zhandry relation is what we shall call *slippery*. This essentially means that given a small number of fixed coordinates for f, the number (x, u) pairs that have non-negligible probability is not too high, and the number of extra coordinates fixed by these (x, u) pairs being in R_f is also not too high. The Yamakawa-Zhandry relation being slippery essentially follows from it using a code that has good list recoverability properties. (The Hidden Matching relation, or its parallel repetition, are not slippery by this definition, and so our construction does not work with these.)

Using the slippery property, we can increase the size of the partial assignment by not too much, and via a hybrid-like argument [5], we can ensure that the first round of the quantum algorithm does not learn much from queries outside this partial assignment. We then restrict our attention to oracles consistent with this partial assignment; on those, we can simulate the first round of the algorithm without making real queries (we simply use the known partial assignment and guess "0" on the rest of the oracle positions, which are highly unlikely to be 1). This way, we get a quantum algorithm with one fewer round, which mimics the original algorithm on a small (but not too small) set of oracles.

Continuing this way, we eliminate all rounds of the algorithm while still maintaining a non-empty set of oracles on which the behavior is preserved. Each such oracle can be "erased", turning a 1-input into a 0-input, so we only need the final 0-round algorithm to preserve the behavior of the original algorithm on at least one input. Using this technique, we can handle up to $o(\log n / \log \log n)$ rounds of $O(\operatorname{poly} n)$ non-adaptive quantum queries each.

1.4 Discussion and further work

We expect our techniques for the QMA vs QCMA separation may also work for a $\mathsf{BQP}_{/\mathrm{qpoly}}$ vs $\mathsf{BQP}_{/\mathrm{poly}}$ separation with boundedly adaptive oracle queries, using the same problem that is described in [9]. Their oracle in the query complexity setting is given by a random function G, which the BQP algorithm has to compute given oracle access to

$$O[f, G](x, u) = \begin{cases} G(x) & \text{if } (x, u) \in R'_f \\ \perp & \text{otherwise,} \end{cases}$$

and a quantum or classical advice. Here R'_f is a modified 1-out-of-n version of the Yamakawa-Zhandry problem, which has better completeness properties, but is similar to the original problem otherwise. Clearly this problem can be solved in $\mathsf{BQP}_{/\mathrm{qpoly}}$ by using the quantum advice for the Yamakawa-Zhandry problem. It cannot be solved on input x with any classical advice and with access to an oracle that outputs $\perp$ for every (x, u). In order to show a $\mathsf{BQP}_{/\mathrm{poly}}$ lower bound for this problem, one needs that there exist many x-s such that a quantum algorithm with classical advice cannot distinguish $O[f, G]$ from a version of $O[f, G]$ that is erased on those x-s. Since $O[f, G]$ essentially serves as a verification oracle for the relation just as $O[f, \emptyset]$ does in the QMA vs QCMA construction, we expect that when the quantum algorithm has bounded rounds, a proof very similar to our QCMA lower bound will work.[3]

[3] R'_f, being a 1-out-of-n version of R_f, has worse slipperiness properties than R_f, which gets in the way of applying our techniques. But instead of using R'_f for better completeness, we can focus on the (large enough) subset of x, f for which the BQP algorithm with quantum advice works for R_f with high probability, and have $O[f, G]$ give $G(x)$ for free outside of this set. This would make the analysis very similar to the QMA vs QCMA case.

The final goal is, of course, to be able to show both these results without a bound on the number of rounds of oracle queries the quantum algorithm makes. As mentioned earlier, we fail to do this because the slipperiness parameters of the relation we picked are not good enough, and our methods would work to separate QMA and QCMA with an analogous problem definition where the Yamakawa-Zhandry relation is replaced by a different relation R_f that has the appropriate slipperiness property.

We now expand more on the required strong slipperiness property. Let R_f be a family of TFNP relations on $\{0,1\}^n \times \{0,1\}^m$ indexed by $f \in \{0,1\}^N$, where $m = \text{poly}(n)$ and $N = \Omega(2^n)$. We further assume R_f satisfies the property that if $(x,u) \in R_f$, then there is a polynomial-sized partial assignment p for f which certifies this, i.e., $(x,u) \in R_f \ \forall f \supseteq p$. Let $\mathcal{P} \subseteq \{0,1,*\}^N$ denote the set of polynomial-sized partial assignments for f. We define the extended version $\widetilde{R}$ of the family of relations R_f as follows:

$$\widetilde{R} = \{(p,x,u) : p \text{ is the minimal partial assignment s.t. } (x,u) \in R_f \, \forall f \supseteq p\}.$$

Since p is polynomial-sized, if we consider the uniform distribution over $\{0,1\}^N$, $\Pr[p \subseteq f]$ is exponentially small. Now consider a partial assignment q for f with size at most $s(n)$; we fix the bits in q and generate the other bits of f uniformly at random, which can make the probability of some other partial assignments p non-negligible. The slipperiness property is concerned with the total additional support (outside of q) of all partial assignments p such that $\Pr[p \subseteq f | q \subseteq f]$ is non-negligible, and $(p,x,u) \in \widetilde{R}$. We say $\widetilde{R}$ is $(\eta, s(n), t(n))$-slippery if for all $s(n)$-sized q, the total additional support of all p-s such that $\Pr[p \subseteq f | q \subseteq f] \geq \eta$ and $(p,x,u) \in \widetilde{R}$ is at most $t(n)$. See Definition 13 for a more formal definition.

Our techniques show that the following conjecture implies an oracle separation between QMA and QCMA.

▶ **Conjecture 3.** *There exists a family of* TFNP *relations R_f such that*

1. *There exists a polynomial-time algorithm $\mathcal{A}$, and for each f, a* poly(n)*-sized quantum state $|z_f\rangle$ such that, given access to x and $|z_f\rangle$, $\mathcal{A}$ can find u such that $(x,u) \in R_f$, with probability at least $1 - 2^{-\Omega(n)}$ over a product distribution $\mu_X \mu_F$ on x, f. Moreover, μ_X and μ_F are required to respectively have min-entropy $2^{\Omega(n)}$ and $\Omega(n)$.*
2. *There exists a function $s(n) = 2^{o(n)}$ such that for all polynomial functions $p(n)$, the extended relation $\widetilde{R}$ is $(1/p(n), s(n), t(n))$-slippery for some $t(n)$ such that $\log(t(n)) = o(\log(s(n)))$.*

Assuming Conjecture 3 is true, the oracle function separating QMA and QCMA would be distinguishing $O[f, \emptyset]$ and $O[f, E]$, for $|E| \geq \frac{2}{3} \cdot 2^n$, which we have defined earlier, using a relation R_f that satisfies the conjecture. (We can only prove the Yamakawa-Zhandry relation is $(\eta, s(n), t(n))$-slippery, with $t(n)$ bigger than $s(n)$, though it is possible that is satisfies the conjecture under finer analysis.)

We further note that any family of relations R_f that satisfies Conjecture 3 must give an exponential separation between quantum and randomized one-way communication complexity, with the communication setting being that Alice gets input f, Bob gets input x, and Bob has to output u such that $(x,u) \in R_f$.[4] This is because, if there was a polynomial-sized classical message w_f that Alice could send to Bob in the communication setting, then w_f could also serve as a QCMA proof. Therefore, it seems that the slipperiness condition could also be used for lower-bounding one-way randomized communication complexity (although weaker slipperiness parameters than in the conjecture would also suffice for this).

[4] Strictly speaking, condition 1 of the conjecture only implies that there exists a one-way communication protocol, in which Alice sends the state $|z_f\rangle$, which works on average over x and f, whereas we usually require worst-case success in communication complexity. However, we can restrict to the set of x and f for which the algorithm $\mathcal{A}$ works, in order to get the communication problem.

2 Preliminaries

2.1 QMA and QCMA in query complexity

In this section, we review the formal definitions of QMA, QCMA, computationally-efficient QMA, and bounded-round QCMA in the context of query complexity.

▶ **Definition 4** (Bounded-round quantum query algorithm). *For $r, T, n \in \mathbb{N}$, give the following definition of a quantum query algorithm Q acting on n bits, using r rounds, with T queries in each round. The algorithm will be a tuple of $r+1$ unitary matrices, $Q = (U_0, U_1, \ldots, U_r)$. These unitary matrices will each act on T "query-input" registers of dimension n, T "query-output" registers of dimension 2, an "output" register of dimension 2, and a work register of arbitrary dimension.*

For each $x \in \{0,1\}^n$, let U^x be the oracle unitary, which acts on the query-input and query-output registers by mapping

$$|i_1\rangle |b_1\rangle |i_2\rangle |b_2\rangle \ldots |i_T\rangle |b_T\rangle \to |i_1\rangle |b_1 \oplus x_{i_1}\rangle |i_2\rangle |b_2 \oplus x_{i_2}\rangle \ldots |i_T\rangle |b_T \oplus x_{i_T}\rangle$$

for all $i_1, \ldots, i_T \in [n]$ and all $b_1, \ldots, b_T \in \{0,1\}$. We extend U^x to other registers via a Kronecker product with identity, so that U^x ignores the other registers.

The action of the algorithm Q on input $x \in \{0,1\}^n$, denoted by the Bernoulli random variable $Q(x)$, will be the result of measuring the output register of the state

$$U_r U^x U_{r-1} U^x \ldots U^x U_1 U^x U_0 |\psi_{init}\rangle ,$$

where $|\psi_{init}\rangle$ is a fixed initial state.

We will use the term "T-query quantum algorithm" without referring to the number of rounds to indicate T rounds with 1 query in each.

▶ **Definition 5** (Query algorithm with witness). *Let Q be a r-query quantum algorithm on n bits with T queries in each round. For any quantum state $|\phi\rangle$ and any $x \in \{0,1\}^n$, let $Q(x, |\phi\rangle)$ be the random variable corresponding to the measured output register after the algorithm terminates, assuming the initial state contained $|\phi\rangle$ in the work register (with ancilla padding) instead of being $|\psi_{init}\rangle$. That is, $Q(x, |\phi\rangle)$ is a Bernoulli random variable corresponding to the measurement outcome of the output register of the final state*

$$U_r U^x U_{r-1} U^x \ldots U_1 U^x U_0 |\phi\rangle |pad\rangle ,$$

where $|pad\rangle$ is the ancilla padding.

▶ **Definition 6** (Query QMA and QCMA). *Let f be a possibly partial Boolean function on n bits, and let Q be a quantum query algorithm on n bits with T total queries. We say that Q is a QMA algorithm for f with witness size k if the following holds:*

1. (Soundness). *For every $x \in f^{-1}(0)$ and every k-qubit state $|\phi\rangle$, we have $\Pr[Q(x, |\phi\rangle) = 1] \leq \epsilon$.*
2. (Completeness). *For every $x \in f^{-1}(1)$, there exists a k-qubit state $|\phi\rangle$ such that $\Pr[Q(x, |\phi\rangle) = 1] \geq 1 - \delta$.*

Here, ϵ and δ govern the soundness and completeness of Q; by default, we take them both to be $1/3$. We denote the QMA query complexity of f by $\mathrm{QMA}_{\epsilon,\delta}(f)$, which is the minimum possible value of $T + k$ over any QMA algorithm for f with the specified soundness and completeness.

We say that Q is a QCMA algorithm for f if the same conditions hold, except with the witness state $|\phi\rangle$ quantifying over only classical k-bit strings in both the soundness and completeness conditions. We define $\mathrm{QCMA}_{\epsilon,\delta}(f)$ *analogously to* $\mathrm{QMA}_{\epsilon,\delta}(f)$*, and we omit the subscripts when they are both* $1/3$*.*

▶ **Definition 7** (Bounded round query QMA and QCMA)**.** *We define r-round QMA and QCMA in exactly the same way as the above definition, except the query algorithms are required to have at most r rounds. We use* $\mathrm{QMA}^r_{\varepsilon,\delta}(f)$ *and* $\mathrm{QCMA}^r_{\varepsilon,\delta}(f)$ *to denote the r-round QMA and QCMA query complexities of f respectively.*

▶ **Definition 8** (Function family)**.** *A* function family *is an indexed set $F = \{f_n\}_{n\in I}$ where $I \subseteq \mathbb{N}$ is an infinite set and where each f_n is a partial Boolean function $f_n\colon \mathrm{Dom}(f_n) \to \{0,1\}$ with $\mathrm{Dom}(f_n) \subseteq \{0,1\}^n$.*

▶ **Definition 9** (Efficiently computable QMA)**.** *Let $F = \{f_n\}_{n\in I}$ be a function family. We say that F is in* efficiently computable query QMA *if there is a polynomial-time Turing machine which takes in the binary encoding $\langle n\rangle$ of a number $n \in I$ and outputs a QMA verifier Q by explicitly writing out the unitaries of Q as quantum circuits (with a fixed universal gate set). The verifier Q must be sound and complete for f_n. Efficiently computable bounded-round QMA is defined analogously.*

In other words, $\mathrm{QMA}(f_n)$ *must be* $O(\mathrm{polylog}(n))$*, and moreover, the different algorithms for f_n must be uniformly generated by a single polynomial-time Turing machine.*

With these definitions, we show in the full version that Theorem 2 implies Theorem 1.

2.2 Error-correcting codes

A Reed-Solomon error-correcting code $\mathrm{RS}_{q,\gamma,k}$ over $\mathbb{F}_q$, with degree parameter $0 < k < q-1$ and generator $\gamma \in \mathbb{F}_q^*$, is defined as

$$\mathrm{RS}_{q,\gamma,k} = \{(f(\gamma), \dots f(\gamma^q)) : f \in \mathbb{F}_q[x]_{\deg\le k}\},$$

where $\mathbb{F}_q[x]_{\deg\le k}$ is the set of polynomials over $\mathbb{F}_q$ of degree at most k.

Let $q-1 = mn$, for some integers m and n. The m-folded version $\mathrm{RS}^{(m)}_{q,\gamma,k}$ of $\mathrm{RS}_{q,\gamma,k}$ is a mapping of the code to the larger alphabet $\mathbb{F}_q^m$ as follows:

$$\mathrm{RS}^{(m)}_{q,\gamma,k} = \{((x_1,\dots,x_m),\dots,(x_{q-m},\dots,x_q)) : (x_1,\dots,x_q) \in \mathrm{RS}_{q,\gamma,k}\}.$$

Note that the alphabet of $\mathrm{RS}^{(m)}_{q,\gamma,k}$ is $\mathbb{F}_q^m$.

▶ **Definition 10.** *We say that a code $C \subseteq \Sigma^n$ is combinatorially (ζ,ℓ,L)-list recoverable if for any subsets $S_i \subseteq \Sigma$ such that $|S_i| \le \ell$, we have,*

$$|\{(x_1,\dots,x_n) \in C : |\{i : x_i \in S_i\}| \ge (1-\zeta)n\}| \le L.$$

▶ **Lemma 11** ([13, 14])**.** *For a prime power q such that $mn = q-1$, any generator $\gamma \in \mathbb{F}_q^*$, and degree $k < q-1$, $\mathrm{RS}^{(m)}_{q,\gamma,k}$ is (ζ,ℓ,q^s)-list recoverable for some $s \le m$ if there exists an integer r such that the following inequalities hold:*

$$(1-\zeta)n(m-s+1) \ge \left(1+\frac{s}{r}\right)(mn\ell k^s)^{1/(s+1)} \tag{1}$$

$$(r+s)\left(\frac{mn\ell}{k}\right)^{1/(s+1)} < q. \tag{2}$$

▶ **Corollary 12.** *Let m be $\Theta(n)$ integer such that $nm+1=q$ is a prime power. Let $k=\frac{5}{6}mn$ and let c,d be constants. Then $\mathrm{RS}^{(m)}_{q,\gamma,k}$ is $(c\log n/n, 2^{(\log n)^d}, 2^{(\log n)^{d+2}})$-list recoverable.*

This corollary is proved simply by checking that the equations (1)–(2) are satisfied with this choice of parameters. The choice of parameters is in fact the same as those as [14]. Therefore, the above code satisfies the other conditions required for the [14] quantum algorithm to succeed in evaluating the relation $R_{C,f}$ defined in the next section.

3 The Yamakawa-Zhandry problem

For a function $f:[n]\times\{0,1\}^m\to\{0,1\}$ and a linear code $C\subseteq\{0,1\}^{nm}$, define the relation $R_{C,f}\subseteq\{0,1\}^n\times\{0,1\}^{nm}$

$$R_{C,f}=\{(x,u)=(x_1\dots x_n,u_1\dots u_n):(u_1\dots u_n\in C)\wedge(\forall i\, f(i,u_i)=x_i)\}.$$

We shall typically work with $m=\Theta(n)$. We shall usually work with a fixed code C, in which case we shall omit the subscript C from $R_{C,f}$.

Let $\mathcal{P}\subseteq\{0,1,*\}^{n2^m}$ denote the set of polynomial-sized partial assignments for functions $f:[n]\times\{0,1\}^m\to\{0,1\}$. We define the extended version $\widetilde{R}_C$ of $\{R_{C,f}\}_f$ over $\mathcal{P}\times\{0,1\}^n\times\{0,1\}^{nm}$ as follows:

$$\widetilde{R}_C=\{(p,x,u):p\text{ is the minimal partial assignment s.t. }(x,u)\in R_{C,f}\,\forall f\supseteq p\}.$$

In particular, (p,x,u) is in $\widetilde{R}_C$ when p is the partial assignment $(f(i,u_i)=x_i)_i$, which is n bits.

▶ **Definition 13.** *Let $\widetilde{R}_n$ be a sequence of relations on $\mathcal{P}_n\times\{0,1\}^n\times\{0,1\}^{\mathrm{poly}(n)}$, where $\mathcal{P}_n$ consists of fixed polynomial-sized partial assignments for $N=2^{\Omega(n)}$-bit strings, and $\mathrm{poly}(n)$ is some fixed polynomial. We say $\widetilde{R}_n$ is $(\eta,s(n),t(n))$-slippery w.r.t. distribution μ on f if for any partial assignment q on N bits with size at most $s(n)$, if we fix the bits of q in f and generate the other bits of f according to μ (conditioned on q), we will have*

$$\left|\bigcup_{\substack{(p,x,u)\in\widetilde{R}_n,\\ \Pr_{f\sim\mu}[p\subseteq f|q\subseteq f]\geq\eta}}\mathrm{supp}(p)\setminus\mathrm{supp}(q)\right|\leq t(n).$$

We omit mentioning the distribution μ explicitly if it is the uniform distribution.

▶ **Lemma 14.** *When C is a code with parameters from Corollary 12, then for $c=\mathrm{polylog}(n)$ and $d=o(\log n/\log\log n)$, $\widetilde{R}_C$ is $(\frac{1}{n^c},2^{(\log n)^d},c\log n\cdot 2^{(\log n)^{d+2}})$-slippery.*

Proof. Let q be a partial assignment of size $2^{(\log n)^d}$. For each $i\in[n]$, let $S_i=\{v:(i,v)\text{ is fixed in }q\}$. Clearly for each i, $|S_i|\leq 2^{(\log n)^d}$. By Corollary 12,

$$C_q=|\{u_1\dots u_n\in C:|\{i:u_i\in S_i\}|\geq n-c\log n\}|\leq 2^{(\log n)^{d+2}}.$$

A tuple (p,x,u) can satisfy $(p,x,u)\in\widetilde{R}_C$ and $\Pr_{f\sim U}[p\subseteq f|q\subseteq f]$ only if $u\in C_q$, so we only need to compute $|\bigcup\mathrm{supp}(p)|$ for such tuples. In fact we only need to worry about the number of (x,u) pairs that could be in $R_{C,f}$, since p is completely fixed by x and u. Each u has at most $c\log n$ many locations that are not fixed by q, and x can take any value in those $c\log n$ locations. The x-s taking different values in these locations have overlapping

p-s (i.e., the same bits are fixed to different values for the different x-s), and since we only care about $|\bigcup \mathrm{supp}(p)|$, we need not count these x-s separately. Therefore, the number of unique indices fixed by such p-s is determined only by the number of u-s in C_q.

Since the total support of each p is outside of q is $c \log n$, we have,

$$\left| \bigcup_{\substack{(p,x,u)\in \widetilde{R}_n, \\ \Pr_f[p\subseteq f | q \subseteq f] \geq \frac{1}{n^c}}} \mathrm{supp}(p) \setminus \mathrm{supp}(q) \right| \leq 2^{(\log n)^{d+2}} \cdot c \log n. \quad \blacktriangleleft$$

▶ **Corollary 15.** *If μ is a distribution such that for all partial assignments p with $|p| = n$, we have $\mu|_q[p] \leq k \cdot u|_q[p]$ (where $\mu|_q[p]$ is the probability mass of strings consistent with p under μ conditioned on q, and $u|_q[p]$ is the same with the uniform distribution), then $\widetilde{R}_C$ from Lemma 14 is also $(\frac{k}{n^c}, 2^{(\log n)^d}, c \log n \cdot 2^{(\log n)^{d+2}})$-slippery w.r.t. μ.*

Proof. Since $\mu[p] \leq k \cdot u|_q[p]$ for all p, partial assignments that have probability at least $\frac{k}{n^c}$ against μ conditioned on q have probability at least $\frac{1}{n^c}$ against the uniform distribution conditioned on q. Now we can apply Lemma 14. $\blacktriangleleft$

▶ **Theorem 16.** *There exists a code C such that*

1. *$\widetilde{R}_C$ is $(\frac{1}{n^c}, 2^{(\log n)^d}, c \log n \cdot 2^{(\log n)^{d+2}})$-slippery for $c = \mathrm{polylog}(n)$ and $d = o(\log n / \log\log n)$.*
2. *There exists a quantum advice $|z_f\rangle$ with polynomially many qubits, and a polynomial-time quantum algorithm $\mathcal{A}$ that has access to $|z_f\rangle, x$, and makes no queries to any oracle, such that for any $x \in \{0,1\}^n$,*

$$\Pr_{f\sim U}[(u \leftarrow \mathcal{A}(|z_f\rangle, x)) \wedge ((x,u) \in R_{C,f})] \geq 1 - 2^{-\Omega(n)},$$

where the probability is over uniformly random functions $f : [n] \times \{0,1\}^m \to \{0,1\}$, and the internal randomness of $\mathcal{A}$.

Proof. Item 1 is due to Lemma 14. As stated before, the problem $\widetilde{R}_C$, and the choice of parameters for the code C in Lemma 14, is the same as [14]. Therefore, item 2 is due to [14, 10].[5] $\blacktriangleleft$

4 Techniques for bounded-round quantum query algorithms

In this section, we prove some results about bounded-round quantum query algorithms that will be useful in proving our QCMA lower bound.

Recall that a non-adaptive quantum algorithm works on T query-input registers and T query-output registers plus an additional work register W, so that its basis states look like

$$|i_1\rangle |b_1\rangle |i_2\rangle |b_2\rangle \ldots |i_T\rangle |b_T\rangle |W\rangle .$$

To clear up notational clutter, we will use $\vec{i} \in [N]^T$ to represent a tuple of T indices in $[N]$. Moreover, for a string $x \in \{0,1\}^N$ and for $\vec{i} \in [N]^T$, we will define $x_{\vec{i}} := (x_{\vec{i}_1}, x_{\vec{i}_2}, \ldots, x_{\vec{i}_T})$.

[5] The quantum algorithm in [14] makes some non-adaptive quantum queries (not depending on x), and does not take an advice state. The modified algorithm, which instead takes an advice state (which is essentially the state of the algorithm in [14] after its non-adaptive queries) and makes no queries, was described in [10].

The basis states can then be written $|\vec{i}\rangle\,|\vec{b}\rangle\,|W\rangle$, and the action of the query unitary U^x to the string x is to map $|\vec{i}\rangle\,|\vec{b}\rangle\,|W\rangle \to |\vec{i}\rangle\,|\vec{b} \oplus x_{\vec{i}}\rangle\,|W\rangle$, extended linearly to the rest of the space. (Here $\oplus$ denotes the bitwise XOR of the two strings of length T.)

Define $\Pi_{\vec{i}} := |\vec{i}\rangle\langle\vec{i}| \otimes I_{\vec{b},W}$ to be the projection onto basis states with $\vec{i}$ in the query-input registers. For $i \in [N]$, define $\Pi_i := \sum_{\vec{i} \ni i} \Pi_{\vec{i}}$ to be the projection onto basis states with i occurring in one of the query-input registers. The projections $\Pi_{\vec{i}}$ are onto orthogonal spaces, though the projections Π_i are not. Observe that $\sum_{\vec{i}} \Pi_{\vec{i}} = I$, and that $\sum_i \Pi_i = \sum_i \sum_{\vec{i} \ni i} \Pi_{\vec{i}} = \sum_{\vec{i}} \sum_{i \in \vec{i}} \Pi_{\vec{i}} = T \cdot I$. Moreover, since the oracle unitary U^x does not change the query-input registers, U^x commutes with both $\Pi_{\vec{i}}$ and Π_i. Another convenient property is that if $x_{\vec{i}} = y_{\vec{i}}$ for two strings $x, y \in \{0,1\}^N$, then $\Pi_{\vec{i}}(U^x - U^y) = 0$; this holds because both U^x and U^y map $|\vec{i}\rangle\,|\vec{b}\rangle\,|W\rangle$ to the same vector when $x_{\vec{i}} = y_{\vec{i}}$. Using these properties, we have the following lemma.

▶ **Lemma 17** (Hybrid argument for nonadaptive queries). *For any strings $x, y \in \{0,1\}^N$ and any quantum state $|\psi\rangle = \sum_{\vec{i},\vec{b},W} \alpha_{\vec{i},\vec{b},W}\,|\vec{i}\rangle\,|\vec{b}\rangle\,|W\rangle$, we have*

$$\| U^x\,|\psi\rangle - U^y\,|\psi\rangle \,\|_2^2 \le 4 \sum_{i : x_i \neq y_i} \| \Pi_i\,|\psi\rangle \,\|_2^2.$$

Proof. We write the following, with justification afterwards.

$$\begin{aligned}
\| U^x\,|\psi\rangle - U^y\,|\psi\rangle \,\|_2^2 &= \left\| \sum_{\vec{i}} \Pi_{\vec{i}}(U^x - U^y)\,|\psi\rangle \right\|_2^2 \\
&= \sum_{\vec{i}} \| \Pi_{\vec{i}}(U^x - U^y)\,|\psi\rangle \,\|_2^2 \\
&= \sum_{\vec{i} : x_{\vec{i}} \neq y_{\vec{i}}} \| \Pi_{\vec{i}}(U^x - U^y)\,|\psi\rangle \,\|_2^2 \\
&\le \sum_{\vec{i}} \sum_{i \in \vec{i} : x_i \neq y_i} \| \Pi_{\vec{i}}(U^x - U^y)\,|\psi\rangle \,\|_2^2 \\
&= \sum_{i : x_i \neq y_i} \sum_{\vec{i} \ni i} \| \Pi_{\vec{i}}(U^x - U^y)\,|\psi\rangle \,\|_2^2 \\
&= \sum_{i : x_i \neq y_i} \left\| \sum_{\vec{i} \ni i} \Pi_{\vec{i}}(U^x - U^y)\,|\psi\rangle \right\|_2^2 \\
&= \sum_{i : x_i \neq y_i} \| \Pi_i(U^x - U^y)\,|\psi\rangle \,\|_2^2 \\
&= \sum_{i : x_i \neq y_i} \| (U^x - U^y)\Pi_i\,|\psi\rangle \,\|_2^2 \\
&\le 4 \sum_{i : x_i \neq y_i} \| \Pi_i\,|\psi\rangle \,\|_2^2.
\end{aligned}$$

In the first line, we used $\sum_{\vec{i}} \Pi_{\vec{i}} = I$. In the second, we used the orthogonality of the images of the projections $\Pi_{\vec{i}}$. In the third, we used $\Pi_{\vec{i}}(U^x - U^y) = 0$ when $x_{\vec{i}} = y_{\vec{i}}$.

In the fourth line, we replaced the sum over $\vec{i}$ containing at least one i with $x_i \neq y_i$ with a weighted sum, where the weight of $\vec{i}$ is the number of $i \in \vec{i}$ such that $x_i \neq y_i$; this weight is 0 when $x_{\vec{i}} = y_{\vec{i}}$ and at least 1 when $x_{\vec{i}} \neq y_{\vec{i}}$. This weight can be represented as a sum over $i \in \vec{i}$ with $x_i \neq y_i$, since we are counting $\vec{i}$ once for each such i in the tuple.

The fifth line flips the order of the sums, and the sixth uses orthogonality of the images of $\Pi_{\vec{i}}$ to put the sum back inside the squared norm. The seventh line is the definition of Π_i, and the eighth holds since Π_i commutes with U^x and U^y. Finally, the last line follows either from the triangle inequality, or from the fact that the spectral norm of $(U^x - U^y)$ is at most 2 (since U^x and U^y are unitary). ◀

For an oracle $x \in \{0,1\}^n$ and a block $B \subseteq [N]$, use $x[B]$ to denote the string x with queries in B erased; that is, $x[B]_i = x_i$ if $i \notin B$, and $x[B]_i = 0$ for $i \in B$. Next, we use this hybrid argument in combination with a Markov inequality to show that if a distribution μ over $\{0,1\}^n$ has a set of queries $B \in [N]$ that nearly always return zero for oracles sampled from μ, then for any non-adaptive quantum algorithm, there exists a large set of oracles (measured against μ) such that the algorithm does not detect whether any subset of B is erased.

▶ **Lemma 18** (Nonadaptive algorithms don't detect oracle erasures)**.** *Fix $|\psi\rangle$ representing the state of a quantum algorithm before a batch of non-adaptive queries. Let μ be a distribution over $\{0,1\}^N$, and let $\epsilon > 0$. Let $B = \{i \in [N] : \Pr_{x\sim\mu}[x_i = 1] \leq \epsilon\}$. Then there exists a set $S \subseteq \{0,1\}^N$ such that $\mu[S] \geq 1/2$ and for all $x \in S$ and all subsets $B_1, B_2 \subseteq B$, we have*

$$\| U^{x[B_1]} |\psi\rangle - U^{x[B_2]} |\psi\rangle \|_2 \leq \sqrt{8\epsilon T}.$$

Proof. We write the following, with justification afterwards.

$$\begin{aligned}
\mathop{\mathbb{E}}_{x\sim\mu}\left[\sum_{i:x_i\neq x[B]_i} \|\Pi_i |\psi\rangle\|_2^2\right] &= \mathop{\mathbb{E}}_{x\sim\mu}\left[\sum_{i\in B} x_i \|\Pi_i |\psi\rangle\|_2^2\right] \\
&= \sum_{i\in B} \|\Pi_i |\psi\rangle\|_2^2 \mathop{\mathbb{E}}_{x\sim\mu}[x_i] \\
&\leq \epsilon \sum_{i\in B} \|\Pi_i |\psi\rangle\|_2^2 \\
&\leq \epsilon \sum_{i\in[N]} \|\Pi_i |\psi\rangle\|_2^2 \\
&= \epsilon \sum_{i\in[N]} \sum_{\vec{i}\ni i} \|\Pi_{\vec{i}} |\psi\rangle\|_2^2 \\
&= \epsilon T \sum_{\vec{i}} \|\Pi_{\vec{i}} |\psi\rangle\|_2^2 \\
&= \epsilon T.
\end{aligned}$$

The first line follows by noting that $x_i \neq x[B]_i$ can only happen if both $i \in B$ and $x_i = 1$; we replace the sum over $i : x_i \neq x[B]_i$ with the sum over $i \in B$, and multiply the summand by the indicator for $x_i = 1$, which is x_i itself.

The second line is the result of pushing the expectation inside the sum, and observing that the norm does not depend on x and can be factored out of the expectation. The third line follows from the definition of B: we know that for all $i \in B$, the probability of $x_i = 1$ is at most ϵ. The fourth replaces the sum over B with that over $[N]$. The fifth uses the definition of Π_i, and exchanges the sum over $\vec{i}$ with the squared norm using orthogonality. The sixth line follows by noting that each $\vec{i}$ appears exactly T times in this double sum. Finally, the last line follows by pushing the sum inside the squared norm (using orthogonality), and recalling that $\sum_{\vec{i}} \Pi_{\vec{i}} = I$, together with the fact that $|\psi\rangle$ is a unit vector.

Given this bound on the expectation, we can apply Markov's inequality to conclude that at least half the strings x (weighted by μ) must satisfy $\sum_{i:x_i \neq x[B]_i} \|\Pi_i |\psi\rangle\|_2^2 \leq 2\epsilon T$. Let S be the set of such strings x; then $\mu[S] \geq 1/2$. Observe that for any $x \in S$ and any $B_1, B_2 \subseteq B$, the set $\{i : x[B_1]_i \neq x[B_2]_i\}$ is a subset of $\{i : x_i \neq x[B]_i\}$. We now apply Lemma 17 to get

$$\|U^{x[B_1]} |\psi\rangle - U^{x[B_2]} |\psi\rangle\|_2^2 \leq 4 \sum_{i:x[B_1]_i \neq x[B_2]_i} \|\Pi_i |\psi\rangle\|_2^2 \leq 4 \sum_{i:x_i \neq x[B]_i} \|\Pi_i |\psi\rangle\|_2^2 \leq 8\epsilon T.$$

The desired result follows by taking square roots. ◀

5 QMA vs QCMA

In this section, we prove Theorem 2. Theorem 19 will define the function F_N and show that it is in QMA, and Theorem 21 will show that it is not in QCMA.

5.1 Construction and QMA protocol

Fix a code C for which Theorem 16 holds, with $c = \log n$. We shall henceforth refer to $R_{C,f}$ as only R_f for this C. For a subset $E \subseteq \{0,1\}^n$, define the oracle $O[f, E]\colon \{0,1\}^n \times \{0,1\}^{nm} \to \{0,1\}$ as

$$O[f,E](x,u) = \begin{cases} 1 & \text{if } (x,u) \in R_f \wedge x \notin E \\ 0 & \text{otherwise.} \end{cases}$$

▶ **Theorem 19.** *There exists an efficient uniform collection of query QMA protocols (generated uniformly by a polynomial time Turing machine) which uses 1 query and polynomial witness size, and which outputs 0 on all oracles $O[f, E]$ with $|E| \geq (2/3) \cdot 2^n$, and outputs 1 on $O[f, \emptyset]$ for $1 - 2^{-\Omega(n)}$ fraction of f-s.*

Proof. The quantum witness for the algorithm will be quantum advice state for R_f from Theorem 16. The quantum algorithm works as follows: it samples a uniformly random $x \in \{0,1\}^n$, and runs the procedure from Theorem 16 to find a u such that $(x,u) \in R_f$. Note that this requires no queries to the oracle. Then it queries the oracle at (x,u) and returns the query output. If the oracle is $O[f,\emptyset]$ and the actual state $|z_f\rangle$ from Theorem 16 is provided as witness, then due to Theorem 16 we have,

$$\Pr_{f \sim U}[\mathcal{A}^{O[f,\emptyset]}(|z_f\rangle) = 1] \geq 1 - 2^{-\Omega(n)}.$$

On the other hand, if the oracle is $O[f,E]$ for $|E| \geq \frac{2}{3} \cdot 2^n$, no matter what witness is provided, and what u is obtained from this witness, the oracle outputs 0 on (x,u) for $\frac{2}{3}$ of the x-s. Since the algorithm samples a uniformly random x and queries it with some u for every f, we have for every f,

$$\Pr[\mathcal{A}^{O[f,E]}(|z_f\rangle) = 1] \leq \frac{1}{3}.$$ ◀

Defining the function F_N. We now define the following partial query function with input size $2^n \times 2^{mn}$: its 1-inputs are all the oracles $O[f,\emptyset]$ for which the algorithm from Theorem 19 accepts with probability at least $2/3$, and its 0-inputs are $O[f,E]$ for which $O[f,\emptyset]$ is a 1-input and $|E| \geq (2/3) \cdot 2^n$. Note that these oracles correspond to the inputs "x" of the query problem. This defines a family F_N of query tasks with $N = 2^n \times 2^{mn}$, and Theorem 19 showed that this family is in efficiently-computable QMA.

5.2 Densification of probability distributions

To prove our QCMA lower bound, we will need some properties of distributions on $\{0,1\}^N$. For such a distribution μ, let $\mathrm{RU}(\mu) \coloneqq \max_{x\in\{0,1\}^N} \log_2(2^N\mu[x])$ be the max relative entropy of μ relative to the uniform distribution. We will generally be interested in distributions μ such that $\mathrm{RU}(\mu)$ is small (say, $\mathrm{polylog}\, N$), which means that no input $x \in \{0,1\}^N$ has probability $\mu[x]$ much larger than 2^{-N}.

For a partial assignment p, let $\mu[p]$ be the probability mass of strings in $\{0,1\}^N$ which are consistent with p. Let $|p|$ be the size of p (the number of revealed bits in p). We define the density of μ to be $\mathrm{density}(\mu) \coloneqq 1 - \max_p \frac{\log_2(2^{|p|}\mu[p])}{|p|}$, with the maximum taken over partial assignments p. The density of the uniform distribution is 1.

For a partial assignment p, we let $\mu|_p$ denote the distribution μ conditioned on the sampled input being consistent with p. Items 1 and 3 of the following lemma essentially follow from results in [8, 6]. We produce a proof here because the version of the lemma we need is simpler than what was shown in [8, 6].

▶ **Lemma 20** (Densification). *Let μ be a distribution over $\{0,1\}^N$, and let $\delta \in (0,1)$. Then there exists a partial assignment p such that*

1. $|p| \le \mathrm{RU}(\mu)/\delta$
2. $\mathrm{RU}(\mu|_p) \le \mathrm{RU}(\mu)/\delta$
3. $\mathrm{density}(\mu|_p) > 1-\delta$, *where the density is measured on the bits not fixed by p.*

Proof. Let p be the largest partial assignment (we can pick the lexicographically first one according to some ordering, if there is a tie) for which $\mu[p] \ge 2^{-(1-\delta)|p|}$. Then

$$2^{-(1-\delta)|p|} \le \mu[p] = \sum_{x\supseteq p} \mu[x] \le 2^{N-|p|}\cdot 2^{-(N-\mathrm{RU}(\mu))} = 2^{\mathrm{RU}(\mu)-|p|},$$

so $\delta|p| \le \mathrm{RU}(\mu)$, from which the first item follows. Next,

$$\mathrm{RU}(\mu|_p) = \max_x \log_2(2^N\mu|_p[x]) = \max_{x\supseteq p} \log_2(2^N\mu[x]/\mu[p]) \le \mathrm{RU}(\mu) + \log_2(1/\mu[p])$$

$$\le \mathrm{RU}(\mu) + \log_2(2^{(1-\delta)|p|}) = \mathrm{RU}(\mu) + (1-\delta)|p| \le \mathrm{RU}(\mu) + (1-\delta)\,\mathrm{RU}(\mu)/\delta = \mathrm{RU}(\mu)/\delta,$$

which gives the second item. Finally, to upper bound the density of $\mu|_p$, let q be a partial assignment on a set of indices disjoint from that of p. By the maximality of p, we must have $\mu[p\cup q] < 2^{-(1-\delta)(|p|+|q|)}$. Now,

$$\log_2(2^{|q|}\mu|_p[q]) = \log_2(2^{|q|}\mu[q\cup p]/\mu[p]) < \log_2(2^{|q|}2^{-(1-\delta)(|p|+|q|)}/2^{-(1-\delta)|p|}) = \delta|q|.$$

From this it follows that $\mathrm{density}(\mu|_p) > 1-\delta$, as desired. ◀

5.3 QCMA lower bound

▶ **Theorem 21.** *There is no bounded-round, polynomial-cost QCMA protocol for the family F_N defined in Section 5.1. More formally, consider any family of QCMA protocols for the query problems F_N. If the number of rounds for these QCMA protocols grows slower than $o(\log\log N/\log\log\log N)$, then either the number of queries or the witness size must grow like $\log^{\omega(1)} N$.*

We will prove this theorem by a sequence of claims. The idea of the proof will be to remove the rounds of the algorithm one by one. We start by moving from QCMA to BQP via the following claim.

▷ Claim 22. If there is a QCMA protocol for F_N with witness size $k = k(N)$, then there is a quantum algorithm Q and a large set of functions S such that Q accepts all oracles $O[f, \emptyset]$ for $f \in S$ and rejects all oracles $O[f, E]$ for $f \in S$ and $|E| \geq (2/3)2^n$. The set S is large enough that the uniform distribution μ over S has $\mathrm{RU}(\mu) \leq 2k$. The algorithm Q makes the same number of rounds and number of queries as the QCMA protocol. By "accepting" and "rejecting", we mean with probability at least 2/3.

Proof. The idea is just to take a witness w that works for as many 1-inputs as possible, and hard-code this witness into the quantum algorithm. S will correspond to the set of 1-inputs on which this witness works.

More explicitly, since the witness is a classical string, there are only 2^k witnesses over which we quantify. Since each 1-input $O[f, \emptyset]$ has some witness accepting it, we conclude that at least one witness w of size k is a valid witness for at least a 2^{-k} fraction of the 1-inputs, and hence also for at least a $2^{-k}(1 - 2^{-\Omega(n)})$ fraction of all oracles $O[f, \emptyset]$ (including those not in the domain of F_N). This is because the fraction of f-s for which the quantum algorithm does not succeed with probability at least 2/3 is at most $2^{-\Omega(n)}$. We can assume $2^{-k}(1 - 2^{-\Omega(n)}) \geq 2^{-2k}$.

Let S be the set of f such that $O[f, \emptyset]$ is accepted by the algorithm given witness w. Let μ be the uniform distribution over S, and observe that $\mathrm{RU}(\mu) \leq 2k$. Let Q be the quantum algorithm which hard-codes the witness w into the verifier; then Q accepts all oracles $O[f, \emptyset]$ for $f \in \mathrm{supp}(\mu)$ and rejects all oracles $O[f, E]$ if $|E| \geq (2/3)2^n$. ◁

Defining the round reduction. Given a pair (Q, μ) of a quantum algorithm and a distribution over functions, we wish to define a pair $(\tilde{Q}, \tilde{\mu})$ such that $\tilde{Q}$ has one less round than Q, $\mathrm{supp}(\tilde{\mu})$ is a subset of $\mathrm{supp}(\mu)$ but "not by much" (i.e. $\mathrm{RU}(\tilde{\mu})$ is not much larger than $\mathrm{RU}(\mu)$), and the two algorithms behave similarly on $\tilde{\mu}$.

To define $(\tilde{Q}, \tilde{\mu})$ given (Q, μ), we proceed in several steps.

1. First, use Lemma 20 with $\delta = 1/n$ to find a partial assignment q with $|q| \leq n\,\mathrm{RU}(\mu)$, $\mathrm{RU}(\mu|_q) \leq n\,\mathrm{RU}(\mu)$, and with $\mu|_q$ being $(1 - \delta)$-dense on the bits not used by q.
2. Second, use Lemma 18 with $\epsilon = 1/3200r^2T$ on the distributions of oracles $O[f, \emptyset]$ when f is sampled from $\mu|_q$. The state $|\psi\rangle$ in the lemma will be the state of the algorithm Q just before the first batch of T queries. The lemma gives a set $S \subseteq \mathrm{supp}(\mu|_q)$ with $\mu|_q[S] \geq 1/2$. It has the property that for all $f \in S$ and all sets B_1, B_2 containing pairs (x, u) with $\Pr_{f \sim \mu|_q}[O[f, \emptyset](x, u) = 1] \leq \epsilon$, we have $\| U^{O[f,B_1]} |\psi\rangle - U^{O[f,B_2]} |\psi\rangle \|_2 \leq 1/20r$. Condition $\mu|_q$ on the set S to get a distribution μ'.
   Note that $O[f, B_1]$ is an abuse of notation, since normally we erase inputs x to f from the oracle, yet B_1 is a set of pairs (x, u). We will use this abuse of notation throughout; if we write $O[f, B]$ where B is a set of pairs, we mean to erase those pairs from the oracle, while if B is a subset of $\mathrm{Dom}(f)$, we mean to erase the pairs (x, u) for $x \in B$ and all u from the oracle.
3. Third, use the slippery property from Corollary 15 on q to conclude that the number of bits used by partial assignments p for which $(p, x, u) \in \tilde{R}_C$ and $\Pr_{f \sim \mu'}[p \subseteq f | q \subseteq f] \geq \epsilon/4$ is small. Recall that $(p, x, u) \in \tilde{R}_C$ means that the condition $O[f, \emptyset](x, u) = 1$ is equivalent to $p \subseteq f$ for all f; such certifying p have $|p| = n$. Corollary 15 can be applied because $\epsilon/4$ is larger than $1/n^c$ for $c = \log n$, since we are choosing $r = o(\log n / \log\log n)$ and $T \leq O(2^{\log^2 n} / \log n)$. Now, since $\mu|_q$ is $(1 - \delta)$-dense outside of q, the probability of a partial assignment p against $\mu|_q$ is at most $2^{\delta|p|}$ times the probability against the uniform distribution conditioned on q. Here $|p| = n$ and $\delta = 1/n$, so the probability against

$\mu|_q$ is at most twice that against the uniform distribution conditioned on q. Moving from $\mu|_q$ to μ' conditions on a set S of probability at least $1/2$, so it can increase the probability of p by at most a factor of 2. Hence the probability of p against μ' is overall at most 4 times its probability against the uniform distribution conditioned on q. By Corollary 15, we conclude the total number of bits used by partial assignments p for which $\Pr_{f\sim\mu'}[O[f,\emptyset](x,u)=1] \geq \epsilon$ is small. Let Z be the set of all such bits.

Our final modification to μ' will be to fix the bits in Z to the highest-probability partial assignment (measured against μ'), and let $\tilde{\mu}$ be μ' conditioned on this partial assignment.

4. Set $\tilde{Q}$ to be the quantum algorithm which is the same as Q, except that the first batch of queries is made to a fake oracle instead of a real one. The fake oracle is defined as follows: on queries (x,u) for which $O[f,\emptyset](x,u)$ is fixed for all $f \in \mathrm{supp}(\tilde{\mu})$, return this value $O[f,\emptyset](x,u)$; on queries (x,u) for which this value is not fixed for $f \in \mathrm{supp}(\tilde{\mu})$, return 0. Note that the fake oracle does not depend on the true input oracle $O[f,\emptyset]$, so queries to it can be implemented by $\tilde{Q}$ without making queries to the real oracle. This replaces the first round of Q, so $\tilde{Q}$ has one less round.

▷ Claim 23. Let Q and μ be as in Claim 22 (with μ the uniform distribution over S). Let $(Q_0,\mu_0)=(Q,\mu)$, and iteratively define $(Q_\ell,\mu_\ell)=(\tilde{Q}_{\ell-1},\tilde{\mu}_{\ell-1})$ for $\ell=1,2,\ldots,r$, where r is the number of rounds of Q.

Then Q_r makes no queries and μ_r "has large support": $\log \mathrm{RU}(\mu_r) \leq (2\log n)^{2r}\log 2k$ (assuming n is sufficiently large).

Proof. That Q_r makes no queries is clear, since each Q_ℓ in the chain makes one less round of queries than $Q_{\ell-1}$, and since the first algorithm $Q_0=Q$ makes r rounds.

To bound $\mathrm{RU}(\mu_r)$, we need to show that $\log \mathrm{RU}(\mu_{\ell+1})$ is at most a factor of $2\log^2 n$ more than $\log \mathrm{RU}(\mu_\ell)$.

Recall the construction of $\mu_{\ell+1}$ from μ_ℓ. The first step moved from μ_ℓ to $\mu_\ell|_q$ with $\mathrm{RU}(\mu_\ell|_q) \leq n\,\mathrm{RU}(\mu_\ell)$. The second step conditioned the latter distribution on a set S of probability mass at least $1/2$, which can only increase $\mathrm{RU}(\cdot)$ by 1, so $\mathrm{RU}(\mu'_\ell) \leq n\,\mathrm{RU}(\mu_\ell)+1$.

The third step found the set of all bits fixed in partial assignments p which certify some (x,u) as evaluating to 1, and picked the highest-probability partial assignment on those bits. The maximum increase in $\mathrm{RU}(\cdot)$ is the number of bits that were fixed in this way. This number comes from Theorem 16, and depends on the number of bits fixed in q; when $|q| = 2^{(\log n)^d}$, the number we are looking for is $c\log n \cdot 2^{(\log n)^{d+2}}$, so we can express this as $c\log n \cdot 2^{(\log^2 n)(\log|q|)}$. We had $|q| \leq n\,\mathrm{RU}(\mu_\ell)$ and $c=\log n$. It is not hard to see that this additive increase dominates $n\,\mathrm{RU}(\mu_\ell)+1$; assuming everything is large enough (e.g. $\log n$ is sufficiently large, and $\mathrm{RU}(\mu_\ell)$ is at least n^2, which is without loss of generality by restricting the original μ_0 to a smaller set if necessary), we can get the upper bound $\mathrm{RU}(\mu_{\ell+1}) \leq 2^{2\log^2 n \log \mathrm{RU}(\mu_\ell)}$, as desired. ◁

▷ Claim 24. Assume the witness size k is $O(\mathrm{poly}(n))$ and the number of rounds r is $o(\log n/\log\log n)$, and let n be large enough. With notation as in Claim 23, there exists $\hat{f} \in \mathrm{supp}(\mu_r)$ and a large set E (with $|E| \geq (2/3)2^n$) of inputs x such that for every $x \in E$ and every u, the pair (x,u) is "not fixed to 1 by μ_r" (that is, there exists $f \in \mathrm{supp}(\mu_r)$ such that $O[f,\emptyset](x,u)=0$).

Proof. We essentially apply another round-reduction iteration (without the second step) to μ_r. Using Lemma 20, we find a partial assignment q' such that $\mu_r|_{q'}$ is $(1-\delta)$-dense outside of q', with $\delta = 1/n$. We then apply Theorem 16 to conclude there are few pairs (x,u) with $\Pr_f[O[f,\emptyset](x,u)=1] \geq 1/2$, and hence few pairs (x,u) with $\Pr_f[O[f,\emptyset](x,u)=1] = 1$

when f is sampled from $\mu_r|_{q'}$; the number of such pairs is at most $2^{(2\log n)^{2r+2}\log k}$. Using $k = O(\text{poly}(n))$ and $r = o(\log n/\log\log n)$, this means that there are at most $2^{o(n)}$ pairs (x,u) that are fixed to 1 for all the oracles $O[f,\emptyset]$ for $f \in \text{supp}(\mu_r|_{q'})$. Therefore, there are $2^{n-o(n)}$ many inputs x such that for all u, the pair (x,u) is not fixed to 1 by $\text{supp}(\mu_r|_{q'})$. Let E be the set of such x; then $|E| \geq (2/3)2^n$. Let $\hat{f} \in \text{supp}(\mu_r|_{q'})$ be arbitrary, and the desired result follows. ◁

Proof of Theorem 21. Start with a QCMA protocol for f_N, and use Claim 22 to get a Q and μ; to get a contradiction, we just need to find $f \in \text{supp}(\mu)$ and a large set E of inputs x such that Q fails to distinguish the oracle $O[f,\emptyset]$ from the oracle $O[f,E]$.

Let (Q_ℓ, μ_ℓ) be as in Claim 23, and let $\hat{f}$ and E be as in Claim 24. To complete the proof, we just need show that $Q = Q_0$ fails to distinguish $O[\hat{f},\emptyset]$ and $O[\hat{f},E]$.

Let $B = \{(x,u) : x \in E, O[\hat{f},\emptyset](x,u) = 1\}$. Moreover, let B_ℓ be the set of pairs (x,u) which had $\Pr_{f\sim\mu_{\ell-1}|_q}[O[f,\emptyset](x,u) = 1] \leq \epsilon$ in iteration ℓ (where q is the partial assignment from step 1 of iteration ℓ). Note that the pairs not in B_ℓ are all fixed in all the oracles in the support of μ_ℓ, because we choose values for the bits used by their proving partial assignments p. This means that $B \subseteq B_\ell$ for all ℓ. Also, let O_ℓ be the oracle used by Q_ℓ to simulate the first query batch of $Q_{\ell-1}$. Recall that $O_\ell(x,u)$ returns 0 unless (x,u) is fixed to 1 in all $O[f,\emptyset]$ for $f \in \text{supp}(\mu_\ell)$. Since the support of μ_ℓ decreases as a subset in each iteration, the bits fixed in μ_ℓ are also fixed in μ_r, and hence also agree with $\hat{f}$. This means that O_ℓ can be written as an erased oracle $O[\hat{f},A_\ell]$ for some set A_ℓ of pairs (x,u) that were not fixed in μ_ℓ; in other words, $A_\ell \subseteq B_\ell$.

We now note the oracle $O[\hat{f},E]$ is the same as $O[\hat{f},B]$. Additionally, since $B, A_\ell \subseteq B_\ell$, we have by Lemma 18,

$$\|U^{O[\hat{f},B]}|\psi\rangle - U^{O[\hat{f},A_\ell]}|\psi\rangle\|_2 \leq 1/20r$$

where $|\psi\rangle$ is the state right before the first query of the algorithm $Q_{\ell-1}$. This can also be written

$$\|U^{O[\hat{f},E]}|\psi\rangle - U^{O_\ell}|\psi\rangle\|_2 \leq 1/20r.$$

Now, applying additional unitary matrices does not change the 2-norm, and Q_ℓ replaces only the first query of $Q_{\ell-1}$ with O_ℓ and applies the same unitaries as $Q_{\ell-1}$ in all other rounds. If we use $Q_\ell(O)$ to denote the final state of Q_ℓ on the oracle O, we therefore get

$$\|Q_\ell(O[\hat{f},E]) - Q_{\ell-1}(O[\hat{f},E])\|_2 \leq 1/20r.$$

By triangle inequality, we then get

$$\|Q(O[\hat{f},E]) - Q_r(O[\hat{f},E])\|_2 \leq 1/20.$$

Since $\emptyset \subseteq B_\ell$ for all ℓ, the same argument also works to show that

$$\|Q(O[\hat{f},\emptyset]) - Q_r(O[\hat{f},\emptyset])\|_2 \leq 1/20,$$

and of course we also have $Q_r(O[\hat{f},\emptyset]) = Q_r(O[\hat{f},E])$ since Q_r makes no queries. A final application of the triangle inequality gives us

$$\|Q(O[\hat{f},E]) - Q(O[\hat{f},\emptyset])\|_2 \leq 1/10.$$

This gives the desired contradiction, as Q failed to sufficiently distinguish these two oracles (it must accept one with probability at least 2/3 and the other with probability at most 1/3; converting this to a lower bound on the 2-norm distance is a straightforward exercise). ◀

References

1 Scott Aaronson, Harry Buhrman, and William Kretschmer. A Qubit, a Coin, and an Advice String Walk into a Relational Problem. In *15th Innovations in Theoretical Computer Science Conference (ITCS 2024)*, 2024. doi:10.4230/LIPIcs.ITCS.2024.1.

2 Scott Aaronson and Greg Kuperberg. Quantum versus classical proofs and advice. In *Twenty-Second Annual IEEE Conference on Computational Complexity (CCC'07)*, pages 115–128, 2007. doi:10.1109/CCC.2007.27.

3 Dorit Aharonov and Tomer Naveh. Quantum NP - A Survey, 2002. doi:10.48550/arXiv.quant-ph/0210077.

4 Roozbeh Bassirian, Bill Fefferman, and Kunal Marwaha. On the Power of Nonstandard Quantum Oracles. In *18th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2023)*, 2023. doi:10.4230/LIPIcs.TQC.2023.11.

5 Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26:1510–1523, 1997. doi:10.1137/S0097539796300933.

6 Sandro Coretti, Yevgeniy Dodis, Siyao Guo, and John Steinberger. Random oracles and non-uniformity. In *Advances in Cryptology – EUROCRYPT 2018*, 2018.

7 Bill Fefferman and Shelby Kimmel. Quantum vs. Classical Proofs and Subset Verification. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, 2018. doi:10.4230/LIPIcs.MFCS.2018.22.

8 Mika Göös, Toniann Pitassi, and Thomas Watson. Query-to-communication lifting for BPP. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2017. doi:10.1109/FOCS.2017.21.

9 Xingjian Li, Qipeng Liu, Angelos Pelecanos, and Takashi Yamakawa. Classical vs Quantum Advice and Proofs Under Classically-Accessible Oracle. In *15th Innovations in Theoretical Computer Science Conference (ITCS 2024)*, 2024. doi:10.4230/LIPIcs.ITCS.2024.72.

10 Qipeng Liu. Non-uniformity and quantum advice in the quantum random oracle model. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, 2023.

11 Anand Natarajan and Chinmay Nirkhe. A Distribution Testing Oracle Separating QMA and QCMA. In *38th Computational Complexity Conference (CCC 2023)*, 2023. doi:10.4230/LIPIcs.CCC.2023.22.

12 Ran Raz and Avishay Tal. Oracle separation of bqp and ph. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, 2019. doi:10.1145/3313276.3316315.

13 Atri Rudra. *List Decoding and Property Testing of Error Correcting Codes*. PhD thesis, University of Washington, 2007. URL: https://cse.buffalo.edu/faculty/atri/papers/coding/thesis.html.

14 Takashi Yamakawa and Mark Zhandry. Verifiable quantum advantage without structure. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 69–74, 2022. doi:10.1109/FOCS54457.2022.00014.

Two-Sets Cut-Uncut on Planar Graphs

Matthias Bentert
University of Bergen, Norway

Pål Grønås Drange
University of Bergen, Norway

Fedor V. Fomin
University of Bergen, Norway

Petr A. Golovach
University of Bergen, Norway

Tuukka Korhonen
University of Bergen, Norway

Abstract

We study Two-Sets Cut-Uncut on planar graphs. Therein, one is given an undirected planar graph G and two disjoint sets S and T of vertices as input. The question is, what is the minimum number of edges to remove from G, such that all vertices in S are separated from all vertices in T, while maintaining that every vertex in S, and respectively in T, stays in the same connected component. We show that this problem can be solved in $2^{|S|+|T|}n^{\mathcal{O}(1)}$ time with a one-sided-error randomized algorithm. Our algorithm implies a polynomial-time algorithm for the network diversion problem on planar graphs, which resolves an open question from the literature. More generally, we show that Two-Sets Cut-Uncut is fixed-parameter tractable when parameterized by the number r of faces in a planar embedding covering the terminals $S \cup T$, by providing a $2^{\mathcal{O}(r)}n^{\mathcal{O}(1)}$-time algorithm.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases planar graphs, cut-uncut, group-constrained paths

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.22

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2305.01314` [2]

Funding The research leading to these results has received funding from the Research Council of Norway via the project BWCA (grant no. 314528) and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416).

1 Introduction

A *cut* in a graph $G = (V, E)$ is a partitioning (A, B) of V, and we denote by $\text{cut}_G(A)$ the *cut-set*, that is, the set of edges with one endpoint in A and the other in $B = V \setminus A$. For two disjoint sets of vertices S and T, (A, B) is an *S-T-cut* if $S \subseteq A$ and $T \subseteq B$. We study the following variant of the *cut-uncut problem*.

Two-Sets Cut-Uncut

Input:	A graph G, two disjoint terminal sets $S, T \subseteq V(G)$, and an integer $k \geq 0$.
Task:	Decide whether there exists an S-T-cut (A, B) of G with $\lvert\text{cut}_G(A)\rvert \leq k$ such that the vertices of S are in the same connected component of $G[A]$ and the vertices of T are in the same connected component of $G[B]$.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 22; pp. 22:1–22:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Our interest in Two-Sets Cut-Uncut is two-fold. First, Two-Sets Cut-Uncut is a natural optimization variant of the 2-Disjoint Connected Subgraphs problem that received considerable attention from the graph-algorithms and computational-geometry communities [14, 25, 30, 40, 43, 44]. In this problem, one asks whether, for two given disjoint sets $S, T \subseteq V(G)$, one can find disjoint sets $A_1 \supseteq S$ and $A_2 \supseteq T$ such that the subgraphs of G induced by A_i, $i = 1, 2$, are connected. In Two-Sets Cut-Uncut we not only want to decide whether there are disjoint connected sets containing terminal sets S and T, but also minimize the size of the corresponding cut (if it exists). Van 't Hof et al. [44] showed that 2-Disjoint Connected Subgraphs is NP-complete in general graphs, even if $|S| = 2$, and Gray et al. [25] proved that the problem is NP-complete on planar graphs. This implies that Two-Sets Cut-Uncut is also NP-complete on planar graphs.

Second, Two-Sets Cut-Uncut is closely related to the Network Diversion problem, which has been studied extensively by the operations research and networks communities [10, 11, 19, 22, 29, 36]. In this problem, we are given an undirected graph G, two terminal vertices s and t, an edge $b = uv$, and an integer k. The task is to decide whether it is possible to delete at most k edges such that the edge b will become a bridge with s on one side and t on the other. Equivalently, the task is to decide whether there exists a *minimal* s-t-cut of size at most $k + 1$ containing b. While this problem seems very similar to the classic s-t-Minimum Cut problem, the complexity status of this problem (P vs. NP) is widely open. Let us observe that a polynomial-time algorithm for the special case of Two-Sets Cut-Uncut with $|S| = |T| = 2$ implies a polynomial time algorithm for Network Diversion: There are two cases, either s is in the same component as u, or s is in the same component as v, and these correspond to instances of Two-Sets Cut-Uncut with $S = \{s, u\}$ and $T = \{t, v\}$ and $S = \{s, v\}$ and $T = \{t, u\}$, respectively.

Network Diversion has important applications in transportation networks and has therefore also been studied on planar graphs. Cullenbine et al. [10] gave a polynomial-time algorithm for Network Diversion on planar graphs for the special case when both terminals s and t are located on the same face. They posed as an open problem whether this polynomial-time algorithm can be generalized to work on arbitrary planar graphs [10]. Duan et al. put out a preprint [18], which among other results, claims an algorithm resolving Network Diversion on planar graphs in polynomial time, but without a description of the algorithm. We were not able to verify the correctness of the result due to several missing details. The result, however, is an immediate consequence of our main contribution, Theorem 1, establishing the fixed-parameter tractability of Two-Sets Cut-Uncut on planar graphs parameterized by $|S| + |T|$. Theorem 1 also establishes a more general result about fixed-parameter tractability of the problem parameterized by the minimum number of faces r of the graph containing all terminals. (Notice that r never exceeds $|S| + |T|$.)

▶ **Theorem 1.** *There is a one-sided-error randomized algorithm solving Two-Sets Cut-Uncut on planar graphs in $2^{|S|+|T|} \cdot n^{\mathcal{O}(1)}$ time. Moreover, there is a one-sided-error randomized algorithm solving the problem in $2^{\mathcal{O}(r)} \cdot n^{\mathcal{O}(1)}$ time, where r is the number of faces needed to cover $S \cup T$ in a planar embedding.*

Theorem 1 provides the first polynomial-time algorithm for Two-Sets Cut-Uncut on planar graphs for non-singleton S and T. Duan and Xu [19] showed how to solve Two-Sets Cut-Uncut on planar graphs for $|S| = 1$ and $|T| = 2$. This was later extended by Bezáková and Langley [4], who present an $O(n^4)$-time algorithm for $|S| = 1$ and arbitrary T on planar graphs. However, the polynomial time solvability of the case $|S| = |T| = 2$ (which is a generalization of Network Diversion) remained open.

The main tool we develop for showing Theorem 1 is a new algorithmic result about computing shortest paths in group-labeled graphs. We believe this new result to be of independent interest. The group that we consider is the *Boolean group* $(\mathbb{Z}_2^d, +)$, consisting of length-d binary vectors, where the operation $+$ is the component-wise exclusive or (xor). Our algorithm finds a shortest s-t-path in a graph, whose edges are labeled by elements of $(\mathbb{Z}_2^d, +)$ such that the sum of the labels assigned to the edges of the path equals a given value. Furthermore, we impose the constraint that the path can visit certain sets of vertices only once. Formally, we consider the following problem.

Xor-Constrained Shortest Path

Input: A graph G, two vertices s and t, an edge labeling function $g\colon E(G) \to \mathbb{Z}_2^d$, a value $c \in \mathbb{Z}_2^d$, and p sets of vertices $X_1, \ldots, X_p \subseteq V$.

Task: Find an s-t-path P in G that satisfies

(i) $\sum_{e \in E(P)} g(e) = c$, and

(ii) for each $i \in [p]$, $|V(P) \cap X_i| \leq 1$,

and among all such paths minimizes the length.

In Section 3, we give an algorithm for Xor-Constrained Shortest Path that in fact works for general graphs instead of only planar graphs. The result is the following theorem.

▶ **Theorem 2.** *Xor-Constrained Shortest Path can be solved in* $2^{d+p} \cdot (n+m)^{\mathcal{O}(1)}$ *time by a one-sided-error randomized algorithm.*

We call the problem variant where we replace path by cycle in the above problem definition Xor-Constrained Shortest Cycle. We will later show that Theorem 2 directly implies an algorithm for Xor-Constrained Shortest Cycle with the same running time.

The proof of Theorem 2 is based on enhancing the technique introduced by Björklund, Husfeldt, and Taslaman [7] for the T-cycle problem. In T-cycle, the task is to find a shortest cycle that visits a list of specified vertices $T \subseteq V(G)$[1], and Björklund et al. gave a $2^{|T|}n^{\mathcal{O}(1)}$-time algorithm for it. Our algorithm generalizes the algorithm of Björklund et al., because T-cycle can be reduced to Xor-Constrained Shortest Cycle with $d = |T|$ and $p = 0$ as follows. We assign each vertex $v \in T$ to one dimension of $\mathbb{Z}_2^d$, and to enforce that the cycle passes through v, we add a true twin u of v, that is, a vertex adjacent to v with the same neighborhood as v to the graph and assign the edge uv the vector in $\mathbb{Z}_2^d$ that has 1 at only the dimension assigned to v. All other edges are assigned the zero vector $\mathbf{0}$. A cycle evaluating to the all-one vector corresponds to a cycle that visits all vertices in T.

Related work. Besides the closely related work on Network Diversion, 2-Disjoint Connected Subgraphs, and Two-Sets Cut-Uncut that we already mentioned above, let us briefly go through other relevant work.

Two-Sets Cut-Uncut is a special case of Multiway Cut-Uncut, where for a given equivalence relation on the set of terminals, the task is to find a cut (or node-cut) separating terminals according to the relation. This problem is well-studied in parameterized complexity [8, 13, 41]. However, all previous work on parameterized algorithms for Multiway Cut-Uncut has focused on parameterizing by the size of the cut.

[1] The algorithm can also take a list of edges as input by subdividing each target edge and add the new vertex to T.

Multiway Cut is also one of the closest relatives of our problem. Here for a given set of k terminals, one looks for a minimum number of edges separating *all* terminals. On planar graph, the seminal paper of Dahlhaus et al. [15] provides an algorithm of running time $n^{\mathcal{O}(k)}$. Klein and Marx [32] improve the running time to $n^{\mathcal{O}(\sqrt{k})}$ and Marx [37] shows that this running time is optimal (assuming the Exponential Time Hypothesis (ETH)).

The second part of Theorem 1 concerns the parameterization by the number of faces covering the terminal vertices. Such parameterization comes naturally for optimization problems about connecting or separating terminals in planar graphs. In particular, parameterization by the face cover was investigated for Multiway Cut [39], Steiner Tree [3, 31], and various flow problems [21, 23, 35].

Our Theorem 2 belongs to the intersection of two areas around paths in graphs. The first area is about polynomial-time algorithms computing shortest paths in group-labeled graphs [16, 26, 33]. Recently, Iwata and Yamaguchi [28] gave an algorithm for shortest *non-zero* paths in arbitrary group-labeled paths. However, for our purposes, we need an algorithm computing a shortest path whose labels sum to a *specific* element of the group.

The second area is about FPT algorithms for finding paths in graphs satisfying certain properties [6, 7, 24, 20, 34, 45]. As mentioned above, our algorithm for Xor-Constrained Shortest Path can be seen as an extension of the algorithm of Björklund, Husfeldt, and Taslaman [7] for the T-cycle problem to a group-labeled setting.

Organization. The remainder of the article is organized as follows. We start with a general overview of how we achieve our two main results. We then present some notation and necessary definitions in Section 2. Afterwards, we show how to solve Xor-Constrained Shortest Path in Section 3. In Section 4, we apply this algorithm to show Theorem 1 by developing a (randomized) FPT-time algorithm for Two-Sets Cut-Uncut parameterized by the minimum number of faces such that each terminal vertex is incident to at least one such face. Section 5 is devoted to showing that Two-Sets Cut-Uncut is W[1]-hard on general graphs when parameterized by the number of terminals. We conclude in Section 6 with several open problems. Due to space constraints, proofs of some statements are omitted in this extended abstract. These statements are marked ($\star$). The detailed proofs can be found in the full arXiv version of the paper [2]. In the full version, we also present two applications of our FPT-time algorithm to generalize known results from the literature.

1.1 Outline of the Proofs for Theorems 1 and 2

In this section, we outline the proofs of Theorems 1 and 2. For Theorem 1, we first outline the $2^{|S|+|T|} \cdot n^{\mathcal{O}(1)}$-time algorithm for planar Two-Sets Cut-Uncut and then discuss the setting when $S \cup T$ can be covered by at most r faces. Then we consider Theorem 2.

We observe that any optimal solution to Two-Sets Cut-Uncut is an (inclusion-wise) minimal cut in the graph G. Our algorithm is based on the relation between minimal cuts in a planar graph and cycles in its dual graph (see Figure 1). In particular, a set of edges $C \subseteq E(G)$ is a cut-set of a minimal cut in G if and only if in the dual graph G^*, the corresponding set $C^* \subseteq E(G^*)$ is a (simple) cycle. Now, to translate Two-Sets Cut-Uncut into a problem about finding a cycle C^* in G^*, we wish to understand, based on C^*, whether two terminal vertices u and v are on the same side of the cut C in G or on different sides. For this, we observe that if $P \subseteq E(G)$ is the set of edges of an (arbitrary) u-v-path in G and P^* is the corresponding set of edges in G^*, then u and v are on different sides of C if and only if $|C^* \cap P^*|$ is odd.

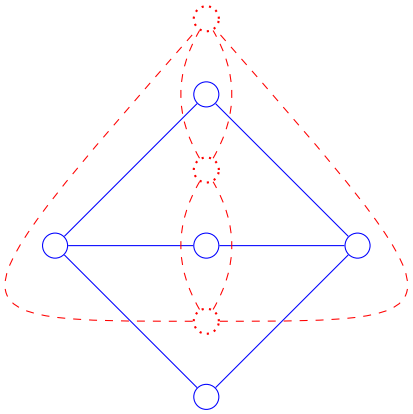


Figure 1 An example of a plane graph (blue) and its dual (multi)graph (dashed/red). Notice that there are bijections between the faces and the vertices, and also between the edges, that is, there is exactly one blue vertex in each red face, one red vertex in each blue face, and each red edge intersects exactly one blue edge and vice versa.

It follows that a constraint stating that u_i and v_i should be on the same/different side of the cut C in G can be expressed as a constraint stating that $|C^* \cap P_i^*|$ should be even/odd for some $P_i^* \subseteq E(G^*)$. By selecting one vertex v in the set of terminals S and writing a "same side" constraint with every other terminal vertex in S and a "different side" constraint with every terminal vertex in T, the Two-Sets Cut-Uncut problem reduces to the problem of finding a shortest cycle C^* in G^* that satisfies $|S| + |T| - 1$ given constraints, each requiring that $|C^* \cap P_i^*| \equiv b_i \pmod 2$ for some $P_i^* \subseteq E(G^*)$ and $b_i \in \{0, 1\}$.

This problem can be equivalently phrased as the Xor-Constrained Shortest Cycle problem with $d = |S| + |T| - 1$, and therefore Theorem 2 indeed implies a $2^{|S|+|T|} \cdot n^{\mathcal{O}(1)}$-time algorithm for Two-Sets Cut-Uncut on planar graphs. Note that here we did not use the condition (ii) in the statement of the Xor-Constrained Shortest Path problem; this condition will be used only for the algorithm utilizing the face cover.

Next, we turn to the more general setting when $S \cup T$ can be covered by at most r faces in a planar embedding of G. First, we observe that by the results of Bienstock and Monma [5], we can decide in $2^{\mathcal{O}(r)} \cdot n$ time whether the input graph has a planar embedding such that the terminals can be covered by at most r faces. Thus, we can assume that G is a plane graph and we are given a face cover of $S \cup T$. Further, we observe that it can be assumed without loss of generality that the input graph is 2-connected. This assumption simplifies arguments because the boundary of each face of a plane 2-connected graph is a cycle [17].

Suppose that f is a face of G that covers some terminals and let C' be the cycle forming the frontier of f. We use the following crucial observation: for the cut-set $C \subseteq E(G)$ of any minimal cut in G separating S and T, it holds that (i) if C' contains vertices of both sets of terminals, then $C \cap E(C')$ separates C' into two connected components (paths) such that each component contains the vertices of exactly one set of terminals, and (ii) if C' contains vertices of one set, then either $C \cap E(C') = \emptyset$ or $C \cap E(C')$ separates C' into two connected components (paths) such that the terminals are in the same component. We use this observation to restrict the behavior of the cycle C^* in G^* corresponding to a potential solution cut-set C. In case (i), we simply delete the edges of G^* that correspond to the edges of C' that should not participate in C (see Figure 2 (a) in the proof of Theorem 1). Case (ii) is more complicated. Suppose that C' contains q terminals. We find q internally vertex disjoint paths $P_1, \ldots, P_q$ in C' whose end-vertices are the terminals. Then we "split"

the vertex f of G^* into q vertices $f_1, \ldots, f_q$ in such a way that each f_i is incident to the edges of G^* corresponding to the edges of P_i (see Figure 2 (b)). However, this splitting would allow a cycle in the dual graph to visit the face f several times. To forbid this, we define $X_f = \{f_1, \ldots, f_q\}$ as used in constraint (ii) of Xor-Constrained Shortest Path and this is the reason why we need constraint (ii) in the problem definition.

We perform the modifications of G^* for all the faces in the cover. This allows us to restrict the number of terminals that we should separate. We pick representatives for each face f in the cover. If the frontier cycle C' of f contains terminals from both sets, we chose one representative from each set from the terminals on C'. If C' contains terminals from one set, we choose one representative. We then apply the same algorithm as for the parameterization by $|S| + |T|$. The difference is that we work only with the representatives and add constraint (ii) to the auxiliary instance of Xor-Constrained Shortest Path given by the sets constructed for the faces from the cover.

We conclude by sketching the main ideas of the algorithm from Theorem 2 for Xor-Constrained Shortest Path. This algorithm works not only on planar graphs but also on general graphs, and it is a generalization of the algorithm by Björklund, Husfeldt, and Taslaman [7] for the T-cycle problem. Our algorithm, like many previous parameterized algorithms for finding paths in graphs [6, 7, 24, 34, 45], exploits the cancellation of monomials in polynomials over fields of characteristic two and randomized polynomial identity testing [42, 46].

The idea of our algorithm is to associate with the input a polynomial over a finite field of characteristic two, and argue that (1) this polynomial is non-zero if and only if a solution exists, and (2) given an assignment of values to variables of the polynomial, the value of the polynomial can be evaluated in $2^{d+p} \cdot n^{\mathcal{O}(1)}$ time.[2] By the DeMillo–Lipton–Schwartz–Zippel lemma [42, 46], the problem can then be solved in $2^{d+p} \cdot n^{\mathcal{O}(1)}$ time by evaluating the polynomial for a random assignment of values. Note that solving the decision version also allows to recover the solution by self-reduction.

In more detail, the polynomial associated with the input is defined as follows. Let us assume that the input graph is a simple graph, as the problem on multigraphs can easily be reduced to simple graphs. For each edge $e \in E(G)$ of the input graph, we associate a variable $f(e)$, and then, for an s-t-walk $W = (e_1, e_2, \ldots, e_\ell)$ of length ℓ, we associate a monomial $f(W) = \prod_{i=1}^{\ell} f(e_i)$. For an integer ℓ, we let $\mathcal{C}_\ell$ denote the set of all s-t-walks of length ℓ that satisfy the conditions (i) and (ii) of the statement of Xor-Constrained Shortest Path, and finally let $f(\mathcal{C}_\ell) = \sum_{W \in \mathcal{C}_\ell} f(W)$ be the polynomial associated with the input. As the monomials of $f(\mathcal{C}_\ell)$ correspond to walks instead of paths, it is not complicated to design a $2^{d+p} \cdot n^{\mathcal{O}(1)}$-time dynamic program for evaluating the value of $f(\mathcal{C}_\ell)$. A more technical part of the proof is to argue that the polynomial $f(\mathcal{C}_\ell)$ is non-zero if and only if a solution exists, in particular, that monomials corresponding to walks that are not paths cancel each other out. This argument is a generalization of the argument used by Björklund et al. [7].

2 Preliminaries

For integers a and b, we use $[a, b]$ to denote the set $\{a, a+1, \ldots, b\}$ and $[b]$ to denote the set $[1, b]$.

[2] Recall that p is the number of constraints for condition (ii) in the Xor-Constrained Shortest Path problem.

Graphs. In this paper, we consider undirected multigraphs, that is, we allow multiple edges and self-loops. We use standard graph-theoretic notation and refer to the textbook by Diestel [17] for undefined notions. Let $G = (V, E)$ be an undirected graph. We use $V(G)$ and $E(G)$ to denote the set of vertices and the set of edges of G, respectively. We use n and m to denote the number of vertices and edges in G, respectively. A *path* P is a graph with vertex set $\{v_0, v_1, \ldots, v_\ell\}$ and edge set $\{v_{i-1}v_i \mid i \in [\ell]\}$. The vertices v_0 and v_ℓ are called the endpoints of P. A *cycle* C is a path with an additional edge between the two endpoints. The length of a path or a cycle is the number of edges in it. For a vertex subset $U \subseteq V$, we use $G[U]$ to denote the subgraph of G induced by the vertices in U and $G - U$ to denote $G[V \setminus V']$. For a set of edges $S \subseteq E$, we write $G - S$ to denote the graph obtained from G by deleting the edges of S.

We are mostly interested in *planar* input graphs. We refer the reader to the textbooks of Diestel [17] and Agnarsson and Greenlaw [1] for rigorous introductions. Informally speaking, a graph is planar if it can be drawn on the plane such that its edges do not cross each other. Such a drawing is called a *planar embedding* of the graph and a planar graph with a planar embedding is called a *plane* graph. We note that checking whether a graph is planar and finding a planar embedding can be done in linear time by the classic algorithm of Hopcroft and Tarjan [27]. The *faces* of a plane graph are the regions bounded by a set of edges and that do not contain any other vertices or edges. The vertices and edges on the boundary of a face form its *frontier*.

Given a plane graph $G = (V, E)$ with faces F, its *dual* graph $G^* = (F, E^*)$ (see Figure 1) is defined as follows. The vertices of G^* are the faces of G and for each $e \in E(G)$, G^* has the *dual* edge e^* whose endpoints are either two faces having e on their frontiers or e^* is a self-loop at f if e is in the frontier of exactly one face f (i.e., e is a bridge of G). Observe that G^* is not necessarily simple even if G is a simple graph as the example in Figure 1 shows. We note that G^* is a planar graph that has a plane embedding where each vertex of G^* corresponding to a face f of G is drawn inside f and each dual edge e^* intersects e only once and e^* does not intersect any other edge of G. Throughout this paper, we assume that G^* has such an embedding.

It is crucial for our results that for a connected plane graph G, each minimal cut in G has a one-to-one correspondence to a cycle in G^*. To be more precise, recall that each cycle on the plane has exactly two faces. Then (A, B) is a minimal cut of a plane graph G if and only if there is a cycle C^* in G^* such that the vertices of A are inside one face of C^* and the vertices of B are inside the other face. Furthermore, C^* is formed by the edges e^* that are dual to the edges $e \in \operatorname{cut}(A)$ and the length of C^* is $|\operatorname{cut}(A)|$.

Let G be a plane graph and let G^* be its dual. We say that a path P (a cycle C) in G *crosses* a cycle C^* of G^* in $e \in E(P)$ ($e \in E(C)$, respectively) if C^* contains the edge $e^* \in E^*$ that is dual to e. The *number of crosses* of P and C^* is the number of edges of P where P and C^* cross. We use the following observation.

▶ **Observation 3.** *Let G be a plane graph, let $s, t \in V$, and let P be an s-t-path. For any cycle C^* of G^*, s and t are in distinct faces of C^* if and only if the number of crosses of P and C^* is odd.*

Lastly, given a subset $U \subseteq V$ of vertices in a plane graph G with faces F, a *face cover* of U is a subset $F' \subseteq F$ of faces such that each vertex in U is on the frontier of a face in F'.

Groups. The group $(\mathbb{Z}_2^d, +)$ consists of the set of all length-d binary strings, and the sum of two strings is defined as the bitwise xor of the strings (or addition without carry over). In this regards, it can be seen as the d-dimensional *bitwise xor vector space* $\mathbb{F}_2^d$. It is easy

to see that this is indeed an (abelian) group: (1) The closure property is trivial, since it by definition contains every length-d binary string. (2) Associativity can be seen by a simple case analysis, i.e., $(a \oplus b) \oplus c = a \oplus (b \oplus c)$ is bitwise 1 if and only if there is an odd number of 1s in the bit's position. (3) The identity element is the all $\mathbf{0}$ vector, i.e. $a \oplus \mathbf{0} = a$. (4) The inverse element of a is a itself, i.e., $a \oplus a = \mathbf{0}$.

3 Shortest Paths under Xor Constraints

In Xor-Constrained Shortest Path, we are given a graph G, two vertices s and t, an edge-labeling function $g\colon E(G) \to \mathbb{Z}_2^d$, a value $c \in \mathbb{Z}_2^d$, and p sets of vertices $X_1, \ldots, X_p \subseteq V(G)$. The problem is to find an s-t-path P that satisfies (i) $\sum_{e \in E(P)} g(e) = c$ and (ii) for each $i \in [p]$, $|V(P) \cap X_i| \leq 1$, and among such paths minimizes the number of edges in P. In this section we prove Theorem 2, which we restate here.

▶ **Theorem 2.** *Xor-Constrained Shortest Path can be solved in $2^{d+p} \cdot (n+m)^{\mathcal{O}(1)}$ time by a one-sided-error randomized algorithm.*

As a corollary, we obtain an algorithm for Xor-Constrained Shortest Cycle by guessing one vertex v in a solution, adding a false twin u of v, that is, a non-neighbor of v with the same neighborhood as v to the input graph such that all edges incident to u are assigned value zero, and then asking for a shortest u-v-path satisfying conditions (i) and (ii).

▶ **Corollary 4.** *Xor-Constrained Shortest Cycle can be solved in $2^{d+p} \cdot (n+m)^{\mathcal{O}(1)}$ time by a one-sided-error randomized algorithm.*

3.1 The Algorithm

In the remainder of this section, we assume that the input graph G is a simple graph. Note that an input n-vertex m-edge multigraph can be turned into an $(n+m)$-vertex $2m$-edge simple graph by first removing self-loops, and then subdividing each edge once, giving the label of the edge to one of the subdivision edges and labeling the other subdivision edge with zero. This exactly doubles the length of the solution. We also assume without loss of generality that $s \neq t$.

Let us next introduce some notation. We say that a sequence $(v_0, v_1, \ldots, v_{\ell-1}, v_\ell)$ of $\ell+1$ vertices is an s-t-walk of length ℓ if $v_0 = s$, $v_\ell = t$, and $v_{i-1}v_i \in E(G)$ for each $i \in [\ell]$. Note that unlike a path, a walk can contain a vertex more than once. We say that an s-t-walk is *feasible* if it satisfies analogies of the contraints (i) and (ii), in particular, if

1. $\sum_{i=1}^{\ell} g(v_{i-1}v_i) = c$, and
2. for each $i \in [p]$, there is at most one $j \in [0, \ell]$ such that $v_j \in X_i$.

For an integer $\ell \geq 1$, let $\mathcal{C}_\ell$ denote the set of all feasible s-t-walks of length (exactly) ℓ. We associate with $\mathcal{C}_\ell$ a polynomial as follows.

Let $q = 2^{\lceil \log_2 n \rceil + 1}$, and recall that $\mathrm{GF}(q)$ is a finite field of characteristic 2 and order q. We define a polynomial over $\mathrm{GF}(q)$ as follows. For each edge $uv \in E(G)$ we associate a variable $f(uv)$. Then, for an s-t-walk $W = (v_0, \ldots, v_\ell)$ of length ℓ, we associate the monomial

$$f(W) = \prod_{i=1}^{\ell} f(v_{i-1}v_i), \tag{1}$$

and for the set $\mathcal{C}_\ell$ of all feasible s-t-walks of length ℓ, we associate the polynomial

$$f(\mathcal{C}_\ell) = \sum_{W \in \mathcal{C}_\ell} f(W).$$

Note that the degree of $f(\mathcal{C}_\ell)$ is ℓ. Our algorithm will be based on the following lemma, which will be proven in Section 3.2.

▶ **Lemma 5.** *The length of a shortest s-t-path satisfying (i) and (ii) is equal to the smallest integer ℓ such that $f(\mathcal{C}_\ell)$ is a non-zero polynomial. If no such ℓ exists, then no such s-t-path exists.*

Given Lemma 5, it remains to design an algorithm for testing if $f(\mathcal{C}_\ell)$ is a non-zero polynomial. For this, we use the DeMillo–Lipton–Schwartz–Zippel lemma.

▶ **Lemma 6** ([42, 46])**.** *Let $f(x_1, \ldots, x_n)$ be a non-zero polynomial of degree d over a field $\mathbb{F}$, and let S be a subset of $\mathbb{F}$. If each x_i is independently assigned a uniformly random value from S, then $f(x_1, \ldots, x_n) = 0$ with probability at most $d/|S|$.*

By Lemma 6 to probabilistically test if $f(\mathcal{C}_\ell)$ is non-zero it suffices to evaluate $f(\mathcal{C}_\ell)$ on a random assignment of values from $\mathrm{GF}(q)$ to the variables $f(uv)$. Because the degree of $f(\mathcal{C}_\ell)$ is $\ell \leq n$ and the order of $\mathrm{GF}(q)$ is $q \geq 2n$, this test is correct with probability at least 0.5 whenever $f(\mathcal{C}_\ell)$ is non-zero. Note that if $f(\mathcal{C}_\ell)$ is the zero polynomial, this test is always correct. Next we show that this evaluation can be done efficiently.

▶ **Lemma 7.** *Given an assignment of values to the variables $f(uv)$ for all $uv \in E(G)$, the value of the polynomial $f(\mathcal{C}_\ell)$ can be evaluated in $\mathcal{O}(2^{d+p}n^2\ell)$ time.*

Proof. We evaluate the polynomial by dynamic programming on walks. For $u \in V(G)$, $l \in [0, \ell]$, $y \in \mathbb{Z}_2^d$, and $T \subseteq [p]$, let $\mathcal{C}(u, l, y, T)$ denote the set of s-u-walks $(s = v_0, v_1, \ldots, v_l = u)$ of length l where

- $\sum_{i=1}^{l} g(v_{i-1}v_i) = y$,
- for each $i \in [p] \setminus T$, it holds that $\{v_0, v_1, \ldots, v_l\} \cap X_i = \emptyset$, and
- for each $i \in T$, there exists exactly one $j \in [0, l]$ such that $v_j \in X_i$.

We denote by $f(\mathcal{C}(u, l, y, T))$ the value $\sum_{W \in \mathcal{C}(u,l,y,T)} f(W)$, where $f(W)$ is defined as in Equation (1), with the empty product interpreted as being equal to 1. Now, we have that $f(\mathcal{C}_\ell) = \sum_{T \subseteq [p]} f(\mathcal{C}(t, \ell, c, T))$. It remains to show that the values $f(\mathcal{C}(u, l, y, T))$ can be computed by dynamic programming.

Let $T_v = \{i \in [p] \mid v \in X_i\}$ for each $v \in V(G)$. Then, the values for $l = 0$ are computed by setting $f(\mathcal{C}(s, 0, 0, T_s)) = 1$ and all other values with $l = 0$ to 0. When $l \geq 1$, the values $f(\mathcal{C}(u, l, y, T))$ are computed by dynamic programming from the values for smaller l as follows.

- If $T_u \subseteq T$, then $f(\mathcal{C}(u, l, y, T)) = \sum_{uw \in E(G)} f(uw) \cdot f(\mathcal{C}(w, l-1, y - g(\{u, w\}), T \setminus T_u))$.
- Otherwise, $f(\mathcal{C}(u, l, y, T)) = 0$.

This clearly computes the values correctly, and runs in overall $\mathcal{O}(2^{d+p}n^2\ell)$ time. ◀

Now, our algorithm works by using Lemma 7 to evaluate $f(\mathcal{C}_\ell)$ for random assignments of values to variables $f(uv)$ for increasing values of $\ell \leq n$, and once it evaluates to non-zero, reports that ℓ is the length of the shortest s-t-path satisfying (i) and (ii). If no such $\ell \leq n$ is found, the algorithm reports that no such s-t-path exists. Note that the correctness of the algorithm depends only on the randomness on the evaluation with the correct ℓ, and therefore the algorithm is correct with probability at least 0.5, and never reports a length shorter than the length of a shortest solution. This probability can be exponentially improved by running the algorithm multiple times. To recover the solution, it suffices to use the algorithm to test which edges can be removed from the graph G until G turns into an s-t-path. Clearly, to both recover the solution and to have an exponentially small error probability it suffices to run the algorithm a polynomial number of times, so this finishes the proof of Theorem 2, modulo the proof of Lemma 5 that will be given in the next subsection.

3.2 Proof of Correctness

This section is devoted to the proof of Lemma 5. We first prove the direction that the existence of a solution of length ℓ implies that $f(\mathcal{C}_\ell)$ is non-zero.

▶ **Lemma 8.** *If an s-t-path of length ℓ satisfying (i) and (ii) exists, then $f(\mathcal{C}_\ell)$ is a non-zero polynomial.*

Proof. Let $W = (s = v_0, v_1, \ldots, v_\ell = t)$ be the sequence of vertices on an s-t-path of length ℓ satisfying conditions (i) and (ii). Note that W is a feasible s-t-walk and $W \in \mathcal{C}_\ell$. Since each vertex occurs in the walk W at most once, we observe that W can be determined uniquely from its set of edges, and W is therefore the only walk in $\mathcal{C}_\ell$ with the monomial $f(W) = \prod_{i=1}^{\ell} f(v_{i-1}v_i)$. Thus, the monomial $f(W)$ occurs in the polynomial $f(\mathcal{C}_\ell)$ with coefficient 1, and therefore $f(\mathcal{C}_\ell)$ is non-zero. ◀

It remains to prove that if no solutions of length at most ℓ exists, then $f(\mathcal{C}_\ell)$ is the zero polynomial. For this, let us state our main lemma, but delay its proof for a bit.

▶ **Lemma 9.** *If no s-t-path of length at most ℓ satisfying conditions (i) and (ii) exists, then there exists a function $\phi : \mathcal{C}_\ell \to \mathcal{C}_\ell$ such that for every $W \in \mathcal{C}_\ell$ it holds that*

1. $\phi(\phi(W)) = W$,
2. $\phi(W) \neq W$, *and*
3. $f(\phi(W)) = f(W)$.

Now, assuming Lemma 9, the proof of Lemma 5 can be finished as follows.

▶ **Lemma 10.** *If no s-t-path of length at most ℓ satisfying conditions (i) and (ii) exists, then $f(\mathcal{C}_\ell)$ is the zero polynomial.*

Proof. Let ϕ be the function given by Lemma 9. By properties 1 and 2, the set $\mathcal{C}_\ell$ can be partitioned into pairs $\{W, \phi(W)\}$. Now, property 3 states that $f(W) = f(\phi(W))$ and since $\mathrm{GF}(q)$ is a field of characteristic 2, it holds that $f(W) + f(\phi(W)) = 0$. Thus, $\sum_{W \in \mathcal{C}_\ell} f(W) = 0$. ◀

Putting Lemmas 8 and 10 together implies Lemma 5. It remains to prove Lemma 9.

Proof of Lemma 9. Assume that no s-t-path of length at most ℓ satisfying (i) and (ii) exists. We will define the function $\phi : \mathcal{C}_\ell \to \mathcal{C}_\ell$ explicitly and show that it satisfies all of the required properties. Let $W = (v_0, v_1, \ldots, v_\ell)$ be an s-t-walk in $\mathcal{C}_\ell$. The idea of the definition of ϕ will be to locate a *subwalk* $(v_i, v_{i+1}, \ldots, v_{j-1}, v_j)$ of W where $0 \leq i < j \leq \ell$, and reverse the subwalk, i.e., map the walk

$$W = (v_0, v_1, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_{j-1}, v_j, v_{j+1}, \ldots, v_\ell)$$

into the walk

$$W\overleftarrow{[i,j]} = (v_0, v_1, \ldots, v_{i-1}, v_j, v_{j-1}, \ldots, v_{i+1}, v_i, v_{j+1}, \ldots, v_\ell).$$

In particular, we will have that $\phi(W) = W\overleftarrow{[i,j]}$ for a carefully chosen pair i, j with $0 \leq i < j \leq \ell$. This pair will be chosen so that $v_i = v_j$, which ensures that $W\overleftarrow{[i,j]} \in \mathcal{C}_\ell$ and $f(W\overleftarrow{[i,j]}) = f(W)$ since the multiset of pairs of adjacent vertices in the walk does not change.

It remains to argue that such a pair i, j can be chosen so that the properties $\phi(\phi(W)) = W$ and $\phi(W) \neq W$ hold. Observe that the property $\phi(W) \neq W$ holds if and only if the subwalk from i to j is not a *palindrome*, i.e., a sequence that is the same when reversed. Now we define a process that outputs a pair i, j so that $0 \leq i < j \leq \ell$, $v_i = v_j$, and the subwalk from i to j is not a palindrome.

The process starts by setting $i = j = 0$. Then, it repeats the following: It first selects i to be the smallest integer $i > j$ so that the vertex v_i occurs in the walk in the indices greater than j more than once. If no such i exists, it outputs FAIL. Then, it sets j to be the largest integer so that $v_i = v_j$, in particular, the index of the last occurrence of v_i in the walk. At this point, it is guaranteed that $0 \leq i < j \leq \ell$ and $v_i = v_j$. Now, if the subwalk from i to j is not a palindrome, it outputs the pair i, j. Otherwise, the process repeats.

Observe that the process always outputs either FAIL or a pair i, j with $0 \leq i < j \leq \ell$ and $v_i = v_j$ such that the subwalk from i to j is not a palindrome. We prove that it actually never outputs FAIL.

▷ Claim 11. The process defined above never outputs FAIL.

Proof of Claim 11. Suppose that the process output FAIL and let $i_1 < j_1 < i_2 < \ldots < j_t$ be the sequence of pairs i, j considered during the process. We define the *contracted walk* W' to be the subsequence of $W = (v_0, \ldots, v_\ell)$ obtained by removing the vertices on the indices in $[i_1 + 1, j_1] \cup [i_2 + 1, j_2] \cup \ldots \cup [i_t + 1, j_t]$ from W. In particular, W' is obtained from W by contracting each palindrome $v_{i_k}, \ldots, v_{j_k}$ considered in the process into a single vertex v_{i_k}.

Now, we claim that W' is a feasible s-t-walk of length at most ℓ, and moreover that no vertex occurs more than once in W'. This is a contradiction, because in that case W' would be in fact an s-t-path of length at most ℓ that satisfies conditions (i) and (ii), but we assumed that no such s-t-path exists. We observe that the contracted walk W' is indeed an s-t-walk, because it was obtained from an s-t-walk by contracting subwalks that each start and end in a same vertex. It also clearly has length at most ℓ, and it satisfies the condition (ii) because the multiset of vertices in W' is a subset of the multiset of vertices in W. For condition (i), we observe that if a subwalk $v_i, \ldots, v_j$ is a palindrome, then $\sum_{k=i+1}^{j} g(v_{k-1}v_k) = 0$ because each pair of adjacent vertices occurs an even number of times as G is a simple graph and we are working in the group $\mathbb{Z}_2^d$. Thus, contracting the palindromes does not change the sum of the edge labels on W, and thus W' satisfies condition (i).

Lastly, we argue that no vertex occurs more than once in W'. For the sake of contradiction, suppose that some vertex occurs more than once in W', which in particular implies that there are indices i',j' with $0 \leq i' < j' \leq \ell$ and $v_{i'} = v_{j'}$ that are not in $[i_1 + 1, j_1] \cup [i_2 + 1, j_2] \cup \ldots \cup [i_t + 1, j_t]$. If $i' < i_1$, then this would contradict the choice of i_1, and if $i' = i_1$, then this would contradict the choice of j_1. Similarly, if $j_k < i' \leq i_{k+1}$ for some $1 \leq k < t$, then this would contradict either the choice of i_{k+1} or j_{k+1}, and if $i' > j_t$, then this would contradict the fact that i_t, j_t was the last pair considered by the algorithm. ◁

Now, the function $\phi : \mathcal{C}_\ell \to \mathcal{C}_\ell$ is defined as $\phi(W) = W\overleftarrow{[i, j]}$, where i, j is the pair output by the process described above. We have already proven that $\phi(W) \neq W$ and $f(\phi(W)) = f(W)$, so it remains to prove that $\phi(\phi(W)) = W$. For this, it remains to observe that the operation $W\overleftarrow{[i, j]}$ does not change how the process for selecting i,j behaves, because it does not change the walk before the index i and it does not change the fact that the last occurrence of v_i is at the index j. ◀

This finishes the proof of Theorem 2.

4 Two-Sets Cut-Uncut Parameterized by the Face Cover Number

In this section, we show that Two-Sets Cut-Uncut is FPT when parameterized by the minimum number of faces in a planar embedding of the input graph covering the terminals. We use the following result by Bienstock and Monma [5] showing that a minimum face cover can be found in FPT time when parameterized by the size of a cover.

▶ **Proposition 12** ([5]). *It can be decided in $2^{\mathcal{O}(r)} \cdot n$ time whether for a set of vertices U of a planar graph G and a positive integer r, there is a planar embedding of G such that at most r faces cover U. Furthermore, if such an embedding exists, it can be found together with the face cover of U in the same time.*

By Proposition 12, we can assume that the input graph is plane, that is, we are given its planar embedding and, furthermore, we are given a face cover of the terminals.

Next, we note that we can consider only minimal cuts.

▶ **Observation 13** ($\star$). *Let (G, S, T, k) be an instance of Two-Sets Cut-Uncut where G is a connected graph. Then, (G, S, T, k) is a yes-instance if and only if there is a minimal cut (A, B) of G with $|\mathrm{cut}(A)| \leq k$ such that $S \subseteq A$ and $T \subseteq B$.*

This observation implies that to solve Two-Sets Cut-Uncut on a plane graph G, we have to find a shortest cycle C^* in the dual graph G^* such that the vertices of S and T are in distinct faces of C^*. First, we prove that we can assume without loss of generality that the input graph is 2-connected. This assumption simplifies arguments because the frontier of each face of a plane 2-connected graph is a cycle [17].

▶ **Lemma 14** ($\star$). *There is a polynomial-time algorithm that, given an instance (G, S, T, k) of Two-Sets Cut-Uncut, solves the problem or outputs an equivalent instance (G', S', T', k) of Two-Sets Cut-Uncut where G' is a 2-connected induced subgraph of G. Furthermore, given a planar embedding of G such that $S \cup T$ can be covered by at most r faces, $S' \cup T'$ can be covered in the induced embedding of G' by at most r faces.*

From now on, we assume that the graph of the considered instances of Two-Sets Cut-Uncut is 2-connected. Hence, the dual graph G^* has no loops. Also, since loops are irrelevant for Two-Sets Cut-Uncut, we assume that the input graph has no loops.

We use the following two separation properties for vertices on the frontier of the same face of a graph.

▶ **Lemma 15.** *Let G be a plane graph, let C be the cycle formed by the frontier of a face f of G, and let X and Y be disjoint nonempty sets of vertices in C. Let C^* be any cycle in G^*. Then, the vertices of X and the vertices of Y are in distinct faces of C^* if and only if $f \in V(C^*)$ and C crosses C^* in two edges e_1 and e_2 such that (i) the vertices of X are in the same connected component of $C - \{e_1, e_2\}$, (ii) the vertices of Y are in the same connected component of $C - \{e_1, e_2\}$, and (iii) the vertices of X and the vertices of Y are in distinct connected components of $C - \{e_1, e_2\}$.*

Proof. Suppose that the vertices of X and the vertices of Y are in distinct faces of C^*. Then, f is a vertex of C^*. Let e_1^* and e_2^* be the edges of C^* incident to f and let e_1 and e_2 be the dual edges of e_1^* and e_2^*, respectively. Note that C contains both e_1 and e_2 and C crosses C^* only in these two edges. We have that $C - \{e_1, e_2\}$ has two connected components P_1 and P_2 that are paths. Since C^* separates X and Y, we have that X is fully contained in P_1 or fully contained in P_2 and Y is fully contained in the respective other path. Thus, conditions (i)–(iii) are fulfilled.

For the opposite direction, assume that C crosses C^* in two edges e_1 and e_2 such that conditions (i)–(iii) are fulfilled. Consider an x-y-path P in C for arbitrary $x \in X$ and $y \in Y$ containing e_1 and excluding e_2 that exists by (i)–(iii). The number of crosses of P and C^* is one. Hence, x and y are in distinct faces of C^* by Observation 3. This concludes the proof. ◀

▶ **Lemma 16.** *Let G be a plane graph, let C be the cycle formed by the frontier of a face f of G, and let X be a nonempty sets of vertices in C. Let C^* be any cycle in G^*. Then, the vertices of X are in the same face of C^* if and only if either $f \notin V(C^*)$ and C does not cross C^* or $f \in V(C^*)$ and C crosses C^* in two edges e_1 and e_2 such that the vertices of X are in the same connected component of $C - \{e_1, e_2\}$.*

Proof. Suppose that the vertices of X are in the same face of C^* and assume that C crosses C^*. Then, f is a vertex of C^* and C^* has two edges e_1^* and e_2^* incident to f. We have that C crosses C^* in the edges e_1 and e_2 that are dual to e_1^* and e_2^*, respectively. Since C is a cycle, $C - \{e_1, e_2\}$ has two connected components P_1 and P_2 that are both paths. We show that either $X \subseteq V(P_1)$ or $X \subseteq V(P_2)$. For the sake of contradiction, assume that there are $x, y \in X$ such that $x \in V(P_1)$ and $y \in V(P_2)$. Then, there is an x-y-path P in C that contains e_1 but excludes e_2. The number of crosses of P and C^* is one and x and y are therefore in distinct faces of C^* by Observation 3; a contradiction. Hence, the vertices of X are in the same connected component of $C - \{e_1, e_2\}$.

For the opposite direction, assume that either C does not cross C^* or C crosses C^* in two edges e_1 and e_2 such that the vertices of X are in the same connected component of $C - \{e_1, e_2\}$. In both cases, for any two vertices $x, y \in X$, there is an x-y-path P that does not cross C^*. Then by Observation 3, the vertices of X are in the same face of C^*. This concludes the proof. ◀

We are now in a position to present the main result of this section.

▶ **Theorem 1.** *There is a one-sided-error randomized algorithm solving* Two-Sets Cut-Uncut *on planar graphs in $2^{|S|+|T|} \cdot n^{\mathcal{O}(1)}$ time. Moreover, there is a one-sided-error randomized algorithm solving the problem in $2^{\mathcal{O}(r)} \cdot n^{\mathcal{O}(1)}$ time, where r is the number of faces needed to cover $S \cup T$ in a planar embedding.*

Proof. We show the claim for the parameterization by the size of a face cover of the terminals and then explain how a simplified version of the algorithm can be used for the parameterization by the number of terminals.

Let (G, S, T, k) be an instance of Two-Sets Cut-Uncut where G is planar. We use Proposition 12 to decide whether there is a planar embedding of G such that the set of terminals $S \cup T$ can be covered by at most r faces. When the algorithm reports that such an embedding does not exist, we stop. Otherwise, we obtain an embedding of G in the plane and a set of faces F' of size at most r covering $S \cup T$. From now on, we assume that G is a plane graph. We remind that G can be assumed to be 2-connected by Lemma 14. We use the embedding of G to construct the dual graph G^* together with its embedding. By Observation 13 and duality, our task is to find a cycle C^* in G^* of length at most k such that S and T are in distinct faces of C^*. We find such a cycle using the algorithm for Xor-Constrained Shortest Cycle from Corollary 4.

We partition F' into two sets where $F_1 \subseteq F'$ is the set of faces having vertices from both S and T on their frontiers and $F_2 \subseteq F'$ consists of the faces $f \in F'$ such that the frontier of f contains either only vertices of S or only vertices of T. We modify G^* by analyzing each face $f \in F'$. The ultimate aim of the modification is to reduce the number of considered terminals.

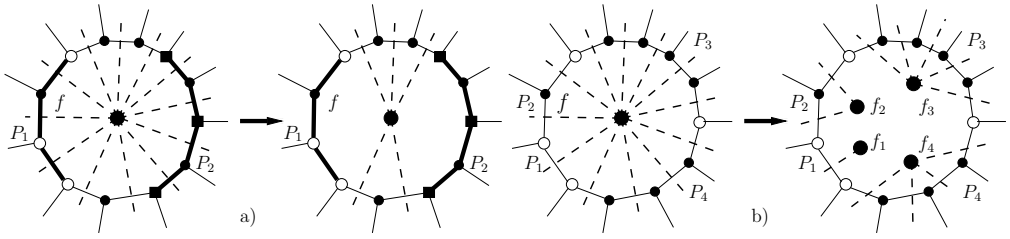


Figure 2 (a) The modification for $f \in F_1$. The vertices of S' are shown by white circles, the vertices of T' are shown by black squares, and the other vertices of G are shown by black circles. The edges of G^* are shown by dashed lines. The paths P_1 and P_2 are shown by thick lines. (b) The modification for $f \in F_2$. The vertices of $L \in \{S, T\}$ are shown by white circles and the other vertices of G are shown by small black circles. The vertex f of G^* and the vertices $f_1, f_2 \ldots, f_4$ are shown by large black circles and the edges of G^* and the constructed new edges are shown by dashed lines.

Modifications for F_1. Let $f \in F_1$ and let C be the cycle of G forming the frontier of f. Recall that $S' = S \cap V(C) \neq \emptyset$ and $T' = T \cap V(C) \neq \emptyset$. If there are no two edges $e_1, e_2 \in E(C)$ such that the vertices of S' and T' are in distinct connected components of $C - \{e_1, e_2\}$, then by Lemma 15, there is no cycle C^* such that the vertices of S' and the vertices of T' are in distinct faces of C^*. This implies that (G, S, T, k) is a no-instance. Hence, we assume that this is not the case and select two inclusion-minimal disjoint paths P_1 and P_2 in C such that $S' \subseteq V(P_1)$ and $T' \subseteq V(P_2)$. We modify G^* by deleting each edge e^* incident to f that is dual to an edge $e \in E(P_1) \cup E(P_2)$ (see Figure 2 (a)).

Modifications for F_2. Let $f \in F_2$, let C be the cycle of G forming the frontier of f, and let $L = V(C) \cap (S \cup T)$. Note that by definition of F_2, either $L \subseteq S$ or $L \subseteq T$. We split the vertex f of G^* into $q = |L|$ vertices $f_1, f_2, \ldots, f_q$ as follows. If $q \geq 2$, then C contains q internally vertex disjoint paths $P_1, P_2, \ldots, P_q$ whose end-vertices are in L (and whose internal vertices are not in L). We then

- delete f and construct a set $X_f = \{f_1, f_2, \ldots, f_q\}$ of q new vertices,
- for each $j \in [q]$ and edge e in P_j, we replace the dual edge e^* of G^* by an edge incident to f_j whose second endpoint is the same as for e^* unless e^* was deleted by some modification for F_1.

For $q = 1$, we set $X_f = \{f\}$ and $f_1 = f$, that is, we do not perform any modification. The construction is shown in Figure 2 (b). Note that the vertices $f_1, f_2, \ldots, f_q$ can be embedded in the face f of G such that the resulting graph H^* is plane. For each edge $e^* \in E(H^*)$, there is an edge $e \in E(G^*)$ such that e^* was constructed from the edge that is dual to e in G^*. Slightly abusing notation, we do not distinguish between the edges of H^* and G^*. In particular, we say that e^* is dual to e.

Next, we assign labels to the edges of H^* from $\mathbb{Z}_2^d$ for some appropriate d. For this, we greedily pick a set R of *representatives* from $S \cup T$ for each $f \in F'$. From each $f \in F_1$, we select two terminals from S and T, respectively, that are on the frontier of the face f of G. For each $f \in F_2$, we pick one terminal from the frontier of the face of f. We then construct an arbitrary inclusion minimal tree Q in G that spans R. This can be done in linear time using standard tools (see, e.g., [9]). We select an arbitrary vertex $u \in R \cap S$ and set $d = |R| - 1$. Observe that $|R| \leq 2|F_1| + |F_2|$ and $d \leq 2|F_1| + |F_2| - 1$. Denote by $v_1, \ldots, v_d$ the vertices of $L \setminus \{u\}$ and let Q_i be the u-v_i-path in Q for each $i \in [d]$. We define $g\colon E(G) \to \mathbb{Z}_2^d$ by setting $g(e) = (\delta_1, \ldots, \delta_d)^\intercal$ where for each $i \in [d]$, $\delta_i = \begin{cases} 1 & \text{if } e \in E(Q_i), \\ 0 & \text{if } e \notin E(Q_i). \end{cases}$

Moreover, let $g^*\colon E(H^*) \to \mathbb{Z}_2^d$ be defined by setting $g^*(e^*) = g(e)$ for each $e^* \in E(H^*)$ that is dual to $e \in E(G)$ and let $c = (c_1, \ldots, c_d)^\intercal \in \mathbb{Z}_2^d$ where

$$c_i = \begin{cases} 0 & \text{if } v_i \in S, \\ 1 & \text{if } v_i \in T \end{cases} \quad \text{for } i \in [d].$$

We show the following claim using Lemma 15 and Lemma 16.

▷ Claim 17 (⋆). The graph G^* contains a cycle C^* of length at most k such that the vertices of S and the vertices of T are in distinct faces of C^* if and only if the instance $(H^*, g^*, c, \{X_f \mid f \in F_2\})$ of Xor-Constrained Shortest Cycle has a solution and the length of a solution cycle is at most k.

By Claim 17, solving Two-Sets Cut-Uncut for (G, S, T, k) is equivalent to solving Xor-Constrained Shortest Cycle for $(H^*, g^*, c, \{X_f \mid f \in F_2\})$. For this, we use the algorithm from Corollary 4.

For the running time, observe that a face cover F' of size r can be constructed in $2^{\mathcal{O}(r)}n$ time by Proposition 12. Given such a cover, the graph H^* together with the sets X_f for $f \in F_2$ can be constructed in polynomial time. Since $d \leq 2|F_1| + |F_2| - 1 \leq 2r$, the labeling g^* and $c \in \mathbb{Z}_2^d$ can also be constructed in polynomial time. Finally, since $d \leq 2|F_1| + |F_2| - 1$ and $p = |\{X_i \mid f \in F_2\}| = |F_2|$, we have that $p + d \leq 2r - 1$ and the algorithm from Corollary 4 runs in $4^r n^{\mathcal{O}(1)}$ time. We conclude that the overall running time is in $2^{\mathcal{O}(r)}n^{\mathcal{O}(1)}$. This concludes the proof.

If we parameterize Two-Sets Cut-Uncut by $\ell = |S| + |T|$, then we can use a simplified variant of the algorithm. Given an instance (G, S, T, k) of Two-Sets Cut-Uncut where G is a planar graph, we use the classic algorithm of Hopcroft and Tarjan [27] to find a plane embedding of G. We then use the variant of the algorithm where we do not modify G^*, that is, we set $H^* = G^*$, and where we assume that all the terminals are representatives, that is, we set $R = S \cup T$. The labeling $g^*\colon E(H^*) \to \mathbb{Z}_2^d$ and c are defined in the same way as in the algorithm for the parameterization by the size of a face cover. By Observation 3, solving Two-Sets Cut-Uncut for (G, S, T, k) is equivalent to solving Xor-Constrained Shortest Cycle for $(H^*, g^*, c, \emptyset)$. Since $d = |R| - 1 = |S| + |T| - 1$, we conclude that we can solve the problem in $2^{|S|+|T|} \cdot n^{\mathcal{O}(1)}$ time using Corollary 4. ◀

5 Hardness

It is known that Two-Sets Cut-Uncut is NP-complete [25] in planar graphs and that it is NP-complete in general graphs even if $|S| = 2$ [44]. We strengthen the latter result by showing that Two-Sets Cut-Uncut remains W[1]-hard parameterized by $|T|$ even if $|S| = 1$ by providing a polynomial-time reduction from Regular Multicolored Clique parameterized by solution size k – a variant of Multicolored Clique where each vertex has the same degree d – such that $|T| = k$. This problem is known to be W[1]-hard and assuming ETH, it cannot be solved in $f(k) \cdot n^{o(k)}$ time [38, 12].

▶ **Proposition 18** (⋆). *Two-Sets Cut-Uncut is* W[1]*-hard when parameterized by* $|T|$ *even if* $|S| = 1$*. Moreover, this restricted version cannot be solved in* $f(|T|) \cdot n^{o(k)}$ *time unless the ETH breaks.*

6 Conclusion

In this paper, we have shown that Two-Sets Cut-Uncut is FPT on planar graphs parameterized by the number of terminals. We have also proven a more general result that the problem remains FPT parameterized by the minimum number of faces required to cover the terminals. Our result implies a polynomial-time algorithm for Network Diversion on planar graphs. We complement this result by showing that Two-Sets Cut-Uncut parameterized by the number of terminals ($|S| + |T|$) is W[1]-hard in general graphs even when $|S| = 1$.

We conclude with a few open problems.

1. First, we repeat the long-standing open question, whether Network Diversion is polynomial-time solvable in general graphs. Even the case for graphs embeddable on a torus remains open.
2. A natural extension of the Two-Sets Cut-Uncut is to extend it to a larger number of sets. Since on general graphs, 3-Way Cut is NP-complete [15], the same holds for Three-Sets Cut-Uncut even when all sets are of size one. However, for planar graphs, k-Way Cut is solvable in polynomial time for fixed k [15, 32, 39]. As a very concrete open question, we ask whether Three-Sets Cut-Uncut is solvable in polynomial time on planar graphs when two sets are of size one and one set is of size two.
3. Our algorithm is randomized and works only on unweighted graphs; can we get rid of either of these restrictions?
4. Finally, is it possible to solve Two-Sets Cut-Uncut in *subexponential* time (in $|S|+|T|$)? In our opinion, this could be a challenging problem. In particular, it is already an open question whether it is possible to find a cycle in a planar graph containing a given set of k terminals in subexponential time (in k).

References

1 Geir Agnarsson and Raymond Greenlaw. *Graph theory: Modeling, applications, and algorithms.* Pearson, 2006.

2 Matthias Bentert, Pål Grønås Drange, Fedor V. Fomin, Petr A. Golovach, and Tuukka Korhonen. Two-sets cut-uncut on planar graphs, 2023. `arXiv:2305.01314`.

3 Marshall Bern. Faster exact algorithms for steiner trees in planar networks. *Networks*, 20(1):109–120, 1990.

4 Ivona Bezáková and Zachary Langley. Minimum planar multi-sink cuts with connectivity priors. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 94–105. Springer, 2014.

5 Daniel Bienstock and Clyde L. Monma. On the complexity of covering vertices by faces in a planar graph. *SIAM Journal on Computing*, 17(1):53–76, 1988.

6 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *Journal of Computer and System Sciences*, 87:119–139, 2017.

7 Andreas Björklund, Thore Husfeldt, and Nina Taslaman. Shortest cycle through specified elements. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1747–1753. Society for Industrial and Applied Mathematics, 2012.

8 Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. *SIAM Journal on Computing*, 45(4):1171–1229, 2016.

9 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press, 2009.

10 Christopher A. Cullenbine, R. Kevin Wood, and Alexandra M. Newman. Theoretical and computational advances for network diversion. *Networks*, 62(3):225–242, 2013.

11 Norman D. Curet. The network diversion problem. *Military Operations Research*, 6(2):35–44, 2001.

12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

13 Marek Cygan, Pawel Komosa, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, Saket Saurabh, and Magnus Wahlström. Randomized contractions meet lean decompositions. *ACM Transactions on Algorithms*, 17(1):6:1–6:30, 2021.

14 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Solving the 2-disjoint connected subgraphs problem faster than 2^n. *Algorithmica*, 70(2):195–207, 2014.

15 Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.

16 Ulrich Derigs. An efficient Dijkstra-like labeling method for computing shortest odd/even paths. *Information Processing Letters*, 21(5):253–258, 1985.

17 Reinhard Diestel. *Graph Theory*. Springer, 2012.

18 Qi Duan, Haadi Jafarian, Ehab Al-Shaer, and Jinhui Xu. On DDoS attack related minimum cut problems. *CoRR*, abs/1412.3359, 2015. `arXiv:1412.3359`.

19 Qi Duan and Jinhui Xu. On the connectivity preserving minimum cut problem. *Journal of Computer and System Sciences*, 80(4):837–848, 2014.

20 Eduard Eiben, Tomohiro Koana, and Magnus Wahlström. Determinantal sieving. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 377–423. Society for Industrial and Applied Mathematics, 2024.

21 Ranel E. Erickson, Clyde L. Monma, and Arthur F. Jr. Veinott. Send-and-split method for minimum-concave-cost network flows. *Mathematics of Operations Research*, 12(4):634–664, 1987.

22 Ozgur Erken. A branch-and-bound algorithm for the network diversion problem. Master's thesis, Naval Postgraduate School, 2002.

23 Arnold Filtser. A face cover perspective to ℓ_1 embeddings of planar graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1945–1954. Society for Industrial and Applied Mathematics, 2020.

24 Fedor V. Fomin, Petr A. Golovach, Tuukka Korhonen, Kirill Simonov, and Giannos Stamoulis. Fixed-parameter tractability of maximum colored path and beyond. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3700–3712. SIAM, 2023.

25 Chris Gray, Frank Kammer, Maarten Löffler, and Rodrigo I. Silveira. Removing local extrema from imprecise terrains. *Computational Geometry*, 45(7):334–349, 2012.

26 Martin Grötschel and William R. Pulleyblank. Weakly bipartite graphs and the max-cut problem. *Operations Research Letters*, 1(1):23–27, 1981.

27 John E. Hopcroft and Robert E. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974.

28 Yoichi Iwata and Yutaro Yamaguchi. Finding a shortest non-zero path in group-labeled graphs. *Combinatorica*, 42(S2):1253–1282, 2022.

29 Benjamin S. Kallemyn. *Modeling Network Interdiction Tasks*. Doctoral thesis, Air Force Institute Of Technology, 2015.

30 Frank Kammer and Torsten Tholey. The complexity of minimum convex coloring. *Discrete Applied Mathematics*, 160(6):810–833, 2012.

31 Sándor Kisfaludi-Bak, Jesper Nederlof, and Erik Jan van Leeuwen. Nearly ETH-tight algorithms for planar Steiner tree with terminals on few faces. *ACM Transactions on Algorithms*, 16(3):1–30, 2020.

32 Philip N. Klein and Dániel Marx. Solving planar k-terminal cut in $O(n^{c\sqrt{k}})$ time. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 569–580. Springer, 2012.

33 Yusuke Kobayashi and Sho Toyooka. Finding a shortest non-zero path in group-labeled graphs via permanent computation. *Algorithmica*, 77(4):1128–1142, 2017.

34 Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 575–586. Springer, 2008.

35 Robert Krauthgamer, James R. Lee, and Havana I. Rika. Flow-cut gaps and face covers in planar graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 525–534. SIAM, 2019.

36 Chungmok Lee, Donghyun Cho, and Sungsoo Park. A combinatorial benders decomposition algorithm for the directed multiflow network diversion problem. *Military Operations Research*, 24(1):23–40, 2019.

37 Dániel Marx. A tight lower bound for planar multiway cut with fixed number of terminals. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 677–688. Springer, 2012.

38 Luke Mathieson and Stefan Szeider. Editing graphs to satisfy degree constraints: A parameterized approach. *Journal of Computer and System Sciences*, 78(1):179–191, 2012.

39 Sukanya Pandey and Erik Jan van Leeuwen. Planar multiway cut with terminals on few faces. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2032–2063. Society for Industrial and Applied Mathematics, 2022.

40 Daniël Paulusma and Johan M. M. van Rooij. On partitioning a graph into two connected subgraphs. *Theor. Comput. Sci.*, 412(48):6761–6769, 2011. doi:10.1016/j.tcs.2011.09.001.

41 Ashutosh Rai, M. S. Ramanujan, and Saket Saurabh. A parameterized algorithm for mixed-cut. In *Proceedings of the 12th Latin American Symposium (LATIN)*, pages 672–685. Springer, 2016.

42 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.

43 Jan Arne Telle and Yngve Villanger. Connecting terminals and 2-disjoint connected subgraphs. In *Proceedings of the 39th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 418–428. Springer, 2013.

44 Pim van't Hof, Daniël Paulusma, and Gerhard J. Woeginger. Partitioning graphs into connected parts. *Theoretical Computer Science*, 410(47-49):4834–4843, 2009.

45 Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Information Processing Letters*, 109(6):315–318, 2009.

46 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposiumon Symbolic and Algebraic Computation (EUROSAM)*, pages 216–226. Springer, 1979.

Splitting-Off in Hypergraphs

Kristóf Bérczi
MTA-ELTE Matroid Optimization Research Group and HUN-REN-ELTE Egerváry Research Group, Department of Operations Research, Eötvös Loránd University, Budapest, Hungary

Karthekeyan Chandrasekaran
University of Illinois, Urbana-Champaign, IL, USA

Tamás Király
MTA-ELTE Matroid Optimization Research Group and HUN-REN-ELTE Egerváry Research Group, Department of Operations Research, Eötvös Loránd University, Budapest, Hungary

Shubhang Kulkarni
University of Illinois, Urbana-Champaign, IL, USA

Abstract

The splitting-off operation in undirected graphs is a fundamental reduction operation that detaches all edges incident to a given vertex and adds new edges between the neighbors of that vertex while preserving their degrees. Lovász [45, 47] and Mader [48] showed the existence of this operation while preserving global and local connectivities respectively in graphs under certain conditions. These results have far-reaching applications in graph algorithms literature [2, 7, 8, 12, 17, 22, 23, 24, 25, 26, 29, 30, 32, 33, 35, 38, 40, 41, 46, 48, 49, 50, 51]. In this work, we introduce a splitting-off operation in hypergraphs. We show that there exists a local connectivity preserving complete splitting-off in hypergraphs and give a strongly polynomial-time algorithm to compute it in weighted hypergraphs. We illustrate the usefulness of our splitting-off operation in hypergraphs by showing two applications: (1) we give a constructive characterization of k-hyperedge-connected hypergraphs and (2) we give an alternate proof of an approximate min-max relation for max Steiner rooted-connected orientation of graphs and hypergraphs (due to Király and Lau [38]). Our proof of the approximate min-max relation for graphs circumvents the Nash-Williams' strong orientation theorem and uses tools developed for hypergraphs.

2012 ACM Subject Classification Theory of computation → Network optimization; Theory of computation → Routing and network design problems

Keywords and phrases Hypergraphs, Hypergraph Connectivity, Splitting-off, Constructive Characterizations, Hypergraph Orientations, Submodular Functions, Combinatorial Optimization

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.23

Category Track A: Algorithms, Complexity and Games

Related Version This is an extended abstract.
Full Version: `https://arxiv.org/abs/2307.08555`

Funding Karthekeyan and Shubhang were supported in part by NSF grants CCF-1814613 and CCF-1907937. Karthekeyan was supported in part by the Distinguished Guest Scientist Fellowship of the Hungarian Academy of Sciences – grant number VK-6/1/2022. Kristóf and Tamás were supported in part by the Lendület Programme of the Hungarian Academy of Sciences – grant number LP2021-1/2021, by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund – grant number ELTE TKP 2021-NKTA-62 funding scheme, and by the Dynasnet European Research Council Synergy project – grant number ERC-2018-SYG 810115.

Acknowledgements Part of this work was done while Karthekeyan and Shubhang were visiting Eötvös Loránd University. Karthekeyan thanks Eklavya Sharma for engaging in preliminary discussions on hypergraph splitting-off.

1 Introduction

The splitting-off operation in undirected graphs is a simple yet powerful operation in graph theory. It is a reduction operation that detaches all edges incident to a given vertex and adds new edges between the neighbors of that vertex while preserving their degrees. Equivalently, it enables a vertex to exit the network by informing its neighbors how to reconfigure the lost links among themselves in order to preserve their degrees. Lovász [45, 47] introduced the splitting-off operation and showed the existence of the operation to preserve global edge-connectivity under certain conditions. Mader [48] showed the existence of the splitting-off operation to preserve local edge-connectivity (i.e., all pairwise edge-connectivities) under certain conditions. Both Lovász's and Mader's results also admit strongly polynomial-time algorithms [24, 27, 49]. Owing to the inductive nature of the splitting-off operation, Lovász's and Mader's results have enabled fundamental results in graph theory as well as efficient algorithms and min-max relations for numerous graph optimization problems. In fact, Mader [48] illustrated the power of his local edge-connectivity preserving splitting-off result by deriving Nash-Williams' strong orientation theorem [52] (also see Frank's exposition of this derivation [26]). Subsequently, the splitting-off operation has been used to give a constructive characterization of k-edge-connected graphs [24] and to address problems in edge-connectivity augmentation [2, 22, 23, 24, 49], graph orientation [25, 38], minimum cuts enumeration [29, 32, 50], network design [12, 30, 35], tree packing [7, 41], congruency-constrained cuts [51], and approximation algorithms for TSP [8,30]. Designing fast algorithms for global/local edge-connectivity preserving splitting-off remains an active area of research (e.g., see recent works [9, 10, 11, 42]) due to these far-reaching applications. In this work, we introduce a splitting-off operation in *hypergraphs*, show the existence of local-connectivity preserving splitting-off operation and design a strongly polynomial-time algorithm to compute it in weighted hypergraphs, and illustrate its usefulness by showing two applications.

Hypergraphs. Hypergraphs offer a richer and more accurate model than graphs for several applications. Consequently, hypergraphs have found applications in several modern areas (e.g., see [44, 54, 57, 60]) and these applications have, in turn, driven exciting recent progress in algorithms for hypergraph optimization problems [1, 4, 13, 14, 15, 18, 19, 21, 28, 31, 34, 36, 37, 39, 43, 55, 58]. A hypergraph $G = (V, E)$ consists of a finite set V of vertices and a set E of hyperedges, where every hyperedge $e \in E$ is a subset of V. We will denote a hypergraph $G = (V, E)$ with hyperedge weights $w : E \to \mathbb{Z}_+$ by the tuple (G, w) and use G_w to denote the unweighted multi-hypergraph over vertex set V containing $w(e)$ copies of every hyperedge $e \in E$. Throughout this work, we will be interested only in hypergraphs with positive integral weights and for algorithmic problems where the input/output is a hypergraph, we will require that the weights are represented in binary. If all hyperedges have size at most 2, then the hyperedges are known as edges and we call such a hypergraph as a graph. We emphasize a subtle but important distinction between hypergraphs and graphs: the number of hyperedges in a hypergraph could be exponential in the number of vertices. This is in sharp contrast to graphs where the number of edges is at most the square of the number of vertices. Consequently, in hypergraph network design problems where the goal is to construct a hypergraph with certain properties, one needs to be mindful of the number of hyperedges in the hypergraph returned by the algorithm. Recent works in hypergraph algorithms literature have focused on this issue in the context of cut/spectral sparsification of hypergraphs [4, 18, 19, 34, 36, 37, 39, 43, 55, 58].

Notation. Let $(G = (V, E), w : E \to \mathbb{Z}_+)$ be a hypergraph. For $X \subseteq V$, let $\delta_G(X) := \{e \in E : e \cap X \neq \emptyset, e \setminus X \neq \emptyset\}$. We define the cut function $d_{(G,w)} : 2^V \to \mathbb{Z}_{\geq 0}$ by $d_{(G,w)}(X) := \sum_{e \in \delta_G(X)} w(e)$ for every $X \subseteq V$. For a vertex $v \in V$, we use $d_{(G,w)}(v)$ to denote $d_{(G,w)}(\{v\})$. We define the degree of a vertex v to be the sum of the weights of hyperedges containing v – we note that the degree of a vertex is not necessarily equal to $d_{(G,w)}(v)$ since we could have $\{v\}$ itself as a hyperedge (i.e., a singleton hyperedge that contains only the vertex v). For distinct vertices $u, v \in V$, let $\lambda_{(G,w)}(u, v) := \min\{d_{(G,w)}(X) : u \in X \subseteq V \setminus \{v\}\}$ – i.e., $\lambda_{(G,w)}(u, v)$ is the value of a minimum $\{u, v\}$-cut in the hypergraph. If all hyperedge weights are unit, then we drop w from the subscript and simply use d_G and λ_G.

Hypergraph Splitting-off. We now introduce our definition of splitting-off in hypergraphs. To compare and contrast our definition of splitting-off for hypergraphs with the classical definition of splitting-off for graphs, we include both our definition and the classical definition and distinguish them by identifying them as h-splitting-off and g-splitting-off. In the definitions below, we encourage the reader to consider the input hypergraph (G, w) to be a graph while considering g-splitting-off terminology and to be a hypergraph while considering h-splitting-off terminology. We encourage the reader to also assume unit weights during first read. See Figure 1 for an example.

▶ **Definition 1.** *Let $(G = (V, E), w : E \to \mathbb{Z}_+)$ be a hypergraph and $s \in V$.*

1. *In* merge almost-disjoint hyperedges*, we pick a pair of hyperedges $e, f \in \delta_G(s)$ such that $e \cap f = \{s\}$, pick a positive integer $\alpha \in \mathbb{Z}_+$ such that $\alpha \leq \min\{w(e), w(f)\}$, reduce the weights of hyperedges e and f by α, and increase the weight of a hyperedge g by α. Here,*
   - **a.** *if we choose $g := e \cup f$, then the associated operation will be called as* h-merge almost-disjoint hyperedges operation.
   - **b.** *if we choose $g := (e \cup f) \setminus \{s\}$, then the associated operation will be called as* g-merge almost-disjoint hyperedges operation.

   In the above, if $\alpha = w(e)$ (resp. if $\alpha = w(f)$), then we discard the hyperedge e (resp. hyperedge f) from the hypergraph obtained after the operation; if the hyperedge $g \notin E$, then we introduce g as a new hyperedge with weight $w(g) := 0$ before performing the weight increase on the hyperedge g.
2. *In* trim hyperedge operation*, we pick a hyperedge $e \in \delta_G(s)$, pick a positive integer $\alpha \in \mathbb{Z}_+$, reduce the weight of the hyperedge e and increase the weight of the hyperedge $g := e \setminus \{s\}$. Here,*
   - **a.** *if we choose $\alpha \leq w(e)$, reduce the weight of the hyperedge e by α, and increase the weight of the hyperedge g by α, then the associated operation will be called as* h-trim operation *(if $\alpha = w(e)$, then we discard e from the hypergraph obtained after the operation; if $g \notin E$, then we add g as a new hyperedge with weight $w(g) := 0$ before performing the weight increase on the hyperedge g).*
   - **b.** *if we choose $\alpha \leq w(e)/2$, reduce the weight of the hyperedge e by 2α, and increase the weight of the hyperedge g by 2α, then the associated operation will be called as* g-trim operation *(if $\alpha = w(e)/2$, then we discard e from the hypergraph obtained after the operation; if $g \notin E$, then we add g as a new hyperedge with weight $w(g) := 0$ before performing the weight increase on the hyperedge g).*
3. *We say that a hypergraph $(H = (V, E_H), w_H : E_H \to \mathbb{Z}_+)$ is obtained by applying a*
   - **a.** h-splitting-off operation at s from (G, w) *if (H, w_H) is obtained from (G, w) by either the h-merge almost-disjoint hyperedges operation or the h-trim hyperedge operation.*
   - **b.** g-splitting-off operation at s from (G, w) *if (H, w_H) is obtained from (G, w) by either the g-merge almost-disjoint hyperedges operation or the g-trim hyperedge operation.*

Certain remarks regarding the definitions are in order. Firstly, the trim operation is valuable and unique to hypergraphs. It has been used in the hypergraph literature to obtain small-sized certificates for hypergraph connectivity [18] and for certain notions of directed hypergraph connectivity [24]. We note that the trim operation has limited value in graphs – trimming an edge leads to a singleton edge and singleton edges contribute only to the degree but not to the cut value of any set. Secondly, all operations mentioned above are degree preserving for vertices $u \in V \setminus \{s\}$: both h-trim and g-trim operations preserve degrees by definition; both h-merge and g-merge *almost-disjoint* hyperedges operations preserve degrees due to the almost-disjoint property of the chosen hyperedges. Thirdly, all operations mentioned above *do not* increase the cut values of subsets $X \subseteq V \setminus \{s\}$. Thus, the relevant goal with these operations is ensuring that the cut values do not decrease too much – i.e., preserving global/local connectivity. We will be interested in repeated application of h-splitting-off operations at a vertex from a given hypergraph to isolate that vertex while preserving global/local connectivity. We define these formally next.

▶ **Definition 2.** *Let $(G = (V, E), w : E \to \mathbb{Z}_+)$ be a hypergraph and $s \in V$.*

1. *We say that a hypergraph $(G^* = (V, E^*), w^* : E^* \to \mathbb{Z}_+)$ is a*
   - **a.** complete h-splitting-off at s from (G, w) *if $d_{(G^*,w^*)}(s) = 0$ and (G^*, w^*) is obtained from (G, w) by repeatedly applying h-splitting-off operations at s from the current hypergraph.*
   - **b.** complete g-splitting-off at s from (G, w) *if $d_{(G^*,w^*)}(s) = 0$ and (G^*, w^*) is obtained from (G, w) by repeatedly applying g-splitting-off operations at s from the current hypergraph.*
2. *Let (G^*, w^*) be a complete h-splitting-off/g-splitting-off at s from (G, w). We say that (G^*, w^*)*
   - **a.** *preserves local connectivity if $\lambda_{(G^*,w^*)}(u, v) = \lambda_{(G,w)}(u, v)$ for every distinct $u, v \in V \setminus \{s\}$ and*
   - **b.** *preserves global connectivity if* $\min\{\lambda_{(G^*,w^*)}(u, v) : u, v \in V \setminus \{s\}, u \neq v\} = \min\{\lambda_{(G,w)}(u, v) : u, v \in V \setminus \{s\}, u \neq v\}$.

Our first contribution in this work is the definition of h-splitting-off operations at a vertex from a hypergraph. To the best of our knowledge, this definition has not appeared in the literature before. A notion of hypergraph splitting-off motivated by hypergraph connectivity augmentation applications has been studied in the literature before [3, 6, 20]. These works have explored local connectivity preserving complete g-splitting-off at a vertex s from a hypergraph under the assumption that *all hyperedges incident to the vertex s are edges (i.e., have size at most 2)*. In contrast, our focus is on local connectivity preserving complete h-splitting-off at a vertex s from a hypergraph without any assumption on the size of the hyperedges incident to the vertex s (i.e., the vertex s could have arbitrary-sized hyperedges incident to it). We refer the reader to Figure 1 for an example of complete h-splitting-off at a vertex from a hypergraph.

We will primarily be concerned with complete *h-splitting-off* at a vertex from a *hypergraph* and complete *g-splitting-off* at a vertex from a *graph*. Complete g-splitting-off at a vertex from a *graph* is equivalent to the classical and well-studied notion of complete splitting-off at a vertex from a graph (for the definition of the classical notion in graphs, see [24, 49]). We cast the results of Lovász [45, 47] and Mader [48] in the framework of our definitions now. Let (G, w) be a *graph* and let s be a vertex in G. Lovász [45, 47] showed that if $d_{(G,w)}(\{s\})$ is even and $\min\{\lambda_{(G,w)}(u, v) : u, v \in V \setminus \{s\}\} \geq K$ for some $K \geq 2$, then there exists a *global* connectivity preserving complete g-splitting-off at the vertex s from (G, w).

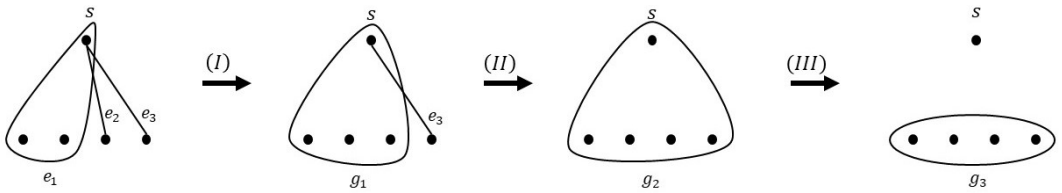


Figure 1 Example of (local connectivity preserving) complete h-splitting-off at a vertex s from a hypergraph. Consider the leftmost hypergraph where all hyperedge weights are one and the vertex s is as labeled. Operations (I) and (II) correspond to h-merge almost-disjoint hyperedges operations and Operation (III) corresponds to an h-trim hyperedge operation.

Mader [48] showed that if $d_{(G,w)}(\{s\})$ is even, there is no cut-edge[1] incident to s, and (G, w) is connected, then there exists a *local* connectivity preserving complete g-splitting-off at the vertex s from (G, w).

We compare and contrast complete h-splitting-off at a vertex from a hypergraph and complete g-splitting-off at a vertex from a graph. Complete h-splitting-off at a vertex s from a hypergraph enables the vertex s to exit the hypergraph by informing its incident hyperedges about how to merge and trim themselves in order to preserve degrees. In this sense, the definition of complete h-splitting-off at a vertex from a hypergraph serves the same role as complete g-splitting-off at a vertex from a graph. On the other hand, there are important differences between the two notions. Firstly, complete h-splitting-off at a vertex from a *graph* may not necessarily be a graph (owing to the creation of hyperedges of size at least 3) while it is an easy exercise to show that complete g-splitting-off at a vertex from a *graph* will necessarily be a graph. Secondly, local/global connectivity preserving complete g-splitting-off at a vertex from a graph may not exist – see Figure 2.

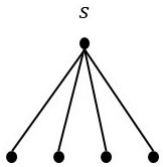


Figure 2 An example showing that global connectivity preserving complete g-splitting-off at a vertex from a graph may not exist. All edge weights are one and the vertex s is as labeled.

As our second main contribution, we show that local connectivity preserving complete h-splitting-off at a vertex from a hypergraph always exists and can be computed in strongly polynomial time (the rightmost hypergraph in Figure 1 is a local connectivity preserving complete h-splitting-off at the vertex s from the hypergraph in Figure 2).

▶ **Theorem 3.** *Given a hypergraph $(G = (V, E), w_G : E \to \mathbb{Z}_+)$ and a vertex $s \in V$, there exists a strongly polynomial-time algorithm to find a local connectivity preserving complete h-splitting-off at s from (G, w_G).*

A difference between Theorem 3 and the graph splitting-off results of Lovász and Mader is that Theorem 3 shows the existence of a local connectivity preserving complete h-splitting-off at a vertex from a hypergraph *without any assumptions on the hypergraph* whereas Lovász's

[1] Equivalently, for every edge $e \in \delta_G(s)$ with $w(e) = 1$, the removal of that edge does not disconnect the graph.

and Mader's results hold only under certain technical assumptions on the graph. In several applications of their results, additional arguments are needed to address cases where those technical assumptions do not hold. For this reason, we believe that Theorem 3 could be useful in simplifying the arguments involved in some of the applications of Lovász's and Mader's graph splitting-off results (e.g., we will later see that the edge version of Menger's theorem in undirected *graphs* follows in a straightforward fashion from Theorem 3).

A crude run-time of our algorithm that proves Theorem 3 is $O(|V|^6(|V|+|E|)^3)$. We understand that this run-time is impractical for applications. Nevertheless, we mention it here explicitly for the sake of completeness and as a potential starting point for future work: it would be interesting to design a near-linear time algorithm to find a local connectivity preserving complete h-splitting-off at a vertex from a weighted hypergraph.

▶ Remark 4. We note that existence of a local/global connectivity preserving complete h-splitting-off at a vertex from a hypergraph does not necessarily imply a polynomial-time algorithm to find it. This is because, a local/global connectivity preserving complete h-splitting-off at a vertex from a hypergraph (G, w_G) could contain exponential number of hyperedges although G contains only polynomial number of hyperedges. We give an example to illustrate this issue. Consider the graph $(G = (V, E), w_G)$ where G is the star graph on $n+1$ vertices with s being the center of the star and all edge weights are $2^{n-1}-1$. Consider the hypergraph $(H = (V, E_H), w_H)$ such that $E_H := \{e : e \subseteq V \setminus \{s\}$ and $|e| \geq 2\}$ with all hyperedge weights being one. The hypergraph (H, w_H) is a local connectivity preserving complete h-splitting-off at s from (G, w_G), but (H, w_H) has exponential number of hyperedges although (G, w_G) has only n edges. In order to design a polynomial-time algorithm to find a local connectivity preserving complete h-splitting-off at a vertex from a hypergraph, a necessary step is to show the existence of a local connectivity preserving complete h-splitting-off at a vertex from a hypergraph that contains only polynomially many *additional* hyperedges. For the star graph (G, w_G) with edge weights $2^{n-1}-1$ mentioned above, the hypergraph $(H' = (V, E_{H'}), w_{H'})$ containing only one hyperedge, namely $E_{H'} := \{V \setminus \{s\}\}$ with the weight of that hyperedge being $2^{n-1}-1$ is also a local connectivity preserving complete h-splitting-off at s from (G, w_G). One of the features of our algorithmic proof of Theorem 3 is the existence of a local connectivity preserving complete h-splitting-off at a vertex from a hypergraph that contains only polynomially many additional hyperedges.

As our third main contribution, we present two applications of Theorem 3.

Application 1: Constructive characterization of k-hyperedge-connected hypergraphs. For the purposes of this application, graphs and hypergraphs will refer to multi-graphs and multi-hypergraphs, respectively. Let k be a positive integer. A graph $G = (V, E)$ is *k-edge-connected* if $d_G(X) \geq k$ for every non-empty proper subset $X \subsetneq V$. Constructive characterization of k-edge-connected graphs is a central problem in graph theory. It is well-known that a graph is 1-edge-connected if and only if it containss a spanning tree. Robbins' [56] showed that a graph is 2-edge-connected if and only if it admits an ear decomposition (see [24] for definition of ear decomposition). Generalizing Robbins' result, Lovász [45, 47] gave a constructive characterization of k-edge-connected graphs for *even* k using his result on global connectivity preserving complete g-splitting-off at a vertex from a graph. Mader [48] gave a constructive characterization of k-edge-connected graphs for *odd* k using his result on local connectivity preserving complete g-splitting-off at a vertex from a graph. Motivated by these results, we present a constructive characterization of k-hyperedge-connected hypergraphs using our splitting-off result in Theorem 3. A hypergraph $G = (V, E)$ is defined to be *k-hyperedge-connected* if $d_G(X) \geq k$ for every non-empty proper subset $X \subsetneq V$.

Both Lovász's and Mader's constructive characterizations of k-edge-connected graphs are based on a pinching operation in graphs. Our constructive characterization of k-hyperedge-connected hypergraphs is also based on a pinching operation, but our pinching operation is defined for hypergraphs. We define this operation now (see Figure 3 for an example).

▶ **Definition 5.** *Let $G = (V, E)$ be a hypergraph and $p, k \in \mathbb{Z}_+$ be positive integers such that $p \leq k$. In* (k, p)-pinching *hyperedges of G, we obtain a new hypergraph by performing the following sequence of operations:*

1. *pick p distinct hyperedges $e_1, \ldots, e_p \in E$,*
2. *pick p positive integers $t_1, \ldots, t_p \in \mathbb{Z}_+$ such that $\sum_{i=1}^{p} t_i = k$,*
3. *for each $i \in [p]$, choose a partition of the hyperedge e_i into t_i non-empty parts $e_i = \biguplus_{j \in [t_i]} f_i^j$,*
4. *remove the hyperedges $e_1, \ldots, e_p$ from the hypergraph G,*
5. *add a new vertex s and hyperedges $\{f_i^j \cup \{s\} : j \in [t_i], i \in [p]\}$ to the hypergraph G.*

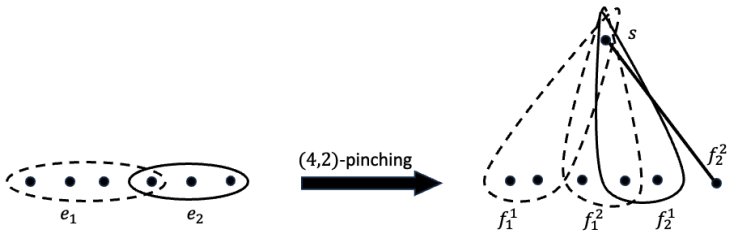


Figure 3 An example of a $(4, 2)$-pinching operation. Here, $t_1 = t_2 = 2$.

With this definition of pinching, we show the following constructive characterization of k-hyperedge-connected hypergraphs.

▶ **Theorem 6.** *Let $k \in \mathbb{Z}_+$ be a positive integer. A hypergraph $G = (V, E)$ is k-hyperedge-connected if and only if G can be obtained by starting from the single vertex hypergraph with no hyperedges and repeatedly applying one of the following two operations:*

1. *add a new hyperedge over a subset of vertices of the existing hypergraph, and*
2. *(k, p)-pinching hyperedges of the existing hypergraph for some positive integer $p \leq k$.*

Our proof of Theorem 6 is constructive: i.e., given a k-hyperedge-connected hypergraph G, our proof gives a polynomial-time algorithm to construct a sequence of hypergraphs $G_0, G_1, G_2, \ldots, G_t$, where G_0 is the single vertex hypergraph with no hyperedges, $G_t = G$ and for each $i \in [t]$, the hypergraph G_i is obtained from G_{i-1} by either adding a new hyperedge over a subset of vertices in G_{i-1} or by (k, p)-pinching hyperedges in G_{i-1} for some positive integer $p \leq k$.

▶ Remark 7. Robbins' constructive characterization of 2-edge-connected graphs and Lovász's constructive characterization of $2k$-edge-connected graphs find applications in graph orientation problems – e.g., Robbins' result leads to an algorithm to find a strongly connected orientation of 2-edge-connected graphs and Lovász's result leads to an algorithm to find a strongly k-arc-connected orientation of $2k$-edge-connected graphs. In fact, the latter leads to an alternative proof of Nash-Williams' *weak orientation theorem* [52]. Along the same vein, we hope that our above characterization of k-edge-connected hypergraphs might find applications in hypergraph orientation problems.

Application 2.1: Steiner Rooted k-arc-connected Orientation of Graphs. Orienting a graph to achieve high connectivity is a fundamental area in graph theory, combinatorial optimization, and algorithms. Let $G = (V, E)$ be an undirected graph. An *orientation* $\overrightarrow{G}$ of G is a directed graph obtained by assigning a direction to each edge of G. Let $G = (V, E)$ be an undirected graph, $T \subseteq V$ be a set of terminals, $r \in T$ be a root vertex, and k be a positive integer. An orientation $\overrightarrow{G}$ of G is defined to be *Steiner rooted k-arc-connected* if there exist k arc-disjoint paths in $\overrightarrow{G}$ from t to r for every terminal $t \in T \setminus \{r\}$. In Max Steiner Rooted-Connected Orientation problem, the goal is to find the maximum integer k and an orientation $\overrightarrow{G}$ of G such that $\overrightarrow{G}$ is Steiner rooted k-arc-connected. Max Steiner Rooted-Connected Orientation generalizes two classic problems in graph theory: The case of $|T| = 2$ is the max edge-disjoint $\{r, v\}$-paths problem and is solved via Menger's theorem. The case of $T = V$ is the max edge-disjoint spanning trees problem and is solved via Tutte and Nash-Williams' theorem [53, 59]. We mention that both these problems are also generalized by the Steiner Tree Packing problem and the associated Kriesell's conjecture [33, 40, 41], but we will not focus on that generalization.

Király and Lau [38] introduced the Max Steiner Rooted-Connected Orientation, showed that it is NP-hard, and gave a 2-approximation via an approximate min-max relation. We state their approximate min-max relation now. An undirected graph G is *Steiner k-edge-connected* if $\lambda_G(u, v) \geq k$ for every pair of distinct terminals $u, v \in T$. It is clear that if the graph G has a Steiner rooted k-arc-connected orientation, then G should be Steiner k-edge-connected. However, the converse is not necessarily true. Király and Lau observed that if the graph is Steiner $2k$-edge-connected, then it has a Steiner rooted k-arc-connected orientation.

▶ **Theorem 8** (Király and Lau [38]). *Let $G = (V, E)$ be an undirected graph, $T \subseteq V$ be a subset of terminals, $r \in T$ be the root vertex, and k be a positive integer. If G is Steiner $2k$-edge-connected, then it has a Steiner rooted k-arc-connected orientation.*

Király and Lau observed that Theorem 8 follows immediately from Nash-Williams' *strong orientation* theorem. Nash-Williams' *strong orientation* theorem [52] states that every undirected graph $G = (V, E)$ admits an orientation $\overrightarrow{G}$ such that $\lambda_{\overrightarrow{G}}(u, v) \geq \lfloor \lambda_G(u, v)/2 \rfloor$ for every distinct $u, v \in V$, where $\lambda_{\overrightarrow{G}}(u, v)$ is the maximum number of arc-disjoint directed paths from u to v in $\overrightarrow{G}$. In this work, we give an alternative proof of Theorem 8 that does not rely on Nash-Williams' strong orientation theorem. Instead, we use Theorem 3. Our proof strategy is unique since it proves an orientation result for *graphs* using tools developed for *hypergraphs*.

▶ Remark 9. Nash-Williams' proof of the strong orientation theorem [52] is a sophisticated inductive argument. Giving a simple and more insightful proof of the strong orientation theorem has been a central topic of interest in graph theory and combinatorial optimization (see [26]). Mader [48] gave a different proof of the strong orientation theorem using his local connectivity preserving splitting-off theorem, but his proof also involved sophisticated technical arguments. Frank [26] condensed the ideas of both Nash-Williams and Mader to present a proof of the strong orientation theorem using Mader's local connectivity preserving splitting-off, but it is still technically complicated. The technical complication in using Mader's local connectivity preserving splitting-off result arises from the assumptions that need to be satisfied by the vertex to be split-off. In contrast, our splitting-off result for hypergraphs (namely, Theorem 3) does not need any assumptions on the vertex to be split-off. In light of these considerations, our proof of Theorem 8 using Theorem 3 provides hope that Theorem 3 (or the ideas therein) could be used to give a conceptually simpler proof of Nash-Williams' strong orientation theorem.

Application 2.2: Steiner Rooted k-hyperarc-connected Orientation of Hypergraphs. Orienting *hypergraphs* is also a fundamental area in graph theory and combinatorial optimization (see Frank's book [24]) with far reaching implications. For example, Woodall's conjecture can be reformulated as a hypergraph orientation problem (see Conjecture 9.4.15 in [24]); moreover, hypergraph orientation results have recently been used in coding theory [1]. Király and Lau [38] showed that the approximate min-max relation in Theorem 8 also holds for hypergraphs. To state their result, we need some terminology in hypergraph orientations.

Let $G = (V, E)$ be a hypergraph. An *orientation* $\overrightarrow{G} = (V, E, \texttt{head} : E \to V)$ of G is a directed hypergraph obtained by assigning a unique head vertex $\texttt{head}(e) \in e$ for each $e \in E$. A pair $(e, \texttt{head}(e))$ is denoted as a *hyperarc* with the head of the hyperarc being $\texttt{head}(e)$ and the tails of the hyperarc being $e \setminus \texttt{head}(e)$. Let $G = (V, E)$ be a hypergraph, $T \subseteq V$ be a set of terminals, $r \in T$ be a root vertex, and k be a positive integer. An orientation $\overrightarrow{G}$ of G is defined to be *Steiner rooted k-hyperarc-connected* if there exist k hyperarc-disjoint paths in $\overrightarrow{G}$ from t to r for every terminal $t \in T \setminus \{r\}$. Here, a path from t to r in a directed hypergraph is an alternating sequence of distinct vertices and hyperarcs $t = v_1, a_1, v_2, a_2, ..., a_{\ell-1}, v_\ell = r$ such that v_i is a tail of a_i and v_{i+1} is the head of a_i for every $i \in [\ell - 1]$. We say that a hypergraph G is *Steiner k-hyperedge-connected* if $\lambda_G(u, v) \geq k$ for every pair of distinct terminals $u, v \in T$. It is clear that if the hypergraph G has a Steiner rooted k-hyperarc-connected orientation, then G should be Steiner k-hyperedge-connected. However, the converse is not necessarily true. Király and Lau [38] showed that if the hypergraph is Steiner $2k$-hyperedge-connected, then it has a Steiner rooted k-hyperarc-connected orientation.

▶ **Theorem 10** (Király and Lau [38]). *Let $G = (V, E)$ be a hypergraph, $T \subseteq V$ be a subset of terminals, $r \in T$ be the root vertex, and k be a positive integer. If G is Steiner $2k$-hyperedge-connected, then it has a Steiner rooted k-hyperarc-connected orientation.*

Király and Lau's proof of Theorem 10 was based on careful uncrossing and contractions. In this work, we give an alternative proof of Theorem 10 using Theorem 3. Our proof of Theorem 10 reveals the source of the 2-factor gap in the approximate min-max relation of Király and Lau for Max Steiner Rooted-Connected Orientation Problem: it arises from the 2-factor gap between connectivity and weak-partition-connectivity of hypergraphs (see Definition 5.1 for the definition of weak-partition-connectivity and Lemma 5.2 in the full version).

▶ Remark 11. Our proof technique for Theorems 8 and 10 using Theorem 3 – i.e., via the local-connectivity preserving splitting-off operation in hypergraphs – also leads to an alternate proof of Menger's theorem in *undirected* graphs and hypergraphs (edge-disjoint version). For details, we refer the reader to Section 5 of the full version where we discuss a proof of Menger's theorem using Theorem 3 as a warm-up towards a proof of Theorems 8 and 10.

Both Theorems 8 and 10 can be extended to weighted graphs/hypergraphs (by considering parallel copies of edges/hyperedges). The weighted version of Theorems 8 and 10 can also be shown to admit strongly polynomial-time algorithms using our proof strategy as well as the proof strategy of Király and Lau. We avoid stating the weighted versions in the interests of brevity.

1.1 Proof Technique for Theorem 6

We outline the proof technique for Theorem 6. The reverse direction follows by observing that if a hypergraph is k-hyperedge-connected, then both operations in the statement of the theorem preserve k-hyperedge-connectivity. We sketch a proof of the forward direction. The

proof is by induction on the number of hyperedges plus the number of vertices. First, suppose that there exists a hyperedge $e \in E$ such that $G - e$ is still k-hyperedge-connected. We note that deleting the hyperedge e is the inverse of operation (1). Consequently, the proof follows by deleting the hyperedge e, using the induction hypothesis on the resulting hypergraph H, and then noting that the hypergraph G is obtained from H by operation (1). Next, suppose that there does not exist a hyperedge $e \in E$ such that $G - e$ is k-hyperedge-connected. We call such a hypergraph to be minimally k-hyperedge-connected. In Lemma 4.1 of the full version, we show that a minimally k-hyperedge-connected hypergraph contains a vertex u with degree exactly k. By Theorem 3, there exists a global-connectivity preserving complete h-splitting-off at the vertex u from the hypergraph G. Let H be a global-connectivity preserving complete h-splitting-off at the vertex u from the hypergraph G. We note that complete h-splitting-off at u followed by deletion of the vertex u is the inverse of operation (2) at u. Consequently, the proof follows by using the induction hypothesis on the hypergraph $H - u$ and then noting that the hypergraph G is obtained from $H - u$ by operation (2).

1.2 Proof Technique for Theorems 8 and 10

We outline the proof technique for Theorem 10 and will remark after the proof about how it also implies a proof for Theorem 8. Our proof of Theorem 10 will be in three steps. Let us denote the set of non-terminals as *Steiner vertices*. Our first step is to obtain a hypergraph $H = (T, E_H)$ by applying our local connectivity preserving complete h-splitting-off at each Steiner vertex of G (sequentially, in arbitrary order of the Steiner vertices) and deleting the isolated vertices. We note that deleting the isolated vertices ensures that the vertex set of H is the set of terminals T. Moreover, our local connectivity preserving complete h-splitting-off ensures that the hypergraph H is $2k$-hyperedge-connected since the hypergraph G is Steiner $2k$-hyperedge-connected. Our second step is to show that this hypergraph $H = (T, E_H)$ admits a rooted k-hyperarc-connected orientation. A known characterization for the existence of a rooted k-hyperarc-connected orientation of a hypergraph is that the hypergraph is k-*weak-partition-connected* (see Definition 5.1 for the definition of weak-partition-connectivity and Theorem 5.1 for the characterization in the full version). We mention that the notion of weak-partition-connectivity in hypergraphs has been used recently in the context of coding theory [1, 31]. In order to use the characterization for the existence of a rooted k-hyperarc-connected orientation of a hypergraph, we relate the connectivity of a hypergraph to its *weak-partition-connectivity* and conclude that if H is $2k$-hyperedge-connected, then it is k-weak-partition-connected (see Lemma 5.2 in the full version). Consequently, the hypergraph H admits a rooted k-hyperarc-connected orientation. We note that such an orientation of H is equivalent to a Steiner rooted k-hyperarc-connected orientation of H since the vertex set of H is the set T of terminals. Our third step is to use this Steiner rooted k-hyperarc-connected orientation of H to obtain a Steiner rooted k-hyperarc-connected orientation of the hypergraph G: we will see that there is a natural way to extend the orientation of hyperedges while reversing the h-splitting-off operations to preserve Steiner rooted k-hyperarc-connected property (see Lemma 5.1 in the full version). This would complete the proof of Theorem 10.

We note that if the hypergraph G is a graph, then the same proof above obtains the required graph orientation, thus proving Theorem 8. In particular, to prove Theorem 8, we start from a graph $G = (V, E)$ that is Steiner $2k$-edge-connected, but our local connectivity preserving complete h-splitting-off operations at Steiner vertices results in a hypergraph $H = (T, E_H)$ that is $2k$-hyperedge-connected; by Lemma 5.2 in the full version, the hypergraph H is k-weak-partition-connected; now Theorem 5.1 in the full version gives a

rooted k-hyperarc-connected orientation of the resulting hypergraph. Such an orientation is extended to a Steiner rooted k-hyperarc-connected orientation of the *graph* $G = (V, E)$ using Lemma 5.1 in the full version. Essentially, the proof starts from the given graph, obtains a related hypergraph, orients that hypergraph, and extends that orientation of the hypergraph back into a desired orientation of the given graph.

Our proof technique for Theorems 8 and 10 also leads to an alternate proof of Menger's theorem in *undirected* graphs and hypergraphs (edge-disjoint version) – see Section 5 in the full version.

1.3 Proof Technique for Theorem 3

We prove a more general statement that implies Theorem 3. We begin with the definitions needed for the more general statement.

▶ **Definition 12.** *Let V be a finite set, $p : 2^V \to \mathbb{Z}$ be a set function, and $(H = (V, E), w : E \to \mathbb{Z}_+)$ be a hypergraph.*

1. *The set function p*
   a. *is* symmetric *if $p(X) = p(V - X)$ for every $X \subseteq V$, and*
   b. *is* skew-supermodular *if for every $X, Y \subseteq V$, at least one of the following inequalities hold:*
      i. $p(X) + p(Y) \leq p(X \cap Y) + p(X \cup Y)$.
      ii. $p(X) + p(Y) \leq p(X - Y) + p(Y - X)$.
2. *The coverage function $b_{(H,w)} : 2^V \to \mathbb{Z}_{\geq 0}$ is defined by $b_{(H,w)}(X) := \sum_{e \in B_H(X)} w(e)$ for every $X \subseteq V$, where $B_H(X) := \{e \in E : e \cap X \neq \emptyset\}$ for every $X \subseteq V$.*
3. *The hypergraph (H, w)* weakly covers *the function p if $b_{(H,w)}(X) \geq p(X)$ for every $X \subseteq V$.*
4. *The hypergraph (H, w)* strongly covers *the function p if $d_{(H,w)}(X) \geq p(X)$ for every $X \subseteq V$.*

If a hypergraph (H, w) strongly covers a function $p : 2^V \to \mathbb{Z}$, then it also weakly covers the function p. However, the converse is false – i.e., a weak cover is not necessarily a strong cover[2]. Bernáth and Király [5] showed that a weak cover of a *symmetric skew-supermodular* function can be converted to a strong cover of the same function by repeated *merging of disjoint hyperedges*. We recall their definition of the merging operation, discuss their result, and its significance now.

▶ **Definition 13.** *Let $(H = (V, E), w : E \to \mathbb{Z}_+)$ be a hypergraph. We use H_w to denote the unweighted multi-hypergraph over vertex set V containing $w(e)$ copies of every hyperedge $e \in E$. By* merging two disjoint hyperedges *of H_w, we refer to the operation of replacing them by their union in H_w. We will say that a hypergraph $(G = (V, E_G), c : E_G \to \mathbb{Z}_+)$ is obtained from (H, w) by* merging hyperedges *if the multi-hypergraph G_c is obtained from the multi-hypergraph H_w by repeatedly merging two disjoint hyperedges in the current hypergraph.*

Bernáth and Király showed the following result:

▶ **Theorem 14** (Bernáth and Király [5])**.** *Let $(H = (V, E), w : E \to \mathbb{Z}_+)$ be a hypergraph and $p : 2^V \to \mathbb{Z}$ be a symmetric skew-supermodular function such that $b_{(H,w)}(X) \geq p(X)$ for every $X \subseteq V$. Then, there exists a hypergraph $\left(H^* = (V, E^*), w^* : E^* \to \mathbb{Z}_+\right)$ such that*
(1) *$d_{(H^*,w^*)}(X) \geq p(X)$ for every $X \subseteq V$ and*
(2) *the hypergraph (H^*, w^*) is obtained by merging hyperedges of the hypergraph (H, w).*

[2] For example, consider the function $p : 2^V \to \mathbb{Z}$ defined by $p(X) := 1$ for every non-empty proper subset $X \subsetneq V$ and $p(\emptyset) := p(V) := 0$, and the hypergraph $(H = (V, E := \{\{u\} : u \in V\}), w : E \to \{1\})$.

We observe that Theorem 14 can be used to prove the existential version of Theorem 3: namely, for every hypergraph $(G = (V, E), w_G : E \to \mathbb{Z}_+)$ and a vertex $s \in V$, *there exists* a local connectivity preserving complete h-splitting-off at s from (G, w_G). This can be shown by setting up the hypergraph (H, w) and the function p suitably based on (G, w_G) and using Theorem 14 (see the first two paragraphs of the proof of Theorem 3.1 in Section 3 of the full version). We emphasize that this conclusion regarding hypergraph splitting-off from Bernáth and Király's result was not known before in the literature and is one of our contributions.

▶ Remark 15. We were also able to prove the existential version of Theorem 3 using *element-connectivity preserving reduction operations* (see [16] for the definition of element-connectivity and the notion of element-connectivity preserving reduction operations) – we omit the details of this alternate proof in the interests of brevity. The alternate proof does not seem to be helpful for the purposes of a strongly polynomial time algorithm. In fact, it remains open to design a strongly polynomial-time algorithm to perform *complete* element-connectivity preserving reduction operations in the weighted setting [16].

We recall that existence of a local connectivity preserving complete h-splitting-off at a vertex from a hypergraph does not immediately imply a polynomial-time algorithm – see the example in Remark 4. However, the above-mentioned proof of existence of a local-connectivity preserving complete splitting-off at an arbitrary vertex from a hypergraph (i.e., existential version of Theorem 3) via Theorem 14 suggests a natural approach towards designing a strongly polynomial time algorithm to find a local-connectivity preserving complete splitting-off at a given vertex from a given hypergraph: it suffices to prove a constructive version of Theorem 14 via a strongly polynomial-time algorithm. Towards this end, the example in Remark 4 suggests a necessary structural step towards a strongly polynomial-time algorithmic version of Theorem 14: we need to show Theorem 14 with the extra conclusion that the number of additional hyperedges in H^* is polynomial in the number of hyperedges and vertices in H.

Bernáth and Király proved Theorem 14 in the context of a reduction between certain hypergraph connectivity augmentation problems. For that reduction, the existential version of Theorem 14 is sufficient. However, for the purposes of our application to hypergraph splitting-off, we need an algorithmic version of Theorem 14. Bernáth and Király's proof of Theorem 14 is in fact algorithmic, but the run-time of the associated algorithm is not necessarily polynomial. Their proof implies that the number of additional hyperedges in the hypergraph (H^*, c^*) returned by their algorithm is at most $\sum_{e \in E} w(e)$ (i.e., $|E^*| - |E| \leq \sum_{e \in E} w(e)$) and the run-time of the algorithm is $O(\sum_{e \in E}(|e| + w(e))$. In particular, their run-time is polynomial only if the input weights are given in unary. Moreover, the exponential-sized hypergraph from Remark 4 could indeed arise as a consequence of their algorithm.

We address both the structural and the algorithmic issues mentioned above by proving a stronger algorithmic version of Theorem 14. In order to phrase an algorithmic version of Theorem 14, we need suitable access to the function p. Bernáth and Király [5] suggested access to a certain function maximization oracle associated with the function p that we describe below.

▶ **Definition 16.** *Let $p : 2^V \to \mathbb{Z}$ be a set function. p-*`max-sc-Oracle`$\left((G_0, c_0), S_0, T_0\right)$ *takes as input a hypergraph $(G_0 = (V, E_0), c_0 : E_0 \to \mathbb{Z}_+)$ and disjoint sets $S_0, T_0 \subseteq V$, and returns a tuple $(Z, p(Z))$, where Z is an optimum solution to the following problem:*

$$\max \left\{ p(Z) - d_{(G_0, c_0)}(Z) : S_0 \subseteq Z \subseteq V - T_0 \right\}. \qquad (p\text{-}\texttt{max-sc-Oracle})$$

For the purposes of our application (namely local connectivity preserving complete h-splitting-off at a vertex from a hypergraph), the above-mentioned function maximization oracle can be implemented to run in strongly polynomial time (see Lemma 3.1 in the full version). Using the above mentioned oracle, we prove the following algorithmic version of Theorem 14.

▶ **Theorem 17.** *Let $(H = (V, E), w : E \to \mathbb{Z}_+)$ be a hypergraph and $p : 2^V \to \mathbb{Z}$ be a symmetric skew-supermodular function such that $b_{(H,w)}(X) \geq p(X)$ for every $X \subseteq V$. Then, there exists a hypergraph $\left(H^* = (V, E^*), w^* : E^* \to \mathbb{Z}_+\right)$ such that*
(1) *$d_{(H^*,w^*)}(X) \geq p(X)$ for every $X \subseteq V$,*
(2) *the hypergraph (H^*, w^*) is obtained by merging hyperedges of the hypergraph (H, w), and*
(3) *$|E^*| - |E| = O(|V|)$.*
Furthermore, given a hypergraph $(H = (V, E), w : E \to \mathbb{Z}_+)$ and access to p-`max-sc-Oracle` of a symmetric skew-supermodular function $p : 2^V \to \mathbb{Z}$ where $b_{(H,w)}(X) \geq p(X)$ for every $X \subseteq V$, there exists an algorithm that runs in $O(|V|^3(|V| + |E|)^2)$ time using $O(|V|^3(|V| + |E|))$ queries to p-`max-sc-Oracle` and returns a hypergraph $\left(H^* = (V, E^*), w^* : E^* \to \mathbb{Z}_+\right)$ satisfying the above three properties. The run-time includes the time to construct the hypergraphs used as input to the queries to p-`max-sc-Oracle`. Moreover, for each query to p-`max-sc-Oracle`, the hypergraph (G_0, c_0) used as input to the query has $O(|V|)$ vertices and $O(|V| + |E|)$ hyperedges.

Theorem 17 is a strengthening of Theorem 14 in two ways. Firstly, our theorem shows the existence of a hypergraph that not only satisfies properties (1) and (2), but also satisfies property (3) – i.e., the number of additional hyperedges in the returned hypergraph is *linear* in the size of the vertex set. Secondly, our Theorem 17 shows the existence of a strongly polynomial-time algorithm that returns a hypergraph satisfying the three properties. Our main contribution is modifying Bernáth and Király's algorithm and analyzing the modified algorithm to bound the number of additional hyperedges and the run-time. We mention that property (3) cannot be tightened to guarantee that $|E^* - E| = O(|V|)$ – we were able to construct an example where $|E^* - E| = \Omega(|V|^2)$ (see Appendix A of the full version).

Theorem 17 immediately leads to a proof of Theorem 3 (see Theorem 3.1 and its proof in Section 3 of the full version). Instead of using Theorem 17 as a black-box, if we delve into the proof of it in the context of the proof of Theorem 3, we obtain the following theorem:

▶ **Theorem 18.** *Let $(G = (V, E), w : E \to \mathbb{Z}_+)$ be a hypergraph and $s \in V$. Then, there exists a hypergraph $(G' = (V, E'), w' : E' \to \mathbb{Z}_+)$ obtained by applying a h-splitting-off operation at s from (G, w) such that $\lambda_{(G',w')}(u, v) = \lambda_{(G,w)}(u, v)$ for every distinct $u, v \in V \setminus \{s\}$.*

We omit the proof of Theorem 18 in the interests of brevity. Theorem 18 closely resembles the *existential edge splitting-off* results of Lovász [45, 47] and Mader [48] for graphs. Lovász's and Mader's existential edge splitting-off results for graphs are important since they have been used to simplify the proofs of fundamental results in graph theory – e.g., Nash-Williams' Strong Orientation Theorem. On the other hand, Theorem 18 does not immediately imply a strongly polynomial-time algorithm for finding a local connectivity preserving complete h-splitting off at a vertex from a given weighted hypergraph. So, Theorem 3 may be useful in algorithmic contexts while Theorem 18 may be useful in graph-theoretical contexts.

1.4 Proof Technique for Theorem 17

In this section, we describe our proof technique for the existential result in Theorem 17. The algorithmic results in that theorem follow from the existential result using standard algorithmic tools for submodular functions, so we focus only on outlining a proof of the

existential result. Let $(H = (V, E), w : E \to \mathbb{Z}_+)$ be a hypergraph and $p : 2^V \to \mathbb{Z}_+$ be a symmetric skew-supermodular function such that (H, w) weakly covers the function p. Our goal is to show that there exists a hypergraph $(H^* = (V, E^*), w^* : E \to \mathbb{Z}_+)$ such that

(1) (H^*, w^*) strongly covers the function p,

(2) (H^*, w^*) is obtained by merging hyperedges of the hypergraph (H, w), and

(3) $|E^*| - |E| = O(|V|)$.

Preliminaries. We define a set $X \subseteq V$ to be (p, H, w)-tight if $b_{(H,w)}(X) = p(X)$. For a function p and hypergraph (H, w), let $T_{p,H,w}$ denote the family of (p, H, w)-tight sets and let $\mathcal{T}_{p,H,w}$ be the family of inclusionwise maximal sets in $T_{p,H,w}$. We will need the following two operations:

(i) For hyperedges $e, f \in E$ and a positive integer $\alpha \leq \min\{w(e), w(f)\}$, the operation Merge $((H, w), e, f, \alpha))$ returns the hypergraph obtained from (H, w) by decreasing the weight of hyperedges e and f by α and increasing the weight of the hyperedge $e \cup f$ by α. All hyperedges with zero weight are discarded.

(ii) For a hyperedge $e \in E$ and a positive integer $\alpha \leq w(e)$, the operation Reduce $((H, w), e, \alpha)$ returns the hypergraph obtained by decreasing the weight of the hyperedge e by α. All hyperedges with zero weight are discarded.

Algorithm of [5]. Our proof of the existential result builds on the techniques of Bernáth and Király [5] who proved the existence of a hypergraph $(H^* = (V, E^*), w^* : E^* \to \mathbb{Z}_+)$ satisfying properties (1) and (2), so we briefly recall their techniques. We present the algorithmic version of their proof since it will be useful for our purposes.

The proof in [5] is inductive, and consequently, the algorithm implicit in their proof is recursive. The algorithm takes as input a hypergraph $((H = (V, E), w : E \to \mathbb{Z}_+)$ and a symmetric skew-supermodular function $p : 2^V \to \mathbb{Z}$ such that the hypergraph (H, w) weakly covers the function p. If $w(E) = 0$, then the algorithm is in its base case and returns the empty hypergraph. Otherwise, $w(E) > 0$; the algorithm chooses an arbitrary hyperedge $e \in E$ and defines hypergraphs (H_0, w_0) and (H', w') and the function p' by considering two cases. First, suppose that the hyperedge e is not contained in any set of the family $\mathcal{T}_{p,H,w}$. In this case, the algorithm defines (H_0, w_0) to be the hypergraph on vertex set V consisting of a single hyperedge e with $w_0(e) = 1$, constructs the hypergraph $(H', w') := \text{Reduce}((H, w), e, 1)$, and defines the function $p' := p - d_{(H_0,w_0)}$. Second, suppose that the hyperedge e is contained in some set $X \in \mathcal{T}_{p,H,w}$. It can be shown that there exists a hyperedge $f \in E$ such that $f \subseteq V - X$. In this case, the algorithm defines (H_0, w_0) to be the empty hypergraph on vertex set V, constructs the hypergraph $(H', w') := \text{Merge}((H, w), e, f, 1)$, and defines the function $p' := p$. In both cases, the algorithm recurses on the inputs (H', w') and p' to obtain a hypergraph (H_0^*, w_0^*) and returns the hypergraph $(H^*, w^*) = (H_0^* + H_0, w_0^* + w_0)$. Here, the hyperedges of H^* are the union of the hyperedges of H_0^* and H_0 with the weight $w^*(e)$ of a hyperedge e being the sum of the weights $w_0^*(e) + w_0(e)$ if the hyperedge e is present in both H_0^* and H_0, being $w_0^*(e)$ if the hyperedge e is present only in H_0^*, and being $w_0(e)$ if the hyperedge e is present only in H_0.

We note that $w'(E') = w(E) - 1$. Furthermore, it can be shown that the function p' is symmetric skew-supermodular and the hypergraph (H', w') weakly covers the function p'. Consequently, by induction on $w(E)$, the algorithm can be shown to terminate in $w(E)$ recursive calls and returns a hypergraph satisfying properties (1) and (2). Moreover, the number of additional hyperedges in the returned hypergraph is at most the number of recursive calls where the Merge operation is performed, which is also at most $w(E)$. Thus,

in order to reduce the number of additional hyperedges and to design a strongly polynomial-time algorithm, our goal is to reduce the recursion depth of the algorithm. We emphasize that the recursion depth of Bernáth and Király's algorithm could indeed be exponential (the exponential sized example mentioned in Remark 4 could arise in the execution of their algorithm), so we need to necessarily modify their algorithm.

Preprocessing for Additional Structure. Similar to Bernáth and Király's algorithm, our algorithm also takes as input a hypergraph $(H = (V, E), w : E \to \mathbb{Z}_+)$ and a symmetric skew-supermodular function $p : 2^V \to \mathbb{Z}$ such that the hypergraph (H, w) weakly covers the function p. However, unlike Bernáth and Király's algorithm, our algorithm performs a preprocessing step so that the inputs (H, w) and p satisfy the following two additional conditions:

(a) every hyperedge $e \in E$ is contained in some set of $\mathcal{T}_{p,H,w}$ and

(b) the degree of every vertex in (H, w) is non-zero.

As a consequence of these additional conditions, the family $\mathcal{T}_{p,H,w}$ will be a *disjoint family*, a property that we leverage heavily throughout our analysis. Furthermore, we modify Bernáth and Király's algorithm to ensure that these conditions hold during every recursive call.

Our Algorithm. We now describe our modification of the above-mentioned Bernáth and Király's algorithm to reduce the recursion depth. Our algorithm is also recursive and its base case is the same as that of Bernáth and Király's algorithm (i.e., $w(E) = 0$). During recursive cases (i.e., if $w(E) > 0$), instead of performing one of the two (i.e., Merge or Reduce) operations, our algorithm performs both operations in a sequential fashion. In particular, we find a pair of disjoint hyperedges $e, f \in E$ contained in distinct sets of $\mathcal{T}_{p,H,w}$ (such a pair exists by condition (a) and the arguments of Bernáth and Király mentioned above). Next, instead of performing one Merge operation (as was done by Bernáth and Király's algorithm), we perform as many Merge operations as possible using the hyperedges e and f. Formally, let

$$\alpha^M := \max\left\{\alpha \in \mathbb{Z}_+ : \text{hypergraph returned by } \textsc{Merge}((H, w), e, f, \alpha) \text{ weakly covers } p\right\}.$$

We let $(H^M, w^M) := \textsc{Merge}((H, w), e, f, \alpha^M)$ and $p^M := p$. Next, instead of recursing on $((H^M, w^M), p^M)$ (as was done by Bernáth and Király's algorithm), we perform as many Reduce operations as possible on the newly created hyperedge $e \cup f$. Formally, let

$$\alpha^R := \max\left\{\alpha \in \mathbb{Z}_{\geq 0} : \begin{array}{c}\text{Hypergraph returned by } \textsc{Reduce}((H^M, w^M), e \cup f, \alpha) \\ \text{weakly covers the function } (p^M - d_{(H_0^\alpha, w_0^\alpha)})\end{array}\right\}$$

where (H_0^α, w_0^α) denotes the hypergraph on vertex set V consisting of a single hyperedge $e \cup f$ with $w_0^\alpha(e \cup f) = \alpha$. We construct the hypergraph $(H_0, w_0) := (H_0^{\alpha^R}, w_0^{\alpha^R})$, the hypergraph $(H^R, w^R) := \textsc{Reduce}((H^M, w^M), e \cup f, \alpha^R)$ and define the function $p^R := p^M - d_{(H_0^{\alpha^R}, w_0^{\alpha^R})}$. This immediate reduce step ensures that the hypergraph (H^R, w^R) and the function p^R satisfy condition (a) – i.e., every hyperedge in (H^R, w^R) is contained in some set of $\mathcal{T}_{p^R,H^R,w^R}$. Finally, we compute sets $\mathcal{Z} := \{u \in V : b_{(H^R,w^R)}(u) = 0\}$ and $V' := V - \mathcal{Z}$, hypergraph $(H' := (V', E' := E^R), w' := w^R)$, and define the function $p' : 2^{V'} \to \mathbb{Z}$ by $p'(X) := \max\{p(X \cup R) : R \subseteq \mathcal{Z}\}$ for every $X \subseteq V'$ – this final step can be viewed as a clean up step since it gets rid of vertices that are not incident to any hyperedges (and revises the function p appropriately). This clean up step ensures that the hypergraph (H', w') satisfies condition (b) – i.e., the degree of every vertex in (H', w') is non-zero. It can be shown that

the function p' is symmetric skew-supermodular and the hypergraph (H', w') weakly covers the function p'. Furthermore, the function p' and hypergraph (H', w') satisfy conditions (a) and (b). We recursively call the algorithm on input $((H', w'), p')$ to obtain a hypergraph (H_0^*, w_0^*). We obtain the hypergraph (G, c) from (H_0^*, w_0^*) by adding the vertices $\mathcal{Z}$ and return the hypergraph $(G + H_0, c + w_0)$. By induction on the total weight of hyperedges in the input hypergraph, it can be shown that our algorithm returns a hypergraph satisfying properties (1) and (2) and also terminates within a finite number of recursive calls.

Recursion Depth and Potential Functions. We now sketch our proof to show that the recursion depth of our algorithm is $|E| + O(|V|)$. We note that this also bounds the number of additional hyperedges in the hypergraph returned by the algorithm, and consequently this hypergraph also satisfies property (3). Let ℓ be the number of recursive calls made by the algorithm on the input instance $((H, w), p)$. We partition the set $[\ell]$ of recursive calls into two parts: let $P_1 \subseteq [\ell]$ be the set of recursive calls during which the merged hyperedge $e \cup f$ survives in the hypergraph (H', w') that is input to the subsequent recursive call and $P_2 \subseteq [\ell]$ be the recursive calls during which the merged hyperedge $e \cup f$ does not survive in the hypergraph (H', w') that is input to the subsequent recursive call. We note that $[\ell] = P_1 \uplus P_2$. We bound $|P_1|$ and $|P_2|$ separately using certain carefully designed potential functions.

First, we show that $|P_1| = O(|V|)$ as follows: for $i \in [\ell]$, consider the maximal tight set family $\mathcal{T}_i = \mathcal{T}_{p_i, H_i, w_i}$ where $((H_i, w_i), p_i)$ is the input to the i^{th} recursive call. Also, let $\mathcal{T}_{\leq 1} := \mathcal{T}_1$ and $\mathcal{T}_{\leq i} := \mathcal{T}_i \cup \{X \cap V_i : X \in \mathcal{T}_{\leq i-1}\}$ for integers i where $2 \leq i \leq \ell$ and V_i is the ground set of the input to the i^{th} recursive call. Thus, $\mathcal{T}_{\leq i}$ is the *projection* of all the maximal tight sets encountered in the first i recursive calls of the algorithm onto the ground set of the inputs at the i^{th} recursive call. We show that $\mathcal{T}_{\leq i}$ is laminar for every $i \in [\ell]$ (Lemma 6.11 in the full version). However, $|\mathcal{T}_{\leq i}|$ is not necessarily non-decreasing with i since projection of a set family to a subset could result in the loss of sets from the family. Consequently, $|\mathcal{T}_{\leq i}|$ is not suitable as a potential function to measure progress. Instead, we use the potential function $\phi(i) := |\mathcal{T}_{\leq i}| + 3|\mathcal{Z}_{\leq i-1}|$, where $\mathcal{Z}_{\leq i}$ is the union of the sets $\mathcal{Z}$ computed up to the i^{th} recursive call. We show that $\phi(i)$ is non-decreasing and strictly increases if $i \in P_1$ (Claim 6.4 in Lemma 6.12 of the full version). Consequently, $|P_1| = O(|V|)$.

Secondly, we bound $|P_2|$ as follows. We use a lookahead-potential function: let $\Phi_1(i)$ be the number of recursive calls between i and ℓ during which the merged hyperedge $e \cup f$ survives in the hypergraph (H', w') that is input to the subsequent recursive call and let $\Phi(i) := |E_i| + \Phi_1(i)$, where E_i is the set of hyperedges in the hypergraph (H_i, w_i) input to the i^{th} recursive call. We show that $\Phi(i)$ is non-increasing and strictly decreases if $i \in P_2$ (Claim 6.5 in Lemma 6.12 of the full version). Hence, $|P_2| \leq \Phi(1) - \Phi(\ell) \leq \Phi(1) \leq |E_1| + |P_1| = |E| + O(|V|)$, where the last equality is because of the bound on $|P_1|$ from the previous paragraph.

▶ Remark 19. Our key technical contributions are twofold. Our first key technical contribution is identifying conditions (a) and (b) under which $\mathcal{T}_{p,H,w}$ becomes a disjoint family. We ensure that conditions (a) and (b) hold in every recursive call by performing *immediate* reduction and clean-up steps in the algorithm. Our second key technical contribution is identifying appropriate potential functions to measure progress of the algorithm. The disjointness of $\mathcal{T}_{p,H,w}$ was crucial for identifying the laminar structure of the family of projected maximal tight sets across recursive calls, which was subsequently helpful in bounding the number of recursive calls corresponding to P_1. Moreover, we bound the number of recursive calls corresponding to P_2 using a lookahead-potential function that relates $|P_2|$ to $|P_1|$. As discussed above, the additive $|E|$ in the recursion depth comes from the bound on $|P_2|$.

2 Conclusion

We introduced a splitting-off operation in (weighted) hypergraphs. Our contribution on this front is conceptualizing an appropriate notion of splitting-off in hypergraphs. Next, we proved that for every hypergraph there exists a local-connectivity preserving complete h-splitting-off at an arbitrary vertex from the hypergraph. Although our proof of existence follows from previously known existential results of Bernáth and Király [5] for an abstract function cover problem, our main contribution is identifying that our newly introduced notion of hypergraph splitting-off falls within their framework. Next, we designed a strongly polynomial-time algorithm to find a local-connectivity preserving complete splitting-off at a vertex. This involved substantial technical challenges to overcome. In particular, our main contribution towards the strongly polynomial-time algorithm is a strengthening of the above-mentioned existential result: we showed that there exists a local connectivity preserving complete h-splitting-off at an arbitrary vertex from the hypergraph *in which the number of additional hyperedges is polynomial in the number of vertices*. The strongly polynomial-time algorithm follows from our techniques to achieve this stronger existential result via standard algorithmic tools in submodularity. Finally, we illustrated the usefulness of the existence of local connectivity preserving complete h-splitting-off at an arbitrary vertex from a hypergraph by presenting two applications. Our first application is to give a constructive characterization of k-hyperedge-connected hypergraphs. Our second application is to give an alternative proof of an approximate min-max relation for the max Steiner rooted-connected orientation problem in graphs and hypergraphs. Two notable features of our proof of this relation for *graphs* are that (1) it avoids the strong orientation theorem of Nash-Williams and (2) it proves a result for graphs using tools developed for hypergraphs.

Local and global connectivity preserving complete splitting-off at a vertex from a graph is a powerful operation for graphs. It finds applications in a variety of graph problems including graph orientation [25, 26, 38, 48], connectivity augmentation [2, 22, 23, 24, 49], minimum cuts enumeration [29, 32, 50], network design [12, 17, 35, 46], tree packing [7, 33, 40, 41], congruency-constrained cuts [51], and approximation algorithms for TSP [8, 30]. We believe that our local connectivity preserving complete splitting-off results for hypergraphs is likely to find future applications akin to its counterpart in graphs. We mention some of the open questions raised by our work:

1. Our work focused on local connectivity preserving complete *h-splitting-off* at a vertex from a hypergraph. We gave an example showing that local/global connectivity preserving complete *g-splitting-off* at a vertex from a hypergraph may not exist (Figure 2). Are there sufficient conditions to guarantee local/global connectivity preserving complete *g-splitting-off* at a vertex from a hypergraph? We recall that Lovász's [45, 47] and Mader's [48] results give sufficient conditions to guarantee local and global connectivity preserving complete g-splitting-off at a vertex from a *graph*.
2. As one of the applications of our splitting-off result, we presented an alternative proof of an approximate min-max relation for the max Steiner *rooted*-connected orientation problem in hypergraphs. The computational complexity of a closely related hypergraph orientation problem is open: In max Steiner connected orientation problem in hypergraphs, the input is a hypergraph $G = (V, E)$ and a subset T of terminals. The goal is to find the maximum k and an orientation $\overrightarrow{G}$ of G such that $\overrightarrow{G}$ contains k hyperarc-disjoint paths from u to v for every pair of distinct terminals $u, v \in T$. Max Steiner connected orientation problem in *graphs* is solvable in polynomial time via the Nash-Williams' strong orientation theorem. Is max Steiner connected orientation problem in *hypergraphs* solvable in polynomial time?

References

1. O. Alrabiah, V. Guruswami, and R. Li. Randomly punctured reed–solomon codes achieve list-decoding capacity over linear-sized fields. In *(To appear) Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC, 2024.
2. J. Bang-Jensen, A. Frank, and B. Jackson. Preserving and increasing local edge-connectivity in mixed graphs. *SIAM Journal on Discrete Mathematics*, 8(2):155–178, 1995.
3. J. Bang-Jenson and B. Jackson. Augmenting hypergraphs by edges of size two. *Mathematical Programming*, 84:467–481, 1999.
4. N. Bansal, O. Svensson, and L. Trevisan. New notions and constructions of sparsification for graphs and hypergraphs. In *IEEE 60th Annual Symposium on Foundations of Computer Science*, FOCS, pages 910–928, 2019.
5. A. Bernáth and T. Király. Covering skew-supermodular functions by hypergraphs of minimum total size. *Operations Research Letters*, 37(5):345–350, 2009.
6. A. Bernáth and T. Király. A unifying approach to splitting-off. *Combinatorica*, 32:373–401, 2012.
7. A. Bhalgat, R. Hariharan, T. Kavitha, and D. Panigrahi. Fast Edge Splitting and Edmonds' Arborescence Construction for Unweighted Graphs. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 455–464, 2008.
8. Jannis Blauth and Martin Nägele. An improved approximation guarantee for prize-collecting tsp. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 1848–1861, New York, NY, USA, 2023. Association for Computing Machinery.
9. R. Cen, W. He, J. Li, and D. Panigrahi. Steiner connectivity augmentation and splitting-off in poly-logarithmic maximum flows. In *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 2449–2488, 2023.
10. R. Cen, J. Li, and D. Panigrahi. Augmenting edge connectivity via isolating cuts. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 3237–3252, 2022.
11. R. Cen, J. Li, and D. Panigrahi. Edge connectivity augmentation in near-linear time. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC, pages 137–150, 2022.
12. Y. H. Chan, W. S. Fung, L. C. Lau, and C. K. Yung. Degree Bounded Network Design with Metric Costs. *SIAM Journal on Computing*, 40(4):953–980, 2011. Prelim. version in FOCS 2008.
13. K. Chandrasekaran and C. Chekuri. Hypergraph k-cut for fixed k in deterministic polynomial time. *Mathematics of Operations Research*, 47(4), 2022. Prelim. version in FOCS 2020.
14. K. Chandrasekaran and C. Chekuri. Min-max partitioning of hypergraphs and symmetric submodular functions. *Combinatorica*, 43:455–477, 2023. Prelim. version in SODA 2021.
15. K. Chandrasekaran, C. Xu, and X. Yu. Hypergraph k-Cut in randomized polynomial time. *Mathematical Programming*, 186:85–113, 2021. Prelim. version in SODA 2018.
16. C. Chekuri and N. Korula. A Graph Reduction Step Preserving Element-Connectivity and Packing Steiner Trees and Forests. *SIAM Journal on Discrete Mathematics*, 28(2):577–597, 2014. Prelim. version in ICALP 2009.
17. C. Chekuri and B. Shepherd. Approximate Integer Decompositions for Undirected Network Design Problems. *SIAM J. Discrete Math.*, 23:163–177, 2008.
18. C. Chekuri and C. Xu. Minimum cuts and sparsification in hypergraphs. *SIAM Journal on Computing*, 47(6):2118–2156, 2018. Prelim. version in SODA 2016.
19. Y. Chen, S. Khanna, and A. Nagda. Near-linear size hypergraph cut sparsifiers. In *IEEE 61st Annual Symposium on Foundations of Computer Science*, FOCS, pages 61–72, 2020.
20. B. Cosh. *Vertex splitting and connectivity augmentation in hypergraphs*. PhD thesis, University of London, 2000.
21. K. Fox, D. Panigrahi, and F. Zhang. Minimum cut and minimum k-cut in hypergraphs via branching contractions. *ACM Trans. Algorithms*, 19(2), 2023. Prelim. version in SODA 2019.

22 A. Frank. Augmenting graphs to meet edge-connectivity requirements. *SIAM Journal on Discrete Mathematics*, 5(1):25–53, 1992. Prelim. version in FOCS 1990.

23 A Frank. Connectivity augmentation problems in network design. *Mathematical Programming: State of the Art 1994*, pages 34–63, 1994.

24 A. Frank. *Connections in Combinatorial Optimization*. Oxford University Press, Oxford, 2011.

25 A. Frank and Z. Király. Graph orientations with edge-connection and parity constraints. *Combinatorica*, 22:47–70, 2002.

26 András Frank. Applications of submodular functions. *Surveys in Combinatorics, 1993 (Keele)*, pages 85–136, 1993.

27 H. N. Gabow. Efficient splitting off algorithms for graphs. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, STOC, pages 696–705, 1994.

28 M. Ghaffari, D. Karger, and D. Panigrahi. Random Contractions and Sampling for Hypergraph and Hedge Connectivity. In *Proceedings of the 28th annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 1101–1114, 2017.

29 M. X. Goemans. Approximate Edge Splitting. *SIAM Journal on Discrete Mathematics*, 14(1):138–141, 2001.

30 M.X. Goemans and D.J. Bertsimas. Survivable networks, linear programming relaxations and the parsimonious property. *Mathematical Programming*, 60:145–166, 1993.

31 Z. Guo, R. Li, C. Shangguan, I. Tamo, and M. Wootters. Improved List-Decodability and List-Recoverability of Reed-Solomon Codes via Tree Packings. In *IEEE 62nd Annual Symposium on Foundations of Computer Science*, FOCS, pages 708–719, 2022.

32 M. Henzinger and D. Williamson. On the number of small cuts in a graph. *Information Processing Letters*, 59:41–44, 1996.

33 K. Jain, M. Mahdian, and M. R. Salavatipour. Packing Steiner Trees. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 266–274, 2003.

34 A. Jambulapati, Y. P. Liu, and A. Sidford. Chaining, group leverage score overestimates, and fast spectral hypergraph sparsification. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC, pages 196–206, 2023.

35 T. Jordán. On minimally k-edge-connected graphs and shortest k-edge-connected Steiner networks. *Discrete Applied Mathematics*, 131(2):421–432, 2003.

36 M. Kapralov, R. Krauthgamer, J. Tardos, and Y. Yoshida. Towards tight bounds for spectral sparsification of hypergraphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC, pages 598–611, 2021.

37 M. Kapralov, R. Krauthgamer, J. Tardos, and Y. Yoshida. Spectral Hypergraph Sparsifiers of Nearly Linear Size. In *IEEE 62nd Annual Symposium on Foundations of Computer Science*, FOCS, pages 1159–1170, 2022.

38 T. Király and L. C. Lau. Approximate min–max theorems for Steiner rooted-orientations of graphs and hypergraphs. *Journal of Combinatorial Theory, Series B*, 98:1233–1252, November 2008. Prelim. version in FOCS 2006.

39 D. Kogan and R. Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS, pages 367–376, 2015.

40 M. Kriesell. Edge-disjoint trees containing some given vertices in a graph. *J. Comb. Theory Ser. B*, 88:53–63, 2003.

41 L. C. Lau. An Approximate Max-Steiner-Tree-Packing Min-Steiner-Cut Theorem. *Combinatorica*, 27:71–90, 2007. Prelim. version in FOCS 2004.

42 L. C. Lau and C. K. Yung. Efficient Edge Splitting-Off Algorithms Maintaining All-Pairs Edge-Connectivities. *SIAM Journal on Computing*, 42(3):1185–1200, 2013. Prelim. version in IPCO 2010.

43 J. R. Lee. Spectral hypergraph sparsification via chaining. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC, pages 207–218, 2023.

44 P. Li and O. Milenkovic. Inhomogeneous hypergraph clustering with applications. *Advances in neural information processing systems*, 30, 2017.

45 L. Lovász. Lecture. *Presented in a Conference on Graph Theory, Prague*, 1974.

46 L. Lovász. On some connectivity properties of Eulerian graphs. *Acta Math. Hungarica*, 28:129–138, 1976.

47 L. Lovász. *Combinatorial Problems and Exercises, Second Edition*. American Mathematical Soc., 1993. First Edition: North-Holland, Amsterdam, 1979.

48 W. Mader. A reduction method for edge-connectivity in graphs. In *Annals of Discrete Mathematics*, volume 3, pages 145–164. Elsevier, 1978.

49 H. Nagamochi and T. Ibaraki. *Algorithmic Aspects of Graph Connectivity*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2008.

50 H. Nagamochi, K. Nishimura, and T. Ibaraki. Computing all small cuts in an undirected network. *SIAM Journal on Discrete Mathematics*, 10(3):469–481, 1997.

51 M. Nägele and R. Zenklusen. A new contraction technique with applications to congruency-constrained cuts. *Mathematical Programming*, 183(2):455–481, 2020. Prelim. version in IPCO 2019.

52 C. St. J. A. Nash-Williams. On orientations, connectivity and odd vertex pairings in finite graphs. *Canad. J. Math*, 12:555–567, 1960.

53 C. St. J. A. Nash-Williams. Edge disjoint spanning trees of finite graphs. *J. London Math. Soc.*, 36:445–450, 1961.

54 S. Ornes. How big data carried graph theory into new dimensions. *Quanta Magazine*, 2021. URL: `https://www.quantamagazine.org/how-big-data-carried-graph-theory-into-new-dimensions-20210819/`.

55 K. Quanrud. Quotient sparsification for submodular functions. Manuscript available at `kentquanrud.com`, November 2022.

56 H. E. Robbins. A Theorem on Graphs with an Application to a Problem of Traffic Control. *Amer. Math. Monthly*, 46:281–283, 1939.

57 S. Schlag, T. Heuer, L. Gottesbüren, Y. Akhremtsev, C. Schulz, and P. Sanders. High-quality hypergraph partitioning. *ACM Journal of Experimental Algorithmics*, 27:1–39, 2023.

58 T. Soma and Y. Yoshida. Spectral sparsification of hypergraphs. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 2570–2581, 2019.

59 W. T. Tutte. On the problem of decomposing a graph into n connected factors. *J. London Math. Soc.*, 36:221–230, 1961.

60 N. Veldt, A. R. Benson, and J. Kleinberg. Hypergraph cuts with general splitting functions. *SIAM Review*, 64(3):650–685, 2022.

Exponential Lower Bounds via Exponential Sums

Somnath Bhattacharjee
Chennai Mathematical Institute, India

Markus Bläser
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

Pranjal Dutta
School of Computing, National University of Singapore, Singapore

Saswata Mukherjee
Chennai Mathematical Institute, India

Abstract

Valiant's famous VP vs. VNP conjecture states that the symbolic permanent polynomial does not have polynomial-size algebraic circuits. However, the best upper bound on the size of the circuits computing the permanent is exponential. Informally, VNP is an exponential sum of VP-circuits. In this paper we study whether, in general, exponential sums (of algebraic circuits) *require* exponential-size algebraic circuits. We show that the famous Shub-Smale τ-conjecture indeed implies such an exponential lower bound for an exponential sum. Our main tools come from parameterized complexity. Along the way, we also prove an exponential fpt (fixed-parameter tractable) lower bound for the parameterized algebraic complexity class $\mathsf{VW}^0_{\mathrm{nb}}[\mathsf{P}]$, assuming the same conjecture. $\mathsf{VW}^0_{\mathrm{nb}}[\mathsf{P}]$ can be thought of as the weighted sums of (unbounded-degree) circuits, where only ± 1 constants are *cost-free*. To the best of our knowledge, this is the *first* time the Shub-Smale τ-conjecture has been applied to prove explicit exponential lower bounds.

Furthermore, we prove that when this class is fpt, then a variant of the counting hierarchy, namely the *linear counting hierarchy* collapses. Moreover, if a certain type of parameterized exponential sums is fpt, then integers, as well as polynomials with coefficients being *definable* in the linear counting hierarchy have subpolynomial τ-complexity.

Finally, we characterize a related class $\mathsf{VW}[\mathsf{F}]$, in terms of permanents, where we consider an exponential sum of algebraic formulas instead of circuits. We show that when we sum over cycle covers that have one long cycle and all other cycles have constant length, then the resulting family of polynomials is *complete* for $\mathsf{VW}[\mathsf{F}]$ on certain types of graphs.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Algebraic complexity, parameterized complexity, exponential sums, counting hierarchy, tau conjecture

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.24

Category Track A: Algorithms, Complexity and Games

Funding *Pranjal Dutta*: Funded by the project "Foundation of Lattice-based Cryptography", by NUS-NCS Joint Laboratory for Cyber Security.

1 Introduction

Valiant [23] proposed an algebraic version of the P versus NP question and defined the class VP, the algebraic analogue of P, which contains polynomial families computable by polynomial sized algebraic circuits. An *algebraic circuit* (or, arithmetic circuit) C is a directed acyclic graph such that (1) every node has either in-degree (*fan-in*) 0 (the *input gates*) or 2 (the *computational gates*), (2) every input gate is labeled by elements from a field $\mathbb{K}$ or variables from $\mathbf{X} = \{X_1, \cdots, X_n\}$, (3) every computational gate is labeled by either $+$

(addition gate) or $\times$ (multiplication gate), with the obvious syntactic meaning, and (4) there is a unique gate of out-degree 0, the *output gate*. Clearly, every gate in a circuit computes a polynomial in $\mathbb{K}[\mathbf{X}]$. We say that the circuit C computes $P(\mathbf{X}) \in \mathbb{K}[\mathbf{X}]$ if the output gate of C computes $P(\mathbf{X})$. The *size* of C, denoted by $\text{size}(C)$, is the number of nodes in the circuit. An algebraic circuit is an *algebraic formula* if every gate in the circuit has out-degree 1 except for the output gate. The class VNP, the algebraic analogue of NP, is definable by taking *exponential sums* of the form

$$f(\mathbf{X}) = \sum_{e \in \{0,1\}^\ell} g(\mathbf{X}, e) \,, \tag{1}$$

where g is computable by a polynomial-size circuit and ℓ is polynomial in the number of variables. It is known that one can also replace algebraic circuits by algebraic formulas, and still get the same class VNP [23, 18]. Valiant further proved that the permanent family is complete for VNP (over fields of characteristic not two). Recall that the permanent of a matrix $(X_{i,j})$ is defined as

$$\operatorname{per} \mathbf{X} = \sum_{\pi \in S_n} X_{1,\pi(1)} \cdots X_{n,\pi(n)}. \tag{2}$$

The famous Valiant's conjecture $\mathsf{VP} \neq \mathsf{VNP}$ is equivalent to the fact that the permanent does not have polynomial-size circuits. The representation of the permanent in (2), although it looks very natural, is not *optimal*. Ryser's formula [19] yields an algebraic formula of size $O(2^n n^2)$. A formula of similar size was later found by Glynn [11]. Ryser's formula is now over sixty years old and has not been improved since. This gives rise to the interesting question whether there is a formula or circuit of subexponential-size (in n) for the permanent? More generally, we can now ask the following question.

▶ **Question 1.** *Is an exponential sum f (as in Eq. (1)) always computable by an algebraic circuit or formulas of size subexponential in ℓ, that is, size $2^{o(\ell)}$? Or are there instances for which exponential-size is necessary?*

Note that exponential-size being necessary is a much *stronger* claim than $\mathsf{VP} \neq \mathsf{VNP}$. It could well be that $\mathsf{VP} \neq \mathsf{VNP}$ but still exponential sums like in (1) have subxponential size circuits! In this paper, we shed some light on the question what happens if exponential sums would always have *subexponential* size circuits.

Question 1 works as driving force between the famous Shub-Smale τ-conjecture [20] and *exponential* lower bounds on exponential sums. The τ-complexity $\tau(f)$ of an integer polynomial is the size of a smallest division-free circuit that computes f starting from the constants ± 1. The *τ-conjecture* states the the number of integer zeroes of f is polynomially bounded in $\tau(f)$, see [20]. [20] shows that the τ-conjecture implies $\mathsf{P}_{\mathbb{C}} \neq \mathsf{NP}_{\mathbb{C}}$, in the Blum–Shub–Smale (BSS) model of computation over the complex numbers [5, 4].

Super-polynomial lower bounds assuming the τ-conjecture. Bürgisser [6] connected the τ-complexity of the permanent to various other conjectures. He showed that the τ-conjecture implies a *superpolynomial* lower bound on $\tau(\text{per}_n)$, implying the constant-free version of $\mathsf{VP} \neq \mathsf{VNP}$, namely $\mathsf{VP}^0 \neq \mathsf{VNP}^0$; for definitions, see Section 2.1. The proof strategy of [6] is as follows: assume $\tau(\text{per}_n) = \mathsf{poly}(n)$, and conclude a complexity-theoretic "collapse" that the counting hierarchy CH (for a definition, see Section 3) is in $\mathsf{P/poly}$. Consider the Pochhammer–Wilkinson polynomial $f_n(x) := \prod_{i=1}^{n}(x-i)$, and construct a unique $O(\log n)$-variate multilinear polynomial B_n such that under a "suitable" substitution, one gets back f_n.

The coefficients of f_n as well as B_n, are efficiently computable (since $\mathsf{CH} \subseteq \mathsf{P/poly}$), implying $B_n \in \mathsf{VNP}^0$. An inspection of Valiant's completeness result reveals that if $B_n \in \mathsf{VNP}^0$, then there is a polynomially bounded sequence $p(n)$ such that $\tau(2^{p(n)}B_n) = \mathsf{poly}(\log n)$, which implies $\tau(2^{p(n)}f_n) = \mathsf{poly}(\log n)$, contradicting the τ-conjecture.

In [6], the superpolynomial lower bound on $\tau(\mathrm{per}_n)$ was also implied by any of the quantities $\tau(n!)$, $\tau(\sum_{k=0}^{n} \frac{1}{k!}T^k)$, or $\tau(\sum_{k=0}^{n} k^r T^k)$ (for any fixed negative integer r) not being poly-logarithmically bounded as a function of n. Here, we remark that the separation proof of VP^0 and VNP^0, even assuming *strong* bounds on the τ-conjecture, is merely *superpolynomial*: we *do not* get the (possibly) desirable exponential separation between VP^0 and VNP^0. This leads to the following question.

▶ **Question 2.** *Does the τ-conjecture imply exponential algebraic lower bounds?*

Here, we mention that there are variants of the τ-conjecture, e.g., the *real τ-conjecture* [15, 21] or *SOS-τ-conjecture* [8], which also give strong algebraic lower bounds. There is also super polynomial lower bound known from a proof complexity theoretic view due to [1] from the original Shub-Smale τ-conjecture. However, the Shub-Smale τ-conjecture is *not known* to give an exponential lower bound for the permanent.

1.1 Our results

The results of our paper revolve around answering both Question 1-2 positively. The main result is the following.

▶ **Theorem 1** (Informal). *The τ-conjecture implies an exponential lower bound for some explicit exponential sum.*

Remarks.

(1) Although the existence of *some* polynomial requiring exponential circuits is clear from dimension/counting, the existence of an (even non-explicit) exponential sum polynomial requiring exponential-size circuits is *unclear*. Explicit here means that the family is in VNP.

(2) One can also think of an exponential sum f in Equation (1), as $f = \sum_{e\in\{0,1\}^{\ell(n)}} U(\mathbf{X}, y, e)$, where $U(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ is a *universal circuit* of size $\mathrm{size}(U) = \mathsf{poly}(\mathrm{size}(g))$ with $\mathbf{Y} = (Y_1, \ldots, Y_r)$ and $\mathbf{Z} = (Z_1, \ldots, Z_{\ell(n)})$ and $y \in \mathbb{F}^r$ is chosen such that $U(\mathbf{X}, y, e) = g(\mathbf{X}, e)$; and the number of variables $\ell(n)$ is linear in n.

(3) Since there's a polynomial (non-linear) blowup in the reduction of the exponential sum on the universal circuit from the permanent, we will only get a subexponential lower bound on the permanent polynomial assuming the τ-conjecture. We leave it as an open question to achieve an exponential lower bound on the permanent assuming the τ-conjecture.

The proof of Theorem 1 is rather indirect, and goes via *exponential sums*, which is our main object of study (and bridge between many results and classes).

log-variate exponential sum polynomial. Let $g(\mathbf{X}, \mathbf{Y})$ be some polynomial in n-many $\mathbf{X}$-variables and $\ell(n)$-many $\mathbf{Y}$-variables, where $\ell(n) = O(n)$. Assume that g is computed by a circuit of size m. Then we define

$$\mathsf{p\text{-}log\text{-}Expsum}_{m,k}(g) := \sum_{y\in\{0,1\}^{\ell(n)}} g(\mathbf{X}, y) ,$$

where $k = n/\log m$. The size of the exponential is measured in the number $\ell(n)$ of $\mathbf{Y}$-variables. In the end, we want to measure in the input size, the number n of $\mathbf{X}$-variables. To talk about subexponential complexity, $\ell(n)$ should be linearly bounded. g will be typically computed by a circuit (of unbounded degree). We want to view $\mathsf{p\text{-}log\text{-}Expsum}_{m,k}$ as a parameterized problem, the parameter will be k. Our definition of p-log-Expsum, as a polynomial-sum, is motivated by the *log-parameterizations* which are used in the definition of the so-called M-hierarchy in the Boolean setting, see [9, 10].

We show that p-log-Expsum is most likely *not* fixed-parameter tractable (fpt). A polynomial family $p_{n,k}$ is fpt if both its size and degree are fpt bounded, i.e., of the form $f(k)q(n)$, for $q \leq \mathsf{poly}(n)$, and $f : \mathbb{N} \to \mathbb{N}$ being *any* computable function. We connect p-log-Expsum with – (1) a linear variant of the counting hierarchy (we denote it by $\mathsf{CH}_{\mathrm{lin}}$), where the size of the oracle calls are bounded linearly in the size of the input; for definition see Section 3; and (2) integers definable in $\mathsf{CH}_{\mathrm{lin}}$, similar to Bürgisser [6]. Informally, an integer is definable in $\mathsf{CH}_{\mathrm{lin}}$, if its sign and bits are computable in the same class.

▶ **Theorem 2** (Informal). *If* p-log-Expsum *is fixed-parameter tractable, then the following results hold.*

1. *The linear counting hierarchy (*$\mathsf{CH}_{\mathrm{lin}}$*) collapses.*
2. *Any sequence $a(n)$ definable in the linear counting hierarchy, as well as univariate polynomials with coefficients being definable in the linear counting hierarchy, have subpolynomial τ-complexity.*

For formal statements, see Theorem 13 and 21.

Finally, many algebraic complexity classes can be defined in terms of permanents. Most prominently, the "regular" permanent family (per_n) is complete for VNP. The class VW[1] is an important class in parameterized complexity. It is defined as a bounded sum over constant depth weft-1 circuits. Bounded sum means that we sum over $\{0,1\}$-vectors with k ones and k is the parameter. Bläser and Engels [3] prove that VW[1] is described by so-called k-permanents with k being the parameter. In a k-permanent, we only sum over permutations with $n-k$ self-loops. The crucial parameterized class of this work is VW[P]: it is defined as a bounded exponential sum over polynomially-sized arithmetic circuits computing a polynomial of degree that is polynomially bounded. While we do not characterise VW[P] in terms of permanents, we characterize the related class VW[F]: Here instead of summing over circuits, we sum over *formulas*.[1] The permutations that we sum over for defining our permanent family will have one cycle of length k and all other cycles bounded by 4. Again, k is the parameter. We call the corresponding polynomials $(k,4)$-restricted permanents. It turns out that we also need to restrict the graph classes. We call a graph $G = (V,E)$ $(4,b)$-nice if we can partition the set $V = V_1 \cup V_2$ disjointly, such that in the induced graph $G[V_1]$, every cycle is either a self-loop or has length > 4 and in the induced graph $G[V_2]$ has tree-width bounded by b. While this looks artificial at a first glance, it turns out that there is a constant b such that $(k,4)$-restricted permanent on $(4,b)$-nice graphs describes the natural class VW[F]. There is a family of $(4,b)$-nice graphs such that the corresponding family of $(k,4)$-restricted permanents is VW[F]-hard. On the other hand, the $(k,4)$-restricted permanent family is in VW[F] for every family of $(4,b)$-nice graphs. Together, this implies:

▶ **Theorem 3** (VW[F]-Completeness). *$(k,4)$-restricted permanent family on $(4,b)$-nice graphs is* VW[F]*-complete.*

[1] Maybe an explanation of the naming convention is helpful: In VW[P], we sum of polynomial-size circuits, which describe the class VP. In VW[F], we sum over polynomial size formulas, which define the class VF, the modern name for VP_e.

We also prove strong separations of algebraic complexity classes and parameterized algebraic complexity classes (Theorem 30), and exponential lower bounds in the parameterized setting (Theorem 36).
For VNP it is known that it does not matter whether we sum over formulas or circuits, that is, $\mathsf{VNP} = \mathsf{VNP}_e$. Whether $\mathsf{VW[P]} = \mathsf{VW[F]}$ remains an open questions for future research.

1.2 Proof ideas

In this section, we briefly sketch the proof ideas. The omitted proofs of the paper can be found in the longer arxiv version of the paper. We first present the proofs of Theorem 2, because the techniques and lemmas involved in proving them are the backbone of Theorem 1.

Proof idea of Theorem 2. We prove them in two parts.
Proof of Part (1): We prove even a stronger statement for the subexponential version of the linear counting hierarchy. The proof goes via induction on the level of the counting hierarchy. The criteria for some language B being in the $(k+1)$-th level is that there should be some language A in the k-th level such that $|\{y \in \{0,1\}^n : \langle x, y\rangle \in A\}| > 2^{n-1}$. Essentially, for a language A in the k-th level, we express $|\{y \in \{0,1\}^n : \langle x, y\rangle \in A\}| > 2^{n-1}$ as an exponential sum over an algebraic circuit $\chi_A(x, y)$, which captures the characteristic function of A. Furthermore, one can show that p-log-Expsum is fpt (in an unbounded constant-free setting) iff $\sum_y g(\mathbf{X}, y)$ has $2^{o(n)}\mathsf{poly}(m)$ size circuits, where g has a circuit of size m; see Theorem 15 and 16. Putting these together, one gets that the exponential sum has a subexponential-size constant-free circuit. Lastly, we want to get the information about the highest bit of the sum (which is equivalent to looking at it mod 2^n), which can be efficiently *arithmetized*. In every step there is polynomial blowup in the size, and hence the size remains subexponential, yielding the desired result. For details, see Theorem 13.
Proof of Part (2): This proof is an adaption of [6, 14] in our context. Take a sequence $(a_n)_n \in \mathsf{CH}_{\mathsf{lin}}\mathsf{P}$. We define a multilinear polynomial $A(\mathbf{Y})$ such that the coefficient of $\mathbf{Y}^{\mathbf{j}}$ is the j-th bit of $a(n)$, where $\mathbf{j}$ is the binary representation of j. Furthermore, checking $a(n, j) = b$ can be done by a subexponential circuit $C(\mathbf{N}, \mathbf{J})$, where $\mathbf{N}$ and $\mathbf{J}$ have $\log n$ and $\mathsf{bit}(n)$-many variables capturing n and j respectively. Moreover, one can define $F(\mathbf{N}, \mathbf{Y}, \mathbf{J}) = C(\mathbf{N}, \mathbf{J}) \cdot \prod_i (J_i Y_i + 1 - J_i)$ and show that A can be expressed as an exponential sum over $F(j, \mathbf{N}, \mathbf{Y})$! This is clearly a p-log-Expsum instance, which finally yields that the τ-complexity of $a(n)$ is subpolynomial. A similar proof strategy also holds for the polynomials with coefficients being definable in $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$. For details, see Section 6.

Proof idea of Theorem 1. Take the Pochhammer polynomial $p_n(X) = \prod_{i=1}^{n}(X + i)$. The coefficient of X^{n-k} in p_n will be $\sigma_k(1, \ldots, n)$, where $\sigma_k(z_1, \ldots, z_n)$ is the k-th elementary symmetric polynomial in variables $z_1, \ldots, z_n$. It is not hard to show that $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$ is closed under polynomially-many additions and multiplications (Theorem 19). Therefore, $(\sigma_k(1, \ldots, n))_{n\in\mathbb{N}, k\le n}$ is definable in the linear counting hierarchy (see Corollary 20). And by Theorem 21, $(p_n)_{n\in\mathbb{N}}$ has $n^{o(1)}$-sized constant-free circuits if p-log-Expsum is fixed-parameter tractable. But p_n has n distinct integer roots. Assuming the τ-conjecture, p-log-Expsum is not *fpt*. On the other hand, one can show that when exponential sums over circuits of size m have circuits have size $2^{o(n)}\mathsf{poly}(m)$, then the p-log-Expsum is fpt, by Theorem 16; in other words, p-log-Expsum is not fpt implies an exponential lower bound on an exponential sum. This finishes the proof.

Proof idea of Theorem 3. The hardness proof is *gadget* based (Theorem 42). The details are however quite complicated since we have to cleverly keep track of the cycle lengths. For the upper bound, we work along a tree decomposition. While it is known that the permanent can be computed in fpt time on graphs of bounded treewidth, we cannot simply adapt these algorithms, since we have to produce a formula. This can be achieved using a *balanced tree decomposition*.

1.3 Previous results

To prove (conditional) exponential lower bounds, the standard assumptions that $\mathsf{P} \neq \mathsf{NP}$ or $\mathsf{VP} \neq \mathsf{VNP}$ are not enough. It is consistent with our current knowledge that for instance $\mathsf{P} \neq \mathsf{NP}$, but NP-hard problems can have subexponential time algorithms. What we need is a complexity assumption stating that certain problems can only be solved in exponential time. This is the exponential time hypothesis (ETH) in the Boolean setting. Dell et al. [7] studied the exponential time complexity of the permanent, they prove that when there is an algorithm for computing the permanent in time $2^{o(n)}$, then this violates the counting version of the exponential time hypothesis #ETH. #ETH states that there is a constant c such that no deterministic algorithm can count the number of satisfying assignments of a formula in 3-CNF in time 2^{cn}. For connections between parameterized and subexponential complexity in the Boolean setting, we refer to [9, 10].

Bläser and Engels [3] transfer the important definitions and results from parameterized complexity in the Boolean world to define a theory of parameterized algebraic complexity classes. In particular, they define the VW-hierarchy and prove that the clique polynomial and the k-permanent are $\mathsf{VW}[1]$-complete (under so-called fpt-substitutions). They also claim the hardness of the restricted permanent for the class $\mathsf{VW}[t]$ for every constant t and sketch a proof. Note that $\mathsf{VW}[\mathsf{F}]$ contains each $\mathsf{VW}[t]$. So we strengthen the hardness proof in [3] and complement it with an upper bound.

The main tool used by Bürgisser [6] to prove the results above is the counting hierarchy. The polynomial counting hierarchy was introduced by Wagner [24] to classify the complexity of Boolean counting problems. The fact that small circuits for the permanent collapses the counting hierarchy is used by Bürgisser to prove the results mentioned above.

Finally, there have been quite a few works [6, 14, 16, 15], where we have conditional separations on the constant-free version of VP and VNP, namely VP^0 and VNP^0, or their variants, depending on the strength of the conjecture. But this is the first time that we are separating algebraic classes and proving exponential lower bounds, assuming the τ-conjecture.

1.4 Structure of the paper

In Section 2, we defined the basics of constant-free Valiant's model and the unbounded and parameterized setting. In Section 3, we introduce the linear counting hierarchy ($\mathsf{CH}_{\mathrm{lin}}$) and its basic properties. Section 4 connects Valiant's model to the counting hierarchy. Here, we formally introduce exponential sums and investigate their relation to the parameterized classes. The main result is that the fixed-parameter tractability of exponential sums collapses the counting hierarchy. The proofs are quite similar to [6], however, we need to pay special attention to the fact the witness size is linear. Section 5 introduces the definability (computability) of integers in the linear counting hierarchy, and some closure properties of the same. Section 6 proves the exponential lower bound on exponential sum assuming τ-conjecture. Section 7 introduces the parameterized VW-classes and its basic properties. In Section 8 we prove some easy conditional collapse results of the VW-hierarchy in various circuit models.

2 Preliminaries I

2.1 Constant-free and unbounded models

Constant-free Valiant's classes. We will say that an algebraic circuit is *constant-free*, if no field elements other than $\{-1, 0, 1\}$ are used for labeling in the circuit. Clearly, constant-free circuits can *only* compute polynomials in $\mathbb{Z}[\mathbf{X}]$. For $f(X) \in \mathbb{Z}[\mathbf{X}]$, $\tau(f)$ is the size of a minimum size constant-free circuit that computes f, while $L(f)$ denotes the minimum size circuit that computes f. It is noteworthy to observe that, *unlike* Valiant's classical models, computing integers in the constant-free model can be costly; e.g., $\tau(2^{2^n}X^n) = \Omega(n)$, while $L(2^{2^n}X^n) = \Theta(\log n)$. On the other hand, for any $f \in \mathbb{Z}[\mathbf{X}]$, $L(f) \leq \tau(f)$.

Before defining the constant-free Valiant classes, we formalize the notion of *formal degree* of a node, denoted formal-deg$(\cdot)$. It is defined recursively as follows: (1) the formal degree of an input gate is 1, (2) if $u = v + w$, then formal-deg$(u) = \max($formal-deg$(v),$ formal-deg$(w))$, and (3) if $u = v \times w$, then formal-deg$(u) =$ formal-deg$(v) +$ formal-deg(w). The formal degree of a circuit is defined as the formal degree of its output node.

The class *constant-free Valiant's* P, denoted by VP^0, contains all p-$\mathsf{families}$ (f) in $\mathbb{Z}[\mathbf{X}]$, such that formal-deg(f) and $\tau(f)$ are both p-$\mathsf{bounded}$. Analogously, VNP^0 contains all p-$\mathsf{families}$ (f_n), such that there exists a p-$\mathsf{bounded}$ function $q(n)$ and $(g_n) \in \mathsf{VP}^0$, where

$$f_n(\mathbf{X}) = \sum_{\overline{y} \in \{0,1\}^{q(n)}} g_n(\mathbf{X}, y_1, \ldots, y_{q(n)}) .$$

It is not clear whether showing $\mathsf{VP}^0 \neq \mathsf{VNP}^0$ implies $\mathsf{VP} \neq \mathsf{VNP}$, it is *not even clear* whether $\mathsf{VP}^0 \neq \mathsf{VNP}^0 \implies \tau(\mathrm{per}_n) = n^{\omega(1)}$. The *subtlety* here is that in the algebraic completeness proof for the permanent, *divisions by two* occur! However, a partial implication is known due to [6, Theorem 2.10]: Showing $\tau(2^{p(n)}f_n) = n^{\omega(1)}$, for some $f_n \in \mathsf{VNP}^0$ and all p-$\mathsf{bounded}$ $p(n)$ would imply that $\tau(\mathrm{per}_n) = n^{\omega(1)}$.

Arithmetization is a well-known technique in complexity theory. To arithmetize a Boolean circuit C computing a Boolean function φ, we use the arithmetization technique wherein we map $\varphi(x_1, \ldots, x_n)$ to a polynomial $p(x_1, \ldots, x_n)$ such that for any assignment of Boolean values $v_i \in \{0, 1\}$ to the x_i, $\varphi(v_1, \ldots, v_n) = p(v_1, \ldots, v_n)$ holds.

We define the arithmetization map Γ for variables x_i, and clauses $c_1, \ldots, c_m$, as follows:

1. $x_i \mapsto x_i$,
2. $\neg x_i \mapsto 1 - x_i$,
3. $c_1 \vee \cdots \vee c_m \mapsto 1 - \prod_{i \in [m]}(1 - \Gamma(c_i))$,
4. $c_1 \wedge \cdots \wedge c_m \mapsto \prod_{i \in [m]} \Gamma(c_i)$.

This map allows us to transform C into an arithmetic circuit for p. For a Boolean circuit C, we denote the arithmetized circuit by $\mathsf{arithmetize}(C)$. Here, we remark that the degree of $\mathsf{arithmetize}(C)$ can become *exponentially* large; this is because there is no known depth-reduction for Boolean circuits, and hence the degree may double at each step, owing to an exponential blowup in the degree.

Valiant's classes in the unbounded setting. It is well-known that an algebraic circuit of size s, can compute polynomials of degree $\exp(s)$; e.g., $f(x) = x^{2^s}$, and $L(f) = O(s)$. This brings us to the next definition, the class $\mathsf{VP}_{\mathrm{nb}}$, originally defined in [17]. A sequence of polynomials $(f) = (f_n)_n \in \mathsf{VP}_{\mathrm{nb}}$, if the number of variables in f_n and $L(f_n)$ are both p-$\mathsf{bounded}$ (the degree *may be* exponentially large). The subscript "nb" signifies the *"not bounded"* phenomenon on the degree of the polynomial, in contrast to the original class VP. Similarly, a sequence of polynomials $(f) = (f_n)_n \in \mathsf{VNP}_{\mathrm{nb}}$, if there exists a p-$\mathsf{bounded}$ function $q(n)$ and $g_n(\mathbf{X}, Y_1, \ldots, Y_{q(n)}) \in \mathsf{VP}_{\mathrm{nb}}$ where

$$f_n(\mathbf{X}) \;=\; \sum_{\overline{y}\in\{0,1\}^{q(n)}} g_n(\mathbf{X}, y_1, \ldots, y_{q(n)}) \;.$$

One can analogously define $\mathsf{VP}^0_{\mathrm{nb}}$ and $\mathsf{VNP}^0_{\mathrm{nb}}$, in the constant-free setting. It is obvious that $\mathsf{VP}_{\mathrm{nb}} = \mathsf{VNP}_{\mathrm{nb}}$ implies $\mathsf{VP} = \mathsf{VNP}$, but the converse is *unclear*. However, [17] showed that over a ring of positive characteristic, the converse holds, i.e., $\mathsf{VP} = \mathsf{VNP}$ implies $\mathsf{VP}_{\mathrm{nb}} = \mathsf{VNP}_{\mathrm{nb}}$! On the other hand, [16] showed that $\mathsf{VP}^0 = \mathsf{VNP}^0$ implies that $\mathsf{VP}^0_{\mathrm{nb}} = \mathsf{VNP}^0_{\mathrm{nb}}$, and the converse is unclear because it seems difficult to rule out the possibility that some polynomial family in VNP^0 does not lie in VP^0, but still in VP (i.e., computable by polynomial-size algebraic circuits using *exponentially large-bit* integers).

2.2 Parameterized Valiant's classes

Parameterized Valiant's classes were introduced in [3]. We will briefly review the definitions and results there and extend them to the constant-free and unbounded setting. We first start with the fixed-parameter tractable classes. The W-hierarchies will be introduced later since we only need them in the second part of this work.

Our families of polynomials will now have two indices. They will be of the form $(p_{n,k})$. Here, n is the index of the family and k is the parameter. We will say a polynomial family $(p_{n,k})$ is a *parameterized* p-family if the number of variables is p-bounded in n and the degree is p-bounded in n, k. If there is no bound on the degree, we say it is *parameterized* family.

The most natural parameterization is by the degree: Let (p_n) be any p-family then we get a parameterized family $(p_{n,k})$ by setting $p_{n,k} :=$ the homogeneous part of degree k of p_n. For more details, we will refer the reader to [3].

We now define fixed-parameter variants of Valiant's classes with the constant-free version.

▶ **Definition 4** (Algebraic FPT classes)**.**

1. *A parameterized* p-family $(p_{n,k})$ *is in* VFPT *iff* $L(p_{n,k})$ *is upper bounded by* $f(k)q(n)$ *for some* p-bounded *function* q *and some function* $f : \mathbb{N} \to \mathbb{N}$ *(such bound will be called an fpt bound). If one removes the requirement of* p-family *on* $p_{n,k}$*, and imposes only that the number of variables is* p-bounded*, one gets the class* $\mathsf{VFPT}_{\mathrm{nb}}$*.*
2. *A parameterized* p-family $p_{n,k}$ *is in* VFPT^0 *iff* $\tau(p_{n,k})$ *is upper bounded by* $f(k)q(n)$ *for some* p-bounded *function* q *and some function* $f : \mathbb{N} \to \mathbb{N}$*. Similarly, one gets* $\mathsf{VFPT}^0_{\mathrm{nb}}$*, if one removes the requirement of* p-family*, and imposes only that the number of variables is* p-bounded*.*

We remark that in the above, f need not be computable as Valiant's model is non-uniform.

▶ **Definition 5** (Fpt-projection)**.** *A parameterized family* $f = (f_{n,k})$ *is an fpt-projection of another parameterized family* $g = (g_{n,k})$ *if there are functions* $r, s, t : \mathbb{N} \to \mathbb{N}$ *such that* r *is* p-bounded*,* s, t *are functions and* $f_{n,k}$ *is a projection of* $g_{r(n)s(k),k'}$ *for some* $k' \leq t(k)$*,*[2]*. We write* $f \leq^{\mathrm{fpt}}_p g$*.*

However p-projection in Valiant's world seems to be *weaker* compared to parsimonious poly-time reduction in the Boolean world; therefore we need a stronger notion of reduction for defining algebraic models of the Boolean $\#W$-classes, see [3]. That's why we are defining substitutions. We will analogously define it for constant-free model as well.

[2] k' might depend on n, but its size is bounded by a function in k. There are examples in the Boolean world, where this dependence on n is used.

▶ **Definition 6** (Fpt-substitution).

1. *A parameterized family $f = (f_{n,k})$ is an fpt-substitution of another parameterized family $g = (g_{n,k})$ if there are functions $r, s, t, u : \mathbb{N} \to \mathbb{N}$ and polynomials $h_1, \ldots, h_{u(r(n)s(k))} \in \mathbb{K}[\mathbf{X}]$ with both $L(h_i)$ and $deg(h_i)$ fpt-bounded such that r, u are p-bounded, s, t are functions, and $f_{n,k}(\mathbf{X}) = g_{r(n)s(k),k'}(h_1, \ldots, h_{u(r(n)s(k))})$ for some $k' \leq t(k)$. We write $f \leq_s^{\text{fpt}} g$. When we allow* **unbounded** *degree substitution of h_i (i.e. only $L(h_i)$ is fpt-bounded), we say that f is an fpt_{nb}-substitution of g. We denote this as $f \leq_s^{\text{fpt}_{\text{nb}}} g$.*
2. *A parameterized family $f = (f_{n,k})$ is a constant-free fpt-substitution of another parameterized family $g = (g_{n,k})$ if there are functions $r, s, t, u : \mathbb{N} \to \mathbb{N}$ and polynomials $h_1, \ldots, h_{u(r(n)s(k))} \in \mathbb{K}[\mathbf{X}]$ with both $\tau(h_i)$ and $deg(h_i)$ are fpt-bounded such that r, u are p-bounded, s, t are functions and $f_{n,k}(\mathbf{X}) = g_{r(n)s(k),k'}(h_1, \ldots, h_{u(r(n)s(k))})$ for some $k' \leq t(k)$. We write $f \leq_s^{\tau\text{-fpt}} g$. If we remove the degree condition, we get fpt_{nb}-substitutions, denoted as $f \leq_s^{\tau\text{-fpt}_{\text{nb}}} g$.*

One can define constant-free fpt-projections analogously. The following lemma should be immediate from the definitions, see [3] for a proof in the case of VFPT.

▶ **Lemma 7.** $\mathsf{VFPT}, \mathsf{VFPT}_{\text{nb}}$ *and their constant-free versions (*VFPT^0, $\mathsf{VFPT}^0_{\text{nb}}$*) are* closed *under fpt-projections and fpt-substitutions (constant-free fpt-projections and constant-free fpt-substitutions, respectively).*

3 Linear counting hierarchy

In this section, we define the linear counting hierarchy, a variant of the counting hierarchy, which will allow us to talk about subexponential complexity. The original counting hierarchy was defined by Wagner [24]. We here restrict the witness length to be linear, which is important when dealing with exponential complexity. Allender et al. [2] also define a linear counting hierarchy. Their definition is not comparable to ours. We use an operator-based definition: The base class is deterministic polynomial time and the witness length is linearly bounded. Allender et al. use an oracle TM definition: The oracle Turing machine is probabilistic and linear time bounded, which automatically bounds the query lengths.

▶ **Definition 8.** *Given a complexity class K, we define $\mathbf{C}.K$ to be the class of all languages A such that there is some $B \in K$ and a function $p : \mathbb{N} \to \mathbb{N}$, $p(n) = O(n^c)$ for some constant c, and some polynomial time computable function $f : \{0,1\}^* \to \mathbb{N}$ such that,*

$$x \in A \iff |\{y \in \{0,1\}^{p(|x|)} : \langle x, y\rangle \in B\}| > f(x).$$

We start from $\mathsf{C}_0\mathsf{P} := \mathsf{P}$ and for all $k \in \mathbb{N}$, $\mathsf{C}_{k+1}\mathsf{P} := \mathbf{C}.\mathsf{C}_k\mathsf{P}$. Then the *counting hierarchy* is defined as $\mathsf{CH} := \bigcup_{k \geq 0} \mathsf{C}_k\mathsf{P}$. We now define our linear counting hierarchy:

▶ **Definition 9.** *Given a complexity class K, we define $\mathbf{C}_{\mathsf{lin}}.K$ to be the class of all languages A such that there is some $B \in K$ and a function $\ell : \mathbb{N} \to \mathbb{N}$, $\ell(n) = O(n)$, and some polynomial time computable function $f : \{0,1\}^* \to \mathbb{N}$ such that,*

$$x \in A \iff |\{y \in \{0,1\}^{\ell(|x|)} : \langle x, y\rangle \in B\}| > f(x).$$

We define $\mathsf{C\text{-}lin}_0\mathsf{P} := \mathsf{P}$ and for all $k \in \mathbb{N}$, $\mathsf{C\text{-}lin}_{k+1}\mathsf{P} := \mathbf{C}_{\mathsf{lin}}.\mathsf{C\text{-}lin}_k\mathsf{P}$. The *linear counting hierarchy* is $\mathsf{CH}_{\mathsf{lin}}\mathsf{P} := \bigcup_{k \geq 0} \mathsf{C\text{-}lin}_k\mathsf{P}$.

Now, we slightly modify the above definition to get $\exists_{\mathsf{lin}}.K$ and $\forall_{\mathsf{lin}}.K$ in the following way: $x \in A \iff \exists y \in \{0,1\}^{\ell(|x|)} : \langle x, y\rangle \in B$ and $x \in A \iff \forall y \in \{0,1\}^{\ell(|x|)} : \langle x, y\rangle \in B$, respectively. Clearly, it can be said that $K \subseteq \exists_{\mathsf{lin}}.K \subseteq \mathbf{C}_{\mathsf{lin}}.K$ and $K \subseteq \forall_{\mathsf{lin}}.K \subseteq \mathbf{C}_{\mathsf{lin}}.K$.

We can define the linear counting hierarchy in a slightly easier manner.

▶ **Definition 10.** *Given a complexity class K, we define $\mathbf{C}'_{\mathsf{lin}}.K$ to be the class of all languages A such that there is some $B \in K$ and a function $\ell : \mathbb{N} \to \mathbb{N}$, $\ell(n) = O(n)$, such that*

$$x \in A \iff |\{y \in \{0,1\}^{\ell(|x|)} : \langle x, y\rangle \in B\}| > 2^{\ell(|x|)-1}.$$

It is clear that $\mathbf{C}'_{\mathsf{lin}}.K \subseteq \mathbf{C}_{\mathsf{lin}}.K$ for any class K. Moreover, by an easy adaption of the proof of [22, Lemma 3.3], for any language $K \in \mathsf{CH}$, $\mathbf{C}_{\mathsf{lin}}.K \subseteq \mathbf{C}'_{\mathsf{lin}}.K$. Also, from the definition, we can say that $\mathsf{CH}_{\mathsf{lin}}\mathsf{P} \subseteq \mathsf{CH}$. Therefore, the following holds.

▶ **Fact 11.** $\mathsf{C\text{-}lin}_{k+1}\mathsf{P} = \mathbf{C}'_{\mathsf{lin}}.\mathsf{C\text{-}lin}_k\mathsf{P}$.

We also need a subexponential version of the counting hierarchy. Let $\mathsf{SUBEXP} = \mathsf{DTime}(2^{o(n)})$. Then we set $\mathsf{C\text{-}lin}_0\mathsf{SUBEXP} = \mathsf{SUBEXP}$ and for all $k \in \mathbb{N}$, $\mathsf{C\text{-}lin}_{k+1}\mathsf{SUBEXP} := \mathbf{C}_{\mathsf{lin}}.\mathsf{C\text{-}lin}_k\mathsf{SUBEXP}$. Moreover, $\mathsf{CH}_{\mathsf{lin}}\mathsf{SUBEXP} = \bigcup_{k\geq 0} \mathsf{C\text{-}lin}_k\mathsf{SUBEXP}$.

Here we define a few more terms that we shall use later in Section 5. We set $\mathsf{NP}_{\mathsf{lin}} = \exists_{\mathsf{lin}}.\mathsf{P}$, NP with linear witness size. In the same way, we can define the levels of the linear polynomial time hierarchy, Σ_i^{lin} and Π_i^{lin}, by applying the operators $\exists_{\mathsf{lin}}$ and $\forall_{\mathsf{lin}}$ in an alternating fashion to P. The linear polynomial hierarchy $\mathsf{PH}_{\mathsf{lin}}$ is the union over all Σ_i^{lin}.

From the above definitions, we get the following conclusion.

▶ **Fact 12.** $\mathsf{NP}_{\mathsf{lin}} \subseteq \mathsf{PH}_{\mathsf{lin}} \subseteq \mathsf{CH}_{\mathsf{lin}}$.

4 Connecting Valiant's model to the counting hierarchy

In this section, we aim to prove that subexponential upper bounds for exponential sums imply a collapse of the linear counting hierarchy (for a definition, see Section 3). To show this, we will define a polynomial family p-log-Expsum and show that p-log-Expsum $\in \mathsf{VFPT}^0_{\mathrm{nb}}$ is equivalent to exponential sums having subexponential circuits (Corollary 34). p-log-Expsum $\in \mathsf{VFPT}^0_{\mathrm{nb}}$ will imply a collapse of the linear counting hierarchy (Theorem 13).

4.1 log-variate exponential sum polynomial family

In this section, we will define a parameterized log-variate exponential sum polynomial family,

$$\mathsf{p\text{-}log\text{-}Expsum}_{m,k}(g) := \sum_{y\in\{0,1\}^{\ell(n)}} g_n(\mathbf{X}, y)\ ,$$

where $\mathbf{X}$ has n variables, $\ell(n) = O(n)$, and g_n has circuits of size m ($n = \Omega(\log m)$), and the parameter is $k = \frac{n}{\log m}$. m and k are functions of n. Note that the running parameter of the family is m. When we write p-log-Expsum $\in \mathsf{VFPT}$, we mean that $\{\mathsf{p\text{-}log\text{-}Expsum}_{m,k}(g)\}_{m,k} \in \mathsf{VFPT}$ for all families g. We are allowing g to have *unbounded* degree, i.e., g may not necessarily be a *p*-family. We will also be using constant-free circuits computing g in the constant-free context.

4.2 Collapsing of $\mathsf{CH}_{\mathsf{lin}}\mathsf{SUBEXP}$

The main theorem of the section is the following:

▶ **Theorem 13.** *If* p-log-Expsum $\in \mathsf{VFPT}^0_{\mathrm{nb}}$, *then for every language L in* $\mathsf{CH}_{\mathsf{lin}}\mathsf{SUBEXP}$, *we have a constant-free algebraic circuit χ_L so that $x \in L \implies \chi_L(x) = 1$, $x \notin L \implies \chi_L(x) = 0$ and χ_L has size $2^{o(n)}$.*

Proof. We prove the above statement by induction on the level of $\mathsf{CH}_{\mathsf{lin}}\mathsf{SUBEXP}$. By definition, $\mathsf{CH}_{\mathsf{lin}}\mathsf{SUBEXP} = \bigcup_{k\geq 0} \mathsf{C\text{-}lin}_k\mathsf{SUBEXP}$. For $k = 0$, $\mathsf{C\text{-}lin}_k\mathsf{SUBEXP} = \mathsf{SUBEXP}$. Now by standard arithmetization, we can get a $2^{o(n)}$ size, unbounded degree constant-free circuit for each $L \in \mathsf{SUBEXP}$, so that the above-mentioned condition holds.

Now, by induction hypothesis say, it is true up to k-th level of the hierarchy. We will prove that it is true for the $(k+1)$-th level. Take any $B \in \mathsf{C\text{-}lin}_{k+1}\mathsf{SUBEXP}$. By Fact 11 and Definition 10, there exists $A \in \mathsf{C\text{-}lin}_k\mathsf{SUBEXP}$ such that

$$x \in B \iff |\{y \in \{0,1\}^{\ell(|x|)} : \langle x, y\rangle \in A\}| > 2^{\ell(|x|)-1},$$

where ℓ is some linear polynomial. By slight abuse of notation, let χ_A denote an algebraic circuit capturing the characteristic function for A, i.e.,

$$\chi_A(x, y) = 1 \iff \langle x, y\rangle \in A .$$

By the induction hypothesis, we can assume that χ_A has size $2^{o(|x|)}$. Now, one can equivalently write the following:

$$x \in B \iff \sum_{y\in\{0,1\}^{\ell(|x|)}} \chi_A(x, y) > 2^{\ell(|x|)-1} .$$

In this way, we get an instance of $\mathsf{p\text{-}log\text{-}Expsum}$, $\sum_{y\in\{0,1\}^{\ell(|x|)}} \chi_A(x, y)$, where the size of χ_A is $m = 2^{o(|x|)}$ and it computes a polynomial of *unbounded degree* (there is no depth-reduction known for Boolean circuits and thus, it cannot be reduced).

As $\mathsf{p\text{-}log\text{-}Expsum} \in \mathsf{VFPT}^0_{\mathsf{nb}}$, there is an algebraic circuit C such that $C(x) := \sum_{y\in\{0,1\}^{\ell(|x|)}} \chi_A(x, y)$ and C has subexponential-size by Theorem 15.

Trivially, $\tau(2^{\ell(|x|)-1}) \leq \mathsf{poly}(|x|)$. So, we can make C first constant-free and then Boolean by the standard procedure of computing on the binary representation modulo $2^{\ell(n)}$. Let $\tilde{C}$ is the Boolean circuit that computes the highest bit. We just arithmetize $\tilde{C}$ and take $\chi_B = \mathsf{arithmetize}(\tilde{C})$. Each time we convert the arithmetic circuit to a Boolean one and arithmetize the Boolean circuit, we incur only a small polynomial blow-up in size. Therefore, χ_B has subexponential-size, as desired. ◀

▶ Remark 14. Clearly, $\mathsf{CH}_{\mathsf{lin}}\mathsf{P} \subseteq \mathsf{CH}_{\mathsf{lin}}\mathsf{SUBEXP}$ and hence, $\mathsf{p\text{-}log\text{-}Expsum} \in \mathsf{VFPT}^0_{\mathrm{nb}}$ implies that every language in $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$ has subexponential-size constant-free algebraic circuits.

▶ **Theorem 15.** *If* $\mathsf{p\text{-}log\text{-}Expsum} \in \mathsf{VFPT}^0_{\mathrm{nb}}$, *then* $\sum_{y\in\{0,1\}^{\ell(n)}} g(\mathbf{X}, y)$ *has circuits of size* $2^{o(n)}\mathsf{poly}(m)$.

Proof. Assume that $\mathsf{p\text{-}log\text{-}Expsum}$ has circuits of size $f(n/\log m)\mathsf{poly}(m)$. We can assume that f is an increasing function. Let $i(n) = \max(\{1\} \cup \{j \mid f(j) \leq n\})$. $i(n)$ is nondecreasing and unbounded. Moreover, $f(i(n)) \leq n$ for all but finitely many n.

We will prove that $\sum_{y\in\{0,1\}^{\ell(n)}} g(\mathbf{X}, y)$ has circuits of size $2^{n/i(n)}\mathsf{poly}(m)$. If $m \geq 2^{n/i(n)}$, then $f(n/\log m) \leq f(i(n)) \leq n$, thus there are circuits of size $n \cdot \mathsf{poly}(m) = \mathsf{poly}(m)$. If $m < 2^{n/i(n)}$, then let $\hat{m} = 2^{n/i(n)}$. We can take a circuit C for g and pad it to a circuit $\hat{C}$ of size s with $\hat{m} \leq s \leq O(\hat{m})$, such that $\hat{C}$ has the same variables as C. Then let $\hat{k} = n/\log \hat{m}$. Thus, $\sum_{y\in\{0,1\}^{\ell(n)}} g(\mathbf{X}, y)$ has circuits of size $f(\hat{k})\mathsf{poly}(\hat{m}) = n \cdot \mathsf{poly}(2^{n/i(n)})$. ◀

We will need the unbounded version as stated above, but a similar proof also works for the bounded case. The same is true of the non-constant-free version. We will also need the following converse direction:

▶ **Theorem 16.** *Let* $\sum_{y\in\{0,1\}^{\ell(n)}} g(\mathbf{X}, y)$ *have circuits of size* $2^{o(n)}\mathsf{poly}(m)$ *for each* g *of size* m. *Then* p-log-Expsum $\in \mathsf{VFPT}^0_{\mathrm{nb}}$.

Proof. Let C_n be a circuit for $\sum_{y\in\{0,1\}^{\ell(n)}} g(\mathbf{X}, y)$ of size $2^{O(n/i(n))}\mathsf{poly}(m)$ for some non-decreasing and unbounded function i. Let f be a nondecreasing function such that $f(i(n)) \geq 2^n$. We claim that p-log-Expsum has circuits of size $f(k)\mathsf{poly}(m)$ with $k = n/\log m$. If $m \geq 2^{n/i(n)}$, then C_n has size $\mathsf{poly}(m) \leq f(k)\mathsf{poly}(m)$. Otherwise, $k = n/\log m \geq i(n)$ and therefore $f(k) \geq 2^n$. Thus, the trivial circuit for $\sum_{y\in\{0,1\}^{\ell(n)}} g(\mathbf{X}, y)$ has size $f(k)\mathsf{poly}(m)$. ◀

5 Integers definable in $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$

In [6, Section 3], integers are studied that are definable in the counting hierarchy. We adapt this notation to the *linear* counting hierarchy. Formally, we are given a sequence of integers $(a(n,k))_{n\in\mathbb{N},k\leq q(n)}$ for some p-bounded function $q : \mathbb{N} \to \mathbb{N}$. We can assume that $|a(n,k)| \leq 2^{n^c}$ for some constant c. In other words, the bit-size of $a(n,k)$ is at most *exponential*, as we think n, k has been represented in binary by $O(\log n)$ bits. Now consider two languages,

$$\mathsf{sgn}(a) := \{(n,k) : a(n,k) \geq 0\} \text{ and}$$
$$\mathsf{Bit}(|a|) := \{(n,k,j,b) : j\text{th bit of } |a(n,k)| \text{ is } b\} .$$

Here in both of these two languages, n, k, j are given in binary representation.

▶ **Definition 17.** *We say an integer sequence* $(a(n,k))_{n\in\mathbb{N},k\leq q(n)}$ *for some* p-bounded *function* q *is definable in* $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$ *whenever both of* $\mathsf{sgn}(a)$ *and* $\mathsf{Bit}(|a|)$ *are in* $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$.

Chinese remainder language. Now, we define another language and make a connection to the definition of an integer sequence to be definable in $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$, via the *Chinese remainder representation*. Given that the bit-size of $a(n,k)$ is at most n^c, we consider the set of all primes $p < n^{2c}$. The product of all such primes is $> 2^{n^c}$. Therefore, from $a(n,k) \bmod p$, for all primes $p < n^{2c}$, we can recover $a(n,k)$. Consider

$$\mathsf{CR}(a) := \left\{(n,k,p,j,b) : p \text{ prime}, p < n^{2c}, j\text{-th bit of } (a(n,k) \bmod p) \text{ is } b\right\} .$$

Now we show an essential criterion for a sequence to be in $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$. It is an adaption with some additional modifications and observations from [12], which were further implemented in [6, Theorem 3.5].

▶ **Theorem 18.** *Let* $(a(n,k))_{n\in\mathbb{N},k\leq q(n)}$ *be a integer sequence of exponential bit-size* $(|a(n,k)| < 2^{n^c})$. *Then,* $(a(n,k))$ *is definable in* $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$ *iff both* $\mathsf{sgn}(a)$ *and* $\mathsf{CR}(a)$ *are in* $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$.

Now, we can prove an important *closure* property of non-negative integers definable in $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$, which we shall use later.

▶ **Theorem 19** (Closure properties). *Let* $(a(n,k))_{n\in\mathbb{N},k\leq q(n)}$ *be a non-negative integer sequence for some* p-bounded *function* $q : \mathbb{N} \to \mathbb{N}$ *with* $a(n,k)$ *having bit-size* $< n^c$ *and it is definable in* $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$. *Consider the sum and product of* $a(n,k)$ *defined as follows:*

$$b(n) := \sum_{k=0}^{q(n)} a(n,k) \quad \text{and} \quad c(n) := \prod_{k=0}^{q(n)} a(n,k) .$$

Then, both of $(b(n))_{n\in\mathbb{N}}$ *and* $(c(n))_{n\in\mathbb{N}}$ *are definable in* $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$.

▶ **Corollary 20.** *Take* $a(n,k) := \sigma_{n,k}(1,\ldots,n)$, $k \leq n$, *where* $\sigma_{n,k}(z_1,\ldots,z_n)$ *is the* k*-th elementary symmetric polynomial on variables* $z_1,\ldots,z_n$. *Then,* $(a(n,k))_{n\in\mathbb{N},k\leq n}$ *is definable in* $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$.

6 Connecting the counting hierarchy to the τ-conjecture

In this section, we connect the τ-conjecture to the counting hierarchy. Specifically, we show that the collapse of $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$ implies that some explicit polynomial, whose coefficients are definable in $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$, is "easy". Formally, we prove the following theorem:

▶ **Theorem 21.** *Say,* $(a(n))_{n\in\mathbb{N}}$ *and* $(b(n,k))_{k\leq q(n),n\in\mathbb{N}}$ *are both definable in* $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$. *Here* q *is some* p-bounded *function. If* $\mathsf{p\text{-}log\text{-}Expsum} \in \mathsf{VFPT}^0_{\mathrm{nb}}$ *then the following holds:*

1. $\tau(a(n)) = n^{o(1)}$,
2. *If* $f_n(X) := \sum_{k=1}^{q(n)} b(n,k)X^k$ *then* $\tau(f_n) = n^{o(1)}$.

Proof. We can assume that if $a(n)$ is definable in $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$, $|a(n)| \leq 2^{n^c}$, that is, the bit-size of any integer definable in $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$ is polynomially bounded. Furthermore, if $\mathsf{p\text{-}log\text{-}Expsum} \in \mathsf{VFPT}^0_{\mathrm{nb}}$, then every language in $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$ has subexponential-size circuits by Theorem 13. We will use both facts below.

Proof of part (1). Let $a(n) = \sum_{j=1}^{p(n)} a(n,j)2^j$ be the binary decomposition of $a(n)$ and $p(n) = O(n^c)$. Define a new polynomial:

$$A_{\lceil \log n\rceil}(Y_1,\ldots,Y_{\mathsf{bit}(n)}) := \sum_{j=0}^{p(n)} a(n,j)Y_1^{j_1}\ldots Y_{\mathsf{bit}(n)}^{j_{\mathsf{bit}(n)}},$$

where $\mathsf{bit}(n) := \lceil \log(p(n))\rceil$. By our assumption, we can decide if $a(n,j) = b$ by a subexponential-size circuit, given input n and j in binary. Say, $C_r(\mathbf{N},\mathbf{J})$ is the corresponding circuit, where $r = \lfloor \log n\rfloor$. We have $C_r(n_1,\ldots,n_{\lfloor \log n\rfloor+1},j_1,\ldots,j_{\mathsf{bit}(n)}) = a(n,j)$, where the n_i's and the j_i's are the bits of n and j, respectively. Consider the polynomial

$$F_r(J_1,\ldots,J_{cr+1},N_1,\ldots,N_{r+1},Y_1,\ldots,Y_{cr+1}) := C_r(\mathbf{N},\mathbf{J})\cdot\prod_{i=1}^{cr+1}(J_iY_i+1-J_i).$$

Now, by our assumption and Theorem 13, we can say that F_r has $2^{o(r)}$ size constant-free algebraic circuits (of unbounded degree). Consider the exponential sum

$$\tilde{F}_r(\mathbf{N},\mathbf{Y}) := \sum_{j\in\{0,1\}^{cr+1}} F_r(j,\mathbf{N},\mathbf{Y}).$$

It is an instance of $\mathsf{p\text{-}log\text{-}Expsum}$ with $\tau(F_r) = 2^{o(r)}$. By assumption, this implies that $\tau(\tilde{F}_r) = 2^{o(r)}$. Finally, note that $A_{\lceil \log n\rceil}(\mathbf{Y}) = \tilde{F}_r(n_1,\ldots,n_{r+1},\mathbf{Y})$, and $a(n) = A_{\lceil \log n\rceil}(2^{2^0},\ldots,2^{2^{\mathsf{bit}(n)-1}})$. Therefore,

$$\tau(a(n)) \leq \tau(\tilde{F}_r) + \tau(2^{2^{\mathsf{bit}(n)-1}}) \leq n^{o(1)},$$

as desired.

Proof of part (2). Again we can assume that $|b(n,k)|$ has polynomially many bits. Let $b(n,k) = \sum_{j=1}^{p(n)} b(n,k,j)2^j$ be the binary decomposition with $p(n) = O(n^{c'})$ and $q(n) = O(n^c)$. Define

$$B_{\lceil \log n \rceil}(Y_1, \ldots, Y_{\mu(n)}, Z_1, \ldots, Z_{\lambda(n)}) \;:=\; \sum_{k=0}^{q(n)} \sum_{j=0}^{p(n)} b(n,k,j) Y_1^{j_1} \ldots Y_{\mu(n)}^{j_{\mu(n)}} Z_1^{k_1} \ldots Z_{\lambda(n)}^{k_{\lambda(n)}} .$$

Here $\mu(n) := \lceil \log(p(n)) \rceil$ and $\lambda(n) := \lceil \log(q(n)) \rceil$. Let the variable sets be $\mathbf{J} = (J_1, \ldots, J_{c'r+1}), \mathbf{N} = (N_1, \ldots, N_{r+1}), \mathbf{K} = (K_1, \ldots, K_{cr+1}), \mathbf{Y} = (Y_1, \ldots, Y_{c'r+1}), \mathbf{Z} = (Z_1, \ldots, Z_{cr+1})$, where again $r = \lfloor \log n \rfloor$. Define a new polynomial F_r as follows:

$$F_r(\mathbf{J}, \mathbf{K}, \mathbf{N}, \mathbf{Y}, \mathbf{Z}) \;:=\; D_r(\mathbf{N}, \mathbf{J}, \mathbf{K}) \cdot \prod_{m=1}^{c'r+1} (J_m Y_m + 1 - J_m) \prod_{s=1}^{cr+1} (K_s Z_s + 1 - Z_s) .$$

Like in the previous part of the proof, $(D_r(\mathbf{N}, \mathbf{J}, \mathbf{K}))_r$ is the circuit family for computing $(b(n,k,j))$. In particular,

$$D_r(n_1, \ldots, n_{r+1}, j_1, \ldots, j_{\mu(n)}, k_1, \ldots, k_{\lambda(n)}) \;=\; b(n,k,j) .$$

By our assumption, D_r has $2^{o(r)}$ size constant-free algebraic circuits (of unbounded degree). Consider,

$$\tilde{F}_r(\mathbf{N}, \mathbf{Y}, \mathbf{Z}) = \sum_{j \in \{0,1\}^{c'r+1}} \sum_{k \in \{0,1\}^{cr+1}} F_r(j, k, \mathbf{N}, \mathbf{Y}, \mathbf{Z}).$$

It is an instance of p-log-Expsum with $\tau(F_r)$ is $2^{o(r)}$. Since p-log-Expsum $\in \mathsf{VFPT}^0_{\mathrm{nb}} \implies \tau(\tilde{F}_r) = 2^{o(r)}$. Now, $B_{\lceil \log n \rceil}(\mathbf{Y}, \mathbf{Z}) = F_r(n_1, \ldots, n_{r+1}, \mathbf{Y}, \mathbf{Z})$ and

$$f_n(X) \;=\; B_{\lceil \log n \rceil}(2^{2^0}, \ldots, 2^{2^{\mu(n)-1}}, X^{2^0}, \ldots, X^{2^{\lambda(n)-1}}) .$$

Therefore, $\tau(f_n) \;\leq\; \tau(B_{\lceil \log n \rceil}) + \tau(2^{2^{\mu(n)}}) + \tau(X^{2^{\lambda(n)}}) \;\leq\; n^{o(1)}$, as desired. ◀

▶ **Theorem 22.** *If the τ-conjecture is true, then* p-log-Expsum $\notin \mathsf{VFPT}^0_{\mathrm{nb}}$.

Proof. Take the Pochhammer polynomial $p_n(X) = \prod_{i=1}^{n}(X+i)$. The coefficient of X^{n-k} in p_n will be $\sigma_k(1, \ldots, n)$, where $\sigma_k(z_1, \ldots, z_n)$ is the k-th elementary symmetric polynomial in variables $z_1, \ldots, z_n$. And $(\sigma_k(1, \ldots, n))_{n \in \mathbb{N}, k \leq n}$ is definable in linear counting hierarchy by Corollary 20. By Theorem 21, $(p_n)_{n \in \mathbb{N}}$ has $n^{o(1)}$ size constant-free circuit if p-log-Expsum is fixed-parameter tractable. But p_n has distinct n many integer roots. So, assuming the τ-conjecture, p-log-Expsum is not *fpt*. ◀

▶ Remark 23. Instead of taking the Pochhammer polynomial, there are many other possible choices for some explicit polynomial, see [6].

Finally, we prove the exponential lower bound for an exponential sum, proving Theorem 1.

▶ **Theorem 24** (Exponential algebraic lower bound)**.** *If the τ-conjecture is true, then there exists an n-variate polynomial family $\sum_{y \in \{0,1\}^n} g_n(X, y)$, which requires $2^{\Omega(n)}$-size circuits.*

Proof. If the τ-conjecture is true, then Theorem 22 shows that p-log-Expsum $\notin \mathsf{VFPT}^0_{\mathrm{nb}}$. By the contrapositive statement of Theorem 16, the existence of such a hard exponential sum follows. ◀

▶ Remark 25. The family g_n simply is a universal circuit of size polynomial in n, where the polynomial is large enough to simulate the computation of the Turing machine that shows that the n-th Pochhammer polynomial is definable in $\mathsf{CH}_{\mathsf{lin}}\mathsf{P}$.

7 Preliminaries II: The VW-hierarchy

In this section, we define different variants of the VW-hierarchy, which will be analogous to $\#W$-hierarchy, see [3]. We will consider circuits that can have unbounded fanin gates.

▶ **Definition 26** (Weft). *For an algebraic circuit C, the weft of C is the maximum number of unbounded fan-in gates on any path from a leaf to the root.*

For $n \geq k \in \mathbb{N}$, let $\langle {n \atop k} \rangle$ be the set of all vectors in $\{0,1\}^n$ which have exactly k many 1s.

▶ **Definition 27.**

1. *A parameterized p-*family $f_{n,k}(\mathbf{X})$ *is in* $\mathsf{VW}[\mathsf{F}]$ *iff there exists a p-*bounded *function $q(n)$ and p-*family $g_n(\mathbf{X}, y_1, \ldots, y_{q(n)})$ *such that* $f_{n,k} \leq_s^{\text{fpt}} \sum_{\overline{y} \in \langle {q(n) \atop k} \rangle} g_n(\mathbf{X}, y_1, \ldots, y_{q(n)})$ *and g_n can be computed by a polynomial-size formula.*
2. *A parameterized family $f_{n,k}(\mathbf{X})$ is in* $\mathsf{VW}_{\text{nb}}[\mathsf{F}]$ *iff there exists a p-*bounded *function $q(n)$ and family $g_n(\mathbf{X}, y_1, \ldots, y_{q(n)})$ such that* $f_{n,k} \leq_s^{\text{fpt}_{\text{nb}}} \sum_{\overline{y} \in \langle {q(n) \atop k} \rangle} g_n(\mathbf{X}, y_1, \ldots, y_{q(n)})$ *and g_n can be computed by a polynomial-size formula.*
3. *A parameterized p-*family $f_{n,k}(\mathbf{X})$ *is in* $\mathsf{VW}^0[\mathsf{F}]$ *iff there exists a p-*bounded *function $q(n)$ and p-*family $g_n(\mathbf{X}, y_1, \ldots, y_{q(n)})$ *such that* $f_{n,k} \leq_s^{\tau\text{-fpt}} \sum_{\overline{y} \in \langle {q(n) \atop k} \rangle} g_n(\mathbf{X}, y_1, \ldots, y_{q(n)})$ *and g_n can be computed by a constant-free, polynomial-size formula.*
4. *A parameterized family $f_{n,k}(\mathbf{X})$ is in* $\mathsf{VW}^0_{\text{nb}}[\mathsf{F}]$ *iff there exists a p-*bounded *function $q(n)$ and family $g_n(\mathbf{X}, y_1, \ldots, y_{q(n)})$ such that* $f_{n,k} \leq_s^{\tau\text{-fpt}_{\text{nb}}} \sum_{\overline{y} \in \langle {q(n) \atop k} \rangle} g_n(\mathbf{X}, y_1, \ldots, y_{q(n)})$ *and g_n can be computed by a constant-free, polynomial-size formula.*

In some sense, $\mathsf{VW}[\mathsf{F}]$ is a substitution of a *weighted sum* of formulas. We will define $\mathsf{VW}[\mathsf{P}]$ as a weighted sum as above, but summing over an arbitrary circuit of polynomial-size. Similarly, we can define $\mathsf{VW}^0[\mathsf{P}]$, and its counterpart in the unbounded setting, i.e. $\mathsf{VW}_{\text{nb}}[\mathsf{P}]$, and $\mathsf{VW}^0_{\text{nb}}[\mathsf{P}]$.

Finally, we will define the completeness notion:

▶ **Definition 28.** *We will say a parameterized p-*family $f_{n,k}$ *is* $\mathsf{VW}[\mathsf{F}]$*-hard if every* $g_{n,k} \in \mathsf{VW}[\mathsf{F}]$, $g_{n,k} \leq_s^{\text{fpt}} f_{n,q}$. *Similarly, we can define completeness for* $\mathsf{VW}[\mathsf{P}]$.

We can also define completeness and hardness in the constant-free and unbounded models.

8 Conditional collapsing of VW-hierarchy and applications

Let us recall the definition of k-degree n-variate ($n \geq k$) *elementary symmetric polynomial* $\sigma_{n,k}(\mathbf{X}) := \sum_{y \in \langle {n \atop k} \rangle} X_1^{y_1} X_2^{y_2} \ldots X_n^{y_n}$. It is known that $(\sigma_{n,k})_n \in \mathsf{VP}^0$, with a simple dynamic programming algorithm; see [13, Section 4]. Let us define a new polynomial family $B_{n,k}(\mathbf{X})$, which will be important in the latter part of the section: $B_{n,k}(\mathbf{X}) := \sum_{t=0}^{n-k} (-1)^t \binom{k+t}{k} \cdot \sigma_{n,k+t}(\mathbf{X})$. The following claim is crucial:

▷ Claim 29. For $y \in \{0,1\}^n$, $B_{n,k}(y) = \begin{cases} 1, & \text{if } y \in \langle {n \atop k} \rangle, \\ 0, & \text{otherwise.} \end{cases}$

Proof. For a string $y \in \{0,1\}^n$, we will call the *weight* of y, denoted $\text{wt}(y)$, the number of 1's present in y. Note that if $\text{wt}(y) < k$, then $\sigma_{n,k}(y) = 0$ implying $B_{n,k}(y) = 0$. Similarly if $\text{wt}(y) = k$, then $B_{n,k}(y) = \sigma_{n,k}(y)$, which will be exactly equal to 1. Now if $\text{wt}(y) = k + r$ where $r > 0$, then

$$B_{n,k}(y) \;=\; \sum_{t=0}^{n-k}(-1)^t\binom{k+t}{k}\cdot\sigma_{n,k+t}(y) \;=\; \sum_{t=0}^{r}(-1)^t\binom{k+t}{k}\cdot\sigma_{n,k+t}(y)$$
$$= \sum_{t=0}^{r}(-1)^t\binom{k+t}{k}\cdot\binom{k+r}{k+t}$$
$$= \sum_{t=0}^{r}(-1)^t\frac{(k+r)!}{k!t!(r-t)!}\,.$$

Let us further define the tri-variate polynomial $Q(x,y,z) := (x+y-z)^{k+r} \in \mathbb{Z}[x,y,z]$. Note that the coefficient of x^k in $Q(x,y,z)$ is

$$\sum_{t=0}^{r} y^{r-t}z^t(-1)^t\cdot\frac{(k+r)!}{k!t!(r-t)!}\,.$$

Now putting $y=z=1$, we get the coefficient exactly equal to $B_{n,k}(y)$; since $r\neq 0$, we can say that the coefficient of x^k in $Q(x,1,1)$ is 0, which finally implies that $B_{n,k}(y)=0$. ◁

Now we are ready to prove the following transfer theorem from the parameterized Valiant's classes to Valiant's algebraic models.

▶ **Theorem 30.** $\mathsf{VW}^0[\mathsf{P}] \neq \mathsf{VFPT}^0 \implies \mathsf{VP}^0 \neq \mathsf{VNP}^0$. *Similarly,* $\mathsf{VW}[\mathsf{P}] \neq \mathsf{VFPT} \implies \mathsf{VP} \neq \mathsf{VNP}$.

Proof. We will prove the contraposition. Assume that $\mathsf{VP}^0 = \mathsf{VNP}^0$. As mentioned before, we know that $(\sigma_{n,k})_n \in \mathsf{VP}^0$. Further, since $k \in [n]$, for $t \leq n-k$, it is trivial to see that $\tau(\binom{k+t}{k}) = n^{O(1)}$. Therefore, for each $0 \leq t \leq n-k$, $(-1)^t\binom{k+t}{k}\cdot\sigma_{n,k+t}(\mathbf{X})$ has a VP^0-circuit. Since VP^0 is closed under polynomially many additions, it follows that $(B_{n,k})_n \in \mathsf{VP}^0$.

Let $q_{n,k} \in \mathsf{VW}^0[\mathsf{P}]$. By definition, there is a polynomial family $p_{n,k}$ of the above form $p_{n,k}(\mathbf{X}) := \sum_{y\in\langle{n \atop k}\rangle} g_n(\mathbf{X},y)$, where $g_n(\mathbf{X},\mathbf{Y})$ is in VP^0, such that $q_{n,k} \leq_s^{fpt} p_{n,k}$. By Claim 29, it follows that

$$p_{n,k} \;=\; \sum_{y\in\{0,1\}^n} g_n(\mathbf{X},y)\cdot B_{n,k}(y)\,.$$

We have already proved above that $B_{n,k}$ has $\mathsf{poly}(n)$ sized constant-free circuits. Hence, $g_n(\mathbf{X},y)B_{n,k}(y)$ has constant-free $\mathsf{poly}(n)$-size circuit. Therefore, by definition and our primary assumption, it follows that $p_{n,k} \in \mathsf{VNP}^0 = \mathsf{VP}^0 \subseteq \mathsf{VFPT}^0$. Since, VFPT^0 is *closed* under constant-free fpt-substitution (Lemma 7), it follows that $q_{n,k} \in \mathsf{VFPT}^0$, implying $\mathsf{VW}^0[\mathsf{P}] \subseteq \mathsf{VFPT}^0$.

The proof in the usual (not constant-free) model also follows essentially along the same line as above. ◀

▶ Remark 31. The above theorem holds in the unbounded regime as well, i.e., $\mathsf{VW}^0_{\mathrm{nb}}[\mathsf{P}] \neq \mathsf{VFPT}^0_{\mathrm{nb}} \implies \mathsf{VP}^0_{\mathrm{nb}} \neq \mathsf{VNP}^0_{\mathrm{nb}}$ (which further implies $\mathsf{VP}^0 \neq \mathsf{VNP}^0$, see [16]). Similarly, $\mathsf{VW}_{\mathrm{nb}}[\mathsf{P}] \neq \mathsf{VFPT}_{\mathrm{nb}} \implies \mathsf{VP}_{\mathrm{nb}} \neq \mathsf{VNP}_{\mathrm{nb}}$.

We now aim to prove a *conditional separation* of $\mathsf{VW}^0_{\mathrm{nb}}[\mathsf{P}]$ and $\mathsf{VFPT}^0_{\mathrm{nb}}$, by showing that $\mathsf{VW}^0_{\mathrm{nb}}[\mathsf{P}] = \mathsf{VFPT}^0_{\mathrm{nb}}$ implies a collapse of the linear counting hierarchy. To show this, we will show that $\mathsf{VW}^0_{\mathrm{nb}}[\mathsf{P}] = \mathsf{VFPT}^0_{\mathrm{nb}} \implies \mathsf{p\text{-}log\text{-}Expsum} \in \mathsf{VFPT}^0_{\mathrm{nb}}$ (Corollary 34) from which the collapse of the linear counting hierarchy follows.

▶ **Theorem 32.** *Let* $f(\mathbf{X}) = \sum_{y\in\{0,1\}^{\ell(n)}} g(\mathbf{X}, y)$, *where* $\ell(\cdot)$ *is a linear function and* g *is computed by an arithmetic circuit of size* $m = 2^{O(n^c)}$ *for some constant* c. *Then,* $f(\mathbf{X})$ *can be written as*

$$f(\mathbf{X}) = \sum_{e\in\langle {b(m) \atop k}\rangle} G(\mathbf{X}, e)\,,$$

for some p-bounded *function* b *and* $k = \ell(n)/\log m$ *and* G *has* poly(m) *size circuits.*

Proof. Let $f(\mathbf{X})$ be an instance of p-log-Expsum, i.e., $f(\mathbf{X}) = \sum_{y\in\{0,1\}^n} g(\mathbf{X}, y)$, where $g(\mathbf{X}, \mathbf{Y})$ has size m constant-free circuit. Here we mention that, although we just take sum over n *variables* here for the ease of presentation, the same proof also works if we sum over $\ell(n)$ many variables for some linear function ℓ.

Let us partition the variable set $\mathbf{Y} = \{Y_1, \ldots, Y_n\} = E_1 \sqcup \cdots \sqcup E_k$. Here $k = n/\log m$, and for all i, $|E_i| = \log m$. For each $S \subseteq E_i$, we take a *new variable* Z_i^S and we do this for all i. Define $\overline{Z_i} := \{Z_i^S : S \subseteq E_i\}$ and $\mathbf{Z} = \bigcup_i \overline{Z_i}$. The number of $\mathbf{Z}$-variables is $2^{\log m} \cdot k$, which is polynomial in m.

Let us call an assignment of $\mathbf{Z}$ variables a *good assignment*, if *exactly* one variable in each set $\overline{Z_i}$ is set to be 1. Below we show that there is a one-to-one correspondence between $\{0,1\}$ assignments to the $\mathbf{Y}$ variables and *good assignments* to the $\mathbf{Z}$ variables.

Let φ be a homomorphism from $R[\mathbf{Y}] \to R[\mathbf{Z}]$, where $R := \mathbb{F}[\mathbf{X}]$, such that $\varphi : Y_i \mapsto \prod_{S\subseteq E_i,\ Y_j\notin S}(1 - Z_i^S)$. Let us define $\tilde{g}(\mathbf{X}, \mathbf{Z}) := \varphi(g)$. Now let us fix an assignment $y \in \{0,1\}^n$ to the $\mathbf{Y}$ variables. We construct a corresponding good assignment of $\mathbf{Z}$. For each E_i of $\mathbf{Y}$, we have some $S_i \subseteq E_i$ such that *each* variable of E_i, which is in S_i, gets value 1. The remaining variables in $E_i \setminus S_i$ get value 0 (so that it corresponds to y). Pick this particular $S_i \subseteq E_i$. Note that this S_i is *unique* (it can be the empty set). Now set $Z_i^{S_i} = 1$ and $Z_i^S = 0$, if $S \neq S_i$, for all $i \in [k]$.

Each variable in $\bigcup_i S_i$ gets the value 1 and variables in $\bigcup_i (E_i \setminus S_i)$ are assigned 0. Under the map φ, any $Y_j \in E_1 \setminus S_1$ is replaced by $\prod_{S\subseteq E_1,\ Y_j\notin S}(1 - Z_1^S)$. Since, $S_1 \subseteq E_1$ and $Y_j \notin S_1$, $(1 - Z_1^{S_1})$ occurs in the product. And, hence the product becomes 0. Now, let $Y_\ell \in S_1$ and $\varphi(Y_\ell) = \prod_{S\subseteq E_1,\ Y_\ell\notin S}(1 - Z_1^S)$. As $Y_\ell \in S_1$, $(1 - Z_1^{S_1})$ does not contribute to the product. Thus, under the assignment defined before, $\varphi(Y_\ell)$ becomes 1. This argument holds for any E_i. Therefore, one can conclude that

$$f = \sum_{e:\ e \text{ is a } good \text{ assignment}} \tilde{g}(\mathbf{X}, e)\,.$$

Note that the weft of the circuit for $\tilde{g}$ has increased by 1 (from that of g), and the size has also increased by a polynomial (in m) factor. To capture a k-weight good assignment exactly, define a new polynomial $p(\mathbf{Z}) \in \mathbb{F}[\mathbf{Z}]$ as follows:

$$p(\mathbf{Z}) := \prod_{i=1}^{k} \left(\sum_{S\subseteq E_i} Z_j^S \right).$$

Clearly, p has a weft-2 circuit of size poly(m). Further, it is simple to see that for any k-weight $\{0,1\}$ assignment e to the $\mathbf{Z}$ variables, $p(e) = 1$ iff e is a *good* assignment because from each of the product terms, only one variable will survive. Therefore,

$$f = \sum_{e\in\langle {b(m) \atop k}\rangle} p(e) \cdot \tilde{g}(\mathbf{X}, e)\,, \qquad \text{where } b(m) = |\mathbf{Z}|\,.$$

We set $G(\mathbf{X},\mathbf{Z}) := p(\mathbf{Z})\tilde{g}(\mathbf{X},\mathbf{Z})$. By the construction, $\tilde{g}$ has weft $\leq t+1$, p has weft ≤ 2, and $\tilde{g}, p$ have $\mathsf{poly}(m)$ size circuits. So, this ends our proof. ◀

▶ Remark 33. The construction above increases the weft by one.

▶ **Corollary 34.** $\mathsf{VW}^0_{\mathrm{nb}}[\mathsf{P}] = \mathsf{VFPT}^0_{\mathrm{nb}} \implies \mathsf{p\text{-}log\text{-}Expsum} \in \mathsf{VFPT}^0_{\mathrm{nb}}$.

Proof. In Theorem 32 we have reduced an instance of p-log-Expsum to an instance of $\mathsf{VW}^0_{\mathrm{nb}}[\mathsf{P}]$ with parameter $k = \ell(n)/\log m$. By our assumption $\mathsf{VW}^0_{\mathrm{nb}}[\mathsf{P}] = \mathsf{VFPT}^0_{\mathrm{nb}}$ and thus we can say that p-log-Expsum $\in \mathsf{VFPT}^0_{\mathrm{nb}}$. ◀

▶ Remark 35. If one restricts p-log-Expsum to exponential sums over g, where g is a p-family (i.e., it has polynomial degree and size), denoted $\mathsf{p\text{-}log\text{-}Expsum}_{bd}$ (bd for bounded-degree), then the above proof similarly implies that $\mathsf{VW}^0[\mathsf{P}] = \mathsf{VFPT}^0 \implies \mathsf{p\text{-}log\text{-}Expsum}_{bd} \in \mathsf{VFPT}^0$.

Similarly, we also prove a lower bound for the class $\mathsf{VW}_{\mathrm{nb}}[\mathsf{P}]$, assuming an fpt lower bound on p-log-Expsum.

▶ **Theorem 36.** *Say that any family* $F_{m,k}(\mathbf{X}) = \sum_{e\in\langle {b(m) \atop k}\rangle} G(\mathbf{X},e) \in \mathsf{VW}^0_{\mathrm{nb}}[\mathsf{P}]$ *has* $2^{o(n)}\mathsf{poly}(m)$ *size constant-free circuits where* $\tau(G) \leq m$, $n := k\log m/c$, *for some constant* c *and* b *is some* p-bounded *function. Then,* p-log-Expsum $\in \mathsf{VFPT}^0_{\mathrm{nb}}$.

Proof. Take an instance of p-log-Expsum, $f(\mathbf{X}) = \sum_{y\in\{0,1\}^{\ell(n)}} g(\mathbf{X},y)$, for some $\ell(n) = O(n)$. And g has a constant-free circuit of size m. By Theorem 32, we can make it an instance of $\mathsf{VW}^0[\mathsf{P}]$ and say,

$$f = \sum_{e\in\langle {b(m) \atop k}\rangle} \tilde{g}(\mathbf{X},e)\,, \qquad \text{where } b \text{ is } p\text{-bounded}, \;\; k = \ell(n)/\log m$$

By our assumption, f has a constant-free circuit of size $2^{o(n)}\mathsf{poly}(m) = 2^{O(n/i(n))}\mathsf{poly}(m)$ for some unbounded and non-decreasing function $i : \mathbb{N} \to \mathbb{N}$. Let h be a non-decreasing function, so that $h(i(n)) \geq 2^n$. We shall prove that f has $h(k)\mathsf{poly}(m)$ size constant-free circuit. If $m \geq 2^{n/i(n)}$, clearly, f has $\mathsf{poly}(m)$ size constant-free circuit. Otherwise, if $m < 2^{n/i(n)}$, this will imply $i(n) \leq n/\log m = k$. And hence, $h(k) \geq 2^n$. So, f has $h(k)\mathsf{poly}(m)$ size constant-free circuit. ◀

9 Restricted permanent

A *cycle cover* of a directed graph is a collection of node-disjoint directed cycles such that each node is contained in exactly one cycle. Cycle covers of a directed graph stand in one-to-one relation with permutations of the nodes.

▶ **Definition 37.** *A cycle cover is* (k,c)*-restricted, if it contains one cycle of length* k *and all other cycles have length* $\leq c$.

Let $G = (V,E)$ be directed graph and $w : E \to R$ be a weight function. Here R is a ring and typically the ring of polynomials. The weight of a cycle cover C of G is the product of the weights of the edges in it, that is, $w(C) = \prod_{e\in C} w(e)$.

▶ **Definition 38.** *The* (k,c)*-restricted permanent of an edge-weighted directed graph* G *is*

$$\mathrm{per}^{(k,\leq c)}(G) = \sum_C w(C),$$

where the sum is over all (k,c)*-restricted cycle covers.*

If $X = (X_{i,j})$ is a variable matrix, then $\mathrm{per}_n(X)$ is the permanent of the complete directed graph with the edge weights $w(i,j) = X_{i,j}$. The (k,c)-restricted permanent family $\mathrm{per}^{(k,\leq c)} = (\mathrm{per}_n^{(k,\leq c)}(X_n))$, where X_n is an $n \times n$-variables matrix. $\mathrm{per}^{(k,\leq c)}$ is a parameterized family, n is the input size, k is the parameter, and c will be some constant to be determined later.

On general graphs, the restricted permanent is very powerful, even if we keep the parameter fixed.

▶ **Proposition 39.** *The $(2,2)$-restricted permanent family is* VNP*-complete.*

If we restrict the underlying graph appropriately, then the restricted permanent is complete for the class VW[F]. Recall that the girth of an undirected graph is the length of a shortest cycle in the graph. When we talk of the girth of a directed graph, we mean the girth of the graph when we disregard the direction of edges. Furthermore, when we talk about the treewidth of a directed graph, we mean the treewidth of the underlying undirected graph.

▶ **Definition 40.** *A directed graph $G = (V,E)$ is* (c,b)-nice *if we can partition the nodes $V = V_1 \cup V_2$ into two disjoint sets, such that*

1. *the graph induced by V_1 has girth $> c$ (not counting self-loops),*
2. *every node in V_1 has a self-loop, and*
3. *the graph induced by V_2 has tree-width bounded by b.*
4. *every cycle that contains vertices from V_1 and V_2 has length $> c$.*

Our main result is the following completeness result.

▶ **Theorem 41.** *Let c and b be constants. Let (G_n) be a family of (c,b)-nice graphs. Then the (k,c)-restricted permanent is in* VW[F].

▶ **Theorem 42.** *Let the underlying field have characteristic 0. There is a constant b and a family of $(4,b)$-nice graphs (H_n) such that the $(3k,4)$-restricted permanent of H_n forms a family of* VW[F]*-hard polynomials.*

References

1 Yaroslav Alekseev, Dima Grigoriev, Edward A. Hirsch, and Iddo Tzameret. Semi-algebraic proofs, IPS lower bounds and the τ-conjecture: Can a natural number be negative? *CoRR*, abs/1911.06738, 2019. arXiv:1911.06738.

2 Eric Allender, Michal Koucký, Detlef Ronneburger, Sambuddha Roy, and V. Vinay. Time-space tradeoffs in the counting hierarchy. In *Proceedings of the 16th Annual IEEE Conference on Computational Complexity, Chicago, Illinois, USA, June 18-21, 2001*, pages 295–302. IEEE Computer Society, 2001. doi:10.1109/CCC.2001.933896.

3 Markus Bläser and Christian Engels. Parameterized Valiant's Classes. In Bart M. P. Jansen and Jan Arne Telle, editors, *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*, volume 148 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2019.3.

4 Lenore Blum, Felipe Cucker, Mike Shub, and Steve Smale. Algebraic settings for the problem "P$\neq$ NP?". In *The Collected Papers of Stephen Smale: Volume 3*, pages 1540–1559. World Scientific, 2000. doi:10.1142/9789812792839_0025.

5 Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin (New Series) of the American Mathematical Society*, 21(1):1–46, 1989. URL: https://www.ams.org/journals/bull/1989-21-01/S0273-0979-1989-15750-9/S0273-0979-1989-15750-9.pdf.

6 Peter Bürgisser. On defining integers and proving arithmetic circuit lower bounds. *computational complexity*, 18:81–103, April 2009. doi:10.1007/s00037-009-0260-x.

7 Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlen. Exponential time complexity of the permanent and the Tutte polynomial. *ACM Trans. Algorithms*, 10(4):21:1–21:32, 2014. doi:10.1145/2635812.

8 Pranjal Dutta. Real τ-conjecture for sum-of-squares: A unified approach to lower bound and derandomization. In *International Computer Science Symposium in Russia*, pages 78–101. Springer, 2021.

9 Jörg Flum and Martin Grohe. Parametrized complexity and subexponential time (column: Computational complexity). *Bull. EATCS*, 84:71–100, 2004.

10 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

11 David G. Glynn. The permanent of a square matrix. *European Journal of Combinatorics*, 31(7):1887–1891, 2010. doi:10.1016/j.ejc.2010.01.010.

12 William Hesse, Eric Allender, and D.A. Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65(4):695–716, 2002. doi:10.1016/S0022-0000(02)00025-9.

13 Stasys Jukna and Georg Schnitger. On the optimality of bellman–ford–moore shortest path algorithm. *Theoretical Computer Science*, 628:101–109, 2016.

14 Pascal Koiran. Valiant's model and the cost of computing integers. *computational complexity*, 13:131–146, 2005.

15 Pascal Koiran. Shallow circuits with high-powered inputs. In *Innovations in Computer Science – ICS*, 2011. URL: https://hal-ens-lyon.archives-ouvertes.fr/ensl-00477023v4/document.

16 Pascal Koiran and Sylvain Perifel. Interpolation in Valiant's theory. *Computational Complexity*, 20:1–20, 2011.

17 Guillaume Malod. The complexity of polynomials and their coefficient functions. In *Twenty-Second Annual IEEE Conference on Computational Complexity (CCC'07)*, pages 193–204. IEEE, 2007.

18 Guillaume Malod and Natacha Portier. Characterizing Valiant's algebraic complexity classes. *J. Complexity*, 24(1):16–38, 2008. doi:10.1016/j.jco.2006.09.006.

19 Herbert John Ryser. *Combinatorial Mathematics*, volume 14 of *Carus Mathematical Monographs*. Mathematical Association of America, 1963.

20 Michael Shub and Steve Smale. On the intractability of Hilbert's Nullstellensatz and an algebraic version of " NP$\neq$ P?". *Duke Mathematical Journal*, 81(1):47–54, 1995. URL: http://www.cityu.edu.hk/ma/doc/people/smales/pap97.pdf.

21 Sébastien Tavenas. *Bornes inferieures et superieures dans les circuits arithmetiques*. PhD thesis, Ecole Normale Supérieure de Lyon, 2014. URL: https://tel.archives-ouvertes.fr/tel-01066752/document.

22 Jacobo Torán. Complexity classes defined by counting quantifiers. *J. ACM*, 38:753–774, July 1991. doi:10.1145/116825.116858.

23 Leslie G. Valiant. Completeness classes in algebra. In Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho, editors, *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 – May 2, 1979, Atlanta, Georgia, USA*, pages 249–261. ACM, 1979. doi:10.1145/800135.804419.

24 Klaus W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23(3):325–356, 1986. doi:10.1007/BF00289117.

Random Separating Hyperplane Theorem and Learning Polytopes

Chiranjib Bhattacharyya
Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India

Ravindran Kannan
Department of Operations Research, Carnegie Mellon University, Pittsburgh, USA

Amit Kumar
Department of Computer Science and Engineering, Indian Institute of Technology Delhi, India

Abstract

The Separating Hyperplane theorem is a fundamental result in Convex Geometry with myriad applications. The theorem asserts that for a point a not in a closed convex set K, there is a hyperplane with K on one side and a strictly on the other side. Our first result, Random Separating Hyperplane Theorem (RSH), is a strengthening of this for polytopes. RSH asserts that if the distance between a and a polytope K with k vertices and unit diameter in $\Re^d$ is at least δ, where δ is a fixed constant in $(0, 1)$, then a randomly chosen hyperplane separates a and K with probability at least $1/\operatorname{poly}(k)$ and margin at least $\Omega\left(\delta/\sqrt{d}\right)$.

RSH has algorithmic applications in learning polytopes. We consider a fundamental problem, denoted the "Hausdorff problem", of learning a unit diameter polytope K within Hausdorff distance δ, given an optimization oracle for K. Using RSH, we show that with polynomially many random queries to the optimization oracle, K can be approximated within error $O(\delta)$. To our knowledge, this is the first provable algorithm for the Hausdorff Problem in this setting. Building on this result, we show that if the vertices of K are well-separated, then an optimization oracle can be used to generate a list of points, each within distance $O(\delta)$ of K, with the property that the list contains a point close to each vertex of K. Further, we show how to prune this list to generate a (unique) approximation to each vertex of the polytope. We prove that in many latent variable settings, e.g., topic modeling, LDA, optimization oracles do exist provided we project to a suitable SVD subspace. Thus, our work yields the first efficient algorithm for finding approximations to the vertices of the latent polytope under the well-separatedness assumption. This assumption states that each vertex of K is far from the convex hull of the remaining vertices of K, and is much weaker than other assumptions behind algorithms in the literature which find vertices of the latent polytope.

2012 ACM Subject Classification Theory of computation → Unsupervised learning and clustering

Keywords and phrases Separating Hyperplane Theorem, Learning Polytopes, Optimization Oracles

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.25

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2307.11371 [9]

1 Introduction

The Separating Hyperplane theorem is a fundamental result in Convex Geometry with myriad applications (see e.g. [6]). The theorem asserts that for a point a not in a closed convex set K, there is a hyperplane with K on one side and a strictly on the other side.

This paper makes two main contributions. Our theoretical contribution, which is an extension of the classical Separating Hyperplane Theorem, is what we call the Random Separating Hyperplane Theorem (RSH). Our main algorithmic contribution is to use RSH to prove that a natural algorithm, which we call the k-OLP algorithm, can learn (vertices of) latent polytopes arising in a number of problems in Latent Variable Models including

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 25; pp. 25:1–25:20


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Clustering, Mixture Learning , LDA (linear discriminant analysis), Topic Models. The algorithmic result is shown by reducing the problem of learning latent polytopes in a variety of settings to that of constructing approximate optimization oracles for the corresponding polytopes. Bulk of our algorithmic contribution is in proving this reduction and the existence of such oracles.

1.1 Random Separating Hyperplane Theorem (RSH)

RSH draws its motivation mainly from the Separating Hyperplane Theorem of Convex Geometry. It also has connections to the Johnson-Lindenstrauss Random Projection theorem [15]. The Separating Hyperplane Theorem formally states that given a closed convex set K and a point $a \notin K$, there exists a vector u such that

$$u \cdot a > \text{Max}_{y \in K}\, u \cdot y.$$

The following question arises: Does a randomly picked u separate a from K? Taking into account some necessary conditions for a positive answer, we can ask if the following inequality holds for a randomly chosen u: (here Δ is the diameter of K, a is at distance at least $\delta\Delta$ from K, where $\delta \in (0,1)$):

$$\mathbf{Pr}\left(u \cdot a \geq \text{Max}_{y \in K} u \cdot y + |u|\alpha\delta\Delta\right) \geq 1/\text{poly}_{\delta},^{1} \tag{1}$$

with α being as high as possible.

The question (1) is also motivated from the Johnson-Lindenstrauss Random Projection theorem (JL theorem) [15] which states that if a, b are points in $\mathbf{R}^d$ and U is a random subspace of dimension s, then with probability bounded away from 0, the distance between the projection of a and b on U is at least $\Omega(|a-b|\sqrt{s}/\sqrt{d})$. The following natural generalization of this is interesting already for $s = 1$:

Instead of b being a point, if it is now a polytope K, does a similar lower bound on the distance of a to K in the projection onto a random line hold?

It is easy to see that in spirit, this is the same question as whether (1) holds. It is also easy (see below) to see that the projection shrinks the distance between a and K by a factor of $\Omega^*(\sqrt{d})$. The RSH theorem proves that the shrinkage is $O(\sqrt{d})$ thus making this parameter nearly (within log factors) tight. We now state the RSH theorem:

▶ **Theorem 1** (Random Separating Hyperplane Theorem(RSH): Informal version)**.** *Suppose K is a k vertex polytope with diameter $\Delta(K)$ and a is a point at distance at least $\delta\Delta(K)$, $\delta \in (0,1)$, from K. Let V be an $m-$dimensional subspace containing $K \cup \{a\}$. For a random Gaussian vector $u \in V$, the following event happens with probability at least 1/poly$_\delta(k)$:*

$$u \cdot a \geq Max_{y \in K} u \cdot y + \frac{\delta\Delta(K)|u|}{10\sqrt{m}}. \tag{2}$$

We provide a simple example where K is a line segment (see Appendix A) to show that the factor $\sqrt{m}$ cannot be improved. It is also interesting to note that the success probability of the event in (2) needs to depend on k. (see Appendix A). In particular, RSH does not hold for general convex sets (where k is not necessarily finite.)

[1] poly$_\delta(z)$ denotes $z^{\text{poly}(1/\delta)}$.

1.2 Algorithmic application of RSH

We now discuss the second main contribution of our work, i.e., applications of RSH to learning vertices of a latent polytope. We begin with the definition of an approximate optimization oracle.

▶ **Definition 2.** *For a non-empty convex set $K \subseteq \Re^d$ and $\varepsilon \in (0,1)$, an Optimization oracle for K with error ε, denoted $\mathsf{OptOr}_\varepsilon(K)$ oracle, takes as input any $u \in \Re^d, |u| = 1$, and returns a point $x(u)$ satisfying* both *these conditions :*

- $x(u) \in K + \varepsilon\Delta(K)B_d$, *where B_d is the unit ball, i.e., $\{x \in \Re^d : |x| \leq 1\}$, and*
- $u \cdot x(u) \geq Max_{y \in K} u \cdot y - \varepsilon\Delta(K)$

Problem Formulation. Several latent variable problems including Clustering, LDA, MMBM can be reduced (See Section 6 for details) to a problem that we call k-OLP: given an ε-optimization oracle for a k vertex polytope $K \subseteq \mathbf{R}^d$, learn the vertices of K (approximately). We define two simpler (than k-OLP) problems – ListLearn and Hausdorff, that are related to k-OLP, and then we define k-OLP.

The first problem, Hausdorff, seeks to find an approximation to a polytope K when we are given an approximate optimization oracle for the polytope.

▶ **Definition 3** ((ε,δ)-Hausdorff-Problem)**.** *Given an $\mathsf{OptOr}_\varepsilon(K)$ oracle for a polytope K in $\mathbf{R}^d$ with k vertices, find a set P of $m = \mathrm{poly}_\delta(dk)$ points such that $\mathsf{Haus}(\mathsf{CH}(P), K) \leq \delta\Delta(K)$, where, Haus denotes Hausdorff distance (see Definition 20 for a formal definition), and $\mathsf{CH}(P)$ is the convex hull of P.*

In the problem ListLearn, we also wish to find a small list of points, such that each vertex of K is close to at least one point in this list.

▶ **Definition 4** ((ε,δ)-ListLearn Problem)**.** *Given an $\mathsf{OptOr}_\varepsilon(K)$ oracle for a polytope K in $\Re^d$ with k vertices, each separated from the convex hull of the other $k-1$ vertices by at least $\delta\Delta(K)$, find a list $P \subseteq K + \delta\Delta(K)B_d$ of $m = \mathrm{poly}_\delta(dk)$ points such that for every vertex v of K, there is some $v' \in P$ with $|v - v'| \leq \delta\Delta(K)/10$.*

When the parameters ε, δ will be clear from the context, we shall abbreviate the above two problems as Hausdorff and ListLearn problems respectively. It is not difficult to see that any solution P to ListLearn is also a solution to Hausdorff, but, the converse need not hold: $\mathrm{CH}(P)$ may nearly contain K without P having any point close to some vertex of K. Our technical results (see below for the informal versions) show that if $\varepsilon \in O_\delta(1/\sqrt{d})$ [2], then we can solve the above-mentioned problems efficiently and indeed then, the following simple algorithm gives the desired answers (the proof crucially uses RSH):

Random Probes Algorithm

- Pick uniformly at random unit vectors $u_1, u_2, \ldots u_m$, where $m = \mathrm{poly}_\delta(dk)$.
- Return P, which is the set of m answers of the $\mathsf{OptOr}_\varepsilon(K)$ oracle to the queries $u_1, u_2, \ldots u_m$.

The first result (see Theorem 21 for a formal statement) states that the convex hull of answers to polynomially many random queries to the approximate optimization oracle approximates K well.

[2] $O_\delta(x)$ stands for $f(\delta)x$ for some function f.

▶ **Theorem 5** (Hausdorff Approximation from oracle (Informal)). *Consider an instance of the* (ε, δ)-Hausdorff *problem for a polytope* $K \subseteq \Re^d$. *Assume that* $\varepsilon \in O_\delta(1/\sqrt{d})$, *and let* P *be the set of points returned by the* **Random Probes Algorithm** *above. Then with high probability,*

$$\mathsf{Haus}(CH(P), K) \leq \delta\Delta(K).$$

The second result (see Theorem 28 for a formal statement) shows that as long as each vertex of K is *well-separated* from the convex hull of the other vertices of K, the set P constructed by the **Random Probes Algorithm** contains an approximation to each of the vertices. Thus, answers to polynomially many random queries list-learns the polytope.

▶ **Theorem 6** (List-Learning from Oracle (Informal)). *Consider an instance of the* ListLearn *problem for a polytope* $K \subseteq \Re^d$, *and assume that* $\varepsilon \in O_\delta(1/\sqrt{d})$. *Then, with high probability, the set* P *output by the* **Random Probes Algorithm** *above has the following property: for every vertex* a *of* K, *there is a point* $a' \in P$ *with*

$$|a' - a| \leq \delta\Delta(K)/10.$$

The $\sqrt{d}$ factor in both the above theorems is near-optimal (within a $\log d$ factor). Indeed, we shall prove:

▶ **Theorem 7** (Oracle Lower Bound). *The problem where, one is required to output a point which is within* $\Delta(K)/10$ *of some vertex of* K, *given only by an* $\mathsf{OptOr}_\varepsilon(K)$ *oracle, cannot be solved in deterministic polynomial time when* $\varepsilon \geq 8 \ln d/\sqrt{d}$.

We now define the k-OLP problem (the parameters ε, δ in the definition will often be clear from the context and may not be mentioned explicitly). The problem statement is similar to that of ListLearn, but we want to output a list of exactly k points.

▶ **Definition 8** ((ε, δ)-k-OLP Problem). *Under the same hypothesis as for the* ListLearn *problem, find a set of points* P, $|P| = k$, *satisfying the following condition: for each vertex* v *of* K, *there is a (unique) point* $v' \in P$ *such that* $|v - v'| \leq \delta\Delta(K)/10$.

Our next result gives a strengthening of Theorem 6.

▶ **Theorem 9.** *Consider an instance of the* k-OLP *problem on a polytope* $K \subseteq \Re^d$, *and assume* $\varepsilon \in O_\delta(1/\sqrt{d})$ *in a* k-OLP *problem. Let* P *be the set of points returned by the* **Random Probes Algorithm**. *Then, in polynomial time, we can find a* $Q \subseteq P, |Q| = k$ *satisfying the following condition: for each vertex* v *of* K, *there is a (unique)* $v' \in P, |v - v'| \leq \delta\Delta(K)/10$.

The algorithm for finding Q from P is likely of independent interest. We call this problem the "Soft Convex Hull" problem and it is described in Section 6.1.

Do Approximate Optimization Oracles exist?

The answer to this question is a qualified Yes. They exist, but unfortunately, as we point out below, for many latent variable problems including the simple mixture of two Gaussians with means separated by $\Omega(1)$ standard deviations, we do not get $\varepsilon \in O_\delta(1/\sqrt{d})$, when, $k < d$. Thus, we do not satisfy the hypothesis of the results mentioned in Theorem 5,Theorem 6, and Theorem 9. But we are able to tackle this hurdle by projecting to the k-SVD subspace (of the input data points which satisfy conditions discussed below) where, we do get the necessary $\varepsilon \in O_\delta(1/\sqrt{k})$.

First we observe that approximate optimization oracles arise in a natural setting – that of latent variable models. [7] show that these models can be reduced to a geometric problem called LkP described below. [We will not reproduce the reduction here.] LkP is the following problem: Let K be a k vertex polytope in $\Re^d$. Let $M_{\cdot,1}, \ldots, M_{\cdot,k}$ denote the vertices of K. Assume that there are latent (hidden) points $P_{\cdot,j}, j = 1, 2, \ldots, n$, in K. The observed data points $A_{\cdot,j}, j = 1, 2, \ldots, n$ are generated (not necessarily under any stochastic assumptions) by adding *displacements* $A_{\cdot,j} - P_{\cdot,j}$ respectively to $P_{\cdot,j}$. Let[3]

$$\sigma_0 := \frac{||\mathbf{P} - \mathbf{A}||}{\sqrt{n}}.$$

We assume that there is a certain w_0 fraction of latent points close to every vertex of K, i.e., for all $\ell \in [k]$,

$$C_\ell := \{j : |P_{\cdot,j} - M_{\cdot,\ell}| \leq \frac{\sigma_0}{\sqrt{w_0}}\} \text{ satisfies } |C_\ell| \geq w_0 n.$$

▶ **Theorem 10** (From Data to Oracles). *Using the above notation, the following "Subset Smoothing algorithm" gives us a polynomial time* $\mathsf{OptOr}_{\frac{4\sigma_0}{\Delta\sqrt{w_0}}}(K)$ *oracle..*

> **Subset Smoothing Algorithm**
> Given query u, let S be the set of the $w_0 n$ j's with the highest $u \cdot A_{\cdot,j}$ values.
> Return $A_{\cdot,S} := \frac{1}{w_0 n} \sum_{j \in S} A_{\cdot,j}$.

The Subset Smoothing algorithm was used in [7]. It is also reminiscent of Super-quantiles [18], though our use here is not directly related to them. While this theorem helps us get optimization oracles, the error guarantee of $O(\sigma_0/\Delta\sqrt{w_0})$ is not good enough in many applications. An elementary example illustrates this issue:

Consider a mixture of two equal weight standard Gaussians centered at $-v$ and v, where, v is a vector of length 10. [This fits the paradigm "means separated by $\Omega(1)$ standard deviations".] Then, data generated by the mixture model fits our data generation process with $K = \{\lambda v, \lambda \in [-1, 1]\}$, and each $P_{\cdot,j}$ is either v or $-v$ depending on the Gaussian from which the point has been sampled. Here $A_{\cdot,j}$ denotes the actual sampled point from the mixture. Now, $\Delta = 20$ and it can be seen from Random Matrix Theorems (see e.g., [19]) that $\sigma_0 = O(1)$ with high probability. So, $\sigma_0/\Delta\sqrt{w_0} \in O(1)$ with high probability, and hence, the Theorem above yields an $\mathsf{OptOr}_\varepsilon(K)$ oracle with $\varepsilon \in \Omega(1)$. But d can be arbitrarily large and so we do not have the required $\mathsf{OptOr}_{O(1/\sqrt{d})}(K)$ oracle.

This elementary example can be tackled in several ways. Our algorithm which we call the "k-OLP algorithm" is simply stated and works in general settings (including on this toy example) for several Latent Variable problems (see Section 6 for details). The main idea is to first project the input points on a suitable SVD subspace and then use the approximate optimization oracle in the projection.

SVD and the k-OLP Algorithm

We now state the result (see Theorem 39 for a formal version) for k-OLP in the setting of LkP. As mentioned above, this uses SVD followed by subset smoothing.

[3] By the standard definition of spectral norm, it is easy to see that σ_0^2 is the maximum mean squared displacement in any direction.

▶ **Theorem 11.** *Recall the notation and assumptions of Theorem 10. In addition, we assume that each vertex of K is $\delta\Delta(K)$ far from the convex hull of other vertices of K, where, δ satisfies:*

$$\sigma_0 \leq c\delta^2\Delta\sqrt{w_0}/\sqrt{k}.$$

Then, the set of points P returned by the following k-OLP algorithm list learns the vertices of K. Further, we can find a subset Q of P with $|Q| = k$ and for each v, vertex of K, Q contains a v' with $|v - v'| \leq \delta\Delta/10$:

Algorithm k-OLP

1. Project to the k-dim SVD subspace V corresponding to the points $A_{\cdot,j}$.
2. Pick $m = \text{poly}_\delta(k)$ random vectors $u_1, u_2, \ldots, u_m$ in V.
3. For each u_i, take the mean of the $A_{\cdot,j}$ with the $w_0 n/2$ highest values of $u_i \cdot A_{\cdot,j}$.
4. Let P be the set of m means computed in the step above.
5. Output a subset Q of P, $|Q| = k$, using Theorem 9.

We sketch here the steps in the proof (the details are contained in the proof of Theorem 39.)

Sketch. Let $\widehat{K}$ denote projection of K onto V. By Theorem 10 , Step 3 of Algorithm k-OLP is an $\mathsf{OptOr}_\varepsilon(\widehat{K})$ oracle, where $\varepsilon = O\left(\frac{\sigma_0}{\Delta\sqrt{w_0}}\right)$. Also, each $\widehat{M}_{\cdot,\ell}$, which is the projection of $\widehat{M}_{\cdot,\ell}$ on V, is $O(\delta\Delta(K))$ far from the convex hull of the other vertices of $\widehat{K}$. Now, Theorem 9 applied to $\widehat{K}$ in the subspace V implies the desired result. ◀

It is worth noting that data obtained from several generative models are known to satisfy the LkP condition stated in Theorem 11, e.g., Stochastic Mixture models with k components, Topic Models, Mixed membership community models.

From List Learning to k-OLP

As outlined above, the k-OLP algorithm works in two stages: (i) Project the data points on the SVD subspace V of dimension k, and (ii) make polynomially calls to the $\mathsf{OptOr}_\varepsilon(K)$ oracle, where each query is given by a randomly chosen unit vector in the subspace V (as in the statement of Theorem 5) – let P be the set of points returned by the oracle. The first statement in Theorem 11 shows that the convex hull of P is close to K.

Obtaining approximations to the vertices of K from P requires addressing a new problem: given a set of points W, find a small subset T of W, such that their convex hulls are close. We call this the *soft convex hull* problem. A similar problem was addressed by [12]; however they gave a bi-criteria approximation algorithm for this problem. Under stronger assumptions, where we assume that there in the optimal solution $T^\star$, each point of $T^\star$ is well-separated from the convex hull of the rest of the points of $T^\star$, we show that one can recover approximations to each of the points in $T^\star$. Applying this result to the set of points P returned by the optimization oracle, we get a set of k points Q, each of which approximates a unique vertex of the polytope K.

The algorithm for obtaining soft convex hull proceeds as follows. We first prune points $w \in W$ which have the following property: consider the subset X of points in W which are sufficiently far from w. Then w is close the convex hull of X. After pruning such points from W, we pick a subset of points which are sufficiently far-apart from each other. The main technical result shows that this procedure outputs the desired set T.

1.3 Related Work

The well known result of [15] shows that given a set of n points in $\Re^d$, projection to a random subspace of dimension $O(\log n/\varepsilon^2)$ preserves all pair-wise distances up to $(1+\varepsilon)$-factor with high probability. Further, this bound on the dimension on which the points are projected is known to be tight [2, 16]. Note that in our setting, there are $k+1$ points "of interest", namely, the k vertices of K and a point $a \notin K$ and by the above, a random projection to $O^*(\log k)$ dimensional space preserves all pairwise distances among them. But this is not sufficient for our problems. We need separation of a not just from the vertices of K, but from all of K in the projection. We achieve this by projecting to a set of random 1-dimensional subspaces, and show that the distance between a point and a polytope does not scale down by more than $O(\sqrt{d})$ factor for at least one of them with high probability (this is an immediate corollary of the RSH Theorem).

The problem of learning vertices of a polytope arises in many settings where data is assumed to be generated by a stochastic process parameterized by a model. Examples include topic models [10], stochastic block models [1], latent Dirichlet allocation [11]. A variety of techniques have been developed for these specific problems (see e.g. [3, 4, 14]). [7] (see also [5]) proposed the latent k-polytope (LkP) model which seeks to unify all of these latent variable models. In this model, there is *latent* polytope with k vertices, and data is generated in a two step process: first we pick *latent* points from this polytope, and then the observed points are obtained by perturbing these latent points in an adversarial manner. They showed that under suitable assumptions on this deterministic setting, one can capture the above-mentioned latent variable problems. Assuming strong separability conditions on the vertices of the polytope (i.e., each vertex of K is far from the *affine* hull of other vertices of K), they showed that one can efficiently recover good approximations to the vertices of the polytope from the input data points. In comparison, our assumption on K is that each vertex of K is far from the convex hull of the remaining vertices of K. This is a much milder condition, e.g., it allows a polytope with more than 2 vertices in a plane. [8] showed how to infer the parameter k from data in the LkP setting (under the strong separation condition).

[12] addressed a problem similar to the Hausdorff problem: instead of an ε-optimization oracle for a polytope K, we are given an explicit set P of points, whose convex hull is within Hausdorff distance at most $\delta\Delta(K)$ from K. They are able to get better dependencies on the parameters k, δ, ε than Theorem 21 under these stronger assumptions.

2 Preliminaries

For two points $x, y \in \Re^d$, $|x-y|$ denotes the Euclidean distance between the points. Given a point $x \in \Re^d$ and a subset $X \subseteq \Re^d$, define $\mathsf{dist}(x, X)$ as the minimum distance between x and a point in X, i.e., $\inf_{y\in X}|x-y|$. For a set of points X, $\Delta(X)$ denotes the diameter of X, i.e., $\sup_{x,y\in X}|x-y|$. We denote the convex hull of X by $\mathsf{CH}(X)$. For two subsets A, B of $\Re^d$, define their Minkowski sum $A+B$ as $\{x+y : x \in A, y \in B\}$. Similarly, define λA, where $\lambda \in \Re$, as $\{\lambda x : x \in A\}$. For an $m \times n$ matrix B, we use $B_{\cdot,j}$ to denote the j^{th} column of B. For a subset $S \subseteq [n]$ of columns of B, $B_{\cdot,S}$ denotes $\frac{1}{|S|}\sum_{j\in S} B_{\cdot,j}$. Often, we represent the vertices of a polytope K in $\Re^d$ by a $d \times k$ matrix M, and so the columns $M_{\cdot,1}, \ldots, M_{\cdot,k}$ would represent the vertices of K. We shall use the notation $\mathrm{poly}_\delta(z)$ to denote a quantity which is $z^{\mathrm{poly}(1/\delta)}$. Further the notation $O_\delta(z)$ shall denote a quantity which is $f(\delta)z$, where $f(\delta)$ is a function depending on δ only (and hence, is constant if δ is constant).

We now give an outline of rest of the paper. In Section 3, we prove the Random Separating Hyperplane theorem. In Section 4, we prove Theorem 5 by showing that an $\mathsf{OptOr}_\varepsilon(K)$ oracle leads to efficient constructions of approximation to K. In Section 5 we give an algorithm

for the ListLearn problem under the stronger assumption that the vertices of K are well separated. We also prove the lower bound result Theorem 7 in this section. In Section 6, we extend the algorithm for ListLearn to the k-OLP problem. This requires the concept of soft convex hulls. The algorithm for constructing soft convex hulls in given in Section 6.1. Finally, in Section 7, we apply the k-OLP algorithm for the latent polytope problem. As note earlier, in the setting of latent polytopes K, we can only guarantee $\mathsf{OptOr}_\varepsilon(K)$ oracles with ε being $O(1/\sqrt{k})$, whereas our algorithm for k-OLP requires ε to be $O(1/\sqrt{d})$. We handle this issue by projecting to a suitable SVD subspace and executing the k-OLP algorithm in this subspace. We conclude with some open problems in Section 8.

3 Random Separating Hyperplane (RSH) Theorem

In this section, we prove RSH for polytopes: If a point p point is at a distance from a polytope K, then, the Gaussian measure of the set of *well-separating* hyperplanes has a positive lower bound depending on the number of vertices in K. More specifically, we show:

▶ **Theorem 12.** *Suppose K is a polytope in $\Re^d$ with k vertices and diameter $\Delta(K)$. Suppose a is a point in $\mathbf{R}^d$ and $\delta \in (0,1]$ with*

$$\min_{y\in K} |a-y| \geq \delta\Delta(K). \tag{3}$$

Let V be an m-dimensional subspace containing $\mathsf{Span}(K \cup \{a\})$ and let u be a random vector drawn from the normal distribution $N(0, I_m)$ in V. Then,

$$\mathbf{Pr}_u\left[(u\cdot a - \max_{y\in K} u\cdot y) \geq |u|\cdot\delta\Delta(K)\cdot\frac{\sqrt{\log k}}{3\sqrt{\log k}+4\delta\sqrt{m}}\right] \geq \frac{1}{40}k^{-10/\delta^2}.$$

Proof Overview

We give an informal overview of the proof. A naive, though incorrect, idea would be the following: let ζ be the closest vertex of K to a. We can argue that with reasonable probability (i.e., $\Omega(k^{-1/\delta^2})$) that $|(\zeta-a)\cdot u|$ is at least $\frac{2\sqrt{\log k}\cdot|\zeta-a|}{\delta\sqrt{m}} \geq \frac{2\sqrt{\log k}\cdot\Delta(K)}{\sqrt{m}}$. Call this event $\mathcal{E}_1$. Now consider a line segment joining two distinct vertices, say ζ_i, ζ_j, of K. The length of such a line segment is at most $\Delta(K)$, and hence there is high probability that $|(\zeta_i-\zeta_j)\cdot u| \leq \frac{1.5\sqrt{\log k}\cdot\Delta(K)}{\sqrt{m}}$. In fact, we can apply union bound, and show that this property holds for all pairs of vertices of K – call this event $\mathcal{E}_2$. If both $\mathcal{E}_1$ and $\mathcal{E}_2$ occur, then it is easy to see that along the direction u, a is separated from all the vertices of K. However, the events $\mathcal{E}_1$ and $\mathcal{E}_2$ are not independent, and hence, we cannot argue that both events will occur with non-trivial probability. Here one could use the union bound, but a calculation shows that for this we need $\delta > 1$. But $\delta < 1$ in our applications, so we cannot use this approach.

Instead, we rely on the following more nuanced idea. Let w be the unit direction along the line joining a and the closest point b of K. As argued above, we can show that $|(b-a)\cdot u|$ is at least $\frac{\Omega(\sqrt{\log k}\cdot|b-a|}{\delta\sqrt{m}} \geq \frac{\Omega(\sqrt{\log k})\cdot\Delta(K)}{\sqrt{m}}$ – this only requires arguing about the component of u along w; let $\mathcal{E}_1'$ denote this event. Now we show that the following event, say $\mathcal{E}_2'$, occurs with high probability: along the component of u orthogonal to w, the projections of the line segments joining any two vertices of K is at most $\frac{O(\sqrt{\log k})\Delta(K)}{\sqrt{m}}$. Since $\mathcal{E}_1'$ and $\mathcal{E}_2'$ are concerned with orthogonal directions, they are independent using properties of the Gaussian. Thus we can show that both of these events happen with non-trivial probability. This suffices

to show the desired result. We now give the formal proof of this result. First, we state the following well-known concentration bound for Gaussian random variables (see e.g. [19]):

▶ **Theorem 13** (Gaussian Concentration Bounds). *Let X be an $N(0,1)$ Gaussian random variable, i.e., with mean 0 and variance 1. Then, for any $t > 0$,*

$$e^{-t^2/4}/2 \leq \Pr[|X| \geq t] \leq 2e^{-t^2}.$$

We shall also need the following result on the sum of squares of i.i.d. normal random variables.

▶ **Theorem 14** ([17]). *Let $X_1, \ldots, X_n$ be n i.i.d. $N(0,1)$ Gaussian random variables. Then,*

$$\Pr[\sum_{i=1}^{n} X_i^2 > 4n] \leq e^{-\sqrt{n}}.$$

We now proceed with the proof of Theorem 12.

Proof of Theorem 12. Let $\zeta_1, \zeta_2, \ldots, \zeta_k$ be the k vertices of K. Let b be the closest point in K to a, and define

$$w = \frac{a-b}{|a-b|}.$$

Then by standard results on separation between a point and a closed convex set (see e.g. [13]), we know that for all points $y \in K$:

$$w \cdot y \leq w \cdot b. \tag{4}$$

Now, we extend w to an orthonormal basis $w_1 := w, w_2, \ldots, w_m$ of the subspace V. The random vector u can be generated as follows: sample m i.i.d. $N(0,1)$ random variables $\lambda_1, \ldots, \lambda_m$, and define $u := \sum_{i=1}^{m} \lambda_i w_i$. We first observe that it is enough to argue that the projection of a along u is well separated from that for each vertex ζ_ℓ of K. Indeed, we can express $(u \cdot a - \max_{y \in K} u \cdot y)$ as

$$u \cdot (a-b) - \max_{y \in K} u \cdot (y-b) = u \cdot (a-b) - \max_{\ell=1,\ldots,k} u \cdot (\zeta_\ell - b) \tag{5}$$

It suffices to show that with reasonably high probability, there is a lower bound on $u \cdot (a-b)$ and an upper bound on $u \cdot (\zeta_\ell - b)$ for all ℓ. We proceed with the latter goal first. Observe that along the direction w, K lies *below* b. Thus, it suffices to argue that the projection of $(\zeta_\ell - b)$ along the direction orthogonal to w is not large for any vertex ζ_ℓ of K. Let z denote $\lambda_2 w_2 + \ldots + \lambda_m w_m$. Then u can be expressed as $\lambda_1 w + z$. We now define events upper bounding $z \cdot (\zeta_\ell - b)$ for all vertices ζ_ℓ of K.

▶ **Definition 15.** *For $\ell = 1, 2, \ldots, k$, define the event $\mathcal{E}_\ell$ as $[|z \cdot (\zeta_\ell - b)| \leq 2\sqrt{\ln k}\Delta(K)]$.*

▶ **Proposition 16.** *Suppose the events $\mathcal{E}_\ell$ occur for all $\ell = 1, \ldots, k$. Then, for any $\ell \in [k]$,*

$$u \cdot (\zeta_\ell - b) \leq 2\sqrt{\ln k}\Delta(K).$$

Proof. Observe that

$$u \cdot (\zeta_\ell - b) = \lambda_1 w \cdot (\zeta_\ell - b) + z \cdot (\zeta_\ell - b) \overset{(4)}{\leq} z \cdot (\zeta_\ell - b).$$

The desired result now follows from the definition of the event $\mathcal{E}_\ell$. ◀

We now define events to lower bound $u \cdot (a-b)$. Observe that $(a-b)$ is along w, and hence, is orthogonal to z. Therefore $u \cdot (a-b) = \lambda_1 w \dot{(}a-b) = \lambda_1 |a-b|$. Therefore, we define an event that lower bounds λ_1. We also define an event that upper bounds $|z|$.

▶ **Definition 17.** *Define $\mathcal{E}_0$ as the event $[|z| \leq 4\sqrt{m}]$. Also define $\mathcal{E}_{k+1}$ as the event $[\lambda \geq 3\sqrt{\ln k}/\delta]$.*

▶ **Proposition 18.** *Suppose the events $\mathcal{E}_0, \ldots, \mathcal{E}_{k+1}$ happen. Then,*

$$u \cdot a - \max_{y \in K} u \cdot y \geq \frac{\lambda \delta \Delta(K)|u|}{3(\lambda + 4\sqrt{m})} \geq \frac{\delta\sqrt{\ln k}\Delta(K)|u|}{3\sqrt{\ln k} + 4\delta\sqrt{m}}.$$

Proof. As observed above, $u \cdot (a-b) = \lambda_1 |a-b|$. Now observe that for any vertex ζ_ℓ of K, Proposition 16 shows that

$$u \cdot (\zeta_\ell - b) \leq 2\sqrt{\ln k}\Delta(K) \overset{(3)}{\leq} 2\sqrt{\ln k}|a-b|/\delta \leq 2\lambda_1|a-b|/3,$$

where the last inequality follows from the definition of of event $\mathcal{E}_{k+1}$. Thus, we see from the above two inequalities and (5) that

$$u \cdot a - \max_{y \in K} u \cdot y \geq \lambda_1 |a-b|/3 \overset{(3)}{\geq} \lambda_1 \delta |\Delta(K)|/3. \tag{6}$$

It remains to upper bound $|u|$. Recall that $u = \lambda_1 w + z$, and hence, $|u| \leq |\lambda_1| + |z| \leq \lambda_1 + 4\sqrt{m}$, where we have used the definition of the event $\mathcal{E}_{k+1}$ and the fact that $\lambda_1 > 0$ because of event $\mathcal{E}_0$. Using this in (6) yields the first inequality in the desired claim. The second desired inequality follows from the fact that $\lambda_1 \delta \geq 3\sqrt{\ln k}$ (definition of event $\mathcal{E}_{k+1}$). ◀

It remains to show that the events $\mathcal{E}_0, \ldots, \mathcal{E}_{k+1}$ occur with reasonable probability.

▶ **Proposition 19.** $\mathbf{Pr}\left[\wedge_{\ell=0}^{k+1} \mathcal{E}_\ell\right] \geq \frac{1}{40} k^{-10/\delta^2}$.

Proof. Consider any $\ell \in [k]$. Observe that $z \cdot (\zeta_\ell - b)$ is a 1-dimensional Gaussian random variable with zero mean and with variance $|\zeta_\ell - b|^2 \leq \Delta(K)^2$. Thus, $\frac{z \cdot (\zeta_\ell - b)}{\Delta(K)}$ is an $N(0,1)$ normal random variable. Therefore, Theorem 13 shows that

$$\Pr[\neg \mathcal{E}_\ell] = \Pr\left[\frac{z \cdot (\zeta_\ell - b)}{\Delta(K)} \geq 2\sqrt{\ln k}\right] \leq \frac{1}{2k^2}.$$

By definition of z, $|z|^2 = \lambda_2 + \ldots + \lambda_m^2$. By Theorem 14,

$$\Pr[\neg \mathcal{E}_0] = \Pr[|z|^2 > 16m] \leq e^{-\sqrt{m}} \leq 1/20,$$

for large enough m. Applying union bound to the above two inequalities, we get

$$\mathbf{Pr}\left(\wedge_{\ell=0}^{k} \mathcal{E}_\ell\right) \geq 1 - \sum_{\ell=0}^{k} \mathbf{Pr}(\neg \mathcal{E}_\ell) \geq 1/4 \tag{7}$$

Finally we consider event $\mathcal{E}_{k+1}$. Observe that λ_1 is $N(0,1)$ Gaussian random variable. Applying Theorem 13 to λ_1, we see that

$$\Pr[\mathcal{E}_{k+1}] = \Pr\left[\lambda_1 \geq 3\sqrt{\ln k}/\delta\right] \geq \frac{1}{10} k^{-10/\delta^2}. \tag{8}$$

Observe that the event $\wedge_{\ell=0}^{k} \mathcal{E}_\ell$ depends on the random variables $\lambda_2, \ldots, \lambda_k$, and hence, is independent of the event $\mathcal{E}_{k+1}$. The desired result now follows from the (7) and (8). ◀

The theorem now follows from Proposition 18 and Proposition 19. ◀

4 From $\mathsf{OptOr}_\varepsilon(K)$ oracles to the Hausdorff Problem

We prove Theorem 5 in this section. We begin by defining Hausdorff distance formally.

▶ **Definition 20.** *The* Hausdorff-distance, $\mathsf{Haus}(K, K')$, *between two polytopes* K *and* K' *is the infimum over all values* α *such that the following condition is satisfied: for every point* $x \in K$, *there is a point* $y \in K'$ *such that* $|x - y| \leq \alpha$, *and vice versa.*

We now give the formal version of Theorem 5:

▶ **Theorem 21.** *Suppose* ε, δ *are reals in* $[0, 1]$ *with*

$$\delta > c\varepsilon\sqrt{d}, \delta > c/\sqrt{d}, \tag{9}$$

where c *is a large enough constant. Let* K *be a* k*-vertex polytope with* $\Delta(K)$ *denoting the diameter of* K. *Suppose we are also given an* $\mathsf{OptOr}_\varepsilon(K)$ *oracle* $\mathcal{O}$. *Let* P *be the set of answers from the oracle on* $m := k^{10+c\delta^{-2}}$ *independent random queries. Then,* $\mathsf{Haus}(K, \mathsf{CH}(P)) \leq \delta \cdot \Delta(K)$.

Before giving the proof, we highlight the main ideas here. Let the random queries correspond to unit directions $u^1, \ldots, u^m$, and for each of these queries u^j, the oracle $\mathcal{O}$ returns a near-optimal point $x(u^j)$. Let P^j denote the subset $\{x(u^1), \ldots, x(u^j)\}$. Let v be a vertex of K. The non-triviality in the proof lies in showing that there is a point in $\mathsf{CH}(P)$ close to v. Suppose there is no such point in $\mathsf{CH}(P^j)$, and let b the closest point in $\mathsf{CH}(P^j)$ to v. Then we show that with constant probability the following events happen: (i) the unit vector u^{j+1} has a non-trivial component along the direction joining b and v, and (ii) its component along the orthogonal direction is not too large. If both of these events happen, we show that $\mathsf{dist}(v, \mathsf{CH}(P^{j+1})) \leq (1 - O(\delta^2))\mathsf{dist}(v, \mathsf{CH}(P^j))$. We now give details of the proof.

Proof. We describe the algorithm for obtaining the desired set S (referred as **Random Probes Algorithm** in Section 1.2) in Algorithm 1. The set P is constructed as follows: we pick a set of m random unit vectors. For each such unit vector u, we add the corresponding point $x(u)$ returned by the oracle $\mathcal{O}$ to the set P.

Algorithm 1 Algorithm for finding the set P such that $\mathsf{CH}(P)$ approximates K.

Input: An $\mathsf{OptOr}_\varepsilon(K)$ oracle $\mathcal{O}$.
Initialize a set P to $\varnothing$
Repeat m times:
  Let u be a random unit vector in $\Re^d$.
  Call $\mathcal{O}$ on u to get a vector $x(u)$.
  Add $x(u)$ to P.
Output P.

One side of the desired result is easy to show:

▷ Claim 22. For each $x \in \mathsf{CH}(P)$, there is a $y \in K$ such that $|x - y| \leq \delta\Delta(K)$.

Proof. For a point $x(u) \in P$, we know by the definition of $\mathsf{OptOr}_\varepsilon(K)$ oracle that there is a point $y \in K$ such that $|x(u) - y| \leq \varepsilon\Delta(K) \leq \delta\Delta(K)$, where $\varepsilon \leq \delta$ follows from (9). The desired result now follows from the convexity of K. ◁

It remains to show that for any vertex v of K, there is a point $y \in \mathsf{CH}(P)$ such that $|v - y| \leq \delta\Delta(K)$. Fix such a vertex v of K for rest of the discussion. Let the random unit vectors considered in Algorithm 1 (in the order they get generated) be $u^1, \ldots, u^m$. Let P^j denote the subset $\{x(u^1), \ldots, x(u^j)\}$ of P. Define an event $\mathcal{E}_j$ as follows:

$$\mathsf{dist}(v, \mathsf{CH}(P^j)) \leq \delta \cdot \Delta(K) \quad \text{or} \quad \mathsf{dist}(v, \mathsf{CH}(P^{j+1})) \leq \left(1 - \frac{\delta^2}{c'}\right) \mathsf{dist}(v, \mathsf{CH}(P^j)),$$

where c' is a large enough constant. Our main technical result is to show that conditioned on *any* choice of $u^1, \ldots, u^j$ the event $\mathcal{E}_j$ happens with reasonably high probability (where the probability is over the choice of u^{j+1}):

▶ **Lemma 23.** *For any index $j \in [m-1]$,*

$$\Pr_{u^{j+1}} \left[\mathcal{E}_j | u^1, \ldots, u^j\right] \geq \frac{1}{100}.$$

Proof. Fix the vectors $u^1, \ldots, u^j$. If $\mathsf{dist}(v, \mathsf{CH}(P^j)) \leq \delta \cdot \Delta(K)$, then we are done. So assume this is not the case. Let b be the closest point in $\mathsf{CH}(P^j)$ to v. Thus,

$$|v - b| \geq \delta \cdot \Delta(K). \tag{10}$$

Define w as

$$w := \frac{v - b}{|v - b|}.$$

We can now express the vector u^{j+1} as $\lambda w + z$, where $\langle z, w \rangle = 0$. We first show the following useful properties of these vectors.

▷ Claim 24. With probability at least $\frac{1}{100}$, the following three events happen:

$$|z| \leq 4\sqrt{d} \tag{11}$$

$$\max_{y \in K} |z \cdot (v - y)| \leq 2\sqrt{\ln k}\Delta(K) \tag{12}$$

$$\lambda \geq \frac{100}{\delta}\sqrt{\ln k} \tag{13}$$

Proof. The proofs of these three inequalities are same as the arguments in Proposition 19 (in order to prove (12), it suffices to show it for points y which are vertices of K). ◁

The following fact is also easy to show:

▶ **Proposition 25.**

$$|v - x(u^{j+1})| \leq 2\Delta(K).$$

Proof. By the definition of $\mathsf{OptOr}_\varepsilon(K)$ oracle, there is a point $p \in K$ such that $|p - x(u^{j+1})| \leq \delta \cdot \Delta(K) \leq \Delta(K)$. The desired result now follows by triangle inequality. ◀

Let δ_1 denote $\frac{\delta^2}{100}$. Let b_1 denote the vector

$$\delta_1 x(u^{j+1}) + (1 - \delta_1)b.$$

Since $b_1 \in \mathsf{CH}(P^{j+1})$, the desired result will follow if we prove the following:

$$|v - b_1|^2 \leq \left(1 - \frac{\delta^2}{100}\right) |v - b|^2. \tag{14}$$

Now,

$$
\begin{aligned}
|v-b_1|^2 &= \delta_1^2|v-x(u^{j+1})|^2 + (1-\delta_1)^2|v-b|^2 + 2\delta_1(1-\delta_1)(v-x(u^{j+1}))\cdot(v-b)\\
&\overset{\text{Prop. 25}}{\leq} 4\delta_1^2\Delta(K)^2 + (1-2\delta_1)|v-b|^2 + \delta_1^2\Delta(K)^2 + 2\delta_1(1-\delta_1)(v-x(u^{j+1}))\cdot(v-b)\\
&\overset{(10)}{\leq} \left(1-\frac{3\delta_1}{2}\right)|v-b|^2 + +2\delta_1(1-\delta_1)|v-b|\cdot(v-x(u^{j+1}))\cdot w\\
&= \left(1-\frac{3\delta_1}{2}\right)|v-b|^2 + +2\delta_1(1-\delta_1)\Bigg(\underbrace{\frac{|v-b|}{\lambda}\cdot(v-x(u^{j+1}))\cdot u^{j+1}}_{:=A}\\
&\qquad \underbrace{-\frac{|v-b|}{\lambda}\cdot(v-x(u^{j+1}))\cdot z}_{:=B}\Bigg) \qquad (15)
\end{aligned}
$$

We now bound each of the terms A and B above. Now,

$$A \leq \frac{|v-b|}{\lambda}\varepsilon\Delta(K) \overset{(9)}{\leq} \frac{|v-b|\delta\Delta(K)}{c\lambda} \overset{(10)}{\leq} \frac{|v-b|^2}{c\lambda} \overset{(13)}{\leq} \frac{|v-b|^2\delta}{c},$$

where the first inequality follows from the definition of $\mathsf{OptOr}_\varepsilon(K)$ oracle. We now bound the quantity B. Let y be the point in K closest to $x(u^{j+1})$. We know that $|y-x(u^{j+1})| \leq \varepsilon\Delta(K)$. Therefore,

$$
\begin{aligned}
B &\leq \frac{|v-b|}{\lambda}\left(|(v-y)\cdot z| + |z|\varepsilon\Delta(K)\right)\\
&\overset{(11),(12)}{\leq} \frac{|v-b|}{\lambda}\left(2\sqrt{\ln k}\Delta(K) + 4\varepsilon\sqrt{d}\Delta(K)\right)\\
&\overset{(9),(13)}{\leq} \frac{\delta|v-b|\Delta(K)}{10} \overset{(10)}{\leq} \frac{|v-b|^2}{10}.
\end{aligned}
$$

Substituting the above bound on A and B in (15) yields the desired result. ◀

We are now almost done. As the following result shows, it suffices to argue that enough number of events $\mathcal{E}_j$ happen:

▷ Claim 26. If at least $\frac{c'}{\delta^2}\ln(2/\varepsilon)$ of the events $\mathcal{E}_j, j \in [m-1]$ happen, then $\mathsf{dist}(v, \mathsf{CH}(P)) \leq \delta\Delta(K)$.

Proof. Assume, for the sake of contradiction, that $\mathsf{dist}(v, \mathsf{CH}(P)) > \varepsilon\Delta(K)$. Assume that events $\mathcal{E}_{j_1}, \ldots, \mathcal{E}_{j_h}$ happen, where $h := \frac{c'}{\delta^2}\ln(2/\varepsilon)$. Now, for any index $i \in [h-1]$, the definition of $\mathcal{E}_{j_{i+1}}$ implies that

$$\mathsf{dist}(v, \mathsf{CH}(P^{j_{i+1}})) \leq \left(1-\frac{\delta^2}{c'}\right)\mathsf{dist}(v, \mathsf{CH}(P^{j_{i+1}-1}) \leq \mathsf{dist}(v, \mathsf{CH}(P^{j_i})).$$

Therefore,

$$\mathsf{dist}(v, \mathsf{CH}(P^{j_h})) \leq \left(1-\frac{\delta^2}{c'}\right)^h \mathsf{dist}(v, P^1) \overset{\text{Proposition 25}}{\leq} \left(1-\frac{\delta^2}{c'}\right)^h 2\Delta(K) \leq \delta\Delta(K). \quad \triangleleft$$

It remains to show that with high probability at least $h := \frac{c'}{\delta^2}\ln(2/\varepsilon)$ of the events happen. In order to prove this, we divide the sequence $[m]$ into $[h]$ subsequences, each of length m/h. Call these subsequences $C_1, \ldots, C_h$. It follows from Lemma 23 that for any $i \in [h]$,

$$\Pr\left[\wedge_{j\in C_i}\neg\mathcal{E}_j\right] \leq 0.99^{m/h} \leq \frac{1}{h^2}.$$

A simple union bound now shows that with probability at least $1 - 1/h$, at least one event $\mathcal{E}_j$ happens during each of the subsequences $C_1, \ldots, C_h$. Claim 26 now proves the theorem. ◀

5 From $\mathsf{OptOr}_\varepsilon(K)$ oracles to ListLearn

We first define the notion of well-separatedness.

▶ **Definition 27.** *We say that a polytope K with vertex set V is* δ-well-separated *if for every vertex $v \in V$, we have* $\mathsf{dist}(v, \mathsf{CH}(V \setminus \{v\})) \geq \delta \cdot \Delta(K)$.

Recall that Theorem 7 states that given access to only an $\mathsf{OptOr}_\varepsilon(K)$ oracle, ε must be $\Omega(\ln d/\sqrt{d})$ for any deterministic algorithm to output a point within $\Delta(K)/10$ of K. We show this result in the full version [9]. We now prove Theorem 6, which gives such an algorithm for the ListLearn problem.

▶ **Theorem 28.** *Suppose ε, δ are reals in $[0, 1]$ with $\delta^2 \geq c\varepsilon\sqrt{d}, \delta^3 \geq c\,\varepsilon$, where c is a large enough constant. Let K be a δ-well-separated k-vertex polytope. Suppose we are also given $\mathsf{OptOr}_\varepsilon(K)$ oracle $\mathcal{O}$. Let W be the set of answers of the oracle to $m = poly(d) \cdot k^{\Omega(1/\delta^2)}$ independent random queries. Then for each vertex v of K, there is a point $v' \in W$ such that $|v - v'| \leq O(\delta^2\Delta(K)/c)$.*

Proof. The algorithm chooses a set U of $k^{\Omega(1/\delta^2)}$ unit length i.i.d. Gaussian vectors. For each $u \in U$, it calls the oracle $\mathcal{O}$ to find a vector $x(u)$. Let W denote the set $\{x(u) : u \in U\}$. We first show that for every vertex of K, there is a direction u in U along which the projection of this vertex is higher than the projection of the remaining vertices by a large enough margin. Let $M_{\cdot,1}, \ldots, M_{\cdot,k}$ be the vertices of K. The proof of the following result is deferred to the full version [9].

▷ Claim 29. With high probability, the following event happens: for each $\ell \in [k]$, there is a vector $u^{(\ell)} \in U$ such that for all $\ell' \in [k], \ell' \neq \ell$, we have

$$u^{(\ell)} \cdot M_{\cdot,\ell} > u^{(\ell)} \cdot M_{\cdot,\ell'} + \frac{c\,\varepsilon \cdot \Delta(K)}{8\delta^2} \tag{16}$$

For rest of the proof, assume that the statement in Claim 29 holds true, i.e., there are directions $u^{(1)}, \ldots, u^{(k)} \in U$ satisfying (16). We now show that for every vertex of $M_{\cdot,\ell}$ of K, the corresponding point $x(u^\ell)$ is close to $M_{\cdot,\ell}$.

▷ Claim 30. For every $\ell \in [k]$, $|x(u^{(\ell)}) - M_{\cdot,\ell}| \leq 17\delta^2\Delta(K)/c$.

Proof. By the definition of $\mathcal{O}$, we know that $x(u^{(\ell)})$ can be written as $y(u^\ell) + z(u^{(\ell)})$, where $y(u^{(\ell)}) \in K$ and $|z(u^{(\ell)})| \leq \varepsilon\Delta(K)$. Thus, there is a convex combination $\lambda_{\ell'}, \ell' \in [k]$, of the vertices $M_{\cdot,\ell'}$ of K such that $x(u^{(\ell)}) = \sum_{\ell'\in[k]} \lambda_{\ell'} M_{\cdot,\ell'} + z(u^{(\ell)})$.

By the definition of $\mathcal{O}$, $x(u^{(\ell)}) \cdot u^{(\ell)} \geq M_{\cdot,\ell} \cdot u^{(\ell)} - \varepsilon\Delta(K)$ and $|z(u^{(\ell)}) \cdot u^{(\ell)}| \leq \varepsilon\Delta(K)$. So, we get

$$M_{\cdot,\ell} \cdot u^{(\ell)} - \varepsilon\Delta(K) \leq \sum_{\ell'\in[k]} \lambda_{\ell'} M_{\cdot,\ell'} \cdot u^{(\ell')} + \varepsilon\Delta(K),$$

which implies (after subtracting $\lambda_\ell M_{\cdot,\ell} \cdot u^{(\ell)}$ from both sides):

$$(1-\lambda_\ell)M_{\cdot,\ell}u^{(\ell)} \leq \sum_{\ell' \neq \ell} \lambda_{\ell'} M_{\cdot,\ell'} \cdot u^{(\ell')} + 2\varepsilon\Delta(K)$$

which, using Claim 29, yields:

$$(1-\lambda_\ell)M_{\cdot,\ell}u^{(\ell)} \leq (1-\lambda_\ell)M_{\cdot,\ell}u^{(\ell)} - (1-\lambda_\ell)\frac{c\varepsilon\Delta(K)}{8\delta^2} + 2\varepsilon\Delta(K).$$

It follows from the above inequality that $1-\lambda_\ell \leq \frac{16\delta^2}{c}$. Therefore,

$$|x(u^{(\ell)}) - M_{\cdot,\ell}| = \left|\sum_{\ell' \neq \ell} \lambda_{\ell'}(M_{\cdot,\ell} - M_{\cdot,\ell'})\right| + \varepsilon\Delta(K) \tag{17}$$

$$\leq \sum_{\ell' \neq \ell} \lambda_{\ell'}\Delta(K) + \varepsilon\Delta(K) \leq \frac{17\delta^2\Delta(K)}{c}.$$ ◁

This completes proof of the Theorem. ◀

6 From ListLearn to the k-OLP Problem

In this section, we show that for well-separated polytopes, a solution for the ListLearn problem can be used to solve the k-OLP problem as well. This algorithm uses the notion of soft convex hulls. We first describe the algorithm for constructing soft convex hulls, and then use it to solve the k-OLP problem.

6.1 Soft Convex Hulls

Let W be a finite set of points in $\Re^d$, and T be the vertices of $\mathsf{CH}(W)$. The subset T of W is the unique subset of W with the following properties:
(P1) $W \subseteq \mathsf{CH}(T)$
(P2) $\forall w \in W$, if $w \notin \mathsf{CH}(W \setminus \{w\})$, then $w \in T$.

We now define a natural notion of *soft* convex hull.

▶ **Definition 31.** *For an $\varepsilon \geq 0$, and $S \subseteq W$, define the ε-convex hull of S, ε-$\mathsf{CH}(S)$, as $\mathsf{CH}(S) + \varepsilon\Delta(W)B$, where B is the unit ball of the Euclidean norm.*

The intuition behind the above definition is that $\mathsf{CH}(W)$ can have many vertices, but there may be a small set of points whose soft convex hull contains W. This is defined more formally as follows:

▶ **Definition 32.** *We call a subset $T \subseteq W$ an ε-envelope of W, written ε-$\mathsf{ENV}(W)$, if $W \subseteq \varepsilon$-$\mathsf{CH}(T)$.*

Remarks. The following observations about the set ε-$\mathsf{ENV}(W)$ are easy to see:
(a) There are several distinct sets T which could qualify as ε-$\mathsf{ENV}(W)$. For example, let W consist of the following set of points in $\Re^2$: a set of points W_1 close to $(0,0)$ and a set of points W_2 close to $(1,0)$. Let T be a pair of points $\{x,y\}$ with $x \in W_1, y \in W_2$. Then it is easy to check that T is an ε-$\mathsf{ENV}(W)$; and (b) Let T be ε-$\mathsf{ENV}(W)$. Unlike property (P2) above, it is not necessary that if $w \in W$ is such that $w \notin \varepsilon$-$\mathsf{CH}(W \setminus \{w\})$, then $w \in T$.

Since the set ε-$\mathsf{ENV}(W)$ is not uniquely determined, we will impose one more condition on it to make it unique (if it exists) and polynomial time computable. This condition requires the points of T to be "far apart" from each other. More precisely:

▶ **Definition 33.** *For $\varepsilon, \delta \in [0,1]$, a set T is called a (ε,δ)-$\mathsf{ENV}(W)$ if it is an ε-$\mathsf{ENV}(W)$ and*

$$\forall w \in T, \mathsf{dist}(w, \mathsf{CH}(T \setminus \{w\})) > \delta\Delta(W) \tag{18}$$

The proof of the following result follows from standard arguments and is deferred to the appendix.

Given a subset T of W, we can check in polynomial time whether T is an (ε,δ)-$\mathsf{ENV}(W)$.

For rest of the section, we address the following question: given the set W, and parameters ε, δ, is there a (ε,δ)-$\mathsf{ENV}(W)$, and if so, can we find in polynomial time an approximation to this set? In the full version [9], we argue that several "natural" greedy strategies do not work. If $\varepsilon = 0$, the above question is easy to answer in polynomial time. The answer is yes iff the set T of vertices of $\mathsf{CH}(W)$ satisfies (18). Also, if $\delta = 1$, T has to be a singleton to satisfy (18).

In rest of this section, we consider the following problenm: For what pairs of values of ε, δ, can we prove that there is *essentially* at most one (ε,δ)-$\mathsf{ENV}(W)$, and if so, can we determine this set efficiently? We do not know the exact answer to this, but our main result here (which suffices for the applications) is (verbally stated) an affirmative answer to the question if the following condition is satisfied: $\delta \in \Omega(\sqrt{\varepsilon})$. This will follow as a corollary of our main result:

▶ **Theorem 34.** *Let $\delta, \varepsilon, \varepsilon_3$ be reals in $(0, 1/8)$ satisfying*

$$\delta > \max\left(\frac{2\varepsilon}{\varepsilon_3 - \varepsilon}, 4\varepsilon_3\right) \tag{19}$$

Let W be a finite set of points in $\Re^d$. We can determine in polynomial time whether exists a set T in (ε,δ)-$\mathsf{ENV}(W)$, and if so, we can efficiently find a subset Q of W such that

$$|Q| = |T| \tag{20}$$

$$\forall w \in T, \exists x \in Q : |w - x| \leq 2\varepsilon_3\Delta(W) \tag{21}$$

▶ **Corollary 35.** *Let δ, ε be reals in $(0, 1/8)$ satisfying $\delta > 16\sqrt{\varepsilon}$ Let W be a finite set of points in $\Re^d$. We can determine in polynomial time whether exists a set T forming a (ε,δ)-$\mathsf{ENV}(W)$, and if so, we can efficiently find a subset Q of W such that*

$$|Q| = |T| \tag{22}$$

$$\forall w \in T, \exists x \in Q : |w - x| \leq 8\sqrt{\varepsilon}\Delta(W) \tag{23}$$

Proof. The Corollary follows from Theorem (34) by taking $\varepsilon_3 = 4\sqrt{\varepsilon}$ ◀

We defer the proof of Theorem 34 to the full version [9]. The procedure for computing the set Q is as follows: We first compute a subset Q'' of W consisting of points $w \in W$ which do not lie in the soft convex hull of the points in W which are "far" from w – this can be done in polynomial time by using arguments similar to those in the proof of Section 6.1. Then Q is defined as a maximal subset of points in Q'' such that the pair-wise distance between the points in it is large.

6.2 Algorithm for k-OLP

We now show how soft convex hulls can be used to generate a solution for the k-OLP problem. The following result, which formalizes Theorem 9, uses the same setting as that in Theorem 28:

▶ **Theorem 36.** *Suppose ε, δ are reals in $[0,1]$ with $\delta^2 \geq c\varepsilon\sqrt{d}, \delta^3 \geq c\,\varepsilon$, where c is a large enough constant. Let K be a δ-well-separated k-vertex polytope. Suppose we are also given an $\mathsf{OptOr}_\varepsilon(K)$ oracle $\mathcal{O}$. Let W be the set of answers of the oracle $\mathcal{O}$ to $m = poly(d) \cdot k^{\Omega(1/\delta^2)}$ independent random queries. We can find $Q \subseteq W, |Q| = k$ in randomized $poly(d) \cdot k^{\Omega(1/\delta^2)}$-time which satisfies the following condition w.h.p.: for every vertex v of K, there is a point v' in Q with $|v - v'| \leq \delta\Delta(K)/10$.*

Proof. The proof of Theorem 28 shows that for every vertex $M_{\cdot,\ell}$ of K, there is a point $x(u^\ell) \in W$ such that

$$|x(u^{(\ell)}) - M_{\cdot,\ell}| \leq 17\delta^2\Delta(K)/c. \tag{24}$$

Let T denote $\{x(u^{(\ell)}) : \ell \in [k]\}$. Our first claim is that the points of T are also well-separated.

▷ Claim 37. For any $\ell \in [k]$, $\mathsf{dist}(x(u^{(\ell)}), \mathsf{CH}(T \setminus \{x(u^{(\ell)})\})) \geq \delta\Delta(K)/2$. Further, the diameter of $\mathsf{CH}(T)$ is at most $2\Delta(K)$.

Proof. Fix an index $\ell \in [k]$ and a point $y \in \mathsf{CH}(T \setminus \{x(u^{(\ell)})\})$. We can express y as a convex combination of points in $T \setminus \{x(u^{(\ell)})\}$, i.e., $y = \sum_{\ell' \neq \ell} \lambda_{\ell'} \cdot x(u^{(\ell')})$, where $\sum_{\ell' \neq \ell} \lambda_{\ell'} = 1$. Now,

$$\begin{aligned} |x(u^{(\ell)}) - y| &\geq \left| M_{\cdot,\ell} - \sum_{\ell' \neq \ell} \lambda_{\ell'} \cdot M_{\cdot,\ell'} \right| - |x(u^{(\ell)}) - M_{\cdot,\ell}| - \sum_{\ell' \neq \ell} \lambda_{\ell'} \cdot |x(u^{(\ell')}) - M_{\cdot,\ell'}| \\ &\geq \left(\delta - \frac{34\delta^2}{c}\right)\Delta(K) \geq \delta\Delta(K)/2, \end{aligned}$$

where the second last inequality follows from (24) and the fact that K is δ-well-separated. Since $\mathsf{dist}(x(u^{(\ell)}), K) \leq \varepsilon\Delta(K)$, it follows that $\Delta(\mathsf{CH}(T)) \leq 2\Delta(K)$. ◁

Recall that W denotes the set $\{x(u) : u \in U\}$. We now show that $\mathsf{CH}(T)$ closely approximates $\mathsf{CH}(W)$.

▷ Claim 38. $W \subseteq \varepsilon'\text{-}\mathsf{CH}(T)$, where $\varepsilon' = \frac{32\delta^2}{c}$.

Proof. Fix a point $x(u) \in W$. We know that $x(u)$ can be written as $x(u) = y(u) + z(u)$, $y(u) \in K, |z(u)| \leq \varepsilon$. Let $y(u) = \sum_{\ell \in [k]} \lambda_\ell \cdot M_{\cdot,\ell}$, where the coefficients λ_ℓ form a convex combination. Then

$$\left| x(u) - \sum_{\ell \in [k]} \lambda_\ell x(u^{(\ell)}) \right| \leq \sum_{\ell \in [k]} \lambda_\ell \cdot |x(u^{(\ell)}) - M_{\cdot,\ell}| + |z(u)| \leq \frac{17\delta^2\Delta(K)}{c} + \varepsilon\Delta(K) \leq \frac{32\delta^2\Delta(K)}{c},$$

where the second last inequality follows from (24) and the last inequality by the assumption in. ◁

Claim 37 and Claim 38 imply that T is (ε', δ')-$\mathsf{ENV}(W)$ with $\delta' = \delta/4, \varepsilon' = \frac{32\delta^2}{c}$. We can now apply Corollary 35 to get approximations to $x(u^{(\ell)})$ within distance $17\sqrt{\varepsilon'}\Delta(K)$. Claim 38 now implies that we can get approximations to $M_{\cdot,\ell}$ within distance

$$17\sqrt{\varepsilon'}\Delta(K) + \frac{16\delta^2\Delta(K)}{c} \leq \frac{\delta\Delta(K)}{10}.$$

This proves the desired result. ◀

7 k-OLP algorithm for Latent Polytopes using Singular Value Decomposition

Theorem 7 showed that a solution to k-OLP problem requires the error parameter ε to be $O^*(1/\sqrt{d})$. Theorem 28 gives an algorithm achieving this bound. However, for many polytopes with k vertices, we can solve the k-OLP problem with ε being $O^*(1/\sqrt{k})$. However, if $k < d$, this error is too high. To tackle this, we find a good approximation to the subspace spanned by the vertices of K, then we project to this subspace and use the result in Theorem 28. One such example is the "Latent $k-$ Polytope" (abbreviated LkP) problem which we now describe.

The LkP problem has been studied in [7]. Certain assumptions were made on the model, namely, the hidden polytope K as well as on the (hidden) process for generating observed data from latent points in K. These assumptions are (a) shown to hold in several important Latent Variable models and (b) are sufficient to enable one to get polynomial time learning algorithms.

Here, we formulate assumptions which are similar, but, weaker in one important aspect. Whereas [7] assumed that each vertex of K has a separation from the **affine hull** of the other vertices (thus, in particular, each vertex is affinely independent of other vertices), we assume here that each vertex is separated only from the convex hull of the others. Under this weaker assumption, the algorithm of [7] does not work. We give a different algorithm which we prove works. It is also simpler to state and carry out and its proof is based on a new general tool we introduce here - the Random Separating Hyperplane theorem (Theorem 12).

Assumptions on data in the LkP problem. Let $M_{.,1}, \ldots, M_{.,k}$ denote the vertices of K and $\mathbf{M}$ be the $d \times k$ matrix with columns representing the vertices of K. We assume there are latent (hidden) points $P_{.,j}, j = 1, 2, \ldots, n$ in K and observed data points $A_{.,j}, j = 1, 2, \ldots, n$ are generated (not necessarily under any stochastic assumptions) by adding *displacements* $A_{.,j} - P_{.,j}$ respectively to $P_{.,j}$. Clearly if the displacements are arbitrary, it is not possible to learn K given only the observed data. So we need some bound on the displacements.

Secondly, if all (or almost all) latent points lie in (or close to) the convex hull of a subset of $k-1$ or fewer vertices of K, the missing vertex cannot be learnt. To avoid this, we will assume that there is a certain w_0 fraction of latent points close to every vertex of K.

Let [4] $\sigma_0 := \frac{||\mathbf{P}-\mathbf{A}||}{\sqrt{n}}$. We now show that the k-OLP-**Algorithm** mentioned in Section 1.2 has the desired properties:

▶ **Theorem 39.** *Suppose K is a latent poytope with k vertices $M_{.,1}, M_{.,2}, \ldots, M_{.,k}$ and $\mathbf{P}, \mathbf{A}$ are latent points (all in K) and observed data respectively. Assume*

$$\text{For all } \ell \in [k]\ ,\ C_\ell := \{j : |P_{.,j} - M_{.,\ell}| \leq \frac{\sigma_0}{\sqrt{w_0}}\} \text{ satisfies } |C_\ell| \geq w_0 n. \tag{25}$$

Suppose $(\sqrt{\log k}/\sqrt{c_0 k}) \leq \delta \leq 1$ and c_0 is a large constant satisfying

$$\sigma_0 \leq \frac{\delta^2 \Delta(K)}{100 c_0} \frac{\sqrt{w_0}}{\sqrt{k}}. \tag{26}$$

Let V be the k-dimensional SVD subspace of $\mathbf{A}$, and $\widehat{K}$ denote the projection of K on V.

[4] By the standard definition of spectral norm, it is easy to see that σ_0^2 is the maximum mean squared displacement in any direction.

- *There is an* $\mathsf{OptOr}_{\frac{10\sigma_0}{\sqrt{w_0}\Delta}}(\widehat{K})$ *oracle* $\mathcal{O}$.
- *The algorithm* k-OLP **Algorithm** *in Section 1.2 outputs a set* Q *of* k *points such that the following condition is satisfied w.h.p.: for every vertex* v *of* K*, there is a point* v' *in* Q *with* $|v - v'| \leq \delta\Delta(K)/5$.

In the Theorem above, (26) implies an upper bound on σ_0 of $\Delta(K)\sqrt{w_0}/(c_0\sqrt{k})$ and so the oracle $\mathcal{O}$ used by the Theorem lies in $\mathsf{opt}_\varepsilon(\widehat{K})$ for $\varepsilon \leq 1/(c_0\sqrt{k})$. Thus, we get around the lower bound result Theorem 7 by working in the k-dimensional SVD subspace of $\mathbf{A}$. The proof of this result is deferred to the full version [9].

8 Open Problems

We now mention some problems that remain open in our work:

(i) In the statement of RSH, the success probability of the desired event is $O\left(1/k^{O(1/\delta^2)}\right)$. Can we improve the exponential dependence of the success probability on $1/\delta$?

(ii) Theorem 21 on the Haus problem returns exponentially many points whose convex hull approximates K. Can this be improved, either via an improvement mentioned in the first open problem above or by feeding the exponentially many points to the algorithm of [12]?

(iii) RSH asserts that if a point is sufficiently far from a k vertex polytope, then, a random hyperplane separates them, where, "sufficienlty" is measured with respect to the diameter of the polytope. An interesting question is whether there is a "dual" statement. The dual of points are facets, so, intuitively, a dual statement might assert that if for a polytope with k facets, there is a sufficiently deep cut, then, a random cut cuts reasonably deep into the polytope. But it is not obvious how to measure "deep" (and reasonably deep) . We leave a dual statement of RSH as an open question.

References

1 Edoardo M. Airoldi, David M. Blei, Elena A. Erosheva, and Stephen E. Fienberg. Introduction to mixed membership models and methods. In Edoardo M. Airoldi, David M. Blei, Elena A. Erosheva, and Stephen E. Fienberg, editors, *Handbook of Mixed Membership Models and Their Applications*, pages 3–13. Chapman and Hall/CRC, 2014. doi:10.1201/b17520-3.

2 Noga Alon. Problems and results in extremal combinatorics—i. *Discrete Mathematics*, 273(1-3):31–53, 2003.

3 Anima Anandkumar, Dean P. Foster, Daniel J. Hsu, Sham M. Kakade, and Yi-Kai Liu. A spectral algorithm for latent dirichlet allocation. *Algorithmica*, 72(1):193–214, 2015. doi:10.1007/s00453-014-9909-1.

4 Sanjeev Arora, Rong Ge, Yoni Halpern, David M. Mimno, Ankur Moitra, David A. Sontag, Yichen Wu, and Michael Zhu. Learning topic models - provably and efficiently. *Commun. ACM*, 61(4):85–93, 2018. doi:10.1145/3186262.

5 Ainesh Bakshi, Chiranjib Bhattacharyya, Ravi Kannan, David P. Woodruff, and Samson Zhou. Learning a latent simplex in input sparsity time. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=04LZCAxMSco.

6 D. Bertsekas. *Convex Optimization Theory*. Athena Scientific optimization and computation series. Athena Scientific, 2009. URL: https://books.google.co.in/books?id=0H1iQwAACAAJ.

7 Chiranjib Bhattacharyya and Ravindran Kannan. Finding a latent k-simplex in O^* ($k \cdot$ nnz(data)) time via subset smoothing. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 122–140. SIAM, 2020. doi:10.1137/1.9781611975994.8.

8 Chiranjib Bhattacharyya, Ravindran Kannan, and Amit Kumar. Finding k in latent k-polytope. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 894–903. PMLR, 2021. URL: http://proceedings.mlr.press/v139/bhattacharyya21a.html.

9 Chiranjib Bhattacharyya, Ravindran Kannan, and Amit Kumar. Random separating hyperplane theorem and learning polytopes, 2023. arXiv:2307.11371.

10 David M. Blei. Probabilistic topic models. *Commun. ACM*, 55(4):77–84, 2012. doi:10.1145/2133806.2133826.

11 David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003. URL: http://jmlr.org/papers/v3/blei03a.html.

12 Avrim Blum, Sariel Har-Peled, and Benjamin Raichel. Sparse approximation via generating point sets. *ACM Trans. Algorithms*, 15(3):32:1–32:16, 2019. doi:10.1145/3302249.

13 Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2014. doi:10.1017/CBO9780511804441.

14 Samuel B. Hopkins and David Steurer. Efficient bayesian estimation from few samples: Community detection and related problems. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 379–390. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.42.

15 William Johnson and Joram Lindenstrauss. Extensions of lipschitz maps into a hilbert space. *Contemporary Mathematics*, 26:189–206, January 1984. doi:10.1090/conm/026/737400.

16 Kasper Green Larsen and Jelani Nelson. Optimality of the johnson-lindenstrauss lemma. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 633–638. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.64.

17 B. Laurent and P. Massart. Adaptive estimation of a quadratic functional by model selection. *The Annals of Statistics*, 28(5):1302–1338, 2000. doi:10.1214/aos/1015957395.

18 R Tyrrell Rockafellar and Johannes O Royset. Random variables, monotone relations, and convex analysis. *Mathematical Programming*, 148:297–331, 2014.

19 Roman Vershynin. Introduction to the non-asymptotic analysis of random matrices. *arXiv preprint*, 2010. arXiv:1011.3027.

A Some Examples

The first example shows that the factor $1/\sqrt{m}$ in the statement of RSH (Theorem 1) cannot be improved, even for a simple polytope K consisting of just one line segment.

▶ **Example 1.** Suppose K is a polytope in $\Re^d$ with just two vertices Δ distance apart and $V = \Re^d, \delta = 1$. It is easy to verify from standard results on random projection on a line that the additive term in inequality (2) is tight up to constant factors.

The next example shows that the success probability of the desired event (2) in Theorem 1 needs to depend on k, the number of vertices of the polytope K.

▶ **Example 2.** Consider the Euclidean space $\Re^d$ and the $d-1$ dimensional sphere $K := \{x \in \Re^d : |x| \leq 1, x_1 = 0\}$. Observe that the diameter Δ of K is 1. Let a denote the point $(1, 0, 0, \ldots, 0)$. It is at distance 1 from K, and so the parameter δ in the statement of Theorem 1 can be set to 1. But, with high probability, a random unit length vector u has $u \cdot a \approx \frac{c}{\sqrt{d}}$ for some constant c, whereas the maximum of $v \cdot x$ over all points x in the sphere K is about 1. Therefore the probability of the event (2) is exponentially small in d.

Another Hamiltonian Cycle in Bipartite Pfaffian Graphs

Andreas Björklund
IT University of Copenhagen, Denmark

Petteri Kaski
Aalto University, Finland

Jesper Nederlof
Utrecht University, The Netherlands

Abstract

Finding a Hamiltonian cycle in a given graph is computationally challenging, and in general remains so even when one is further given one Hamiltonian cycle in the graph and asked to find another. In fact, no significantly faster algorithms are known for finding another Hamiltonian cycle than for finding a first one even in the setting where another Hamiltonian cycle is structurally guaranteed to exist, such as for odd-degree graphs. We identify a graph class – the bipartite Pfaffian graphs of minimum degree three – where it is NP-complete to decide whether a given graph in the class is Hamiltonian, but when presented with a Hamiltonian cycle as part of the input, another Hamiltonian cycle can be found efficiently.

We prove that Thomason's lollipop method [Ann. Discrete Math., 1978], a well-known algorithm for finding another Hamiltonian cycle, runs in a linear number of steps in cubic bipartite Pfaffian graphs. This was conjectured for cubic bipartite planar graphs by Haddadan [MSc thesis, Waterloo, 2015]; in contrast, examples are known of both cubic bipartite graphs and cubic planar graphs where the lollipop method takes exponential time.

Beyond the reach of the lollipop method, we address a slightly more general graph class and present two algorithms, one running in linear-time and one operating in logarithmic space, that take as input (i) a bipartite Pfaffian graph G of minimum degree three, (ii) a Hamiltonian cycle H in G, and (iii) an edge e in H, and output at least three other Hamiltonian cycles through the edge e in G.

We also present further improved algorithms for finding optimal traveling salesperson tours and counting Hamiltonian cycles in bipartite planar graphs with running times that are not achieved yet in general planar graphs.

Our technique also has purely graph-theoretical consequences; for example, we show that every cubic bipartite Pfaffian graph has either zero or at least six distinct Hamiltonian cycles; the latter case is tight for the cube graph.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph theory

Keywords and phrases Another Hamiltonian cycle, Pfaffian graph, planar graph, Thomason's lollipop method

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.26

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2308.01574 [4]

Funding *Andreas Björklund*: Supported by the VILLUM Foundation, Grant 16582.
Jesper Nederlof: Supported by the project CRACKNP that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 Research and Innovation Programme, Grant 853234.

Acknowledgements We thank the anonymous reviewers of an earlier version of this paper for their comments, in particular for pointing out [13].

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 26; pp. 26:1–26:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Finding a Hamiltonian cycle in a given undirected graph is a well-known, well-researched, and hard problem. This paper studies the question whether knowledge of one Hamiltonian cycle helps in finding another one. More precisely, the Another Hamiltonian Cycle problem asks, given as input (i) a graph[1] G, (ii) a Hamiltonian cycle H in G, and (iii) an edge $e \in E(H)$, to find another Hamiltonian cycle $H' \neq H$ in G with $e \in E(H')$.

Our interest is in the class of bipartite *Pfaffian*[2] graphs, a superclass of the bipartite planar graphs. This focus is partly motivated by the fact that both in cubic bipartite graphs and cubic planar graphs, a well-known general algorithm for finding another Hamiltonian cycle, Thomason's lollipop method [27], requires exponential time in the worst case, cf. Section 1.1. Also, the problem of deciding if the graph contains a Hamiltonian cycle at all remains NP-hard in the family of cubic bipartite planar graphs, as proved by Akiyama, Nishizeki, and Saito [1].

As our main result, we show that Another Hamiltonian Cycle admits both a linear-time algorithm as well as a logarithmic-space algorithm in bipartite Pfaffian graphs of minimum degree three. Further restricted to cubic bipartite Pfaffian graphs, we prove that Thomason's lollipop method runs in a linear number of steps and can be implemented to run in linear time. This is to our knowledge a first example of a nontrivial graph class where Another Hamiltonian Cycle is efficiently solvable; such an example was solicited by Kintali [18]. By trivial we here intend a graph class in which Hamiltonicity detection is NP-hard but is artificially constructed to ensure a simple local rerouting of any Hamiltonian cycle. Rather, in our case the global properties of bipartiteness, Pfaffianity, and the everywhere-local property of minimum-degree three, interplay to enable an efficient algorithm. Without the minimum-degree constraint, the problem is NP-hard (see the full version [4]).

Our techniques have also purely graph-theoretic consequences. We show that every cubic bipartite Pfaffian graph has at least three other Hamiltonian cycles through any edge of a Hamiltonian cycle. All three Hamiltonian cycles can be found in linear time. We also show that such Hamiltonian graphs must have at least six Hamiltonian cycles. The 8-vertex cube graph (), the canonical example in this class, is an extremal example to both results. It has precisely six distinct Hamiltonian cycles with every graph edge in exactly four of them.

1.1 Motivation and earlier work

While a graph need not be Hamiltonian, and a Hamiltonian graph need not admit another Hamiltonian cycle, there exist graph families with Hamiltonian members where another Hamiltonian cycle is always known to exist. Perhaps the most prominent such family are the odd-degree graphs, which via Smith's Theorem (see [30]) have an even number of Hamiltonian cycles through any given edge. Thomason [27] gave a constructive proof by describing an algorithm that solves for another Hamiltonian cycle in cubic graphs. The algorithm is often called Thomason's lollipop method as it transforms a Hamiltonian cycle to another one by a sequence of lollipop graphs, see Section 3.3 for a precise description of the algorithm. Dropping the requirement that the Hamiltonian cycle should go through a specific edge,

[1] We tacitly work with undirected simple loopless graphs unless mentioned otherwise, as well as assume knowledge of standard graph-theoretic terminology [32]. Our conventions with graphs can be found in Section 1.6.

[2] We postpone a precise definition and motivation of Pfaffian graphs to Section 1.5. Planar graphs are Pfaffian.

Bosák [5] proved that every cubic bipartite graph has an even number of Hamiltonian cycles. Thomassen [28, 29] showed that no bipartite graph in which every vertex in one of the two parts of the bipartition has degree at least three, has a unique Hamiltonian cycle. A famous conjecture due to Sheehan [26] claims that no 4-regular graph can have a unique Hamiltonian cycle; Thomassen's result proves Sheehan's conjecture for bipartite graphs.

Papadimitriou [24] popularized the Another Hamiltonian Cycle problem and Thomason's algorithm by introducing the complexity class PPA and showed the containment of the problem in odd-degree graphs; completeness for PPA remains open. It is also open whether the problem can be solved in polynomial time – indeed, the drawback of Thomason's algorithm is that it may run for a very long time; Krawczyk [19] and Cameron [8] showed that Thomason's algorithm requires exponential time for a family of cubic planar graphs. Later this was shown by Zhong [33] also for cubic bipartite graphs. The best bound to date is the recent result of Briański and Szady [6], which shows that there are cubic 3-connected planar graphs on n vertices, in which Thomason's lollipop algorithm runs in $\Omega(1.18^n)$ time.

The above papers reason about Thomason's algorithm specifically, but it may of course be other algorithms that solve the problem more efficiently. Some progress in this direction was provided by Bazgan, Santha, and Tuza [2] that showed that one given a cubic graph on n vertices and one of its Hamiltonian cycles can find another cycle of length $(1-\epsilon)n$ for any fixed constant $\epsilon > 0$ in polynomial time. Deligkas, Mertzios, Spirakis, and Zamaraev [12] derived an exponential-time polynomial-space deterministic algorithm that given a cubic graph along with one of its Hamiltonian cycles finds another Hamiltonian cycle; the algorithm is shown to be faster than the fastest known exponential-time polynomial-space deterministic algorithm for finding a Hamiltonian cycle in cubic graphs.

1.2 Main results for bipartite Pfaffian graphs

Let us now review our main results for bipartite Pfaffian graphs and the underlying techniques in more detail. Our main theorems are as follows.

▶ **Theorem 1** (Main; Linear–time Another Hamiltonian Cycle in minimum degree three). *There exists a deterministic linear-time algorithm that, given as input (i) a bipartite Pfaffian graph G with minimum degree three, (ii) a Hamiltonian cycle H in G, and (iii) an edge $e \in E(H)$, outputs a Hamiltonian cycle $H' \neq H$ in G with $e \in E(H')$.*

▶ **Theorem 2** (Main; Logarithmic–space Another Hamiltonian Cycle in minimum degree three). *There exists a deterministic logarithmic-space algorithm that, given as input (i) a bipartite Pfaffian graph G with minimum degree three, (ii) a Hamiltonian cycle H in G, and (iii) an edge $e \in E(H)$, outputs a Hamiltonian cycle $H' \neq H$ in G with $e \in E(H')$.*

The framework underlying our main theorems can be used to prove an upper bound on the number of steps needed for Thomason's lollipop method to terminate.

▶ **Theorem 3** (Thomason's lollipop method in cubic bipartite Pfaffian graphs). *Thomason's lollipop method starting from any Hamiltonian cycle H and any edge $e \in E(H)$ in an n-vertex cubic bipartite Pfaffian graph G, terminates after at most n steps.*

It was conjectured in a master thesis at Waterloo by Haddadan [14] that Thomason's lollipop method runs in a linear number of steps in cubic bipartite planar graphs. It was there also proven to hold for the subfamily of such graphs that does not have the wheel graph on six vertices as a minor. However, as the author himself points out, finding a first Hamiltonian cycle in this limited graph family does not seem intractable. We are not aware of any other papers providing a polynomial time bound on Thomason's lollipop method in any graph class.

Our framework also can be used to prove the following structural results for Hamiltonian cycles. Their proofs appear in the full version [4] of this paper.

▶ **Corollary 4** (Non-uniqueness in minimum degree three)**.** *For every bipartite Pfaffian graph G of minimum degree three and for every edge $e \in E(G)$, it holds that G has either zero or at least four distinct Hamiltonian cycles H with $e \in E(H)$.*

The cube graph is a cubic bipartite Pfaffian graph with six distinct Hamiltonian cycles. We show that no Hamiltonian graph in this class can have fewer Hamiltonian cycles.

▶ **Corollary 5** (Cubic tight lower bound)**.** *Every cubic bipartite Pfaffian graph has either zero or at least six distinct Hamiltonian cycles.*

Chia and Ong [9] (in the paragraph after Theorem 10) asked whether there exists cubic bipartite planar graphs with exactly four Hamiltonian cycles. The above Corollary rules out that possibility.

Remarks. One consequence of Theorem 1 is that if the Another Hamiltonian Cycle problem in general odd degree graphs is PPA-complete as hypothesized by Papadimitriou [24, Open Problem (4)], then any proof cannot carry over to cubic bipartite planar graphs unless also PPA = FP. Our result also seems related to another well-known conjecture, namely Barnette's conjecture (cf. Tutte [31, Unsolved Problem V]), which states that every cubic 3-connected bipartite planar graph (*Barnette graph*) has a Hamiltonian cycle. Gorsky, Steiner, and Wiederrecht [13] recently extended the conjecture by showing that if Barnette's conjecture is true, it also holds that every cubic bipartite 3-connected Pfaffian graph has a Hamiltonian cycle. It is known that if Barnette's conjecture is true, then there is a Hamiltonian cycle through every edge in every such graph, see Kelmans [17]. Moreover, it was indirectly shown by Holton, Manvel, and McKay [15] that any Barnette graph larger than the smallest such graph – namely the cube graph – can be reduced to a smaller Barnette graph in such a way that if the smaller graph has a Hamiltonian cycle through every edge, then the larger one must also have a Hamiltonian cycle. This means that if it was possible given any single Hamiltonian cycle in a Barnette graph to generate a Hamiltonian cycle through any specific edge *not* on the initial cycle, then Barnette's conjecture would be constructively true. We remark that our algorithm is not known to be able to do this, not even indirectly by applying it several times in a chain of Hamiltonian cycle transformations. We do note however, that our algorithms in Theorems 1 and 2 not only make sure the edge $e = \{s, t\}$ is part of both Hamiltonian cycles, they also retain the other edge incident to s on H; this can be observed by Lemma 10, that is, no edge incident to s is changed by the algorithms since it is not on the alternating cycle we use. In particular this makes it possible given H and any given edge $f \in E(G)$ *not* on H, to find another Hamiltonian cycle H' such that f is also not part of H'.

1.3 Overview of techniques

At the heart of our algorithms and structural results is what we believe to be a new framework for efficiently witnessing a Hamiltonian cycle H through an arbitrary *anchor* edge e in a bipartite Pfaffian graph G. We associate a (not necessarily proper) two-coloring $\chi_H : V(G) \to \{0, 1\}$ to H that is unique to H (but dependent on a fixed but arbitrary Pfaffian orientation of G as well as e) and that defines a unique *acyclic* Hamiltonian[3] orientation of

[3] In precise terms, the orientation is a directed acyclic graph that contains as a directed subgraph a directed Hamiltonian *path* from one end of e to the other.

$G \setminus e$. We will refer to such an χ_H as a *good coloring*. This acyclicity in particular enables the unique recovery of H in linear time by standard topological sorting when given χ_H as advice. The reader may want to consult Figure 1 (on page 9) for an advance illustration at this point; the framework itself is developed in Section 2.

We also show that it suffices to know χ_H in only one of the parts of a bipartition of G to efficiently extend to a Hamiltonian cycle, which is not necessarily equal to H however. More precisely, we show that a coloring λ of one of the parts leads to an auxiliary bipartite graph F_λ whose perfect matchings correspond to the good colorings χ_H extending λ, which in turn each define a unique Hamiltonian cycle H. We refer to Figure 2 (on page 11) for an advance illustration of this setting.

Finally, when G has minimum degree three, we observe that we can use F_λ to efficiently switch from one Hamiltonian cycle H in G (described by a perfect matching M_H in F_λ) to another Hamiltonian cycle $H' \neq H$ in G by switching along an alternating cycle in F_λ which can be discovered through a directed cycle in an auxiliary directed graph $D_{\lambda,H}$. Moreover, we show that this construction and discovery can be executed in deterministic linear time – we refer to Figure 3 (on page 14) for an advance illustration; the switching construction itself is developed in Section 3.

1.4 Further results

Our framework for witnessing Hamiltonicity should be contrasted with the Cut&Count approach for detecting Hamiltonian cycles by Cygan, Nederlof, Pilipczuk, Pilipczuk, Van Rooij, and Wojtaszczyk [10], which reduces the Hamiltonian cycle problem to a local problem by showing that a cycle cover of the input graph is Hamiltonian if and only if the number of the exponentially many vertex partitions that are consistent (defined in a certain local way) with it is *odd*. This approach therefore necessarily reduces the original decision problem to a parity counting problem, which has several disadvantages that seem inherent to the approach, including the need for randomization and a running time factor that is pseudo-polynomial in the integer weights for edge-weighted problem variants. Our framework shows that for bipartite Pfaffian graphs there is a more natural way to witness that a cycle cover is Hamiltonian using only a *single* vertex partition; that is, χ_H.

We explore the algorithmic consequences of our technique in the context of counting Hamiltonian cycles and in the context of the Traveling Salesperson Problem (TSP) in bipartite Pfaffian/planar graphs. For reasons of space, we postpone a detailed statement of these results, a discussion of pertinent earlier work, as well as the proofs, to the full version [4] of this paper.

1.5 Pfaffian graphs

Let us now define and motivate Pfaffian graphs in more detail. An *orientation* of a graph G replaces every edge $\{u, v\} \in E(G)$ with either the directed arc (u, v) or the directed arc (v, u), thereby obtaining a directed graph $\vec{G}$. A cycle C in G is *central* if the graph $G \setminus V(C)$ admits a perfect matching. We say that an orientation of a cycle is *consistent* if it is strongly connected. An orientation $\vec{G}$ of G is *Pfaffian* if for every central cycle C in G it holds that both consistent orientations of C have an odd number of arcs in common with $\vec{G}$. A graph is *Pfaffian* if it admits a Pfaffian orientation.

The bipartite Pfaffian graphs are most famous as the graph class in which Pólya's permanent problem has a solution, the bipartite graphs in which one can compute the number of perfect matchings efficiently by reduction to a matrix determinant, see e.g. Robertson,

Seymour, and Thomas [25] and McCuaig [22]. This brings us to one of our motivations to study the complexity of the detection (and counting) of Hamiltonian cycles on this graph class: Previous algorithms for Hamiltonian cycles (such as the one by Björklund [3]) use determinant-based methods previously designed for counting matchings (modulo 2) in polynomial time; given this close connection between the two problems it is natural to ask whether Pfaffianity can be exploited for detecting and counting Hamiltonian cycles, similarly as for counting perfect matchings.

The bipartite Pfaffian graphs were characterized by Little [21] as those graphs G that do not have a vertex set $U \subseteq V(G)$ such that $G \setminus U$ has a perfect matching and the induced subgraph $G[U]$ admits an even subdivision of $K_{3,3}$ as a subgraph. McCuaig [22] and Robertson, Seymour, and Thomas [25] gave a structural characterization of bipartite Pfaffian graphs and the latter also outlined an $O(n^3)$ time algorithm for their recognition; this algorithm can also produce a Pfaffian orientation when one exists.

A general Pfaffian graph, as opposed to a bipartite one, can be very dense as observed by Norine [23]: there is an infinite family of n-vertex Pfaffian graphs with $\Omega(n^2)$ edges. This construction in particular poses obstacles to find characterizations as the ones mentioned above for bipartite graphs. Indeed, it is not known how to efficiently recognize a general Pfaffian graph.

The most famous Pfaffian graphs are the *planar* ones, graphs whose vertices can be embedded in the plane with straight lines connecting the vertices of every edge without line crossings except at endpoints. That planar graphs are Pfaffian was discovered by Kasteleyn [16]; there is also a linear-time algorithm that finds a Pfaffian orientation given a planar graph by Little [20].

1.6 Conventions and organization

We assume knowledge of standard graph-theoretic terminology; see e.g. West [32]. Graphs in this paper are undirected unless otherwise mentioned; this in particular also applies to subgraphs such as paths, cycles, and Hamiltonian cycles. No graph or directed graph in this paper has loops or multiple edges. For a graph or directed graph G, we write $V(G)$ for the vertex set of G and $E(G)$ for the edge set of G. We identify the *edges* of a graph with two-subsets $\{u, w\}$ where u and w are distinct vertices. We call the edges of a directed graph *arcs* in what follows, and identify each arc with a two-tuple (u, w) where u and w are distinct vertices. We recall our conventions with orientations and Pfaffian graphs from Section 1.5.

We work with Iverson's bracket notation – for a logical proposition P, we define

$$[\![P]\!] = \begin{cases} 1 & \text{if } P \text{ is true}\,; \\ 0 & \text{if } P \text{ is false}\,. \end{cases}$$

The rest of this paper is organized as follows. Section 2 presents our novel witnessing technique for Hamiltonian cycles in bipartite Pfaffian graphs. We prove our main theorems and their indirect structural corollaries in Section 3. Proofs of Lemmas, Theorems, and Corollaries that are omitted due to space constraints can be found in the full version [4] of this paper. In the full version we also prove the NP-hardness of Another Hamiltonian Cycle in bipartite Pfaffian graphs without constrained vertex degrees, and state, motivate, and prove further results on counting Hamiltonian cycles as well as on TSP.

2 Hamiltonian cycles in bipartite Pfaffian graphs

This section presents what we believe to be a novel technique to efficiently witness Hamiltonian cycles in bipartite Pfaffian graphs via (not necessarily proper) two-colorings of the vertices. We also show how to efficiently construct a Pfaffian orientation from a known Hamiltonian cycle in a bipartite Pfaffian graph, as well as show how to efficiently find a witness by extending a given partial witness defined on only one of the parts of a bipartition.

Throughout this section G is an n-vertex bipartite Pfaffian graph and $\vec{G}$ is a fixed but otherwise arbitrary Pfaffian orientation of G. Since we are interested in whether G is Hamiltonian, without loss of generality we may assume that n is even and $n \geq 4$ in what follows.

Select an arbitrary edge $e \in E(G)$ and call it the *anchor* edge.

2.1 Preliminaries: The structure of Pfaffian orientations

We start by recalling the known structure of Pfaffian orientations of G. Namely, de Carvalho, Lucchesi, and Murty [11] observed that all Pfaffian orientations of G are obtainable from each other by reversals of arcs across vertex cuts. More precisely, for any Pfaffian orientation $\vec{G}$ and any vertex $u \in V(G)$, it holds that reversing the arcs incident to u in $\vec{G}$ results in another Pfaffian orientation; furthermore, every Pfaffian orientation of G can be obtained by starting with an arbitrary Pfaffian orientation of G and repeating such operations for different vertices [11].

2.2 The two-coloring defined by an anchored Hamiltonian cycle

We are interested in characterising each Hamiltonian cycle H in G that traverses the selected anchor edge e – we say that such an H is *anchored* – using a function $\chi_H : V(G) \to \{0,1\}$ that is unique[4] to H and from which we will (in the next subsection) see H can be efficiently constructed.

Towards this end, let us study the Pfaffian orientation $\vec{G}$ at the anchor e. Let $(s,t) \in E(\vec{G})$ be the arc in $\vec{G}$ whose underlying edge in G is the anchor edge $e = \{s,t\}$. Construct from the Pfaffian orientation $\vec{G}$ a new orientation $\vec{G}_e$ of G that is otherwise identical to $\vec{G}$ except that the arc (s,t) has been replaced with the arc (t,s). That is, by definition we have $(t,s) \in E(\vec{G}_e)$.

Now consider an arbitrary anchored Hamiltonian cycle H in G. Since $e \in E(H)$, there is a unique consistent orientation $\vec{H}$ of H such that $(t,s) \in E(\vec{H})$. Let us write $v_0, v_1, \ldots, v_{n-1}$ for the vertices of G indexed in the directed $\vec{H}$-path order from s to t; that is,

$$v_0 = s\,, \quad v_{n-1} = t\,, \quad \text{and} \quad (v_i, v_{(i+1) \bmod n}) \in E(\vec{H}) \quad \text{for all } i = 0, 1, \ldots, n-1\,. \tag{1}$$

Associate with H the (not necessarily proper) vertex-coloring function $\chi_H : V(G) \to \{0,1\}$ defined by setting

$$\begin{aligned} &\chi_H(v_0) = 0 \quad \text{and} \\ &\chi_H(v_{i+1}) \equiv \chi_H(v_i) + [\![(v_i, v_{i+1}) \in E(\vec{G}_e)]\!] \pmod 2 \quad \text{for all } i = 0, 1, \ldots, n-2\,. \end{aligned} \tag{2}$$

[4] Unique but not canonical; as we will see, the function χ_H will depend not only on the Hamiltonian cycle H but also on the choice of our assumed fixed but arbitrary Pfaffian orientation $\vec{G}$ of G.

Because $\vec{G}$ is a Pfaffian orientation and H is a central cycle of G, we have

$$|E(\vec{H}) \cap E(\vec{G})| = \sum_{i=0}^{n-1} [\![(v_i, v_{(i+1) \bmod n}) \in E(\vec{G})]\!] \equiv 1 \pmod 2 . \tag{3}$$

Since $\vec{G}$ and $\vec{G}_e$ differ only in the orientation of $e \in E(H)$, from (3) we immediately have

$$|E(\vec{H}) \cap E(\vec{G}_e)| = \sum_{i=0}^{n-1} [\![(v_i, v_{(i+1) \bmod n}) \in E(\vec{G}_e)]\!] \equiv 0 \pmod 2 . \tag{4}$$

We thus conclude

$$\begin{aligned} \chi_H(v_0) \overset{(2)}{=} 0 \overset{(4)}{\equiv} |E(\vec{H}) \cap E(\vec{G}_e)| &= \sum_{i=0}^{n-2} [\![(v_i, v_{i+1}) \in E(\vec{G}_e)]\!] + [\![(v_{n-1}, v_0) \in E(\vec{G}_e)]\!] \\ &\overset{(2)}{\equiv} \chi_H(v_{n-1}) + [\![(v_{n-1}, v_0) \in E(\vec{G}_e)]\!] \pmod 2 . \end{aligned} \tag{5}$$

Since by definition of $\vec{G}_e$ we have $(t, s) \in E(\vec{G}_e)$, from (1) and (5) we conclude that $\chi_H(t) = 1$. Furthermore, from (2) and (5) we have for all $(u, w) \in E(\vec{H})$ that

$$\chi_H(w) \equiv \chi_H(u) + [\![(u, w) \in E(\vec{G}_e)]\!] \pmod 2 . \tag{6}$$

That is, each arc $(u, w) \in E(\vec{H})$ is χ_H-monochromatic (i.e. both endpoints are assigned the same value by χ_H) if and only if $(w, u) \in E(\vec{G}_e)$.

2.3 The orientation induced by a good coloring

Suppose now that we do not know anything about the (anchored) Hamiltonian cycles of G, if any, and have access only to the Pfaffian orientation $\vec{G}$ and the orientation $\vec{G}_e$; the latter is easily obtainable from $\vec{G}$, cf. Section 2.2.

Consider an arbitrary vertex coloring $\chi : V(G) \to \{0, 1\}$ with $\chi(s) = 0$ and $\chi(t) = 1$. The last observation in Section 2.2 suggests that we should explore reversing exactly the χ-monochromatic arcs in $\vec{G}_e$. Let us make this formal as follows. Let the orientation $\vec{G}_e^{\chi}$ *induced* by the coloring χ be the unique orientation of G that for each edge $\{u, w\} \in E(G)$ satisfies

$$(u, w) \in E(\vec{G}_e^{\chi}) \quad \text{if and only if} \quad \chi(w) \equiv \chi(u) + [\![(u, w) \in E(\vec{G}_e)]\!] \pmod 2 . \tag{7}$$

To witness the serendipity of (7), suppose that G admits an anchored Hamiltonian cycle H; it follows immediately from (6) and (7) that $E(\vec{H}) \subseteq E(\vec{G}_e^{\chi_H})$. Thus, if we know only the coloring χ_H but not H, we can search for $E(\vec{H})$ in $E(\vec{G}_e^{\chi_H})$; let us next analyse this situation in more detail from the standpoint of our arbitrary χ.

Call the coloring χ *good* if there exists an anchored Hamiltonian cycle H in G with $\chi = \chi_H$; otherwise call χ *bad*. The following lemma shows that the orientation $\vec{G}_e^{\chi}$ for a good χ enables *linear-time* and unique algorithmic recovery of H by standard longest-path search in a directed acyclic graph (DAG); in fact, mere topological sorting suffices, as is apparent from the proof. See also Figure 1 for an illustration of the concepts involved in a Hamiltonian planar graph.

▶ **Lemma 6** (Acyclic Hamiltonicity of good-coloring-induced orientations). *Let χ be good. Then, the orientation $\vec{G}_e^{\chi} \setminus (t, s)$ of $G \setminus e$ is acyclic with the unique source vertex s and the unique sink vertex t. Moreover, the longest directed path in $\vec{G}_e^{\chi} \setminus (t, s)$ is unique and a directed Hamiltonian path.*

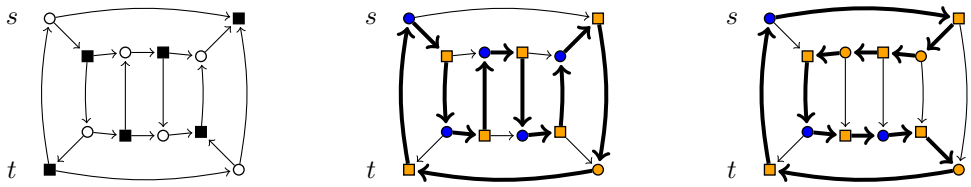


Figure 1 Illustration of orientations induced by good colorings. Left: an undirected bipartite planar graph G drawn in one of its orientations $\vec{G}_e$ with $e = \{s, t\}$, one arc reversal away from a Pfaffian orientation $\vec{G}$. Middle and right: two vertex colorings χ_H and coloring-induced orientations $\vec{G}_e^{\chi_H}$ for two different Hamiltonian cycles H, with the arcs of $\vec{H}$ drawn in bold in each case. Observe that every monochromatic arc reverses its orientation with respect to $\vec{G}_e$, whereas bichromatic arcs keep their orientation. Observe also that the removal of the arc (t, s) from $\vec{G}_e^{\chi_H}$ leaves an acyclic Hamiltonian directed graph, whereby the directed Hamiltonian path and hence H can be found, for example, by topological sorting; cf. Lemma 6.

Proof. Since χ is good, there exists an anchored Hamiltonian cycle H with $\chi = \chi_H$. Furthermore, we can follow the notational conventions in Section 2.2 with respect to this H, including the vertex-indexing $v_0, v_1, \ldots, v_{n-1}$ for G and (1) in particular. From $E(\vec{H}) \subseteq E(\vec{G}_e^{\chi})$ we thus conclude that the sequence $v_0, v_1, \ldots, v_{n-1}$ defines a longest directed path (which is also a directed Hamiltonian path from s to t) in the directed graph $\vec{G}_e^{\chi} \setminus (t, s)$. It follows immediately that s is the only possible source vertex and t is the only possible sink vertex in $\vec{G}_e^{\chi} \setminus (t, s)$.

Let us next show that $\vec{G}_e^{\chi} \setminus (t, s)$ is acyclic as a directed graph. To reach a contradiction, suppose that $\vec{D}$ is a directed cycle in $\vec{G}_e^{\chi} \setminus (t, s)$. Since $(t, s) = (v_{n-1}, v_0) \notin E(\vec{D})$ and $\vec{D}$ is a directed cycle with $V(\vec{D}) \subseteq \{v_0, v_1, \ldots, v_{n-1}\}$, there must exist $0 \leq i < j \leq n-1$ with at least two proper inequalities among the three such that $(v_j, v_i) \in E(\vec{D})$ and thus $(v_j, v_i) \in E(\vec{G}_e^{\chi})$. Let $\vec{C}$ be the directed cycle with $V(\vec{C}) = \{v_i, v_{i+1}, \ldots, v_j\}$ and $E(\vec{C}) = \{(v_i, v_{i+1}), (v_{i+1}, v_{i+2}), \ldots, (v_{j-1}, v_j), (v_j, v_i)\}$. In particular, $\vec{C} \neq \vec{H}$ since $(t, s) \notin E(\vec{C})$. Let C be the underlying undirected cycle of $\vec{C}$, and observe that C is a cycle of G. Since G is bipartite, C is even and has at least four vertices. Thus, the edges of $H \setminus V(C)$ contain a perfect matching of $G \setminus V(C)$, implying that C is central. Since C avoids e and $\vec{C}$ is a consistent orientation of C, we conclude by Pfaffianity that

$$|E(\vec{C}) \cap E(\vec{G}_e)| = |E(\vec{C}) \cap E(\vec{G})| \equiv 1 \pmod 2 .$$

But this is a contradiction since for all $(u, w) \in E(\vec{C})$ we have $(u, w) \in E(\vec{G}_e^{\chi})$, and thus by (7) it holds that $\chi(w) \equiv \chi(u) + [\![(u, w) \in E(\vec{G}_e)]\!] \pmod 2$; take the sum of these congruences over all arcs $(u, w) \in E(\vec{C})$ to conclude that $|E(\vec{C}) \cap E(\vec{G}_e)| \equiv 0 \pmod 2$, a contradiction. Thus, $\vec{G}_e^{\chi} \setminus (t, s)$ is acyclic as a directed graph.

From acyclicity it also immediately follows that s is a source vertex and t is a sink vertex of $\vec{G}_e^{\chi} \setminus (t, s)$; indeed, any arc into s or any arc out of t would complete a directed cycle together with an appropriate proper segment of the directed Hamiltonian path $s = v_0, v_1, \ldots, v_{n-1} = t$. This longest path (of n vertices) is also seen to be unique in $\vec{G}_e^{\chi} \setminus (t, s)$; indeed, the existence of any other such path would again imply an arc that would complete a directed cycle together with an appropriate proper segment of $v_0, v_1, \ldots, v_{n-1}$. ◀

An immediate corollary of the proof Lemma 6 is that there are at most 2^{n-2} Hamiltonian cycles through any fixed edge in an n-vertex bipartite Pfaffian graph. For comparison, there are planar graphs with at least 2.08^n Hamiltonian cycles, see [7].

2.4 Constructing a Pfaffian orientation from a Hamiltonian cycle

Next we address the task of constructing a Pfaffian orientation if we know one Hamiltonian cycle, with the intent of constructing possible further Hamiltonian cycles with the help of the Pfaffian orientation obtained.

▶ **Lemma 7** (Constructing a Pfaffian orientation from a Hamiltonian cycle)**.** *There exists a linear-time algorithm that, given as input a bipartite Pfaffian graph G and a Hamiltonian cycle H in G, outputs a Pfaffian orientation $\vec{G}$ of G.* ⋆*See full version [4] for proof*⋆

2.5 Finding a good coloring

Let us now study the task of finding a good coloring χ given the bipartite Pfaffian graph G, the Pfaffian orientation $\vec{G}$, and the anchor edge e as input; also recall the conventions and further notation – in particular, the vertices s and t – from Sections 2.2 and 2.3. In this setting, a natural question to ask is how much one needs to reveal from a good coloring χ to enable efficient completion to a good coloring. We now show that it suffices to reveal χ in one of the parts of the bipartition of G by a reduction to bipartite perfect matching in an auxiliary bipartite graph.

More precisely, let the sets L ("left") and R ("right") form a partition of the vertices of G such that $s \in L$, $t \in R$, and every edge of G has one end in L and the other end in R.[5] Let $\lambda : L \to \{0,1\}$ with $\lambda(s) = 0$ be a given further input. Our task is to find whether there exists a good coloring $\chi : V(G) \to \{0,1\}$ with $\chi(\ell) = \lambda(\ell)$ for all $\ell \in L \subseteq V(G)$; that is, whether there exists a good coloring that extends the partial coloring λ.

Construct an auxiliary bipartite graph F_λ as follows. Let the vertex set $V(F_\lambda) = V(G) \times \{0,1\}$. To avoid notational confusion between arcs and vertices of F_λ, we will use bracketed notation $[u,k]$ for vertices of F_λ with $u \in V(G)$ and $k \in \{0,1\}$. The edge set $E(F_\lambda)$ is defined by the following rule. For all $\ell \in L$, $r \in R$, $p \in \{0,1\}$, and $\rho \in \{0,1\}$ with $\{\ell, r\} \in E(G)$, we have

$$\{[\ell,p],[r,\rho]\} \in E(F_\lambda) \tag{8}$$

if and only if both

$$\rho \equiv \lambda(\ell) + [\![(\ell,r) \in \vec{G}_e]\!] \pmod 2 \tag{9}$$

and

$$\ell \neq s \quad \text{or} \quad p \neq 0 \quad \text{or} \quad r = t\,. \tag{10}$$

For an edge $\{[\ell,p],[r,\rho]\} \in E(F_\lambda)$, we say that the edge $\{\ell, r\} \in E(G)$ is the *projection* of the edge (to G) and call p the *port* at ℓ and ρ the *parity* at r, stressing that port and parity have asymmetric roles in our construction even though both range in $\{0,1\}$.

Let us now start analysing the structure of F_λ in more detail. First, the parts $L \times \{0,1\}$ and $R \times \{0,1\}$ witness by (8) that F_λ is bipartite. In particular, F_λ has $2n$ vertices with $|L| = |R| = n/2$, where we recall that $n \geq 4$ is the number of vertices in G with n even. Second, recalling that $s \in L$ and $t \in R$, the constraint (10) effectively states that $[s,0]$ is adjacent only to $[t,0]$ in F_λ; indeed, recalling that $\lambda(s) = 0$ and $(s,t) \notin \vec{G}_e$, from (9) we

[5] This bipartition (L,R) of G is in fact unique unless G is not Hamiltonian. Moreover, (L,R) is computable in linear time from the given input.

have that $[s,0]$ is not adjacent to $[t,1]$. Third, for all $\ell \in L \setminus \{s\}$, we observe from (9) and (10), the latter being trivially true, that the vertices $[\ell,0]$ and $[\ell,1]$ have identical vertex neighborhoods in F_λ.

We are now ready for our next key lemma. Recall the coloring χ_H associated to an anchored Hamiltonian cycle H of G from Section 2.2. The first lemma shows that every perfect matching in F_λ gives rise to an anchored Hamiltonian cycle; different perfect matchings may give rise to the same anchored Hamiltonian cycle however. This structure is illustrated in Figure 2.

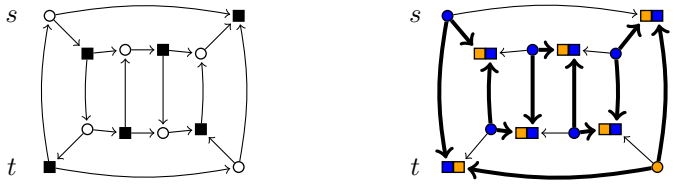


Figure 2 Illustration of a perfect matching in the graph F_λ. Left: The graph G drawn in one of its orientations $\vec{G}_e$ with $e = \{s,t\}$ and the bipartition (L,R) with vertices in L drawn as white circles and the vertices in R drawn as black boxes. Right: The graph F_λ and the coloring λ drawn as an oriented overlay of G. Observe that each vertex $r \in R$ has two copies $[r,\rho]$ in F_λ, one for each parity $\rho \in \{0,1\}$, with a blue square indicating parity 0 and an orange square indicating parity 1. Although each vertex $\ell \in L$ has two copies $[\ell,p]$ in F_λ, one copy for each port $p \in \{0,1\}$, we contract these two copies into one vertex (circle) in the drawing, and display for each vertex its color $\lambda(\ell) \in \{0,1\}$ (blue or orange) instead. Each arc in the drawing is oriented from L to R and represents two edges of F_λ with opposite ports. A perfect matching M in F_λ is represented by the bold arcs. In particular, observe that each circle is incident to two bold arcs; these two bold arcs represent two edges in M with opposite ports. These opposite ports are otherwise arbitrary except for the edge of M that projects to $\{s,t\}$, which must have port 0. Also observe that from the drawn M it is visually intuitive how to obtain a Hamiltonian cycle in G corresponding to M by turning the bold arcs into the edges of a Hamiltonian cycle in G; this intuition is made rigorous in Lemma 8 by the Hamiltonian cycle $\mathrm{H}[M]$ of G obtained from M.

▶ **Lemma 8** (Perfect matchings witness good extensions)**.** *For every perfect matching M in F_λ, there exists a Hamiltonian cycle $\mathrm{H}[M]$ in G with $\chi_{\mathrm{H}[M]}(\ell) = \lambda(\ell)$ for all $\ell \in L$.*

Proof. Let M be an arbitrary perfect matching in F_λ. For $[\ell,p] \in L \times \{0,1\}$ and $[r,\rho] \in L \times \{0,1\}$, let us use functional notation $M([\ell,p]) = [r,\rho]$ or $M([r,\rho]) = [\ell,p]$ to signal that the vertices $[\ell,p]$ and $[r,\rho]$ are matched by M in F_λ. We construct the anchored Hamiltonian cycle $\mathrm{H}[M]$ as well as the coloring $\chi = \chi_{\mathrm{H}[M]}$ with $\chi(\ell) = \lambda(\ell)$ for all $\ell \in L$ by traversing all the vertices of F_λ in an order determined by M to yield the Hamiltonian cycle $\mathrm{H}[M]$. In particular, we will define $\mathrm{H}[M]$ in steps by introducing, one vertex at a time, a vertex order $v_0, v_1, \ldots, v_{n-1}$ for the vertices of G with $v_0 = s$, $v_{n-1} = t$, and $(v_i, v_{(i+1) \bmod n}) \in E(\overrightarrow{\mathrm{H}[M]}) \subseteq E(\vec{G}_e^\chi)$ for all $i = 0,1,\ldots,n-1$.

Our traversal starts from the vertex $[\ell_0,p_0]$ of F_λ defined by $\ell_0 = s$ and $p_0 = 1$. The traversal then follows edges in the perfect matching M, changing parity (at vertices in $R \times \{0,1\}$) and port (at vertices in $L \times \{0,1\}$) to arrive at subsequent edges; for these changes, for $z \in \{0,1\}$ it is convenient to write $\overline{z} = (z+1) \bmod 2$ for notational brevity; that is, $\overline{0} = 1$ and $\overline{1} = 0$.

In precise terms, the traversal is as follows. Assuming we have defined $\ell_j \in L$ and $p_j \in \{0,1\}$ for all $j \in \{0,1,\ldots,i\}$ with $i \geq 0$, we proceed to define $\ell_{i+1} \in L$ and $p_{i+1} \in \{0,1\}$ as follows. Set $v_{2i} = \ell_i$ and $\chi(\ell_i) = \lambda(\ell_i)$. Define $r_i \in R$ and $\rho_i \in \{0,1\}$ by $M([\ell_i,p_i]) = [r_i,\rho_i]$.

Set $v_{2i+1} = r_i$ and $\chi(r_i) = \rho_i$. Define $\ell'_i \in L$ and $p'_i \in \{0,1\}$ by $M([r_i, \overline{\rho_i}]) = [\ell'_i, p'_i]$. Set $\ell_{i+1} = \ell'_i$ and $p_{i+1} = \overline{p'_i}$ as well as $\chi(\ell_{i+1}) = \lambda(\ell_{i+1})$ and $v_{2(i+1)} = \ell_{i+1}$. We continue this process for $i = 0, 1, \ldots$ and claim that eventually $\ell_{i+1} = \ell_0$ and $p_{i+1} = p_0$ with $i+1 = n/2$, at which point $\{v_0, v_1, \ldots, v_{n-1}\} = V(G)$ and $\mathrm{H}[M]$ is a Hamiltonian cycle in G with $\chi_{\mathrm{H}[M]} = \chi$ with $\chi(\ell) = \lambda(\ell)$ for all $\ell \in L$.

Let us now analyse the traversal process in more detail. First, we observe that $\ell_{i+1} \neq \ell_i$; indeed, suppose that $\ell_{i+1} = \ell_i$ and observe that the traversal step from ℓ_i to ℓ_{i+1} changes parity from ρ_i to $\bar{\rho}_i$ at r_i, yet from (9) we observe that every edge of F_λ that projects to the edge $\{\ell_i, r_i\} = \{\ell_{i+1}, r_i\}$ has the same parity at r_i, a contradiction. Next, let us observe that $(v_{2i}, v_{2i+1}) = (\ell_i, r_i) \in E(\vec{G}_e^\chi)$. Indeed, the identity is immediate, and membership holds by (7) and

$$\chi(r_i) = \rho_i \overset{(9)}{\equiv} \lambda(\ell_i) + [\![(\ell_i, r_i) \in \vec{G}_e]\!] = \chi(\ell_i) + [\![(\ell_i, r_i) \in \vec{G}_e]\!] \pmod 2 .$$

Let us then observe that $(v_{2i+1}, v_{2i+2}) = (r_i, \ell_{i+1}) \in E(\vec{G}_e^\chi)$. Again the identity is immediate, and membership holds by (7), the fact that $\vec{G}_e^\chi$ orients $\{\ell_{i+1}, r_i\} \in E(G)$ in one of two possible orientations, and

$$\chi(r_i) = \rho_i \neq \bar{\rho}_i \overset{(9)}{\equiv} \lambda(\ell_{i+1}) + [\![(\ell_{i+1}, r_i) \in \vec{G}_e]\!] = \chi(\ell_{i+1}) + [\![(\ell_{i+1}, r_i) \in \vec{G}_e]\!] \pmod 2 .$$

Next let us show that all the vertices ℓ_i and r_i traversed by the process are distinct, until $\ell_{i+1} = \ell_0$ for some $i \geq 1$, noting that the case $i = 0$ has already been excluded earlier. Suppose $\ell_1, \ell_2, \ldots, \ell_i$ are distinct; since M contains exactly two edges (of opposite parities) that project to edges incident to any fixed $r \in R$, we observe that these two edges of M have been each traversed once by the process for each $r_0, r_1, \ldots, r_i$ since $\ell_0, \ell_1, \ldots, \ell_i$ are distinct, implying that $r_0, r_1, \ldots, r_i$ are distinct, and thus that $v_0, v_1, \ldots, v_{2i+1}$ are distinct. So suppose that $\ell_{i+1} = \ell_j$ for some $0 \leq j \leq i$; also note that this must happen for some $i < |L| = n/2$. If $j \geq 1$, we have a contradiction since M contains exactly two edges (of opposite ports) that project to edges incident to any fixed $\ell \in L$, and for $\ell = \ell_j$ these two edges (projecting to $\{\ell_j, r_{j-1}\}$ and $\{\ell_j, r_j\}$) have already been traversed; so there is no edge in M that projects to $\{\ell_j, r_i\} = \{\ell_{i+1}, r_i\}$, a contradiction. So we must have $j = 0$. This implies in particular that $(v_k, v_{(k+1) \bmod (2i+2)}) \in E(\vec{G}_e^\chi)$ for all $k = 0, 1, \ldots, 2i+1$. Furthermore, since the edge of M that is incident to $[\ell_0, p_0] = [s, 1]$ has already been traversed, we have that the edge $\{[\ell_{i+1}, \overline{p_{i+1}}], [r_i, \bar{\rho}_i]\}$ in M must be the edge $\{[s,0], [t,0]\}$ (recall our analysis earlier that $[s,0]$ is adjacent only to $[t,0]$ in F_λ); thus we conclude that $\chi(t) = \chi(r_i) = \rho_i \neq \bar{\rho}_i = 0$; that is, $\chi(t) = 1$.

Let us next show that $i = n/2 - 1$. So to reach a contradiction, suppose that $i < n/2 - 1$. In particular, the edges of G underlying the arcs $(v_k, v_{(k+1) \bmod (2i+2)}) \in E(\vec{G}_e^\chi)$ for $k = 0, 1, \ldots, 2i+1$ trace a cycle of even length $2i + 2 < n$ in G. This leaves some of the vertices in G, and thus all corresponding vertices of F_λ regardless of port/parity, unvisited by the traversal process. By starting the traversal process again from an arbitrary unvisited vertex in $L \times \{0,1\}$, we end up tracing a further even-length cycle in G, and repeating the process until all vertices of G are visited, we obtain a vertex-disjoint union of even-length cycles that together cover the vertices of G, as well as a coloring χ such that all the cycles (in their consistently oriented form as they were traversed) occur as directed subgraphs of $\vec{G}_e^\chi$. Since $2i + 2 < n$, this cycle cover thus contains a cycle C that does not contain the anchor edge e and whose consistent orientation $\vec{C}$ is a subgraph of $\vec{G}_e^\chi$; observing that C is central – indeed, use every other edge from each even cycle other than C in the cover to witness a perfect matching in $G \setminus V(C)$ – this leads to a contradiction via Pfaffianity by the same argument as was used in the proof of Lemma 6; thus, $i = n/2 - 1$.

Since $i = n/2 - 1$, it follows from $(v_k, v_{(k+1) \bmod n}) \in E(\vec{G}_e^{\chi})$ for all $k = 0, 1, \ldots, n-1$ and from Lemma 6 we conclude that $\chi = \chi_{\mathrm{H}[M]}$ for the anchored Hamiltonian cycle $\mathrm{H}[M]$ in G defined by $V(\mathrm{H}[M]) = \{v_0, v_1, \ldots, v_{n-1}\}$ and $E(\mathrm{H}[M]) = \{\{v_k, v_{(k+1) \bmod n}\} : k = 0, 1, \ldots, n-1\}$. ◀

Conversely, we show that good extensions of λ are witnessed by perfect matchings in F_λ.

▶ **Lemma 9** (Good extensions witness perfect matchings). *For every anchored Hamiltonian cycle H in G with $\chi_H(\ell) = \lambda(\ell)$ for all $\ell \in L$, there exists a perfect matching M_H in F_λ with $\mathrm{H}[M_H] = H$.* ⋆*See full version [4] for proof*⋆

Thus, F_λ has a perfect matching if and only if λ has a good extension. Moreover, from the proofs of Lemma 8 and Lemma 9 we observe that the transformations $M \mapsto \mathrm{H}[M]$ and $H \mapsto M_H$ are computable in linear time. We also observe that for every anchored Hamiltonian cycle H in G there are exactly $2^{n/2-1}$ perfect matchings M in F_λ with $\mathrm{H}[M] = H$; these M are all obtainable from each other by transposing ports at zero or more vertices $\ell \in L \setminus \{s\}$.

3 Another Hamiltonian cycle in bipartite Pfaffian graphs

This section studies the problem of finding another Hamiltonian cycle when given as input (i) a bipartite Pfaffian graph G and (ii) a Hamiltonian cycle H in G. Recall from Lemma 7 that we can in linear time construct a Pfaffian orientation $\vec{G}$ from this input. In what follows we thus tacitly assume that such a $\vec{G}$ is available and fixed together with an arbitrary anchor edge $e \in E(H)$.

3.1 Linear-time solvability in minimum degree three

Our first objective in this section is our main theorem, which we restate below for convenience.

▶ **Theorem 1** (Main; Linear–time Another Hamiltonian Cycle in minimum degree three). *There exists a deterministic linear-time algorithm that, given as input (i) a bipartite Pfaffian graph G with minimum degree three, (ii) a Hamiltonian cycle H in G, and (iii) an edge $e \in E(H)$, outputs a Hamiltonian cycle $H' \neq H$ in G with $e \in E(H')$.*

We now proceed to prove Theorem 1. Recall and assume the setting of Section 2.5. Observe that from the given input, we can in linear total time (a) find a Pfaffian orientation $\vec{G}$ using H, (b) compute the orientation $\vec{G}_e$, (c) compute the coloring χ_H, (d) compute the vertex bipartition (L, R) of G, (e) restrict χ_H to L to obtain the coloring λ, (f) construct the graph F_λ, as well as (g) construct the perfect matching M_H in F_λ.

Using M_H and F_λ, introduce the directed graph $D_{\lambda,H}$ with the vertex set $V(D_{\lambda,H}) = L$ and the arc set defined for all distinct $\ell, \ell' \in L$ by the rule $(\ell, \ell') \in E(D_{\lambda,H})$ if and only if there exist $p, p' \in \{0, 1\}$, $r \in R$, and $\rho \in \{0, 1\}$ such that

$$\{[\ell, p], [r, \rho]\} \in E(F_\lambda) \setminus M_H \quad \text{and} \quad \{[\ell', p'], [r, \rho]\} \in M_H\,. \tag{11}$$

That is, an arc $(\ell, \ell') \in E(D_{\lambda,H})$ indicates that (disregarding ports p and p') we can walk from ℓ to ℓ' in F_λ by traversing first an edge not in M_H, followed by an edge in M_H. We stress that the traversal (11) *preserves* the parity ρ for consecutive edges, whereas the traversal in the proof of Lemma 8 *changes* parity for consecutive edges; the latter also uses edges only in M_H.

We recall that G has minimum degree at least three; this enables us to find a Hamiltonian cycle other than H in G with the help of a directed cycle in $D_{\lambda,H}$ revealed in the following lemma.

▶ **Lemma 10** (Existence of an s-avoiding directed cycle in $D_{\lambda,H}$). *Suppose that every vertex of G has degree at least three. Then, the directed graph $D_{\lambda,H}$ contains at least one directed cycle that avoids the vertex s.* ⋆*See full version [4] for proof*⋆

By the previous lemma we thus know that $D_{\lambda,H}$ contains an s-avoiding directed cycle $\vec{Q}$ with $V(\vec{Q}) = \{\ell_0, \ell_1, \ldots, \ell_{k-1}\} \subseteq L$ and $(\ell_j, \ell_{(j+1) \bmod k}) \in E(\vec{Q})$ for each $j = 0, 1, \ldots, k-1$ and $k \geq 2$. We will use $\vec{Q}$ to construct from M_H another Hamiltonian cycle $H' \neq H$ in G. Figure 3 illustrates the construction.

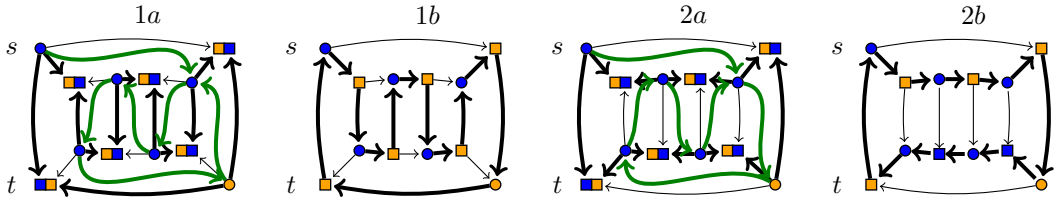


Figure 3 Obtaining another Hamiltonian cycle using a directed cycle in $D_{\lambda,H}$. $1a, 2a$: Two perfect matchings M_H in F_λ drawn as oriented overlays of G (cf. Figure 2), with further overlaying drawn in green and constituting the arcs of $D_{\lambda,H}$. $1b, 2b$: The corresponding two orientations $\vec{G}_e^{\chi_H}$ and colorings χ_H. In both cases we have that $D_{\lambda,H}$ contains a unique s-avoiding directed cycle $\vec{Q}$. Using $\vec{Q}$ in each case we can switch between the left and right Hamiltonian cycles in G. Note in particular that the two vertex colorings agree in L but differ in R.

From (11) applied to each arc of $\vec{Q}$ in turn we conclude that for $j = 0, 1, \ldots, k-1$ there exist $r_j \in R$ and $\rho_j, p_j \in \{0, 1\}$ with

$$\{[\ell_j, p_j], [r_j, \rho_j]\} \in E(F_\lambda) \setminus M_H \quad \text{and} \quad \{[\ell_{(j+1) \bmod k}, p'_j], [r_j, \rho_j]\} \in M_H\,. \tag{12}$$

We observe that the vertices $[r_j, \rho_j]$ for $j = 0, 1, \ldots, k-1$ are distinct because M_H is a perfect matching and $\ell_{(j+1) \bmod k}$ for $j = 0, 1, \ldots, k-1$ are distinct. In spite of this, the edges (12) for $j = 0, 1, 2, \ldots, k-1$ need *not* form a cycle in F_λ because we can have $p'_j \neq p_{(j+1) \bmod k}$. Here is where the fact that $\vec{Q}$ is s-avoiding pays off. Let $M = M_H$ and recall that the vertices $[\ell, 0]$ and $[\ell, 1]$ for each $\ell \in L \setminus \{s\}$ have identical vertex neighborhoods in F_λ. Thus, whenever we have $p'_j \neq p_{(j+1) \bmod k}$, we can modify M by transposing the vertices $[\ell_{(j+1) \bmod k}, 0]$ and $[\ell_{(j+1) \bmod k}, 1]$ in the edges of M; by the identical vertex neighborhoods property, the resulting M will still be a perfect matching in F_λ. Moreover, we have $H = \mathrm{H}[M_H] = \mathrm{H}[M]$; indeed, the traversal construction in Lemma 8 is insensitive to the specific values of the (opposite) ports. For $j = 0, 1, \ldots, k-1$ we thus now have

$$\{[\ell_j, p_j], [r_j, \rho_j]\} \in E(F_\lambda) \setminus M \quad \text{and} \quad \{[\ell_{(j+1) \bmod k}, p_{(j+1) \bmod k}], [r_j, \rho_j]\} \in M\,; \tag{13}$$

that is, these edges now form a $2k$-vertex cycle in F_λ. Let us write A for this cycle in F_λ.

Observe in particular from (13) that the edges of A alternate between edges in M and edges not in M. Thus, we have that the symmetric difference $M' = (M \setminus E(A)) \cup (E(A) \setminus M)$ is a perfect matching in F_λ. Furthermore, M' and M project to a different set of edges of G; indeed, from (13) we have that r_j changes adjacency from $\ell_{(j+1) \bmod k}$ in $\mathrm{H}[M]$ to ℓ_j in $\mathrm{H}[M']$ for $j = 0, 1, \ldots, k-1$. It follows that $H' = \mathrm{H}[M'] \neq \mathrm{H}[M] = H$. Thus, we have constructed a Hamiltonian cycle H' in G that is different from H. Moreover, this construction is computable in deterministic linear time. This completes the proof of Theorem 1. ◀

3.2 Logarithmic-space solvability in minimum degree three

We can implement the ideas of the previous section in an algorithm that uses little space. We consider as input a bipartite Pfaffian graph G of minimum degree three, a consistently oriented Hamiltonian cycle $\vec{H}$ in G, and an arc $(t,s) \in \vec{H}$. More precisely, we assume both graphs G and $\vec{H}$ are given in the input as a list of adjacency lists for each vertex. We seek to output a list of edges of another Hamiltonian cycle $H' \neq H$ with $\{s,t\} \in E(H')$. We assume that the vertices of G are represented as $O(\log n)$-bit integers in the input, where n is the number of vertices in G.

▶ **Theorem 2** (Main; Logarithmic–space Another Hamiltonian Cycle in minimum degree three). *There exists a deterministic logarithmic-space algorithm that, given as input (i) a bipartite Pfaffian graph G with minimum degree three, (ii) a Hamiltonian cycle H in G, and (iii) an edge $e \in E(H)$, outputs a Hamiltonian cycle $H' \neq H$ in G with $e \in E(H')$.*

⋆See full version [4] for proof⋆

3.3 Thomason's lollipop method in cubic bipartite Pfaffian graphs

In this section we prove that Thomason's lollipop method runs in linear time in cubic bipartite Pfaffian graphs. Let us first set up some preliminaries and then describe Thomason's lollipop method.

Let G be a Hamiltonian cubic graph and let H be a Hamiltonian cycle in G. Select an edge $e = \{s,t\}$ in the Hamiltonian cycle H. Let us call e the *anchor* edge. A *lollipop* is a connected graph with one vertex of degree one, one vertex of degree three, and all other vertices of degree two. All lollipops considered in what follows are subgraphs of G such that t is the unique degree-one vertex and e is the edge incident to t on the lollipop.

The lollipop method is best described as operating on a family of Hamiltonian paths in G. We say that a Hamiltonian path in G that starts at the vertex t and continues via the anchor edge e is an *e-anchored* Hamiltonian path. Now recall that G is cubic, so the vertex t is adjacent to s (via the anchor edge e) and to two other vertices a and b. The lollipop method transforms a given e-anchored Hamiltonian path P_e that ends at either a or b into an e-anchored Hamiltonian path $P'_e \neq P_e$ that ends at either a or b. Observe in particular that both Hamiltonian paths P_e and P'_e can be completed into e-anchored Hamiltonian cycles by adding the missing edge $\{a,t\}$ or $\{b,t\}$ into the respective path.

The transformation from P_e to P'_e is via a sequence of *lollipop steps*. A lollipop step consists of adding one edge to an e-anchored Hamiltonian path and removing another one, so that another e-anchored Hamiltonian path is formed. More precisely, let Q be an e-anchored Hamiltonian path ending at some vertex u. Since G is cubic, u is adjacent to two other vertices, x and y, such that the edges $\{u,x\}$ and $\{u,y\}$ of G are not in Q. Assume that $x \neq t$. Add the edge $\{u,x\}$ into Q to obtain a lollipop Ω where the unique degree-three vertex is x. Now observe that among the three adjacent vertices to x there is a unique vertex $v \notin \{u,t\}$ such that both $\{v,x\} \in E(\Omega)$ and removing $\{v,x\}$ from Ω leaves an e-anchored Hamiltonian path Q' ending at v. The transformation from Q to Q' now constitutes one lollipop step. Observe also that lollipop steps are *reversible*; that is, we can go back to Q from Q' by performing a lollipop step starting from Q'.

The *lollipop state graph* $\mathcal{L}(G,s,t)$ has as its vertices the e-anchored Hamiltonian paths in G and two vertices are joined by an edge if and only if it is possible to transform between the e-anchored Hamiltonian paths by one lollipop step. We observe immediately that $\mathcal{L}(G,s,t)$ has no isolated vertices – indeed, from any vertex Q we can arrive at another vertex $Q' \neq Q$ by a lollipop step – and the degree-one vertices are exactly the e-anchored Hamiltonian paths

Q that end at a vertex u adjacent to t in G; that is, $u \in \{a, b\}$; moreover, all other vertices have degree two. Thus, we can transform from P_e to $P'_e \neq P_e$ by tracing a path in $\mathcal{L}(G, s, t)$ from P_e to P'_e.

We now proceed to prove an upper bound on the maximum length of a path in $\mathcal{L}(G, s, t)$ on a cubic bipartite Pfaffian graph G. An example of the lollipop method applied to a cubic bipartite planar graph using the terminology in the subsequent proof is given in Figure 4.

▶ **Theorem 3** (Thomason's lollipop method in cubic bipartite Pfaffian graphs). *Thomason's lollipop method starting from any Hamiltonian cycle H and any edge $e \in E(H)$ in an n-vertex cubic bipartite Pfaffian graph G, terminates after at most n steps.*

Proof. To analyze the lollipop method in cubic bipartite Pfaffian graphs, let a cubic bipartite Pfaffian graph G, a Hamiltonian cycle H in G, and $e = \{s, t\} \in E(H)$ be given as input. This input enables us to work in the setting of Section 3.1; let the vertex bipartition (L, R) of G, the coloring λ of L, the graph F_λ, the perfect matching M_H in F_λ, and the directed graph $D_{\lambda,H}$ be constructed accordingly. Recall that $s \in L$ and $t \in R$.

The lollipop method starts by removing the edge $\{t, u\}$ with $u \neq s$ from H to obtain e-anchored Hamiltonian path P_e. Let $Q_0, Q_1, \ldots, Q_h$ be the sequence of e-anchored Hamiltonian paths traversed by consecutive lollipop steps in $\mathcal{L}(G, s, t)$ with $P_e = Q_0$ and $Q_h = P'_e$. We will show that P'_e ends at u and thus we can obtain a Hamiltonian cycle $H' \neq H$ by inserting the edge $\{t, u\}$ into P'_e. Moreover and crucially, we will show that $h \leq n$.

Our analysis of the lollipop method is based on the directed graph $D_{\lambda,H}$. We recommend consulting Figure 4 for intuition at this point. Recall from the proof of Lemma 10 that, in the directed graph $D_{\lambda,H}$, the vertex s has in-degree zero and every vertex has out-degree at least one. In particular, by traversing out-arcs from the vertex u in $D_{\lambda,H}$, and traversing the eventual directed cycle encountered, as well as traversing backwards to u from the directed cycle, in precise terms we observe that there exist vertices $w_0, w_1, \ldots, w_{d-1}$ with $(w_j, w_{(j+1) \bmod d}) \in E(D_{\lambda,H})$ for $j = 0, 1, \ldots, d-1$ as well as vertices $w'_0, w'_1, \ldots, w'_{d'}$ with $w_0 = w'_{d'}$, $w'_0 = u$, $\{w_0, w_1, \ldots, w_{d-1}\} \cap \{w'_0, w'_1, \ldots, w'_{d'-1}\} = \emptyset$, and $(w'_j, w'_{j+1}) \in E(D_{\lambda,H})$ for $j = 0, 1, \ldots, d'-1$. That is, the sequence $w'_0, w'_1, \ldots, w'_{d'}$ forms a directed path starting at the vertex $u = w'_0$ and ending at the vertex $w_{d'} = w_0$, which is on the directed cycle formed by the vertices $w_0, w_1, \ldots, w_{d-1}$ in $D_{\lambda,H}$; the directed cycle and the directed path intersect exactly at the vertex $w_{d'} = w_0$. In particular $d + d' \leq |L| = n/2$.

It will be convenient to introduce the following sequence of vertices visited on the traversal of $D_{\lambda,H}$ from u. For $i = 0, 1, \ldots, 2d' + d$, define

$$v_i = \begin{cases} w'_i & \text{for } i = 0, 1, \ldots, d'-1\,; \\ w_{i-d'} & \text{for } i = d', d'+1, \ldots, d'+d-1\,; \\ w'_{2d'+d-i} & \text{for } i = d'+d, d'+d+1, \ldots, 2d'+d\,. \end{cases} \tag{14}$$

We have $(v_i, v_{i+1}) \in E(D_{\lambda,H})$ for $i = 0, 1, \ldots, d'+d-1$; these arcs are precisely the arcs traversed forward. We have $(v_{i+1}, v_i) \in E(D_{\lambda,H})$ for $i = d'+d, d'+d+1, \ldots, 2d'+d-1$; these arcs are precisely the arcs traversed backward. For an arc $(\ell, \ell') \in E(D_{\lambda,H})$, let us write $r(\ell, \ell')$, $\rho(\ell, \ell')$, and $p'(\ell, \ell')$, respectively, for the unique $r \in R$, $\rho \in \{0, 1\}$, and $p' \in \{0, 1\}$ such that (11) holds. Also, let us write $p(\ell, \ell')$ for the minimum $p \in \{0, 1\}$ such that that (11) holds.

Let M^- be a matching with $n-1$ edges in F_λ such that the vertex $[t, 1]$ is left unmatched by M^-; we call such matchings *almost perfect* – indeed, any perfect matching M in F_λ has n edges. Also observe that the other vertex left unmatched by M^- is $[\ell, p]$ for some $\ell \in L$ and $p \in \{0, 1\}$. Recall the parity-and-port-changing traversal of M in the proof of Lemma 8 resulting in the Hamiltonian cycle $\mathrm{H}[M]$. Define a similar parity-and-port-changing traversal

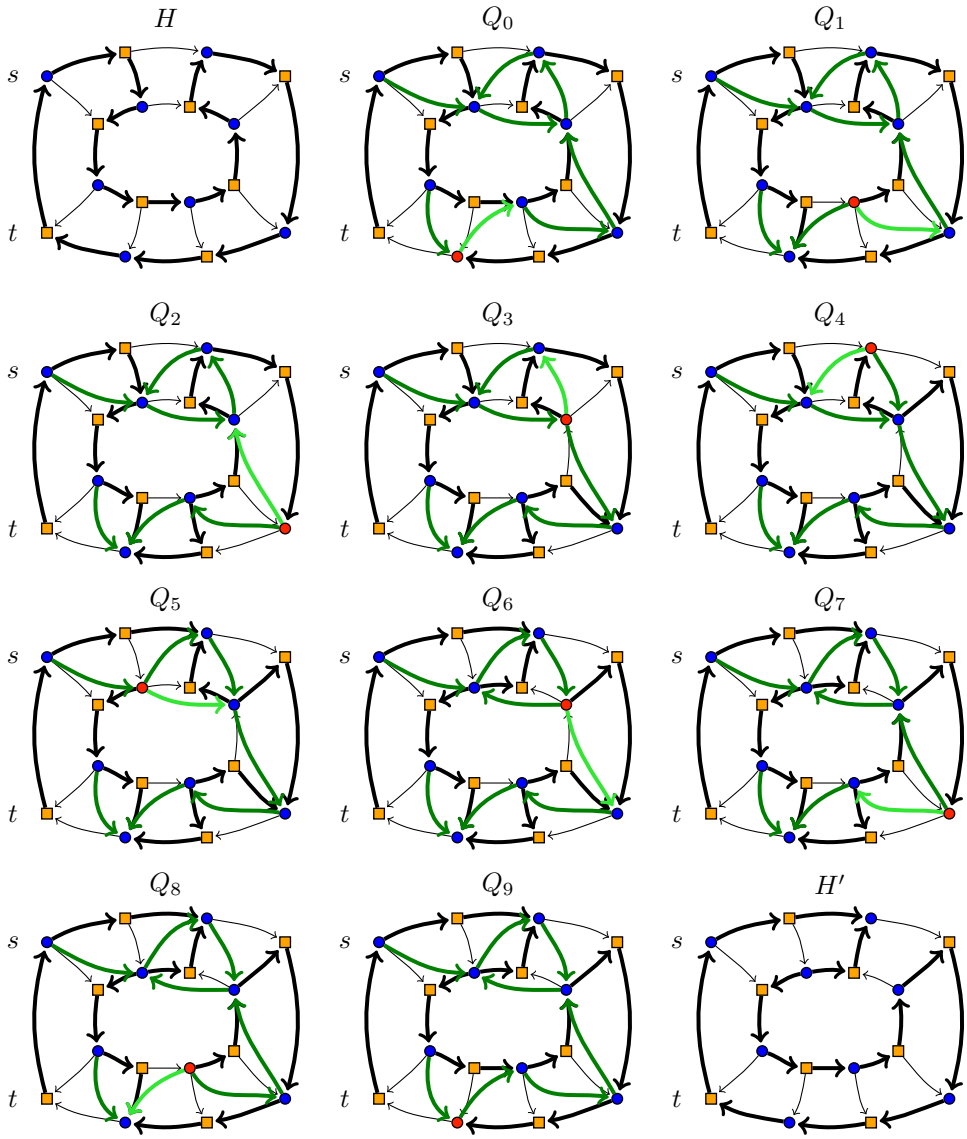


Figure 4 An example of the sequence of steps of Thomason's lollipop method in an n-vertex cubic bipartite planar graph viewed as a sequence of arc reversals in the directed graph $D_{\lambda,H}$. The given input G and H together with $e = \{s, t\}$ is displayed in the top left; the black arcs are oriented as in $\vec{G}_e$ obtained from Lemma 7 on input H. We display the initial Hamiltonian cycle H (top left) and the final Hamiltonian cycle H' (bottom right) obtained by the method, as well as the intermediate e-anchored Hamiltonian paths $Q_0, Q_1, \ldots, Q_9$ obtained in consecutive lollipop steps; the end-vertex of each Q_i is highlighted with red. The green arcs in Q_0 are the arcs of $D_{\lambda,H}$. Observe that each lollipop step from Q_i to Q_{i+1} can be understood as reversing the light-green arc in Q_i; the method terminates when the end-vertices of Q_0 and Q_{i+1} agree. By the structure of $D_{\lambda,H}$, we must have $i \leq n$; cf. Theorem 3.

of M^- by starting at the vertex $[\ell, \vec{p}]$ and observe by a similar argument as in the proof of Lemma 8 that this traversal defines an e-anchored Hamiltonian path $\mathrm{P}[M^-]$ from the vertex ℓ to the vertex t in G; in particular, observe that $\mathrm{P}[M^-]$ is e-anchored since by the structure of F_λ the almost perfect M^- must contain the edge $\{[s,0],[t,0]\}$.

We now proceed to characterize the e-anchored Hamiltonian paths $Q_0, Q_1, \ldots, Q_h$ using corresponding almost perfect matchings $M_0^-, M_1^-, \ldots, M_h^-$, and conclude that $h = 2d'+d \leq n$ in the process. For $i = 0, 1, \ldots, h$, let us write u_i for the end-vertex of Q_i other than t. Recalling that $Q_0 = P_e$ is constructed by deleting the edge $\{u,t\}$ from the Hamiltonian cycle H, let $p \in \{0,1\}$ be the port and $f \in E(F_\lambda)$ the edge with $f = \{[u,p],[t,1]\} \in M_H$. Take $M_0^- = M_H \setminus \{f\}$. In particular, we have $Q_0 = P_e = \mathrm{P}[M_0^-]$ and $u_0 = v_0 = u$. Let $p_0 = p$; we will fix values $p_i \in \{0,1\}$ for $i = 1, 2, \ldots, h$ as we progress in what follows.

We split the analysis into two ranges based on the parameter i. The first range corresponds to the forward-traversal of arcs in $D_{\lambda,H}$. For $i = 0, 1, \ldots, d' + d - 1$, we say an almost perfect matching M^- has *property* i if

- **(i)** $[v_i, p_i]$ is left unmatched by M^-; and
- **(ii)** we have $\{[v_j, p(v_j, v_{j+1})], [r(v_j, v_{j+1}), \rho(v_j, v_{j+1})]\} \in M^-$ and $\{[r(v_j, v_{j+1}), \rho(v_j, v_{j+1})], [v_{j+1}, p'(v_j, v_{j+1})]\} \notin M^-$ for all $0 \leq j \leq i-1$; and
- **(iii)** we have $\{[v_j, p(v_j, v_{j+1})], [r(v_j, v_{j+1}), \rho(v_j, v_{j+1})]\} \notin M^-$ and $\{[r(v_j, v_{j+1}), \rho(v_j, v_{j+1})], [v_{j+1}, p'(v_j, v_{j+1})]\} \in M^-$ for all $i \leq j \leq d' + d$.

The second range corresponds to the backward-traversal of arcs in $D_{\lambda,H}$. For $i = d'+d, d'+d+1, \ldots, 2d'+d$, we we say an almost perfect matching M^- has *property* i if

- **(i')** $[v_i, p_i]$ is left unmatched by M^-; and
- **(ii')** we have $\{[v_j, p(v_j, v_{j+1})], [r(v_j, v_{j+1}), \rho(v_j, v_{j+1})]\} \in M^-$ and $\{[r(v_j, v_{j+1}), \rho(v_j, v_{j+1})], [v_{j+1}, p'(v_j, v_{j+1})]\} \notin M^-$ for all $d' \leq j \leq d' + d - 1$ as well as for all $0 \leq j \leq 2d' + d - 1 - i$; and
- **(iii')** we have $\{[v_j, p(v_j, v_{j+1})], [r(v_j, v_{j+1}), \rho(v_j, v_{j+1})]\} \notin M^-$ and $\{[r(v_j, v_{j+1}), \rho(v_j, v_{j+1})], [v_{j+1}, p'(v_j, v_{j+1})]\} \in M^-$ for all $2d' + d - i \leq j \leq d' - 1$.

From previous observations and (11) we have that M_0^- satisfies property 0.

Let us now analyse the lollipop step mapping Q_i to Q_{i+1} one value $i = 0, 1, \ldots, d' + d - 1$ at a time. Suppose that there is an almost perfect matching M_i^- of F_λ that satisfies property i and that $Q_i = \mathrm{P}[M_i^-]$. In particular, we have $u_i = v_i$ by (i) and $Q_i = \mathrm{P}[M_i^-]$. We claim that the vertex $r(v_i, v_{i+1})$ is the unique degree-three vertex in the lollipop formed by the lollipop step transforming Q_i to Q_{i+1}. Observe by (iii) that $\{[r(v_i, v_{i+1}), \rho(v_i, v_{i+1})], [v_{i+1}, p'(v_i, v_{i+1})]\} \in M_i^-$, implying that $\{r(v_i, v_{i+1}), v_{i+1}\}$ is an edge in $Q_i = \mathrm{P}[M_i^-]$. Recalling that M_i^- is almost perfect, all vertices in $R \times \{0,1\}$ are matched, so $r(v_i, v_{i+1}) \in R$ is in fact adjacent to another vertex $\omega_i \neq v_{i+1}$ along an edge in $Q_i = \mathrm{P}[M_i^-]$. By (iii) and (11) we have $\{v_i, r(v_i, v_{i+1})\}$ is an edge in G but not in $Q_i = \mathrm{P}[M_i^-]$, and Q_i ends at v_i. Thus, $r(v_i, v_{i+1})$ is the unique degree-three vertex in the lollipop. Next, the lollipop step proceeds to delete an edge adjacent to the degree-three vertex $r(v_i, v_{i+1})$ in the lollipop. This edge is $\{v_{i+1}, r(v_i, v_{i+1})\}$ by the previous analysis. It follows that Q_{i+1} is obtained from Q_i by deleting $\{v_{i+1}, r(v_i, v_{i+1})\}$ and inserting $\{v_i, r(v_i, v_{i+1})\}$. Thus, Q_{i+1} ends at $u_{i+1} = v_{i+1}$. Define M_{i+1}^- by starting with M_i^- and deleting the edge $\{[r(v_i, v_{i+1}), \rho(v_i, v_{i+1})], [v_{i+1}, p'(v_i, v_{i+1})]\}$ as well as inserting the edge $\{[v_i, p(v_i, v_{i+1})], [r(v_i, v_{i+1}), \rho(v_i, v_{i+1})]\}$. Fix $p_{i+1} = p'(v_i, v_{i+1})$. From (i), (ii), and (iii) we have that M_{i+1}^- is an almost perfect matching that satisfies property $i+1$. Furthermore, $Q_{i+1} = \mathrm{P}[M_{i+1}^-]$.

The analysis of the lollipop step mapping Q_i to Q_{i+1} for $i = d'+d, d'+d+1, \ldots, 2d'+d-1$ is now similar, but relying on properties (i'), (ii'), (iii') instead. From the existence of an almost perfect matching M_i^- of F_λ that satisfies property i and $Q_i = \mathrm{P}[M_i^-]$, by a similar

analysis we conclude that there exists an almost perfect matching M^-_{i+1} of F_λ that satisfies property $i+1$ and $Q_{i+1} = \mathrm{P}[M^-_{i+1}]$. Since $v_{2d'+d} = u$ and u is adjacent to t in G, from (i') we conclude in particular that $P'_e = Q_{2d'+d}$ and thus $h = 2d' + d$. Since $2d' + d \leq n$, we have shown that the lollipop method terminates in at most n lollipop steps. ◀

We note that the algorithm implicit in the proof not only uses at most a linear number of lollipop steps, but also can be implemented with the guidance of $D_{\lambda,H}$ to run in linear time.

References

1 Takanori Akiyama, Takao Nishizeki, and Nobuji Saito. NP-completeness of the Hamiltonian cycle problem for bipartite graphs. *Journal of Information Processing*, 3:73–76, 1980.

2 Cristina Bazgan, Miklos Santha, and Zsolt Tuza. On the approximation of finding a(nother) Hamiltonian cycle in cubic Hamiltonian graphs. *J. Algorithms*, 31(1):249–268, 1999. doi:10.1006/jagm.1998.0998.

3 Andreas Björklund. Determinant sums for undirected Hamiltonicity. *SIAM J. Comput.*, 43(1):280–299, 2014. doi:10.1137/110839229.

4 Andreas Björklund, Petteri Kaski, and Jesper Nederlof. Another Hamiltonian cycle in bipartite pfaffian graphs, 2024. doi:10.48550/arXiv.2308.01574.

5 J. Bosák. Hamiltonian lines in cubic graphs. In *Theory of Graphs, International Symposium, Rome, July 1966*, pages 35–46. Gordon & Breach, New York, 1967.

6 Marcin Briański and Adam Szady. A short note on graphs with long Thomason chains. *Discrete Math.*, 345(1):112624, 2022. doi:10.1016/j.disc.2021.112624.

7 Kevin Buchin, Christian Knauer, Klaus Kriegel, André Schulz, and Raimund Seidel. On the number of cycles in planar graphs. In Guohui Lin, editor, *Computing and Combinatorics, 13th Annual International Conference, COCOON 2007, Banff, Canada, July 16-19, 2007, Proceedings*, volume 4598 of *Lecture Notes in Computer Science*, pages 97–107. Springer, 2007. doi:10.1007/978-3-540-73545-8_12.

8 Kathie Cameron. Thomason's algorithm for finding a second hamiltonian circuit through a given edge in a cubic graph is exponential on Krawczyk's graphs. *Discrete Math.*, 235(1):69–77, 2001. Chech and Slovak 3. doi:10.1016/S0012-365X(00)00260-0.

9 G.L. Chia and Siew-Hui Ong. Hamilton cycles in cubic graphs. *AKCE Int. J. Graphs Comb.*, 4(3):251–259, 2007. doi:10.1080/09728600.2007.12088840.

10 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. Van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Trans. Algorithms*, 18(2), 2022. doi:10.1145/3506707.

11 Marcelo H. de Carvalho, Cláudio L. Lucchesi, and U. S.R. Murty. On the number of dissimilar pfaffian orientations of graphs. *RAIRO - Theoretical Informatics and Applications*, 39(1):93–113, 2005. doi:10.1051/ita:2005005.

12 Argyrios Deligkas, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Exact and approximate algorithms for computing a second Hamiltonian cycle. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 27:1–27:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.27.

13 Maximilian Gorsky, Raphael Steiner, and Sebastian Wiederrecht. Matching theory and Barnette's conjecture. *Discrete Math.*, 346(2):113249, 2023. doi:10.1016/j.disc.2022.113249.

14 Haddadan, Arash. Finding a second Hamiltonian cycle in Barnette graphs. Master's thesis, University of Waterloo, 2015. URL: http://hdl.handle.net/10012/9630.

15 Derek A. Holton, Bennet Manvel, and Brendan D. McKay. Hamiltonian cycles in cubic 3-connected bipartite planar graphs. *J. Combin. Theory Ser. B*, 38:279–297, 1985. doi:10.1016/0095-8956(85)90072-3.

16 Pieter W. Kasteleyn. Graph theory and crystal physics. In *Graph Theory and Theoretical Physics*, pages 43–110. Academic Press, London, 1967.

17 A.K. Kelmans. Constructions of cubic bipartite 3-connected graphs without Hamiltonian cycles. *Amer. Math. Soc. Transl. Ser. 2*, 158:127–140, 1994. doi:10.1090/trans2/158/12.

18 Shiva Kintali. Finding a second Hamilton circuit, 2009. URL: https://kintali.wordpress.com/2009/07/25/finding-a-second-hamilton-circuit/.

19 Adam Krawczyk. The complexity of finding a second Hamiltonian cycle in cubic graphs. *J. Comput. System Sci.*, 58(3):641–647, 1999. doi:10.1006/jcss.1998.1611.

20 Charles H. C. Little. An extension of Kasteleyn's method of enumerating the 1-factors of planar graphs. In *Combinatorial Mathematics (Proc. 2nd Australian Conf., Univ. Melbourne, 1973)*, number 403 in Lecture Notes in Mathematics, pages 63–72. Springer, Berlin, 1974.

21 Charles H. C. Little. A characterization of convertible (0, 1)-matrices. *J. Combinatorial Theory*, 18:187–208, 1975. doi:10.1016/0095-8956(75)90048-9.

22 William McCuaig. Pólya's permanent problem. *Electron. J. Combin.*, 11(1), 2004. doi:10.37236/1832.

23 Serguei Norine. *Matching Structure and Pfaffian Orientations of Graphs*. PhD thesis, Georgia Institute of Technology, 2005.

24 Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. System Sci.*, 48(3):498–532, 1994. doi:10.1016/S0022-0000(05)80063-7.

25 Neil Robertson, Paul D. Seymour, and Robin Thomas. Permanents, Pfaffian orientations, and even directed circuits. *Ann. of Math. (2)*, 150(3):929–975, 1999. doi:10.2307/121059.

26 John Sheehan. The multiplicity of Hamiltonian circuits in a graph. In *Recent Advances in Graph Theory (Proc. 2nd Czechoslovak Sympos., Prague, 1974)*, pages 477–480. Academia, Prague, 1975.

27 Andrew G. Thomason. Hamiltonian cycles and uniquely edge colourable graphs. *Ann. Discrete Math.*, 3:259–268, 1978. doi:10.1016/S0167-5060(08)70511-9.

28 Carsten Thomassen. On the number of Hamiltonian cycles in bipartite graphs. *Combinatorics, Probability and Computing*, 5:437–442, 1996. doi:10.1017/S0963548300002182.

29 Carsten Thomassen. Chords of longest cycles in cubic graphs. *J. Combin. Theory Ser. B*, 71(2):211–214, 1997. doi:10.1006/jctb.1997.1776.

30 William T. Tutte. On Hamiltonian circuits. *J. London Math. Soc.*, 21:98–101, 1946. doi:10.1112/jlms/s1-21.2.98.

31 William T. Tutte, editor. *Recent Progress in Combinatorics: Proceedings. New York*. Academic Press, 1969.

32 Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, Upper Saddle River, NJ, 1996.

33 Liang Zhong. The complexity of Thomason's algorithm for finding a second Hamiltonian cycle. *Bull. Aust. Math. Soc.*, 98(1):18–26, 2018. doi:10.1017/S0004972718000242.

The Discrepancy of Shortest Paths

Greg Bodwin
Computer Science and Engineering, University of Michigan, Ann Arbor, MI, USA

Chengyuan Deng
Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Jie Gao
Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Gary Hoppenworth
Computer Science and Engineering, University of Michigan, Ann Arbor, MI, USA

Jalaj Upadhyay
Management Science and Information Systems, Rutgers University, Piscataway, NJ, USA

Chen Wang
Department of Computer Science, Rice University, Houston, TX, USA
Computer Science and Engineering, Texas A&M University, College Station, TX, USA

Abstract

The *hereditary discrepancy* of a set system is a quantitative measure of the pseudorandom properties of the system. Roughly speaking, hereditary discrepancy measures how well one can 2-color the elements of the system so that each set contains approximately the same number of elements of each color. Hereditary discrepancy has numerous applications in computational geometry, communication complexity and derandomization. More recently, the hereditary discrepancy of the set system of *shortest paths* has found applications in differential privacy [Chen et al. SODA 23].

The contribution of this paper is to improve the upper and lower bounds on the hereditary discrepancy of set systems of unique shortest paths in graphs. In particular, we show that any system of unique shortest paths in an undirected weighted graph has hereditary discrepancy $O(n^{1/4})$, and we construct lower bound examples demonstrating that this bound is tight up to polylog n factors. Our lower bounds hold even for planar graphs and bipartite graphs, and improve a previous lower bound of $\Omega(n^{1/6})$ obtained by applying the trace bound of Chazelle and Lvov [SoCG'00] to a classical point-line system of Erdős.

As applications, we improve the lower bound on the additive error for differentially-private all pairs shortest distances from $\Omega(n^{1/6})$ [Chen et al. SODA 23] to $\widetilde{\Omega}(n^{1/4})$, and we improve the lower bound on additive error for the differentially-private all sets range queries problem to $\widetilde{\Omega}(n^{1/4})$, which is tight up to polylog n factors [Deng et al. WADS 23].

2012 ACM Subject Classification Theory of computation → Shortest paths; Theory of computation → Computational geometry

Keywords and phrases Discrepancy, hereditary discrepancy, shortest paths, differential privacy

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.27

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2401.15781` [9]

Funding *Greg Bodwin and Gary Hoppenworth*: Supported by NSF:AF 2153680.
Jie Gao and Chengyuan Deng: Supported partially by NSF DMS-2311064, DMS-2220271, IIS-2229876, CCF-2208663, CCF-2118953.
Jalaj Upadhyay: Funded by Decanal Research Grant number 302918.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 27; pp. 27:1–27:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In graph algorithms, a fundamental problem is to efficiently compute distance or shortest path information of a given input graph. Over the last decade or so, the community has increasingly sought a principled understanding of the *combinatorial structure* of shortest paths, with the goal to exploit this structure in algorithm design. That is, in various graph settings, we can ask:

> *What notable structural properties hold for* ***shortest*** *path systems, that do not necessarily hold for* ***arbitrary*** *path systems?*

The following are a few of the major successes of this line of work:

- An extremely popular strategy in the literature is to use *hitting sets*, in which we (often randomly) generate a set of nodes S and argue that it will hit a shortest path for every pair of nodes that are sufficiently far apart. Hitting sets rarely exploit any structure of shortest paths, as evidenced by the fact that most hitting set algorithms generalize immediately to arbitrary set systems. However, they have inspired a successful line of work into graphs of bounded *highway dimension* [1, 6, 7]; very roughly, these are graphs whose shortest paths admit unusually efficient hitting sets of a certain kind.
- Shortest paths exhibit the notable structural property of *consistency*, i.e., any subpath of a shortest path is itself a shortest path. This fact is used throughout the literature on graph algorithms [21, 22, 8], including e.g. in the classic Floyd-Warshall algorithm for All-Pairs Shortest Paths. A recent line of work has sought to characterize the additional structure exhibited by shortest path systems, beyond consistency [8, 21, 19, 20, 17, 4, 2].
- Planar graphs have received special attention within this research program, and planar shortest path systems carry some notable additional structure. For example, it is known that planar shortest paths have unusually efficient tree coverings [5, 11], and that their shortest paths can be compressed into surprisingly small space [12, 13]. Shortest path algorithms also often benefit from more general structural facts about planar graphs, such as separator theorems [29, 28].

The main result of this paper is a new structural separation between shortest path systems and arbitrary path systems, expressed through the lens of *discrepancy theory*. We will come to formal definitions of discrepancy in just a moment, but at a high level, discrepancy has been described as a quantitative measure of the combinatorial pseudorandomness of a discrete system [18], and it has widespread applications in discrete and computational geometry, random sampling and derandomization, communication complexity, and much more[1]. We will show the following:

▶ **Theorem 1** (Main Result, Informal). *The discrepancy of unique shortest path systems in weighted graphs is inherently smaller than the discrepancy of arbitrary path systems in graphs.*

This separation between unique shortest paths and arbitrary paths is due to the structural property of *consistency* of unique shortest path systems, which is well-studied in the literature [21, 22, 8].

[1] We refer to the excellent textbooks of Alexander, Beck, and Chen [3], Chazelle [14], Matoušek [33] for discussion and further applications.

Our results can be placed within a larger context of prior work in computational geometry. A classical topic in this area is to determine the discrepancy of incidence structures between points and geometric range spaces such as axis-parallel rectangles, half-spaces, lines, and curves (cf. [14, Section 1.5]). These results have been used to show lower bounds for geometric range searching [37, 34].

Indeed, systems of unique shortest paths in graphs capture some of the geometric range spaces studied in prior work. For instance, arrangements of straight lines in Euclidean space can be interpreted as systems of unique shortest paths in an associated graph, implying a relation between the discrepancies of these two set systems. This connection has recently found applications in the study of differential privacy on shortest path distance and range query algorithms [16, 23].

More generally, discrepancy on graphs have also found applications in proving tight lower bounds on answering cut queries on graphs [26, 32]. We provide a detailed literature review for discrepancy on graphs in the full version of our paper [9]. The full version of our paper further discusses the connection between our results and the discrepancy of arrangements of curves.

1.1 Formal Definitions of Discrepancy

We first collect the basic definitions needed to understand this paper.

▶ **Definition 2** (Edge and Vertex Incidence Matrices). *Given a graph $G = (V, E)$ and a set of paths Π in G, the associated* vertex incidence matrix *is given by $A \in \mathbb{R}^{|\Pi| \times |V|}$, where for each $v \in V$ and $\pi \in \Pi$ the corresponding entry is*

$$A_{\pi,v} = \begin{cases} 1 & \text{if } v \in \pi \\ 0 & \text{if } v \notin \pi. \end{cases}$$

The associated edge incidence matrix *is given by $A \in \mathbb{R}^{|\Pi| \times |E|}$, where for each $e \in E$ and $\pi \in \Pi$ the corresponding entry is*

$$A_{\pi,e} = \begin{cases} 1 & \text{if } e \in \pi \\ 0 & \text{if } e \notin \pi. \end{cases}$$

▶ **Definition 3** (Discrepancy and Hereditary Discrepancy). *Given a matrix $A \in \mathbb{R}^{m \times n}$, its discrepancy is the quantity*

$$\operatorname{disc}(A) = \min_{x \in \{1,-1\}^n} \|Ax\|_\infty.$$

Its hereditary discrepancy is the maximum discrepancy of any submatrix A_Y obtained by keeping all rows but only a subset $Y \subseteq [n]$ of the columns; that is,

$$\operatorname{herdisc}(A) = \max_{Y \subseteq [n]} \operatorname{disc}_v(A_Y).$$

For a system of paths Π in a graph G, we will write $\operatorname{disc}_v(\Pi), \operatorname{herdisc}_v(\Pi)$ to denote the (hereditary) discrepancy of its vertex incidence matrix, and $\operatorname{disc}_e(\Pi), \operatorname{herdisc}_e(\Pi)$ to denote the (hereditary) discrepancy of its edge incidence matrix.

For intuition, the vertex discrepancy of a system of paths Π can be equivalently understood as follows. Suppose that we color each node in G either red or blue, with the goal to balance the red and blue nodes on each path as evenly as possible. The discrepancy associated to that particular coloring is the quantity

$$\max_{\pi \in \Pi} \Big| \; |\{v \in \pi \;\mid\; v \text{ colored red}\}| - |\{v \in \pi \;\mid\; v \text{ colored blue}\}| \; \Big|.$$

The discrepancy of the system Π is the minimum possible discrepancy over all colorings. The hereditary discrepancy is the maximum discrepancy taken over all *induced path subsystems* Π' of Π; that is, Π' is obtained from Π by selecting zero or more vertices from G, deleting these vertices, and deleting all instances of these vertices from all paths.[2] We may delete nodes from the middle of some paths $\pi \in \Pi$, in which case Π' may no longer be a system of paths in G, but rather a system of paths in some other graph G' with fewer nodes and some additional edges. Nonetheless, its vertex incidence matrix and therefore $\text{herdisc}_v(\Pi')$ remain well-defined with respect to this new graph G'. Edge discrepancy can be understood in a similar way, coloring edges rather than vertices.

1.2 Our Results

Our main result is an upper and lower bound on the hereditary discrepancy of unique shortest path systems in weighted graphs, which match up to hidden polylog n factors.

▶ **Theorem 4** (Main Result).

- **(Upper Bound).** *For any n-node undirected weighted graph G with a unique shortest path between each pair of nodes, there exists a polynomial-time algorithm that finds a coloring for the system of shortest paths Π such that:*

$$\text{herdisc}_v(\Pi) \leq \widetilde{O}(n^{1/4}) \qquad \textit{and} \qquad \text{herdisc}_e(\Pi) \leq \widetilde{O}(n^{1/4}).$$

- **(Lower Bound).** *There are examples of n-node undirected weighted graphs G with a unique shortest path between each pair of nodes in which this system of shortest paths Π has $\text{herdisc}_v(\Pi) \geq \widetilde{\Omega}(n^{1/4})$ and $\text{herdisc}_e(\Pi) \geq \widetilde{\Omega}(n^{1/4})$. In fact, in these lower bound examples we can take G to be planar or bipartite.*

This theorem has immediate applications in differential privacy; we refer to Theorem 6 discussed below. We can strengthen the hereditary discrepancy lower bound into a vertex (non-hereditary) discrepancy lower bound in the undirected and directed settings. We leave open whether our lower bound extends to (non-hereditary) edge discrepancy as well, and to vertex or edge discrepancy of planar graphs. We refer to Table 1 for a list of our results in these settings.

Table 1 Overview of vertex/edge (hereditary) discrepancy on general graphs and special families of graph: tree, bipartite and planar graphs. Here n is the number of vertices of the graph and m is the number of edges. D is the graph diameter or the longest number of hops of paths considered.

		Tree	Bipartite	Planar	Undirected Graph	Directed Graph
V	disc	$\Theta(1)$	$\Theta(1)$	$O(n^{1/4})$	$\widetilde{\Theta}(n^{1/4})$	$\widetilde{\Theta}(n^{1/4})$
	herdisc	$\Theta(1)$	$\widetilde{\Theta}(n^{1/4})$	$\widetilde{\Theta}(n^{1/4})$	$\Omega(n^{1/6})$[15] $\rightarrow \widetilde{\Theta}(n^{1/4})$	$\widetilde{\Theta}(n^{1/4})$
E	disc	$\Theta(1)$	$\Theta(1)$	$O(n^{1/4})$	$O(n^{1/4})$	$\min\left\{O(m^{1/4}), \widetilde{O}(D^{1/2})\right\}$
	herdisc	$\Theta(1)$	$\widetilde{\Theta}(n^{1/4})$	$\widetilde{\Theta}(n^{1/4})$	$\Omega(n^{1/6})$[15] $\rightarrow \widetilde{\Theta}(n^{1/4})$	$\widetilde{\Omega}(n^{1/4})$

[2] In the coloring interpretation, hereditary discrepancy allows a different choice of coloring for each subsystem Π', rather than fixing a coloring for Π and considering the induced coloring on each Π'.

The upper bound in Theorem 4 is constructive and algorithmic; that is, we provide an algorithm that colors vertices (resp. edges) of the input graph to achieve vertex (resp. edge) discrepancy $\widetilde{O}(n^{1/4})$ on its shortest paths (or on a given subsystem of its shortest paths). Notably, Theorem 4 should be contrasted with the fact that the maximum possible discrepancy of *any* simple path system of polynomial size in a general graph is known to be $\widetilde{\Theta}(n^{1/2})$.[3] In fact, the lower bound on discrepancy (as well as hereditary discrepancy) for a grid graph for a polynomial number of *simple* paths can be $\Omega(\sqrt{n})$ (see the full version of our paper [9] for a proof and more discussion on grid graphs). Thus, Theorem 4 represents a concrete separation between unique shortest path systems and general path systems.

The main open question that we leave in this work is on the hereditary edge discrepancy of shortest paths in *directed* weighted graphs. We show the following:

▶ **Theorem 5.** *For any n-node, m-edge directed weighted graph G with a unique shortest path between each pair of nodes, the system of shortest paths Π satisfies*

$$\text{herdisc}_v(\Pi) \leq O(n^{1/4}) \qquad \textit{and} \qquad \text{herdisc}_e(\Pi) \leq O(m^{1/4}).$$

Lower bounds in the undirected setting immediately apply to the directed setting as well, and so this essentially closes the problem for directed hereditary *vertex* discrepancy. It is an interesting open problem whether the bound for directed hereditary *edge* discrepancy can be improved to $\widetilde{O}(n^{1/4})$ as well.

Applications to Differential Privacy. One application of our discrepancy lower bound on unique shortest paths is in differential privacy (DP) [24, 25]. An algorithm is differentially private if its output distributions are relatively close regardless of whether an individual's data is present in the data set. More formally, for two databases Y and Y' that are identical except for one data entry, a randomized algorithm $\mathcal{M}$ is (ε, δ) differentially private if for any measurable set A in the range of $\mathcal{M}$, $\Pr[\mathcal{M}(Y) \in A] \leq e^{\varepsilon} \Pr[\mathcal{M}(Y') \in A] + \delta$.

The topic of discrepancy of paths on a graph is related to two problems already studied in differential privacy: *All Pairs Shortest Distances (APSD)* ([16, 27, 36]) and *All Sets Range Queries (ASRQ)* ([23]), both assuming the graph topology is public. In APSD problem, the edge weights are not publicly known. A query in APSD is a pair of vertices $(u, v) \in V \times V$ and the answer is the shortest distance between u and v. In contrast, in ASRQ problem, the edge weights are assumed to be known, and every edge also has a private *attribute*. Here, the range is defined by the shortest path between two vertices (based on publicly known edge weights). The answer to the query $(u, v) \in V \times V$ then is the sum of private attributes along the shortest path. In what follows, we give a high-level argument for the lower bound on DP-APSD problem; the lower bound of $\widetilde{\Omega}(n^{1/4})$ for the DP-ASRQ problem also follows nearly the same arguement (see the full version of our paper [9] for details).

Chen et al. [16] showed that DP-APSD can be formulated as a linear query problem. In this setting, we are given a vertex incidence matrix A of the $\binom{n}{2}$ shortest paths of a graph and a vector x of length n and asked to output Ax. They show that the hereditary discrepancy of the matrix A provides a lower bound on the ℓ_∞ error for any (ε, δ)-DP mechanism for this problem. With this argument, our new discrepancy lower bound immediately implies:

[3] A path system is *simple* if no individual path repeats nodes. The upper bound of $\widetilde{O}(n^{1/2})$ follows by coloring the nodes randomly and applying standard Chernoff bounds. The lower bound is nontrivial and follows from an analysis of the Hadamard matrix; see [14], Section 1.5.

▶ **Theorem 6** (Informal version of Corollaries 7.1 and F.1 in [9]). *The (ε, δ)-DP APSD problem and (ε, δ)-DP ASRQ problem require additive error at least $\widetilde{\Omega}(n^{1/4})$.*

The best known additive error bound for the DP-ASRQ problem is $\widetilde{O}(n^{1/4})$ [23], which, by Theorem 6, is tight up to a polylog(n) factor. Prior to this work, the only known lower bounds for DP-ASRQ and DP-APSD were from a point-line system with hereditary discrepancy of $\Omega(n^{1/6})$ [16]. The best known additive error upper bound for DP-APSD is $\widetilde{O}(n^{1/2})$ [16, 27]. Closing this gap remains an interesting open problem.

In addition to differential privacy, our hereditary discrepancy results also have implications for matrix analysis. In short, we can show that the factorization norm of the shortest path incidence matrix is $\widetilde{\Theta}(n^{1/4})$. We delay a detailed discussion to the full version of our paper [9].

1.3 Our Techniques

We will overview our upper and lower bounds on discrepancy separately.

Upper Bound Techniques. A folklore structural property of unique shortest paths is *consistency*. Formally, a system of undirected paths Π is consistent if for any two paths π_1, π_2, their intersection $\pi_1 \cap \pi_2$ is a (possibly empty) contiguous subpath of each. It is well known that, for any undirected graph $G = (V, E, w)$ with unique shortest paths, its system of shortest paths Π is consistent. An analogous fact holds for directed graphs. Our discrepancy upper bounds will actually apply to *any* consistent system of paths – not just those that arise as unique shortest paths in a graph.

We give our upper bounds on the discrepancy of consistent systems in two steps. First, we prove the existence of a low-discrepancy coloring using a standard application of *primal shatter functions* (see the the full version of our paper [9] for a definition). For consistent paths, the primal shatter function has degree two in both directed and undirected graphs. This immediately gives us an upper bound of $O(n^{1/4})$ for vertex discrepancy and $O(m^{1/4})$ for edge discrepancy (since edge discrepancy is defined on a ground set of m edges in the graph G).

When the graph is dense, this upper bound on edge discrepancy deteriorates, becoming trivial when $m = \Theta(n^2)$. We thus present a second proof of $\tilde{O}(n^{1/4})$ for *both* vertex and edge discrepancy, which explicitly constructs a low-discrepancy coloring. This improves the bound for vertex discrepancy by polylogarithmic factors and edge discrepancy by polynomial factors. The main idea in this construction is to adapt the *path cover* technique, used in the recent breakthrough on shortcut sets [30]. That is, we start by finding a small base set of roughly $n^{1/2}$ node-disjoint shortest paths in the distance closure of the graph. These paths have the property that any other shortest path π in the graph contains at most $O(n^{1/2})$ nodes that are not in any paths in the base set. We then color *randomly*, as follows:

- For every node that is not contained in any path in the base set, we assign its color randomly. Thus, applying concentration bounds, the contribution of these nodes to the discrepancy of π will be bounded by $\pm\widetilde{O}(n^{1/4})$.
- For every path in the base set, we choose the color of the first node in the path at random, and then alternate colors along the path after that. Then we can argue that by consistency, the nodes in each base path randomly contribute $+1$ or -1 (or 0) to the discrepancy of π (see Figure 1 for a visualization). Since there are only $n^{1/2}$ paths in the base set, we may again apply concentration bounds to argue that the contribution to discrepancy from these base paths will only be $\pm\widetilde{O}(n^{1/4})$.

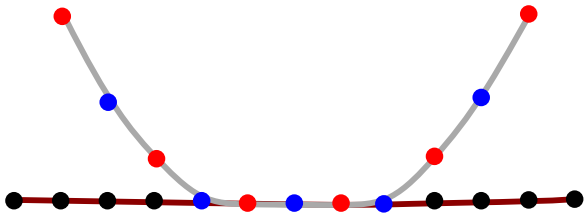


Figure 1 If we color the nodes of a unique shortest path with alternating colors, then its nodes will contribute discrepancy 0, +1, or −1 to all unique shortest paths that intersect it.

Summing together these two parts, we obtain a bound of $\widetilde{O}(n^{1/4})$ on discrepancy, which holds with high probability. We can translate this to a bound on *hereditary* discrepancy using the fact that consistency is a hereditary property of path systems.

Lower Bound Techniques. For lower bounds, we apply the trace bound of [15] on hereditary discrepancy together with an explicit graph construction [10] that was recently proposed as a lower bound against *hopsets* in graphs. An (exact) hopset of a graph G with hopbound β is a small set of additional edges H in the distance closure of G, such that every pair of nodes has a shortest path in $G \cup H$ containing at most β edges.

Until recently, the state-of-the-art hopset lower bounds were achieved using a point-line construction of Erdős [35], which had n points and n lines in $\mathbb{R}^2$ with each point staying on $\Theta(n^{1/3})$ lines and each line going through $\Theta(n^{1/3})$ points. This point-line system also implies tight lower bounds for the Szemerédi-Trotter theorem and the discrepancy of arrangements of lines in the plane [15], as well as the previous state-of-the-art lower bound on the discrepancy of unique shortest paths.

This point-line construction can be associated with a graph that possesses useful properties derived from geometry. If edges in this graph are weighted by Euclidean distance, then the paths in the graph corresponding to straight lines are unique shortest paths by design. On the other hand, two such shortest paths (along straight lines) only intersect at most once.

Recently, a construction in Bodwin and Hoppenworth [10] obtained stronger hopset lower bounds with a different geometric graph construction, which still took place in $\mathbb{R}^2$ but allowed shortest paths to have many vertices/edges in common. We show that this construction can be repurposed to derive a stronger lower bound of $\widetilde{\Omega}(n^{1/4})$ on vertex hereditary discrepancy, by applying the trace bound of [15]. Combined with our upper bounds, this substantially improves our understanding of the discrepancy of unique shortest paths.

The above upper and lower bounds are for general graphs. Naturally, one can ask if we have better bounds for special families of graphs. We further show that the lower bounds remain the same for two interesting families: planar graphs and bipartite graphs. The lower bound construction mentioned above is not planar, and so this requires some additional work. A natural attempt is to restore planarity by adding vertices to the construction wherever two edges cross. However, this comes at a cost of an increase in the number of vertices, and also with a potential danger of altering the shortest paths. In the full version of our paper [9], we first show that the number of crossings is not too much higher than n. Then, by carefully changing the weights of the edges and by exploiting the geometric properties of the construction, we show that the topology and incidence of shortest paths are not altered. For bipartite graphs, although the vertex discrepancy can be made very low – by coloring the vertices on one side +1 and vertices on the other side −1 – the hereditary discrepancy can be as high as the general graph setting. Specifically, we show a 2-lift of any graph G to a bipartite graph which essentially keeps the same hereditary discrepancy. Details can be found in the full version of our paper [9].

2 Preliminaries

A *path system* is a pair $S = (V, \Pi)$ where V is a ground set of nodes and Π is a set of vertex sequences called *paths*. Each path may contain at most one instance of each node. We now formally define consistency, a structural property of unique shortest paths that will be useful.

▶ **Definition 7.** *A path system $S = (V, \Pi)$ is consistent if no two paths in S intersect, split apart, and then intersect again later. Formally:*

- *In the undirected setting, consistency means that for all $u, v \in V$ and all $\pi_1, \pi_2 \in \Pi$ such that $u, v \in \pi_1 \cap \pi_2$, we have that $\pi_1[u, v] = \pi_2[u, v]$, i.e., the intersection of π_1 and π_2 is a contiguous subpath (subsequence) of π_1 and π_2.*
- *In the directed setting, consistency means that for all $u, v \in V$ and all $\pi_1, \pi_2 \in \Pi$ such that u **precedes** v in both π_1 and π_2, we have that $\pi_1[u, v] = \pi_2[u, v]$.*

In every weighted graph for which all pairs shortest paths exist (i.e. no negative cycles), we can represent all-pairs shortest paths using a consistent path system. In particular, if all shortest paths are *unique*, then consistency is implied immediately.

We will investigate the combinatorial discrepancy of path systems (V, Π). Usually, we will assume that $|V| = n$ and $|\Pi|$ is polynomial in n. We define a vertex coloring $\chi : V \mapsto \{-1, 1\}$ and define the *discrepancy* of Π as

$$\operatorname{disc}(\Pi) = \min_{\chi} \chi(\Pi), \quad \text{where } \chi(\Pi) = \max_{\pi \in \Pi} |\chi(\pi)|, \ \chi(\pi) = \sum_{v \in \pi} \chi(v).$$

Using a random coloring χ, we can guarantee that for all paths $\pi \in \Pi$ [14]:

$$|\chi(\pi)| \leq \sqrt{2|\pi| \ln(4|\Pi|)}.$$

This immediately provides a few observations.

▶ **Observation 8.** *When Π is a set of paths with size polynomial in n, then $\operatorname{disc}(\Pi) = O(\sqrt{n \log n})$. This bound is true even for paths that are possibly non-consistent.*

▶ **Observation 9.** *When the longest path in Π has D vertices we have $\operatorname{disc}(\Pi) = O(\sqrt{D \log n})$. Thus, for graphs that have a small diameter (e.g., small world graphs), the discrepancy of shortest paths is automatically small.*

Hereditary discrepancy is a more robust measure of the complexity of a path system (V, Π), defined as $\operatorname{herdisc}(\Pi) = \max_{Y \subseteq V} \operatorname{disc}(\Pi|_Y)$, where $\Pi|_Y$ is the collection of sets of the form $\pi \cap Y$ with $\pi \in \Pi$. Clearly, $\operatorname{herdisc}(\Pi) \geq \operatorname{disc}(\Pi)$. Sometimes the discrepancy of a set system may be small while the hereditary discrepancy is large [14]. Thus in the literature, we often talk about lower bounds on the hereditary discrepancy.

Now that we have defined vertex and edge (hereditary) discrepancy, one may wonder if there is an underlying relationship between vertex and edge (hereditary) discrepancy since they share the same bounds in most settings presented in Table 1. The following observation shows that vertex discrepancy bounds directly imply bounds on edge discrepancy.

▶ **Observation 10.** *Denote by $\operatorname{disc}(n)$ (and $\operatorname{herdisc}(n)$) the maximum discrepancy (minimum hereditary discrepancy, respectively) of a consistent path system of a (undirected or directed) graph of n vertices. We have that*

1. *Let $g(x)$ be a non-decreasing function. If $\operatorname{herdisc}_v(n) \geq g(n)$, then $\operatorname{herdisc}_e(n) \geq g(n/2)$.*
2. *Let $f(x)$ be a non-decreasing function. If $\operatorname{disc}_v(n) \leq f(n)$, then $\operatorname{disc}_e(m) \leq f(m)$.*

The proof of Observation 10 is deferred to the full version of our paper [9]. We also use some technical tools from discrepancy theory and statistics. For details please refer to the full version of our paper [9].

3 Undirected Graphs: Lower Bound and Explicit Colorings

We now discuss the main result (Theorem 4). We first show in Section 3.1 a hereditary discrepancy lower bound of $\Omega(n^{1/4}/\sqrt{\log n})$ for both edge and vertex discrepancy in general undirected graphs. Then in Section 3.2 we present a vertex coloring achieving hereditary discrepancy of $\widetilde{O}(n^{1/4})$. Finally, we present an explicit edge coloring with the same hereditary discrepancy bound in Section 3.3.

3.1 Lower Bound

As suggested by Observation 10, we focus on the vertex hereditary discrepancy, and our goal is to prove the following statement (Theorem 11). In Theorem 10 of the full version of our paper [9], we show that this theorem implies the same lower bound on (non-hereditary) vertex discrepancy as well.

▶ **Theorem 11.** *There are examples of n-vertex undirected weighted graphs G with a unique shortest path between each pair of vertices in which this system of shortest paths Π has*

$$\text{herdisc}_v(\Pi) \geq \Omega(n^{1/4}/\sqrt{\log n}).$$

To obtain the lower bound, we employ the new graph construction by [10], which shows that any exact hopset with $O(n)$ edges must have at least $\widetilde{\Omega}(n^{1/2})$ hop diameter. Despite seeming unrelated, this construction also sheds light on our problem. Another technique we use to show the hereditary discrepancy lower bound is the trace bound [31] (and restated in the full version of our paper [9]). In the following proof section, we first summarize the construction related to our objective, then show the calculation using the trace bound that leads to our lower bound.

Proof. The key properties of the graph construction in [10] (see also Section 5 of [9]) that we need can be summarized in the following lemma.

▶ **Lemma 12** (Lemma 1 of [10]). *For any $p \in [1, n^2]$, there is an infinite family of n-node undirected weighted graphs $G = (V, E, w)$ and sets Π of p paths in G such that*

- *G has $\ell = \Theta\left(\frac{n}{\sqrt{p \log n}}\right)$ layers. Each path in Π starts in the first layer, ends in the last layer, and contains exactly one node in each layer.*
- *Each path in Π is the unique shortest path between its endpoints in G.*
- *For any two nodes $u, v \in V$, there are at most $\frac{\ell}{h(u,v)}$ paths in Π that contain both u and v, where $h(u, v)$ is the hopdistance (number of edges on the shortest path) between u and v in G and $1 \leq h(u, v) \leq \ell$.*
- *Each node $v \in V$ lies on at most $O\left(\frac{\ell p}{n}\right)$ distinct paths in Π.*

We will make use of the shortest path vertex incidence matrix of this graph. Recall that hereditary discrepancy considers the sub-incidence matrix induced by columns corresponding to a set of vertices. We select the set of vertices occurring in the paths in Π, and show it leads to hereditary discrepancy at least $\Omega(n^{1/4}/\sqrt{\log n})$. Specifically, take A as the incidence matrix such that each row corresponds to one path in Π. A has dimension $p \times n$ where n is the number of vertices in G and the (i, j)-th entry of A is 1 is the vertex j is in the path i.

Now define $M = A^T A$. Recall that $\text{tr}(M)$ is the number of 1s in the matrix A. Since by construction, every path has length ℓ, we have $\text{tr}(M) = p\ell$. Furthermore, let m_{ij} be the (i, j)-th element of matrix M, and observe that it is exactly the number of paths that contain vertices i and j. (Note that $m_{ij} = m_{ji}$.) Additionally, $\text{tr}(M^2)$ is the number of length 4 closed walks in the bipartite graph representing the incidence matrix A. This implies that

$$\mathrm{tr}(M^2) = \sum_{j=1}^{p} \sum_{\substack{u,v \in P_j, \\ u \neq v}} m_{u,v} + \sum_{i=1}^{n} m_{ii}^2 = \sum_{j=1}^{p} \sum_{i=1}^{\ell} \sum_{\substack{u,v \in P_j, \\ h(u,v)=i}} m_{u,v} + n \cdot O\left(\frac{p\ell}{n}\right)^2$$

$$\leq \sum_{j=1}^{p} \sum_{i=1}^{\ell} \ell \cdot \frac{\ell}{i} + O\left(\frac{p^2\ell^2}{n}\right) \leq p\ell^2 \log \ell + O\left(\frac{p^2\ell^2}{n}\right) = \frac{np\ell^2 \log(\ell) + O(p^2\ell^2)}{n}. \quad (1)$$

By setting $p = n \log n$, it follows that $\ell = \Theta(\sqrt{n}/\log n)$ and $\mathrm{tr}(M) = p\ell$. Further,

$$np\ell^2 \log \ell = O(n \cdot n \log n \cdot \frac{n}{\log^2 n} \cdot \log n) = O(n^3) = O(p^2\ell^2).$$

By Equation (1), we have $\mathrm{tr}(M^2) = O(p^2\ell^2/n)$. Using this and $\mathrm{tr}(M) = p\ell$ in the trace bound of [31] gives us

$$\mathrm{herdisc}(A) \geq \frac{(\mathrm{tr}(M))^2}{8e \min\{p,n\} \cdot \mathrm{tr}(M^2)} \sqrt{\frac{\mathrm{tr}(M)}{\max\{p,n\}}} = \frac{(\mathrm{tr}(M))^2}{8en \cdot \mathrm{tr}(M^2)} \sqrt{\frac{\mathrm{tr}(M)}{p}}$$

$$\geq \Omega\left(\frac{p^2\ell^2}{p^2\ell^2}\sqrt{\ell}\right) = \Omega(\sqrt{\ell}) = \Omega\left(\frac{n^{1/4}}{\sqrt{\log n}}\right).$$

The trace bound is formally stated in the full version of our paper [9]. ◀

3.2 Vertex Discrepancy Upper Bound – Explicit Coloring

In this subsection, we will upper bound the discrepancy $\chi(\Pi)$ of a consistent path system (V, Π) with $|V| = n$ and $|\Pi| = \mathrm{poly}(n)$. This will immediately imply an upper bound for the hereditary vertex discrepancy of unique shortest paths in undirected graphs.

▶ **Theorem 13.** *For a consistent path system $S = (V, \Pi)$ where $|V| = n$ and $|\Pi| = \mathrm{poly}(n)$, there exists a labeling χ such that $\chi(\Pi) = O(n^{1/4} \log^{1/2} n)$. Consequently, every n-vertex undirected graph has hereditary vertex discrepancy $O(n^{1/4} \log^{1/2} n)$.*

Let $S = (V, \Pi)$ be a consistent path system with $|V| = n$ and $|\Pi| = \mathrm{poly}(n)$. As the first step towards constructing our labeling $\chi : V \mapsto \{-1, 1\}$, we will construct a collection of paths Π' on V that will have a useful covering property over the paths in Π.

Constructing path cover Π'. Initially, we let $\Pi' = \emptyset$. We define V' to be the set of all nodes in V belonging to a path in Π', i.e., $V' := \bigcup_{\pi' \in \Pi'} \pi'$. While $|\pi \setminus V'| \geq n^{1/2}$ for some $\pi \in \Pi$, find a (possibly non-contiguous) subpath of π of length $n^{1/2}$ that is vertex-disjoint from all paths in Π'. Formally, find a subpath $\pi' \subseteq \pi$ such that $|\pi'| = n^{1/2}$ and $\pi' \cap V' = \emptyset$. Add path π' to path cover Π' and update V'. Repeatedly add paths to path cover Π' in this manner until $|\pi \setminus V'| < n^{1/2}$ for all $\pi \in \Pi$.

▶ **Proposition 14.** *Path cover Π' satisfies the following properties:*

1. *for all $\pi \in \Pi'$, $|\pi| = n^{1/2}$,*
2. *the number of paths in Π' is $|\Pi'| \leq n^{1/2}$,*
3. (Disjointness Property). *The paths in Π' are pairwise vertex-disjoint,*
4. (Covering Property). *For all $\pi \in \Pi$, the number of nodes in π that do not lie in any path in path cover Π' is at most $n^{1/2}$. Formally, let $V' = \cup_{\pi' \in \Pi'} \pi'$. Then $\forall \pi \in \Pi, |\pi \setminus V'| \leq n^{1/2}$,*
5. (Consistency Property). *For all $\pi \in \Pi$ and $\pi' \in \Pi'$, the intersection $\pi \cap \pi'$ is a (possibly empty) contiguous subpath of π'.*[4]

[4] Note that it may not be true that $\pi \cap \pi'$ is a contiguous subpath of π.

Proof. Properties 1, 3, and 4 follows from the construction of Π'. Property 2 follows from Properties 1 and 3 and the fact that $|V| = n$. The Consistency Property of Π' is inherited from the consistency of path system S. Specifically, by the construction of Π', path $\pi' \in \Pi'$ is a subpath of a path $\pi'' \in \Pi$. Recall that by the consistency of path system S, the intersection $\pi \cap \pi''$ is a (possibly empty) contiguous subpath of π''. Then $\pi \cap \pi'$ is a contiguous subpath of π' since $\pi' \subseteq \pi''$. This concludes the proof. ◀

Constructing labeling χ. Let $\pi' = (v_1, \ldots, v_k) \in \Pi'$ be a path in our path cover. We will label the nodes of π' using the following random process. With probability $1/2$ we define $\chi : \pi' \mapsto \{-1, 1\}$ to be

$$\chi(v_i) = \begin{cases} 1 & i \equiv 0 \mod 2 \text{ and } i \in [1, k] \\ -1 & i \equiv 1 \mod 2 \text{ and } i \in [1, k] \end{cases},$$

and with probability $1/2$ we define $\chi : \pi' \mapsto \{-1, 1\}$ to be

$$\chi(v_i) = \begin{cases} -1 & i \equiv 0 \mod 2 \text{ and } i \in [1, k] \\ 1 & i \equiv 1 \mod 2 \text{ and } i \in [1, k] \end{cases}.$$

The labels of consecutive nodes in π' alternate between 1 and -1, with vertex v_1 taking labels 1 and -1 with equal probability. Since the paths in path cover Π' are pairwise vertex-disjoint, the labeling χ is well-defined over $V' := \cup_{\pi' \in \Pi'} \pi'$. We choose a random labeling for all nodes in $V \setminus V'$, i.e., we independently label each node $v \in V \setminus V'$ with $\chi(v) = -1$ with probability $1/2$ and $\chi(v) = 1$ with probability $1/2$. An illustration can be found in Figure 2.

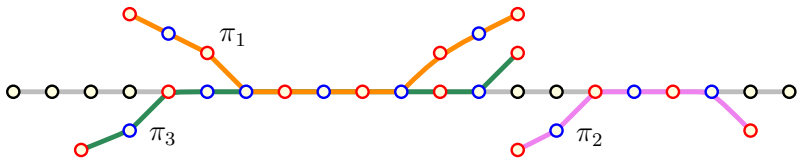


Figure 2 In this figure, paths $\pi_1, \pi_2, \pi_3 \in \Pi'$ from the path cover are intersecting a path $\pi \in \Pi$. Paths in the path cover are pairwise vertex-disjoint, and each path in the cover contributes discrepancy 0, -1, or $+1$ to π.

Bounding the discrepancy $\chi(\Pi)$. Fix a path $\pi \in \Pi$. We will show that $\left|\sum_{v \in \pi} \chi(v)\right| = O(n^{1/4} \log^{1/2} n)$ with high probability. Theorem 13 will follow as $|\Pi| = \text{poly}(n)$.

▶ **Proposition 15.** *For each path π' in path cover Π',*

$$\sum_{v \in \pi \cap \pi'} \chi(v) \in \{-1, 0, 1\}.$$

If $|\pi \cap \pi'| \equiv 0 \mod 2$, then $\sum_{v \in \pi \cap \pi'} \chi(v) = 0$. Moreover,

$$\Pr\left[\sum_{v \in \pi \cap \pi'} \chi(v) = -1\right] = \Pr\left[\sum_{v \in \pi \cap \pi'} \chi(v) = 1\right].$$

Proof. By the Consistency Property of Π' (as proven in Proposition 14), path $\pi \cap \pi'$ is a (possibly empty) contiguous subpath of π'. Then since consecutive nodes in π' alternate between -1 and 1, it follows that $\sum_{v \in \pi \cap \pi'} \chi(v) \in \{-1, 0, 1\}$.

Now note that $\sum_{v \in \pi \cap \pi'} \chi(v) \neq 0$ iff $|\pi \cap \pi'|$ is odd. Moreover, the first vertex of $\pi \cap \pi'$ takes labels 1 and -1 with equal probability. This concludes the proof of Proposition 15. ◀

We are now ready to bound the discrepancy of π.

▶ **Proposition 16.** *With high probability,* $\chi(\pi) = O(n^{1/4}\log^{1/2} n)$.

Proof. We partition the nodes of π into two sources of discrepancy that we will bound separately. Let $V' := \cup_{\pi' \in \Pi'} \pi'$.

Discrepancy of $\pi \cap V'$. For each path $\pi' \in \Pi'$, let $X_{\pi'}$ be the random variable defined as

$$X_{\pi'} := \sum_{v \in \pi \cap \pi'} \chi(v).$$

We can restate the discrepancy of $\pi \cap V'$ as

$$\left| \sum_{v \in \pi \cap V'} \chi(v) \right| = \left| \sum_{\pi' \in \Pi'} X_{\pi'} \right|.$$

By Proposition 15, if $|\pi \cap \pi'| \equiv 0 \mod 2$, then $X_{\pi'} = 0$, so we may assume without any loss of generality that $|\pi \cap \pi'|$ is odd for all $\pi' \in \Pi'$. In this case, $\Pr[X_{\pi'} = -1] = \Pr[X_{\pi'} = 1] = 1/2$, implying that $\mathbb{E}[\sum_{\pi' \in \Pi'} X_{\pi'}] = 0$. Then by Proposition 14 and Chernoff, it follows that for any constant $c \geq 1$,

$$\Pr\left[\left| \sum_{\pi' \in \Pi'} X_{\pi'} \right| \geq c \cdot n^{1/4}\log^{1/2} n\right] \leq e^{-c^2 \frac{n^{1/2}\log n}{2|\Pi'|}} \leq e^{-c^2/(2\cdot\log(n))} = n^{-c^2/2}.$$

Discrepancy of $\pi \setminus V'$. Note that by the Covering Property of the path cover (as proven in Proposition 14), $|\pi \setminus V'| \leq n^{1/2}$. Moreover, the nodes in $V \setminus V'$ are labeled independently at random, implying that $\mathbb{E}[\sum_{v \in \pi \setminus V'} \chi(v)] = 0$. Then we may apply a Chernoff bound to argue that for any constant $c \geq 1$,

$$\Pr\left[\left| \sum_{v \in \pi \setminus V'} \chi(v) \right| \geq c \cdot n^{1/4}\log^{1/2} n\right] \leq \exp\{-c^2 \frac{n^{1/2}\log n}{2|\pi \setminus V'|}\} \leq e^{-c^2/(2\cdot\log(n))} = n^{-c^2/2}.$$

We have shown that with high probability, the discrepancy of our labeling is $O(n^{1/4}\log^{1/2} n)$ for $\pi \cap V'$ and $O(n^{1/4}\log^{1/2} n)$ for $\pi \setminus V'$, so we conclude that the total discrepancy of π is $O(n^{1/4}\log^{1/2} n)$, completing the proof of Proposition 16. ◀

Extending to hereditary discrepancy. Let A be the vertex incidence matrix of a path system $S = (V, \Pi)$ on n nodes, and let A_Y be the submatrix of A obtained by taking all of its rows but only a subset Y of its columns. Then there exists a subset $V_Y \subseteq V$ of the nodes in V such that A_Y is the vertex incidence matrix of the path system $S[V_Y]$ (path system S induced on V_Y). Moreover, if path system S is consistent, then $S[V_Y]$ is also consistent. Then we may apply our explicit vertex discrepancy upper bound to $S[V_Y]$. We conclude that the hereditary vertex discrepancy of S is $O(n^{1/4}\log^{1/2} n)$.

3.3 Edge Discrepancy Upper Bound – Explicit Coloring

By Theorem 5, the edge discrepancy of the unique shortest paths of a (possibly directed) graph on m edges is $O(m^{1/4})$. However, in the case of undirected graphs and DAGs, we can improve the edge discrepancy to $O(n^{1/4}\log^{1/2} n)$, where n is the number of vertices

in the graph, by modifying the explicit construction for vertex discrepancy in Section 3.2. Our proof strategy will follow the same framework as the explicit construction for vertex discrepancy, but with some added complications in the construction and analysis.

We first introduce some new notation that will be useful in this section. Given a path π and nodes $u, v \in \pi$, we say that $u <_\pi v$ if u occurs before v on path π. Additionally, given a path system $S = (V, \Pi)$, we define the edge set $E \subseteq V \times V$ of the path system as the set of all pairs of nodes $u, v \in V$ that appear consecutively in some path in Π. Likewise, for any path π over the vertex set V, we define the edge set of π, $E(\pi) \subseteq \pi \times \pi$, as the set of all pairs of nodes $u, v \in \pi$ such that u, v appear consecutively in π and $(u, v) \in E$. Note that if path system S corresponds to paths in a graph G, then E will be precisely the edge set of G.

Recall that we wish to construct an edge labeling $\chi : E \mapsto \{-1, 1\}$ so that

$$\chi(\Pi) = \max_{\pi \in \Pi} \left| \sum_{e \in E(\pi)} \chi(e) \right|$$

is minimized. We will upper bound the discrepancy $\chi(\Pi)$ of consistent path systems such that $|V| = n$ and $|\Pi| = \text{ poly}(n)$. This will immediately imply an upper bound on the edge discrepancy of unique shortest paths in undirected graphs.

▶ **Theorem 17.** *For all consistent path systems $S = (V, \Pi)$ where $|V| = n$ and $|\Pi| = \text{poly}(n)$ with edge set E, there exists a labeling $\chi : E \to \{-1, 1\}$ such that*

$$\chi(\Pi) = O(n^{1/4} \log^{1/2} n).$$

Consequently, every n-vertex undirected graph has hereditary edge discrepancy $O(n^{1/4} \log^{1/2} n)$.

Let $S = (V, \Pi)$ be a consistent path system with $|V| = n$ and $|\Pi| = \text{poly}(n)$. As the first step towards constructing our labeling $\chi : E \mapsto \{-1, 1\}$, we will construct a collection of paths Π' on V that will have a useful covering property over the paths in Π.

3.3.1 Constructing path cover Π'

Initially, we let $\Pi' = \emptyset$. We define V' to be the set of all nodes in V belonging to a path in Π', i.e.,

$$V' := \bigcup_{\pi' \in \Pi'} \pi'.$$

While there exists a path $\pi \in \Pi$ such that $|\pi \setminus V'| \geq n^{1/2}$, find a (possibly non-contiguous) subpath of π of length $n^{1/2}$ that is vertex-disjoint from all paths in Π'. Specifically, let $\pi' \subseteq \pi$ be a (possibly non-contiguous) subpath of π containing exactly the first $n^{1/2}$ nodes in $\pi \setminus V'$. Add path π' to path cover Π' and update V'. Repeatedly add paths to path cover Π' in this manner until $|\pi \setminus V'| < n^{1/2}$ for all $\pi \in \Pi$.

Note that our path cover Π' is very similar to the path cover used in the explicit vertex discrepancy upper bound. Indeed, path cover Π' inherits all properties of the path cover defined in Subsection 3.2. The key difference here is that we require subpaths $\pi' \subseteq \pi$ in Π' to contain the *first* $n^{1/2}$ nodes in $\pi \setminus V'$. This will imply an additional property of our path cover, which we call the No Repeats Property.

▶ **Proposition 18.** *Path cover Π' satisfies all properties of Proposition 14, as well as the following additional properties:*

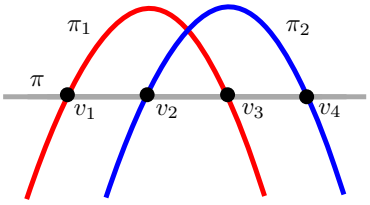


Figure 3 In this figure, paths $\pi_1, \pi_2 \in \Pi'$ are intersecting a path $\pi \in \Pi$. This arrangement of paths is forbidden by the No Repeats Property of Proposition 18.

- (Edge Covering Property). *For all $\pi \in \Pi$, the number of edges in π that are not incident to any node lying in a path in path cover Π' is at most $n^{1/2}$. Formally, let $V' = \cup_{\pi' \in \Pi'} \pi'$. For all $\pi \in \Pi$,*

  $$|\{(u, v) \in E(\pi) \mid u \notin V' \text{ and } v \notin V'\}| \leq n^{1/2},$$

- (No Repeats Property). *For all paths $\pi \in \Pi$, $\pi_1, \pi_2 \in \Pi'$, and nodes $v_1, v_2, v_3, v_4 \in \pi$ such that $v_1, v_3 \in \pi_1$ and $v_2, v_4 \in \pi_2$, the following ordering of the vertices in Π is impossible:*

  $$v_1 <_\pi v_2 <_\pi v_3 <_\pi v_4,$$

  where $x <_\pi y$ indicates that node x occurs in π before node y.

Proof. All properties from Proposition 14 follow from an identical argument as in the original proof. The Edge Covering Property follows immediately from the Covering Property of Proposition 14. What remains is to prove the No Repeats Property.

Suppose for the sake of contradiction that there exist paths $\pi \in \Pi$, $\pi_1, \pi_2 \in \Pi'$, and nodes $v_1, v_2, v_3, v_4 \in \pi$ such that $v_1, v_3 \in \pi_1$ and $v_2, v_4 \in \pi_2$, where $v_1 <_\pi v_2 <_\pi v_3 <_\pi v_4$. We will assume that path π_1 was added to Π' before path π_2 (the case where π_2 was added to Π' first is symmetric). By the construction of Π', path $\pi_1 \in \Pi'$ is a (possibly non-contiguous) subpath of a path $\pi_1'' \in \Pi$ that it was constructed from. Additionally, by the consistency of the path system S, the intersection $\pi \cap \pi_1''$ is a contiguous subpath of π. Then $v_2 \in \pi \cap \pi_1''$, and specifically, $v_2 \in \pi_1''$.

We assumed that $v_2 \in \pi_2$, which implies that $v_2 \notin \pi_1$, since paths in Π' are pairwise vertex-disjoint. Since path π_1 was added to Π' before path π_2, this means that when π_1 was added to Π', node v_2 did not belong to any path in Π' (i.e., v_2 was not in V'). Recall that in our construction of Π', we constructed subpath $\pi_1 \subseteq \pi_1''$ so that it contained exactly the *first* $n^{1/2}$ nodes in $\pi_1'' \setminus V'$. However, $v_2 \notin \pi_1$, but $v_3 \in \pi_1$, and v_2 comes before v_3 in π_1''. This contradicts our construction of path π_1 in path cover Π'. ◀

3.3.2 Constructing labeling χ

Let $\pi' \in \Pi'$ be a path of length k in our path cover. Let $e_1, \ldots, e_k \in E(\pi')$ be the edges in π' listed in the order they appear in π'. Note that since π' is a possibly non-contiguous subpath of a path in Π, pairs of nodes $u, v \in V$ that appear consecutively in π do not necessarily correspond to edges in edge set E.

We will label the edges in $E(\pi')$ using the following random process. With probability $1/2$ we define $\chi : E(\pi') \mapsto \{-1, 1\}$ to be

$$\chi(e_i) = \begin{cases} 1 & i \equiv 0 \mod 2 \text{ and } i \in [1, k] \\ -1 & i \equiv 1 \mod 2 \text{ and } i \in [1, k] \end{cases},$$

and with probability 1/2 we define $\chi : E(\pi') \mapsto \{-1, 1\}$ to be

$$\chi(e_i) = \begin{cases} -1 & i \equiv 0 \mod 2 \text{ and } i \in [1, k] \\ 1 & i \equiv 1 \mod 2 \text{ and } i \in [1, k] \end{cases}.$$

Note that the labels of consecutive edges e_i, e_{i+1} in π' alternate between 1 and -1, with edge e_1 taking labels 1 and -1 with equal probability.

Since the paths in path cover Π' are pairwise vertex-disjoint, the labeling χ is well-defined over $E' := \cup_{\pi' \in \Pi'} E(\pi')$. We take a random labeling for all edges in $E \setminus E'$, i.e., we independently label each edge $e \in E \setminus E'$ with $\chi(e) = -1$ with probability 1/2 and $\chi(e) = 1$ with probability 1/2.

3.3.3 Bounding the discrepancy ϕ

Fix a path $\pi := \pi[s, t] \in \Pi$. We will show that

$$\left| \sum_{e \in E(\pi)} \chi(e) \right| = O(n^{1/4} \log^{1/2} n)$$

with high probability. This will complete the proof of Lemma 17 since $|\Pi| = \text{poly}(n)$. The proof of the following proposition follows from an argument identical to Proposition 15 and hence omitted.

▶ **Proposition 19.** *For each path π' in path cover Π',*

$$\sum_{e \in E(\pi) \cap E(\pi')} \chi(e) \in \{-1, 0, 1\}.$$

If $|E(\pi) \cap E(\pi')| \equiv 0 \mod 2$, then $\sum_{e \in E(\pi) \cap E(\pi')} \chi(e) = 0$. Moreover,

$$\Pr\left[\sum_{e \in E(\pi) \cap E(\pi')} \chi(e) = -1 \right] = \Pr\left[\sum_{e \in E(\pi) \cap E(\pi')} \chi(e) = 1 \right].$$

We are now ready to bound the edge discrepancy of π. Define

$$V' := \bigcup_{\pi' \in \Pi'} \pi' \quad \text{and} \quad E' := \bigcup_{\pi' \in \Pi'} E(\pi').$$

We partition the edges of the path π into three sources of discrepancy that we will bound separately. Specifically, we split $E(\pi) \subseteq \pi \times \pi$ into the following sets E_1, E_2, E_3:

- $E_1 := E(\pi) \cap E'$,
- $E_2 := E(\pi) \cap ((V \setminus V') \times (V \setminus V'))$, and
- $E_3 := E(\pi) \setminus (E_1 \cup E_2)$.

Sets E_1 and E_2 roughly correspond to the two sources of discrepancy considered in the vertex discrepancy upper bound, while set E_3 corresponds to a new source of discrepancy that will require new arguments to bound. We begin with set E_1.

▶ **Proposition 20** (Discrepancy of E_1). *With high probability, $\left|\sum_{e \in E_1} \chi(e)\right| = O(n^{1/4} \log^{1/2} n)$.*

Proof. The proposition follow from an argument similar to Proposition 16. For each path $\pi' \in \Pi'$, let $X_{\pi'}$ be the random variable defined as

$$X_{\pi'} := \sum_{e \in E(\pi) \cap E(\pi')} \chi(e).$$

We can restate the discrepancy of E_1 as

$$\left| \sum_{e \in E_1} \chi(e) \right| = \left| \sum_{\pi' \in \Pi'} X_{\pi'} \right|.$$

By Proposition 19, if $|E(\pi) \cap E(\pi')| \equiv 0 \mod 2$, then $X_{\pi'} = 0$, so without any loss of generality, we may assume that $|E(\pi) \cap E(\pi')|$ is odd for all $\pi' \in \Pi'$. In this case,

$$\Pr[X_{\pi'} = -1] = \Pr[X_{\pi'} = 1] = 1/2,$$

implying that $\mathbb{E}[\sum_{\pi' \in \Pi'} X_{\pi'}] = 0$. Then by Proposition 18 and the Chernoff bound, it follows that for any constant $c \geq 1$,

$$\Pr\left[\left| \sum_{\pi' \in \Pi'} X_{\pi'} \right| \geq c \cdot n^{1/4} \log^{1/2} n \right] \leq e^{-c^2 \frac{n^{1/2} \log n}{2|\Pi'|}} \leq e^{-c^2/2 \cdot \log n} \leq n^{-c^2/2}. \quad \blacktriangleleft$$

We now bound the discrepancy of E_2.

▶ **Proposition 21** (Discrepancy of E_2). *With high probability,* $\left|\sum_{e \in E_2} \chi(e)\right| = O(n^{1/4} \log^{1/2} n)$.

Proof. The proposition follow from an argument similar to Proposition 16. Note that by the Edge Covering Property of the path cover (Proposition 18),

$$|E_2| = |\{(u, v) \in E(\pi) \mid u, v \notin V'\}| \leq n^{1/2}.$$

Moreover, the edges in $E \setminus E'$ are labeled independently at random, so we may apply a Chernoff bound to argue that for any constant $c \geq 1$,

$$\Pr\left[\left| \sum_{e \in E_2} \chi(e) \right| \geq c \cdot n^{1/4} \log^{1/2} n \right] \leq e^{-c^2 \frac{n^{1/2} \log n}{2|E_2|}} \leq e^{-c^2/2 \cdot \log n} \leq n^{-c^2/2}.$$

completing the proof. ◀

Finally, we upper bound the discrepancy of E_3.

▶ **Proposition 22** (Discrepancy of E_3). *With high probability,* $\left|\sum_{e \in E_3} \chi(e)\right| = O(n^{1/4} \log^{1/2} n)$.

Proof. Let

$$k := |\{\pi' \in \Pi' \mid \pi \cap \pi' \neq \emptyset\}|$$

denote the number of paths in our path cover that intersect π. We define a function $f : \mathbb{Z}_{\geq 0} \mapsto \mathbb{Z}_{\geq 0}$ such that $f(\phi)$ equals the largest possible value of $|E_3|$ when $\phi = k$. Note that f is well-defined since $0 \leq |E_3| \leq |E|$. We will prove that $f(\phi) \leq 4\phi$, by recursively decomposing path π.

When $\phi = 1$, there is only one path $\pi' \in \Pi'$ that intersects π. Then the only edges in E_3 are of the form

$$E(\pi) \cap ((V' \times (V \setminus V')) \cup ((V \setminus V') \times V')) = E(\pi) \cap ((\pi' \times (V \setminus \pi')) \cup ((V \setminus \pi') \times \pi')).$$

By the Consistency Property of Proposition 18, path π' can intersect π and then split apart at most once. Then

$$f(1) = |E_3| = |E(\pi) \cap ((\pi' \times (V \setminus \pi')) \cup ((V \setminus \pi') \times \pi'))| \leq 2.$$

When $\phi > 1$, we will split our analysis into the two cases:

- **Case 1.** There exists paths $\pi'_1, \pi'_2 \in \Pi'$ and nodes $v_1, v_2, v_3 \in \pi$ such that $v_1, v_3 \in \pi'_1$ and $v_2 \in \pi'_2$ and $v_1 <_\pi v_2 <_\pi v_3$. In this case, we can assume without any loss of generality that $\pi[v_1, v_3] \cap \pi'_1 = \{v_1, v_3\}$ (e.g., by choosing v_1, v_3 so that this equality holds). Let x be the node immediately following v_1 in π, and let y be the node immediately preceding v_3 in π. Recall that s is the first node of π and t is the last node of π. It will be useful for the analysis to split π into three subpaths:

  $$\pi = \pi[s, v_1] \circ \pi[x, y] \circ \pi[v_3, t],$$

  where $\circ$ denotes the concatenation operation. Let

  $$\phi_1 := |\{\pi' \in \Pi' \mid \pi[x, y] \cap \pi' \neq \emptyset\}| \text{ and } \phi_2 := |\{\pi' \in \Pi' \mid (\pi[s, v_1] \circ \pi[v_3, t]) \cap \pi' \neq \emptyset\}|.$$

  We claim that $\phi_1 < \phi$, $\phi_2 < \phi$, and $\phi_1 + \phi_2 = \phi$. We will use these facts to establish a recurrence relation for f. By our assumption that $\pi[v_1, v_3] \cap \pi'_1 = \{v_1, v_3\}$, it follows that $\pi[x, y] \cap \pi'_1 = \emptyset$, and so $\phi_1 < \phi$. Likewise, by the No Repeats Property of Proposition 18,

  $$(\pi[s, v_1] \circ \pi[v_3, t]) \cap \pi'_2 = \emptyset,$$

  so $\phi_2 < \phi$. Finally, observe that more generally, if there exists a path $\pi' \in \Pi'$ such that $\pi' \cap \pi[x, y] \neq \emptyset$ and $\pi' \cap (\pi[s, v_1] \circ \pi[v_3, t]) \neq \emptyset$, then the No Repeats Property of Proposition 18 is violated. We conclude that $\phi_1 + \phi_2 = \phi$.
  Now $|E_3|$ can be upper bounded by the following inequality:

  $$|E_3| \leq |E_3 \cap E(\pi[x, y])| + |E_3 \cap E(\pi[s, v_1] \circ \pi[v_3, t])| + 2.$$

  Then using the observations about ϕ_1, ϕ_2, and ϕ in the previous paragraph, we obtain the following recurrence for f:

  $$f(\phi) \leq f(\phi_1) + f(\phi_2) + 2 = f(i) + f(\phi - i) + 2,$$

  where $0 < i < \phi$.
- **Case 2.** There exists a path $\pi' \in \Pi'$ and $v_1, v_2 \in \pi$ such that $\pi \cap \pi' = \pi[v_1, v_2] \cap V'$. Let x be the node immediately preceding v_1 in π, and let y be the node immediately following v_2 in π. Again, we split π into three subpaths:

  $$\pi[s, t] = \pi[s, x] \circ \pi[v_1, v_2] \circ \pi[y, t].$$

  Let

  $$\phi_1 := |\{\pi' \in \Pi' \mid \pi[v_1, v_2] \cap \pi' \neq \emptyset\}| \text{ and } \phi_2 := |\{\pi' \in \Pi' \mid (\pi[s, x] \circ \pi[y, t]) \cap \pi' \neq \emptyset\}|.$$

By our assumption in Case 2, it follows that $\phi_1 = 1$ and $\phi_2 = \phi - 1$. Since $|E_3|$ can be upper bounded by the inequality

$$|E_3| \leq |E_3 \cap E(\pi[v_1, v_2])| + |E_3 \cap E(\pi[s, x] \circ \pi[y, t])| + 2,$$

we immediately obtain the recurrence

$$f(\phi) \leq f(\phi_1) + f(\phi_2) + 2 \leq f(1) + f(\phi - 1) + 2.$$

Taking our results from Case 1 and Case 2 together, we obtain the recurrence relation

$$f(\phi) \leq \begin{cases} \max\{f(i) + f(\phi - i) + 2,\ f(1) + f(\phi - 1) + 2\} & \phi > 1 \text{ and } 1 < i < \phi \\ 2 & \phi = 1 \end{cases}$$

Applying this recurrence $\leq \phi$ times, we find that

$$f(\phi) \leq \phi \cdot f(1) + 2\phi \leq 4\phi.$$

Finally, since $k \leq |\Pi'| \leq n^{1/2}$ and we defined f so that $f(k)$ equals the largest possible value of $|E_3|$, we conclude that

$$|E_3| \leq f(k) \leq f(n^{1/2}) = O(n^{1/2}).$$

Since the edges in $E_3 \subseteq E \setminus E'$ are labeled independently at random, we may apply a Chernoff bound as in Proposition 21 to argue that $\chi(E_3) = O(n^{1/4} \log^{1/2} n)$ with high probability. ◀

We have shown that with high probability, the discrepancy of our edge labeling is $O(n^{1/4} \log^{1/2} n)$ for E_1, E_2, and E_3, so we conclude that the total discrepancy of π is $O(n^{1/4} \log^{1/2} n)$. A straightforward extension of this argument implies identical bounds for hereditary discrepancy. We defer this proof to the full version of our paper [9].

4 Conclusion and Open Problems

This paper reported new bounds on the hereditary discrepancy of set systems of unique shortest paths in graphs. An open problem is to improve our edge discrepancy upper bound in directed graphs. Standard techniques in discrepancy theory imply an upper bound of $\min\{O(m^{1/4}), \widetilde{O}(D^{1/2})\}$ for this problem, leaving a gap with our $\Omega(n^{1/4}/\sqrt{\log n})$ lower bound when $m = \omega(n)$. Unfortunately, we were not able to extend our low-discrepancy edge and vertex coloring arguments for undirected graphs to the directed setting, due to the pathological example in Figure 4.

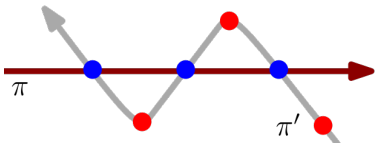


Figure 4 An example in directed graphs that demonstrates how coloring unique shortest paths with alternating colors can fail to imply low discrepancy.

References

1 Ittai Abraham, Amos Fiat, Andrew V Goldberg, and Renato F Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 782–793. SIAM, 2010.

2 Shyan Akmal and Nicole Wein. A local-to-global theorem for congested shortest paths. In *31st Annual European Symposium on Algorithms (ESA 2023)*, pages 8:1–8:17, 2023.

3 J Ralph Alexander, József Beck, and William WL Chen. Geometric discrepancy theory and uniform distribution. *Handbook of discrete and computational geometry*, page 279, 2004.

4 Saeed Akhoondian Amiri and Julian Wargalla. Disjoint shortest paths with congestion on DAGs. *arXiv preprint*, 2020. arXiv:2008.08368.

5 Lorenzo Balzotti. Non-crossing shortest paths are covered with exactly four forests. *arXiv preprint*, 2022. arXiv:2210.13036.

6 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route planning in transportation networks. *Algorithm engineering: Selected results and surveys*, pages 19–80, 2016.

7 Johannes Blum, Yann Disser, Andreas Emil Feldmann, Siddharth Gupta, and Anna Zych-Pawlewicz. On Sparse Hitting Sets: From Fair Vertex Cover to Highway Dimension. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation (IPEC 2022)*, volume 249 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:23, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2022.5.

8 Greg Bodwin. On the structure of unique shortest paths in graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2071–2089. SIAM, 2019.

9 Greg Bodwin, Chengyuan Deng, Jie Gao, Gary Hoppenworth, Jalaj Upadhyay, and Chen Wang. The discrepancy of shortest paths. *arXiv preprint*, 2024. arXiv:2401.15781.

10 Greg Bodwin and Gary Hoppenworth. Folklore sampling is optimal for exact hopsets: Confirming the $\sqrt{n}$ barrier. In *Proceedings of the 64th IEEE Symposium on Foundations of Computer Science (FOCS) 2023*, pages 701–720, 2023.

11 Hsien-Chih Chang, Jonathan Conroy, Hung Le, Lazar Milenkovic, and Shay Solomon. Covering planar metrics (and beyond): O(1) trees suffice. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2231–2261, 2023. doi:10.1109/FOCS57990.2023.00139.

12 Hsien-Chih Chang, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Near-optimal distance emulator for planar graphs. In *26th Annual European Symposium on Algorithms (ESA 2018)*, pages 16:1–16:17, 2018.

13 Hsien-Chih Chang, Robert Krauthgamer, and Zihan Tan. Near-linear ε-emulators for planar graphs. *arXiv preprint*, 2022. arXiv:2206.10681.

14 Bernard Chazelle. *The Discrepancy Method: Randomness and Complexity*. Cambridge University Press, 2000.

15 Bernard Chazelle and Alexey Lvov. A trace bound for the hereditary discrepancy. In *Proceedings of the sixteenth annual symposium on Computational geometry*, SCG '00, pages 64–69, New York, NY, USA, May 2000. Association for Computing Machinery.

16 Justin Y. Chen, Badih Ghazi, Ravi Kumar, Pasin Manurangsi, Shyam Narayanan, Jelani Nelson, and Yinzhan Xu. Differentially private all-pairs shortest path distances: Improved algorithms and lower bounds. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 5040–5067, 2023.

17 Maria Chudnovsky, Daniel Cizma, and Nati Linial. The structure of metrizable graphs. *arXiv preprint*, 2023. arXiv:2311.09364.

18 Fan R. K. Chung, Ronald L. Graham, and Richard M. Wilson. Quasi-random graphs. *Combinatorica*, 9:345–362, 1989.

19 Daniel Cizma and Nati Linial. Geodesic geometry on graphs. *Discrete & Computational Geometry*, 68(1):298–347, 2022.

20 Daniel Cizma and Nati Linial. Irreducible nonmetrizable path systems in graphs. *Journal of Graph Theory*, 102(1):5–14, 2023.

21 Don Coppersmith and Michael Elkin. Sparse sourcewise and pairwise distance preservers. *SIAM Journal on Discrete Mathematics*, 20(2):463–501, 2006.

22 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

23 Chengyuan Deng, Jie Gao, Jalaj Upadhyay, and Chen Wang. Differentially private range query on shortest paths. In *Algorithms and Data Structures Symposium*, pages 340–370. Springer, 2023.

24 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, pages 265–284. Springer, 2006.

25 Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

26 Marek Eliáš, Michael Kapralov, Janardhan Kulkarni, and Yin Tat Lee. Differentially private release of synthetic graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 560–578. SIAM, 2020.

27 Chenglin Fan, Ping Li, and Xiaoyun Li. Private graph all-pairwise-shortest-path distance release with improved error rate. In *Proceedings of Advances in Neural Information Processing Systems*, volume 35, pages 17844–17856, 2022.

28 Sariel Har-Peled. A simple proof of the existence of a planar separator. *arXiv preprint*, 2011. `arXiv:1105.0103`.

29 Monika R Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997.

30 Shimon Kogan and Merav Parter. New diameter-reducing shortcuts and directed hopsets: Breaking the barrier. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1326–1341. SIAM, 2022.

31 Kasper Green Larsen. Constructive discrepancy minimization with hereditary L2 guarantees. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, volume 126 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 48:1–48:13, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

32 Jingcheng Liu, Jalaj Upadhyay, and Zongrui Zou. Optimal bounds on private graph approximation. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1019–1049. SIAM, 2024.

33 Jiři Matoušek. *Geometric Discrepancy: An Illustrated Guide*. Springer Science & Business Media, May 1999.

34 Shanmugavelayutham Muthukrishnan and Aleksandar Nikolov. Optimal private halfspace counting via discrepancy. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1285–1292, 2012.

35 János Pach and Pankaj K Agarwal. *Combinatorial Geometry*. John Wiley & Sons, October 2011.

36 Adam Sealfon. Shortest paths and distances with differential privacy. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 29–41, 2016.

37 Csaba D Toth, Joseph O'Rourke, and Jacob E Goodman. *Handbook of discrete and computational geometry*. CRC Press, 2017.

Additive Spanner Lower Bounds with Optimal Inner Graph Structure

Greg Bodwin
University of Michigan, Ann Arbor, MI, USA

Gary Hoppenworth
University of Michigan, Ann Arbor, Mi, USA

Virginia Vassilevska Williams
Massachusetts Institute of Technology, Cambridge, MA, USA

Nicole Wein
University of Michigan, Ann Arbor, MI, USA

Zixuan Xu
Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

We construct n-node graphs on which any $O(n)$-size spanner has additive error at least $+\Omega(n^{3/17})$, improving on the previous best lower bound of $\Omega(n^{1/7})$ [Bodwin-Hoppenworth FOCS '22]. Our construction completes the first two steps of a particular three-step research program, introduced in prior work and overviewed here, aimed at producing tight bounds for the problem by aligning aspects of the upper and lower bound constructions. More specifically, we develop techniques that enable the use of *inner graphs* in the lower bound framework whose technical properties are provably tight with the corresponding assumptions made in the upper bounds. As an additional application of our techniques, we improve the corresponding lower bound for $O(n)$-size additive emulators to $+\Omega(n^{1/14})$.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

Keywords and phrases Additive Spanners, Graph Theory

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.28

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2404.18337

Funding *Greg Bodwin and Gary Hoppenworth*: Supported by NSF:AF 2153680.
Virginia Vassilevska Williams: Supported by NSF Grant CCF-2330048, BSF Grant 2020356 and a Simons Investigator Award.
Zixuan Xu: Partially supported by NSF Grant CCF-2330048.

1 Introduction

Suppose that we want to compute shortest paths or distances in an enormous graph G. When G is too big to store in memory, a popular strategy is to instead use a *spanner* of G, which is a much sparser subgraph H with approximately the same shortest path metric as G. This can substantially improve storage or runtime costs, in exchange for a small error in the distance information. Perhaps the most well-applied case is when the spanner is asymptotically as sparse as possible; that is, $|E(H)| = O(n)$ for an n-node input graph G (note that $\Omega(n)$ edges are needed just to preserve connectivity).

There are several ways to measure the quality of approximation of a spanner. The two most popular are as follows:

Table 1 The progression of upper and lower bounds on the additive error associated to n-node spanners on $O(n)$ edges; current state of the art bounds are highlighted in red. See also [10, 11, 6, 3, 2] for work on additive spanners of superlinear size.

	Upper Bound		Lower Bound	
$O(n)$-size Spanners			$\Omega(\log n)$	[23]
	$\widetilde{O}(n^{9/16})$	[20]	$\Omega(n^{1/22})$	[1]
	$\widetilde{O}(n^{1/2})$	[9]	$\Omega(n^{1/11})$	[16, 18]
	$O(n^{3/7+\varepsilon})$	[8]	$\Omega(n^{2/21})$	[19]
	$O(n^{\frac{15-\sqrt{54}}{19}<0.403})$	[21]	$\Omega(n^{1/7})$	[7]
			$\Omega(n^{3/17})$	this paper

▶ **Definition 1** (Multiplicative and Additive Spanners). *Given a graph G, a subgraph*[1] *$H \subseteq G$ is a* multiplicative $\cdot k$ spanner *if for all nodes s, t we have $dist_H(s,t) \leq dist_G(s,t) \cdot k$. It is an* additive $+k$ spanner *if we have $dist_H(s,t) \leq dist_G(s,t) + k$.*

The parameter k is called the (additive or multiplicative) *stretch* of the spanner. A famous paper of Althöfer, Das, Dobkin, Joseph, and Soares [4] settled the optimal *multiplicative* stretch for $O(n)$-size spanners:[2]

▶ **Theorem 2** ([4]). *Every n-node graph has a spanner H of size $|E(H)| = O(n)$ and multiplicative stretch $O(\log n)$. This stretch cannot generally be improved to $o(\log n)$.*

The goal of this paper is to make progress on the corresponding question for *additive* error. This question has been intensively studied; see Table 1 for the progression of results. Our contributions are on the lower bounds side:

▶ **Theorem 3** (Main Result). *There exists an infinite family of n-vertex undirected graphs for which any additive spanner on $O(n)$ edges has additive stretch $\Omega(n^{3/17})$.*

Our techniques also lead to progress on related questions for $O(n)$-size emulators, which we discuss further in Section 1.2. Before we explain this, we contextualize Theorem 3 by explaining in more depth the sense in which it moves the upper and lower bounds closer together.

1.1 Our Contribution and Next Steps for the Area

There are well-established frameworks in place for proving upper and lower bounds for $O(n)$-size spanners, and the current sentiment among experts is that these two frameworks *could* eventually produce near-matching (likely within n^{ε} factors) upper and lower bounds. Both frameworks can be broken down into three corresponding steps, and over the last few years, a research program has emerged in which the long-term goal is to find optimal bounds for the problem by making each of these three steps align.[3] That is, we can investigate what "should" happen in each step if a hypothetical optimal version of the upper bound framework were run on the graph from a hypothetical optimal version of the lower bound framework. This thought experiment leads to a list of three concrete features that should be realized in an ideal lower bound, which we overview at a high level in Table 2.

[1] Throughout the paper, for brevity, we write "subgraph" to specifically mean a subgraph over the same vertex set as the original graph.
[2] Although we generally treat input graphs G as undirected and unweighted in this paper, this particular theorem also extends to the setting where G is weighted.
[3] This program was made somewhat explicit in [7] (c.f. Section 2.4), but was implicit in work before that.

However, it is easier to write down this wishlist for the lower bound than it is to actually achieve the listed features in a construction; we discuss the various technical barriers in Section 2. The contribution of the current paper is to achieve the first two steps of alignment (i.e., the first two items in Table 2) simultaneously, which both have to do with optimizing properties of the so-called *inner graph* in the lower bound construction. That said, the ideal structure of an inner graph has been known since [8], and well before that Coppersmith and Elkin [12] found graph constructions achieving this ideal structure ("subset distance preserver lower bound graphs"). Our main technical contributions are not in designing new inner graphs, but rather, in improving the *outer graph* in a way that allows these previously known optimal inner graphs to be used within the framework for the first time.

This paper makes no real progress on the third and final point of alignment, which contends with optimizing certain quantitative properties of the shortest paths in the outer graph. Here there is still significant misalignment between the upper and lower bounds, which is responsible for essentially all of the remaining numeric gap between the current upper and lower bounds for $O(n)$-size spanners. Improving this third point, either on the upper bounds side or the lower bounds side, is the clear next step for the area and it may first require advances in our understanding of distance preservers [12]; see [7] for discussion.

1.2 Additional Results

The technical improvements to the construction that enable our improved spanner lower bounds also imply improvements for two nearby objects, which we overview next. First, an *emulator* is similar to a spanner, but not required to be a subgraph:

▶ **Definition 4** (Additive Emulators). *Given a graph G, a graph H on the same vertex set as G is an* additive $+k$ emulator *if for all nodes s, t we have*

$$dist_G(s,t) \leq dist_H(s,t) \leq dist_G(s,t) + k.$$

An emulator H is allowed to be weighted, even when the input graph G is unweighted. Emulators generalize spanners, and hence the upper and lower bounds known for $O(n)$-size emulators are a bit lower than the corresponding bounds for spanners. See Table 3 for the progression of results on the additive error that can be obtained for $O(n)$-size emulators.

A similar lower bound framework is used to achieve lower bounds for emulators, and hence our new technical machinery improves the current lower bounds for emulators as well:

▶ **Theorem 5.** *There exists an infinite family of n-vertex undirected graphs for which any additive emulator on $O(n)$ edges has additive stretch $\Omega(n^{1/14})$.*

Our numeric improvement in the lower bound for emulators is more modest than our improvement for spanners; at a high level, this is because our main improvement is to enable stronger inner graphs in the lower bound framework, but the role of the inner graph is generally less important in emulator lower bounds.

We next provide a more fine-grained overview of our lower bound framework, and we describe our technical improvements that lead to our new results in more detail.

2 Technical Overview

In this section we will give an overview of the different technical components in our lower bound graph construction. We start by reviewing the obstacle product framework in Section 2.1 and recalling some ideas from prior work in Section 2.2. Finally we will discuss the new components in our construction in Section 2.3.

Table 2 A point-by-point comparison of the frameworks used to prove upper and lower bounds. Our main technical contributions are to satisfy the first point of alignment by enabling the use of inner graphs with $\Theta(r^{4/3})$ nodes (where $+\Omega(r)$ is the desired lower bound on spanner error), and to satisfy the second point of alignment by enabling the use of subset distance preserver lower bounds for our inner graphs. Neither of these properties were fully achieved in prior work.

Step in Upper Bounds	Step in Lower Bounds	What should ideally happen when we run the upper bound framework on a lower bound graph?
Cover the input graph by clusters C of radius r each. These clusters are classified as either *small* or *large*, depending on whether their number of nodes is smaller or larger than $r^{4/3}$.	Start with an *outer graph*, and systematically replace each node with a disjoint copy of an *inner graph*.	The upper bound should select the inner graphs as its clusters. All inner graphs should have $\Theta(r^{4/3})$ nodes, since the worst case for the upper bound is when all clusters are near the large/small threshold.
Small clusters C have a node separator of size $\leq |C|^{1/4}$. Construct a *subset distance preserver* on each small cluster, preserving all shortest paths between separator nodes, at cost $O(\|C\|)$ [12].	The inner graphs should be selected as the union of many long unique shortest paths among nodes that form a separator for the graph, and also any two of these shortest paths may intersect on at most one node.	The inner graph should be a lower bound graph against subset distance preservers with $\Theta(\|C\|^{1/4})$ source nodes (with a large implicit constant), so that the approach of constructing a subset distance preserver is too expensive to be used in an attack against the lower bound.
Large clusters C are handled by adding some additional shortest paths in the spanner to connect far-away clusters to each other. Using the *path-buying framework* [6], we can limit the total number and length of the shortest paths we need to add.	The outer graph is selected to be the union of as many long unique shortest paths as possible, and any two of these shortest paths may intersect on at most one edge. That is, the outer graph is a slightly modified distance preserver lower bound graph.	The shortest paths added for large clusters should coincide with the shortest paths in the original outer graph (before inner graph replacement). The path-buying bounds on the number and length of these shortest paths should coincide with the number and length of these shortest paths in the outer graph.

Table 3 The progression of upper and lower bounds on the additive error associated to n-node emulators on $O(n)$ edges; current state of the art bounds are highlighted in red. See also [13].

	Upper Bound		Lower Bound	
$O(n)$-size Emulators	$O(n^{1/3+\varepsilon})$	[9]	$\Omega(\log n)$	[23]
	$O(n^{3/11+\varepsilon})$	[8]	$\Omega(n^{1/22})$	[1]
	$\widetilde{O}(n^{1/4})$	[20]	$\Omega(n^{1/18})$	[16]
	$\widetilde{O}(n^{2/9-1/1600<0.222})$	[17]	$\Omega(n^{2/29})$	[19]
	$O(n^{\frac{1}{3+\sqrt{5}}+\varepsilon<0.191})$	[15]	$\Omega(n^{1/14})$	this paper

2.1 The obstacle product framework

Similar to all previous works including [1, 16, 19, 7] on proving stretch lower bounds for linear-sized additive spanners, our construction falls under the *obstacle product* framework introduced in [1]. Any construction under this framework consists of an *outer graph* $G_O = (V_O, E_O)$ and an *inner graph* $G_I = (V_I, E_I)$ where every vertex in the outer graph is replaced by a copy of the inner graph. The desired outer graph should contain a set pairs $P_O \subseteq V_O \times V_O$ often called the *critical pairs* such that the following holds:

1. For each pair $(s, t) \in P_O$, the shortest path from s to t is *unique*. These unique shortest paths connecting between pairs in P_O are often called the *critical paths*.
2. The critical paths have roughly the same length $\Theta(k)$.
3. The critical paths are pairwise edge disjoint.

When we replace each vertex in the outer graph with a copy of the inner graph G_I, we make sure that the critical paths remain the unique shortest paths between their endpoints and pairwise edge-disjoint by attaching each incoming edge and outgoing edge to distinct vertices of the inner graph. Finally, we subdivide the edges originally in G_O into paths of length $\Theta(k)$. Now in the resulting graph denoted as $G_{obs} = (V_{obs}, E_{obs})$ with critical pairs P_{obs}, each critical path between the endpoints in P_{obs} uniquely corresponds to a critical path in G_O and it takes the form of traveling alternatingly between subdivided edges in G_O and paths in G_I. In particular, each critical path travels through $\Theta(k)$ subdivided paths of length $\Theta(k)$, and $\Theta(k)$ inner graph copies.

Now let us see how to show that any sparse spanner on G_{obs} must suffer additive distortion $+\Omega(k)$. The goal is to argue that if lots of edges are missing in the spanner $H \subseteq G_{obs}$ compared to G_{obs}, then there exists some pair $(s, t) \in P_{obs}$ whose shortest path π in H falls into one of the following two cases:

1. If π traverses the same sequence of inner graph copies as the critical path in G_{obs}, then it must use at least one *extra* edge in each inner graph copy compared to the critical path in the original graph due to missing edges. Since the critical path passes through $\Theta(k)$ inner graph copies, the path π must suffer a $+\Omega(k)$ distortion in total.
2. If π traverses a different sequence of inner graph copies, then it must traverse a different set of subdivided paths corresponding to the edges in G_O. Since the critical paths in G_O are the unique shortest paths between its endpoints, π must traverse at least one more subdivided path of length $\Theta(k)$ and thus suffer a $+\Omega(k)$ distortion.

Furthermore, note that the reason behind doing inner graph replacement is that without the inner graphs, subdividing each edge of the outer graph would significantly sparsify the graph so that even a trivial spanner including all the edges would have linear size. Adding the inner graphs helps balance the overall density of the graph so that any linear-sized spanner needs to be nontrivial. Thus, ideally we would want the inner graphs to be dense.

2.2 The outer graph: distance preservers and the alternation product

In this subsection, we review the two key components for the outer graph construction: the distance preserver lower bound graph given in [12] and the alternation product first used in [14].

2.2.1 Distance preserver lower bound graph

Given a graph $G = (V, E)$ and a set of pairs $P \subseteq V \times V$, a distance preserver H is a sparse subgraph of G that preserves the distances for every pair in P *exactly*. Previously, Coppersmith and Elkin [12] obtained a lower bound instance for distance preservers by

constructing a large set of vertex pairs with pairwise *edge-disjoint* unique shortest paths that are as long as possible; the union of the edges of these paths is the lower bound instance. Following the intuition outlined in Section 2.1, it is natural to consider using the distance preserver lower bound construction of [12] as the outer graph. From now on, we will abbreviate the term "distance preserver lower bound graph" to "DP LB graph" and the term "Coppersmith-Elkin construction" to "CE construction" for convenience.

Indeed, all prior work uses some version of the CE construction of DP LB graph as the outer graph, and so do we. The CE construction is a geometric construction where the vertex set corresponds to a d-dimensional integer grid $[n]^d$ and edges are added corresponding to a d-dimensional convex set $B_d(r)$ defined to be the vertices of the convex hull of integer points contained in a ball of radius $r > 0$. More specifically, the vertices corresponding to the points $\vec{x}, \vec{y}$ are connected by an edge if $\vec{y} - \vec{x} \in B_d(r)$. Then the critical paths are defined to be the paths corresponding to the straight lines starting from a "start zone" passing through the grid, i.e. the paths that repeatedly take the edge corresponding to the same vector in $B_d(r)$. By convexity of the set $B_d(r)$, one can show that these critical paths are edge-disjoint and they are the unique shortest paths between their endpoints.

Prior to the work of [7], works including [16, 19] all considered a *layered* version of the DP LB graph as the outer graph. Namely the graph contains $\ell + 1$ layers where each layer corresponds to a d-dimensional integer grid $[n]^d$ and edges are added between adjacent layers corresponding to the convex set $B_d(r)$ similarly as defined in [12]. Then the critical paths are defined to be the paths that start in the first layer and end in the last layer that repeatedly take the edge corresponding to the same vector in $B_d(r)$. This layering simplifies the stretch analysis for additive spanner lower bounds because it is easy to argue that all the critical paths have length exactly ℓ and the shortest path should not take any backward edges as it will then need to traverse more layers. However, the layered version resulted in worse bounds compared to the original unlayered version but it was unclear at the time how to analyze an unlayered outer graph. Most recently, Bodwin and Hoppenworth [7] developed a new analysis framework and successfully analyzed an obstacle product graph with a modified version of the unlayered DP LB graph as the outer graph. As a result, they improved the lower bound to $\Omega(n^{1/7})$ from $\Omega(n^{1/10.5})$ where the former remains the current best known lower bound before this work. We use the unlayered outer graph construction in [7] as an ingredient in our construction.

2.2.2 The alternation product

Another important idea that goes in to the outer graph construction is the alternation product first used in [14]. Subsequent works including [1, 16, 19] all use the alternation product in the outer graph construction. Consider two copies G_1, G_2 of the same 2-dimensional layered DP LB graph with $\ell + 1$ layers and convex set $B_2(r)$. Namely, each layer corresponds to the $[n]^2$ grid and the edges correspond to the 2-dimensional convex set $B_2(r)$ of radius r. The original implementation of the alternation product graph G_{alt} used in [14, 1, 16] of G_1 and G_2 is a graph on $2\ell + 1$ layers with each layer corresponding to the 4-dimensional grid $[n]^4$. Each vertex in G_{alt} corresponds the pair (v_1, v_2) where $v_1 \in G_1, v_2 \in G_2$ and the edges are added alternatingly between adjacent layers according to G_1 and G_2, respectively. Specifically, between layer i and $i+1$ for i odd, we connect the vertex $(\vec{x}, \vec{y})$ for $\vec{x}, \vec{y} \in [n]^2$ to $(\vec{x} + \vec{w}, \vec{y})$ for $\vec{w} \in B_2(r)$; for i even, we connect the vertex $(\vec{x}, \vec{y})$ to $(\vec{x}, \vec{y} + \vec{w})$ for $\vec{w} \in B_2(r)$. In other words, G_{alt} keeps track of G_1 using the first two coordinates and G_2 using the last two coordinates. Then a critical path π in G_{alt} corresponds to a pair of critical paths π_1 in G_1 and π_2 in G_2 by taking alternating steps from π_1 and π_2. So the main advantage of the alternation product for us is that it gives an extra product structure over the set of critical paths that we want in our construction.

Unlike in our construction, prior works including [14, 1, 16, 19] apply the alternation product in order to obtain a different relative count between the number of vertices and the number of critical pairs rather than to obtain the extra product structure. However, these changes in parameters are in fact unfavorable to the construction for linear-sized spanner lower bounds. To see this, notice that one can equivalently think of G_{alt} as 4-dimensional CE construction graph using the smaller convex set $\{(\vec{w}_1, \vec{w}_2) \mid \vec{w}_1, \vec{w}_2 \in B_2(r)\}$ instead of $B_4(r)$, which means that G_{alt} has fewer critical pairs (see Section 2.3 for a more detailed discussion). In fact, in [16], Huang and Pettie gave an $\Omega(n^{1/11})$ lower bound construction without the alternation product that improved on their own construction that uses the alternation product which gave a bound of $\Omega(n^{1/13})$ in the same paper. Later in [19], Lu, Vassilevska Wiliams, Wein and Xu improved on the alternation product that reduces the loss in the number of critical pairs compared to the CE construction, thereby obtaining an $\Omega(n^{1/10.5})$ lower bound that improved on the previous best bound of $\Omega(n^{1/11})$. Most recently, Vassilevska Wiliams, Xu and Xu implicitly constructed an alternation product graph in their $O(m)$-shortcut lower bound construction in [22] that asymptotically matches the number critical pairs in the CE construction. Unfortunately, their construction is under a different setting so we cannot directly apply their technique to our construction as a blackbox. However, by isolating a main observation implied in their work, we were able to integrate such an alternation product into our construction (see Section 2.3).

2.3 Our construction: optimal unlayered alternation product and optimal inner graph structure

Our main technical contribution is a linear-sized additive spanner lower bound construction that carefully combines the following ideas:

1. An unlayered DP LB graph as the outer graph, as in [7].
2. An optimal alternation product implicit in [22].
3. An optimal subset DP LB graph as the inner graphs, as motivated in Table 2.

We start with comparing our construction with the previously known lower bound constructions in Table 4.

Table 4 All known lower bound constructions.

Citation	Lower bound	Outer graph	Inner graph
Woodruff [23]	$\Omega(\log n)$	Butterfly	Biclique
Abboud, Bodwin [1]	$\Omega(n^{1/22})$	Layered DP LB + Alt Product	Biclique
Huang, Pettie [16]	$\Omega(n^{1/13})$	Layered DP LB + Alt Product	Biclique
Huang, Pettie [16]	$\Omega(n^{1/11})$	Layered DP LB	Layered DP LB
Lu, Vassilevska W., Wein, Xu [19]	$\Omega(n^{1/10.5})$	Layered DP LB + Improved Alt Product	Biclique
Bodwin, Hoppenworth [7]	$\Omega(n^{1/7})$	Unlayered DP LB	DP LB
This work	$\Omega(n^{3/17})$	Unlayered DP LB + Optimal Alt Product	Subset DP LB

In the following, we will discuss the main components of our construction.

2.3.1 Outer Graph: Unlayered DP LB graph with optimal alternation product

As mentioned in Section 2.2, we would like to be able to apply the implicit alternation product in [22] to unlayered DP LB graphs. By isolating the main idea that one can use the set $\{(x, y, x^2 + y^2) \mid x, y \in [r]\}$ as the convex set in the alternation product graph, we are able to apply the implicit alternation product in [22] on unlayered DP LB graphs successfully after certain modifications (See Section 4.1 for more details). In the following, we give a more detailed discussion of the informal intuition behind why the alternation product we use is more desirable than the alternation product used in prior works including [1, 16, 19].

Recall that in Section 2.2, one may view an alternation product graph as a CE construction with a different convex set that determines the set of edges of the graph. In addition, a vector from the convex set and a vertex in the "start zone" determines a critical path, so we get more critical pairs if we use a larger convex set. More precisely, we want to use a convex set that is "large" with respect to the total number of integer points contained in the convex hull of the set. We recall from [5] that $|B_d(r)| = \Theta(r^{d \cdot \frac{d-1}{d+1}})$. Let us compare the convex sets used in the various alternation product graphs against $B_d(r)$ in the same number of dimension that is scaled to contain the same number of points in its convex hull asymptotically in Table 5. Then we can see from Table 5 that all prior constructions use a convex set that contains less points than the respective $B_d(r)$ while the convex set we use in this work matches the quality of $B_3(r)$, which is optimal in 3-dimensions (see [5] for more details). That is, the construction that we use is as good as the CE construction in 3-dimensions.

Table 5 Comparison between known constructions of the alternation product and the CE construction. The top row in each pair is the corresponding CE construction in the same number of dimensions and scaled to contain the same number of points in its convex hull. The bottom row in each pair indicates the alternation product construction used in the work cited.

Citation	Convex Set Used	Convex Set Size	Convex Hull Size
4-dim [12]	$B_4(r)$	$\Theta(r^{12/5})$	$\Theta(r^4)$
[1, 16]	$\{(\vec{x}, \vec{y}) \mid \vec{x}, \vec{y} \in B_2(r)\}$	$\Theta(r^{4/3})$	$\Theta(r^4)$
3-dim [12]	$B_3(r)$	$\Theta(r^{3/2})$	$\Theta(r^3)$
[19]	$\{(x_1, x_2 + y_1, y_2) \mid \vec{x}, \vec{y} \in B_2(r)\}$	$\Theta(r^{4/3})$	$\Theta(r^3)$
3-dim [12]	$B_3(r^{4/3})$	$\Theta(r^2)$	$\Theta(r^4)$
This work (based on [22])	$\{(x, y, x^2 + y^2) \mid x, y \in [r]\}$	$\Theta(r^2)$	$\Theta(r^4)$

One may wonder why we do not simply use the CE construction as our outer graph. The reason is that the alternation product has extra structure that is crucial for allowing us to use our desired inner graph. The CE construction lacks these properties. We elaborate on this below.

2.3.2 Inner graph: Optimal subset DP LB graph

For our inner graphs, we use the CE construction of subset DP LB graphs in the regime where the pairs $S \times S$ has size $|S| = \Theta(n^{1/4})$ where n denotes the number of vertices in the graph. In fact, this construction is tight in the sense that it has $\Omega(n)$ edges while on the other hand it is known that there exists subset distance preservers of size $O(n)$ for every set of sources S of size $O(n^{1/4})$. So not only are we using an inner graph structure that aligns with the upper bound algorithm as illustrated in Table 2, we are in fact using a tight construction of the desired structure.

The main reason why we are able to use subset DP LB graphs as inner graphs in our construction is that we have an alternation product graph as our outer graph. We discuss below why an alternation product is necessary for using subset DP LB graphs as inner graphs. In the inner graph replacement step under the obstacle product framework, we need to attach the incoming edges and outgoing edges adjacent to a vertex v to vertices in the corresponding inner graph copy so that each critical path passing through v in the outer graph will pass through a unique critical path in the inner graph copy as well. Since the subset DP LB graph has critical pairs of the form $S \times S$ for some subset S of the vertex set, it is required that the critical paths passing through v in the outer graph also be equipped with a product structure. In DP LB graphs, we have no such product structure over the critical paths. However, notice that applying an alternation product would exactly give us a product structure over the critical paths as desired.

3 Preliminaries

We use the following notations:

- We use $\texttt{Conv}(\cdot)$ to denote the convex hull of a set.
- We use $\langle \cdot, \cdot \rangle$ to denote the standard Euclidean inner product, $\| \cdot \|$ the Euclidean norm, and $\texttt{proj}_{\vec{w}}(\cdot)$ the Euclidean scalar projection onto $\vec{w}$.
- We use $[x]$, where x is a positive integer, to denote the set $\{1, \ldots, x\}$.

4 Outer Graph G_O

The goal of this section will be to construct the outer graph G_O of our additive spanner and emulator lower bound constructions. The key properties of G_O are summarized in Theorem 6.

▶ **Theorem 6** (Properties of Outer Graph). *For any $a, r > 0 \in \mathbb{Z}$, there exists a graph $G_O(a, r) = (V_O, E_O)$ with a set Π_O of critical paths in G_O that has the following properties:*

1. *The number of nodes, edges, and critical paths in G_O is:*

$$|V_O| = \Theta(a^3 r),$$
$$|E_O| = \Theta(a^3 r^2),$$
$$|\Pi_O| = \Theta(a^2 r^4).$$

2. *Every critical path $\pi \in \Pi_O$ is a unique shortest path in G_O of length at least $|\pi| \geq \frac{a}{4r}$.*
3. *Every pair of distinct critical paths $\pi_1, \pi_2 \in \Pi_O$ intersect on at most two nodes.*
4. *Every edge $e \in E_O$ lies on some critical path in Π_O.*

The rest of the section is devoted to constructing the graph $G_O(a, r)$ and paths Π_O that satisfy Theorem 6.

4.1 Convex Set of Vectors

Before specifying the construction of the graph G_O, we begin by specifying our construction of a set of vectors $W \subseteq \mathbb{R}^3$ that is crucial to the construction of G_O. Set W will be parameterized by a positive integer r, i.e., $W = W(r)$. The vectors in W will satisfy a certain strict convexity property that we will use to ensure the unique shortest paths property of paths Π_O in G_O.

▶ **Definition 7** ($W(r)$). *Given a positive integer r, let*

$$W_1(r) := \{(x, 0, x^2) \mid x \in \{r/2, \ldots, r\}\} \quad \text{and} \quad W_2(r) := \{(0, y, y^2) \mid y \in \{r/2, \ldots, r\}\}.$$

We define $W(r)$ to be the sumset

$$W(r) := W_1(r) + W_2(r) = \{(x, y, x^2 + y^2) \mid x, y \in \{r/2, \ldots, r\}\}.$$

We now verify that sets of vectors $W_1(r), W_2(r), W(r)$ have the necessary convexity property to ensure that graph G_O has unique shortest paths (Property 2 of Theorem 6). The convexity property of W stated in Lemma 8 is roughly similar to the notion of "strong convexity" in [7], but is in fact stronger.

▶ **Lemma 8** (Convexity property). *Let W_1, W_2, W be the sets defined in Definition 7 for some positive integer r. Let W' be the set*

$$W' = W \cup (-W) \cup (W_1 - W_2) \cup (W_2 - W_1).$$

Then each vector $\vec{w} \in W$ is an extreme point of the convex hull $\texttt{Conv}(W')$ of W'.

We omit the proof as it simply follows from checking that every vector in W' is an extreme point of $\texttt{Conv}(W')$.

4.2 Construction of G_O

Let $a, r > 0 \in \mathbb{Z}$ be the input parameters for our construction of outer graph $G_O = (V_O, E_O)$. Let $W_1 = W_1(r)$, $W_2 = W_2(r)$, and $W = W(r)$ be the sets of vectors constructed in Definition 7 and parameterized by our choice of r.

Vertex Set V_O

- Our vertex set V_O will correspond to two copies of integer points arranged in a grid in $\mathbb{R}^3$. These two copies will be denoted as V_O^L and V_O^R. For a point $p \in \mathbb{R}^3$, we will use p_L to denote the copy of point p in V_O^L, and p_R to denote the copy of point p in V_O^R. Likewise, for a set of points $P \subseteq \mathbb{R}^3$, we will use P_L to denote the copy of set P in V_O^L, and P_R to denote the copy of set P in V_O^R. Then we define V_O^L and V_O^R as:

  $$V_O^L = ([a] \times [a] \times [ar])_L, \qquad \text{and} \qquad V_O^R = ([a] \times [a] \times [ar])_R.$$

  When denoting a node v_L or v_R in V_O, we will drop the subscript and simply denote this node as v when its membership in V_O^L and V_O^R is clear from the context or otherwise irrelevant.

Edge Set E_O

- The edges E_O in G_O will pass between V_O^L and V_O^R, so that $E_O \subseteq V_O^L \times V_O^R$.
- Just as the nodes in V_O are integer points in $\mathbb{R}^3$, we will identify the edges in E_O with integer vectors in $\mathbb{R}^3$. Specifically, for each edge (x_L, y_R) in E_O, we identify $x_L \to y_R$ with the vector $y - x \in \mathbb{R}^3$. Note that $y - x$ corresponds to the orientation $x_L \to y_R$ of edge (x_L, y_R); we would use vector $x - y \in \mathbb{R}^3$ to denote $y_R \to x_L$.
- For each node $v_L \in V_O^L$ and each vector $\vec{w} \in W_1$, if $(v + \vec{w})_R \in V_O^R$, then add edge $(v_L, (v + \vec{w})_R)$ to E_O. Likewise, for each node $v_R \in V_O^R$ and each vector $\vec{w} \in W_2$, if $(v + \vec{w})_L \in V_O^L$, then add edge $(v_R, (v + \vec{w})_L)$ to E_O.

Critical Paths Π_O

- Let $S \subseteq \mathbb{R}^3$ denote the set of points $S = [a] \times [a] \times [r^2/8]$.
- Let s be a point in S. Additionally, let $\vec{w}_1$ be a vector in W_1, and let $\vec{w}_2$ be a vector in W_2, where $\vec{w}_1 + \vec{w}_2 \in W$.[4] If $s + \vec{w}_1 \notin S$, then we define a corresponding path in G_O starting from $s_L \in S_L$ as

$$s_L \to (s + \vec{w}_1)_R \to (s + \vec{w}_1 + \vec{w}_2)_L \to (s + 2\vec{w}_1 + \vec{w}_2)_R \to (s + 2\vec{w}_1 + 2\vec{w}_2)_L \to \cdots \to (s + i \cdot \vec{w}_1 + i \cdot \vec{w}_2)_L,$$

  where i is the largest integer i such that node $(s + i \cdot \vec{w}_1 + i \cdot \vec{w}_2)_L \in V_O^L$. Let $t = (s + i \cdot \vec{w}_1 + i \cdot \vec{w}_2)_L$ be the endpoint of this path, and add this $s \rightsquigarrow t$ path to our set of critical paths Π_O.
- Note that every critical path $\pi \in \Pi_O$ constructed this way is uniquely specified by a start node $s_L \in S_L$ and vectors $\vec{w}_1 \in W_1$ and $\vec{w}_2 \in W_2$.
- For every critical path $\pi \in \Pi_O$ where $|\pi| < \frac{a}{4r}$, remove π from Π_O.

As a final step in our construction of G_O, we remove all edges in G_O that do not lie on some critical path $\pi \in \Pi_O$.

4.3 Properties of G_O

We will now verify that G_O and Π_O satisfy the properties specified in Theorem 6 via the following claims. We omit the proofs as most of them follow from definition and Lemma 8.

▷ Claim 9. Every critical path $\pi \in \Pi_O$ is the unique shortest path between its endpoints in G_O. Moreover, $|\pi| \geq \frac{a}{4r}$.

▷ Claim 10. Every pair of distinct critical paths $\pi_1, \pi_2 \in \Pi_O$ can intersect on either a single vertex or a single edge.

▷ Claim 11. Every edge in G_O is used by at most $r/2$ critical paths $\pi \in \Pi_O$.

▷ Claim 12. The number of nodes, edges, and critical paths in $G_O(a, r)$ is:

$$|V_O| = \Theta(a^3 r),$$
$$|E_O| = \Theta(a^3 r^2),$$
$$|\Pi_O| = \Theta(a^2 r^4).$$

Proof of Theorem 6. Note that graph G_O and critical paths Π_O satisfy Properties 1, 2, and 3 of Theorem 6 by Claim 9, Claim 10, and Claim 12. Moreover, Property 4 of Theorem 6 follows immediately from the final step in our construction of G_O. This completes the proof of Theorem 6. ◀

5 Spanner Lower Bound Construction

In this section we present our lower bound construction for additive spanners. This construction will have a similar structure to the obstacle product graph G constructed for our emulator lower bound, but with several complications. We now describe these modifications to the obstacle product argument:

[4] Note that $\vec{w}_1 + \vec{w}_2 \in W$ for all $\vec{w}_1 \in W_1$ and $\vec{w}_2 \in W_2$, since $W = W_1 + W_2$. However, in Section 5.1, we will modify W so that W is a strict subset of $W_1 + W_2$, which will make this requirement relevant.

- **Convex Sets W_1, W_2, and W.** In Section 5.1, we modify the convex sets of vectors W_1, W_2, and W_3 that we defined in Section 4. The purpose of this modification is technical, but it has to do with the projection argument we employ in our analysis. Our new convex sets of vectors $W_1(r,c), W_2(r,c), W(r,c)$ will now be parameterized by an additional integer $c > 0$. These new sets of vectors will roughly resemble the outer graph vectors in Lemma 8 of [7] and will play a similar role in our analysis of G.
- **Inner Graph G_I.** We choose our inner graphs to be the sourcewise distance preserver lower bound graphs constructed in [12]. Lemma 19 specifies the exact properties of these new inner graphs G_I that we require in our analysis. See Subsections 1.1 and 2.2 for an overview of why we make this design choice.

5.1 Modifying Convex Sets W_1, W_2, and W in G_O

In this section, we modify the definitions of the convex sets of vectors W_1, W_2, and W used to construct outer graph G_O. Let $r, c > 0 \in \mathbb{Z}$ be the input parameters to our convex sets of vectors W_1, W_2, and W.

We define I_i be the interval

$$I_i := \left[\frac{r}{2} + \frac{(2i-2)\cdot r}{4c}, \quad \frac{r}{2} + \frac{(2i-2)\cdot r}{4c} + \frac{r}{16c^3}\right],$$

for $i \in [1, c]$. The following claim is immediate from the definition of intervals I_i.

▷ Claim 13. Intervals $\{I_i\}_{i\in[1,c]}$ satisfy the following properties:
- $I_i \subseteq [r/2, r]$,
- $|I_i| = \frac{r}{16c^3}$
- if $x, y \in I_i$, then $|x - y| \leq \frac{r}{16c^3}$, and
- if $x \in I_i$ and $y \in I_j$, where $i \neq j$, then $|x - y| \geq r/(2c)$.

We will use intervals $\{I_i\}_{i\in[1,c]}$ to construct our sets of vectors $W_1(r,c)$ and $W_2(r,c)$.

▶ **Definition 14** ($W_1(r,c)$ and $W_2(r,c)$). *Let r, c be positive integers. We define $W_1(r,c)$ and $W_2(r,c)$ as*

$$W_1(r,c) := \left\{(x, 0, x^2) \mid x \in I_i, i \in [1,c]\right\} \quad \text{and} \quad W_2(r,c) := \left\{(0, y, y^2) \mid y \in I_i, i \in [1,c]\right\}.$$

Now we partition the vectors in $W_1(r,c)$ into c sets $\mathcal{S}_1^1, \ldots, \mathcal{S}_c^1$ we call *stripes*. We define the ith stripe $\mathcal{S}_i^1$ of $W_1(r,c)$ as $\{(x, 0, x^2) \mid x \in I_i\}$. Likewise, we define the ith stripe $\mathcal{S}_i^2$ of $W_2(r,c)$ as $\{(0, y, y^2) \mid y \in I_i\}$. The key properties of our stripes are summarized in Claim 15, which follows immediately from Claim 13.

▷ Claim 15. Stripes $\{\mathcal{S}_i^1\}_{i\in[1,c]}$ satisfy the following properties:
- $\mathcal{S}_i^1 \subseteq [r/2, r]$,
- $|\mathcal{S}_i^1| = \frac{r}{16c^3}$,
- if $(x, 0, x^2), (y, 0, y^2) \in \mathcal{S}_i^1$, then $|x - y| \leq \frac{r}{16c^3}$, and
- if $(x, 0, x^2) \in \mathcal{S}_i^1$ and $(y, 0, y^2) \in \mathcal{S}_j^1$, where $i \neq j$, then $|x - y| \geq \frac{r}{2c}$.

Moreover, stripes $\{\mathcal{S}_i^2\}_{i\in[1,c]}$ satisfy analogous properties.

Roughly, Claim 15 states that vectors in the same stripe in $W_1(r,c)$ or $W_2(r,c)$ are "close" to each other in some sense, and vectors in different stripes in $W_1(r,c)$ and $W_2(r,c)$ are "far" from each other in some sense. This notion of partitioning a set of vectors into stripes satisfying these properties was introduced in the spanner lower bound construction of [7]. We are now ready to define our set of vectors $W(r,c)$.

▶ **Definition 16** ($W(r,c)$). *Let r, c be positive integers. Unlike in Definition 7, we will define $W(r,c)$ to be a subset of the sumset $W_1(r,c) + W_2(r,c)$. In particular, for $\vec{w}_1 \in W_1(r,c)$ and $\vec{w}_2 \in W_2(r,c)$, we only add $\vec{w}_1 + \vec{w}_2$ to $W(r,c)$ if $\vec{w}_1$ and $\vec{w}_2$ share the same stripe index $i \in [1,c]$. Formally,*

$$W(r,c) := \left\{(x, y, x^2 + y^2) \mid x, y \in I_i, i \in [1,c]\right\} \subset W_1(r,c) + W_2(r,c).$$

The following claim is immediate from the definitions of $W_1(r,c)$, $W_2(r,c)$ and $W(r,c)$.

▷ Claim 17. Sets $W_1(r,c)$, $W_2(r,c)$ and $W(r,c)$ satisfy the following properties:
- $|W_1(r,c)| = |W_2(r,c)| = \Theta\left(\frac{r}{c^2}\right)$,
- $|W(r,c)| = \Theta\left(\frac{r^2}{c^5}\right)$,
- $W_1(r,c) \subset W_1(r)$, $W_2(r,c) \subset W_2(r)$, and $W(r,c) \subset W(r)$, so $W(r,c)$ satisfies the convexity property stated in Lemma 8.

We modify the construction of outer graph G_O in Section 4 by replacing sets $W_1(r), W_2(r)$, and $W(r)$ defined in Section 4.1 with the new sets $W_1(r,c), W_2(r,c)$, and $W(r,c)$. Note that our new choice of sets $W_1(r,c)$ and $W_2(r,c)$ changes the the set of vectors E_O in G_O, while our new choice of set $W(r,c)$ changes the set of critical paths Π_O (see Footnote 4).

By inserting convex sets $W_1(r,c), W_2(r,c)$, and $W(r,c)$ into G_O in place of the sets $W_1(r), W_2(r)$, and $W(r)$, we obtain the following theorem about our modified outer graph $G_O = G_O(a,r,c)$.

▶ **Theorem 18** (Properties of Modified Outer Graph). *For any $a, r, c > 0 \in \mathbb{Z}$, there exists a graph $G_O(a,r,c) = (V_O, E_O)$ with a set of critical paths Π_O that has the following properties:*
1. *The number of nodes, edges, and critical paths in G_O is:*

$$|V_O| = \Theta(a^3 r), \quad |E_O| = \Theta\left(\frac{a^3 r^2}{c^2}\right), \quad |\Pi_O| = \Theta\left(\frac{a^2 r^4}{c^5}\right).$$

2. *Every critical path $\pi \in \Pi_O$ is a unique shortest path in G_O of length at least $|\pi| \geq \frac{a}{4r}$.*
3. *Every pair of distinct critical paths π_1 and π_2 intersect on at most two nodes.*
4. *Every edge $e \in E_O$ lies on some critical path in Π_O.*

Just like in the original construction of G_O in Section 4, every critical path $\pi \in \Pi_O$ corresponds to a unique vector $\vec{w} \in W(r,c)$. Specifically, by the definition of $W_1(r,c), W_2(r,c)$, and $W(r,c)$, path π is constructed using vectors $\vec{w}_1 \in W_1(r,c)$ and $\vec{w}_2 \in W_2(r,c)$, where
- $\vec{w} = \vec{w}_1 + \vec{w}_2$, and
- $\vec{w}_1 \in \mathcal{S}_i^1$ and $\vec{w}_2 \in \mathcal{S}_i^2$, for some $i \in [1,c]$.

Critically, $\vec{w}_1$ and $\vec{w}_2$ both lie in the ith stripe $\mathcal{S}_i^1$ and $\mathcal{S}_i^2$, respectively.

5.2 Inner Graph G_I

In this subsection, we formally state the properties of the family of graphs we choose for our inner graphs G_I when constructing the obstacle product graph G. We will choose our inner graphs to be the sourcewise distance preserver lower bound graphs constructed in [12]. Lemma 19 formally captures the exact properties of this family of graphs that we need for spanner lower bound argument. We will defer our proof of Lemma 19 to the appendix, as it largely follows from the proof of Theorem 5.10 in [12].

▶ **Lemma 19** (cf. Theorem 5.10 of [12]). *For any $a, c > 0 \in \mathbb{Z}$, there exists a graph $G_I(a,c) = (V_I, E_I)$ with a set $S_I \subseteq V_I$ of sources, a set $T_I \subseteq V_I$ of sinks, and a set $P_I \subseteq S_I \times T_I$ of critical pairs that has the following properties:*

1. *The number of nodes, edges, sources, sinks, and critical pairs in G_I is:*

$$|V_I| = \Theta(a^2),$$
$$|E_I| = \Theta(a^2 c),$$
$$|S_I| = \Theta(a^{1/2} c^{11/4}),$$
$$|T_I| = \Theta(a^{1/2} c^{11/4}),$$
$$|P_I| = \Theta(a c^{5/2}).$$

2. *Every path $\pi_{s,t}$, where $(s,t) \in P_I$, contains $\Theta(a/c^{3/2})$ edges that do not lie on any other path $\pi_{s',t'}$, where $(s',t') \in P_I$.*
3. *For every source $s \in S_I$ and sink $t \in T_I$, the distance between s and t in G_I satisfies the following:*

$$dist_{G_I}(s,t) = \Theta(ac^{1/2}).$$

4. *The set of sources S_I can be partitioned into $b = \Theta(c^3)$ sets $S_I^1, \ldots, S_I^b$, where $|S_I^i| = \Theta(a^{1/2}c^{-1/4})$ for all $i \in [b]$. Let $T_I^i = \{t \in T_I \mid (S_I^i \times \{t\}) \cap P_I \neq \emptyset\}$ be the set of all sinks that belong to a critical pair with a source in S_I^i. Then for all $i \in [b]$ the following properties hold:*
   - *$|T_I^i| = \Theta(a^{1/2}c^{-1/4})$ for all $i \in [b]$,*
   - *$S_I^i \times T_I^i \subseteq P_I$, and*
   - *for all $(s,t) \in P_I$ such that $s \in S_I^i$ and $t \in T_I^i$,*

$$dist_{G_I}(s,t) \leq dist_{G_I}(S_I^i, T_I^i),$$

   where $dist_{G_I}(S_I^i, T_I^i)$ denotes the minimum distance between S_I^i and T_I^i in G_I.

5.3 Construction of Obstacle Product Graph G

Let $a, r, c > 0 \in \mathbb{Z}$ be the input parameters of an instance of outer graph $G_O = G_O(a, r, c)$. Let $W_1 = W_1(r, c)$, $W_2 = W_2(r, c)$, and $W = W(r, c)$ be the sets of vectors constructed in Section 5.1. Additionally, let $a', c' > 0 \in \mathbb{Z}$ be the input parameters of an instance of inner graph $G_I = G_I(a', c')$. We will specify the precise values of a, r, c, a', and c' later, as needed. Roughly, our choices of parameters a, r, and a' will grow with the size of our final graph G, while parameters c and c' will be (sufficiently large) integer constants.

We will construct our final graph G by performing the obstacle product. The obstacle product is performed in two steps: the edge subdivision step and the inner graph replacement. In the inner graph replacement step, we will need to carefully define two functions, $\phi_1 : W_1 \mapsto S_I \times T_I$ and $\phi_2 : W_2 \mapsto S_I \times T_I$ between vectors in W_1 and W_2 and pairs of nodes in $S_I \times T_I$ in inner graph G_I.

Edge Subdivision

We subdivide each edge in G_O into a path of length ψ. Denote the resulting graph as G'_O. For any edge $e = (u, v) \in E_O$, let P_e denote the resulting $u \rightsquigarrow v$ path of length ψ. We will take $\psi = \Theta\left(\frac{r^3}{c^{29/3}}\right)$.

Inner Graph Replacement

We now perform the inner graph replacement step of the obstacle product.

- For each node v in $V(G'_O)$ originally in G_O, replace v with a copy of G_I. We refer to this copy of G_I as G_I^v. Likewise, refer to the sources and sinks S_I and T_I in G_I^v as S_I^v and T_I^v.
- After applying the previous operation, the endpoints of the subdivided paths P_e in G'_O no longer exist in the graph. If $e = (u, v) \in E_O$, then P_e will have endpoints u and v. We will replace the endpoints u and v of P_e with nodes in G_I^u and G_I^v, respectively.
- In order to precisely define this replacement operation, it will be helpful to define two functions, $\phi_1 : W_1 \mapsto S_I \times T_I$ and $\phi_2 : W_2 \mapsto S_I \times T_I$. For ease of understanding, we will first assume the existence functions ϕ_1 and ϕ_2. We will specify our choices of ϕ_1 and ϕ_2 later.
- Let $e = (u, v) \in E_O$. If $v - u \in W_1$, then let $\phi_1(v - u) = (x, y) \in S_I \times T_I$. We will replace the endpoints u and v of P_e with nodes $y \in T_I^u$ in G_I^u and $x \in S_I^v$ in G_I^v, respectively. Otherwise, if $v - u \in W_2$, then let $\phi_2(v - u) = (x, y) \in S_I \times T_I$, and replace the endpoints u and v of P_e with nodes $y \in T_I^u$ in G_I^u and $x \in S_I^v$ in G_I^v, respectively. We repeat this operation for each $e \in E_O$ to obtain the obstacle product graph G.
- Note that after performing the previous operation, every subdivided path P_e, where $e = (u, v)$, will have a start node in T_I^u and an end node in S_I^v. We will use t_e to denote the start node of P_e in T_I^u and s_e to the end node of P_e in S_I^v.

This completes the construction of the obstacle product graph G (up to defining functions ϕ_1 and ϕ_2).

Defining functions ϕ_1 and ϕ_2

Let $\mathcal{S}_1^1, \ldots, \mathcal{S}_c^1$ be the stripes of W_1, and let $\mathcal{S}_1^2, \ldots, \mathcal{S}_c^2$ be the stripes of W_2. Let $S_I^1, \ldots, S_I^b$ and $T_I^1, \ldots, T_I^b$ be the partition of sources S_I and sinks T_I as described in Lemma 19, where $b = \Theta(c'^4)$. In order to construct our desired functions, we will require the following relations to hold:

- $b \geq c$,
- $|S_I^i| \geq |\mathcal{S}_i^j|$, for all $i \in [1, c]$ and $j \in \{1, 2\}$, and
- $|T_I^i| \geq |\mathcal{S}_i^j|$, for all $i \in [1, c]$ and $j \in \{1, 2\}$.

This can be achieved by setting

$$c' = \Theta(c^{1/3}) \quad \text{and} \quad a' = \Theta\left(\frac{r^2}{c^{35/6}}\right),$$

using the fact that $|\mathcal{S}_i^j| = \Theta(r/c^3)$, $|S_I^i| = \Theta(a'^{1/2}c'^{-1/4})$, and $|T_I^i| = \Theta(a'^{1/2}c'^{-1/4})$ by Claim 15 and Lemma 19.

We are now ready to define our functions ϕ_1 and ϕ_2. Let $\vec{w}_{i,j}^k$ denote the jth vector of $\mathcal{S}_i^k$, where $i \in [1, c]$, $j \in [1, |\mathcal{S}_i^k|]$, and $k \in \{1, 2\}$. Let s_j^i denote the jth node of S_I^i, where $i \in [1, c]$ and $j \in [1, |S_I^i|]$. Likewise, let t_j^i denote the jth node of T_I^i, where $i \in [1, c]$ and $j \in [1, |T_I^i|]$. We define ϕ_1 and ϕ_2 as follows:

$$\phi_1(\vec{w}_{i,j}^1) = (s_j^i, t_j^i) \qquad \text{and} \qquad \phi_2(\vec{w}_{i,j}^2) = (s_j^i, t_j^i) \qquad \text{for } i \in [1, c],\ j \in [1, |\mathcal{S}_j^1|].$$

The key properties of functions ϕ_1 and ϕ_2 are summarized in Claim 20.

▷ Claim 20. Functions ϕ_1 and ϕ_2 satisfy the following properties:

1. Our choice of ϕ_1 and ϕ_2 imply that for each node $u \in S_I \cup T_I$ in an inner graph copy G_I^v, there is at most one subdivided path P_e incident to u in G.
2. $\phi_k(\mathcal{S}_i^1) \subseteq S_I^i \times T_I^i$ for $k \in \{1, 2\}$ and $i \in [1, c]$, where $\phi_k(\mathcal{S}_i^1)$ denotes the image of $\mathcal{S}_i^1$ under ϕ_k.

Critical Paths Π

- Fix a critical path $\pi_O \in \Pi_O$ with associated vectors $\vec{w}_1 \in W_1$ and $\vec{w}_2 \in W_2$.
- By our construction of G_O, there exists $\vec{w} \in W$ such that $\vec{w} = \vec{w}_1 + \vec{w}_2$ (see Footnote 4). Then by the construction of W there exists some index $i \in [1, c]$ such that every edge $(u, v) \in \pi_O$ satisfies $v - u \in \{\vec{w}_1, \vec{w}_2\} \subseteq \mathcal{S}_i^1 \cup \mathcal{S}_i^2$. Let $\chi \in [1, c]$ denote this index.
- Let e_i denote the ith edge of π_O for $i \in [1, k]$. Note that by Property 2 of Claim 20, it follows that $s_{e_i} \in S_I^\chi$ and $t_{e_i} \in T_I^\chi$ for $i \in [1, k]$. Then by Property 5 of Lemma 19, $(s_{e_i}, t_{e_{i+1}}) \in P_I$. By Property 2 of Lemma 19, path $\pi_{s_{e_i}, t_{e_{i+1}}}$ is a unique shortest $s_{e_i} \rightsquigarrow t_{e_{i+1}}$ path in G_I.
- We now define a corresponding path π in G:

$$\pi = P_{e_1} \circ \pi_{s_{e_1}, t_{e_2}} \circ P_{e_2} \circ \cdots \circ P_{e_{k-1}} \circ \pi_{s_{e_{k-1}}, t_{e_k}} \circ P_{e_k},$$

  Note that if $e_i = (x, y)$ and $e_{i+1} = (y, z)$, then $\pi_{s_{e_i}, t_{e_{i+1}}}$ corresponds to the unique shortest path between $s_{e_i} \in S_I^\chi$ and $t_{e_i} \in T_I^\chi$ in inner graph copy G_I^y. We add path π to our set of critical paths Π.
- We repeat this process for all critical paths in Π_O to obtain our set of critical paths Π in G. Each critical path $\pi \in \Pi$ is uniquely constructed from a critical path $\pi_O \in \Pi_O$, so $|\Pi| = |\Pi_O|$. We will use $\phi : \Pi \mapsto \Pi_O$ to denote the bijection between Π and Π_O implicit in the construction.

As the final step in our construction of obstacle product graph G, we remove all edges in G that do not lie on some critical path $\pi \in \Pi$. Note that this will only remove edges in G that are inside copies of the inner graph G_I. Theorem 21 summarizes some of the key properties of obstacle product graph G. The proof of Theorem 21 follows from straightforward calculations and arguments similar to those in Section 5.2 in the full paper.

▶ **Theorem 21** (Properties of Obstacle Product Graph). *For any $a, r, c > 0 \in \mathbb{Z}$, there exists a graph $G(a, r, c) = (V, E)$ with a set of critical paths Π that has the following properties:*

1. *The number of nodes, edges, and critical paths in G is:*

$$|V| = \Theta(a^3 r^5 c^{-23/2}),$$
$$|E| = \Theta\left(c^{1/4} \cdot |V|\right),$$
$$|\Pi| = \Theta\left(\frac{a^2 r^4}{c^5}\right).$$

2. *Every path $\pi \in \Pi$ that passes through inner graph copy G_I^v contains $\Theta(a'/c'^{3/2})$ edges that do not lie on any other path $\pi' \in \Pi$, for all $v \in V_O$.*

For the full analysis of our spanner construction, please refer to the full version of the paper on ArXiv.

References

1 Amir Abboud and Greg Bodwin. The 4/3 additive spanner exponent is tight. *Journal of the ACM (JACM)*, 64(4):1–20, 2017.

2 Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. *SIAM Journal on Computing*, 47(6):2203–2236, 2018.

3 Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999.

4 Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993.

5 Imre Bárány and David G. Larman. The convex hull of the integer points in a large ball. *Math. Ann.*, 312(1):167–181, 1998. doi:10.1007/s002080050217.

6 Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. Additive spanners and (α, β)-spanners. *ACM Transactions on Algorithms (TALG)*, 7(1):1–26, 2010.

7 Greg Bodwin and Gary Hoppenworth. New additive spanner lower bounds by an unlayered obstacle product. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 – November 3, 2022*, pages 778–788. IEEE, 2022. doi:10.1109/FOCS54457.2022.00079.

8 Greg Bodwin and Virginia Vassilevska Williams. Better distance preservers and additive spanners. *ACM Transactions on Algorithms (TALG)*, 17(4):1–24, 2021.

9 Gregory Bodwin and Virginia Vassilevska Williams. Very sparse additive spanners and emulators. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 377–382, 2015.

10 Béla Bollobás, Don Coppersmith, and Michael Elkin. Sparse distance preservers and additive spanners. *SIAM Journal on Discrete Mathematics*, 19(4):1029–1055, 2005.

11 Shiri Chechik. New additive spanners. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 498–512. SIAM, 2013.

12 Don Coppersmith and Michael Elkin. Sparse sourcewise and pairwise distance preservers. *SIAM J. Discret. Math.*, 20(2):463–501, February 2006. doi:10.1137/050630696.

13 Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000.

14 William Hesse. Directed graphs requiring large numbers of shortcuts. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 665–669, USA, 2003. Society for Industrial and Applied Mathematics.

15 Gary Hoppenworth. Simple linear-size additive emulators. In *2024 Symposium on Simplicity in Algorithms (SOSA)*, pages 1–8. SIAM, 2024.

16 Shang-En Huang and Seth Pettie. Lower Bounds on Sparse Spanners, Emulators, and Diameter-reducing shortcuts. In David Eppstein, editor, *16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018)*, volume 101 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:12, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SWAT.2018.26.

17 Shimon Kogan and Merav Parter. New additive emulators. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

18 Kevin Lu. *New methods for approximating shortest paths.* PhD thesis, Massachusetts Institute of Technology, 2019.

19 Kevin Lu, Virginia Vassilevska Williams, Nicole Wein, and Zixuan Xu. Better lower bounds for shortcut sets and additive spanners via an improved alternation product. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3311–3331, 2022.

20 Seth Pettie. Low distortion spanners. *ACM Transactions on Algorithms (TALG)*, 6(1):1–22, 2009.

21 Zihan Tan and Tianyi Zhang. Almost-optimal sublinear additive spanners. *arXiv preprint*, 2023. arXiv:2303.12768.

22 Virginia Vassilevska Williams, Yinzhan Xu, and Zixuan Xu. Simpler and higher lower bounds for shortcut sets. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 2643–2656. SIAM, 2024. doi:10.1137/1.9781611977912.94.

23 David P Woodruff. Lower bounds for additive spanners, emulators, and more. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 389–398. IEEE, 2006.

A Tight Monte-Carlo Algorithm for Steiner Tree Parameterized by Clique-Width

Narek Bojikian
Humboldt-Universität zu Berlin, Germany

Stefan Kratsch
Humboldt-Universität zu Berlin, Germany

Abstract

Given a graph $G = (V, E)$, a set $T \subseteq V$, and an integer $\bar{b}$, the Steiner Tree problem asks whether G has a connected subgraph H with at most $\bar{b}$ vertices that spans all of T. This work presents a $3^k \cdot n^{\mathcal{O}(1)}$ time one-sided Monte-Carlo algorithm for solving Steiner Tree when additionally a clique-expression of width k is provided. Known lower bounds for less expressive parameters imply that this dependence on the clique-width of G is optimal assuming the Strong Exponential-Time Hypothesis (SETH). Indeed our work establishes that the parameter dependence of Steiner Tree is the same for any graph parameter between cutwidth and clique-width, assuming SETH.

Our work contributes to the program of determining the exact parameterized complexity of fundamental hard problems relative to structural graph parameters such as treewidth, which was initiated by Lokshtanov et al. [SODA 2011 & TALG 2018] and which by now has seen a plethora of results. Since the cut-and-count framework of Cygan et al. [FOCS 2011 & TALG 2022], connectivity problems have played a key role in this program as they pose many challenges for developing tight upper and lower bounds. Recently, Hegerfeld and Kratsch [ESA 2023] gave the first application of the cut-and-count technique to problems parameterized by clique-width and obtained tight bounds for Connected Dominating Set and Connected Vertex Cover, leaving open the complexity of other benchmark connectivity problems such as Steiner Tree and Feedback Vertex Set.

Our algorithm for Steiner Tree does not follow the cut-and-count technique and instead works with the connectivity patterns of partial solutions. As a first technical contribution we identify a special family of so-called complete patterns that has strong (existential) representation properties, and using these at least one solution will be preserved. Furthermore, there is a family of 3^k basis patterns that (parity) represents the complete patterns, i.e., it has the same number of solutions modulo two. Our main technical contribution, a new technique called "isolating a representative," allows us to leverage both forms of representation (existential and parity). Both complete patterns and isolation of a representative will likely be applicable to other (connectivity) problems.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized complexity, Steiner tree, clique-width

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.29

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2307.14264` [5]

1 Introduction

This work presents a $3^k \cdot n^{\mathcal{O}(1)}$ time one-sided Monte-Carlo algorithm for Steiner Tree[cw], i.e., Steiner Tree parameterized by clique-width.[1] Under the Strong Exponential-Time Hypothesis (SETH)[2] this dependence on the clique-width is optimal, i.e., time $(3-\varepsilon)^k \cdot n^{\mathcal{O}(1)}$

[1] Given a graph $G = (V, E)$, a set $T \subseteq V$ of terminals, a number $\bar{b}$, and a k-clique-expression of G, with k being the parameter, is there a connected subgraph H of G with at most $\bar{b}$ vertices that spans all of T?

[2] SETH posits that for each $c < 2$ there is $q \in \mathbb{N}$ such that q-CNF SAT cannot be solved in time $\mathcal{O}(c^n)$.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 29; pp. 29:1–29:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is known to be impossible [15]. Our algorithm relies on a new technique, called *isolating a representative*, that allows it to employ both counting modulo two and reduction to a specific smaller family of representative partial solutions. Implicitly, it thus works with a reduced compatibility matrix of $GF(2)$-rank 3^k, rather than the full matrix of $GF(2)$-rank 4^k, which is key for obtaining the claimed tight running time.

Our work fits into the program of determining the exact complexity of parameterized problems (modulo SETH), especially relative to width parameters like treewidth, which was initiated by Lokshtanov et al. [35, 36]. Since their work, there has been a plethora of such tight bounds relative to treewidth, see, e.g., [7, 13, 12, 14, 40, 26, 37, 41, 18, 22, 32, 21], but also relative to other parameters such as cutwidth [45, 30, 37, 44, 24, 4], modular treewidth [34, 28], and clique-width [29, 13, 31, 34, 23, 27]. An important breakthrough was made by Cygan et al. [15, 17], who were the first to obtain tight complexity bounds for connectivity problems such as Steiner Tree, Connected Vertex Cover, and Feedback Vertex Set (all parameterized by treewidth). In particular, their novel cut-and-count paradigm led to the surprising result that many of these problems admit $\mathcal{O}^*(c^{tw})$ time (randomized) dynamic programming (DP) algorithms. Intuitively, this is done by counting cuts of relaxed solutions modulo two and relying on cancellation of disconnected solutions. Note that this makes such algorithms inherently probabilistic as they rely on the isolation lemma to reduce, with high probability, to the case that there is a unique solution of minimum weight. In this way, they are able to reduce decision to counting modulo two.

The exact complexity of Steiner Tree[cw] is a natural and important question: Steiner Tree is central among connectivity problems, which have been key benchmarks in the exact complexity program. Clique-width is a well-studied graph parameter that, in particular, also allows to capture structure present in dense graphs (unlike treewidth and cutwidth etc.). So far, there are only two results regarding exact complexity (modulo SETH) of connectivity problems relative to clique-width: Hegerfeld and Kratsch [27] showed tight bounds of $6^k \cdot n^{\mathcal{O}(1)}$ for Connected Vertex Cover[cw] and $5^k \cdot n^{\mathcal{O}(1)}$ for Connected Dominating Set[cw], with both algorithms relying on cut-and-count. They explicitly leave open the exact complexities of Steiner Tree[cw], Connected Odd Cycle Transversal[cw], and Feedback Vertex Set[cw] as not being within range of their techniques. For the latter, the algorithm for Feedback Vertex Set[tw] [17] counts edges outside the tentative feedback vertex set to ensure that no cycles remain, but this seems infeasible relative to clique-width (cf. [1, 2]). For the former two, it is mentioned that their approach via cut-and-count would give time $4^k \cdot n^{\mathcal{O}(1)}$ for Steiner Tree[cw] and time $14^k \cdot n^{\mathcal{O}(1)}$ for Connected Odd Cycle Transversal[cw]. However, matching lower bounds were not in sight, as current lower bound methods rely on sufficiently large triangular submatrices inside certain compatibility matrices that correspond to the problem, which were lacking. Indeed, it is this gap between upper and lower bound that our technique allows us to close.

Our result and techniques. Formally, our result for Steiner Tree[cw] reads as follows.

▶ **Theorem 1.** *There is a one-sided error Monte Carlo algorithm (no false positives) that, given a graph $G = (V, E)$, a set of terminals $T \subseteq V$, a number $\bar{b}$, and a k-clique-expression of G, takes time $3^k \cdot n^{\mathcal{O}(1)}$ and determines, with high probability, whether a connected subgraph H of G of exactly $\bar{b}$ vertices exists that spans all of T.*

Since the clique-width of any graph G is at most its pathwidth plus two (folklore), i.e., $\mathrm{cw}(G) \leq \mathrm{pw}(G)+2$, the lower bound for Steiner Tree[pw] [15] rules out time $(3-\varepsilon)^k \cdot n^{\mathcal{O}(1)}$ for Steiner Tree[cw]. In fact, recent work of Bojikian et al. [4] shows that the same lower

Table 1 Tight complexity bounds (modulo SETH) for a selection of (connectivity) problems relative to cutwidth, treewidth, modular-treewidth, and clique-width. (Table adapted from [27].)

	cutwidth	treewidth	modular-tw	clique-width
q-Coloring	$\mathcal{O}^*(2^k)$	$\mathcal{O}^*(q^k)$	$\mathcal{O}^*(\binom{q}{\lfloor q/2 \rfloor}^k)$	$\mathcal{O}^*((2^q-2)^k)$
Vertex Cover	$\mathcal{O}^*(2^k)$	$\mathcal{O}^*(2^k)$	$\mathcal{O}^*(2^k)$	$\mathcal{O}^*(2^k)$
Connected Vertex Cover	$\mathcal{O}^*(2^k)$	$\mathcal{O}^*(3^k)$	$\mathcal{O}^*(5^k)$	$\mathcal{O}^*(6^k)$
Connected Dominating Set	$\mathcal{O}^*(3^k)$	$\mathcal{O}^*(4^k)$	$\mathcal{O}^*(4^k)$	$\mathcal{O}^*(5^k)$
Steiner Tree	$\mathcal{O}^*(3^k)$	$\mathcal{O}^*(3^k)$	$\mathcal{O}^*(3^k)$	$\boldsymbol{\mathcal{O}^*(3^k)}$
Feedback Vertex Set	$\mathcal{O}^*(2^k)$	$\mathcal{O}^*(3^k)$	$\mathcal{O}^*(5^k)$	?
Connected Odd Cycle Transversal	$\mathcal{O}^*(4^k)$	$\mathcal{O}^*(4^k)$	?	?
References	[4, 30, 45]	[16, 17, 36]	[28, 34]	[27, 34]

bound already holds relative to cutwidth. Thus, for all parameters between cutwidth and clique-width we get the same tight dependence of $3^k \cdot n^{\mathcal{O}(1)}$ for solving Steiner Tree, which contrasts the growing dependence exhibited by most other problems (see Table 1).

Our algorithm does not apply cut-and-count, i.e., counting consistent cuts of relaxed partial solutions. Instead, it works with the connectivity patterns of partial solutions for the graphs that correspond to parts of the given k-clique-expression. Roughly, a connectivity pattern indicates how the connected components of a partial solution connect the different label classes, noting that several components may intersect the same label class. Of course, the full set of possible patterns is much too large to be of use for us. Instead, we identify a family of patterns, called *complete patterns*, that are representative for the class of all patterns in a strong and constructive sense: For each pattern p there is a set $R_p = \{q_1, \ldots, q_\ell\}$ of complete patterns that together completes into exactly the same set of solutions as p does. Moreover, we identify a subfamily of 3^k basis patterns such that for each complete pattern there is a set of basis patterns that completes into the same number *modulo two* of solutions; again, this necessitates randomization through the use of the isolation lemma to guarantee existence of a unique minimum weight solution.

Now, if we simply combine these two means of representation, namely existential and modulo two, we will not end up with a correct algorithm: Existential representation makes no guarantee about the number, and hence the parity, of representations, even if there is a unique global solution. This is where our technique of *isolating a representative* comes in: When a step in the DP creates a partial solution whose connectivity pattern p is not complete, then each representing complete pattern has its weight increased by a globally set random weight (with four weights per DP step sufficing for our algorithm). In this way, a unique minimum weight representation is isolated with high probability. Combined with the usual setup of isolating a unique solution of minimum weight (and fast convolution), this allows our algorithm to count modulo two the partial solutions relative to the 3^k basis patterns and arrive at the correct answer with high probability. Overall, we have two applications of the isolation lemma, once for isolating a solution and once for isolating a representation of the solution. This makes our algorithm inherently probabilistic, though with low error bounds, and it may fail (like preceding work) by returning a false negative when either isolation fails. Success chance (of this one-sided error algorithm) can be boosted in the usual way.

Our approach can be easily generalized to solve the vertex-weighted version of this problem, when weights are bounded polynomially by the size of the graph, in the obvious way with weights replacing cardinalities of partial solutions. This contrasts the status of the edge-weighted version of this problem, which can be easily seen to be para-NP-hard when

parameterized by clique-width: One can reduce any instance of the unweighted Steiner Tree problem to the edge-weighted version of this problem on a complete graph (of clique-width 2) by assigning weight one to existing edges, and some large weight to all other edges.

Related work. It has been observed that many results on exact parameterized complexity are closely related to properties of certain matrices that are related to the problem in question and the considered graph structure, and this was recently surveyed by Nederlof [39]: E.g., methods such as cut-and-count rely on low-rank factorizations of the corresponding matrices. Since cut-and-count works modulo two, i.e., over $GF(2)$, and since it can be verified that the relevant matrix for Steiner Tree[cw] has $GF(2)$-rank (at least) 4^k, this makes it unlikely that a pure cut-and-count approach could succeed in getting time $3^k \cdot n^{\mathcal{O}(1)}$. (Indeed, it was this fact combined with the intuition that current lower bound techniques would not stretch further than the $(3-\varepsilon)^k \cdot n^{\mathcal{O}(1)}$, known already relative to pathwidth, that motivated the present work.) This leaves open the possibility that the same time bound of $3^k \cdot n^{\mathcal{O}(1)}$ could be obtained by working over a different field and/or studying the support rank[3] of the matrix (cf. [39]). So far, there are few results that obtain tight complexity bounds that do not coincide with the rank of the corresponding matrices [30, 4].

Clique-width was introduced by Courcelle and Olariu [11] building on work of Courcelle et al. [9] and it is similar to the NLC-width of Wanke [46]. Courcelle et al. [10] showed that every graph problem expressible in MSO_1 logic (monadic second order logic of graphs with quantification over vertex sets but not edge sets) can be solved in linear time for graphs with a given k-clique-expression, i.e., in time $f(k) \cdot n$, though the function f depends on the formula capturing the problem and may be non-elementary (cf. [33]), so likely far from being tight. This tractability is not restricted to MSO_1-expressible problems but many other important problems are solvable in time $f(\mathrm{cw}) \cdot n^{\mathcal{O}(1)}$ or at least time $n^{f(\mathrm{cw})}$ (see, e.g., [19]). Nevertheless, some problems, like Disjoint Paths, are NP-complete on graphs of bounded clique-width, while being solvable in time $f(k) \cdot n^{\mathcal{O}(1)}$ with respect to treewidth (cf. [25]). The first single-exponential time algorithms for connectivity problems parameterized by clique-width were given by Bergougnoux and Kanté [1, 2], e.g., Steiner Tree, Connected Dominating Set, and Connected Vertex Cover each in time $2^{\mathcal{O}(\mathrm{cw})} \cdot n$, but building on the rank-based approach these bounds are likely not tight.

The main drawback of clique-width lies in the difficulty of finding good expressions, i.e., good bounds on the clique-width of given graphs, in reasonable time. Fellows et al. [20] showed that it is NP-complete to determine, on input of graph G and integer k, whether the clique-width of G is at most k. It is open, however, whether for each fixed value of k there is an efficient algorithm for recognizing graphs of clique-width at most k; such algorithms are known only for $k \leq 3$ [8]. In particular, there is neither an FPT-algorithm for Clique-Width[k] known, nor is it known to be W[1]-hard with respect to k. The arguably best way for using low clique-width is an exponential-ratio FPT-algorithm due to Seymour and Oum [43] (made faster by Oum [42]). That being said, better algorithms for computing clique-width are still possible, and there may be variants of clique-width that give similar complexity (used as parameters) but are easier to compute.

Organization. In Section 3 we introduce connectivity patterns and provide related notation. In Section 4 we show how to represent partial solutions through patterns. In Section 5 we show that one can represent arbitrary patterns using complete patterns only. In Section 6 we

[3] The smallest rank of any matrix with the same zero vs. non-zero pattern over any field, so this is only interesting for fields other than $GF(2)$.

show that one can count complete patterns by reducing them a smaller family of patterns. In Section 7 we show how to count these families efficiently, providing the main algorithm of this paper. Finally, we conclude with final thoughts in Section 8.

2 Preliminaries

In this work we deal with undirected graphs only. Given a natural number k, we denote by $[k] = \{1, 2, \dots k\}$ the set of natural numbers smaller than or equal to k, and $[k]_0 = [k] \cup \{0\}$. A *labeled graph* is a graph $G = (V, E)$ together with a *labeling function* $\text{lab}\colon V \to \mathbb{N}$. We usually omit the function lab and assume that it is given implicitly when defining a labeled graph G. We say that G is *k-labeled*, if it holds that $\text{lab}(v) \leq k$ for all $v \in V$. We define a *clique-expression* μ as a well-formed expression that consists only of the following operations on labeled graphs:

- *Create vertex* $i(v)$ for $i \in \mathbb{N}$. This operation constructs a graph containing a single vertex and assigns label i to this vertex.
- The *union* operation $G_1 \uplus G_2$. The constructed graph consists of the disjoint union of the labeled graphs G_1 and G_2.
- The *relabel* operation $\rho_{i\to j}(G)$ for $i, j \in \mathbb{N}$. This operations changes the labels of all vertices in G labeled i to the label j.
- The *join* operation $\eta_{i,j}(G)$ for $i, j \in \mathbb{N}, i \neq j$. The constructed graph results from G by adding all edges between the vertices labeled i and the vertices labeled j, i.e.

$$\eta_{i,j}(G) = (V, E \cup \{\{u, v\}\colon \text{lab}(u) = i \wedge \text{lab}(v) = j\}).$$

We denote the graph resulting from a clique-expression μ by G_μ, and the constructed labeling function by lab_μ. We associate with a clique-expression μ a syntax tree $\mathcal{T}_\mu$ (we omit the symbol μ when clear from the context) in the natural way, and associate with each node $x \in V(\mathcal{T})$ the corresponding operation. For $x \in V(\mathcal{T})$, the subtree rooted at x induces a subexpression μ_x. We define $G_x = G_{\mu_x}$, $V_x = V(G_x)$, $E_x = E(G_x)$ and $\text{lab}_x = \text{lab}_{\mu_x}$. Finally, we say that a clique-expression μ is a *k-expression* if G_x is a k-labeled graph for all $x \in V(\mathcal{T})$. We define the clique-width of a graph G (denoted by $\text{cw}(G)$) as the smallest value k such that there exists a k-expression μ with G_μ isomorphic to G. We can assume without loss of generality, that any given k-expression defining a graph $G = (V, E)$ uses at most $O(|V|)$ union operations, and at most $O(|V|k^2)$ unary operations [1, 11].

We define the Steiner Tree problem as follows: Given an undirected graph $G = (V, E)$, a terminal set $T \subseteq V$ and some value $\overline{b} \in \mathbb{N}$, asked is whether there exists a set of vertices $S \subseteq V$ of size at most $\overline{b}$, such that $T \subseteq S$ and $G[S]$ is connected. For $k \in \mathbb{N}$, and f some computable function, we denote by $\mathcal{O}^*(f(k))$ the running time $f(k)\,\text{poly}(n)$, where n is the size of the input.

Let $\mathbb{F}$ be a field, and let S be some set. For a vector $T \in \mathbb{F}^S$, and some value $x \in S$, we denote by $T[x]$ the element of T indexed by x. In general, we will only deal with binary vectors, i.e. $\mathbb{F} = \text{GF}(2)$. By $\overline{0}_S$ we denote the vector T with $T[x] = 0$ for all $x \in S$. We omit the subscript S when clear from the context. For two sets S, T, let $M \in \mathbb{F}^{S\times T}$ be some matrix, whose rows are indexed by S and columns by T. We denote by $M[s, t]$ the element of M indexed by s and t.

For some integer k, given sets $S_1, T_1, \dots, S_k, T_k$, let $M_i \in \mathbb{F}^{S_i\times T_i}$ for each $i \in [k]$. We define the Kronecker product of $M_1, \dots M_k$ as the matrix $M = M_1 \otimes \cdots \otimes M_k \in \mathbb{F}^{(S_1\times\cdots\times S_k)\times(T_1\times\cdots\times T_k)}$, where for $s_i \in S_i$ and $t_i \in T_i$ for all $i \in [k]$,

$$M[(s_1, \dots, s_k), (t_1, \dots, t_k)] = \prod_{i\in[k]} M_i[s_i, t_i].$$

It is well-known that $\text{rk}(M_1 \otimes \cdots \otimes M_k) = \text{rk}(M_1) \cdot \text{rk}(M_2) \cdot \cdots \cdot \text{rk}(M_k)$.

Given two sets S, T, we denote by $S \triangle T$ the symmetric difference of S and T. It holds by a simple counting argument that $|S \triangle T| \equiv_2 |S| + |T|$. Given a finite set U, the power-set of U (denoted by 2^U) is the set of all subsets of U. A partition is a family of pairwise disjoint subsets of U that span all its elements.

A *lattice* is a partial order $(\mathcal{L}, \preceq)$ over a set $\mathcal{L}$, such that any two elements have a greatest lower bound (called *meet*), and a least upper bound (called *join*). We call two lattices isomorphic, if the underlying orders are isomorphic.

Let $f : U \to V$ be a mapping between two sets U and V, and $x \notin U$ a new element. Let $i \in V$. We define the extension $f|_{x \mapsto i} : U \cup \{x\} \to V$ of f in the natural way. We also define the disjoint union of two mappings defined over two disjoint sets in the natural way.

Let U, V be two sets, and $f : U^k \to V$ a mapping. Given sets $S_1, \dots S_k \subseteq U$, we define

$$f(S_1, \dots S_k) = \{f(s_1, \dots s_k) \colon (s_1, \ \dots s_k) \in S_1 \times S_2 \times \dots S_k\}.$$

For a set $S \subseteq V$ we also define

$$f^{-1}(S) = \{(s_1, \ \dots s_k) \in S_1 \times S_2 \times \dots S_k \colon f(s_1, \dots s_k) \in S\}.$$

An exception to this notation, is when we explicitly mention that $f : U \to V$ is a weight function, and $V \subseteq \mathbb{Z}$. In this case, for some set $S \subseteq U$ we define $f(S) = \sum_{u \in S} f(u)$, where we compute the sum over $\mathbb{Z}$.

Given a logical expression ρ, we denote by $[\rho]$ the Iverson bracket of ρ. We refer the reader to the full version [5] for formal definitions of all mentioned notation.

3 Connectivity patterns

3.1 Definition and terminology

Along this work, let U be some totally ordered ground set. In this section we define the main structures we build this paper on, we call them patterns. A pattern, as we shall see, represents the state of a partial solution, and carries enough information to extend it to a solution over the whole graph.

▶ **Definition 2.** *Let* 0 *be an element not in* U*. A* pattern p *is a subset of the power-set of* $U \cup \{0\}$*, such that there exists exactly one set of* p *containing the element* 0*. With* $\mathcal{P}(U)$ *we denote the family of all patterns over* U*. We omit* U *when clear from the context. By* $Z_p \in p$ *we denote the only set of* p *containing the element* 0*, and we call it the* zero-set *of* p*.*

Given a pattern $p \in \mathcal{P}$*, we define* $\mathrm{label}(p) \subseteq U$ *as the set of all labels in* U *occurring in* p*, and with* $\mathrm{singleton}(p) \subseteq U$ *the set of all labels in* U *appearing as singletons in* p*, i.e.*

$$\mathrm{label}(p) = \bigcup_{S \in p} S \setminus \{0\}, \qquad \textit{and} \qquad \mathrm{singleton}(p) = \big\{u \colon \{u\} \in p \wedge u \neq 0\big\}.$$

▶ **Definition 3.** *We define a more concise way to write patterns, that we use quite often along this paper. For a pattern* $p = \{\{u_1^1, \dots, u_{r_1}^1\}, \dots, \{u_1^\ell, \dots, u_{r_\ell}^\ell\}\}$*, we write*

$$p = [u_1^1 u_2^1 \dots u_{r_1}^1, \dots, u_1^\ell \dots u_{r_\ell}^\ell],$$

where we use square brackets to enclose the pattern, we do not use any separator between the elements of the same set, and separate different sets with comas. We also sometimes omit the symbol 0 *when it appears as a singleton.*

When we use this notation, we assume that each element is represented by a single symbol. Hence, such patterns admit a unique interpretation. For example, both $[12]$ *and* $[0, 12]$ *denote the pattern* $\{\{0\}, \{1, 2\}\}$*, while the pattern* $\{\{0, 10\}\}$ *cannot be written in this concise way.*

Now we introduce pattern operations. These operations will allow us to extend families of patterns over the syntax tree of a clique-expression to build larger solutions recursively. We define these operations in a similar way to the operations on partitions defined in [3].

▶ **Definition 4.** *Let $p, q \in \mathcal{P}$. We define the following operations:*

Join: $r = p \sqcup q$. *Let* $\sim_I = \{(S,T) \colon S \in p \wedge T \in q \wedge S \cap T \neq \emptyset\}$ *be a relation over* $p \cup q$, *and* $\sim_I^T$ *be its reflexive, transitive and symmetric closure. Let* $\mathcal{R}$ *be the equivalence classes of* $\sim_I^T$, *then* $r = \{\bigcup_{S \in P} S \colon P \in \mathcal{R}\}$.

Relabel: $r = p_{i \to j}$, *for* $i, j \in U$, *where* r *results from* p *by replacing* i *with* j *in each set of* p.

Union: $r = p \oplus q$, *where* $r = (p \setminus \{Z_p\}) \cup (q \setminus \{Z_q\}) \cup \{Z_p \cup Z_q\}$.

▶ **Definition 5.** *For* $i, j \in U, i \neq j$, *we define the operation* $\square_{i,j} p$ *for* $p \in \mathcal{P}$ *as*

$$\square_{i,j} p = \begin{cases} p & : \{i,j\} \not\subseteq \text{label}(p), \\ p \sqcup [ij] & : \textit{otherwise.} \end{cases}$$

This combines all sets containing the labels i *or* j *into one set, if both labels appear in* p.

▶ **Definition 6.** *Let* $p, q \in \mathcal{P}$. *We say that* p *and* q *are* consistent *(denoted by* $p \sim q$*), if for* $r = p \sqcup q$ *it holds that* $r = \{Z_r\}$ *contains the zero-set only, i.e. the join operation merges all sets into one set only. We say that a pattern* p dominates *another pattern* q *(denoted by* $p \geq q$*), if it holds for all patterns* $r \in \mathcal{P}$ *that* $q \sim r$ *implies* $p \sim r$.

3.2 Characteristics of patterns

In the full version of this paper we introduce the notion of alternating walks between two patterns. Using this notion, we provide a new characterization of the join operation, upon which, we prove a multitude of properties of patterns. We use these properties to prove domination and consistency results on manipulated and derived patterns. In particular, this allows us to correctly reduce the states of a dynamic programming solution keeping the transitions correct. We present these properties here and refer the reader to the full version for a formal proof.

▶ **Lemma 7.** *Let* $p, q, r \in \mathcal{P}$ *be arbitrary patterns, and* $i, j \in U$ *two arbitrary labels, with* $i \neq j$. *Then all the following is true:*

1. *It holds that* $\square_{i,j} p \geq p$
2. *If* $\{i,j\} \subseteq \text{label}(p)$, *then* $p \sqcup [ij] \sim q$ *if and only if* $p \sim q \oplus [ij, i, j]$.
3. *If* $\text{singleton}(p) \setminus \text{label}(q) \neq \emptyset$, *then* $p \not\sim q$.
4. *For* $S \in p$, *and* $u \in S$, *let* $S' = S \setminus \{u\}$, *and* $p' = (p \setminus \{S\}) \cup \{S'\}$. *It holds that* $p \geq p'$.
5. *For* $S \in p$, $S' \subseteq S$, *and* $p' = p \cup \{S'\}$, *it holds that* $p \geq p'$.
6. *If* $p \sim q$ *holds, then it must hold that* $\text{label}(p) = \text{label}(q)$ *and* $\text{singleton}(p) = \text{singleton}(q)$.
7. *Let* q' *be the pattern that results from* q *by removing all labels in* $\text{label}(q) \setminus \text{label}(p)$ *from each set of* q. *Then* $p \sim q$ *if and only if* $p \sim q'$.

▶ **Definition 8.** *Given a pattern* $p \in \mathcal{P}$ *and two different labels* $i, j \in U$. *We define the operation* $p' = p_{j \curvearrowleft i}$ *over patterns, where* p' *results from* p *by removing the label* i *from all sets, and then adding* i *to each set that contains the label* j, *i.e.*

$$p_{j \curvearrowleft i} = \bigcup_{\substack{S \in p \\ j \notin S}} S \setminus \{i\} \cup \bigcup_{\substack{S \in p \\ j \in S}} S \cup \{i\}.$$

▶ **Lemma 9.** *For all $p, q, r \in \mathcal{P}$ the following two equivalences hold*
1. *It holds that $p \sqcup q \sim r$ if and only if $p \sim q \sqcup r$.*
2. *For $i, j \in U$ with $i \neq j$, it holds that $p_{i \to j} \sim q$ if and only if $p \sim q_{j \curvearrowleft i}$.*

▶ **Lemma 10.** *Given two patterns $p, q \in \mathcal{P}(U)$. Let U' be a new label set disjoint from U and ρ a one-to-one mapping from U to U'. Let q' be the pattern resulting from q by relabeling u to $\rho(u)$ for each $u \in U$. Then it holds for each pattern $r \in \mathcal{P}(U)$ that $p \oplus q \sim r$ if and only if $p \sim q' \sqcup r'$ for $r' \in \mathcal{P}(U \cup U')$ the pattern resulting from r by adding $\rho(u)$ to each set containing u for each $u \in U$.*

4 Partial solutions as patterns

From now on, let $(G, T, \bar{b})$ be the given instance of the Steiner Tree problem, for some graph $G = (V, E)$, a set of terminals $T \subseteq V$, and $\bar{b} \in \mathbb{N}$. Let $n = |V|$ and $m = |E|$. Let μ be a k-expression of G for some value $k \in \mathbb{N}$. We fix $U = [k]$. Let $\mathcal{T}$ be the corresponding syntax tree, and r its root. We also fix a value $W \in \mathbb{N}$, and a weight function $\mathbf{w} : V \to [W]$. Both will be chosen later. Let $v_0 \in T$ be an arbitrary but fixed terminal vertex. For a node $x \in V(\mathcal{T})$, we define $T_x = T \cap V_x$.

▶ **Definition 11.** *Let $x \in V(\mathcal{T})$. We call a set $S \subseteq V_x$ where $T_x \subseteq S$ a* partial solution. *Each partial solution defines a pattern $p = p_x(S)$ over G_x as follows: Let $C_1, \dots C_\ell$ be the connected components of $G_x[S]$. For $i \in [\ell]$, let $S_i = \mathrm{lab}_x(C_i)$ be the set of all labels appearing in C_i. If $v_0 \notin S$, we define $p = \{S_1, \dots, S_\ell\} \cup \{\{0\}\}$. Otherwise, assume that $v_0 \in C_\ell$ (otherwise swap C_ℓ and the component containing v_0). We define $p = \{S_1, \dots, S_{\ell-1}\} \cup \{S_\ell \cup \{0\}\}$. That means the zero-set is defined by the component containing v_0 if $v_0 \in S$, or as a singleton otherwise. We call $p_x(S)$ the pattern corresponding to S in G_x. We denote by $p(S)$ the pattern $p_r(S)$ corresponding to a partial solution S in the whole graph G.*

▶ **Lemma 12.** *Given a set $S \subseteq V$ such that $T \subseteq S$, it holds that S is a Steiner tree in G, if and only if $p(S)$ consists of one set only.*

We refer the reader to the full version for a formal proof. Now we define families of partial solutions that allow to build recursive formulas for a dynamic programming scheme over $\mathcal{T}$.

▶ **Definition 13.** *For $x \in V(\mathcal{T}), b \in [\bar{b}]_0, c \in [n \cdot W]_0$, we define the family $\mathcal{S}_x[b, c] \subseteq \mathcal{P}$ of patterns corresponding to partial solutions of cardinality b and weight c over G_x as*

$$\mathcal{S}_x[b, c] = \{p \in \mathcal{P} \colon \exists S \subseteq V_x, \text{ where } T_x \subseteq S \wedge p_x(S) = p \wedge |S| = b \wedge \mathbf{w}(S) = c\}.$$

From now on, we always assume that $x \in V(\mathcal{T}), b \in [\bar{b}]_0$, and $c \in [n \cdot W]_0$. We skip repeating this to avoid redundancy.

▶ **Lemma 14.** *The families $\mathcal{S}_x$ can be built recursively over the nodes of $\mathcal{T}$ in a bottom-up manner using the operations defined in Definition 4 and Definition 5.*

Proof. We sketch the recursive definitions of $\mathcal{S}_x$ for different types of nodes $x \in V(\mathcal{T})$, and refer to the full version for a complete proof of correctness.

For a create node $\mu_x = i(v)$ (a leaf of $\mathcal{T}$), let $c = \mathbf{w}(v)$. Let $p = [0, i]$ if $v \neq v_0$, and $p = [0i]$ otherwise. Then we set $\mathcal{S}_x[1, c] = \{p\}$. We set $\mathcal{S}_x[0, 0]$ to $\{[0]\}$, if $v \notin T$, and to $\emptyset$ otherwise. For all other values of b and c, we set $\mathcal{S}_x[b, c] = \emptyset$. We define $\mathcal{S}_x$ for the other cases as follows:

$$\begin{array}{lll} \text{Join node} & \mu_x = \eta_{i,j}(\mu_{x'}), & \text{then } \mathcal{S}_x[b,c] = \square_{i,j}\big(\mathcal{S}_{x'}[b,c]\big), \\ \text{Relabel node} & \mu_x = \rho_{i\to j}(\mu_{x'}), & \text{then } \mathcal{S}_x[b,c] = \big(\mathcal{S}_{x'}[b,c]\big)_{i\to j}, \\ \text{Union node} & \mu_x = \mu_{x_1} \uplus \mu_{x_2}, & \text{then } \mathcal{S}_x[b,c] = \bigcup_{\substack{b_1+b_2=b \\ c_1+c_2=c}} \big(\mathcal{S}_{x_1}[b_1,c_1] \oplus \mathcal{S}_{x_2}[b_2,c_2]\big). \end{array}$$

◀

▶ **Lemma 15.** *The graph G admits a Steiner tree of size $\overline{b}$ and weight $\overline{c}$, if and only if there exists a pattern $p \in \mathcal{S}_r\left[\overline{b},\overline{c}\right]$ with $p \sim [0]$.*

This lemma follows from Lemma 12 (see the full version for a proof). So far, we can solve the Steiner Tree problem by defining a dynamic programming over $\mathcal{T}$ that computes all families $\mathcal{S}_x\left[\overline{b},\overline{c}\right]$ for all values of $\overline{b}$ and $\overline{c}$. However, since there exist approximately 2^{2^k} different patterns over k labels, we seek to reduce the family of all patterns into a family of representing patterns in a sense that allow us to apply the dynamic programming scheme over the resulting families.

5 Pattern representation

Now we define a special family of patterns (called *complete patterns*), and show that we can represent any family of patterns by a family of complete patterns only.

5.1 Representation and complete patterns

▶ **Definition 16.** *Given two families $\mathcal{S},\mathcal{R} \subseteq \mathcal{P}$, we say that $\mathcal{R}$* represents *$\mathcal{S}$, if for each $q \in \mathcal{P}$ the following holds: there exists a pattern $p \in \mathcal{S}$ such that $p \sim q$ if and only if there exists a pattern $p' \in \mathcal{R}$ such that $p' \sim q$. Given a family $\mathcal{R} \subseteq \mathcal{P}$, and a pattern $p \in \mathcal{P}$, we say that $\mathcal{R}$ represents p if $\mathcal{R}$ represents $\{p\}$.*

Clearly, it holds that representation is an equivalence relation. In the full version, we define the notion of *representation-preserving* operations, and show that the operations defined in Definition 4 preserve representation. This allows to restrict a dynamic programming scheme to representing families, preserving the correctness of the transitions.

▶ **Definition 17.** *A pattern $p \in \mathcal{P}$ is* complete, *if* $\text{label}(p) = \text{singleton}(p)$, *i.e. a pattern p is complete, if each label of p appears as a singleton in p as well. We denote by $\mathcal{P}_C(U)$ the family of all complete patterns over U.*

▶ **Lemma 18.** *For all $p,q \in \mathcal{P}_C$ the following holds: $p \sim q$ implies that* $\text{label}(p) = \text{label}(q)$.

We refer to the full version for a proof.

▶ **Definition 19.** *Let $p \in \mathcal{P}$. For $i \in U$, we define the operations* $\text{fix}(p,i)$ *and* $\text{forget}(p,i)$ *as*

$$\text{fix}(p,i) = p \cup \big\{\{i\}\big\} \qquad \text{,and} \qquad \text{forget}(p,i) = \big\{S \setminus \{i\} \colon S \in p\big\},$$

if $i \in \text{label}(p) \setminus \text{singleton}(p)$. Otherwise, we define $\text{fix}(p,j) = \text{forget}(p,j) = p$. We say that $p' = \text{fix}(p,i)$ results from p by fixing *the label i, and $p'' = \text{forget}(p,i)$ results from p by* forgetting *the label i. Given a pattern $p \in \mathcal{P}$, we define* $\text{inc}(p) = \text{label}(p) \setminus \text{singleton}(p)$.

▶ **Definition 20.** *Given a pattern $p \in \mathcal{P}$. Let $\ell = |\text{inc}(p)|$, and $i_1, \ldots i_\ell$ be the elements of* $\text{inc}(p)$ *in an increasing order. Let $R_0 = \{p\}$, and $R_j = \text{forget}(R_{j-1}, i_j) \cup \text{fix}(R_{j-1}, i_j)$ for all $j \in [\ell]$. We define the family $R_p = R_\ell$ and call it a* complete representation *of p.*

The following lemmas show that R_p represents p, which proves that any set of patterns admits a representation of complete patterns only. We refer to the full version for proofs.

▶ **Lemma 21.** *Let $p \in \mathcal{P}$, and $i \in U$. It holds that $\{\mathrm{fix}(p,i), \mathrm{forget}(p,i)\}$ represents p.*

▶ **Lemma 22.** *It holds for all $p \in \mathcal{P}$ that $R_p \subseteq \mathcal{P}_C$, $|R_p| \leq 2^{|\operatorname{inc}(p)|}$, and R_p represents p.*

▶ **Corollary 23.** *Let $\mathcal{S} \subseteq \mathcal{P}$. Then the family $\mathcal{R} = \bigcup_{p\in\mathcal{S}} R_p \subseteq \mathcal{P}_C$ represents $\mathcal{S}$.*

▶ **Observation 24.** *It holds that the family of complete patterns is closed under both relabel operation $i \to j$ for all $i, j \in U$, and union operation $\oplus$. On the other hand, it holds for $i, j \in U, i \neq j$, $R \subseteq \mathcal{P}_C$, $R' = \square_{i,j} R$ and for each $p \in R'$ that $\operatorname{inc}(p) \in \{\emptyset, \{i,j\}\}$. If $\operatorname{inc}(p) = \{i,j\}$, then R_p contains at most four patterns, that result from p by fixing or forgetting either labels independently.*

5.2 Counting representations

▶ **Definition 25.** *Let $\mathrm{Op} : \mathcal{P}^k \to \mathcal{P}$ be a k-ary pattern operation. We define the operation $\overset{\triangle}{\mathrm{Op}} : \left(2^{\mathcal{P}}\right)^k \to 2^{\mathcal{P}}$ over subsets of $\mathcal{P}$ as $\overset{\triangle}{\mathrm{Op}}(S_1, \ldots, S_k) = \bigtriangleup_{\substack{(p_1,\ldots,p_k)\in \\ S_1\times\cdots\times S_k}} \{\mathrm{Op}(p_1, \ldots, s_k)\}$, for all $S_1, \ldots, S_k \subseteq \mathcal{P}$, and we call it the* exclusive *version of* Op. *We compare this operation to the notion $\mathrm{Op}(S_1, \ldots, S_k)$ given by the union over all resulting patterns from the operation. Given two sets of patterns $S_1, S_2 \subseteq \mathcal{P}$, we define $S_1 \overset{\triangle}{\sqcup} S_2, S_1 \overset{\triangle}{\oplus} S_2, S_{1_{i\overset{\triangle}{\to} j}}$ and $\overset{\triangle}{\square}_{i,j} S_1$ as the exclusive version of join, union, relabel and $\square_{i,j}$ of S_1 (and S_2) respectively.*

The following lemma can be proven by a simple counting argument. We refer to the full version for a formal proof.

▶ **Lemma 26.** *Let $\mathrm{Op} : \mathcal{P}^k \to \mathcal{P}$ be a k-ary pattern operation, and $S_1, \ldots S_k \subseteq \mathcal{P}$. It holds for $q \in \mathcal{P}$, that*

$$|\{p \in \overset{\triangle}{\mathrm{Op}}(S_1, \ldots S_k) \colon p \sim q\}| \equiv_2 |\{(p_1, \ldots p_k) \in S_1 \times \cdots \times S_k \colon \mathrm{Op}(p_1, \ldots p_k) \sim q\}|$$

As we have already mentioned, for $p \in \mathcal{P}_C$, and $p' = \square_{i,j} p$, it holds that either $R_{p'} = p'$, or $R_{p'}$ contains four patterns that result from p' by either forgetting or fixing both labels i and j independently. We can enumerate these patterns as follows:

▶ **Definition 27.** *We define a new operation called* actions $\mathrm{Ac} \colon \mathcal{P} \times [4] \to \mathcal{P}$ *as follows: Given $p \in \mathcal{P}$, if $p \in \mathcal{P}_C$, we define $\mathrm{Ac}(p, \ell) = p$, for all $\ell \in [4]$. For $\operatorname{inc}(p) = \{i\}$, we define*

- $\mathrm{Ac}(p, 1) = \mathrm{fix}(p, i)$,
- $\mathrm{Ac}(p, 2) = \mathrm{forget}(p, i)$.

We set $\mathrm{Ac}(p, 3)$ and $\mathrm{Ac}(p, 4)$ to undefined in this case. If $\operatorname{inc}(p) = \{i, j\}$ for $i < j$, we define

- $\mathrm{Ac}(p, 1) = \mathrm{fix}(\mathrm{fix}(p, i), j)$,
- $\mathrm{Ac}(p, 2) = \mathrm{forget}(\mathrm{fix}(p, i), j)$,
- $\mathrm{Ac}(p, 3) = \mathrm{fix}(\mathrm{forget}(p, i), j)$,
- $\mathrm{Ac}(p, 4) = \mathrm{forget}(\mathrm{forget}(p, i), j)$.

In all other cases we set $\mathrm{Ac}(p, \ell)$ to be undefined for all $\ell \in [4]$.

Using this enumeration, we can assign to each action at a create or a join node in the subtree of $\mathcal{T}_x$ a weight, which defines a different weight to each pattern in a complete pattern representation of the patterns in $\mathcal{S}_x \left[\overline{b}, \overline{c}\right]$.

▶ **Definition 28.** *For $x \in V(\mathcal{T})$ let $\pi : V(\mathcal{T}_x) \to [4]$ be some mapping, such that $\pi(x') \in [2]$ whenever x' is a create node, and $\pi(x') = 1$ whenever x' is a union node or a relabel node. We call π an* action sequence. *Let $D \in \mathbb{N}$ be some value that will be fixed later, and let $\mathbf{d} : V(\mathcal{T}) \times [4] \to [D]$ be a weight function. We define the weight of π as*

$$\mathbf{d}(\pi) = \sum_{\substack{x' \in V(\mathcal{T}_x),\\ x' \text{ is a join or create node}}} \mathbf{d}\big(x', \pi(x')\big).$$

Given a partial solution $S \subseteq V_x$ where $T_x \subseteq S$, the action sequence π defines a pattern p_S^π, that can be defined recursively over $\mathcal{T}_x$, where for a create node $i(v)$, let v be the vertex introduced at this node. If $v \notin S$, then we set $p_S^\pi = [0]$. Otherwise, we set $p_S^\pi = \mathrm{Ac}(p_x(\{v\}), \pi(x))$. For a relabel or union node, p_S^π results by applying the corresponding operation on the patterns resulting at the children of x in the natural way. For a join node $\mu_x = \eta_{i,j}(\mu_{x'})$, let $p' = \square_{i,j} p_S^{\pi'}$, where π' is the restriction of π to the nodes in $V(\mathcal{T}_{x'})$. We set $p_S^\pi = \mathrm{Ac}(p', \pi(x))$. We say that π is an action sequence generating the pattern p_S^π from S in G_x.

The proof of the following lemma follows by induction over $\mathcal{T}$ using transitivity of representation. We refer the reader to the full version for more details.

▶ **Lemma 29.** *Given a node $x \in V(\mathcal{T})$, and a partial solution $S \subseteq V_x$, where $T_x \subseteq S$. Let Π_x be the set of all action sequences at x. Let $P_x^S = \{p_S^\pi : \pi \in \Pi\}$. Then P_x^S represents $p_x(S)$.*

▶ **Definition 30.** *For $x \in V(\mathcal{T}), \overline{b} \in [n], \overline{c} \in [n \cdot W], \overline{d} \in [|V(\mathcal{T})| \cdot D]$, we define the families $\mathcal{D}_x\left[\overline{b}, \overline{c}, \overline{d}\right]$ as follows:*

- *Create node $i(v)$, let $c = \mathbf{w}(v)$. For $\ell \in [2]$, we set $\mathcal{D}_x\left[1, c, \mathbf{d}(x, \ell)\right] = \left\{\mathrm{Ac}\left(p_x(\{v\}), \ell\right)\right\}$, and set $\mathcal{D}_x\left[0, 0, \mathbf{d}(x, \ell)\right]$ to $\{[0]\}$ if $v \notin T$, or to $\emptyset$ otherwise. For all other values of $\overline{b}, \overline{c}$ and $\overline{d}$, we set $\mathcal{D}_x\left[\overline{b}, \overline{c}, \overline{d}\right] = \emptyset$.*
- *Relabel node $\mu_x = \rho_{i \to j}(\mu_{x'})$, then $\mathcal{D}_x\left[\overline{b}, \overline{c}, \overline{d}\right] = \left(\mathcal{D}_{x'}\left[\overline{b}, \overline{c}, \overline{d}\right]\right)_{i \overset{\triangle}{\to} j}$.*
- *Union node $\mu_x = \mu_{x_1} \uplus \mu_{x_2}$, then $\mathcal{D}_x\left[\overline{b}, \overline{c}, \overline{d}\right] = \triangle_{\substack{b_1+b_2=\overline{b}\\ c_1+c_2=\overline{c}\\ d_1+d_2=\overline{d}}} \left(\mathcal{D}_{x_1}\left[b_1, c_1, d_1\right] \overset{\triangle}{\oplus} \mathcal{D}_{x_2}\left[b_2, c_2, d_2\right]\right)$.*
- *Join node $\mu_x = \eta_{i,j}(\mu_{x'})$, we define the families $\mathcal{D}'_x\left[\overline{b}, \overline{c}, \overline{d}\right] = \overset{\triangle}{\square}_{i,j}\left(\mathcal{D}_{x'}\left[\overline{b}, \overline{c}, \overline{d}\right]\right)$. Now we set $\mathcal{D}_x\left[\overline{b}, \overline{c}, \overline{d}\right] = \triangle_{\ell \in [4]} \overset{\triangle}{\mathrm{Ac}}\left(\mathcal{D}'_x\left[\overline{b}, \overline{c}, \overline{d} - \mathbf{d}(x, \ell)\right], i\right)$.*

The following lemma shows that the families $\mathcal{D}_x$ correctly count action sequences. A formal proof of the lemma and the corollary following it can be found in the full version.

▶ **Lemma 31.** *For $x \in V(\mathcal{T})$, and all values $\overline{b}, \overline{c}$ and $\overline{d}$, it holds for $p \in \mathcal{P}_C$ that $p \in \mathcal{D}_x\left[\overline{b}, \overline{c}, \overline{d}\right]$ if and only if there exist odd many pairs (S, π) where $S \subseteq V_x$ with $T_x \subseteq S, |S| = \overline{b}, \mathbf{w}(S) = \overline{c}$ and π is an action sequence of weight $\overline{d}$ generating p from S in G_x.*

▶ **Corollary 32.** *If G does not admit a Steiner tree of size $\overline{b}$, then it holds for all values $\overline{c} \in [W \cdot n]$, $\overline{d} \in [D \cdot |V(\mathcal{T})|]$ that $[0] \notin \mathcal{D}_r\left[\overline{b}, \overline{c}, \overline{d}\right]$.*

5.3 Isolation Lemma

Now we fix $W = (2 + \sqrt{2})|V|$, and $D = 4(2 + \sqrt{2})|V(\mathcal{T})|$. We choose $\mathbf{w} \in [W]^V$ and $\mathbf{d} \in [D]^{V(\mathcal{T}) \times [4]}$ uniformly and independently at random. We use the isolation lemma to show that, if a Steiner tree of size $\overline{b}$ exists, then there exist two positive integers $\overline{c}$ and $\overline{d}$, such that, with high probability, there exists a unique solution S of size $\overline{b}$ and weight $\overline{c}$, and a unique action sequence of weight $\overline{d}$ that generates the pattern $[0]$ from S.

▶ **Definition 33.** *A function $\omega : U \to \mathbb{Z}$ isolates a set family $\mathcal{F} \subseteq 2^U$ if there exists a unique $S' \in \mathcal{F}$ with $\omega(S') = \min_{S \in \mathcal{F}} \omega(S)$.*

▶ **Lemma 34** ([38, 17])**.** *Let $\mathcal{F} \subseteq 2^U$ be a set family over a universe U with $|\mathcal{F}| > 0$, and let $N > |U|$ be an integer. For each $u \in U$, choose a weight $\omega(u) \in \{1, 2, \dots N\}$ uniformly and independently at random. Then it holds that $P[\omega \text{ isolates } \mathcal{F}] \geq 1 - |U|/N$.*

We use the following lemmas to show that, if a solution exists, then, with high probability, there exists a unique minimum weight representation of a minimum weight solution. We refer the reader to the full version for a formal proof.

▶ **Lemma 35.** *If a Steiner tree of size $\overline{b}$ exists, let $\overline{c}$ be the smallest weight of such a tree. Then there exists a unique Steiner tree of size $\overline{b}$ and weight $\overline{c}$ with probability at least $1 - \frac{1}{2+\sqrt{2}}$.*

▶ **Lemma 36.** *Let S be a Steiner tree in G, and π be a minimum weight action sequence generating $[0]$ from S in G. Then π is unique with probability at least $1 - \frac{1}{2+\sqrt{2}}$.*

▶ **Lemma 37.** *If G admits a Steiner tree of size $\overline{b}$, then with probability at least $1/2$ it holds that $[0] \in \mathcal{D}_r\left[\overline{b}, \overline{c}, \overline{d}\right]$ for some values of $\overline{c}$ and $\overline{d}$.*

6 Parity-representation

Now we define a new kind of representation, called the parity-representation. This allows to count complete patterns (modulo 2) that are consistent with a specific pattern.

▶ **Definition 38.** *We define the family of CS-patterns $\mathcal{P}_{CS}(U) \subseteq \mathcal{P}_C(U)$ as the family of complete patterns containing only a zero-set and singletons, i.e.*

$$\mathcal{P}_{CS}(U) = \Big\{\big\{X \cup \{0\}\big\} \cup \big\{\{u\} \colon u \in Y\big\} \colon X \subseteq Y \subseteq U\Big\}.$$

▶ **Observation 39.** *For each ground set U it holds that $|\mathcal{P}_{CS}(U)| = 3^{|U|}$.*

▶ **Observation 40.** *For $p, q \in \mathcal{P}_{CS}$ and all $i, j \in U$, it holds that $p_{i \to j}, p \oplus q \in \mathcal{P}_{CS}$.*

▶ **Definition 41.** *Given two sets of patterns $D, C \subseteq \mathcal{P}$, we say that C* parity-represents *D, if and only if for each $q \in \mathcal{P}_C$ it holds*

$$|\{p \in C \colon p \sim q\}| \equiv_2 |\{p \in D \colon p \sim q\}|.$$

For a set of patterns $C \subseteq \mathcal{P}$ and a single pattern $p \in \mathcal{P}$, we say that C parity-represents p if it holds that C parity-represents $\{p\}$.

▶ **Observation 42.** *Parity-representation is an equivalence relation.*

Now we show how to build a concrete parity-representation of an arbitrary complete pattern using CS-patterns only. We refer the reader to the full version for formal proofs.

▶ **Lemma 43.** *Given a pattern $p \in \mathcal{P}_C \setminus \mathcal{P}_{CS}$. Let $S \in p \setminus \{Z_p\}$ be an inclusion-wise minimal set, such that $|S| > 1$. Let $A = A_p^S$ be the family of patterns defined from p as follows $A_p^S = \big\{(p \setminus \{Z_p, S\}) \cup (Z_p \cup S') \colon S' \subset S\big\}$. Then A parity-represents of p.*

▶ **Lemma 44.** *Given a complete pattern $p \in \mathcal{P}_C$. Let $S_1, \dots S_r$ be the sets in p such that $S_i \neq Z_p$ and $|S_i| \geq 2$ for each $i \in [r]$. Let $A_0 = \{p\}$. For $i \in [r]$, we define $A_i = \triangle_{q \in A_{i-1}} A_q^{S_i}$. Let $A_p = A_r$. Then it holds that $A_p \subseteq \mathcal{P}_{CS}$, and that it parity-represents p.*

▶ **Corollary 45.** *For each family of complete patterns $D \subseteq \mathcal{P}_C$, there exists a family of CS-patterns $C \subseteq \mathcal{P}_{CS}$ that parity-represents D.*

Similar to pattern representation, in the full version of this paper we define the notion of *parity-representation-preserving* operations. We show that all defined operations (including actions and symmetric difference) preserve parity-representation. This allows to restrict a dynamic programming algorithm to a parity-representation of the underlying families.

Now we define families of partial solutions over $\mathcal{P}_{CS}$ that parity-represent the families $\mathcal{D}_x\left[\overline{b},\overline{c},\overline{d}\right]$. In the next section, we show that we can efficiently compute these families, and hence, solve the decision version of the Steiner Tree problem with high probability.

▶ **Definition 46.** *For $p \in \mathcal{P}_C$ let A_p be the set defined in Lemma 44. We define the operator $\mathrm{Ac}^* : \mathcal{P} \times 4 \to 2^{\mathcal{P}_{CS}}$ as $\mathrm{Ac}^*(p,\ell) = A_{\mathrm{Ac}(p,\ell)}$ if $\mathrm{Ac}(p,\ell) \in \mathcal{P}_C$ or as undefined otherwise. We denote the* exclusive *version of Ac^* by $\overset{\triangle}{\mathrm{Ac}}{}^*$.*

▶ **Definition 47.** *For $x \in V(\mathcal{T}), \overline{b} \in [n], \overline{c} \in [n \cdot W], \overline{d} \in [|V(\mathcal{T})| \cdot D]$, we define the families $\mathcal{C}_x\left[\overline{b},\overline{c},\overline{d}\right]$ as follows: For a create, relabel or union nodes x, we define the families $\mathcal{C}_x$ in an analogous way to $\mathcal{D}_x$, where we replace each $\mathcal{C}$ with $\mathcal{D}$ in the definition. For a join node $\mu_x = \eta_{i,j}(\mu_{x'})$, we define the families*

$$\mathcal{C}'_x\left[\overline{b},\overline{c},\overline{d}\right] = \overset{\triangle}{\square}_{i,j}\left(\mathcal{C}_{x'}\left[\overline{b},\overline{c},\overline{d}\right]\right) \quad \text{,and set} \quad \mathcal{C}_x\left[\overline{b},\overline{c},\overline{d}\right] = \bigtriangleup_{\ell\in[4]} \overset{\triangle}{\mathrm{Ac}}{}^*\left(\mathcal{C}'_x\left[\overline{b},\overline{c},\overline{d}-\mathbf{d}(x,\ell)\right],\ell\right).$$

In the full version, we show that $\mathcal{C}_x\left[\overline{b},\overline{c},\overline{d}\right]$ parity-represents $\mathcal{D}_x\left[\overline{b},\overline{c},\overline{d}\right]$ for all $x \in V(\mathcal{T})$ and all values $\overline{b},\overline{c},\overline{d}$. The following corollary follows

▶ **Corollary 48.** *It holds for all values $\overline{b},\overline{c},\overline{d}$ that $[0] \in \mathcal{C}_r\left[\overline{b},\overline{c},\overline{d}\right] \iff [0] \in \mathcal{D}_r\left[\overline{b},\overline{c},\overline{d}\right]$.*

7 Algorithm

In this section, we define the final dynamic programming table that computes $\mathcal{C}_x\left[\overline{b},\overline{c},\overline{d}\right]$ for $x \in V(\mathcal{T})$ and all values $\overline{b},\overline{c}$ and $\overline{d}$ in time $O^*(3^k)$.

▶ **Definition 49.** *For all $x \in V(\mathcal{T})$ and all values $\overline{b} \in [k]_0, \overline{c} \in [|V|\cdot W]_0$ and $\overline{d} \in [|V(\mathcal{T})|\cdot D]_0$, we define the vectors $T = T\left[x,\overline{b},\overline{c},\overline{d}\right] \in \{0,1\}^{\mathcal{P}_{CS}}$ as follows:*

- *Create node $i(v)$. Let $p = [0,i]$ if $v \neq v_0$, and $p = [0i]$ otherwise. First we initialize $T[x,\overline{b},\overline{c},\overline{d}] = \overline{0}$ for all values $\overline{b},\overline{c},\overline{d}$. Now we set $T[x,0,0,0][[0]] = [v \notin T]$, and for $\ell \in [2]$ we set $T[x,1,\mathbf{w}(v),\mathbf{d}(x,\ell)][\mathrm{Ac}(p,\ell)] = 1$.*
- *Relabel node $\mu_x = \rho_{i\to j}(\mu_{x'})$, we define $T\left[x,\overline{b},\overline{c},\overline{d}\right][p] = \sum_{\substack{q\in\mathcal{P}_{CS}\\ q_{i\to j}=p}} T\left[x',\overline{b},\overline{c},\overline{d}\right][q]$.*
- *Join node $\mu_x = \eta_{i,j}(\mu_{x'})$. For all values $\overline{b},\overline{c},\overline{d}$, we define $T'\left[x,\overline{b},\overline{c},\overline{d}\right] \in \{0,1\}^{\mathcal{P}}$ as*

$$T'\left[x,\overline{b},\overline{c},\overline{d}\right][p] = \sum_{\substack{q\in\mathcal{P}_{CS},\\ \square_{ij}q=p}} T\left[x',\overline{b},\overline{c},\overline{d}\right][q],$$

for all patterns $p \in \mathcal{P}$. Now for all $p \in \mathcal{P}_{CS}$ we set

$$T\left[x,\overline{b},\overline{c},\overline{d}\right][p] = \sum_{\ell\in[4]} \sum_{\substack{q\in\mathcal{P},\\ p\in\mathrm{Ac}^*(q,\ell)}} T'\left[x,\overline{b},\overline{c},\overline{d}-\mathbf{d}(x,\ell)\right][q].$$

- *Union node* $\mu_x = \mu_{x_1} \uplus \mu_{x_2}$. *For all* $p \in \mathcal{P}_{CS}$, *we define*

$$T\big[x, \overline{b}, \overline{c}, \overline{d}\big][p] = \sum_{\substack{b_1+b_2=\overline{b} \\ c_1+c_2=\overline{c} \\ d_1+d_2=\overline{d}}} \sum_{\substack{p_1,p_2 \in \mathcal{P}_{CS} \\ p_1 \oplus p_2 = p}} T[x_1, b_1, c_1, d_1][p_1] \cdot T[x_2, b_2, c_2, d_2][p_2].$$

The following lemmas show the correctness of the algorithm (proof in the full version).

▶ **Lemma 50.** *It holds that* $T\big[x, \overline{b}, \overline{c}, \overline{d}\big][p] = \big[p \in \mathcal{C}_x\big[\overline{b}, \overline{c}, \overline{d}\big]\big]$ *for all* $x, \overline{b}, \overline{c}, \overline{d}$, *and* $p \in \mathcal{P}_{CS}$.

▶ **Corollary 51.** *It holds for all* $\overline{b}, \overline{c}, \overline{d}$ *that* $T\left[r, \overline{b}, \overline{c}, \overline{d}\right][[0]] \equiv_2 1 \iff [0] \in \mathcal{D}_r\big[\overline{b}, \overline{c}, \overline{d}\big]$.

Although computing the tables $T\big[x, \overline{b}, \overline{c}, \overline{d}\big]$ for a union node x in the naive way yields a running time polynomial in $|\mathcal{P}_{CS}|^2$ which exceeds the bound we seek by far, we make use of a result by Hegerfeld and Kratsch [27] to compute convolutions over lattices more efficiently.

▶ **Definition 52.** *Given a lattice* $(\mathcal{L}, \preceq)$, *and two tables* $A, B : \mathcal{L} \to \mathbb{F}$ *for some field* $\mathbb{F}$, *we define the* join-product *(or* $\vee$*-product)* $A \otimes_{\mathcal{L}} B$ *as follows: for each* $x \in \mathcal{L}$ *we define*

$$A \otimes_{\mathcal{L}} B(x) = \sum_{\substack{y,z \in \mathcal{L} \\ y \vee z = x}} A(y) \cdot B(z).$$

To avoid unnecessary notation, we state a modified, more restrictive version of the mentioned result by Hegerfeld and Kratsch [27, Corollary A.10], and refer the reader to the full version for an exact statement and background definitions.

▶ **Corollary 53.** *Let* $(\mathcal{L}, \preceq)$ *be a finite lattice such that the join operation over* $\mathcal{L}$ *can be computed in polynomial time. Let* k *be a natural number. Given two tables* $A, B \colon \mathcal{L}^k \to \mathbb{Z}_2$, *the* $\vee$*-product* $A \otimes_{\mathcal{L}^k} B$ *can be computed in time* $\mathcal{O}^*(|\mathcal{L}|^k)$.

▶ **Definition 54.** *We define a new ordering* $\preceq_{CS}$ *over* $\mathcal{P}_{CS}$ *where* $p_1 \preceq_{CS} p_2$, *if for each* $i \in U$ *it holds that* $i \in Z_{p_1}$ *if* $i \in Z_{p_2}$ *and* $i \in \mathrm{label}(p_1)$ *if* $i \in \mathrm{label}(p_2)$.

▶ **Observation 55.** *The join operation over the lattice* $(\mathcal{P}_{CS}, \preceq_{CS})$ *is given by* $p \vee q = r$, *where* $Z_r = Z_p \cup Z_q$ *and* $\mathrm{label}(r) = \mathrm{label}(p) \cup \mathrm{label}(q)$. *This corresponds exactly to the union operation over CS-patterns* $r = p \oplus q$. *Hence, it holds for a union node that*

$$T\big[x, \overline{b}, \overline{c}, \overline{d}\big] = \sum_{\substack{b_1+b_2=\overline{b} \\ c_1+c_2=\overline{c} \\ d_1+d_2=\overline{d}}} T[x', b_1, c_1, d_1] \otimes_{\mathcal{P}_{CS}} T[x'', b_2, c_2, d_2].$$

In the full version, we show that the lattice $(\mathcal{P}_{CS}(U), \preceq_{CS})$ is isomorphic to the lattice $\mathcal{L}^k$, where $\mathcal{L} = (\{1, 2, 3\}, \leq)$ is defined in the natural way. Using Corollary 53, it follows that $T\big[x, \overline{b}, \overline{c}, \overline{d}\big]$ for a union node x can be computed in time $O^*(3^k)$. The following lemma states that this is indeed the case for all nodes x of $\mathcal{T}$. We refer the reader to the full version for formal proofs.

▶ **Lemma 56.** *The families* $T\big[x, \overline{b}, \overline{c}, \overline{d}\big]$ *for all* $x \in V(\mathcal{T})$ *and all values* $\overline{b}, \overline{c}, \overline{d}$ *can be computed in time* $\mathcal{O}^*(3^k)$.

Now we are ready to prove the main theorem of this work (Theorem 1).

Proof (Theorem 1). The algorithm first computes all families $T\left[x, b, \overline{c}, \overline{d}\right]$ for all $x \in V(\mathcal{T})$ and all values of $b, \overline{c}, \overline{d}$. The algorithm states that there exists a Steiner tree of size $\overline{b}$, if and only if there exist two values $\overline{c}, \overline{d}$ with $T\left[r, \overline{b}, \overline{c}, \overline{d}\right][[0]] = 1$.

By Lemma 56, we can compute all families $T\left[x, b, \overline{c}, \overline{d}\right]$ in time $\mathcal{O}^*(3^k)$. By Lemma 50, it holds for all values $\overline{b}, \overline{c}, \overline{d}$ that

$$T\left[r, \overline{b}, \overline{c}, \overline{d}\right][[0]] \equiv_2 1 \iff [0] \in \mathcal{D}_r\left[\overline{b}, \overline{c}, \overline{d}\right].$$

By Corollary 32 it holds that if no Steiner tree of size $\overline{b}$ exists, then the algorithm will always return false. On the other hand if a Steiner tree of size $\overline{b}$ exists, by choosing $W = (2+\sqrt{2})|V|$, and $D = 4(2+\sqrt{2})|V(\mathcal{T})|$, and by choosing both function $\mathbf{w} \in [W]^V$, and $\mathbf{d} \in [D]^{V(\mathcal{T})\times[4]}$ uniformly and independently at random, it holds by Lemma 37 that the algorithm will correctly decide the existence of such a tree with probability at least one half. ◀

8 Conclusion

We have presented a $3^k \cdot n^{\mathcal{O}(1)}$ time one-sided Monte-Carlo algorithm for Steiner Tree[cw]. Under the Strong Exponential-Time Hypothesis, lower bounds for Steiner Tree relative to less expressive parameters [15, 4] rule out time $(3-\varepsilon)^k \cdot n^{\mathcal{O}(1)}$, and show that this parameter dependence is optimal for all parameters between cutwidth and clique-width.

Rather than using cut-and-count, which is the go-to technique for connectivity problems, our algorithm works with connectivity patterns. Two technical contributions, make this work: The family of complete connectivity patterns has strong existential representation properties, so that each connectivity pattern has an explicit existential replacement by complete patterns. This avoids spending more time for getting the representative partial solutions (such as for Gaussian elimination in the rank-based approach [3]). The second technical contribution, the technique of *isolating a representative*, allows to combine this with working via a 3^k-sized family of basis patterns, that represents complete patterns modulo two. We expect that both techniques will be of use for settling the complexity of other (connectivity) problems.

Very recently, the present authors announced a tight bound of $12^k \cdot n^{\mathcal{O}(1)}$ (modulo SETH) for Connected Odd Cycle Transversal[cw] [6].

References

1. Benjamin Bergougnoux and Mamadou Moustapha Kanté. Fast exact algorithms for some connectivity problems parameterized by clique-width. *Theor. Comput. Sci.*, 782:30–53, 2019. doi:10.1016/j.tcs.2019.02.030.
2. Benjamin Bergougnoux and Mamadou Moustapha Kanté. More applications of the d-neighbor equivalence: Acyclicity and connectivity constraints. *SIAM J. Discret. Math.*, 35(3):1881–1926, 2021. doi:10.1137/20M1350571.
3. Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.
4. Narek Bojikian, Vera Chekan, Falko Hegerfeld, and Stefan Kratsch. Tight bounds for connectivity problems parameterized by cutwidth. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 14:1–14:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.STACS.2023.14.
5. Narek Bojikian and Stefan Kratsch. A tight monte-carlo algorithm for steiner tree parameterized by clique-width. *CoRR*, abs/2307.14264, 2023. doi:10.48550/arXiv.2307.14264.
6. Narek Bojikian and Stefan Kratsch. Tight algorithm for connected odd cycle transversal parameterized by clique-width, 2024. arXiv:2402.08046.

7 Glencora Borradaile and Hung Le. Optimal dynamic program for r-domination problems over tree decompositions. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.8.

8 Derek G. Corneil, Michel Habib, Jean-Marc Lanlignel, Bruce A. Reed, and Udi Rotics. Polynomial-time recognition of clique-width ≤3 graphs. *Discret. Appl. Math.*, 160(6):834–865, 2012. doi:10.1016/j.dam.2011.03.020.

9 Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *J. Comput. Syst. Sci.*, 46(2):218–270, 1993. doi:10.1016/0022-0000(93)90004-G.

10 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

11 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discret. Appl. Math.*, 101(1-3):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

12 Radu Curticapean, Nathan Lindzey, and Jesper Nederlof. A tight lower bound for counting hamiltonian cycles via matrix rank. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1080–1099. SIAM, 2018. doi:10.1137/1.9781611975031.70.

13 Radu Curticapean and Dániel Marx. Tight conditional lower bounds for counting perfect matchings on graphs of bounded treewidth, cliquewidth, and genus. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1650–1669. SIAM, 2016. doi:10.1137/1.9781611974331.ch113.

14 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3):12:1–12:46, 2018. doi:10.1145/3148227.

15 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.23.

16 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *CoRR*, abs/1103.0534, 2011. arXiv:1103.0534.

17 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

18 Baris Can Esmer, Jacob Focke, Dániel Marx, and Pawel Rzazewski. List homomorphisms by deleting edges and vertices: tight complexity bounds for bounded-treewidth graphs. *CoRR*, abs/2210.10677, 2022. doi:10.48550/arXiv.2210.10677.

19 Wolfgang Espelage, Frank Gurski, and Egon Wanke. How to solve np-hard graph problems on clique-width bounded graphs in polynomial time. In Andreas Brandstädt and Van Bang Le, editors, *Graph-Theoretic Concepts in Computer Science, 27th International Workshop, WG 2001, Boltenhagen, Germany, June 14-16, 2001, Proceedings*, volume 2204 of *Lecture Notes in Computer Science*, pages 117–128. Springer, 2001. doi:10.1007/3-540-45477-2_12.

20 Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is np-complete. *SIAM J. Discret. Math.*, 23(2):909–939, 2009. doi:10.1137/070687256.

21 Jacob Focke, Dániel Marx, Fionn Mc Inerney, Daniel Neuen, Govind S. Sankar, Philipp Schepper, and Philip Wellnitz. Tight complexity bounds for counting generalized dominating sets in bounded-treewidth graphs. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3664–3683. SIAM, 2023. doi:10.1137/1.9781611977554.ch140.

22 Jacob Focke, Dániel Marx, and Pawel Rzazewski. Counting list homomorphisms from graphs of bounded treewidth: tight complexity bounds. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 431–458. SIAM, 2022. doi:10.1137/1.9781611977073.22.

23 Robert Ganian, Thekla Hamm, Viktoriia Korchemna, Karolina Okrasa, and Kirill Simonov. The fine-grained complexity of graph homomorphism parameterized by clique-width. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 66:1–66:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.66.

24 Carla Groenland, Isja Mannens, Jesper Nederlof, and Krisztina Szilágyi. Tight bounds for counting colorings and connected edge sets parameterized by cutwidth. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPIcs*, pages 36:1–36:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.36.

25 Frank Gurski and Egon Wanke. Vertex disjoint paths on clique-width bounded graphs. *Theor. Comput. Sci.*, 359(1-3):188–199, 2006. doi:10.1016/j.tcs.2006.02.026.

26 Tesshu Hanaka, Yasuaki Kobayashi, and Taiga Sone. A (probably) optimal algorithm for bisection on bounded-treewidth graphs. *Theor. Comput. Sci.*, 873:38–46, 2021. doi:10.1016/j.tcs.2021.04.023.

27 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by clique-width. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 59:1–59:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.59.

28 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by modular-treewidth. In Daniël Paulusma and Bernard Ries, editors, *Graph-Theoretic Concepts in Computer Science - 49th International Workshop, WG 2023, Fribourg, Switzerland, June 28-30, 2023, Revised Selected Papers*, volume 14093 of *Lecture Notes in Computer Science*, pages 388–402. Springer, 2023. doi:10.1007/978-3-031-43380-1_28.

29 Yoichi Iwata and Yuichi Yoshida. On the equivalence among problems of bounded width. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 754–765. Springer, 2015. doi:10.1007/978-3-662-48350-3_63.

30 Bart M. P. Jansen and Jesper Nederlof. Computing the chromatic number using graph decompositions via matrix rank. *Theor. Comput. Sci.*, 795:520–539, 2019. doi:10.1016/j.tcs.2019.08.006.

31 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discret. Appl. Math.*, 264:90–117, 2019. doi:10.1016/j.dam.2018.11.002.

32 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structurally parameterized d-scattered set. *Discret. Appl. Math.*, 308:168–186, 2022. doi:10.1016/j.dam.2020.03.052.

33 Michael Lampis. Model checking lower bounds for simple graphs. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 673–683. Springer, 2013. doi:10.1007/978-3-642-39206-1_57.

34 Michael Lampis. Finer tight bounds for coloring on clique-width. *SIAM J. Discret. Math.*, 34(3):1538–1558, 2020. doi:10.1137/19M1280326.

35 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 777–789. SIAM, 2011. doi:10.1137/1.9781611973082.61.

36 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

37 Dániel Marx, Govind S. Sankar, and Philipp Schepper. Degrees and gaps: Tight complexity results of general factor problems parameterized by treewidth and cutwidth. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 95:1–95:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.95.

38 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Comb.*, 7(1):105–113, 1987. doi:10.1007/BF02579206.

39 Jesper Nederlof. Algorithms for np-hard problems via rank-related parameters of matrices. In Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen, editors, *Treewidth, Kernels, and Algorithms - Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *Lecture Notes in Computer Science*, pages 145–164. Springer, 2020. doi:10.1007/978-3-030-42071-0_11.

40 Karolina Okrasa, Marta Piecyk, and Pawel Rzazewski. Full complexity classification of the list homomorphism problem for bounded-treewidth graphs. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 74:1–74:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.74.

41 Karolina Okrasa and Pawel Rzazewski. Fine-grained complexity of the graph homomorphism problem for bounded-treewidth graphs. *SIAM J. Comput.*, 50(2):487–508, 2021. doi:10.1137/20M1320146.

42 Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Trans. Algorithms*, 5(1):10:1–10:20, 2008. doi:10.1145/1435375.1435385.

43 Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

44 Marta Piecyk and Pawel Rzazewski. Fine-grained complexity of the list homomorphism problem: Feedback vertex set and cutwidth. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 56:1–56:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.56.

45 Bas A. M. van Geffen, Bart M. P. Jansen, Arnoud A. W. M. de Kroon, and Rolf Morel. Lower bounds for dynamic programming on planar graphs of bounded cutwidth. *J. Graph Algorithms Appl.*, 24(3):461–482, 2020. doi:10.7155/jgaa.00542.

46 Egon Wanke. k-nlc graphs and polynomial algorithms. *Discret. Appl. Math.*, 54(2-3):251–266, 1994. doi:10.1016/0166-218X(94)90026-4.

Õptimal Dynamic Time Warping on Run-Length Encoded Strings

Itai Boneh
Reichman University, Herzliya, Israel
University of Haifa, Israel

Shay Golan
Reichman University, Herzliya, Israel
University of Haifa, Israel

Shay Mozes
Reichman University, Herzliya, Israel

Oren Weimann
University of Haifa, Israel

Abstract

Dynamic Time Warping (DTW) distance is the optimal cost of matching two strings when extending runs of letters is for free. Therefore, it is natural to measure the time complexity of DTW in terms of the number of runs n (rather than the string lengths N).

In this paper, we give an $\tilde{O}(n^2)$ time algorithm for computing the DTW distance. This matches (up to log factors) the known (conditional) lower bound, and should be compared with the previous fastest $O(n^3)$ time exact algorithm and the $\tilde{O}(n^2)$ time approximation algorithm. Our method also immediately implies an $\tilde{O}(nk)$ time algorithm when the distance is bounded by k. This should be compared with the previous fastest $O(n^2k)$ and $O(Nk)$ time exact algorithms and the $\tilde{O}(nk)$ time approximation algorithm.

2012 ACM Subject Classification Theory of computation → Pattern matching; Theory of computation → Shortest paths

Keywords and phrases Dynamic time warping, Fréchet distance, edit distance, run-length encoding

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.30

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2302.06252` [9]

Funding Israel Science Foundation grant 810/21

1 Introduction

Dynamic Time Warping (DTW) [39] is one of the most popular methods for comparing time-series (see e.g. [2, 5, 8, 25, 27, 30, 33, 40, 43]). It is appealing in numerous applications such as bioinformatics, signature verification, and speech recognition, where two time-series can vary in speed but still be considered similar. For example, in speech recognition, DTW can detect similarities even if one person is talking faster than the other.

To define DTW, recall that a run-length encoding $S = s_1^{\ell_1} s_2^{\ell_2} \cdots s_n^{\ell_n}$ of a string S over an alphabet Σ is a concise (length n) representation of the (length $N = \sum_i \ell_i$) string S. Here $s_i^{\ell_i}$ denotes a letter $s_i \in \Sigma$ repeated ℓ_i times. For example, the string $S = aaaabbbaaaaa$ is encoded as $a^4b^3a^5$. A string $S' = s_1^{\ell'_1} s_2^{\ell'_2} \cdots s_n^{\ell'_n}$ is a *time-warp* of string $S = s_1^{\ell_1} s_2^{\ell_2} \cdots s_n^{\ell_n}$ if every $\ell'_i \geq \ell_i$.

▶ **Definition 1** (Dynamic Time Warping). *For a function $\delta : \Sigma^2 \to \mathbb{R}^+$, the Dynamic Time Warping distance of two strings S and T over alphabet Σ is defined as*

$$\mathsf{DTW}(S,T) = \min_{|S'|=|T'|} \sum_{i=1}^{|S'|} \delta(S'[i], T'[i]),$$

where S' and T' range over all time-warps of S and T respectively.

In 1968, Vintzyuk [39] gave an $O(MN)$ time dynamic programming algorithm for computing the DTW of two strings S and T of lengths N and M respectively. His algorithm is one of the earliest uses of dynamic programming and is taught today in basic algorithms courses and textbooks. Apart from logarithmic factor improvements [17], the $O(MN)$ quadratic time complexity remains the fastest known and a strongly subquadratic-time $O((MN)^{1-\varepsilon})$ algorithm is unlikely as it would refute the popular Strong Exponential Time Hypothesis (SETH) [3, 10].

The complexity of DTW in terms of N and M is thus well understood. Special cases of DTW are also well understood. These include DTW on binary strings [23, 38], approximation algorithms [4, 22, 42], the low distance regime [22], sparse inputs [20, 31, 32], and reductions to other similarity measures [22, 36, 37]. However, the complexity of DTW is not yet resolved in terms of n and m (the run-length encoding sizes of S and T respectively). Namely, in the (especially appealing) case where the strings contain long runs. The currently fastest algorithms are $O(Nm+Mn)$ [13,15,22] and $O(n^2m+m^2n)$ [15]. In particular, an $\tilde{O}(nm)$ time algorithm is only known to be possible if we are willing to settle for a $(1+\varepsilon)$-approximation [41]. It remained an open question whether it is possible to obtain an exact $\tilde{O}(nm)$ algorithm (which is optimal up to log factors). In this paper we answer this open question in the affirmative.

Prior work on DTW. The classical dynamic programming for DTW is as follows. Let $\mathsf{DTW}(i,j) = \mathsf{DTW}(S[1\dots i], T[1\dots j])$, then $\mathsf{DTW}(0,0) = 0$, $\mathsf{DTW}(i,0) = \mathsf{DTW}(0,j) = \infty$ for every $i > 0$ and $j > 0$, and otherwise:

$$\mathsf{DTW}(i,j) = \delta(S[i], T[j]) + \min \begin{cases} \mathsf{DTW}(i-1,j) \\ \mathsf{DTW}(i,j-1) \\ \mathsf{DTW}(i-1,j-1) \end{cases} \tag{1}$$

The above dynamic programming is equivalent to a single-source shortest path (SSSP) computation in the following grid graph. We denote $[n] = \{1, 2, \dots, n\}$.

▶ **Definition 2** (The Alignment Graph). *The alignment graph of S and T is a directed weighted graph G with vertices $V = [0\dots N] \times [0\dots M]$. Every vertex $(i,j) \in [N] \times [M]$ has three entering edges, all with weight $\delta(S[i], T[j])$: A vertical edge from $(i-1,j)$, a horizontal edge from $(i,j-1)$, and a diagonal edge from $(i-1,j-1)$.*

We denote the distance from vertex $(0,0)$ to (i,j) as $\mathsf{dist}(i,j)$.[1] Clearly, $\mathsf{DTW}(i,j) = \mathsf{dist}(i,j)$. Therefore, $\mathsf{DTW}(S,T) = \mathsf{dist}(N,M)$ and can be computed in $O(MN)$ time by an SSSP algorithm (that explicitly computes the distances from $(0,0)$ to all the $O(MN)$ vertices of the graph). The way to beat $O(MN)$ is to only compute distances to a subset of vertices.

[1] Abusing notation, we will later also use $\mathsf{dist}((x,y),(x',y'))$ to denote the distance from vertex (x,y) to vertex (x',y').

Namely, partition the alignment graph into *blocks* where each block is the subgraph corresponding to a single run in S and a single run in T. Then, proceed block-by-block and for each block compute its *output* (the last row and last column) given its *input* (the last row of the block above and the last column of the block to the left). Since blocks are highly regular (i.e., all edges inside a block have the same weight), it is not difficult to compute the output in time linear in the size of the output. Since the total size of all outputs (and all inputs) is $O(Nm + Mn)$, this leads to an overall $O(Nm + Mn)$ time algorithm [13, 15, 22].

In order to go below $O(Nm + Mn)$, in [15] it was observed that we do not really need to compute the entire output. It suffices to compute only the intersection of the output with a set of $O(mn)$ diagonals. Specifically, each block contributes one diagonal starting in its top-left corner, so there are overall $O(mn)$ diagonals and each diagonal intersects with $O(m + n)$ blocks. This leads to an $O(n^2m + m^2n)$ time algorithm. In [41] it was shown that if we are willing to settle for a $(1 + \varepsilon)$-approximation, then it suffices to compute only $\tilde{O}(1)$ output values per block.

Prior work on edit distance. There are many similarities between DTW and the edit distance problem: (1) like DTW, edit distance can be computed in $O(MN)$ time using the alignment graph [35, 39]. The only difference is in the edge-weights. (2) like DTW, edit distance has a lower bound prohibiting strongly subquadratic time algorithms conditioned on the SETH [3, 10, 22], and (3) like DTW, edit distance can be computed in $O(Nm + Mn)$ time by proceeding block-by-block and computing the outputs from the inputs. However, unlike DTW, it is known how to compute the edit distance of run-length encoded strings in $\tilde{O}(nm)$ time [6, 7, 11–13, 19, 26, 28, 29]. Specifically, Clifford et. al. [13] showed that the input and output of a block can be implicitly represented by a piecewise linear function, and, that the representation of the output can be computed in amortized $O(\mathsf{polylog}(mn))$ time from the representation of the input. This implies an $\tilde{O}(nm)$ time algorithm for edit distance.

In [41], Xi and Kuszmaul write about the prospects of obtaining an $\tilde{O}(nm)$ time algorithm for DTW: "*Such an algorithm would finally unify edit distance and DTW in the run-length-encoded setting*".

Our result and techniques. We present an $\tilde{O}(nm)$ time algorithm for DTW. This is optimal up to logarithmic factors under the SETH. Our algorithm is independent of the alphabet size $|\Sigma|$ and of the function δ. In fact, δ need not even satisfy the triangle inequality.

We follow the approach for edit distance by Clifford et. al. [13] of representing and manipulating inputs and outputs with a piecewise-linear function. However, the manipulation is more challenging for several reasons which were highlighted by Xi and Kuszmaul [41]: (1) unlike edit distance, DTW does not satisfy the triangle inequality. (2) we are interested in arbitrary cost functions δ for DTW, whereas the $\tilde{O}(nm)$ algorithm for edit distance [13] works only for Levenshtein distance (when $\delta(\cdot, \cdot) \in \{0, 1\}$). (3) in the standard setting (i.e. not the run-length encoded setting) edit distance actually reduces to DTW [22].

In Section 2, we show that the required manipulation of inputs and outputs naturally reduces to $O(nm)$ operations on a data structure that, given an array A of size $M + N$ initialized to all zeros, supports the following range operations:

▶ **Definition 3** (Range Operations).

- $\mathsf{Lookup}(i)$ - *return* $A[i]$.
- $\mathsf{AddConst}(i, j, c)$ - *for every* $k \in [i \dots j]$, *set* $A[k] \leftarrow A[k] + c$.
- $\mathsf{AddGradient}(i, j, g)$ - *for every* $k \in [i \dots j]$, *set* $A[k] \leftarrow A[k] + k \cdot g$.
- $\mathsf{LeftLinearWave}(i, j, \alpha)$ - *for every* $k \in [i \dots j]$, *set* $A[k] \leftarrow \min_{t \in [i \dots k]} \big(A[t] + (k - t)\alpha\big)$.
- $\mathsf{RightLinearWave}(i, j, \alpha)$ - *for every* $k \in [i \dots j]$, *set* $A[k] \leftarrow \min_{t \in [k \dots j]} \big(A[t] + (t - k)\alpha\big)$.

In Section 3, we show our main technical contribution:

▶ **Theorem 4.** *Performing s range operations of Definition 3 can be done in amortized $O(\mathsf{polylog}(s))$ time per operation.*

The proof of Theorem 4 can be roughly described as follows: We represent the array A by the line segments of the linear interpolation of A. This way, the range operations of Definition 3 translate to creating and deleting segments, changing their slopes, and shiftings segments up and down. For most operations, these changes apply to a single contiguous range of A and are therefore quite simple to implement in polylog time. The difficult operations are LeftLinearWave and RightLinearWave. These operations may need to replace each of $\Omega(n)$ different sets of consecutive segments with a single new segment. We refer to the process of replacing a set of consecutive segments with a single new segment as a *ray shooting* process. Shooting each of these rays separately would be too costly. More accurately, a ray shooting process that replaces many segments with a single one is not problematic since its cost can be charged to the decrease in the number of segments. The challenge is in shooting rays that replace a single segment with another one, as this does not decrease the number of segments.

Our main technical contribution is a lazy approach for handling the problematic ray shooting processes. We study the structural properties of ray shooting processes, and characterize long rays which we can afford to shoot explicitly, and short rays, which we cannot. The structure we identify allows us to divide the segments representing A into mega-segments, and keep track of a single pending short ray in each mega-segment such that executing the pending ray shooting process in each mega-segment would result in the correct representation of the array A. While we cannot afford to actually carry out all of these pending ray shooting processes, we can afford to perform the process locally, e.g., in order to support Lookup for a specific element of A, or to facilitate the other range operations.

One component of our lazy approach is a data structure (sometimes called *Segment tree beats* in programming competitions) for the following problem: Maintain an array A under lookup queries and two kinds of update: $\mathsf{AddConst}(i, j, c)$ - for every $k \in [i \dots j]$ set $A[k] \leftarrow A[k] + c$, and $\mathsf{Min}(i, j, c)$ - for every $k \in [i \dots j]$ set $A[k] \leftarrow \min\{A[k], c\}$. Though we are not aware of any official publication, it is known (see e.g. [1]) that this problem can be solved in amortized polylog time. We show a different and *worst-case* polylog time solution.[2]

Implications for low regime DTW. In Section 4, we show that our $\tilde{O}(n \cdot m)$ algorithm for computing $\mathsf{DTW}(S, T)$ immediately implies an $\tilde{O}(n \cdot k)$ time algorithm where $k = \mathsf{DTW}(S, T)$. This is useful when k is small. It is achieved using the standard trick of limiting the computation to blocks that are in the k-neighborhood of the alignment graph's main diagonal. It improves the $O(N \cdot k)$ algorithm of [22], the $\tilde{O}(n^2 \cdot k)$ algorithm of [15], and the $\tilde{O}(n \cdot k)$ time approximation algorithm of [41] (all obtained with the same k-neighborhood idea).

We note that for the closely related problem of low regime *edit distance*, using the same k-neighborhood idea, the algorithm of [13] runs in $\tilde{O}(n \cdot k)$ time (now k is the edit distance between S and T). However, unlike DTW, there is a vast literature on low regime edit distance (and the approximation algorithms inspired by it). Most notable are the celebrated $O(N + k^2)$ time algorithms of Myers [34] and Landau-Vishkin [24] for unweighted edit distance, and the very recent $O(N + k^5)$ time algorithm for weighted edit distance [14].

[2] We note that the solution in [1] also supports range-sum queries and for such a conditional lower bound (from the Online Matrix-Vector Multiplication (OMV) problem) is known [16]. The lower bound implies that *worst-case* operations unlikely to be possible in $O(n^{1/2-\varepsilon})$ time. We are able to circumvent this lower bound because we only support lookups, but not range-sum queries.

Implications for pattern matching DTW. The pattern matching version of DTW asks to compute, for every index $j \in [1 \dots |T|]$ the value $\min_{i \in [1 \dots j]}(\mathsf{DTW}(S, T[i \dots j]))$. In [18], an $O(NM)$ algorithm was presented for pattern matching DTW. Additionally, they provided an $O(nmk)$ algorithm for the low regime version of the problem, in which the goal is to report every index j such that $\min_{i \in [1 \dots j]}(\mathsf{DTW}(S, T[i \dots j])) \leq k$. The key ingredient of these algorithms (See [18, Lemma 2]) is a dynamic programming formula that is identical to Equation (1), except for the initialization. Since our $\tilde{O}(nm)$ algorithm for DTW is obtained by implementing the dynamic programming implicitly, by changing the initialization step, our algorithm implies an $\tilde{O}(nm)$ time algorithm for pattern matching DTW. This improves upon both the $O(NM)$ algorithm for DTW pattern matching and the $O(nmk)$ algorithm for the low regime DTW pattern matching (when k is super poly-logarithmic).

2 DTW via Range Operations

In this section we prove that Theorem 4 implies an $\tilde{O}(nm)$ algorithm for DTW. Namely, that DTW reduces to efficiently supporting the range operations of Definition 3.

Blocks in the alignment graph. Let $S[i_1 \dots i_2]$ and $T[j_1 \dots j_2]$ be the i'th run in S and the j'th run in T respectively. The block $B_{i,j}$ in the alignment graph is the set of vertices (a, b) with $a \in [i_1 \dots i_2]$ and $b \in [j_1 \dots j_2]$. All of the edges entering any vertex in block $B_{i,j}$ have the same weight $\delta(S[i_1], T[j_1])$, which we denote by $c_{B_{i,j}}$. We call the blocks $B_{i-1,j}$, $B_{i,j-1}$ and $B_{i-1,j-1}$ the *entering blocks* of $B_{i,j}$. The *input* of a block consists of all vertices belonging to the first row or first column of the block. The *output* of a block consists of all vertices belonging to the last row or last column of the block. The following structural lemma was also used implicitly in previous works (see formal proof in the full version).

▶ **Lemma 5.** *Let B be a block.*

- *If $(x, y), (x, y+1) \in B$ then there is a shortest path from $(0, 0)$ to $(x+1, y+1)$ that does not visit $(x, y+1)$.*
- *If $(x, y), (x+1, y) \in B$ then there is a shortest path from $(0, 0)$ to $(x+1, y+1)$ that does not visit $(x+1, y)$.*
- *If $(x, y), (x+1, y+1) \in B$ then there is a shortest path from $(0, 0)$ to $(x+1, y+1)$ that goes through (x, y).*

Frontiers in the alignment graph. Our algorithm for DTW processes all blocks in the alignment graph. At every step, the algorithm can process any block B as long as all its entering blocks have already been processed. When block B is processed, the algorithm computes $\mathsf{dist}(x, y)$ for every output vertex (x, y) of B. After processing block B, we say that the output vertices of B are *resolved*. At every step of the algorithm, the *frontier* is the set of resolved vertices with an outgoing edge to a block that was not yet processed. Observe that, at any given time in the execution of the algorithm, for every value $d \in [-N \dots M]$, the frontier includes exactly one vertex (x, y) such that $y - x = d$. At every step t of the algorithm, we will maintain an array $\mathsf{F}_t[-N \dots M]$ where $F_t[d] = \mathsf{dist}(x, y)$ such that vertex (x, y) belongs to the current frontier and $y - x = d$. In the full version of this paper ([9]), we prove the following lemma.

▶ **Lemma 6.** *F_{t+1} can be obtained by using $O(1)$ range operations (Definition 3) on F_t.*

In the rest of this section, we prove that Lemma 6 implies our main result:

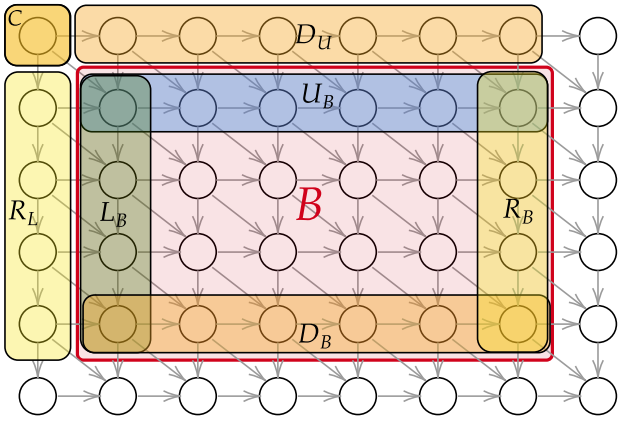


Figure 1 A block B, its inputs $L_B \cup U_B$ and its outputs $D_B \cup R_B$. The entering edges to L_B are from R_L, the corner of its diagonally adjacent block C, and the leftmost node of D_U. The entering edges to U_B are from D_U, the corner of C, and the topmost node of R_L.

▶ **Theorem 7.** *The Dynamic Time Warping distance of two run-length encoded strings S and T with n and m runs respectively can be computed in $\tilde{O}(nm)$ time.*

Proof. We initialize the data structure of Theorem 4 as an array of length $N + M + 1$. We treat the indices of A as if they are in $[-N \dots M]$[3]. Initially, the frontier consists of the vertices $(x, 0)$ with $x \in [0 \dots N]$ and $(0, y)$ with $y \in [M]$. We start by turning A into F_0. According to Equation (1), we need to set $A[0] = 0$ and $A[i] = \infty$ for $i \neq 0$. This can be done by applying $\mathsf{AddConst}(1, M, \infty)$ and $\mathsf{AddConst}(-N, -1, \infty)$.

The algorithm runs in nm iterations. At the beginning of iteration t, we have $A = \mathsf{F}_t$. The algorithm picks any block B whose entering blocks have already been processed, and applies $O(1)$ range operations (due to Lemma 6) on A in order to obtain $A = \mathsf{F}_{t+1}$. After the last iteration, it is guaranteed that the block $B_{n,m}$ has been processed. Therefore, $\mathsf{F}_{nm}[M - N] = \mathsf{DTW}(S, T)$. Every iteration requires $O(1)$ range operations each in $O(\mathsf{polylog}(nm))$ time, so overall the algorithm performs $O(nm)$ operations in total $\tilde{O}(nm)$ time. ◀

3 Implementing the Range Operations

In this section, we prove Theorem 4. We view the array A as a piecewise linear function. We associate with A a set $\mathcal{P} = \{p_1 = (x_1, y_1), p_2 = (x_2, y_2), \dots\}$ of points satisfying $A[x_i] = y_i$. The set $\mathcal{P}$ is uniquely defined by A as the endpoints of the maximal linear segments of the linear interpolation of A. Note that the first point of $\mathcal{P}$ is always $(1, A[1])$ and the last point is $(n, A[n])$.[4] Let $\ell_i(x) = \alpha_i x + \beta_i$ be the line segment between p_i and p_{i+1}. Our representation will maintain the α_i's and β_i's. With this representation we can retrieve $A[x]$ for any $x \in [1, n]$ from α_i and β_i where x_i is predecessor of x in the sequence $(x_1, x_2, \dots)$. Upon initialization, A is represented as one linear segment, with $\alpha_1 = 0$, and $\beta_1 = 0$.

We will use the following simple data structure.[5]

[3] When a gradient update $\mathsf{AddGradient}(i, j, c)$ affects a value $A[k]$, we would like $A[k]$ to be increased by $k \cdot c$ with $k \in [-N \dots M]$ being the 'simulated' index rather then the actual index $k + N + 1$. This can be achieved by applying an additional operation $\mathsf{AddConst}(i, j, (-N - 1) \cdot c)$.

[4] Here we use n to denote the size of the array A.

[5] The data structure can be implemented using a balanced search tree with a delta-representation (where

▶ **Lemma 8** (Interval-add Data Structure). *There is a data structure supporting the following operations in $O(\log n)$ time per operation on a set of n points with distinct first coordinates.*

- Lookup(x) - *return the second coordinate of the point with first coordinate x, if exists.*
- Insert(x, y) - *insert the point (x, y).*
- Remove(x) - *remove the point with first coordinate x, if exists.*
- AddToRange(i, j, c) - *for every point (x,y) with $x \in [i \dots j]$ set $y \leftarrow y + c$.*
- nextGT(x, y) - *return the point $p' = (x', y')$ with smallest $x' > x$ among points with $y' > y$.*
- prevLT(x, y) - *return the point $p' = (x', y')$ with largest $x' < x$ among points with $y' < y$.*

3.1 A Warmup Algorithm

We first present a naive and inefficient implementation of a range operations data structure. We maintain the sequence $\mathcal{P}$ in a predecessor/successor data structure over the sequence $(x_1, x_2, \dots)$. With a slight abuse of notation we shall also use $\mathcal{P}$ to refer to this data structure. We maintain the α_i's and β_i's using two Interval-add data structures D_α and D_β, respectively. The parameters α_i, β_i of the linear segment ℓ_i starting at x_i are represented by points (x_i, α_i) in D_α and (x_i, β_i) in D_β. In what follows, whenever we say we add a point $p = (x, y)$ to $\mathcal{P}$ we mean that (x, y) is inserted into the predecessor/successor data structure $\mathcal{P}$, and that points with first coordinate x are inserted into D_α and D_β, with their second coordinates appropriately set to reflect the parameters α, β of the segment ending at p and the segment starting at p. This process requires $O(1)$ operations on $\mathcal{P}, D_\alpha$ and D_β.

The effect of AddConst(i, j, c) (see Figure 2) is to break the segment containing i into at most three linear segments (a prefix ending at $i-1$, a segment $[i-1, i]$, and a suffix starting at i), and similarly for the segment containing j. Thus, to apply AddConst(i, j, c), we first replace the segments containing i and j with these $O(1)$ new segments by inserting or updating the endpoints of the segments in $\mathcal{P}, D_\alpha$, and D_β. We then invoke AddToRange(i, j, c) on D_β to shift all segments between i and j by c. Next, we set the parameters for the segment $[i-1, i]$ and for the segment $[j, j+1]$ by $O(1)$ additional calls to AddToRange on D_α and D_β. Finally, we check if any of the new segments we inserted has the same slope as its adjacent segments and, if so, we merge them into a single segment by removing their common point from $\mathcal{P}, D_\alpha$ and D_β. This guarantees that the set $\mathcal{P}$ we maintain is indeed the set $\mathcal{P}$ defined by A. Supporting AddGradient(i, j, g) is similar. The only difference is that we invoke AddToRange(i, j, g) on D_α instead of on D_β because the slopes of the segments are shifted rather than their values.

The challenge is thus in supporting LeftLinearWave(i, j, α). We first describe its effect and then describe how it is implemented. We assume without loss of generality that i and j are both endpoints of segments (otherwise we break the segments containing them into $O(1)$ segments as above). Let $p_a = (i, A[i])$ and $p_{b+1} = (j, A[j])$ be the points corresponding to i and j. Thus, the segments contained within $[i \dots j]$ are $\ell_a, \ell_{a+1} \dots \ell_b$.

If $\alpha_a \le \alpha$ then the segment ℓ_a is not affected by the linear wave. This is because for every $k \in [x_a \dots x_{a+1}]$, the linear wave assigns

$$\begin{aligned} A[k] &\leftarrow \min_{x_a \le t \le k} (A[t] + (k-t)\alpha) = \min_{x_a \le t \le k} (A[k] - (k-t)\alpha_a + (k-t)\alpha) \\ &= \min_{x_a \le t \le k} (A[k] + (k-t)(\alpha - \alpha_a)) = A[k]. \end{aligned}$$

the value of a node is represented by the sum of values of its ancestors), and having every node also store the minimal and maximal values in its subtree. See e.g. [21].

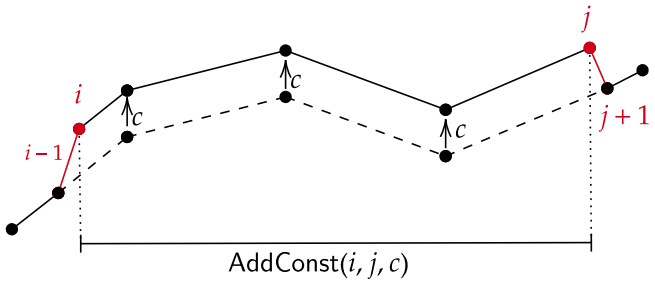


Figure 2 An illustration of applying the AddConst(i, j, c) operation. The dashed line represents the segments before the operation. After the operation, new points are created with x coordinates $i-1, i, j$ and $j+1$ and the segments in $[i \dots j]$ are shifted by c.

Let $z \in [a \dots b]$ be the minimum index such that $\alpha_z > \alpha$. By the same reasoning, none of the segments $\ell_a, \ell_{a+1}, \dots, \ell_{z-1}$ are affected by the linear wave. Let $r_z(x)$ be the (positive) ray with slope[6] α starting at p_z. Since $\alpha_z > \alpha$, the ray r_z is below the linear segment ℓ_z. Hence, the segment ℓ_z starting at p_z is affected by the linear wave; its slope changes from α_z to α, and it extends beyond x_{z+1} as long as $A[x] \geq r_z(x)$. We describe this effect of LeftLinearWave by a *ray shooting* process from p_z (See Figure 3). This process identifies the new endpoint p' of ℓ_z, and removes all the existing segments between p_z and p', as follows.

Let $z' \in [z+1..b+1]$ be the minimum index with $y_{z'} < r_z(x_{z'})$, i.e. the first point in $\mathcal{P}$ that lies strictly below the ray r_z. Let $p^* = (x^*, y^*)$ be the intersection point of the ray r_z with $\ell_{z'-1}$ (if z' does not exist, then $p^* = p_b$). The new endpoint of ℓ_z is the point $p' = (x', y') = (\lfloor x^* \rfloor, r_z(\lfloor x^* \rfloor))$, and it replaces all the points p_w for $w \in (z \dots z')$. If x^* is not an integer (or if z' does not exist) then a new segment is formed between p' and $p'' = (x'+1, A[x'+1])$.

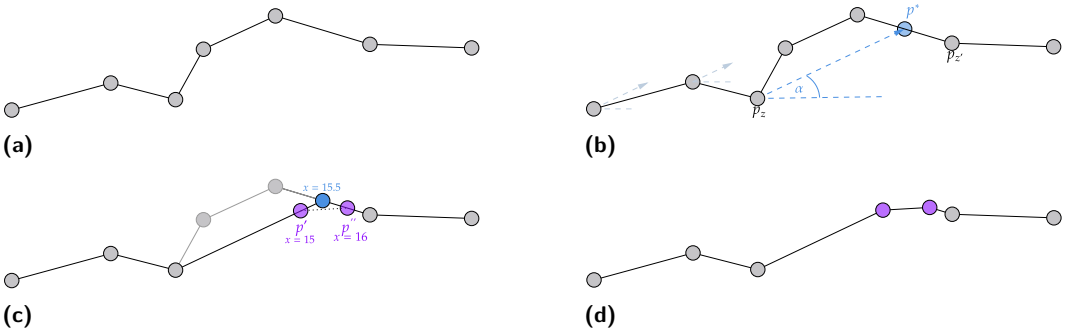


Figure 3 The effect of LeftLinearWave(i, j, α). The segments before p_z are not affected. The segments between p_z and $p_{z'}$ are affected. Namely, a ray r_z with slope α (in dashed blue) is shot from p_z and intersects at point $p^* = (x^*, y^*)$. The new endpoint of ℓ_z becomes p' and all the segments between p_z and p' are removed. Since $x^* = 15.5$ is not an integer, a new segment is formed between p' (with x coordinate 15) and p'' (with x coordinate 16).

The effect of LeftLinearWave(i, j, α) on the remaining part of A, namely on $A[x_{z'} \dots j]$ is analyzed in the same way as above, this time starting from $p_{z'}$ instead of from p_a. In particular, the prefix of segments with slopes less than α is not affected, and a ray with slope

[6] Note that in Figure 3 and in all subsequent figures we indicate the slope α of the ray r_z by drawing an angle α between the ray and the positive direction of the x-axis. However, formally α is the slope of the ray, not the indicated angle.

α is shot from the next p_w with $\alpha_w > \alpha$, and so on. In the the full version of this paper ([9]) we formally prove that the above characterization indeed represents the new values of $A[i \dots j]$.

We now describe a naive, non-efficient implementation of LeftLinearWave(i, j, α) according to the description above. Recall that i and j are assumed to be endpoints p_a and p_{b+1} of segments. We begin by finding the first p_z with $x_z \in [i \dots j]$ and $\alpha_z > \alpha$ by querying D_α.nextGT(i, α). A ray shooting process is then performed from p_z (if p_z exists) as follows: Recall that $r_z(x)$ denotes the positive ray with slope α shot from p_z. We scan the successor points of p_z one by one in order, and for every point p_w we check whether the ray $r_z(x_w) \leq y_w$. If so, p_w is removed by removing x_w from $\mathcal{P}$, D_β and D_α. Otherwise, we compute $p^* = (x^*, y^*)$, the intersection point of r_z and ℓ_{w-1}, and from it the points $p' = (x', y') = (\lfloor x^* \rfloor, r_z(\lfloor x^* \rfloor)$ and, if x^* is not an integer, also $p'' = (\lceil x^* \rceil, \ell_{w-1}(\lceil x^* \rceil))$. Then, we insert the new points p' and p'' just before p_w, as discussed above for AddConst. The scanning then continues with another nextGT query from p_w, and so on. If, at the end of the process, the last point p_b is removed since it is above some r_z, we insert a new point $(x_b, r_z(x_b))$.

Time Complexity. We now analyze the time complexity of this naive implementation. Each AddConst and AddGradient operation requires $O(1)$ operations on the Interval-add data structures, and therefore takes $O(\text{polylog}|\mathcal{P}|)$ time per operation, with $|\mathcal{P}|$ being the cardinality of $\mathcal{P}$ when the operation is applied.

Regarding LeftLinearWave operations, one might hope that the cost of each ray shooting process can be charged to the removal of points from $\mathcal{P}$ during the process. However, each ray shooting process might also add up to two new points, which might result in the size of $\mathcal{P}$ increasing. Indeed, a LeftLinearWave operation may give rise to many such ray shooting processes, and hence may significantly increase the size of $\mathcal{P}$ and take too much time. This is the main technical challenge we need to address.

The idea is to distinguish between *long* ray shootings for which we can globally charge the new insertions, and *short* ray shootings for which we cannot. We handle the long rays as in the naive solution and devise a separate lazy mechanism that delays the application of all the short rays stemming from a single LeftLinearWave operation using a constant number of updates to a separate data structure that keeps track of the delayed rays.

Symmetry of RightLinearWave. The discussion so far was focused on the LeftLinearWave operation. We note that the analysis of RightLinearWave is symmetric. In particular, the execution of RightLinearWave(i, j, α) can be described as a sequence of ray shootings with *negative* rays. The first point from which a ray is shot is p_z with largest $z \in [a \dots b]$ such that $\alpha_{z-1} < -\alpha$ (p_z is found using D_α.prevLT). Note that the condition for starting a ray shooting process for RightLinearWave is on α_{z-1} rather than α_z since the slope of the segment to the left of p_z is α_{z-1}. To simplify the presentation, we will keep describing only LeftLinearWave, and will comment at the very end about the minor adjustments required to also handle the symmetric RightLinearWave.

3.2 Active and Passive Points, Long and Short Rays

On our way to formally define long rays and short rays we first observe that ray shootings only occur at points where slopes increase. We call such points *active* points.

▶ **Definition 9** (Active and Passive points). *A point p_z in $\mathcal{P}$ is called* active *if $z \in \{1, |\mathcal{P}|\}$ or $\alpha_z > \alpha_{z-1}$. A point that is not active, is called* passive. *We denote the sets of active points by $\mathcal{P}_{\text{active}}$.*

▶ **Lemma 10.** *Ray shootings stemming from* LeftLinearWave(i, j, α) *occur either at point* $p_a = (i, A[i])$ *or at active points. Ray shootings stemming from* RightLinearWave(i, j, α) *occur either at point* $p_b = (j, A[j])$ *or at active points.*

Proof. We focus on LeftLinearWave. The proof for RightLinearWave is symmetric. Assume to the contrary that a ray shooting process starts at a passive point $p_z \neq p_a$. If p_z is the first point where a ray shooting starts, then z is the minimal index in $[a \dots b]$ with $\alpha_z > \alpha$. But since p_z is passive, we have $\alpha < \alpha_z \leq \alpha_{z-1}$, contradicting the minimality of z (note that $p_z \neq p_a$ so $z - 1 \in [a \dots b]$).

Otherwise, let p_q be the last point before p_z from which a ray shooting process occurred. Let $p_{q'}$ be the first point below the ray shot from p_q. Since p_z is the next point from which a ray is shot, z is the first point in $[q' \dots b]$ with $\alpha_z \geq \alpha$. Since p_z is passive, we have $\alpha < \alpha_z \leq \alpha_{z-1}$. If $z \neq q'$, we have $z - 1 \in [q' \dots b]$, a contradiction to the minimality of z. Otherwise, $p_z = p_{q'}$ is the first point below the ray with slope α shot from p_q. It follows that p_{z-1} is above the ray, and $\alpha_{z-1} > \alpha$. It must be the case that $p_{q'}$ is above the ray, a contradiction. ◀

Let $\mathcal{P}_{\mathsf{active}} = (q_1, q_2, \dots)$ be the restriction of the sequence $\mathcal{P}$ to the active points. We can think of the active points as defining a piecewise linear function whose segments are a coarsening of the segments of A. We refer to these segments as *mega-segments*. Let γ_z denote the slope of the mega-segment whose endpoints are q_z and q_{z+1}. The following lemma asserts that the segments of A are never below their corresponding mega-segments, and that the slope of a segment starting at an active point is never smaller than the slope of the mega-segment starting at the same point.

▶ **Lemma 11.** *Let* $q_z = p_w$ *and* $q_{z+1} = p_{w'}$ *be two consecutive active points. For every* $k \in [w \dots w']$*, the passive point* p_k *is not below the mega-segment connecting* q_z *and* q_{z+1}*. Furthermore,* $\alpha_w \geq \gamma_z$*.*

Proof. (See Figure 4) Clearly, p_w and $p_{w'}$ are on the mega-segment, and in particular not below it. Assume by contradiction that there is a point below the mega-segment, and let $k' \in (w \dots w')$ be the smallest index of such a point. Since $p_{k'-1}$ is not below the mega-segment and $p_{k'}$ is below the mega-segment, we must have $\alpha_{k'-1} < \gamma_z$. Moreover, since the points p_k with $k \in [k' \dots w')$ are passive, the slopes are non-increasing and therefore every $\alpha_k \leq \gamma_z$. This means that all these points and in particular $p_{w'}$ are below the mega-segment. In contradiction to $p_{w'}$ lying on the mega-segment. Furthermore, since p_{w+1} is not below the mega-segment, we have $\alpha_w \geq \gamma_z$. ◀

We next show that if a ray shooting process starts at an active point q_z with $\gamma_z < \alpha$ then the process ends before q_{z+1}, and the only affected points are the passive points between q_z and q_{z+1}. On the other hand, if $\gamma_z \geq \alpha$ then as a result of the process q_{z+1} ceases to be an active point, so $|\mathcal{P}_{\mathsf{active}}|$ decreases.

▶ **Lemma 12.** *Consider a ray shooting process starting from point* $p_w = q_z \in \mathcal{P}_{\mathsf{active}}$ *during the application of* LeftLinearWave(i, j, α)*. Let* $q_{z+1} = p_{w'}$*.*

1. *No new active points* $p = (x, y)$ *with* $x \neq j$ *are created in this process.*
2. *If* $\gamma_z < \alpha$ *then the points that are deleted by this process are the (passive) points* p_k *with* $k \in [w + 1 \dots r]$ *for some* $w < r < w'$*. No other points between* p_w *and* $p_{w'}$ *are deleted by* LeftLinearWave(i, j, α)*.*
3. *If* $\gamma_z \geq \alpha$ *then* q_{z+1} *is either deleted or becomes passive.*

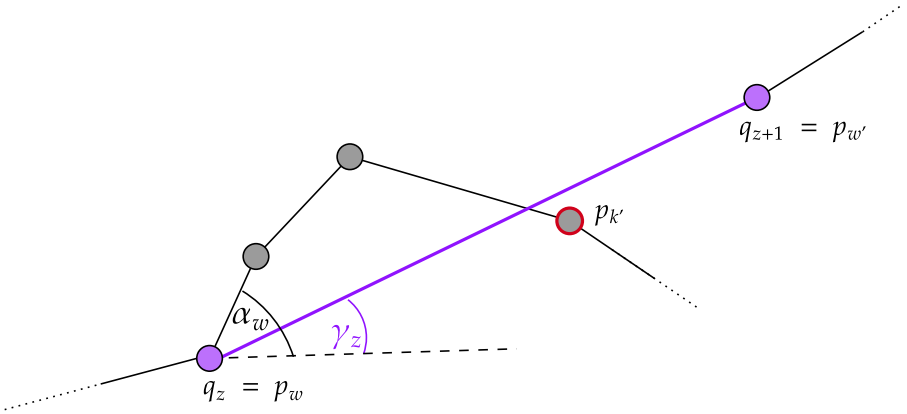


Figure 4 The impossible configuration in Lemma 11. The points q_z and q_{z+1} are represented by the purple points and the mega-segment connecting them is represented by a thick purple line. The first point $p_{k'}$ below the segment is marked with red stroke. Since the points strictly within the mega-segment are passive, the points following $p_{k'}$ within the mega-segment (and in particular q_{z+1}) must remain below the mega-segment.

Proof. Let ℓ be the ray starting from $p_w = q_z$. Assume the process terminates by finding the first point $p^* = (x^*, y^*)$ below ℓ (the only process that does not end this way is the one that ends by reaching $(j, A[j])$). The ray shooting process adds at most two new points p' and p'' with decreasing slopes, so no new active points are created by the process. The slope of p' is decreasing because the segment entering p' is (a sub-segment of) ℓ and the segment leaving p' is to a point below ℓ. The slope of p'' is decreasing because the line segment entering p'' is a line from p' (a point on ℓ) and the line segment leaving p'' is to the suffix of a line segment below ℓ.

Consider the case $\gamma_z < \alpha$. Then q_{z+1} is below the ray with slope α starting at q_z. Hence the ray shooting process terminates at a point after p_{w+1} and before q_{z+1}. Since no active points are created, the next ray will be shot from q_{z+1} or later, so no other points between q_z and q_{z+1} are deleted by $\mathsf{LeftLinearWave}(i, j, \alpha)$.

Now consider the case $\gamma_z > \alpha$. Then the mega-segment between q_z and q_{z+1} is above the ray with slope α shot from q_z. By Lemma 11, all the (passive) points between q_z and q_{z+1} are also above this ray. Hence q_{z+1} is deleted by the ray shooting process.

Finally, consider the case $\gamma_z = \alpha$. Then the mega-segment between q_z and q_{z+1} coincides with the ray with slope α shot from q_z. By Lemma 11, all the (passive) points between q_z and q_{z+1} will be deleted by the ray shooting process. Let w' be such that $q_{z+1} = p_{w'}$. If $\alpha_{w'} \geq \alpha$ then q_{z+1} will be deleted by the process. Otherwise, $\alpha_{w'} < \alpha$, so the ray shooting process terminates at q_{z+1}. Since all the passive points between q_z and q_{z+1} were deleted, q_z and q_{z+1} become consecutive in $\mathcal{P}$, and the slope of the corresponding segment is $\gamma_z = \alpha$. But the slope of the segment starting at q_{z+1} is $\alpha_{w'} < \alpha$, so q_{z+1} becomes passive. ◀

We call rays with $\gamma_z > \alpha$ *long* rays, and those with $\gamma_z \leq \alpha$ *short* rays. Since long rays decrease the size of $\mathcal{P}_{\mathsf{active}}$ we can handle them explicitly as in the warmup, charging the deletion of passive points during the process to their creation, and charging the insertion of the at most two passive points at the end of the process to the decrease in $|\mathcal{P}_{\mathsf{active}}|$. The short rays, which do not decrease $|\mathcal{P}_{\mathsf{active}}|$, will be handled lazily. Namely, instead of explicitly shooting a short ray in the mega-segment starting at an active point q_z, we only store the slope of the ray and postpone its execution until it is required (e.g., by a Lookup operation). Note that subsequent short rays shot in this mega-segment may further change the stored slope, and subsequent long rays may also affect it. We explain this in detail next.

3.3 The Data Structure

In this section, we provide a technical overview of the construction of the range operation data structure of Definition 3. The complete implementation details and proofs appear in the full version of this paper ([9]). Since our data structure is lazy, the sequence of points it maintains will be different than the sequence $\mathcal{P}$ that would have been maintained had we used the warmup algorithm from Section 3.1. We will therefore use $\tilde{\mathcal{P}}$ to denote the set of points actually maintained by the data structure. The points $\tilde{\mathcal{P}}$ define linear segments $\tilde{\ell}_i(x)$ in the usual way. For $x \in [1, n]$ we denote by $\tilde{A}[x]$ the value $\tilde{\ell}_i(x)$, where $\tilde{\ell}_i$ is the segment containing x. We stress that our algorithm does not maintain $\mathcal{P}$. However, for the sake of description and analysis only we shall keep referring to the original $\mathcal{P}$, and array A. The definition of active and passive points, of the slopes γ of mega-segments, and of short and long rays are now with respect to the slopes of the $\tilde{\ell}_i$'s.[7] However, we shall maintain that the set of active points with respect to $\mathcal{P}$ and $\mathcal{P}_{\mathsf{active}}$ is the same:

▶ **Invariant 1.** $\tilde{\mathcal{P}}_{\mathsf{active}} = \mathcal{P}_{\mathsf{active}}$.

Following Section 3.1, we maintain $\tilde{\mathcal{P}}$ in a predecessor/successor data structure, as well as the Interval-add data structures D_α and D_β representing the parameters of the linear segments $\tilde{\ell}_i(x)$ defined by the points of $\tilde{\mathcal{P}}$. By implementing AddConst, AddGradient and long ray shootings similarly to Section 3.1 (the exact details will be spelled out below), we shall maintain the invariant that this part of the data structure correctly represents the values of active points.[8]

We maintain the set of active points $\tilde{\mathcal{P}}_{\mathsf{active}} = (q_1, q_2, \ldots)$ using a predecessor/successor structure on their x-coordinates. For each $q_z \in \tilde{\mathcal{P}}_{\mathsf{active}}$, we maintain the slope γ_z of the mega-segment starting at q_z in an Interval-add data structure D_γ. In addition, we maintain a pending short ray r_z with slope ρ_z passing through q_z (see Figure 5) by maintaining ρ_z in a data structure D_ρ. This data structure, which we call the Add-min data structure is summarized below and proved in the full version of this paper.

▶ **Lemma 13** (Add-min Data Structure). *There exists a data structure supporting the following operations in $O(\mathsf{polylog} n)$ time on a set of points S.*

1. Insert(x, y) - *insert the point (x, y) to S.*
2. Remove(x) - *remove the point $p = (x, y)$ from S, if such a point exists.*
3. Lookup(x) - *return y such that $p = (x, y)$ is in S, or report that there is no such point.*
4. AddToRange(i, j, c) - *for every $p = (x, y) \in S$ with $x \in [i \ldots j]$ set $y \leftarrow y + c$.*
5. Min(i, j, c) - *for every $p = (x, y) \in S$ with $x \in [i \ldots j]$ set $y \leftarrow \min(y, c)$.*

Note that storing ρ_z suffices to compute $r_z(x)$ since the active point q_z that determines the free coefficient of r_z is correctly represented by D_α and D_β. We shall show that storing a single pending ray suffices to represent all the pending changes in a mega-segment. This property will rely on maintaining the following invariant.

▶ **Invariant 2.** *For every active point q_z we have $\rho_z > \gamma_z$. (Recall that γ_z is the slope of the mega-segment connecting q_z and q_{z+1}.)*

---

[7] It would have been more accurate to use $\tilde{\alpha}, \tilde{\beta}$, and $\tilde{\gamma}$, but this would be too cumbersome, so we stick to using α, β, γ.

[8] See Invariant 3 and the note following it.

The idea is that with this representation, for any x, the value of $A[x]$ is given by the minimum of the value $\tilde{A}[x] = \ell_w(x)$ of the segment of $\tilde{\mathcal{P}}$ containing x, and the value $r_z(x)$ of the pending short ray for the mega-segment containing x. This is captured by the following main invariant maintained by the data structure.

▶ **Invariant 3.** *Let $x \in [1, n]$, and let p_w and q_z be the predecessor of x in $\tilde{\mathcal{P}}$ and in $\tilde{\mathcal{P}}_{\mathsf{active}}$, respectively. It holds that $A[x] = \min(\tilde{\ell}_w(x), r_z(x))$. Furthermore, if $p = (x, A[x])$ is an active point in $\mathcal{P}$, then $A[x] = \tilde{A}[x]$.*

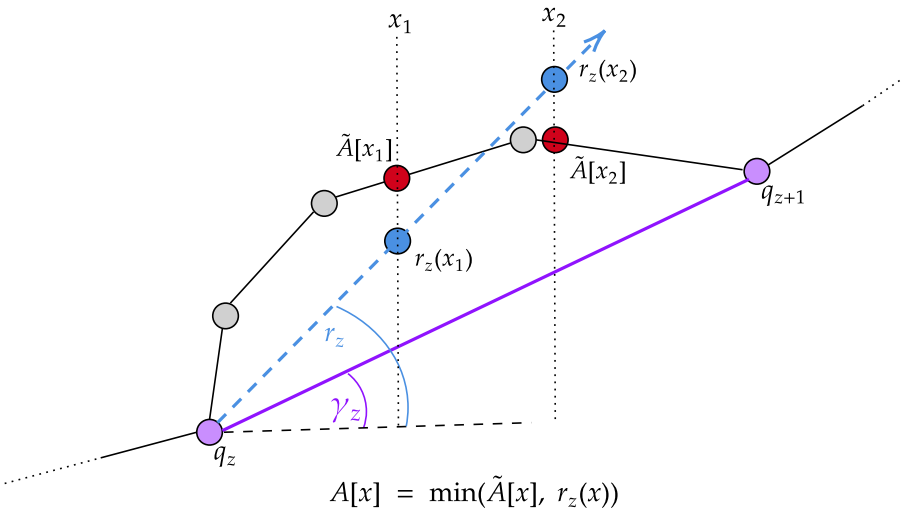


Figure 5 An illustration of the data stored for a mega-segment between two consecutive active points q_z and q_{z+1} (purple points). The slope γ_z is the slope of the mega-segment. The slope $\rho_z > \gamma_z$ stored in q_z represents a pending ray r_z (dashed blue) that should be shot from q_z. The value of $A[x]$ is the minimum between $r_z(x)$ (a blue point) and $\tilde{A}[x]$ (a red point), the value of the piece-wise linear function defined by $\tilde{\mathcal{P}}$ (in grey).

Note that the first part of Invariant 3, together with Invariant 1 implies the second part of Invariant 3. This is because the predecessor of x for an active point $p = (x, A[x])$ in $\mathcal{P}_{\mathsf{active}}$ is itself. Since $\mathcal{P}_{\mathsf{active}} = \tilde{\mathcal{P}}_{\mathsf{active}}$ we have that $p \in \tilde{\mathcal{P}}_{\mathsf{active}}$ is the predecessor of x in $\tilde{\mathcal{P}}_{\mathsf{active}}$ as well. By definition r_z goes through $p = q_z$, so $r_z(x) = \tilde{A}[x]$, and $\tilde{A}[x] = \tilde{\ell}_w(x)$ by definition. Hence, when proving that the invariants are maintained, we will not need to explicitly establish the second statement in Invariant 3.

Initially, $\tilde{\mathcal{P}} = \mathcal{P} = \{(1, 0), (|A|, 0)\}$, and $\rho_1 = \rho_2 = \infty$ (We interpret a line with a slope of ∞ as $y = \infty$). Indeed, $A[x] = \min(\tilde{A}[x], r_1(x)) = \min(0, \infty) = 0$ and Invariant 3 is satisfied. It remains to specify the implementation of the various operations supported by the data structure, to prove that the invariants are maintained, and to analyze the running times.

The flush Operation. We first describe a service operation $\mathsf{flush}(q_z)$ which explicitly shoots the pending short ray in the mega-segment starting at the active point q_z. It will be useful to invoke flush before serving Lookup operations, but also when serving the other operations in order to guarantee that the lazy implementation properly follows the explicit implementation in the warmup. This is particularly important in operations which may create $O(1)$ new active points and thus change the partition into mega-segments, but is also useful to streamline the proof of correctness. Recall that the reason we avoided shooting local rays in the first place was that there could be many of them, and we could not afford to pay for the possible

creation of $O(1)$ new passive points at the end of each of them. We can afford, however, to perform $O(1)$ flush operations before each Lookup, AddConst or AddGradient operation, because the cost of adding the $O(1)$ new points can be charged to the operation itself.

A flush of $q_z = p_w \in \mathcal{P}_{\text{active}}$ is performed as follows. Starting from p_{w+1}, we scan the points in $\tilde{\mathcal{P}}$. When scanning $p = (x, y)$, we compare y and r_z. If $r_z(x) \leq y$, we remove p from $\tilde{\mathcal{P}}$. Otherwise, the scan halts. Let p_{end} be the point on which the scan halts. If no point was deleted throughout the scan, we set $\rho_z = \infty$ and terminate. Otherwise, let p_{del} be the last point deleted by the scan. We compute the intersection $p^* = (x^*, y^*)$ of r_z and the line $\tilde{\ell}$ between p_{del} and p_{end}. Finally, we insert $p' = (\lfloor x^* \rfloor, r_z(\lfloor x^* \rfloor))$ and $p'' = (\lceil x^* \rceil, \tilde{\ell}(\lceil x^* \rceil))$ to $\tilde{\mathcal{P}}$ (as in the warmup algorithm of Section 3.1), update D_α and D_β with the new parameters of the segments ending and starting at p' or at p'', and set $\rho_z = \infty$.

▶ **Lemma 14.** *Applying* flush *to an active point* $q_z \in \mathcal{P}_{\text{active}}$ *preserves Invariants 1–3. Furthermore, it guarantees that the restriction of* $\mathcal{P}$ *and* $\tilde{\mathcal{P}}$ *to the (passive) points between* q_z *and* q_{z+1} *is identical, and that for every* $x \in [x_z \dots x_{z+1}]$, $A[x] = \tilde{A}[x]$.

Proof. Invariant 2 is maintained because the flush operation sets ρ_z to ∞. Since $\rho_z > \gamma_z$, it is guaranteed by Lemma 12 that the scan of flush ends at q_{z+1} or before q_{z+1}. It follows that Invariant 1 is maintained because $\mathcal{P}_{\text{active}}$ does not change and flush only deletes passive points of $\tilde{\mathcal{P}}$. We proceed to prove that Invariant 3 is maintained. Note that ρ_z is set to ∞ by the end of flush, and that q_z remains the predecessor active point of every $x \in [x_z \dots x_{z+1}]$, so we need to show $A[x] = \tilde{A}[x]$. Let $x \in [x_z \dots x_{z+1}]$. If $x \leq x^*$, then before flush was applied, we had $\tilde{A}[x] \geq r_z(x)$, and therefore by Invariant 3 $A[x] = \min(\tilde{A}[x], r_z(x)) = r_z(x)$. Since flush sets the value of $\tilde{A}[x]$ to be $r_z(x)$ for $x < x^*$, Invariant 3 still holds. If $x > x^*$, the value of $\tilde{A}[x]$ is not changed by flush. Since the line $\tilde{\ell}$ between p_{del} and p_{end} starts not below the r_z and ends below r_z, its slope is smaller than ρ_z. Since the points between p_{end} and q_z (excluding q_z) are passive, the slopes of the corresponding segments are also lower than ρ_z and therefore $(x, \tilde{A}[x])$ is below r_z for every $x \in (x' \dots x_{z+1}]$. Due to Invariant 3 before the application of flush, we have $A[x] = \min(\tilde{A}[x], r_z(x)) = \tilde{A}[x]$. Therefore, assigning $\rho_z \leftarrow \infty$ and not changing $\tilde{A}[x]$ satisfies Invariant 3. ◀

4 Bounded DTW

In this section, we study the k-bounded version of DTW. In this version, every $\delta(a, b) \geq 1$, and we wish to compute $\mathsf{DTW}_k(S, T) = \min(\mathsf{DTW}(S, T), k + 1)$ for a given integer k. In this section, we prove the following theorem:

▶ **Theorem 15.** *The Dynamic Time Warping distance of two run-length encoded strings* S *and* T *with* n *and* m *runs respectively can be computed in* $\tilde{O}(nk)$ *time if* $\mathsf{DTW}(S, T) \leq k$.

The key structural insight for Theorem 15 is that there is a set of $O(nk)$ blocks containing all the vertices (x, y) with $\mathsf{dist}(x, y) \leq k$. Therefore, it is sufficient to process only those blocks instead of the entire grid. Informally, the set of $O(nk)$ blocks is a band of width $\Theta(k)$ around the main diagonal of blocks. This property holds since a path to a vertex outside the band requires $\Omega(k)$ orthogonal steps between blocks. Note that since every $\delta(a, b) \geq 1$, at least one of any two orthogonally adjacent blocks is a non-zero block, and the part of the path that goes through this block must incur a cost of at least 1. Formally:

▷ Claim 16. Let (x, y) be a vertex in the alignment graph. Let $B_{i,j}$ be the block containing (x, y). If $|j - i| > 2k$, then $\mathsf{dist}(x, y) > k$.

Proof. We assume without loss of generality that $j - i > 2k$. Let p be a path from $(0,0)$ to (x,y). Let $P = B_{i_1,j_1}, B_{i_2,j_2} \dots B_{i_{|P|},j_{|P|}}$ be the sequence of blocks visited by p (where $B_{i_1,j_1} = B_{0,0}$ and $B_{i_{|P|},j_{|P|}} = B_{i,j}$). Note that for every $a \in [1 \dots |P| - 1]$ we have $(i_{a+1}, j_{a+1}) \in \{(i_a + 1, j_a), (i_a, j_a + 1), (i_a + 1, j_a + 1)\}$. Since $j_{|P|} - i_{|P|} > 2k$, and $i_1 - j_1 = 0$, there must be at least $2k + 1$ values of $a \in [1 \dots |P| - 1]$ such that $(i_{a+1}, j_{a+1}) = (i_a, j_a + 1)$.

Consider a value of a with this property. Since the j_a'th run and the $(j_a + 1)$'th run in T are adjacent, they must consist of different symbols. It follows that either $c_{B_{i_a,j_a}} \geq 1$ or $c_{B_{i_a,j_a+1}} \geq 1$. Let $B' \in \{B_{i_a,j_a}, B_{i_a,j_a+1}\}$ be the block with non-zero weight. The fragment of p that goes through B' incurs a weight of at least 1. Note that every block may be associated with at most 2 different values of a - once when p enters the block and once when it leaves the block. Therefore, $2k + 1$ different values of a indicate that the cost of p is at least $k + 1$. ◁

We denote the set of blocks $B_{i,j}$ such that $|j - i| \leq 2k$ as the *band*. It follows directly from Claim 16 that if $\mathsf{dist}(x,y) \leq k$ for some vertex (x,y) then there is a shortest path from $(0,0)$ to (x,y) that uses only the vertices of the band. Therefore, we can set the weight of every block outside of the band to ∞. Then, instead of processing all the blocks, we only process the blocks of the band. Any block not in the band is considered vacuously processed. Before processing a block with an input that is not in the band, we initialize the values in the frontier corresponding to this input to ∞. This is implemented by an $\mathsf{AddConst}(i, j, \infty)$ for the appropriate interval $[i, j]$. Note that with this assignment, the inputs have the same values as if the algorithm would have processed all the blocks. Finally, the algorithm reports $\mathsf{DTW}_k(S,T) = \min(\mathsf{dist}(N,M), k+1)$.

References

1 Segment tree beats. `https://codeforces.com/blog/entry/57319`.
2 John Aach and George M. Church. Aligning gene expression time series with time warping algorithms. *Bioinformatics*, 17(6):495–508, 2001.
3 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *56th FOCS*, pages 59–78, 2015.
4 Pankaj K. Agarwal, Kyle Fox, Jiangwei Pan, and Rex Ying. Approximating dynamic time warping and edit distance for a pair of point sequences. In *32nd SoCG*, pages 14–18, 2016.
5 Saeed Reza Aghabozorgi, Ali Seyed Shirkhorshidi, and Ying Wah Teh. Time-series clustering - A decade review. *Inf. Syst.*, 53:16–38, 2015.
6 Alberto Apostolico, Gad M. Landau, and Steven Skiena. Matching for run-length encoded strings. *Journal of Complexity*, 15(1):4–16, 1999.
7 Ora Arbell, Gad M. Landau, and Joseph S. B. Mitchell. Edit distance of run-length encoded strings. *Information Processing Letters*, 83(6):307–314, 2002.
8 Anthony J. Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn J. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Min. Knowl. Discov.*, 31(3):606–660, 2017.
9 Itai Boneh, Shay Golan, Shay Mozes, and Oren Weimann. Near-optimal dynamic time warping on run-length encoded strings, 2023. `arXiv:2302.06252`.
10 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *56th FOCS*, pages 79–97. IEEE, 2015.
11 Horst Bunke and János Csirik. Edit distance of run-length coded strings. In *1992 ACM/SIGAPP Symposium on Applied computing: Technological challenges of the 1990's*, pages 137–143, 1992.
12 Kuan-Yu Chen and Kun-Mao Chao. A fully compressed algorithm for computing the edit distance of run-length encoded strings. *Algorithmica*, 65(2):354–370, 2013.

13 Raphaël Clifford, Pawel Gawrychowski, Tomasz Kociumaka, Daniel P. Martin, and Przemyslaw Uznanski. RLE edit distance in near optimal time. In *44th MFCS*, pages 66:1–66:13, 2019.

14 Debarati Das, Jacob Gilbert, MohammadTaghi Hajiaghayi, Tomasz Kociumaka, and Barna Saha. Weighted edit distance computation: Strings, trees, and dyck. In *55th STOC*, pages 377–390, 2023.

15 Vincent Froese, Brijnesh J. Jain, Maciej Rymar, and Mathias Weller. Fast exact dynamic time warping on run-length encoded time series. *Algorithmica*, 85(2):492–508, 2022.

16 Pawel Gawrychowski and Yanir Edri. private communication, 2016.

17 Omer Gold and Micha Sharir. Dynamic time warping and geometric edit distance: Breaking the quadratic barrier. In *44th ICALP*, volume 80, pages 25:1–25:14, 2017.

18 Garance Gourdel, Anne Driemel, Pierre Peterlongo, and Tatiana Starikovskaya. Pattern matching under DTW distance. In *29th SPIRE*, pages 315–330, 2022.

19 Guan-Shieng Huang, Jia Jie Liu, and Yue-Li Wang. Sequence alignment algorithms for run-length-encoded strings. In *14th COCOON*, volume 5092, pages 319–330, 2008.

20 Youngha Hwang and Saul B. Gelfand. Fast sparse dynamic time warping. In *26th ICPR*, pages 3872–3877, 2022.

21 Philip N. Klein and Shay Mozes. Optimization algorithms for planar graphs. `http://planarity.org`. Book draft.

22 William Kuszmaul. Dynamic time warping in strongly subquadratic time: Algorithms for the low-distance regime and approximate evaluation. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th ICALP*, volume 132, pages 80:1–80:15, 2019.

23 William Kuszmaul. Binary dynamic time warping in linear time, 2021. arXiv preprint.

24 Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *J. Comput. Syst. Sci.*, 37(1):63–78, 1988.

25 T. Warren Liao. Clustering of time series data - a survey. *Pattern Recognit*, 38(11):1857–1874, 2005.

26 Jia Jie Liu, Guan-Shieng Huang, Yue-Li Wang, and Richard C. T. Lee. Edit distance for a run-length-encoded string and an uncompressed string. *Information Processing Letters*, 105(1):12–16, 2007.

27 Alexander De Luca, Alina Hang, Frederik Brudy, Christian Lindner, and Heinrich Hussmann. Touch me once and i know it's you!: implicit authentication based on touch screen patterns. In *SIGCHI Conference on Human Factors in Computing Systems*, pages 987–996. ACM, 2012.

28 Veli Mäkinen, Gonzalo Navarro, and Esko Ukkonen. Approximate matching of run-length compressed strings. *Algorithmica*, 35(4):347–369, 2003.

29 J. Mitchell. *A geometric shortest path problem, with application to computing a longest common subsequence in run-length encoded strings.* Tech-nical Report, Department of Applied Mathemat-ics, SUNY StonyBrook, NY, 1997.

30 Lindasalwa Muda, Mumtaj Begam, and Irraivan Elamvazuthi. Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques. arXiv preprint, 2010.

31 Abdullah Mueen, Nikan Chavoshi, Noor Abu-El-Rub, Hossein Hamooni, and Amanda J. Minnich. Awarp: Fast warping distance for sparse time series. In *16th ICDM*, pages 350–359, 2016.

32 Abdullah Mueen, Nikan Chavoshi, Noor Abu-El-Rub, Hossein Hamooni, Amanda J. Minnich, and Jonathan MacCarthy. Speeding up dynamic time warping distance for sparse time series data. *Knowl. Inf. Syst.*, 54(1):237–263, 2018.

33 Mario E. Munich and Pietro Perona. Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In *7th ICCV*, pages 108–115, 1999.

34 Eugene W. Myers. An O(ND) difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986.

35 Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

36 Yoshifumi Sakai and Shunsuke Inenaga. A reduction of the dynamic time warping distance to the longest increasing subsequence length. In *31st ISAAC*, pages 6:1–6:16, 2020.

37 Yoshifumi Sakai and Shunsuke Inenaga. A faster reduction of the dynamic time warping distance to the longest increasing subsequence length. *Algorithmica*, 84(9):2581–2596, 2022.

38 Nathan Schaar, Vincent Froese, and Rolf Niedermeier. Faster binary mean computation under dynamic time warping. In *31st CPM*, pages 28:1–28:13, 2020.

39 Taras K. Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57, 1968.

40 Xiaoyue Wang, Abdullah Mueen, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn J. Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Min. Knowl. Discov.*, 26(2):275–309, 2013.

41 Zoe Xi and William Kuszmaul. Approximating dynamic time warping distance between run-length encoded strings. In *30th ESA*, pages 90:1–90:19, 2022.

42 Rex Ying, Jiangwei Pan, Kyle Fox, and Pankaj K. Agarwal. A simple efficient approximation algorithm for dynamic time warping. In *24th ACM SIGSPATIAL*, pages 21:1–21:10, 2016.

43 Yunyue Zhu and Dennis Shasha. Warping indexes with envelope transforms for query by humming. In *22nd ACM SIGMOD*, pages 181–192, 2003.

Tight Bounds on Adjacency Labels for Monotone Graph Classes

Édouard Bonnet
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Julien Duron
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

John Sylvester
Department of Computer Science, University of Liverpool, UK

Viktor Zamaraev
Department of Computer Science, University of Liverpool, UK

Maksim Zhukovskii
Department of Computer Science, University of Sheffield, UK

Abstract

A class of graphs admits an adjacency labeling scheme of size $b(n)$, if the vertices in each of its n-vertex graphs can be assigned binary strings (called labels) of length $b(n)$ so that the adjacency of two vertices can be determined solely from their labels.

We give bounds on the size of adjacency labels for every family of monotone (i.e., subgraph-closed) classes with a "well-behaved" growth function between $2^{\Omega(n \log n)}$ and $2^{O(n^{2-\delta})}$ for any $\delta > 0$. Specifically, we show that for any function $f : \mathbb{N} \to \mathbb{R}$ satisfying $\log n \leqslant f(n) \leqslant n^{1-\delta}$ for any fixed $\delta > 0$, and some sub-multiplicativity condition, there are monotone graph classes with growth $2^{O(nf(n))}$ that do not admit adjacency labels of size at most $f(n) \log n$. On the other hand, any such class does admit adjacency labels of size $O(f(n) \log n)$. Surprisingly this bound is a $\Theta(\log n)$ factor away from the information-theoretic bound of $\Omega(f(n))$. Our bounds are tight upto constant factors, and the special case when $f = \log$ implies that the recently-refuted Implicit Graph Conjecture [Hatami and Hatami, FOCS 2022] also fails within monotone classes.

We further show that the Implicit Graph Conjecture holds for all monotone *small* classes. In other words, any monotone class with growth rate at most $n!\, c^n$ for some constant $c > 0$, admits adjacency labels of information-theoretic order optimal size. In fact, we show a more general result that is of independent interest: any monotone small class of graphs has bounded degeneracy. We conjecture that the Implicit Graph Conjecture holds for all hereditary small classes.

2012 ACM Subject Classification Mathematics of computing → Combinatorics; Mathematics of computing → Graph theory

Keywords and phrases Adjacency labeling, degeneracy, monotone classes, small classes, factorial classes, implicit graph conjecture

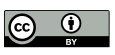
Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.31

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2310.20522` [11]

Funding This work has been supported by Research England funding to enhance research culture, by the Royal Society (IES\R1\231083), by the ANR projects TWIN-WIDTH (ANR-21-CE48-0014) and Digraphs (ANR-19-CE48-0013), and also the EPSRC project EP/T004878/1: Multilayer Algorithmics to Leverage Graph Structure.

Acknowledgements We are grateful to Nathan Harms for valuable feedback on the early version of this paper.



1 Introduction

A *class* of graphs is a set of graphs which is closed under isomorphism. For a class of graphs $\mathcal{X}$ we denote by $\mathcal{X}_n$ the set of graphs in $\mathcal{X}$ with vertex set $[n]$. The function $n \mapsto |\mathcal{X}_n|$ is called the *speed* of $\mathcal{X}$. A *coding* of graphs is a representation of graphs by words in the binary alphabet $\{0,1\}$. One of the main considerations with graph representations is their succinctness; clearly, any representation of n-vertex graphs in a class $\mathcal{X}$ would require at least $\lceil \log |\mathcal{X}_n| \rceil$ bits for some graphs in $\mathcal{X}_n$.

Another consideration is whether the representation is global or local. Standard graph representations, such as adjacency matrix or adjacency lists, are examples of *global* representations, where a graph is stored in a single data structure that needs to be accessed in order to query some information about the graph, e.g., adjacency between a pair of vertices. By contrast, in *local* graph representations, the encoding of a graph is distributed over its vertices in such a way that the queries can be answered by looking only into the local information associated with the vertices involved in the query. In this work we are concerned with local graph representations for adjacency queries, i.e., queries that given two vertices answer whether they are adjacent or not.

Let $\mathcal{X}$ be a class of graphs and $b : \mathbb{N} \to \mathbb{N}$ be a function. A *$b(n)$-bit adjacency labeling scheme* (or simply *$b(n)$-bit labeling scheme*) for $\mathcal{X}$ is a pair (encoder, decoder) of algorithms where for any n-vertex graph $G \in \mathcal{X}_n$ the encoder assigns binary strings, called *labels*, of length $b(n)$ to the vertices of G such that the adjacency between any pair of vertices can be inferred by the decoder only from their labels. We note that the decoder depends on the class $\mathcal{X}$, but not on the graph G. The function $b(\cdot)$ is the *size* of the labeling scheme. Adjacency labeling schemes were introduced by Kannan, Naor, and Rudich [24, 25], and independently by Muller [28] in the late 1980s and have been actively studied since then. Adjacency labeling schemes are closely related to induced universal graphs, which we will refer to simply as universal graphs. For a function $u : \mathbb{N} \to \mathbb{N}$, a *universal graph sequence* or simply *universal graph* of size $u(n)$ is a sequence of graphs $(U_n)_{n\in\mathbb{N}}$ such that for every $n \in \mathbb{N}$ the graph U_n has at most $u(n)$ vertices and every n-vertex graph in $\mathcal{X}$ is an induced subgraph of U_n. It was observed in [25] that for a class of graphs the existence of a $b(n)$-bit labeling scheme is equivalent to the existence of a universal graph of size $2^{b(n)}$.

The binary word, obtained by concatenating labels of the vertices of a graph $G \in \mathcal{X}_n$ assigned by an adjacency labeling scheme, uniquely determines graph G. Thus, a $b(n)$-bit labeling scheme cannot represent more than $2^{nb(n)}$ graphs on n vertices, and therefore, if $\mathcal{X}$ admits a $b(n)$-bit labeling scheme, then $|\mathcal{X}_n| \leqslant 2^{nb(n)}$. This implies a lower bound of $\frac{\log |\mathcal{X}_n|}{n}$ on the size $b(n)$ of any adjacency labeling scheme for $\mathcal{X}$. A natural and important question is: which classes admit an adjacency labeling scheme of a size that matches this information-theoretic lower bound?

We say that a graph class $\mathcal{X}$ admits an *implicit representation*, if it admits an information-theoretic *order optimal* adjacency labeling scheme, i.e., if $\mathcal{X}$ has a $b(n)$-bit labeling scheme, where $b(n) = O(\log |\mathcal{X}_n|/n)$. Equivalently, $\mathcal{X}$ admits an implicit representation if $\mathcal{X}$ has a universal graph of size $\exp(O(\log |\mathcal{X}_n|/n))$. For example, the class $\mathcal{A}$ of all graphs admits an implicit representation, because

$$|\mathcal{A}_n| = 2^{\binom{n}{2}} = 2^{\Theta(n^2)} \text{ and } b(n) = O\left(\frac{\log |\mathcal{A}_n|}{n}\right) = O(n),$$

and one can easily design an $O(n)$-bit labeling scheme for $\mathcal{A}$, e.g., by assigning to each vertex of a graph an $(n + \lceil \log n \rceil)$-bit label consisting of the row in an adjacency matrix of the graph corresponding to the vertex and the index of that row; in fact, as we discuss below, the class of all graphs admits an asymptotically optimal $(1 + o(1))n/2$-bit labeling scheme [3].

However, not every class admits an implicit representation. The following example is due to Muller [28] (see also [32]). Let $\mathcal{Y}$ be the class of graphs in which the number of edges does not exceed the number of vertices. It is easy to estimate that $|\mathcal{Y}_n| = 2^{O(n \log n)}$. To show that this class does not admit an implicit representation, consider an arbitrary n-vertex graph G. Obviously, G does not necessarily belong to $\mathcal{Y}$, but after adding $n^2 - n$ isolated vertices to G, we obtain a graph H on $N = n^2$ vertices that belongs to $\mathcal{Y}$. Now, if an $O(\log n)$-bit labeling scheme for $\mathcal{Y}$ existed, then the $O(\log N)$-bit adjacency labels for H could be used as $O(\log n)$-bit adjacency labels for G. Since, G was chosen arbitrarily, this is in contradiction with the lower bound of $\frac{\log |\mathcal{A}_n|}{n} = \Omega(n)$ on the size of any labeling scheme for the class $\mathcal{A}$ of all graphs.

The crucial property used in the above example is that by adding isolated vertices to a graph not in $\mathcal{Y}$, one can obtain a graph in $\mathcal{Y}$. Using more familiar terminology, one would say that class $\mathcal{Y}$ is not *hereditary*, i.e., it is not closed under vertex removal or, equivalently, under taking induced subgraphs. Many natural graph classes (e.g., forests, planar graphs, bipartite graphs, geometric intersection graphs) are hereditary. It turns out that finding a hereditary graph class that does not admit an implicit representation is a non-trivial question. The first instance of this question was asked by Kannan, Naor, and Rudich [24] for *factorial classes* (i.e., graph classes $\mathcal{X}$ with the speed $|\mathcal{X}_n| = 2^{O(n \log n)}$), which was later stated by Spinrad [32] in the form of a conjecture, that became known as the *Implicit Graph Conjecture*.

(*IGC*): Any hereditary graph class of at most factorial speed admits an $O(\log n)$-bit labeling scheme.

This question remained open for over 30 years until a recent breakthrough by Hatami and Hatami [23]. They showed that, for any $\delta > 0$, there exists a hereditary factorial class that does not admit a labeling scheme of size $n^{1/2-\delta}$, which is very far from the information-theoretic lower bound of $\Omega(\log n)$. This result leaves wide open the question of characterizing factorial hereditary graph classes that admit an implicit representation (see e.g. [22] for more discussion).

Factorial hereditary classes form an important family, as many classes of theoretical or practical interest are factorial (e.g., forests, planar graphs, disk graphs, graphs of bounded twin-width). However, as was noted by Spinrad [32], there is nothing that prevents one from considering implicit representability of other hereditary graph classes. Spinrad [32] raised this as the *Generalized Implicit Graph Question*, which we restate using the terminology of our paper as follows.

▶ **Question 1** ([32]). *Which hereditary graph classes admit implicit representations?*

The answer to this question is known for classes with $|\mathcal{X}_n| = 2^{\Omega(n^2)}$, and for *subfactorial* graph classes, i.e., classes $\mathcal{X}$ with $|\mathcal{X}_n| = 2^{o(n \log n)}$. Indeed, for the latter classes, it is known that they have at most exponential speed, i.e., $|\mathcal{X}_n| = 2^{O(n)}$ [2, 31], and also admit $O(1)$-bit labeling schemes [30]. For the former classes, the $O(n)$-bit labeling scheme mentioned above for the class $\mathcal{A}$ of all graphs is an order optimal labeling scheme. In fact, in this regime, *information-theoretic asymptotically optimal* (up to the second-order term) labeling schemes are available. For the class of all graphs, such results (in the language of universal graphs) were available since 1965 [27, 6, 3]. For proper hereditary graph classes $\mathcal{X}$ with the speed $2^{\Omega(n^2)}$, by the Alekseev–Bollobás–Thomason theorem [1, 9], their speed is $|\mathcal{X}_n| = 2^{(1-1/k(\mathcal{X}))n^2/2+o(n^2)}$, where $k(\mathcal{X})$ is an integer greater than 1. Recently, Bonamy, Esperet, Groenland, and Scott showed [10] that all such classes have asymptotically optimal adjacency labeling schemes of size $(1 - 1/k(\mathcal{X}))n/2 + o(n)$.

For the classes in the intermediate range, i.e., the classes with the speed between $2^{\Omega(n \log n)}$ and $2^{o(n^2)}$ the picture is much less understood (see Figure 1). Most known information is concentrated on the lower extreme of the range, i.e., around factorial speed, which was promoted by the Implicit Graph Conjecture. Factorial graph classes from certain families are known to admit implicit representations: proper minor-closed graph classes [20], graph classes of bounded degeneracy (equivalently, of bounded arboricity) [24], clique-width [15, 32] (see also [7]), and twin-width [13] all admit implicit representations. The only lower bound witnessing (non-constructively) factorial classes[1] that do not admit an implicit representation is the above-mentioned result by Hatami and Hatami [23]. A notable family of hereditary graph classes where Question 1 remains open is the *small* graph classes, i.e., classes $\mathcal{X}$ with $|\mathcal{X}_n| \leqslant n! \, c^n$ for some constant c. These classes encompass only the bottom part of the factorial layer and include proper minor-closed classes [8, 29], and more generally, classes of bounded twin-width [13]. However, it is still unknown if all such classes admit an implicit representation (see [12] for more details on implicit representation of small classes). Alon showed [4] that every hereditary graph class $\mathcal{X}$ with $|\mathcal{X}_n| = 2^{o(n^2)}$ admits an $n^{1-\delta}$-bit labeling scheme for some $\delta > 0$.

1.1 Our contribution

In this paper, we study Question 1 for *monotone* graph classes, i.e., graph classes that are closed under taking subgraphs. Monotone graph classes form a subfamily of hereditary graph classes. Many interesting and natural classes are monotone, for example classes of bounded chromatic number/index, bounded clique number, bounded genus, triangle free, and so on. Together with some previous results mentioned in the introduction, the contribution of this paper is to give a near complete resolution of Question 1 for monotone classes.

The *degeneracy* of a graph G is the minimum d such that every subgraph of G has a vertex of degree at most d. We say that a class of graphs $\mathcal{X}$ has bounded degeneracy, if there exists a constant d such that the degeneracy of every graph $G \in \mathcal{X}$ is at most d; otherwise, we say that $\mathcal{X}$ has unbounded degeneracy. Our first main result shows that degeneracy is bounded for monotone small classes.

▶ **Theorem 2.** *Let $\mathcal{X}$ be a monotone small class. Then, $\mathcal{X}$ has bounded degeneracy.*

Theorem 2 has wider reaching implications than just labeling schemes, and is of independent interest. In the context of Question 1, we obtain the following result from Theorem 2 and a classical labeling scheme for classes of bounded degeneracy [24] (see also Lemma 14).

▶ **Theorem 3.** *Any monotone small class admits an implicit representation.*

This answers Question 1 from [12] for monotone graph classes.

We now turn to monotone classes that are not small. Our next result shows that any monotone class with non-decreasing speed admits a labeling scheme of size at most $O(\log n)$ away from the information-theoretic lower bound.

▶ **Proposition 4.** *Let $f : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ be a non-decreasing function. Then, any monotone class of graphs $\mathcal{X}$ with speed $|\mathcal{X}_n| = 2^{O(nf(n))}$ admits an adjacency labeling scheme of size $O(f(n) \log n)$.*

[1] This lower bound is sufficiently large to rule out the existence of implicit representations even for hereditary classes of size $2^{\Theta(n^{3/2-\delta})}$, for any fixed $0 < \delta < 1/2$.

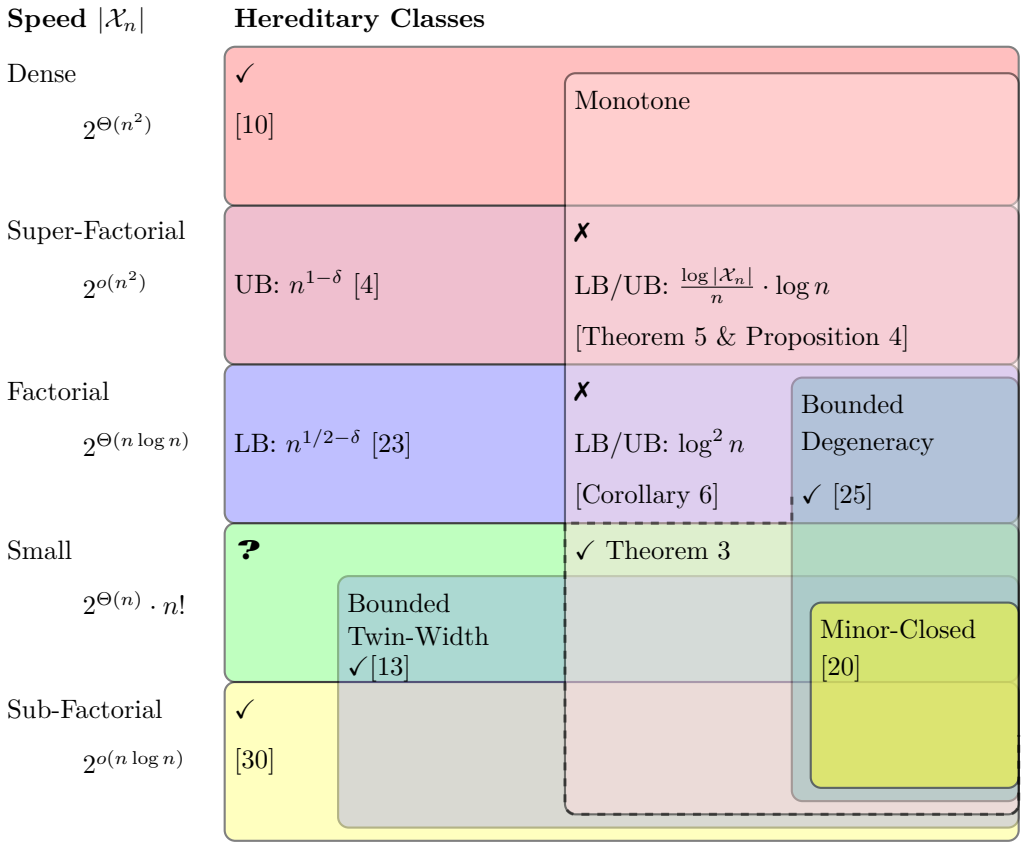


Figure 1 A ✓ indicates that all classes of the given type have an implicit representation, a ✗ shows that they do not, and a **?** signals that the question is open. A ✓ is inherited by every sub-region, a ✗ is inherited to the left of the marked region, and a **?** only holds in that region. The upper and lower bounds (UB and LB respectively) are stated up to constants which may depend on the class. The dashed extension of the *bounded degeneracy* region illustrates its containment of monotone small classes (Theorem 3).

This upper bound is an easy consequence of an estimation of the number of edges in graphs from monotone classes combined with a standard labeling scheme for k-degenerate graphs [24]. Our second main result shows that this upper bound is attained by some monotone classes. Before stating the result formally we must briefly introduce a family of non-decreasing functions we call "decent". Roughly speaking, on some domain $[s, \infty)$, decent functions are sub-multiplicative, i.e., $f(xy) \leqslant f(x)f(y)$, and moderate-growing, that is $\log x \leqslant f(x) \leqslant x^{1-\delta}$ for some constant $\delta \in (0, 1)$, see Definition 16 for the formal definition of decent functions.

▶ **Theorem 5.** *Let $f : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ be a decent function. Then, there exists a monotone graph class $\mathcal{X}$ with speed $|\mathcal{X}_n| = 2^{O(nf(n))}$ that does not admit a universal graph of size at most $2^{f(n)\log n}$. Equivalently, $\mathcal{X}$ admits no adjacency labeling scheme of size at most $f(n)\log n$.*

Theorem 5 gives the existence of monotone classes requiring labels whose size is a $\log n$-factor above the information-theoretic lower bound. In particular, this shows that Proposition 4 is tight. A special case of Theorem 5 (when $f(x) = \log x$) implies that the Implicit Graph Conjecture does not hold even for monotone graph classes. Combining this observation with Proposition 4 gives the following result.

▶ **Corollary 6.** *For any constant $c > 0$, there are factorial monotone classes that do not admit a $(c \log^2 n)$-bit labeling scheme, while any factorial monotone class admits an $O(\log^2 n)$-bit labeling scheme.*

This result (more generally Theorem 5 and Proposition 4) gives the first example of tight bounds for families of graph classes that do not admit information-theoretic order optimal adjacency labeling schemes. Chandoo [14] observed that the proof of the refutation of the IGC by Hatami and Hatami [23] implies that the family of factorial classes cannot be "described" by a countable set of factorial classes. Using the same ideas, we establish the following result from our proof for monotone classes.

▶ **Theorem 7.** *Let $f : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ be any decent function, and $\mathbb{X}$ be any countable set of graph classes, each with speed at most $2^{nf(n)\log n}$. Then, there exists a monotone graph class $\mathcal{X}$ of speed $2^{O(nf(n))}$ such that there does not exist a $\mathcal{D} \in \mathbb{X}$ with $\mathcal{X} \subseteq \mathcal{D}$.*

This shows that monotone classes are complex in the sense that they cannot be covered by a countably infinite family of classes growing slightly faster, even if these classes are not restricted to being hereditary (thus, also to being monotone).

1.2 Proof outline and techniques

Monotone small classes have bounded degeneracy and implicit representations

We establish Theorem 2 in the contrapositive: if a monotone class $\mathcal{X}$ has unbounded degeneracy, then it is not small. To prove this we establish the following two intermediate steps:

1. We first show that every graph of minimum degree d admits an induced subgraph with minimum degree at least d that has a spanning tree of *maximum* degree at most d.
2. Next, we show that if $\mathcal{G} = \mathrm{Mon}(\{G\})$, where G is any graph with minimum degree $d \geqslant 1000$, then there exists a $k \in \mathbb{N}$ such that $|\mathcal{G}_k| \geqslant k! \cdot 2^{kd/3}$.

   To achieve this, we start from an induced graph H of G satisfying the previous item, with $k = |V(H)|$ and $m = |E(H)|$. Graph H can be shown to have at least $2^{4m/5}$ pairwise non-isomorphic spanning subgraphs, due to its large minimum degree. Let us denote by $\mathcal{F}$ this set of subgraphs. Crucially each member of $\mathcal{F}$ has at most $2^{m/10}$ automorphisms, due to the spanning tree of bounded maximum degree. We conclude since $|\mathcal{G}_k| \geqslant \sum_{F \in \mathcal{F}} \frac{k!}{\mathrm{aut}(F)}$.

Finally, to show the contrapositive of Theorem 2, we consider an arbitrary monotone class $\mathcal{X}$ of unbounded degeneracy and assume that for some constant c we have $|\mathcal{X}_n| \leqslant n!\, c^n$ for every $n \in \mathbb{N}$. Since $\mathcal{X}$ has unbounded degeneracy, it contains a graph G with arbitrarily large minimum degree d. If we take d suitably large, then applying Step 2 to such a graph yields a contradiction with the assumption of smallness of $\mathcal{X}$.

Having established Theorem 2, any small monotone class $\mathcal{X}$ has bounded degeneracy. Thus, Theorem 3 follows by applying a classical $O(\log n)$-bit labeling scheme for classes of bounded degeneracy [25], see Lemma 14 for a description of this scheme.

Monotone classes that do not admit implicit representations

Recall that, roughly speaking[2], a function $f : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ is decent if $\log x \leqslant f(x) \leqslant x^{1-\delta}$ for some constant $\delta \in (0, 1)$, and f is sub-multiplicative, i.e., $f(xy) \leqslant f(x) \cdot f(y)$, for all x, y in the domain. Our approach is inspired by the refutation of the IGC by Hatami and Hatami [23]. Namely, for any decent function f, we expose so many *monotone* classes of speed $2^{nf(n)}$ that there are not enough universal graphs of size $2^{f(n)\log n}$ to capture all of them. The approach involves several key ingredients:

1. Estimation of the number of sets of graphs of fixed cardinality representable by universal graphs. A set of graphs $\mathcal{M}$ is *representable* by a universal graph U, if every graph in $\mathcal{M}$ is an induced subgraph of U. A direct estimation shows that the number of sets of cardinality $k_n := \lceil 2^{\sqrt{nf(n)}} \rceil$ of n-vertex graphs that are representable by a u_n-vertex universal graph, with $u_n := 2^{f(n)\log n}$, is at most
$$2^{u_n^2} \cdot u_n^{nk_n} = 2^{2^{2f(n)\log n} + k_n \cdot nf(n)\log n}. \tag{1}$$
2. Notion of *f-good graphs*. We will construct our monotone classes of speed $2^{nf(n)}$ by taking the monotone closure of an appropriately chosen set of graphs. The monotonicity and the speed of target classes impose a natural restriction on the number of edges in graphs that can be used in such constructions. To explain, let $\mathcal{X}$ be a monotone class with $|\mathcal{X}_n| \leqslant 2^{nf(n)}$. Since $\mathcal{X}$ is closed under taking subgraphs, if $\mathcal{X}$ contains an n-vertex graph with m edges, then $\mathcal{X}$ contains at least 2^m labeled n-vertex graphs. This, together with the speed assumption, imply that for any $G \in \mathcal{X}$ and k, every subgraph of G on k vertices contains at most $kf(k)$ edges.

   This restriction, however, is not strong enough for our purposes. Indeed, while each graph with the above property contributes to the monotone closure an appropriate number of subgraphs at every *level* (i.e., on every number of vertices), we build our desired classes by taking the monotone closure of *infinitely* many of such graphs, and this can result in some levels having too many graphs. To overcome this difficulty, we introduce the notion of *f-good* graphs, which are n-vertex graphs in which the number of edges in every k-vertex subgraph is at most $kf(k)$ if $k > \sqrt{n}$, and at most $\frac{kf(k)}{\log k}$ if $2 \leqslant k \leqslant \sqrt{n}$. The latter condition ensures that if we take the monotone closure of a set of f-good graphs, then all sufficiently small subgraphs of any graph in this class belong to a *fixed* monotone class of speed $2^{nf(n)}$. Namely, the class of all n-vertex graphs in which very k-vertex subgraph has at most $\frac{kf(k)}{\log k}$ edges for every $2 \leqslant k \leqslant n$.
3. Construction of monotone classes of speed $2^{nf(n)}$ from sets of f-good graphs. We show that for any sequence $(\mathcal{M}_n)_{n\in\mathbb{N}}$, where $\mathcal{M}_n$ is a set of f-good n-vertex graphs of cardinality k_n, the monotone closure $\mathrm{Mon}(\cup_{n\in\mathbb{N}}\mathcal{M}_n)$ has speed at most $2^{nf(n)}$.
4. Lower bound on the number of sets of cardinality k_n of f-good n-vertex graphs. We show that for any $\gamma > 1$, there exists some $c := c(\gamma, \delta) > 0$ such that for every $n \in \mathbb{N}$ there are at least $2^{(\gamma\delta/2 - o(1)) \cdot nf(n)\log n}$ many unlabeled cf-good n-vertex graphs. Thus, the number of sets of cardinality k_n of cf-good n-vertex graphs is at least
$$2^{k_n \cdot (\gamma\delta/2 - o(1)) \cdot nf(n)\log n}. \tag{2}$$

[2] The formal definition of decent (Definition 16) is more general and depends on three parameters δ, C, s. For this proof sketch it suffices to work with the simplified (informal) definition above which only has one parameter δ.

By setting $\gamma = 4/\delta$ and recalling that $k_n = \lceil 2^{\sqrt{nf(n)}} \rceil$, we show that Equation (2) is larger than Equation (1). Therefore, there exists a monotone class $\mathrm{Mon}(\cup_{n\in\mathbb{N}}\mathcal{M}_n)$ of speed $2^{nf(n)}$ that is not representable by a universal graph of size $2^{f(n)\log n}$.

Many f-good graphs

A core step in the above approach is to show that for any $\gamma > 1$, there exists some $c := c(\gamma, \delta) > 0$ such that the number of n-vertex cf-good graphs grows as $2^{(\gamma\delta/2-o(1))\cdot nf(n)\log n}$. To do so, we show that a random graph $G_n \sim G(n, \gamma f(n)/n)$ is cf-good with high probability (*w.h.p.*). It is in this step that we really need to use the sub-multiplicativity property of decent functions, as we need to relate the magnitude of f at two different parts of its domain.

In particular, to show that w.h.p. G_n is cf-good, we apply a first moment bound to show there are no "large" k-vertex subgraphs of G_n with more than $ckf(k)$ edges, and "small" ones with more than $ckf(k)/\log k$ edges. Observe that the number of edges ξ in a given k-vertex subgraph has expectation $\binom{k}{2}\frac{\gamma f(n)}{n}$. Thus, for "large" subgraphs, the probability that ξ is constant factor larger than $ckf(k)$ decays with exponent $\propto -f(k)\cdot\ln\frac{nf(k)}{kf(n)}$ by the Chernoff bound. From this we see that unless $f(k)/f(n) > k/n$, then the bound fails. Sub-multiplicativity helps us here as it allows us to say $f(n) = f(k\cdot(n/k)) \leqslant f(k)\cdot f(n/k)$, moderate-growth then bounds the term $f(n/k)$. A similar issue occurs for "small" subgraphs.

From the explanation above it may seem that needing such tight control over the ratio of $f(k)$ to $f(n)$ for all $k \leqslant n$ is an artifact of our proof, however some "smoothness" condition on the function is necessary. To see this, consider a function $f : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ such that $f(n) = \log n$, if n is odd, and $f(n) = \sqrt{n}$, if n is even. Then, for any $c > 0$, and large enough even n, $G(n, f(n)/n)$ will not be cf-good as the restriction on the subgraphs with odd number of vertices is far too stringent. Sub-multiplicativity was the most natural and broad condition we could find to combat this issue, and we show in Lemma 17 that many common functions growing at a suitable rate satisfy this.

It would be interesting to see if sub-multiplicativity can be replaced with something more general. We also used sub-multiplicativity in Step 3 above (which corresponds to Lemma 22) to bound the speed of $\mathrm{Mon}(\cup_{n\in\mathbb{N}}\mathcal{M}_n)$, however it is possible that some less stringent property can be used there.

A matching upper bound on the size of adjacency labels

We show that for any non-decreasing function $f : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$, any monotone class with speed $2^{O(nf(n))}$ admits an $O(f(n)\log n)$-bit labeling scheme. This follows from an easy observation that any such class is $O(f(n))$-degenerate, followed by the same standard $O(k\log n)$-bit labeling scheme for k-degenerate graphs used to prove Theorem 3. One consequence of this upper bound is that our result on the "f-goodness" of a random graph (Theorem 18) is tight: for any $p \in \omega(f(n)/n)$ and $c \geqslant 0$, a random graph $G_n \sim G(n,p)$ is not cf-good w.h.p.

1.3 Discussion

A natural question arising from our work is to characterize monotone classes that admit an implicit representation. Motivated by the Implicit Graph Conjecture, of particular interest is the case of factorial classes.

▶ **Question 8.** *Which* monotone *factorial graph classes admit an* $O(\log n)$*-bit labeling scheme?*

An analogous question is completely understood for constant-size *adjacency sketches* (a probabilistic version of adjacency labeling schemes) that were studied in [18, 21, 22]. The importance of constant-size adjacency sketches is that they can be derandomized to $O(\log n)$-bit adjacency labels [21, 22]. Thus, if a class admits constant-size adjacency sketches, then it admits an $O(\log n)$-bit labeling scheme. Though, the converse is not always true. Esperet, Harms, and Kupavskii showed [16] that a monotone class admits constant-size adjacency sketches if and only if it has bounded degeneracy. This result may suggest that bounded degeneracy also characterizes monotone classes that admit $O(\log n)$-bit labeling schemes. This, however, is not the case, as the class of subgraphs of hypercubes is monotone, has unbounded degeneracy, and admits an $O(\log n)$-bit labeling scheme [17].

Recall that Question 1 (first raised in [32]), asks which hereditary graph classes admit implicit representations. A prominent instance of Question 1 is whether every hereditary small class admits an implicit representation. It was shown in [12] that for any $\kappa > 0$ there is a monotone small class which does not admit a $(\kappa \log n)$-bit labeling scheme; in particular, some monotone small classes admit no *information-theoretic asymptotically optimal* labeling scheme. One of our main results (Theorem 3) shows that every monotone class admits an *information-theoretic order optimal* labeling scheme, i.e., an implicit representation. We conjecture that the same holds for all hereditary small classes.

▶ **Conjecture 9** (Small Implicit Graph Conjecture). *Any hereditary small class admits an implicit representation.*

Conjecture 9 is also known to hold for classes of bounded twin-width [13].

We conclude this discussion with a more technical (yet natural) question of whether the conditions (moderate-growth and sub-multiplicativity) of "decent" can be relaxed. Due to the discussion under the heading "Many f-good graphs" in Section 1.2, the moderate-growth condition is essentially necessary, and if one is to follow our method, some notion of global "smoothness" is required to prove Theorem 18. However, it is not so clear to what extent the sub-multiplicativity condition is necessary to achieve the required "smoothness".

1.4 Organization

The rest of the paper is organized as follows. In Section 2, we cover some common notation and definitions. In Section 3, we prove our first main result, namely, that any monotone small class has bounded degeneracy, and therefore admits an implicit representation. Sections 4 and 5 are devoted to our second main result, namely, tight bounds on the size of adjacency labeling schemes for monotone classes. In Section 4.1 we introduce two key concepts used in the proofs. Firstly, we give the notion of *f-good* graphs, which are the building blocks for the monotone classes used to prove the lower bounds. Secondly, we formally define *decent* functions which describe the speeds of these monotone graph classes, before concluding Section 4.1 with some natural examples of decent functions. In Section 4.2, we prove a result about random graphs which is the main technical ingredient of our lower bounds. In Section 5, we establish the lower and upper bounds on labeling schemes for monotone classes, along with the result on the complexity of monotone graph classes.

2 Standard definitions and notation

We let $[n]$ denote the set $\{1, \dots, n\}$ of natural numbers, and use $\ln^c x$ as a shorthand for $(\ln x)^c$. We take $\mathbb{R}_{\geqslant 0}$ to denote the set of non-negative real numbers. We use $X \sim \mathcal{D}$ to denote that the random variable X has distribution $\mathcal{D}$. We say that a sequence of events (A_n) holds *with high probability (w.h.p.)* if $\mathbb{P}[A_n] \to 1$ as $n \to \infty$.

We consider finite undirected graphs, without loops or multiple edges. Given a graph G, we write $V(G)$ for its vertex set, and $E(G)$ for its edge set. A graph H is a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. Thus, H can be obtained from G by vertex and edge deletions. The graph H is an *induced subgraph* of G if $V(H) \subseteq V(G)$, and $E(H)$ consists exactly of the edges in $E(G)$ with both endpoints in $V(H)$. In that case, H can be obtained from G by vertex deletions only. In the usual way, for a set of vertices $U \subseteq V(G)$, we denote by $G[U]$ the induced subgraph of G with the set of vertices U. We denote by $e(G)$ the number of edges in G

When we refer to an n-vertex graph G as *labeled*, we mean that the vertex set of G is $[n]$, and we distinguish two different labeled graphs even if they are isomorphic. In contrast, if we refer to G as *unlabeled* graph, its vertices are indistinguishable and two isomorphic graphs correspond to the same unlabeled graph.

A graph class is *hereditary* if it is closed under taking induced subgraphs, and it is *monotone* if it closed under taking subgraphs. For a set $\mathcal{X}$ of graphs we let $\mathrm{Her}(\mathcal{X})$ denote the hereditary closure of $\mathcal{X}$, i.e., the inclusion-wise minimal hereditary class that contains $\mathcal{X}$; and $\mathrm{Mon}(\mathcal{X})$ denote the monotone closure of $\mathcal{X}$, i.e., the minimal monotone class that contains $\mathcal{X}$.

3 Monotone small classes admit an implicit representation

In this section we show that any monotone small class admits an implicit representation. To do so, we first establish (Theorem 2) that any small monotone classes has bounded degeneracy. This result has a broader scope than just implicit representations, and is of independent interest. For example, it generalizes the fact that monotone classes of bounded twin-width have bounded degeneracy [13, $(iv) \Rightarrow (iii)$ in Theorem 2.12]. The labeling scheme for monotone small classes then follows from a classical labeling scheme for graphs of bounded degeneracy, see Lemma 14.

We proceed with some notation and known auxiliary facts that we will employ in the proof. Recall that for a class of graphs $\mathcal{X}$, we denote by $\mathcal{X}_n$ the set of graphs in $\mathcal{X}$ with vertex set $[n]$. We will denote by $\mathcal{X}_n^u$ the set of *unlabeled* n-vertex graphs in $\mathcal{X}$, i.e., the set of isomorphism classes in $\mathcal{X}_n$. Observe that for an unlabeled n-vertex graph G there are exactly $\frac{n!}{\mathrm{aut}(G)}$ labeled graphs isomorphic to G, where $\mathrm{aut}(G)$ is the order of the automorphism group of G. Thus we have

$$|\mathcal{X}_n| = \sum_{G \in \mathcal{X}_n^u} \frac{n!}{\mathrm{aut}(G)}.$$

Let F be a spanning subgraph of a fixed labeled graph G. Thus, we recall, F is defined by a subset of $E(G)$. We denote by $\#\mathrm{Sub}(F \to G)$ the number of subgraphs of G isomorphic to F, and by $\#\mathrm{Emb}(F \to G)$ the number of embeddings of F into G, i.e., the number of permutations from $\mathfrak{S}_n$ that map F to an isomorphic copy of F in G. Thus,

$$\#\mathrm{Emb}(F \to G) = \#\mathrm{Sub}(F \to G) \cdot \mathrm{aut}(F).$$

We will use the following well known facts (see e.g. [26]).

▶ **Lemma 10.** *Let F be a spanning subgraph of a graph G. Then*

$$\mathrm{aut}(G) \leqslant \#\mathrm{Emb}(F \to G) = \#\mathrm{Sub}(F \to G) \cdot \mathrm{aut}(F).$$

▶ **Lemma 11.** *Let G be a connected graph of maximum degree Δ. Then*

$$\mathrm{aut}(G) \leqslant n \cdot \Delta! \cdot (\Delta - 1)^{n-\Delta-1} \leqslant n\Delta^{n-1}.$$

We start with the following deceptively simple lemma.

▶ **Lemma 12.** *Let G be a graph of minimum degree d. Then G has an induced subgraph H of minimum degree at least d with a spanning tree of* maximum degree *at most d.*

Proof. Let T be an inclusion-wise maximal tree among subgraphs of G with maximum degree at most d. Let H be $G[V(T)]$. Note that any vertex v of H that has degree less than d in T, has no neighbors among $V(G) \setminus V(H)$ in G, as otherwise T could have been extended by adding any neighbor of v from $V(G) \setminus V(H)$, which would contradict the maximality of T. Thus all neighbors of v are in $V(H)$, which implies that the minimum degree of H is at least d. ◀

We show next that if a graph has large minimum degree and a spanning tree of bounded maximum degree, then it contains many pairwise non-isomorphic spanning connected subgraphs with a small number of automorphisms.

▶ **Lemma 13.** *Let G be an n-vertex m-edge connected graph of minimum degree $d \geqslant 1000$, with a spanning tree of maximum degree at most d. Then, G contains at least $2^{4m/5}$ pairwise non-isomorphic spanning connected subgraphs F with* $\mathrm{aut}(F) \leqslant 2^{m/10}$. *Consequently,* $\mathrm{Mon}(\{G\})$ *contains at least $n! \cdot 2^{nd/3}$ graphs on vertex set $[n]$.*

Proof. Fix a spanning tree T of G of maximum degree at most d. Denote by $\mathcal{F}$ the family of all subgraphs of G containing T. Then,

$$|\mathcal{F}| = 2^{m-n+1} \geqslant 2^{m-\frac{2m}{d}} \geqslant 2^{9m/10},$$

where we used the fact that $n \leqslant 2m/d$ by the assumption on the minimum degree of G, and the assumption $d \geqslant 1000$.

For a fixed graph $F \in \mathcal{F}$, we will now estimate the number $N_{\mathcal{F}}(F)$ of graphs in $\mathcal{F}$ that are isomorphic to F. This number is at most the number $\#\mathrm{Emb}(T \to G) = \#\mathrm{Sub}(T \to G) \cdot \mathrm{aut}(T)$ of embeddings of T into G. The number $\#\mathrm{Sub}(T \to G)$ of subgraphs of G isomorphic to T is at most

$$\binom{m}{n-1} \leqslant \binom{m}{2m/d} \leqslant \left(\frac{ed}{2}\right)^{2m/d} \leqslant 2^{m/20},$$

where the last inequality holds for every $d \geqslant 1000$. Recalling that the maximum degree of T is at most d and using Lemma 11, we conclude that $\mathrm{aut}(T) \leqslant nd^{n-1} \leqslant 2^{2n \log d}$. Consequently,

$$N_{\mathcal{F}}(F) \leqslant \#\mathrm{Emb}(T \to G) \leqslant 2^{m/20} \cdot 2^{2n \log d} \leqslant 2^{m/20 + 2(2m/d)\log d} \leqslant 2^{m/10}, \tag{3}$$

where again we used $n \leqslant 2m/d$ and $d \geqslant 1000$. Note that the bound in (3) holds for any $F \in \mathcal{F}$. Thus, the number of pairwise non-isomorphic graphs in $\mathcal{F}$ is at least

$$\frac{|\mathcal{F}|}{\max_{F \in \mathcal{F}} N_{\mathcal{F}}(F)} \geqslant 2^{9m/10} \cdot 2^{-m/10} = 2^{4m/5}.$$

Furthermore, for any $F \in \mathcal{F}$, we have

$$\mathrm{aut}(F) \leqslant \#\mathrm{Sub}(T \to F) \cdot \mathrm{aut}(T) \leqslant \#\mathrm{Sub}(T \to G) \cdot \mathrm{aut}(T) = \#\mathrm{Emb}(T \to G) \leqslant 2^{m/10},$$

where we used Lemma 10, the fact that $\#\mathrm{Sub}(T \to F) \leqslant \#\mathrm{Sub}(T \to G)$, and (3).

Finally, the number of graphs with vertex set $[n]$ isomorphic to a subgraph of G is at least

$$\sum_{F \in \mathcal{F}} \frac{n!}{\mathrm{aut}(F)} \geqslant 2^{4m/5} \cdot \frac{n!}{2^{m/10}} = n! \cdot 2^{7m/10} \geqslant n! \cdot 2^{dn/3},$$

where in the last inequality we used $m \geqslant dn/2$. ◀

We can now prove the main result of this section.

▶ **Theorem 2.** *Let $\mathcal{X}$ be a monotone small class. Then, $\mathcal{X}$ has bounded degeneracy.*

Proof. To prove the theorem we will show the contrapositive, i.e., if a class $\mathcal{X}$ has unbounded degeneracy, then it is not small. Suppose towards a contradiction that $\mathcal{X}$ has unbounded degeneracy, but there exists a constant c, such that $|\mathcal{X}_n| \leqslant n! \cdot c^n$ holds for every $n \in \mathbb{N}$.

Since $\mathcal{X}$ has unbounded degeneracy and is closed under taking subgraphs, for any positive d, the class $\mathcal{X}$ contains a connected graph with minimum degree at least d. Fix any $d > \max\{1000, 3\log c\}$, and let $G \in \mathcal{X}$ be a connected graph of minimum degree at least d. Let $\mathcal{G} = \mathrm{Mon}(\{G\})$. By Lemmas 12 and 13, for some $n \in \mathbb{N}$ we have

$$|\mathcal{G}_n| \geqslant n! \cdot 2^{nd/3} > n! \cdot 2^{n \log c} = n! \cdot c^n,$$

where the first strict inequality holds due to $d > 3\log c$. Since $\mathcal{G} \subseteq \mathcal{X}$, this is in contradiction with the assumed upper bound on the number of labeled graphs in $\mathcal{X}$. Thus $\mathcal{X}$ is not small. ◀

The previous result is of independent interest, and provides some structural insight on monotone small classes. To show that some property generally holds on small monotone classes, one can now use their degeneracy. We give the first such application of Theorem 2.

▶ **Theorem 3.** *Any monotone small class admits an implicit representation.*

The relevance of Theorem 2 to labeling schemes should be clear from the following folklore bound [25], which we recall for completeness.

▶ **Lemma 14.** *The class of k-degenerate graphs has a $(k+1) \cdot \lceil \log n \rceil$-bit adjacency labeling scheme.*

Proof. For any k-degenerate graph G on n-vertices, we first order vertices so that each vertex has at most k neighbors appearing after it in the ordering. This can be done greedily since each subgraph has a vertex of degree at most k. One can then assign each vertex a label consisting of its place in the order, followed by the places of the at most k neighbor vertices following it in the ordering. ◀

Theorem 3 follows directly from Lemma 14 and Theorem 2.

4 Ingredients for the proof of the lower bound

This section contains many of the components needed to construct the classes used in the proof of the lower bound (Theorem 5). In Section 4.1 we introduce several notions related to subgraph density, which are then applied to random graphs in Section 4.2.

4.1 Good graphs and decent functions

Our first definition describes graphs which do not have overly dense subgraphs.

▶ **Definition 15** (f-good)**.** *Let $f : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ be a function. An n-vertex graph G is* f-good *if the number of edges in any subgraph on k vertices is bounded from above by*

$$\begin{cases} \frac{k \cdot f(k)}{\log k} & \text{if } 2 \leqslant k \leqslant \sqrt{n} \\ k \cdot f(k) & \text{if } \sqrt{n} < k \leqslant n \end{cases}.$$

We observe that f-goodness is a monotone property, i.e., if a graph G is f-good, then so is any of its subgraphs. Indeed, moving the threshold (between the first and the second, more relaxed, upper bound) from $\sqrt{n}$ down to a smaller value may only help in satisfying these bounds.

The next definition gives a class of functions used to describe speeds of monotone classes, where, for such a function $f(n)$, we will consider classes of growth $2^{nf(n)}$.

▶ **Definition 16** (*(δ, C, s)-decent*). *For constants $\delta \in (0,1)$, $C \geqslant 1$ and $s \geqslant 2$, we say that a non-decreasing function $f : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ is* (δ, C, s)-decent *if the following properties hold*
(Moderate-growth): *$\log x \leqslant f(x) \leqslant C \cdot x^{1-\delta}$ holds for every $x \in [s, \infty)$,*
(Sub-multiplicativity): *$f(xy) \leqslant C \cdot f(x) \cdot f(y)$ holds for any $x, y \in [s, \infty)$.*

We say that a function f is *decent* if there exist some constants $\delta \in (0,1)$, $C \geqslant 1$, and $s \geqslant 2$ such that f is (δ, C, s)-decent. For any constant $\kappa \geqslant 1$, the function $f(x) = \kappa \log x$ is decent; this captures factorial growth. We now give some other natural examples of decent functions, due to space constraints a proof can be found in the full version [11].

▶ **Lemma 17.** *For any fixed $\alpha > 0, \beta \geqslant 1, \gamma \geqslant 1$ and $d \in (0,1)$, the following are decent:*
(i) $f(x) = \alpha x^d$,
(ii) $f(x) = \exp\left(\alpha \cdot \ln^d x\right)$,
(iii) $f(x) = \exp\left(\beta \cdot \ln^\gamma(\log x)\right)$,
(iv) $f(x) = \beta \cdot g(x)$, *where $g(x)$ is decent,*
(v) $f(x) = g(x) \cdot h(x)$, *where $g(x), h(x)$ are decent and $g(x) \cdot h(x)$ is moderately-growing.*

4.2 Growth of the number of edges in subgraphs of random graphs

The aim of this Section is to show that there are many graphs which are suitable for building the classes we need to prove Theorem 5. We will achieve this using random graphs, where $G(n,p)$ denotes the distribution on n-vertex graphs where each edge is included independently with probability p, see (for example) [19]. Our main result shows that random graphs are suitable with high probably.

▶ **Theorem 18.** *Let $f : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ be (δ, C, s)-decent for some constants $\delta \in (0,1)$, $C \geqslant 1$, and $s \geqslant 2$. Then, for any fixed $\gamma > 1$, there exists $c := c(\delta, C, s, \gamma) > 0$ such that, for large n,*

$$\mathbb{P}\left[\, G(n, \gamma f(n)/n) \text{ is not } (cf)\text{-good} \,\right] \leqslant n^{-2}.$$

To prove this we will utilize the following version of the Chernoff bound (see [5, Theorem A.1.15]), where $\mathrm{Bin}(N,p)$ denotes the binomial distribution with parameters N and p.

▶ **Lemma 19** (Chernoff bound). *Let $\xi \sim \mathrm{Bin}(N,p)$, $\mu = Np$, and $a, t > 0$. Then,*

$$\mathbb{P}(\xi > (1+a)\mu) \leqslant \left(\frac{e^a}{(1+a)^{1+a}}\right)^\mu \leqslant \exp\left(-(1+a)\mu \cdot \ln\frac{1+a}{e}\right).$$

Proof of Theorem 18. Let $p := p(n) = \gamma f(n)/n$, and let c_1, c_2 be sufficiently large constants (depending on γ) fixed later. Let $\mathcal{E}_{1,k}$ (respectively $\mathcal{E}_{2,k}$) be the event that there are no subgraphs of size k with more than $c_1 k f(k)/\log k$ edges (respectively $c_2 k f(k)$ edges). Observe that if $c = \max\{c_1, c_2, \binom{s}{2}\}$, then

$$\{G(n,p) \text{ is not } (cf)\text{-good}\} \subseteq \left(\bigcup_{k=s}^{\lfloor\sqrt{n}\rfloor} \neg\mathcal{E}_{1,k}\right) \cup \left(\bigcup_{k=\lfloor\sqrt{n}\rfloor+1}^{n} \neg\mathcal{E}_{2,k}\right). \tag{4}$$

Let k denote the number of vertices in a subgraph, and thus $\xi \sim \mathrm{Bin}\left(\binom{k}{2}, p\right)$ denotes the number of edges in a given k-vertex subgraph. The expectation of ξ is

$$\mu := \binom{k}{2}p = \frac{\gamma}{2} \cdot \frac{k(k-1)f(n)}{n}.$$

On the other hand, the number of ways to select a k-vertex subgraph is

$$\binom{n}{k} \leqslant \left(\frac{en}{k}\right)^k = \exp\left(k \ln \frac{n}{k} + k\right) \leqslant \exp\left(2k \ln n\right). \tag{5}$$

Our strategy will be to bound the probability of the events on the right-hand side of (4) using the union and Chernoff bounds.

We begin by considering events of the form $\mathcal{E}_{2,k}$ and thus can assume that $\lfloor\sqrt{n}\rfloor+1 \leqslant k \leqslant n$. Observe that since f is sub-multiplicative, non-decreasing, and moderately-growing, we have

$$\frac{f(k)}{f(n)} = \frac{f(k)}{f\left(\frac{n}{k} \cdot k\right)} \geqslant \frac{f(k)}{C \cdot f(\frac{n}{k}) \cdot f(k)} \geqslant \frac{f(k)}{C \cdot f(\frac{sn}{k}) \cdot f(k)} \geqslant \frac{f(k)}{C^2 \cdot (\frac{sn}{k})^{1-\delta} \cdot f(k)} \geqslant \frac{k}{C^2 s \cdot n}. \tag{6}$$

If we now fix

$$c_2 = C^2 s \cdot e^2 \cdot \gamma > 6, \tag{7}$$

then by (6) we have

$$\frac{2c_2 n f(k)}{e\gamma(k-1)f(n)} = \frac{2C^2 se \cdot nf(k)}{(k-1)f(n)} \geqslant \frac{2ek}{k-1} > e. \tag{8}$$

So, applying Chernoff bound (Lemma 19) with $1 + a = \frac{c_2 kf(k)}{\mu} = \frac{2c_2 nf(k)}{\gamma(k-1)f(n)}$ gives

$$\begin{aligned}
\mathbb{P}(\xi > c_2 kf(k)) &\leqslant \exp\left(-(1+a)\mu \cdot \ln \frac{1+a}{e}\right) \\
&= \exp\left(-c_2 kf(k) \cdot \ln \frac{2c_2 nf(k)}{e\gamma(k-1)f(n)}\right) \\
&\overset{(8)}{\leqslant} \exp\left(-c_2 kf(k)\right) \\
&\overset{(7)}{\leqslant} \exp\left(-6kf(k)\right).
\end{aligned} \tag{9}$$

Thus, by (5), (9), the union bound, and as $f(k) \geqslant \log k > \ln k$, we have

$$\begin{aligned}
\mathbb{P}\left(\bigcup_{k=\lfloor\sqrt{n}\rfloor+1}^{n} \neg\mathcal{E}_{2,k}\right) &\leqslant \sum_{k=\lfloor\sqrt{n}\rfloor+1}^{n} \exp\left(2k \ln n\right) \cdot \exp\left(-6kf(k)\right) \leqslant \sum_{k=\lfloor\sqrt{n}\rfloor+1}^{n} k^{-k} \\
&\leqslant \exp(-\sqrt{n}).
\end{aligned} \tag{10}$$

We now treat events of the form $\mathcal{E}_{1,k}$, and thus we can assume that $s \leqslant k \leqslant \lfloor\sqrt{n}\rfloor$. Observe that for any fixed constant $d > 0$ and sufficiently large n we have $\frac{n^{2/3}}{k(\log k)^d} \geqslant s$ as $k \leqslant \sqrt{n}$. Thus, by sub-multiplicativity, and moderate-growth we have

$$\begin{aligned}
f\left(\frac{n^{2/3}}{(\log k)^d}\right) &= f\left(\frac{n^{2/3}}{k(\log k)^d} \cdot k\right) \\
&\leqslant C \cdot f\left(\frac{n^{2/3}}{k(\log k)^d}\right) \cdot f\left(k\right) \\
&\leqslant C^2 \cdot \left(\frac{n^{2/3}}{k(\log k)^d}\right)^{1-\delta} \cdot f(k) \\
&\leqslant C^2 \cdot \frac{n^{2/3}}{k(\log k)^d} \cdot f(k).
\end{aligned}$$

Similarly, by sub-multiplicativity and moderate-growth, we have

$$\begin{aligned} f(n) &= f\left(\frac{n^{2/3}}{(\log k)^d}\cdot n^{1/3}(\log k)^d\right) \\ &\leqslant C\cdot f\left(\frac{n^{2/3}}{(\log k)^d}\right)\cdot f\left(n^{1/3}(\log k)^d\right) \\ &\leqslant C^2\cdot f\left(\frac{n^{2/3}}{(\log k)^d}\right)\cdot n^{(1-\delta)/3}(\log k)^{(1-\delta)\cdot d}. \end{aligned}$$

If we set $d = 1/\delta > 0$ then the two bounds above give

$$\frac{f(k)}{f(n)} \geqslant \frac{f\left(\frac{n^{2/3}}{(\log k)^d}\right)\cdot\frac{k(\log k)^d}{C^2 n^{2/3}}}{C^2\cdot f\left(\frac{n^{2/3}}{(\log k)^d}\right)\cdot n^{(1-\delta)/3}(\log k)^{(1-\delta)\cdot d}} = \frac{k(\log k)^{\delta d}}{C^4 n^{1-\delta/3}} = \frac{k\log k}{C^4 n}\cdot n^{\delta/3}. \tag{11}$$

Foreseeing the need for the constant 15 later on, we now set

$$c_1 = e\cdot 15\cdot C^4\gamma/\delta. \tag{12}$$

We now set $1+a := \frac{c_1 k f(k)}{\mu\cdot\log k}$, which by (11) satisfies

$$1+a = \frac{c_1 k f(k)}{\mu\cdot\log k} = \frac{2c_1 n f(k)}{\gamma(k-1)f(n)\log k} \geqslant \frac{2c_1 k}{\gamma(k-1)C^4}\cdot n^{\delta/3} > e\cdot n^{\delta/3}. \tag{13}$$

As before, Chernoff bound (Lemma 19) with this $1+a$ gives

$$\mathbb{P}\left(\xi > \frac{c_1 k f(k)}{\log k}\right) \leqslant \exp\left(-\frac{c_1 k f(k)}{\log k}\cdot\ln\frac{1+a}{e}\right) \overset{(13)}{\leqslant} \exp\left(-\frac{c_1 k f(k)}{\log k}\cdot\frac{\delta}{3}\ln n\right). \tag{14}$$

Recall that the bound $f(k)\geqslant\log k$ holds by moderate-growth. Applying this to (14) yields

$$\mathbb{P}\left(\xi > \frac{c_1 k f(k)}{\log k}\right) \leqslant \exp\left(-c_1 k\cdot\frac{\delta}{3}\ln n\right) \overset{(12)}{\leqslant} \exp\left(-5k\ln n\right). \tag{15}$$

Thus, by (5), (15), and the union bound,

$$\mathbb{P}\left(\bigcup_{k=s}^{\lfloor\sqrt{n}\rfloor}\neg\mathcal{E}_{1,k}\right) \leqslant \sum_{k=s}^{\lfloor\sqrt{n}\rfloor}\exp\left(2k\ln n\right)\cdot\exp\left(-5k\ln n\right) \leqslant \sqrt{n}\cdot n^{-3s} \leqslant n^{-5}. \tag{16}$$

The result follows by taking $c = \max\{c_1, c_2, \binom{s}{2}\}$, (4), and the union bound over (10) and (16). ◀

We now use Theorem 18 to prove Lemma 21, which bounds the number of cf-good graphs from below. To prove Lemma 21, it is convenient to switch to an alternative model of random graphs with a fixed number of edges. We let $G(n,m)$ to denote the uniform distribution on n-vertex graphs with m edges. The following lemma allows us to transfer results from one graph model to another.

▶ **Lemma 20.** *Let $\mathcal{P}$ be any graph property (i.e., graph class) and $0\leqslant p\leqslant 1$ satisfy $p\binom{n}{2}\to\infty$ and $\binom{n}{2}-p\binom{n}{2}\to\infty$ and $m = \left\lceil p\binom{n}{2}\right\rceil$. Then, for $G_n\sim G(n,m)$ and $G'_n\sim G(n,p)$, we have*

$$\mathbb{P}\left[G_n\in\mathcal{P}\right] \leqslant 10\sqrt{m}\cdot\mathbb{P}\left[G'_n\in\mathcal{P}\right].$$

Lemma 20 follows by a very minor adaption of [19, Lemma 3.2], the only difference is a ceiling in the number of edges, which makes no difference in the proof.

▶ **Lemma 21.** *Let $f : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ be (δ, C, s)-decent for some constants $\delta \in (0,1)$, $C \geqslant 1$, and $s \geqslant 2$. Then, for any fixed $\gamma > 1$, there exists some $c := c(\gamma, \delta, C, s) > 0$ such that for every $n \in \mathbb{N}$ there are at least $2^{(\gamma\delta/2 - o(1)) \cdot nf(n)\log n}$ many unlabeled (cf)-good n-vertex graphs.*

Proof. Let $m := \left\lceil \frac{\gamma(n-1)f(n)}{2} \right\rceil$ and $G_n \sim G\big(n, m\big)$. Observe that by Theorem 18 and Lemma 20, there exists some fixed $c > 0$ such that for sufficiently large n

$$\mathbb{P}\left[\, G_n \text{ is } (cf)\text{-good} \,\right] \geqslant 1 - 10\sqrt{\left\lceil \tfrac{\gamma(n-1)f(n)}{2} \right\rceil} \cdot n^{-2} = 1 - o(1). \tag{17}$$

The number of labeled graphs in the support of $G\big(n, m\big)$ is

$$\binom{\binom{n}{2}}{m} = \binom{\binom{n}{2}}{\left\lceil \frac{\gamma(n-1)f(n)}{2} \right\rceil} \geqslant \left(\frac{n}{\gamma f(n)} \right)^{\frac{\gamma(n-1)f(n)}{2}} = 2^{\frac{\gamma}{2} \cdot (n-1)f(n) \cdot (\log n - \log(\gamma f(n)))}.$$

By (17), a $1 - o(1)$ fraction of these labeled graphs are (cf)-good. Furthermore, there are at most $n! \leqslant n^n$ labelings of a given unlabeled graph. Thus, the number of unlabeled n-vertex (cf)-good graphs is bounded from below by

$$\begin{aligned}(1 - o(1)) \cdot \frac{1}{n^n} \cdot 2^{\frac{\gamma}{2} \cdot (n-1)f(n) \cdot (\log n - \log(\gamma f(n)))} &= 2^{\frac{\gamma}{2} \cdot nf(n) \cdot (\log n - \log(f(n)) - O(1))} \\ &\geqslant 2^{\frac{\gamma}{2} \cdot nf(n) \cdot (\log n - (1-\delta)\log(n) - O(1))} \\ &= 2^{(\delta\gamma/2 - o(1)) \cdot nf(n)\log n},\end{aligned}$$

as claimed, since $\log n \leqslant f(n) \leqslant Cn^{1-\delta}$ by moderate-growth. ◀

5 Tight bounds on labeling schemes for monotone classes

We begin in Section 5.1 with a lemma which is useful for bounding the speed when constructing monotone classes with no implicit representation. This is then used to prove our lower bound in Section 5.2. Finally, in Section 5.3 we give a matching upper bound on labeling schemes for monotone classes, this follows from [25] and included mainly for completeness.

5.1 Construction of monotone classes

We begin with a lemma showing that, for a decent function f, we can create monotone classes from the union of many f-good graphs and still maintain control over the speed. The proof follows the broad idea of [23, Claim 3.1].

▶ **Lemma 22.** *Let $f : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ be (δ, C, s)-decent for some constants $\delta \in (0,1)$, $C \geqslant 1$, and $s \geqslant 2$. Let $c > 0$ be a constant, and, for every $n \in \mathbb{N}$, let M_n be any set of (cf)-good unlabeled n-vertex graphs satisfying $|M_n| \leqslant \left\lceil 2^{\sqrt{nf(n)}} \right\rceil$. Then the speed of $\mathcal{X} := \mathrm{Mon}(\cup_{n\in\mathbb{N}} M_n)$ is $2^{O(nf(n))}$.*

Proof. Let $\mathcal{Y} := \mathrm{Her}(\cup_{n\in\mathbb{N}} M_n)$. Note that $\mathcal{X} = \mathrm{Mon}(\mathcal{Y})$. We first estimate the speed of $\mathcal{Y}$. For an n-vertex graph $G \in \mathcal{Y}$, let N be the smallest integer such that G is an induced subgraph of a graph $H \in M_N$. We split the proof over two cases: (i): $N \geqslant n^2$, and (ii): $N < n^2$.

Case (i): Since H is a (cf)-good N-vertex graph and G is its n-vertex induced subgraph, where $n \leqslant \sqrt{N}$, it follows from Definition 15 that G must have at most $g(n) := cnf(n)/\log n$ many edges. The number of such graphs is at most

$$\binom{\binom{n}{2}}{g(n)} \leqslant \left(\frac{n^2 e}{g(n)}\right)^{g(n)} = 2^{g(n)\cdot\log\frac{n^2 e}{g(n)}} = 2^{c\frac{nf(n)}{\log n}\cdot\log\frac{n^2 e}{g(n)}} = 2^{O(nf(n))},$$

and so $\mathcal{Y}$ contains $2^{O(nf(n))}$ many n-vertex labeled graphs each of which is an induced subgraph of a graph in M_N for some N with $n \leqslant \sqrt{N}$.

Case (ii): For this case, we simply use the fact that any $H \in M_N$ has at most N^n many n-vertex induced subgraphs. Thus, the number of n-vertex labeled graphs in $\mathcal{Y}$ each of which is an induced subgraph of a graph in M_N for some N with $N < n^2$ is bounded from above by

$$\begin{aligned} n! \cdot \sum_{N=n}^{n^2} N^n \cdot |M_N| &\leqslant n! \cdot \sum_{N=n}^{n^2} N^n \cdot \left\lceil 2^{\sqrt{Nf(N)}} \right\rceil \\ &\leqslant n! \cdot n^2 \cdot (n^2)^n \cdot \left\lceil 2^{\sqrt{n^2 f(n^2)}} \right\rceil \\ &\leqslant 2^{O(n\log n)} \cdot \left\lceil 2^{\sqrt{C}nf(n)} \right\rceil \\ &= 2^{O(nf(n))}, \end{aligned}$$

where in the last inequality we used sub-multiplicativity of f, and in the final equality we used the fact that $f(x) \geqslant \log x$.

Thus, $|\mathcal{Y}_n| = 2^{O(nf(n))}$. Now, since every n-vertex labeled graph in $\mathcal{X}$ is a subgraph of an n-vertex labeled graph in $\mathcal{Y}$, and, due to (cf)-goodness, every graph in $\mathcal{Y}_n$ has at most $2^{cnf(n)}$ n-vertex subgraphs, we conclude that $|\mathcal{X}_n| \leqslant |\mathcal{Y}_n| \cdot 2^{cnf(n)} = 2^{O(nf(n))}$. ◀

5.2 Lower bound

We can now show the main result of the paper, which we recall for convenience.

▶ **Theorem 5.** *Let $f : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ be a decent function. Then, there exists a monotone graph class $\mathcal{X}$ with speed $|\mathcal{X}_n| = 2^{O(nf(n))}$ that does not admit a universal graph of size at most $2^{f(n)\log n}$. Equivalently, $\mathcal{X}$ admits no adjacency labeling scheme of size at most $f(n)\log n$.*

Proof. By assumption $f : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ is (δ, C, s)-decent for some constants $\delta \in (0,1)$, $C \geqslant 1$, and $s \geqslant 2$. We will construct a monotone class (via the probabilistic method) with the speed $2^{O(nf(n))}$ that does not admit a universal graph of size $u_n := 2^{f(n)\log n}$. Fix $\gamma := 4/\delta > 1$ and let $c := c(\gamma, \delta, C, s) > 0$ be the satisfying constant from Theorem 18 corresponding to this choice of γ. Let $k_n := \left\lceil 2^{\sqrt{nf(n)}} \right\rceil$.

The number of distinct u_n-vertex graphs is at most $2^{u_n^2}$ and the number of n-vertex induced subgraphs of a fixed u_n-vertex graph is at most $\binom{u_n}{n}$. Hence the number of collections of k_n graphs on n vertices that are induced subgraphs of a u_n-vertex (universal) graph is at most

$$2^{u_n^2} \cdot \binom{\binom{u_n}{n}}{k_n} \leqslant 2^{u_n^2} \cdot u_n^{k_n \cdot n}. \tag{18}$$

On the other hand, from Lemma 21, the number of different collections of n-vertex (cf)-good graphs of cardinality k_n is at least

$$\binom{2^{(\gamma\delta/2-o(1))\cdot nf(n)\log n}}{k_n} \geqslant \left(\frac{2^{(\gamma\delta/2-o(1))\cdot nf(n)\log n}}{k_n}\right)^{k_n} = 2^{k_n\cdot(\gamma\delta/2-o(1))\cdot nf(n)\log n}, \tag{19}$$

as $\log k_n = O(\sqrt{nf(n)}) = o(nf(n)\log n)$. By taking logarithms, we can see that for sufficiently large n the upper bound (18) is smaller than the lower bound (19). In particular, taking the logarithm of (18) gives

$$\begin{aligned}\log\left(2^{u_n^2}\cdot u_n^{k_n\cdot n}\right) &= u_n^2 + k_n\cdot n\log u_n\\ &= 2^{2f(n)\log n} + k_n\cdot nf(n)\log n\\ &= (1+o(1))\cdot k_n\cdot nf(n)\log n,\end{aligned}$$

as $k_n := \left\lceil 2^{\sqrt{nf(n)}}\right\rceil = \omega(2^{2f(n)\log n})$. However, since $\gamma = 4/\delta$, the logarithm of (19) is

$$\begin{aligned}\log\left(2^{k_n\cdot(\gamma\delta/2-o(1))\cdot nf(n)\log n}\right) &= k_n\cdot(\gamma\delta/2-o(1))\cdot nf(n)\log n\\ &= (2-o(1))\cdot k_n\cdot nf(n)\log n.\end{aligned}$$

Thus, for any sufficiently large n, there exists a collection M_n of k_n (cf)-good n-vertex graphs that are not representable by any universal graph of size at most $u_n = 2^{f(n)\log n}$. Consequently, by Lemma 22, the speed of $\mathcal{X} := \mathrm{Mon}(\cup_n M_n)$ is $|\mathcal{X}_n| = 2^{O(nf(n))}$ and $\mathcal{X}$ does not admit a universal graph of size at most $2^{f(n)\log n}$. ◀

5.3 Upper bound

In this section we prove the following upper bound on labeling schemes for monotone classes.

▶ **Proposition 4.** *Let $f : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ be a non-decreasing function. Then, any monotone class of graphs $\mathcal{X}$ with speed $|\mathcal{X}_n| = 2^{O(nf(n))}$ admits an adjacency labeling scheme of size $O(f(n)\log n)$.*

Proof. Let $\mathcal{X}$ be a monotone class with at most $2^{Cnf(n)}$ labeled n-vertex graphs for every n. If an n-vertex graph $G \in \mathcal{X}$ has m edges, then $\mathcal{X}$ contains at least 2^m labeled n-vertex graphs, as every subgraph of G also belongs to $\mathcal{X}$ due to monotonicity.

This implies that every n-vertex graph G in $\mathcal{X}$ contains at most $Cnf(n)$ edges, and hence, has a vertex of degree at most $2Cf(n)$. Due to monotonicity of f, the same is true for every subgraph of G. Indeed, if H is a k-vertex subgraph of G, then, since H belongs to $\mathcal{X}$, the number of edges in H is at most $Ckf(k) \leqslant Ckf(n)$, and therefore H has a vertex of degree at most $2Cf(n)$. Thus, every n-vertex graph in $\mathcal{X}$ is $2Cf(n)$-degenerate, and Lemma 14 implies that $\mathcal{X}$ admits a $(2Cf(n)+1)\cdot\lceil\log n\rceil$-bit labeling scheme. ◀

5.4 Complexity of monotone classes

The following result shows that monotone classes are complex in the sense that they cannot be "described" by even a countable number of classes of a slightly larger speed. The proof of this theorem follows the exact same idea as [14, Lemma 2.4], also see [12, Theorem 1.2] for the proof of a similar theorem in the context of small classes.

▶ **Theorem 7.** *Let $f : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ be any decent function, and $\mathbb{X}$ be any countable set of graph classes, each with speed at most $2^{nf(n)\log n}$. Then, there exists a monotone graph class $\mathcal{X}$ of speed $2^{O(nf(n))}$ such that there does not exist a $\mathcal{D} \in \mathbb{X}$ with $\mathcal{X} \subseteq \mathcal{D}$.*

Proof. Let $\mathcal{X}$ be a monotone class with at most $2^{Cnf(n)}$ labeled n-vertex graphs for every n. If an n-vertex graph $G \in \mathcal{X}$ has m edges, then $\mathcal{X}$ contains at least 2^m labeled n-vertex graphs, as every subgraph of G also belongs to $\mathcal{X}$ due to monotonicity.

This implies that every n-vertex graph G in $\mathcal{X}$ contains at most $Cnf(n)$ edges, and hence, has a vertex of degree at most $2Cf(n)$. Due to monotonicity of f, the same is true for every subgraph of G. Indeed, if H is a k-vertex subgraph of G, then, since H belongs to $\mathcal{X}$, the number of edges in H is at most $Ckf(k) \leqslant Ckf(n)$, and therefore H has a vertex of degree at most $2Cf(n)$. Thus, every n-vertex graph in $\mathcal{X}$ is $2Cf(n)$-degenerate, and Lemma 14 implies that $\mathcal{X}$ admits a $(2Cf(n)+1) \cdot \lceil \log n \rceil$-bit labeling scheme. ◀

References

1. Vladimir E. Alekseev. Range of values of entropy of hereditary classes of graphs. *Diskretnaya Matematika*, 4(2):148–157, 1992.
2. Vladimir E. Alekseev. On lower layers of a lattice of hereditary classes of graphs. *Diskretnyi Analiz i Issledovanie Operatsii*, 4(1):3–12, 1997.
3. Noga Alon. Asymptotically optimal induced universal graphs. *Geom. Funct. Anal.*, 27(1):1–32, 2017. doi:10.1007/s00039-017-0396-9.
4. Noga Alon. Implicit representation of sparse hereditary families. *Discret. Comput. Geom.*, to appear, 2023. doi:10.1007/s00454-023-00521-0.
5. Noga Alon and Joel H. Spencer. *The Probabilistic Method, Third Edition*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 2008.
6. Stephen Alstrup, Haim Kaplan, Mikkel Thorup, and Uri Zwick. Adjacency labeling schemes and induced-universal graphs. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 625–634, 2015.
7. Avah Banerjee. An adjacency labeling scheme based on a decomposition of trees into caterpillars. In *Combinatorial Algorithms – 33rd International Workshop, IWOCA 2022*, volume 13270 of *Lecture Notes in Computer Science*, pages 114–127. Springer, 2022.
8. Robin L. Blankenship. *Book embeddings of graphs*. Louisiana State University and Agricultural & Mechanical College, 2003.
9. Béla Bollobás and Andrew Thomason. Projections of bodies and hereditary properties of hypergraphs. *Bulletin of the London Mathematical Society*, 27(5):417–424, 1995.
10. Marthe Bonamy, Louis Esperet, Carla Groenland, and Alex Scott. Optimal labelling schemes for adjacency, comparability, and reachability. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1109–1117, 2021.
11. Édouard Bonnet, Julien Duron, John Sylvester, Viktor Zamaraev, and Maksim Zhukovskii. Tight bounds on adjacency labels for monotone graph classes. *arXiv:2310.20522*, 2023.
12. Édouard Bonnet, Julien Duron, John Sylvester, Viktor Zamaraev, and Maksim Zhukovskii. Small but unwieldy: A lower bound on adjacency labels for small classes. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2024)*, 2024.
13. Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. *Combinatorial Theory, 2 (2)*, 2022.
14. Maurice Chandoo. Logical labeling schemes. *Discrete Mathematics*, 346(10):113565, 2023.
15. Bruno Courcelle and Rémi Vanicat. Query efficient implementation of graphs of bounded clique-width. *Discrete Applied Mathematics*, 131(1):129–150, 2003.
16. Louis Esperet, Nathaniel Harms, and Andrey Kupavskii. Sketching distances in monotone graph classes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
17. Louis Esperet, Nathaniel Harms, and Viktor Zamaraev. Optimal adjacency labels for subgraphs of cartesian products. In *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023*, volume 261 of *LIPIcs*, pages 57:1–57:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.57.

18 Pierre Fraigniaud and Amos Korman. On randomized representations of graphs using short labels. In *SPAA 2009: Proceedings of the 21st Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 131–137. ACM, 2009. doi:10.1145/1583991.1584031.

19 Alan Frieze and Michał Karoński. *Random Graphs and Networks: A First Course*. Cambridge University Press, 2023. doi:10.1017/9781009260268.

20 Cyril Gavoille and Arnaud Labourel. Shorter implicit representation for planar graphs and bounded treewidth graphs. In *15th Annual European Symposium on Algorithms, ESA 2007*, volume 4698 of *Lecture Notes in Computer Science*, pages 582–593. Springer, 2007.

21 Nathaniel Harms. Universal communication, universal graphs, and graph labeling. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

22 Nathaniel Harms, Sebastian Wild, and Viktor Zamaraev. Randomized communication and implicit graph representations. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2022)*, pages 1220–1233, 2022.

23 Hamed Hatami and Pooya Hatami. The implicit graph conjecture is false. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS 2022)*, pages 1134–1137. IEEE, 2022.

24 Sampath Kannan, Moni Naor, and Steven Rudich. Implicit representation of graphs. In *Proceedings of the twentieth annual ACM symposium on Theory of computing (STOC 1988)*, pages 334–343, 1988.

25 Sampath Kannan, Moni Naor, and Steven Rudich. Implicit representation of graphs. *SIAM Journal on Discrete Mathematics*, 5(4):596–603, 1992.

26 Ilia Krasikov, Arieh Lev, and Bhalchandra D Thatte. Upper bounds on the automorphism group of a graph0. *Discrete Mathematics*, 256(1-2):489–493, 2002.

27 John W Moon. On minimal n-universal graphs. *Glasgow Mathematical Journal*, 7(1):32–33, 1965.

28 John H. Muller. *Local Structure in Graph Classes*. PhD thesis, Georgia Institute of Technology, 1988.

29 Serguei Norine, Paul Seymour, Robin Thomas, and Paul Wollan. Proper minor-closed families are small. *Journal of Combinatorial Theory, Series B*, 96(5):754–757, 2006.

30 Edward R Scheinerman. Local representations using very short labels. *Discrete mathematics*, 203(1-3):287–290, 1999.

31 Edward R Scheinerman and Jennifer Zito. On the size of hereditary classes of graphs. *Journal of Combinatorial Theory, Series B*, 61(1):16–39, 1994.

32 Jeremy P Spinrad. *Efficient Graph Representations*, volume 19. Fields Institute monographs. American Mathematical Soc., 2003.

Two Choices Are Enough for P-LCPs, USOs, and Colorful Tangents

Michaela Borzechowski
Department of Mathematics and Computer Science, Freie Universität Berlin, Germany

John Fearnley
Department of Computer Science, University of Liverpool, UK

Spencer Gordon
Department of Computer Science, University of Liverpool, UK

Rahul Savani
The Alan Turing Institute, London, UK
Department of Computer Science, University of Liverpool, UK

Patrick Schnider
Department of Computer Science, ETH Zürich, Switzerland

Simon Weber
Department of Computer Science, ETH Zürich, Switzerland

Abstract

We provide polynomial-time reductions between three search problems from three distinct areas: the P-matrix linear complementarity problem (P-LCP), finding the sink of a unique sink orientation (USO), and a variant of the α-Ham Sandwich problem. For all three settings, we show that "two choices are enough", meaning that the general non-binary version of the problem can be reduced in polynomial time to the binary version. This specifically means that generalized P-LCPs are equivalent to P-LCPs, and grid USOs are equivalent to cube USOs. These results are obtained by showing that both the P-LCP and our α-Ham Sandwich variant are equivalent to a new problem we introduce, P-Lin-Bellman. This problem can be seen as a new tool for formulating problems as P-LCPs.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Continuous optimization; Mathematics of computing → Combinatoric problems; Theory of computation → Computational geometry

Keywords and phrases P-LCP, Unique Sink Orientation, α-Ham Sandwich, search complexity, TFNP, UEOPL

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.32

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2402.07683`

Funding *Michaela Borzechowski*: DFG within GRK 2434 *Facets of Complexity*
John Fearnley: EPSRC Grant EP/W014750/1
Spencer Gordon: EPSRC Grant EP/W014750/1.
Rahul Savani: EPSRC Grant EP/W014750/1
Simon Weber: Swiss National Science Foundation under project no. 204320

Acknowledgements We wish to thank Bernd Gärtner for introducing the authors to each other.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 32; pp. 32:1–32:18

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

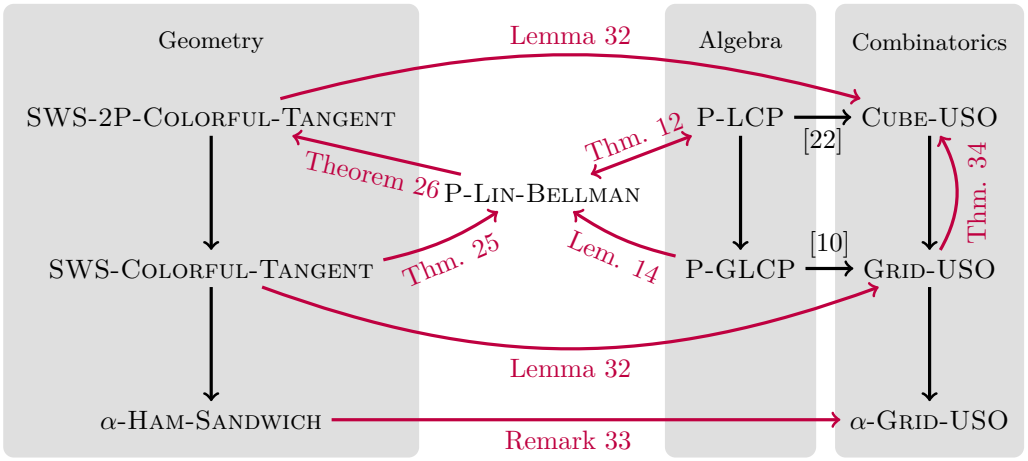


Figure 1 Red: Reductions we show in this paper. Black: Existing reductions and trivial inclusions.

In this paper we study three search problems from three distinct areas: the problem of solving a P-matrix linear complementarity problem (P-LCP), an algebraic problem, the problem of finding the sink of a unique sink orientation (USO), a combinatorial problem, and a variant of the α-Ham Sandwich problem, a geometric problem. Our results are a suite of reductions between these problems, which are shown in Figure 1.

There are two main themes for these reductions.

- **Two choices are enough.** For all three settings, the problems can be restricted to variants in which all choices are binary. In all three cases we show that the general non-binary problem can be reduced in polynomial time to the binary version.
- **A new tool for working with P-LCPs.** We introduce the P-Lin-Bellman problem, which serves as a crucial intermediate problem for our reductions with P-LCPs, and provides a new tool for showing equivalence to the P-LCP problem.

We now describe each of the three problems and our results.

P-Matrix LCPs. In the *Linear Complementarity Problem* (LCP), we are given an $n \times n$ matrix M and an n-dimensional vector q, and we are asked to find two n-dimensional vectors w, z such that $w = Mz + q$, both w and z are non-negative, and w and z satisfy the *complementarity* condition of $w^T z = 0$. In general, there is no guarantee that an LCP has a solution. However, if M is promised to be a P-matrix, that is, all its principal minors are positive, then the LCP problem always has a unique solution for every possible q [19], and we call this the P-LCP problem.

The P-LCP problem is important because many optimization problems reduce to it, for example, *Linear Programming* [12, 16] and *Strictly Convex Quadratic Programming* [5, 20], and solving a *Simple Stochastic Game (SSG)* [11, 23]. However, the complexity status of the P-LCP remains a major open question. The P-LCP problem is not known to be polynomial-time solvable, but NP-hardness (in the sense that an oracle for it could be used to solve SAT in polynomial time) would imply NP = co-NP [18].

The P-LCP problem naturally encodes problems that have *two choices*. This property arises from the complementarity condition, where for each index i, one must choose either $w_i = 0$ or $z_i = 0$. Thus the problem can be seen as making one of two choices for each of the

n possible dimensions of w and z. For example, this means that the direct encoding of a Simple Stochastic Game as a P-LCP [16] only works for *binary* games, in which each vertex has exactly two outgoing edges, and for each vertex the choice between the two outgoing edges is encoded by choosing between w_i and z_i.

To directly encode a non-binary SSG one must instead turn to *generalized* LCPs, which allow for more than two choices in the complementarity condition [23]. Generalized LCPs, which were defined by Habetler and Szanc [15], also have a P-matrix version, which we will refer to as P-GLCP.

Two choices are enough for P-LCPs. Our first main result is to show that P-GLCP and P-LCP are polynomial-time equivalent problems. Every P-LCP is a P-GLCP by definition, so our contribution is to give a polynomial-time reduction from P-GLCP to P-LCP, meaning that any problem that can be formulated as a P-GLCP can also be efficiently recast as a P-LCP. Such a result was already claimed in 1995, but later a counterexample to a crucial step in the proof was found by the same author [6, 7].

To show this result we draw inspiration from the world of infinite games. SSGs are a special case of *stochastic discounted games* (SDGs), which are known to be reducible in polynomial-time to the P-LCP problem [16, 17]. This reduction first writes down a system of *Bellman equations* for the game, which are a system of linear equations using min and max operations. These equations are then formulated as a P-LCP using the complementarity condition to encode the min and max operations.

We introduce a generalization of the Bellman equations for SDGs, which we call P-Lin-Bellman. We show that the existing reduction from SDGs to P-LCP continues to work for P-Lin-Bellman, and we show that we can polynomial-time reduce P-LCP to P-Lin-Bellman. Thus, we obtain a Bellman-equation type problem that is equivalent to P-LCP.

Then we use P-Lin-Bellman as a tool to reduce P-GLCP to P-LCP. Specifically we show that a P-GLCP can be reduced to P-Lin-Bellman. Here our use of P-Lin-Bellman as an intermediary shows its usefulness: while it is not at all clear how one can reduce P-GLCP to P-LCP directly, when both problems are formulated as P-Lin-Bellman instances, their equivalence essentially boils down to the fact that $\max(a, b, c) = \max(a, \max(b, c))$.

Unique Sink Orientations. A Unique Sink Orientation (USO) is an orientation of the n-dimensional hypercube such that every subcube contains a unique sink. An example of an USO can be found in Figure 2. The goal is to find the unique sink of the entire hypercube. USOs were introduced by Stickney and Watson [22] as a combinatorial abstraction of the candidate solutions of the P-LCP and have been studied ever since Szabó and Welzl formally defined them in 2001 [24].

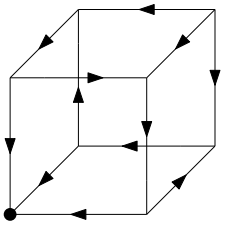


Figure 2 A 3-dimensional USO. The marked vertex denotes its unique global sink.

USOs have received much attention because many problems are known to reduce to them. This includes linear programming and more generally convex quadratic programming, simple stochastic games, and finding the smallest enclosing ball of given points.

As with P-LCPs, USOs naturally capture problems in which there are two choices. Each dimension of the cube allows us to pick between one of two faces, and thus each vertex of the cube is the result of making n binary choices. To remove this restriction, prior work has studied unique sink orientations of products of complete graphs [10], which the literature somewhat confusingly calls *grid USOs*. For example, as with P-LCP, the direct reduction from SSGs to USOs only works for binary games, while a direct reduction from non-binary games yields a grid USO.

Two choices are enough for USOs. Our second main result is to show that grid USOs and cube USOs are polynomial-time equivalent problems. Every cube USO is a grid USO by definition, so our contribution is to provide a polynomial-time reduction from grid USOs to cube USOs. Despite many researchers suspecting that grid USOs are a strict generalization of cube USOs, our result shows that at least in the promise setting, the two problems are computationally equivalent. For the reduction we embed a k-regular grid into a (also k-regular) k-cube. The main challenge to overcome is that the k-cube contains many more vertices and significantly more edges. These edges need to be oriented in such a way to be consistent with the orientation of the grid, and this orientation also needs to be computable locally, i.e., without looking at the whole orientation of the grid.

It should be noted that while P-LCP is known to reduce to cube USOs, and P-GLCP is known to reduce to grid USOs, neither of our "two choices is enough" results imply each other. This is because there is no known reduction from an USO-type problem to an LCP-type problem.

Equivalence between P-LCP and Colorful Tangent Problems. The Ham Sandwich theorem is a classical theorem in computational geometry: given d sets of points in $\mathbb{R}^d$, we can find a hyperplane that simultaneously *bisects* all of the sets. Steiger and Zhao [21] have shown that in a restricted input setting we can not only bisect, but cut off arbitrary fractions of each point set. This is the so-called α-Ham Sandwich theorem: if the point sets are *well-separated*, then for each vector α there exists a unique choice of one point per set, such that the hyperplane spanned by these points has exactly α_i of the points from set i above. The α-Ham Sandwich problem asks to determine this unique choice of one point per set.

In this paper, we consider a restricted variant of the α-Ham Sandwich problem. Firstly, we slightly strengthen the assumption of well-separation to *strong* well-separation. Secondly we restrict the input vector α such that each entry takes either be the minimum or maximum possible value. This means that for every point set, either all points of the set must lie above, or all below the desired α-cut. Thus, the α-cuts we search for are tangents to all the point sets. Combining the assumption of strong well-separation and the solutions being tangents, we call this problem SWS-Colorful-Tangent. We also consider the binary variant of the problem, which we call SWS-2P-Colorful-Tangent, where we restrict the size of each point set to 2; finding an α-cut then corresponds to a series of binary choices, one per set.

Here our contribution is to show that SWS-Colorful-Tangent is polynomial-time equivalent to the P-LCP problem. Our new intermediate problem P-Lin-Bellman plays a crucial role in this result: we give a polynomial-time reduction from SWS-Colorful-Tangent to P-Lin-Bellman, and a polynomial-time reduction from P-Lin-Bellman to SWS-2P-Colorful-Tangent. This also shows that two choices are enough for this problem as well.

For these reductions, we consider the point-hyperplane dual of the colorful tangent problems. In the dual, every input point becomes a hyperplane, and the solution we search for is a point lying on one hyperplane of each color, and lying either above or below all

other hyperplanes. This can be encoded in the min and max operations of the P-Lin-Bellman problem. This demonstrates the usefulness of P-Lin-Bellman as an intermediate problem, since it has allowed us to show what is – to the best of our knowledge – the first polynomial-time equivalence between P-LCP and another problem.

Related Work. All problems we consider in this paper are promise search problems which lie in the complexity class PromiseUEOPL [2, 3], which is the promise setting analogue of the class UEOPL (short for Unique End of Potential Line) [9]. The latter has recently attracted the attention of many researchers trying to further the understanding of the hierarchy of total search problem classes, with a breakthrough result separating UEOPL from EOPL [14]. There is only one known natural[1] complete problem for UEOPL, called One Permutation Discrete Contraction (OPDC). OPDC also admits a natural binary variant, and it has already been shown implicitly that the binary variant is polynomial-time equivalent to the general variant [9]. Our reductions may help in finding another natural UEOPL-complete problem, even though our results for now only hold in the promise setting.

2 A New Intermediate Problem: P-Lin-Bellman

Our new P-Lin-Bellman problem is motivated by discounted games. A stochastic discounted game (SDG) is defined by a set of states S, which are partitioned into S_{Max} and S_{Min}, a set of actions A, a transition function $p : S \times A \times S \to \mathbb{R}$, a reward function $r : S \times A \to \mathbb{R}$, and a discount factor λ. The value of a SDG is known to be the solution to the following system of *Bellman equations*. For each state s we have an equation

$$x_s = \begin{cases} \max_{a \in A} \left(r(s,a) + \lambda \cdot \sum_{s' \in S} p(s,a,s') \cdot x_{s'} \right) & \text{if } s \in S_{\text{Max}}, \\ \min_{a \in A} \left(r(s,a) + \lambda \cdot \sum_{s' \in S} p(s,a,s') \cdot x_{s'} \right) & \text{if } s \in S_{\text{Min}}. \end{cases}$$

Prior work has shown that, if the game is binary, meaning that $|A| = 2$, then these Bellman equations can be formulated as a P-LCP [17, 16].

For our purposes, we are interested in the format of these equations. Note that each equation is a max or min over affine functions of the other variables. We capture this idea in the following generalized definition.

▶ **Definition 1.** *A Lin-Bellman system $G = (L, R, q, S)$ is defined by two matrices $L, R \in \mathbb{R}^{n \times n}$, a vector $q \in \mathbb{R}^n$, and a set $S \subseteq \{1, 2, \ldots, n\}$. These inputs define the following system of equations over a vector of variables $x \in \mathbb{R}^n$:*

$$x_i = \begin{cases} \max(\sum_{1 \le j \le n} L_{ij} x_j + q_i, \ \sum_{1 \le j \le n} R_{ij} x_j) & \text{if } i \in S, \\ \min(\sum_{1 \le j \le n} L_{ij} x_j + q_i, \ \sum_{1 \le j \le n} R_{ij} x_j) & \text{if } i \notin S. \end{cases} \tag{1}$$

Observe that this definition captures systems of equations in which a min or max operation is taken over two affine expressions in the other variables. Note that here we have included the additive q term only in the first of the two expressions (the second is thus linear), whereas the SDG equations have additive terms in all of the affine expressions. This is because, for our reductions, we do not need the second additive term. Otherwise, this definition captures the style of Bellman equations that are used in SDGs and other infinite games.

[1] By natural we mean a problem that is not a variant of the Unique End of Potential Line problem which naturally characterizes the class.

We say that $x \in \mathbb{R}^n$ is a solution of a Lin-Bellman system G if Equation (1) is satisfied for all i. In general, however, such an x may not exist or may not be unique. To get a problem that is equivalent to P-LCP, we need a restriction that ensures that the problem always has a unique solution, like the P-matrix restriction on the LCP problem.

To make sure that a unique solution exists, we introduce a promise to yield the promise problem P-Lin-Bellman. As the promise we guarantee the same property that is implied by the promise of the P-LCP: For every $q' \in \mathbb{R}^n$, the given Lin-Bellman system shall have a unique solution. We use the following restriction.

▶ **Definition 2.** *P-Lin-Bellman*
Input: *A Lin-Bellman system $G = (L, R, q, S)$.*
Promise: *The Lin-Bellman system (L, R, q', S) has a unique solution for every $q' \in \mathbb{R}^n$.*
Output: *A solution $x \in \mathbb{R}^n$ of G.*

This promise insists that the linear equations not only have a unique solution for the given vector q, but that they also have a unique solution no matter what vector q is given. This is analogous to a property from SDGs: an SDG has a unique solution no matter the rewards for the actions, and this corresponds to the additive q vector in the problem above.

Similarly to the LCP, where unique solutions for any right-hand side q imply that the matrix M is a P-matrix (and vice versa), this promise of P-Lin-Bellman implies something about the involved matrices. However, for P-Lin-Bellman we do not have a full characterization of the systems $(L, R, \cdot, S)$ that fulfill the promise. We only have the following rather weak necessary condition (proven in the full version of the paper), which is however useful for several of our reductions.

▶ **Lemma 3.** *In any P-Lin-Bellman instance, $R - I$ is invertible.*

Since P-Lin-Bellman is so similar to the P-LCP, we will prove the equivalence of P-LCP and P-Lin-Bellman first, in the next section.

3 Linear Complementarity Problems

We begin by giving the definitions of the (binary) linear complementarity problems.

▶ **Definition 4.** *$LCP(M, q)$*
Input: *An $n \times n$ matrix M and a vector $q \in \mathbb{R}^n$.*
Output: *Two vectors $w, z \in \mathbb{R}^n$ such that*
- *$w = Mz + q$,*
- *$w, z \geq 0$, and*
- *$w^T \cdot z = 0$.*

We are particularly interested in the case where the input matrix M is a P-matrix.

▶ **Definition 5.** *An $n \times n$ matrix M is a P-matrix if every principal minor of M is positive.*

One particularly interesting feature of the P-matrix Linear Complementarity Problem is that it always has a unique solution.

▶ **Theorem 6** ([5])**.** *M is a P-matrix if and only if $LCP(M, q)$ has a unique solution for every $q \in \mathbb{R}^n$.*

There are two problems associated with P-matrix LCPs. The first problem is a total search problem in which we must either solve the LCP, or show that the input matrix M is not a P-matrix by producing a non-positive principal minor of M. The second problem is a promise problem, where the promise is that the input is a P-matrix.

▶ **Definition 7.** *P-LCP*(M, q)
Input: *An* $n \times n$ *matrix* M *and a vector* $q \in \mathbb{R}^n$.
Promise: M *is a P-matrix.*
Output: *Two vectors* $w, z \in \mathbb{R}^n$ *that are solutions of LCP*(M, q).

In the next two sections, we will show that P-Lin-Bellman and P-LCP are equivalent under polynomial-time many-one reductions.

3.1 P-LCP to P-Lin-Bellman

Let $M \in \mathbb{R}^{n \times n}$ and $q \in \mathbb{R}^n$ be an instance of P-LCP. We will build a P-Lin-Bellman instance $G(M, q)$ in the following way. For each i in the range $1 \leq i \leq n$, we set

$$x_i = \min((Mx + x)_i + q_i, 2x_i). \tag{2}$$

In other words, we set $L = M + I$, $R = 2I$, and $S = \emptyset$. We first show that every solution of $G(M, q)$ corresponds to a solution of the LCP. We do not need to use any properties of the P-matrix in this part of the proof.

▶ **Lemma 8.** *A vector* $z \in \mathbb{R}^n$ *is a solution of the Lin-Bellman system* $G(M, q)$ *if and only if* z *and* $w = Mz + q$ *are a solution of LCP*(M, q).

Proof. If z solves Equation (2) then we have $\min(Mz + z + q, 2z) - z = 0$, and hence that $\min(Mz + q, z) = 0$. This implies that both z and w are non-negative:

$$z \geq \min(Mz + q, z) = 0,$$
$$w = Mz + q \geq \min(Mz + q, z) = 0.$$

Moreover, since $\min(Mz + q, z) = 0$, the complementarity condition also holds. Hence w and z are a solution of the LCP.

In the other direction, if w and z are solutions of the LCP, then we argue that z must be a solution of $G(M, q)$. This is because any solution of the LCP satisfies $\min(Mz + q, z) = 0$ due to the complementarity condition, and so we must have that z solves Equation (2). ◀

Next we show that the promise of P-Lin-Bellman is also satisfied.

▶ **Lemma 9.** *If* M *is a P-matrix, then* $G(M, q)$ *satisfies the promise of P-Lin-Bellman.*

Proof. We must show that the Lin-Bellman system $G(M, q')$ has a unique solution for every q'. This follows from Lemma 8, which shows that $G(M, q')$ has a unique solution if and only if the LCP defined by M and q' has a unique solution, which follows from the fact that M is a P-matrix. ◀

3.2 P-Lin-Bellman to P-LCP

For this direction, we follow and generalize the approach used by Jurdziński and Savani [17]. Suppose that we have a P-Lin-Bellman instance defined by $G = (L, R, q, S)$.

The reduction is based on the idea of simulating the operation $x = \max(a, b)$ by introducing two non-negative slack variables w and z:

$$x - w = a, \qquad x - z = b, \qquad w, z \geq 0, \qquad w \cdot z = 0.$$

Since w and z are both non-negative, any solution to the system of equations above must satisfy $x \geq \max(a, b)$. Moreover, since the complementarity condition insists that either $w = 0$ or $z = 0$, we have that $x = \max(a, b)$.

An operation $x = \min(a, b)$ can likewise be simulated by

$$x + w = a, \qquad x + z = b, \qquad w, z \geq 0, \qquad w \cdot z = 0.$$

We use this technique to rewrite the system of equations given in (1) in the following way. For each i in $1 \leq i \leq n$ we have the following.

$$x_i + w_i = \sum_{1 \leq j \leq n} L_{ij} \cdot x_j + q_i, \qquad \text{if } i \notin S \tag{3}$$

$$x_i - w_i = \sum_{1 \leq j \leq n} L_{ij} \cdot x_j + q_i, \qquad \text{if } i \in S \tag{4}$$

$$x_i + z_i = \sum_{1 \leq j \leq n} R_{ij} \cdot x_j, \qquad \text{if } i \notin S \tag{5}$$

$$x_i - z_i = \sum_{1 \leq j \leq n} R_{ij} \cdot x_j, \qquad \text{if } i \in S \tag{6}$$

$$w_i, z_i \geq 0 \tag{7}$$

$$w_i \cdot z_i = 0. \tag{8}$$

We have already argued that this will correctly simulate the max and min operations in Equation (1), and so we have the following lemma.

▶ **Lemma 10.** *$x \in \mathbb{R}^n$ is a solution of G if and only if x is a solution of the system defined by Equations* (3)–(8)*.*

We shall now reformulate Equations (3)–(8) as an LCP. To achieve this we first introduce some helpful notation: For every $n \times n$ matrix A, we define $\widehat{A}$ in the following way. For each $i, j \in \{1, 2, \ldots, n\}$ we let

$$\widehat{A}_{ij} := \begin{cases} A_{ij} & \text{if } i \in S, \\ -A_{ij} & \text{if } i \notin S. \end{cases}$$

Equations (3)–(6) can be rewritten as the following matrix equations:

$$\widehat{I}x = w + \widehat{L}x + \widehat{I}q,$$
$$\widehat{I}x = z + \widehat{R}x,$$

Eliminating x yields $(\widehat{I} - \widehat{L})(\widehat{I} - \widehat{R})^{-1}z = w + \widehat{I}q$, which is equivalent to $w = Mz + q'$ for $M = (\widehat{I} - \widehat{L})(\widehat{I} - \widehat{R})^{-1}$, $q' = -\widehat{I}q$.

We rely here on the fact that by Lemma 3, $R - I$ and thus also $(\widehat{I} - \widehat{R})$ is invertible. We have so far shown the following lemma.

▶ **Lemma 11.** *A vector $x \in \mathbb{R}^n$ is a solution of G if and only if there are vectors w, z such that w, z are a solution of $LCP(M, q)$ and x, w, and z are a solution of Equations* (3)–(8)*.*

To show that M is a P matrix, we must show that $\mathrm{LCP}(M, q'')$ has a unique solution for every $q'' \in \mathbb{R}^n$. By Lemma 11, this holds if and only if $G(L, R, q'', S)$ has a unique solution for every q'', which holds due to the P-Lin-Bellman promise. Our reduction is thus correct, and we have established our first main result:

▶ **Theorem 12.** *P-Lin-Bellman and P-LCP are equivalent under polynomial-time many-one reductions.*

3.3 P-matrix Generalized LCP to P-LCP

Generalized LCPs were originally introduced by Cottle and Dantzig [4]. A generalized LCP instance is defined by a vertical block matrix M, and a vertical block vector q where

$$M = \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_n \end{bmatrix} \qquad q = \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{pmatrix}$$

Each matrix M_i has dimensionality $b_i \times n$, while each vector q_i has b_i dimensions. Thus, if $N = \sum_i b_i$, we have that $M \in \mathbb{R}^{N \times n}$ and $q \in \mathbb{R}^N$. Given a vertical block vector x with n blocks, we use x_i^j to refer to the jth element of the ith block of x.

Given such an M and q, the *generalized linear complementarity problem* (GLCP) is to find vectors $w \in \mathbb{R}^N$ and $z \in \mathbb{R}^n$ such that

- $w = Mz + q$,
- $w \geq 0$,
- $z \geq 0$, and
- $z_i \cdot \prod_{j=1}^{b_i} w_i^j = 0$ for all i.

Given a tuple $p = (p_1, p_2, \ldots, p_n)$ such that each p_i lies in the range $1 \leq i \leq b_i$, the representative submatrix of M defined by p is given by $M(p) \in \mathbb{R}^{n \times n}$ and is constructed by selecting row p_i from each block-matrix M_i. We then say that M is a P-matrix if all of its representative submatrices are P-matrices.

We define the P-matrix GLCP (P-GLCP) as a promise problem, analogously to the P-LCP: The input is a GLCP instance (M, q) with the promise that M is a P-matrix. This promise again guarantees unique solutions:

▶ **Theorem 13** (Habetler, Szanc [15]). *A vertical block matrix M is a P-matrix if and only if the GLCP instance (M, q') has a unique solution for every $q' \in \mathbb{R}^N$.*

We are now ready to show our reduction of P-GLCP to P-Lin-Bellman.

▶ **Lemma 14.** *There is a poly-time many-one reduction from P-GLCP to P-Lin-Bellman.*

Proof. We can turn a P-matrix GLCP instance into a P-Lin-Bellman instance in much the same way as we did for P-LCPs in Section 3.1. For each i in the range $1 \leq i \leq n$ we set

$$z_i = \min\Big(\min\big\{(Mz+q)_i^j + z_i \ : 1 \leq j \leq b_i\big\}, 2z_i\Big). \tag{9}$$

We claim that any solution of the system of equations defined above yields a solution of the GLCP. In any solution to the system we have

$$\min\Big(\min\big\{(Mz+q)_i^j \ : 1 \leq j \leq b_i\big\}, z_i\Big) = 0.$$

So if we set $w_i^j = (Mz+q)_i^j$ then we have the following non-negativities

$$w_i^j = (Mz+q)_i^j \geq \min\Big(\min\big\{(Mz+q)_i^j \ : 1 \leq j \leq b_i\big\}, z_i\Big) = 0.$$
$$z_i \geq \min\Big(\min\big\{(Mz+q)_i^j \ : 1 \leq j \leq b_i\big\}, z_i\Big) = 0.$$

We can also see that the complementarity condition is satisfied, since either $z_i = 0$ or $w_i^j = 0$ for some j.

To write this as a Lin-Bellman system, we must carefully write Equation (9) using two-input min operations. We will introduce a variable x_i^j for each i in the range $1 \leq i \leq n$ and each j in the range $1 \leq j \leq b_i$. We use $\mathbf{x}^1$ to denote the vector $(x_1^1, x_2^1, \ldots, x_n^1)$. For each i, and each j in the range $1 \leq j < b_i$ we use the following.

$$x_i^j = \min\Big((M\mathbf{x}^1 + q)_i^j + x_i^1,\ x_i^{j+1}\Big) \tag{10}$$

We also use the following when $j = b_i$.

$$x_i^{b_i} = \min\Big((M\mathbf{x}^1 + q)_i^{b_i} + x_i^1,\ 2x_i^1\Big) \tag{11}$$

Let $G = (L, R, q, S)$ denote the resulting Lin-Bellman system. It is now easy to see that on the one hand, for every solution $\mathbf{x}$ of G the vectors $z := \mathbf{x}^1$ and w with $w_i^j = (Mz + q)_i^j$ form a solution of the input P-matrix GLCP. On the other hand, from any solution to the GLCP we can extract the vector $\mathbf{x}^1 := z$. Given this vector $\mathbf{x}^1$ there is only one way to extend this to an assignment for all x_i^j that is a solution to G. We thus get a one-to-one correspondence between solutions to G and solutions to the GLCP. Since the GLCP defined by M is promised to have a unique solution for every q', this thus implies that the Lin-Bellman system $G' = (L, R, q', S)$ also has a unique solution for every q'. Thus, this is indeed a P-Lin-Bellman instance. ◀

Combining this with the previously proven Theorem 12, we get the following corollary:

▶ **Corollary 15.** *P-GLCP and P-LCP are polynomial-time equivalent.*

4 Colorful Tangents and P-Lin-Bellman

Let us start by introducing the definitions and prior results on the α-Ham Sandwich theorem.

▶ **Definition 16.** *A family of point sets $P_1 = \{p_{1,1}, \ldots, p_{1,s_1}\}, P_2, \ldots, P_d \subset \mathbb{R}^d$ is said to be* well-separated *if for any non-empty index set $I \subset [d]$, there exists a hyperplane h such that h separates the points in $\bigcup_{i\in I} P_i$ from those in $\bigcup_{i\in[d]\setminus I} P_i$.*

In the setting of well-separated point sets, the classical Ham Sandwich theorem can be strengthened to the more modern α-Ham Sandwich theorem due to Steiger and Zhao [21], for which we need the definition of an $(\alpha_1, \ldots, \alpha_d)$-cut:

▶ **Definition 17.** *Given positive integers $\alpha_i \leq |P_i|$ for all $i \in [d]$, an $(\alpha_1, \ldots, \alpha_d)$-cut is a hyperplane h such that $h \cap P_i \neq \emptyset$ and $|h^+ \cap P_i| = \alpha_i$ for all $i \in [d]$.*

In this definition and in the rest of this section, h^+ and h^- denote the two closed halfspaces bounded by a hyperplane h. When h is a colorful hyperplane (i.e., a hyperplane containing a colorful point set, which is a set $\{p_1, \ldots, p_d\}$ with $p_i \in P_i$), its positive side is determined by the orientation of the points $p_i \in P_i$ spanning it.

▶ **Theorem 18** (α-Ham Sandwich theorem [21])**.** *Given d point sets $P_1, \ldots, P_d \subset \mathbb{R}^d$ that are well-separated and in weak general position*[2]*, for every $(\alpha_1, \ldots, \alpha_d)$ where $1 \leq \alpha_i \leq |P_i|$, there exists a unique $(\alpha_1, \ldots, \alpha_d)$-cut.*

[2] Weak general position is a weaker version of general position. We will not go further into this since we will always ensure classical general position when invoking this theorem.

The α-Ham Sandwich theorem guarantees us existence and uniqueness of an $(\alpha_1, \ldots, \alpha_d)$-cut, and it is thus not too surprising that the problem of finding such a cut is contained in UEOPL [3].

In this section, we wish to show equivalence of this problem to P-Lin-Bellman, however, we only manage this with some slight changes. Firstly, we wish to drop the assumption of weak general position, since it is not easily guaranteed when reducing from P-Lin-Bellman. Because of this we need to weaken the definition of $(\alpha_1, \ldots, \alpha_d)$-cuts accordingly:

▶ **Definition 19.** *Given a well-separated family of point sets $P_1, \ldots, P_d \subset \mathbb{R}^d$ and $(\alpha_1, \ldots, \alpha_d)$ such that $1 \leq \alpha_i \leq |P_i|$, a hyperplane h is an $(\alpha_1, \ldots, \alpha_d)$-cut if for all $i \in [d]$, $h \cap P_i \neq \emptyset$ and $|(h^+ \setminus h) \cap P_i| + 1 \leq \alpha_i \leq |h^+ \cap P_i|$.*

In other words, we allow a hyperplane to contain more than one point per color, and all the additional points may be "counted" for either side of the hyperplane. Note that even without any general position assumption, the affine hull of every colorful set of points must have dimension at least $d-1$, and thus span a unique hyperplane (see the full version of the paper). We further show in the full version that every colorful subset of points on the same hyperplane h must be oriented the same way, and thus h^+ is uniquely defined even without the weak general position assumption.

Note that the α-Ham Sandwich theorem (Theorem 18) generalizes directly to the setting where the assumption of weak general position is dropped and Definition 17 is replaced by Definition 19.

▶ **Theorem 20.** *Given a well-separated family of point sets $P_1, P_2, \ldots, P_d \subset \mathbb{R}^d$ and $(\alpha_1, \ldots, \alpha_d)$ for $1 \leq \alpha_i \leq |P_i|$, there exists a unique hyperplane h that is an $(\alpha_1, \ldots, \alpha_d)$-cut according to Definition 19.*

Proof. This simply follows from perturbing the points and applying Theorem 18. ◀

Secondly, we have to strengthen well-separation to *strong well-separation*. We illustrate well-separation and strong well-separation in Figure 3.

▶ **Definition 21.** *A family of point sets $P_1 = \{p_{1,1}, \ldots, p_{1,s_1}\}, P_2, \ldots, P_d \subset \mathbb{R}^d$ is said to be* strongly well-separated *if the point sets $P'_1, \ldots, P'_d$ obtained by projecting $P_1, \ldots, P_d$ to the hyperplane spanned by $p_{1,1}, \ldots, p_{d,1}$ are well-separated.*

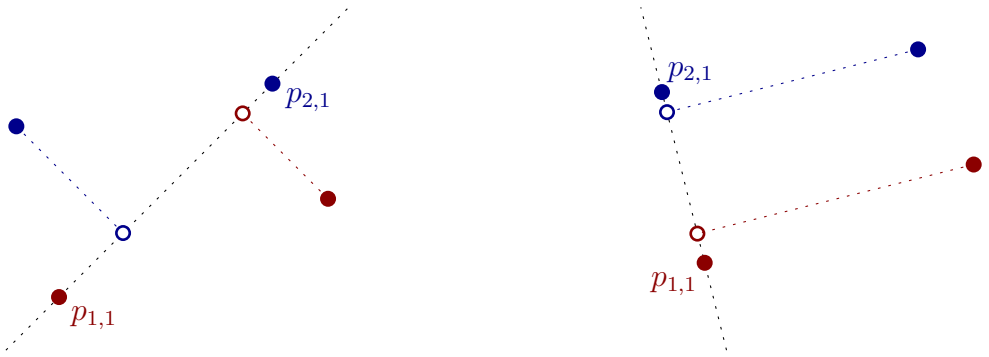


Figure 3 The point sets (dots) on the left are well-separated but not strongly well-separated, since their projections (circles) are not well-separated. The point sets on the right are strongly well-separated.

We are now ready to introduce the problem we wish to study.

▶ **Definition 22.** *SWS-Colorful-Tangent*$(P_1, \ldots, P_d, \alpha_1, \ldots, \alpha_d)$
Input: *d point sets $P_1, \ldots, P_d \subset \mathbb{R}^d$ and a vector $(\alpha_1, \ldots, \alpha_d)$, where $\alpha_i \in \{1, |P_i|\}$.*
Promise: *The point sets $P_1, \ldots, P_d$ are strongly well-separated.*
Output: *An $(\alpha_1, \ldots, \alpha_d)$-cut.*
We depart from the *Ham Sandwich* name for this problem since for $\alpha_i \in \{1, |P_i|\}$, the α-"cuts" are really just colorful tangent hyperplanes, as can be seen in Figure 4.

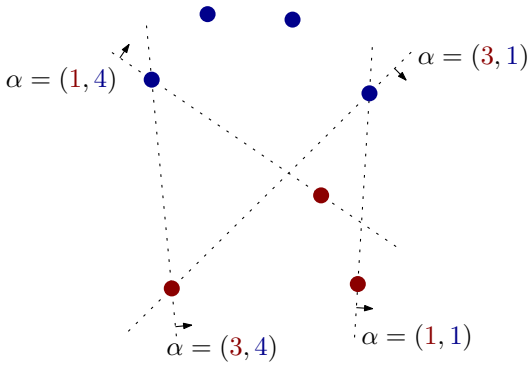


Figure 4 Two well-separated point sets and the solution α-cuts to all four possible α-vectors. Note that the positive side of a colorful line is considered to be the right side when orienting the line from the red point to the blue point.

Similarly to P-GLCP and its binary variant P-LCP, we also define SWS-2P-Colorful-Tangent as the restriction of SWS-Colorful-Tangent to inputs where $|P_i| = 2$ for all $i \in [d]$. We now wish to prove these two problems polynomial-time equivalent to P-Lin-Bellman. To achieve this, we reduce SWS-Colorful-Tangent to P-Lin-Bellman, and P-Lin-Bellman to SWS-2P-Colorful-Tangent, with the reduction by inclusion from SWS-2P-Colorful-Tangent to SWS-Colorful-Tangent closing the cycle.

For our reductions, we will use the concept of *point-hyperplane duality*, as described by Edelsbrunner in [8, p.13]. Point-hyperplane duality is a bijective mapping from all points in $\mathbb{R}^d$ to all non-vertical hyperplanes in $\mathbb{R}^d$. A point $p = (p_1, \ldots, p_d)$ is mapped to the hyperplane $p^* = \{x \in \mathbb{R}^d \mid 2p_1x_1 + \ldots 2p_{d-1}x_{d-1} - x_d = p_d\}$. The hyperplane p^* is called *the dual of* p, and for a non-vertical hyperplane h, its dual h^* is the unique point p such that $p^* = h$.

Point-hyperplane duality has the nice properties of *incidence preservation* and *order preservation*: for any point p and non-vertical hyperplane h, we have that $p \in h$ if and only if $h^* \in p^*$, and furthermore we have that p lies above h if and only if h^* lies above p^*.

We are now ready to begin presenting our reductions. For all of these reductions, we use the following crucial yet simple observation on strongly well-separated point sets:

▶ **Observation 23.** *If a set of point sets $P_1, \ldots, P_d$ is strongly well-separated, then every set of point sets $P'_1, \ldots, P'_d$ obtained by moving points $p_{i,j}$ $(j \neq 1)$ orthogonally to the hyperplane spanned by $p_{1,1}, \ldots, p_{d,1}$ is also strongly well-separated.*

The general idea of the reduction is to represent the linear inputs to a min or max operation in a Lin-Bellman system by a point $p_{i,1}$ and the affine inputs by a point $p_{i,j}$ $(j \neq 1)$. As a warm-up, we first only reduce from the two-point version.

▶ **Lemma 24.** *SWS-2P-Colorful-Tangent poly-time reduces to P-Lin-Bellman.*

Proof. We first linearly transform our point sets such that the plane through $p_{1,1}, \dots, p_{d,1}$ is mapped to the horizontal plane $\{(x_1, \dots, x_d) \in \mathbb{R}^d \mid x_d = 0\}$. Without loss of generality, we assume that after this transformation the colorful hyperplane h spanned by $p_{1,1}, \dots, p_{d,1}$ is oriented upwards, i.e., its positive halfspace contains $(0, \dots, 0, +\infty)$.

Then, we apply point-hyperplane duality. Each point $p_{i,j}$ becomes a hyperplane $h_{i,j} = p^*_{i,j}$, which can be described by some function $h_{i,j} : \mathbb{R}^d \to \mathbb{R}$ such that a point $x \in \mathbb{R}^d$ lies strictly above $h_{i,j}$ if $h_{i,j}(x) > 0$, and on $h_{i,j}$ if $h_{i,j}(x) = 0$. For the hyperplanes $h_{i,1}$ dual to our points $p_{i,1}$, we furthermore have that they go through the origin, i.e., $h_{i,j}$ is a linear function (and not only an affine function; it has no additive constant term).

Before applying duality, the desired $(\alpha_1, \dots, \alpha_d)$-cut hyperplane is a hyperplane h containing at least one point p'_i per set P_i, such that the other point of P_i lies on or above h if and only if $\alpha_i = |P_i|$, and on or below h if and only if $\alpha_i = 1$. In the dual, this now means that we need to find a point p which lies on at least one of the hyperplanes $h_{i,1}, h_{i,2}$ for each i, and such that it lies above (below) or on both of these hyperplanes if $\alpha_i = |P_i|$ ($\alpha_i = 1$). This can be described by the equations $\min\{h_{i,1}(x), h_{i,2}(x)\} = 0$ if $\alpha_i = |P_i|$ (and max instead of min, otherwise). This can of course be rewritten as $x_i = \min\{h_{i,2}(x) + x_i, h_{i,1}(x) + x_i\}$. Note again that the second input to this minimum is *linear* in x (there is no additive term). With one such constraint each per point set P_i, we thus get a system of d equations over d variables, which together form a Lin-Bellman system.

It remains to check whether this system is also a P-Lin-Bellman instance. We first see that changing q simply corresponds to moving the hyperplanes $h_{i,2}$ in a parallel fashion (i.e., without changing their normal vectors), which in the primal corresponds to moving them in direction x_d. By Observation 23 and Theorem 20, we always have unique $(\alpha_1, \dots, \alpha_d)$-cuts in this modified family of point sets, and thus the Lin-Bellman instance has a unique solution for all q'. ◀

▶ **Theorem 25.** *SWS-Colorful-Tangent poly-time reduces to P-Lin-Bellman.*

Proof (sketch). The proof works exactly the same way as the previous proof of Lemma 24. The only difference is that the equations that need to be encoded are of the form

$$x_i = \min\{h_{i,1}(x) + x_i, \dots, h_{i,|P_i|}(x) + x_i\}, \tag{12}$$

where still only $h_{i,1}(x)$ is guaranteed to be a linear function. To encode this as a Lin-Bellman system, we apply the same trick as in the proof of Lemma 14 to split the multi-input minimum into multiple two-input minima, with the remaining details found in the full version of the paper. ◀

To complete our cycle of reductions, we need to reduce from P-Lin-Bellman to SWS-2P-Colorful-Tangent. For this reduction we basically perform the process from the proof of Lemma 24 in reverse. The details are found in the full version of the paper.

▶ **Theorem 26.** *P-Lin-Bellman poly-time reduces to SWS-2P-Colorful-Tangent.*

To end this section, we want to note that the assumption of strong well-separation is not too strong, at least combinatorially:

▶ **Theorem 27.** *For every family of well-separated point sets, there exists a family of strongly well-separated point sets with the same combinatorial structure, i.e., the two families have the exact same order type.*

The proof can be found in the full version of the paper. Note that the proof is merely existential and it is unlikely that the translation can be performed efficiently.

5 Grid-USO and Cube-USO

We saw in the two previous sections that both P-GLCP and SWS-Colorful-Tangent can be reduced to their respective "binary" variants, P-LCP and SWS-2P-Colorful-Tangent. In this section we discuss grid USOs and cube USOs, which are a combinatorial framework that can be used to model all the problems studied in the sections above as well as many more algebraic and geometric problems. Similarly to the previous sections, cube USOs are a restriction of grid USOs to the "binary" case.

5.1 Definitions

We define C as a d-dimensional hypercube. We say $V(C) := \{0,1\}^d$ and $\{J,K\} \in E(C)$ iff $|J \oplus K| = 1$, where $\oplus$ is the bit-wise xor operation (i.e., addition in $\mathbb{Z}_2^d$) and $|\cdot|$ counts the number of 1-entries. For notational simplicity, in this section we use the same name both for a bitvector in $\{0,1\}^d$ and for the set of dimensions $i \in [d]$ in which the vector is 1. For example for $J, K \in \{0,1\}^d$ we write $J \subseteq K$ if for all $i \in [d]$ with $J_i = 1$ we also have $K_i = 1$.

The orientation of the edges of the cube C is given by an *orientation* function $O : V(C) \to \{0,1\}^d$, where $O(J)_i = 0$ means J has an incoming edge in dimension i and $O(J)_i = 1$ is an outgoing edge in dimension i.

▶ **Definition 28.** *An orientation is a* unique sink orientation (USO) *if and only if every induced subcube has a unique sink.*

The common search problem version of this problem is to find the global sink of the cube, given the function O as a boolean circuit. Note that it is co-NP-complete to test whether a given orientation is USO [13]. We consider the promise version Cube-USO, which was one of the first search problems proven to lie in the complexity class Promise-UEOPL [9].

▶ **Definition 29.** *Cube-USO*
Input: *A circuit computing the orientation function O on a d-dimensional cube C.*
Promise: *O is a Unique Sink Orientation.*
Output: *A vertex $J \in V(C)$ which is a sink, i.e., $\forall i \in [d] : O(J)_i = 0$.*

While the d-dimensional hypercube is the product of d copies of K_2 (the complete graph on two vertices), a *grid graph* is the product of complete graphs of arbitrary size: A d-dimensional grid graph Γ is given by $n_1, \ldots, n_d \in \mathbb{N}^+$:

$$V(\Gamma) := \{0, \ldots, n_1\} \times \cdots \times \{0, \ldots, n_d\},$$
$$E(\Gamma) := \{\{v, w\} \mid v, w \in V(\Gamma), \exists! i \in [d] : v_i \neq w_i\}.$$

We say the grid has d *dimensions* and each dimension i has $n_i + 1$ *directions*.

The subgraph Γ' of Γ induced by the vertices $V(\Gamma') = N_1 \times \cdots \times N_d$ for non-empty $N_i \subseteq \{0, \ldots, n_i\}$ is called an *induced subgrid* of Γ. Note that if for some i we have $|N_i| = 1$, the induced subgrid loses a dimension. If $|N_i| = 1$ for all $i \in [d]$ except one, we say that the induced subgrid Γ' is a *simplex*. A simplex is a complete graph K_{n_j+1} for some $j \in [d]$.

The orientation of a grid is given by the *outmap function*, which assigns each vertex a binary vector that encodes whether its incident edges are incoming or outgoing. More formally, the outmap function is a function $\sigma : V(\Gamma) \to \{0,1\}^{n_1+\ldots+n_d}$, where $\sigma(v)_{n_1+\ldots+n_i+j} = 1$ denotes that the edge from v to its j-th neighbor w in dimension $i+1$ is outgoing, i.e., oriented from v to w. Note that any circuit computing σ has $n_1 + \ldots + n_d$ outputs, and is thus of size $\Omega(n_1 + \ldots + n_d)$.

For notational convenience, we denote the entry of $\sigma(v)$ relevant to the edge $\{v, w\}$ by $\sigma(v, w)$. In other words, for each pair of vertices v, w such that $\{v, w\} \in E(\Gamma)$, $\sigma(v, w) = 1$ iff the edge between v and w is oriented towards w.

▶ **Definition 30** (Gärtner, Morris, Rüst [10]). *An orientation of a grid graph is a* unique sink orientation (USO) *if and only if every induced subgrid has a unique sink.*

A unique sink orientation of a simplex with n vertices is equivalent to a permutation of n elements, i.e. every unique sink orientation of one simplex can be given by a total order of its vertices. The minimum element of the order is the source, the maximum element is the sink.

Since every cube is also a grid, the complexity of checking whether σ is USO is also co-NP-hard. So again, we consider the promise search version of this problem:

▶ **Definition 31.** *Grid-USO*
Input: *A d-dimensional grid $\Gamma = (n_1, \ldots, n_d)$ and a circuit computing its outmap σ.*
Promise: *σ is a Unique Sink Orientation.*
Output: *A sink, i.e. a vertex $v \in V(\Gamma)$ s.t. $\forall w \in V(\Gamma)$ with $\{v, w\} \in E(\Gamma) : \sigma(v, w) = 0$.*

Just like Cube-USO, Grid-USO also lies in the search problem complexity class Promise-UEOPL [1]. It is well known that P-LCP reduces to Cube-USO [22] and P-GLCP reduces to Grid-USO [10]. Since SWS-2P-Colorful-Tangent and SWS-Colorful-Tangent can both be reduced to P-LCP and P-GLCP respectively, they also reduce to Cube-USO and Grid-USO respectively. However, we show a direct and straightforward reduction from SWS-2P-Colorful-Tangent to Cube-USO, and from SWS-Colorful-Tangent to Grid-USO. These direct reductions do not require strong well-separation, only classical well-separation. The proof can be found in the full version of the paper.

▶ **Lemma 32.** *Assuming general position of the input points, finding an α-cut in a well-separated point set family $P_1, \ldots, P_d \subset \mathbb{R}^d$ for $\alpha_i \in \{1, |P_i|\}$ can be reduced to Grid-USO in polynomial time. The reduction goes to Cube-USO if for all i, $|P_i| = 2$.*

▶ Remark 33. If we instead want to find an α'-cut for arbitrary α', we can use the reduction from Lemma 32 with $\alpha = (1, \ldots, 1)$, but instead of searching for a sink in the resulting grid USO Γ, we need to find a vertex with $\alpha'_i - 1$ outgoing edges in each dimension i.

By [10, Theorem 2.14], this vertex is guaranteed to exist and to be unique. We call the problem of searching for such a vertex α-Grid-USO. Note that α-Grid-USO is not known to reduce to regular Grid-USO, nor is it known to lie in Promise-UEOPL (compared to α-Ham Sandwich which does lie in Promise-UEOPL [3]).

5.2 Grid-USO to Cube-USO

In this section we show that every Grid-USO instance (Γ, σ) can be reduced to a Cube-USO instance (C,O) in polynomial time such that given the global sink of O, we can derive the global sink of σ in polynomial time.

We are given a d-dimensional grid $\Gamma = (n_1, \ldots, n_d)$ and turn it into a hypercube of dimension $n := \sum n_i$. Note that Γ is n-regular and so is C. For every dimension of the grid spanned by $n_i + 1$ directions, we assign a block of n_i dimensions in C. For simplicity, we index into the dimensions of the hypercube using double indices: for every bitstring $J \in \{0, 1\}^n$, we write $J_{i,j}$ (where $j \in [n_i]$) as a shorthand for $J_{j+\sum_{l \in [i-1]} n_l}$.

Each vertex $J \in V(C)$ is assigned a corresponding vertex in the grid Γ as follows: For every grid dimension $i \in [d]$, we assign J a *color* j_i with

$$j_i := max(\{0\} \cup \{h \mid h \in [n_i], J_{i,h} = 1\}). \tag{13}$$

The tuple of colors $(j_1, \ldots, j_d)$ is the vertex in the grid that we associate with J. When we orient J, we orient each block i of dimensions of C independently, according to the corresponding simplex in dimension i in Γ that contains this vertex $(j_1, \ldots, j_d)$.

A single simplex of Γ with $n_i + 1$ directions is turned into an n_i-cube as shown in Figure 5. The vertex (0) of the simplex is mapped to the vertex 0^{n_i}, and all the other vertices of the grid are mapped to its neighbors. We call these $n_i + 1$ vertices the *grid-vertices*, all others are called *non-grid-vertices*. The paths of length one and two between these grid-vertices encode the orientation of the grid. This completely orients the grid-vertices. The rest of the cube is oriented according to the colors of the non-grid-vertices; edges between vertices of two different colors are oriented as the corresponding edge in the grid. Vertices of the same color are oriented in such a way that in the subcube corresponding to all vertices of the same color all the edges of the same dimension are oriented the same way. This orientation is uniquely defined since the unique grid-vertex of this color is already completely oriented.

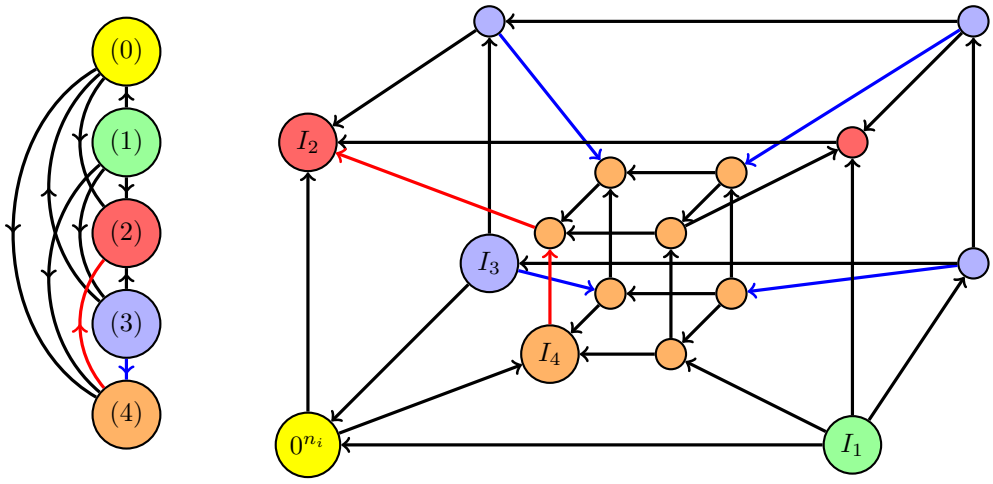


Figure 5 Example of placing grid-vertices (large) in the cube, coloring the non-grid-vertices (small) and orienting the edges. Each edge of the grid becomes a path of length two between two grid-vertices, see the red edges. Edges between vertices of different colors are oriented the same as in the grid, see the blue edges. Each subcube of the same color is oriented in a uniform way.

To formalize this construction for the complete grid, we need to define a secondary color j_i^* for each vertex $J \in V(C)$ and each grid dimension $i \in [d]$, which for all vertices with color $j_i \neq 0$ corresponds to the color of the neighboring vertex of J in dimension (i, j_i):

$$j_i^* := max(\{0\} \cup \{h \mid h \in [n_i] \setminus \{j_i\}, J_{i,h} = 1\}). \tag{14}$$

The orientation O of the cube C is now defined as follows:

$$O(J)_{i,h} := \begin{cases} \sigma((j_1, \ldots, j_i, \ldots, j_d), (j_1, \ldots, h, \ldots j_d)) \oplus J_{i,h} & \text{for } h < j_i, \\ \sigma((j_1, \ldots, j_i, \ldots, j_d), (j_1, \ldots, h, \ldots j_d)) & \text{for } h > j_i, \\ \sigma((j_1, \ldots, j_i, \ldots, j_d), (j_1, \ldots, j_i^*, \ldots j_d)) & \text{for } h = j_i. \end{cases} \tag{15}$$

▶ **Theorem 34.** *If σ is a unique sink orientation, then the orientation constructed by Equation (15) is a unique sink orientation. A circuit computing O can be computed in polynomial time. Given the sink of O, the sink of σ can be found in polynomial time.*

The proof can be found in the full version of the paper.

6 Open Questions

Total Search Problem Versions. All reductions we provided in this paper are between the promise problem versions of the involved problems. It may be interesting to also find reductions between the total search problem versions.

Missing Reductions. We were unable to show that finding an α-cut for arbitrary α is not more difficult than finding it for $\alpha_i \in \{1, |P_i|\}$. There might thus be a difference in the computational complexity of α-Ham-Sandwich and SWS-Colorful-Tangent. Similarly, in Grid-USO, we also do not know whether it is not more difficult to find a vertex with a specific refined index (which must also be unique) rather than searching for a sink. Note that in the case of α-Ham-Sandwich, it is at least known that the problem is contained in UEOPL. This is not known for α-Grid-USO.

Semantics of the Grid-USO to Cube-USO Reduction. On the levels of USOs, we do not know the exact operations that the reductions from P-GLCP to P-LCP and SWS-Colorful-Tangent to SWS-2P-Colorful-Tangent perform. It would be very interesting to analyze whether these reductions actually perform the same operation as the Grid-USO to Cube-USO reduction (Theorem 34), i.e., whether these reductions commute. It would also be interesting to study whether the Grid-USO to Cube-USO preserves realizability, i.e., whether if there exists a P-GLCP instance inducing a certain grid USO, there also exists a P-LCP instance inducing the resulting cube USO.

References

1 Michaela Borzechowski. The complexity class unique end of potential line. Master's thesis, Freie Universität Berlin, 2021.

2 Michaela Borzechowski and Wolfgang Mulzer. Unique sink orientations of grids is in unique end of potential line, 2022. doi:10.48550/arXiv.2209.02101.

3 Man-Kwun Chiu, Aruni Choudhary, and Wolfgang Mulzer. Computational Complexity of the α-Ham-Sandwich Problem. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.31.

4 Richard W. Cottle and George B. Dantzig. A generalization of the linear complementarity problem. *Journal of Combinatorial Theory*, 8(1):79–90, 1970.

5 Richard W. Cottle, Jong-Shi Pang, and Richard E. Stone. *The Linear Complementarity Problem*. Society for Industrial and Applied Mathematics (SIAM), 2009.

6 Aniekan A. Ebiefung. Existence theory and Q-matrix characterization for the generalized linear complementarity problem. *Linear Algebra and its Applications*, 223-224:155–169, 1995. Honoring Miroslav Fiedler and Vlastimil Ptak. doi:10.1016/0024-3795(95)00091-5.

7 Aniekan A. Ebiefung. Existence theory and Q-matrix characterization for the generalized linear complementarity problem: Revisited. *Journal of Mathematical Analysis and Applications*, 329(2):1421–1429, 2007. doi:10.1016/j.jmaa.2006.07.036.

8 Herbert Edelsbrunner. Algorithms in combinatorial geometry. In A. Salomaa W. Brauer, G. Rozenberg, editor, *EATCS Monographs on Theoretical Computer Science*, volume 10. Springer Berlin Heidelberg, 1987. doi:10.1007/978-3-642-61568-9.

9 John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *Journal of Computer and System Sciences*, 114:1–35, 2020. doi:10.1016/j.jcss.2020.05.007.

10 Bernd Gärtner, Walter D. Morris jr., and Leo Rüst. Unique sink orientations of grids. *Algorithmica*, 51(2):200–235, 2008. doi:10.1007/s00453-007-9090-x.

11 Bernd Gärtner and Leo Rüst. Simple stochastic games and P-matrix generalized linear complementarity problems. In Maciej Liśkiewicz and Rüdiger Reischuk, editors, *Fundamentals of Computation Theory*, pages 209–220. Springer Berlin Heidelberg, 2005. doi:10.1007/11537311_19.

12 Bernd Gärtner and Ingo Schurr. Linear programming and unique sink orientations. In *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 749–757, 2006. doi:10.5555/1109557.1109639.

13 Bernd Gärtner and Antonis Thomas. The complexity of recognizing unique sink orientations. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 341–353, Dagstuhl, Germany, 2015. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2015.341.

14 Mika Göös, Alexandros Hollender, Siddhartha Jain, Gilbert Maystre, William Pires, Robert Robere, and Ran Tao. Separations in proof complexity and tfnp. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1150–1161, 2022. doi:10.1109/FOCS54457.2022.00111.

15 George J. Habetler and Bohdan P. Szanc. Existence and uniqueness of solutions for the generalized linear complementarity problem. *Journal of Optimization Theory and Applications*, 84(1):103–116, January 1995. doi:10.1007/BF02191738.

16 Thomas Dueholm Hansen and Rasmus Ibsen-Jensen. The complexity of interior point methods for solving discounted turn-based stochastic games. In *Proc. of CiE*, volume 7921, pages 252–262. Springer, 2013.

17 Marcin Jurdzinski and Rahul Savani. A simple P-matrix linear complementarity problem for discounted games. In *Proc. of Conference on Computability in Europe (CiE)*, pages 283–293, 2008. doi:10.1007/978-3-540-69407-6_32.

18 Nimrod Megiddo. A note on the complexity of P-matrix LCP and computing an equilibrium. Technical report, IBM Research, 1988.

19 Katta G. Murty. On the number of solutions to the complementarity problem and spanning properties of complementary cones. *Linear Algebra and its Applications*, 5(1):65–108, 1972. doi:10.1016/0024-3795(72)90019-5.

20 Katta G. Murty and Feng-Tien Yu. *Linear complementarity, linear and nonlinear programming*. Heldermann, 1988.

21 William Steiger and Jihui Zhao. Generalized ham-sandwich cuts. *Discrete & Computational Geometry*, 44(3):535–545, 2010. doi:10.1007/s00454-009-9225-8.

22 Alan Stickney and Layne Watson. Digraph models of Bard-type algorithms for the linear complementarity problem. *Mathematics of Operations Research*, 3(4):322–333, 1978. URL: https://www.jstor.org/stable/3689630.

23 Ola Svensson and Sergei G. Vorobyov. Linear complementarity and P-matrices for stochastic games. In Irina B. Virbitskaite and Andrei Voronkov, editors, *Perspectives of Systems Informatics, 6th International Andrei Ershov Memorial Conference, PSI 2006, Novosibirsk, Russia, June 27-30, 2006. Revised Papers*, volume 4378, pages 409–423, 2006.

24 Tibor Szabó and Emo Welzl. Unique sink orientations of cubes. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 547–555, 2001. doi:10.1109/SFCS.2001.959931.

Kernelization Dichotomies for Hitting Subgraphs Under Structural Parameterizations

Marin Bougeret
LIRMM, Université de Montpellier, CNRS, France

Bart M. P. Jansen
Eindhoven University of Technology, The Netherlands

Ignasi Sau
LIRMM, Université de Montpellier, CNRS, France

Abstract

For a fixed graph H, the H-Subgraph Hitting problem consists in deleting the minimum number of vertices from an input graph to obtain a graph without any occurrence of H as a subgraph. This problem can be seen as a generalization of Vertex Cover, which corresponds to the case $H = K_2$. We initiate a study of H-Subgraph Hitting from the point of view of characterizing structural parameterizations that allow for polynomial kernels, within the recently active framework of taking as the parameter the number of vertex deletions to obtain a graph in a "simple" class $\mathcal{C}$. Our main contribution is to identify graph parameters that, when H-Subgraph Hitting is parameterized by the vertex-deletion distance to a class $\mathcal{C}$ where any of these parameters is bounded, and assuming standard complexity assumptions and that H is biconnected, allow us to prove the following sharp dichotomy: the problem admits a polynomial kernel if and only if H is a clique. These new graph parameters are inspired by the notion of *$\mathcal{C}$-elimination distance* introduced by Bulian and Dawar [Algorithmica 2016], and generalize it in two directions. Our results also apply to the version of the problem where one wants to hit H as an induced subgraph, and imply in particular, that the problems of hitting minors and hitting (induced) subgraphs have a substantially different behavior with respect to the existence of polynomial kernels under structural parameterizations.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases hitting subgraphs, hitting induced subgraphs, parameterized complexity, polynomial kernel, complexity dichotomy, elimination distance

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.33

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2404.16695` [11]

Funding *Bart M. P. Jansen*: Supported by the Dutch Research Council (NWO) through Gravitation-grant NETWORKS-024.002.003.
Ignasi Sau: Supported by the French project ELIT (ANR-20-CE48-0008-01).

1 Introduction

The theory of parameterized complexity deals with *parameterized problems*, which are decision problems in which a positive integer k, called the *parameter*, is associated with every instance x. One of the pivotal notions in the domain is that of *kernelization* [5, 15, 20, 24, 39], which is a polynomial-time algorithm that reduces any instance (x, k) of a parameterized problem to an equivalent instance (x', k') of the same problem whose size is bounded by $f(k)$ for some function f, which is the *size* of the kernelization. A kernelization algorithm, or just *kernel*, can be seen as a preprocessing procedure with provable guarantees, and it is fundamental to


find kernels of the smallest possible size, ideally polynomial. Identifying which parameterized problems admit polynomial kernels is one of the most active areas within Parameterized Complexity (cf. for instance [24]).

When dealing with a problem where the goal is to find a (say, small) subset of vertices S of an input graph G satisfying some property, such as Vertex Cover, it is natural to consider as the parameter the size of the desired set S. Assuming that the problem admits a polynomial kernel parameterized by $|S|$, as it is the case for Vertex Cover and many other problems [15, 24], we can ask whether the problem still admits polynomial kernels when the parameter is (asymptotically) smaller than the solution size. The goal of this approach is to provide better preprocessing guarantees, as well as to understand what is the limit of the polynomial-time "compressibility" of the considered problem. For problems defined on graphs, apart from using the solution size as a parameter, it is common to consider so-called *structural parameters*, which quantify some structural property of the input graph that can be seen as a measure of its "complexity". Among structural parameters, the most successful is probably *treewidth* [15, 35], but unfortunately taking the treewidth of the input graph as the parameter does not allow for polynomial kernels for essentially all natural optimization problems, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ [6, 8]. The same applies to another relevant graph parameter called *treedepth*, denoted by td and defined as the minimum number of rounds needed to obtain the empty graph, where each round consists of removing one vertex from each connected component.

In fact, the lower bound proofs go through for parameterizations for which the value on a disconnected graph is the *maximum*, rather than the *sum*, of the values of its components, and whose value is polynomially bounded in the size of the graph. Hence, to be able to obtain positive kernelization results, we need to turn to parameterizations other than *width measures*. This motivates to consider structural parameters that quantify the "distance from triviality", a concept first coined by Guo, Hüffner, and Niedermeier [28]. The idea is to take as the parameter the vertex-deletion distance of a graph to a "trivial" graph class where the considered problem can be solved efficiently. This paradigm has proved very successful for a number of problems, in particular for Vertex Cover. In an influential work, Jansen and Bodlaender [30] showed that Vertex Cover admits a polynomial kernel when parameterized by the feedback vertex number of the input graph, which is the vertex-deletion distance to the "trivial" class of forests. This result triggered a number of results in the area, aiming to characterize the "trivial" families $\mathcal{F}$ for which Vertex Cover admits a polynomial kernel under this parameterization [10, 12, 25, 29, 40].

Let us mention some of these results that are relevant to our work. Bougeret and Sau [12] proved that Vertex Cover admits a polynomial kernel parameterized by the vertex-deletion distance to a graph of bounded treedepth. This result was further generalized into two orthogonal directions, namely by considering a more general problem or a more general target graph class $\mathcal{F}$. For the former generalization, Jansen and Pieterse [33] proved that the following problem also admits a polynomial kernel parameterized by the vertex-deletion distance to a graph of bounded treedepth: for a fixed finite set of connected graphs $\mathcal{M}$, the $\mathcal{M}$-Minor Deletion problem consists in deleting the minimum number of vertices from an input graph to obtain a graph that does not contain any of the graphs in $\mathcal{M}$ as a minor. Note that this problem (vastly) generalizes Vertex Cover, which corresponds to the case $\mathcal{M} = \{K_2\}$. For the latter generalization, Bougeret, Jansen, and Sau [10] proved that Vertex Cover admits a polynomial kernel parameterized by the vertex-deletion distance to a graph of bounded *bridge-depth*, which is a parameter that generalizes treedepth and the feedback vertex number. It turns out that, under the assumption that the target

graph class $\mathcal{F}$ is minor-closed, the property of $\mathcal{F}$ having bounded bridge-depth is also a *necessary* condition for Vertex Cover admitting a polynomial kernel. Another complexity dichotomy of this flavor has been achieved by Dekker and Jansen [18] for the Feedback Vertex Set problem (with another characterization of the target graph class $\mathcal{F}$). These complexity dichotomies, while precious, are unfortunately quite hard to obtain, and the current knowledge seems still far from obtaining dichotomies of this type for general families of problems, such as for $\mathcal{M}$-Minor Deletion for any finite family of graphs $\mathcal{M}$. Indeed, it is wide open whether $\mathcal{M}$-Minor Deletion admits a polynomial kernel parameterized by the solution size for any set $\mathcal{M}$ containing only non-planar graphs [23, 34], so considering parameters smaller than the solution size is still out of reach.

Our contribution. We consider an alternative generalization of Vertex Cover by considering (induced) *subgraphs* instead of *minors*. Namely, for a fixed graph H, the H-Subgraph Hitting problem is defined as deleting the minimum number of vertices from an input graph to obtain a graph without any occurrence of H as a subgraph. The H-Induced Subgraph Hitting problem is defined analogously by forbidding occurrences of H as an *induced* subgraph. (As we shall see later, both problems behave in the same way with respect to our results.) Note that both problems correspond to Vertex Cover for the case $H = K_2$ and therefore are indeed generalization of it. As opposed to the case for hitting minors, it is well-known that both the H-Subgraph Hitting and H-Induced Subgraph Hitting problems admit polynomial kernels parameterized by the solution size for any graph H [1, 21].

Therefore, it does make sense to parameterize these problems by structural parameters in the "distance from triviality" spirit, and this is the main focus of this article. To the best of our knowledge, this is an unexplored topic, besides all the literature for Vertex Cover discussed above. Our main result is to identify structural parameters that allow to provide sharp dichotomies for these problems *depending on the forbidden (induced) subgraph* H.

Before presenting our results, we proceed to motivate and define these structural parameters. They are inspired by the following parameter, first introduced by Bulian and Dawar [13, 14] and further studied, for instance, in [2, 22, 29, 31, 41, 42]. For a fixed class of graphs $\mathcal{H}$, the $\mathcal{H}$-*elimination distance* of a graph G, denoted by $\mathsf{ed}_{\mathcal{H}}(G)$, is defined by mimicking the above definition of treedepth and replacing "empty graph" with "a graph in $\mathcal{H}$". The recursive definitions of treedepth and $\mathcal{H}$-elimination distance suggest the notion of *elimination forest*, which is the forest-like process of vertex removals from the considered graph to obtain either an empty graph for treedepth, or a graph in $\mathcal{H}$ for $\mathcal{H}$-elimination distance. Suppose now that $\mathcal{H}$ is defined by the exclusion of a fixed graph H as a subgraph or as an induced subgraph. Formally, let $\mathcal{F}_{\bar{H}}$ (resp. $\mathcal{F}_{\bar{H}}^{\mathsf{ind}}$) be the class of graphs that exclude a fixed graph H as a subgraph (resp. induced subgraph). For this particular case, the notion of $\mathcal{F}_{\bar{H}}$-elimination distance (or $\mathcal{F}_{\bar{H}}^{\mathsf{ind}}$-elimination distance) can be interpreted as a generalization of treedepth where, in the last round of the elimination process, the vertices that do not belong to any occurrence of $\mathcal{H}$ as a subgraph (or induced subgraph) can be deleted "for free". We generalize the notion of $\mathcal{H}$-elimination distance by allowing "free removal" of vertices not contained in a copy of H in *every round* of the elimination process, rather than just the last; we denote the corresponding parameter by $\mathsf{ved}^{+}_{\mathcal{F}_{\bar{H}}}$ (or $\mathsf{ved}^{+}_{\mathcal{F}^{\mathsf{ind}}_{\bar{H}}}$), where "$\mathsf{v}$" stands for the removal of *vertices*, in order to distinguish this parameter from the one defined below (see Section 2 for the formal definitions of these parameters). Our first main result is the following somehow surprising dichotomy, which states that, under the assumption that H is biconnected, whenever H has a non-edge, the problem is unlikely to admit a polynomial kernel. Our result applies to both the induced and non-induced versions of the problem.

▶ **Theorem 1.1.** *Let H be a biconnected graph, let $\lambda \geq 1$ be an integer, and assume that* $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$. *$H$-*Subgraph Hitting *(resp. H-*Induced Subgraph Hitting*) admits a polynomial kernel parameterized by the size of a given vertex set X of the input graph G such that $\mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}(G - X) \leq \lambda$ (resp. $\mathsf{ved}^+_{\mathcal{F}^{\mathsf{ind}}_{\bar{H}}}(G - X) \leq \lambda$) if and only if H is a clique.*

Note that a graph G satisfies $\mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}(G) = 0$ (resp. $\mathsf{ved}^+_{\mathcal{F}^{\mathsf{ind}}_{\bar{H}}}(G) = 0$) if and only if G does not contain H as a subgraph (resp. induced subgraph), so the setting $\lambda = 0$ corresponds to the parameterization by solution size which always admits a polynomial kernel [1, 21]; this is why we assume that $\lambda \geq 1$ in the statement of Theorem 1.1.

Theorem 1.1 shows that the behavior of the considered problems in terms of the existence of polynomial kernels drastically changes as soon as one edge is missing from H (under the biconnectivity assumption, which is needed in the reduction). To the best of our knowledge, this is the first time that such a dichotomy, in terms of H, is found with respect to the existence of polynomial kernels. It is worth mentioning that, with respect to the existence of certain fixed-parameter tractable algorithms parameterized by treewidth, dichotomies of this flavor exist for hitting subgraphs [17], induced subgraphs [43], or minors [4].

The proof of Theorem 1.1 consists of two independent pieces. On the one hand, we need to prove that both problems admit a polynomial kernel when H is a clique (note that, in that case, both problems are equivalent, as any H-subgraph is induced). On the other hand, we need to provide a kernelization lower bound for all other graphs H (cf. Theorem 3.2), and here is where we need the hypothesis that $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$ and, for technical aspects of the reduction, that H is biconnected.

In fact, we provide a kernel that is more general than the one stated in Theorem 1.1. Also, on the negative side, we present another lower bound incomparable to that of Theorem 1.1. For the former, we provide a polynomial kernel, when H is a clique, for a parameter that is more powerful than $\mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}$ (or $\mathsf{ved}^+_{\mathcal{F}^{\mathsf{ind}}_{\bar{H}}}$). This more powerful parameter is somehow inspired by the parameter bridge-depth mentioned before [10], which is a generalization of treedepth in which, in every round of the elimination process, we are allowed to remove subgraphs in each component that are more general than just single vertices. In our setting, it turns out that we can afford to remove vertex sets $T \subseteq V(G)$ that induce connected subgraphs that do not contain H as a subgraph (or induced subgraph) and that are "weakly attached" to the rest of the graph, meaning that each connected component of $G - T$ has at most one neighbor in T. If H is biconnected, it is easily seen that the "candidate" sets T to be removed can be assumed to be connected unions of blocks (biconnected components) of G, and this is why we call this parameter $\mathsf{bed}^+_{\mathcal{F}_{\bar{H}}}$ (or $\mathsf{bed}^+_{\mathcal{F}^{\mathsf{ind}}_{\bar{H}}}$), where "$\mathsf{b}$" stands for the removal of *blocks*. For any two graphs G and H, the following inequalities, as well as the corresponding ones for the induced version, follow easily from the definitions (cf. Section 2):

$$\mathsf{td}(G) \geq \mathsf{ed}_{\mathcal{F}_{\bar{H}}}(G) \geq \mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}(G) \geq \mathsf{bed}^+_{\mathcal{F}_{\bar{H}}}(G). \tag{1}$$

We prove the following result, where K_t denotes the clique on t-vertices, and note that in this case the induced and non-induced versions of the problem coincide.

▶ **Theorem 1.2.** *Let $t \geq 3$ and $\lambda \geq 1$ be fixed integers. The K_t-*Subgraph Hitting *problem admits a polynomial kernel parameterized by the size of a given vertex set X of the input graph G such that $\mathsf{bed}^+_{\mathcal{F}_{\bar{K}_t}}(G - X) \leq \lambda$.*

Note that for $t = 2$, the parameter $\mathsf{bed}^+_{\mathcal{F}_{\bar{K}_t}}$ is exactly treedepth, and therefore Theorem 1.2 can be seen as a far-reaching generalization of the main result of [12], that is, a polynomial kernel for Vertex Cover parameterized by the vertex deletion distance to a graph of bounded

treedepth. Also, note that Equation 1 and Theorem 1.2 imply that the same dichotomy stated in Theorem 1.1 holds if we substitute "$\mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}(G-X) \leq \lambda$" for "$\mathsf{bed}^+_{\mathcal{F}_{\bar{H}}}(G-X) \leq \lambda$", and the same for the induced version.

As for strengthening our hardness results, we present kernelization lower bounds for H-Subgraph Hitting and H-Induced Subgraph Hitting, when H is not a clique, parameterized by the vertex-deletion distance to a graph of constant *treedepth*. By Equation 1, lower bounds for treedepth are stronger than the ones of Theorem 1.1. However, in our next main result, we need a condition on H that is stronger than biconnectivity, namely the non-existence of a *stable cutset*, that is, a vertex separator that induces an independent set.

▶ **Theorem 1.3.** *Let H be a graph on h vertices that is not a clique and that has no stable cutset. H-Subgraph Hitting and H-Induced Subgraph Hitting do not admit a polynomial kernel parameterized by the size of a given vertex set X of the input graph G such that $\mathsf{td}(G-X) = \mathcal{O}(h)$, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.*

Note that, for $t \geq 4$, the graph K_t minus one edge satisfies the conditions of Theorem 1.3. The mere existence of a graph H satisfying the conditions of Theorem 1.3 is remarkable, as it shows that (induced) subgraph hitting problems behave differently than minor hitting problems. Indeed, as mentioned before, it is known [33] that, for *every* finite family $\mathcal{M}$ of connected graphs, the $\mathcal{M}$-Minor Deletion problem admits a polynomial kernel parameterized by the vertex-deletion distance to a graph of constant treedepth.

Dekker and Jansen [18] asked if for every finite set of graphs $\mathcal{M}$, $\mathcal{M}$-Minor Deletion admits a polynomial kernel parameterized by the vertex-deletion distance to a graph with constant $\mathsf{exc}(\mathcal{M})$-elimination distance, where $\mathsf{exc}(\mathcal{M})$ is the class of graphs that excludes all the graphs in $\mathcal{M}$ as a minor. Theorem 1.3 shows that, by Equation 1, for the problems of excluding subgraphs or induced subgraphs, the answer to this question is negative.

Finally, let us mention another consequence of our results. Agrawal et al. [2] proved, among other results, that for every hereditary target graph class $\mathcal{C}$ satisfying some mild assumptions, parameterizing by the vertex-deletion distance to $\mathcal{C}$ and by the $\mathcal{C}$-elimination distance are equivalent from the point of view of the existence of fixed-parameter tractable algorithms. Our results imply, in particular, that the same equivalence does *not* hold with respect to the existence of polynomial kernels in this "distance from triviality" setting, namely for problems defined by the exclusion of (induced) subgraphs.

Organization of the paper. In Section 2 we provide an overview of the main ideas of the kernelization algorithm, which is our main technical contribution. The formal description of the kernel and its analysis, which are quite lengthy, can be found in the full version of the article [11]. In Section 3 we present our hardness results (with some proofs deferred to the full version as well), and in Section 4 we discuss some directions for further research.

2 Overview of the kernelization algorithm

In this section we sketch the main ideas of the kernelization algorithm stated in Theorem 1.2, along with intuitive explanations and the required definitions.

Preliminaries. Given a graph G and $C \subseteq V(G)$, we denote by $N(C) = \bigcup_{v \in C} N(v) \setminus C$ and $G - C = G[V(G) \setminus C]$. Given a graph H, and a subgraph (resp. induced subgraph) F of G, we say that F is a *copy* (resp. *induced copy*) of H if F is isomorphic to H. We say that G is H-free (resp. H-induced free) if there is no copy of H (resp. induced copy) in G. Given two

disjoint subsets $A, B \subseteq V(G)$, we say that there is an edge between A and B if there exists $e \in E(G)$ such that $|e \cap A| = |e \cap B| = 1$. Otherwise, A and B are said to be *anticomplete*. When $\mathcal{P} = \{V_1, \ldots, V_m\}$ is a set of subsets of $V(G)$, we let $V(\mathcal{P}) = \bigcup_{V_i \in \mathcal{P}} V_i$. A *$t$-clique* is a set $K \subseteq V(G)$ such that $G[K]$ is a clique and $|K| = t$. For any integer $n \in \mathbb{N}$, we denote by $[n] = \{1, \ldots, n\}$. We study the following problem(s) for a fixed graph H.

H-SH (for H-Subgraph Hitting)
Input: A graph G.
Objective: Find a set $S \subseteq V(G)$ of minimum size such that $G - S$ is H-free.

We denote by H-ISH (for H-Induced Subgraph Hitting) the variant of the above problem where we impose that $G - S$ is *H-induced* free. We denote by $\mathsf{opt}^H(G)$ the optimal value of the considered problem for G, or simply $\mathsf{opt}(G)$ when H is clear from the context.

Let us now introduce the main graph measures used in this paper.

▶ **Definition 2.1.** *Let H be a fixed graph. For a graph G, define $\mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}(G)$ as*

$$\begin{cases} 0 & \text{if } V(G) = \emptyset, \\ \mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}(G - v) & \text{if } v \text{ is a vertex that is not in any copy of } H, \\ \max_{C_i} \mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}(C_i) & \text{if } G \text{ has connected components } C_1, \ldots, C_c \text{ with } c \geq 2, \\ 1 + \min_{v \in V(G)} \mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}(G - v) & \text{otherwise.} \end{cases}$$

We define $\mathsf{ved}^+_{\mathcal{F}^{\mathsf{ind}}_{\bar{H}}}$ in the same way as $\mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}$, except that we replace "in any copy of H" by "in any induced copy of H". Note that the notation $\mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}$ is motivated by the fact that it corresponds to *vertex* elimination distance, with additional power of removing "free" vertices not in any copy of H. Note also that even though there could be multiple vertices v which satisfy the second criterion, the value is well-defined since it does not matter which one is picked; the second case will apply until all such vertices have been removed. As the case where $H = K_t$ plays an important role in this paper, for the sake of shorter notation we use the shortcut ved^+_t to denote the parameter $\mathsf{ved}^+_{\mathcal{F}_{\bar{K}_t}}$ (or $\mathsf{ved}^+_{\mathcal{F}^{\mathsf{ind}}_{\bar{K}_t}}$, which is the same).

To define the parameters $\mathsf{bed}^+_{\mathcal{F}_{\bar{H}}}$ and $\mathsf{bed}^+_{\mathcal{F}^{\mathsf{ind}}_{\bar{H}}}$, it is convenient to introduce the following definitions (see Figure 1).

▶ **Definition 2.2** (root and pending component)**.** *Given a fixed graph H and a connected graph G, we say that a set $T \subseteq V(G)$ is a* root of G *if*

- *$T \neq \emptyset$, $G[T]$ is connected and H-free, and*
- *for any connected component C of $G - T$, $|N(C) \cap T| = 1$.*

We extend to notion of root to any graph G as follows. For any graph G with connected components $\mathcal{C}$, we say that a set $\mathcal{T} = \{T_C \mid C \in \mathcal{C}\}$ is a root of G *if for any $C \in \mathcal{C}$, T_C is a root of $G[C]$. We define $V(\mathcal{T}) = \bigcup_{C \in \mathcal{C}} T_C$, and $E(\mathcal{T})$ as the set of edges that have both their endpoints inside $V(\mathcal{T})$.*

Given a graph G, a root $\mathcal{T}$ of G, and a vertex $v \in V(\mathcal{T})$, we define the pending component *of v relatively to $\mathcal{T}$, denoted by $C^{\mathcal{T}}(v)$, as the connected component of v in the graph obtained from G by removing all edges $e \subseteq E(\mathcal{T})$. We extend the notation to any subset $Z \subseteq V(\mathcal{T})$ with $C^{\mathcal{T}}(Z) = \bigcup_{v \in Z} C^{\mathcal{T}}(v)$. When the root is clear from context, we use $C(v)$ instead of $C^{\mathcal{T}}(v)$.*

We define an *induced root* in the same way, except that we replace "$G[T]$ is connected and H-free" by "$G[T]$ is connected and H-induced free". Note that any graph admits a root, by taking for example a single vertex (to play the role of T_C) in each connected component.

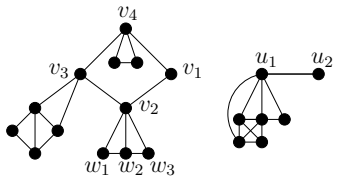


Figure 1 In this example we consider that $H = K_4$, and we denote by C_1 and C_2 the two connected components of G (where $v_1 \in C_1$). Observe that $\mathcal{T} = (T_1, T_2)$ is a root of the depicted graph with $T_1 = \{v_1, v_2, v_3, v_4\}$ and $T_2 = \{u_1, u_2\}$. We have $C(v_1) = \{v_1\}$ and $C(v_2) = \{v_2, w_1, w_2, w_3\}$. Finally, taking $T_1' = T_1 \cup \{w_1\}$ and $\mathcal{T}' = \{T_1', T_2\}$ would not be a root as T_1' is not a root of $G[C_1]$.

▶ **Observation 2.3.** *Let G be a graph and $\mathcal{T}$ be a root of G. For any $v \in V(\mathcal{T})$, there is no edge between $C(v) \setminus \{v\}$ and $V(G) \setminus C(v)$.*

▶ **Definition 2.4.** *Let H be a fixed graph. For a graph G, we define $\mathsf{bed}^+_{\mathcal{F}_{\bar{H}}}(G)$ as*

$$\begin{cases} 0 & \text{if } V(G) = \emptyset, \\ \mathsf{bed}^+_{\mathcal{F}_{\bar{H}}}(G - v) & \text{if } v \text{ is a vertex that is not in any copy of } H, \\ \max_{C_i} \mathsf{bed}^+_{\mathcal{F}_{\bar{H}}}(C_i) & \text{if } G \text{ has connected components } C_1, \ldots, C_c \text{ with } c \geq 2, \\ 1 + \min_{T \subset V(G)} \mathsf{bed}^+_{\mathcal{F}_{\bar{H}}}(G - T) & \text{otherwise, where } T \text{ ranges over all roots of } G. \end{cases}$$

We define $\mathsf{bed}^+_{\mathcal{F}^{\mathsf{ind}}_{\bar{H}}}$ in the same way as $\mathsf{bed}^+_{\mathcal{F}_{\bar{H}}}$, except that we replace "where T ranges over all roots of G" by "where T ranges over all induced roots of G". Again, as the case where $H = K_t$ plays an important role in this paper, for the sake of shorter notation we use the shortcut ved^+_t to denote the parameter $\mathsf{bed}^+_{\mathcal{F}_{\bar{K}_t}}$ (or $\mathsf{bed}^+_{\mathcal{F}^{\mathsf{ind}}_{\bar{K}_t}}$, which is the same). We point out that, to make the definition of $\mathsf{bed}^+_{\mathcal{F}_{\bar{H}}}$ as simple as possible, we allowed T to range over all roots of G. However, as shown in [11, Lemma 6.9], as soon as H is biconnected, there always exists a root that is the connected union of H-(induced-)free blocks of G, hence our choice of notation to differentiate $\mathsf{bed}^+_{\mathcal{F}_{\bar{H}}}$ from $\mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}$.

Given $\lambda \in \mathbb{N}$, let us now define the following variant of the considered problem, where we suppose that we are given as an additional input a modulator (corresponding to set X) to a "simple" graph $G - X$, where the simplicity is captured by bed^+_t being at most λ.

K_t-SHM$^\lambda$ (for K_t-Subgraph Hitting given a modulator to bed^+_t at most λ)
Input: A graph G and a set $X \subseteq V(G)$ such that $\mathsf{bed}^+_t(G[R]) \leq \lambda$, where $R = V(G) \setminus X$.
Objective: Find a set $S \subseteq V(G)$ of minimum size such that for any t-clique Z of G, $S \cap Z \neq \emptyset$.

We denote by K_t-SHM$^\lambda_{\mathsf{p}}$ the associated parameterized decision problem with an additional k in the input, where the goal to decide whether $\mathsf{opt}(G) \leq k$, and the parameter is $|X|$.

In [11, Section 5] we prove our main positive result that we restate here with less details, and which is a reformulation of Theorem 1.2 with the notation introduced in this section:

▶ **Theorem 2.5.** *There is a polynomial kernel for K_t-SHM$^\lambda_{\mathsf{p}}$ of size $\mathcal{O}_{\lambda,t}(|X|^{\delta(\lambda,t)})$ for some function $\delta(\lambda, t)$.*

Let us now present an overview of the techniques used to establish the above result.

Warming up with Vertex Cover**.** As an extreme simplification of our set up, let us consider the case where $t = 2$, corresponding to Vertex Cover, and assume that X is a modulator to the simplest graph class, namely an independent set. Our kernel uses a marking procedure (cf. [11, Definition 5.5]) that corresponds to the following marking algorithm for Vertex Cover. For any $u \in X$, mark up to $|X|+1$ vertices $v \in R$ such that $\{u, v\} \in E(G)$. Let $M \subseteq R$ be the set of marked vertices. Observe the following "packing property" of the marking algorithm: if there exists $v \in R \setminus M$ and $u \in X$ with $\{u, v\} \in E(G)$, then there exists a "packing" $\mathcal{P} \subseteq M$ of $|X| + 1$ vertices such that for any $v' \in \mathcal{P}$, $\{v', u\} \in E(G)$ (the term "packing" may seem inappropriate here, but becomes natural for $t > 2$ as the marking algorithm will mark disjoint sets of vertices instead of distinct vertices). Now, if $R = M$ then $|R| \leq \mathcal{O}(|X|^2)$ and the instance is kernelized. Otherwise, if there exists $v \in R \setminus M$, then define a *reduced instance* as $G' = G - v$ and $k' = k$. Let us sketch why this removing step is safe, as the arguments also correspond to a very simplified version of [11, Lemma 5.14]. The only non-trivial direction is that if (G', k') is a yes-instance, then (G, k) is also a yes-instance. Given a solution Z' of (G', k'), if there exists $u \in X \setminus Z'$ such that $\{u, v\} \in E(G)$, then the packing property implies the existence of the above set $\mathcal{P}$. Thus, we get that Z' *overpays for* u: it contains one extra vertex (in this case, one instead of zero) for each $v' \in \mathcal{P}$ as we must have $\mathcal{P} \subseteq Z'$. This implies that we can restructure Z' into $\tilde{Z} = X$, while ensuring that $|\tilde{Z}| \leq |Z'|$. Now, $\tilde{Z}$ can be easily completed to a solution of G of size k (in this case, by doing nothing). By repeating this reduction rule, we get the kernel of size $\mathcal{O}(|X|^2)$.

Parts, chunks, and conflicts. Let us now point out some important ideas used to lift up the previous kernel for Vertex Cover to K_t-SHM$^{\lambda}_{\mathsf{p}}$. In the previous setting, the key property when proving the safeness of the reduction rule, given a solution Z' of (G', k'), is the following: when "adding back" a non-marked vertex $v \in R \setminus M$ to G', either there exists $u \in X \setminus Z'$ such that Z' overpays for u, or there is no edge $\{u, v\}$ for any $u \in X \setminus Z'$.

Let us now rephrase this key property in the setting of hitting t-cliques using the adapted concepts of part, chunk, and conflict; we will formally define these terms later. When "adding back" a non-marked *part* $V' \subseteq R \setminus M$ to G', we know that either there exists *a chunk* $X' \subseteq X \setminus Z'$ such that Z' overpays for X', or there is no *conflict between* X' *and* V' for any chunk X'. Observe first that, as $G - X$ is now more general than an independent set, we have to consider a packing of "parts" (subsets of vertices of R), meaning that if there is a non-marked part V' that we remove, we now set $k' = k - \mathsf{opt}(G[V'])$. The second difference is the notion of "conflict between X' and V'" that plays the role of "edge $\{u, v\}$". We say that there is no *conflict* between X' and V' if $\mathsf{conf}^t_{X'}(V') = 0$, the condition $\mathsf{conf}^t_{X'}(V') = 0$ being equivalent to the fact that we can pick only $\mathsf{opt}(G[V'])$ vertices in V', while still hitting all t-cliques in $G[X' \cup V']$ (see [11, Definition 5.2] for the formal definition of conf^t). The third difference is the notion of chunk and blocking set. A good starting point when trying to complete a solution Z' of G' to a solution Z of G is that $\mathsf{conf}^t_{X \setminus Z'}(V') = 0$. Indeed, this condition implies that there exists a set $S^\star_{V'}$ of size $\mathsf{opt}(G[V'])$ such that $S^\star_{V'}$ hits all t-cliques in $G[V' \cup (X \setminus Z')]$. Thus, $S^\star_{V'}$ is a good candidate to build a solution $Z = Z' \cup S^\star_{V'}$ of (G, k). Note that this only remains a good starting point, as Z may not be a solution: it could miss cliques using V', $(X \setminus Z')$, and other vertices in $R \setminus V'$. This condition $\mathsf{conf}^t_{X \setminus Z'}(V') = 0$ could be achieved by a marking algorithm that, for any $U \subseteq X$, marks up to $|X| + 1$ parts V' such that $\mathsf{conf}^t_U(V') > 0$, which is a generalization of the previous marking algorithm for Vertex Cover. However, the running time and the number of marked parts by such an algorithm would not be polynomial in $|X|$, as there are too many subsets U to consider.

To overcome this issue, the trick is the notion of *maximum minimal blocking sets*, denoted by mmbs_t (cf. [11, Definition 4.1]), which is a graph parameter for which we skip the definition for the moment. What is important to state about mmbs_t here is that in [11, Theorem 2.6] we prove that there exists a function $\beta : \mathbb{N}^2 \to \mathbb{N}$ such that for every graph G, $\mathsf{mmbs}_t(G) \leq \beta(\mathsf{bed}_t^+(G), t)$. As in the kernelization algorithm we apply this to $G[R]$ and $\mathsf{bed}_t^+(G[R]) \leq \lambda$, we obtain $\mathsf{mmbs}_t(G[R]) \leq \beta(\lambda, t)$. Moreover, [11, Lemma 5.4] implies that the previous marking condition "there exists $U \subseteq X$ such that $\mathsf{conf}_U^t(V') > 0$" is equivalent to "there exists $X' \in \mathcal{X}$ such that $\mathsf{conf}_{X'}^t(V') > 0$", where $\mathcal{X} = \{X' \subseteq X \mid |X'| \leq (t-1)\beta(\lambda, t)$ and X' does not contain a t-clique$\}$ is the set of *chunks*. Observe that, as the chunks have bounded size, the marking algorithm runs in time $\mathcal{O}^\star(|X|^{(t-1)\beta(\lambda,t)})$. The conclusion is that the "triviality" of $G[R]$ ($\mathsf{bed}_t^+(G[R]) \leq \lambda$) implies that $G[R]$ has bounded mmbs_t, which allows to certify the absence of conflict (for any $U \subseteq X$) in polynomial time.

These notions of conflict, chunk, and minimal blocking set were also critical in previous work on kernelization: the notion of conflict was introduced in [30], and bounds on mmbs_2 (for Vertex Cover) have been proved for different triviality measures of $G[R]$ [3, 10, 12, 29, 30]. A first difference here is the study of mmbs_t for K_t-Subgraph Hitting, whose behavior is more complex than Vertex Cover, as discussed below when mentioning the new challenges.

Decreasing $\mathsf{bed}_t^+(G[R])$ and using recursion. Let us now informally describe the main steps of the kernel (see Figure 2). Given a graph G, we denote by $N^t(G) = \{v \in V(G) \mid \nexists\ t$-clique K with $v \in K\}$ the set of vertices of G that do not occur in any copy of K_t, called the *non-K_t-vertices*. Given an input (G, X, k) of K_t-SHM$_\mathsf{p}^\lambda$, we first compute $N^t(G[R])$ and, using the algorithm of [11, Lemma 6.9], a bed_t^+-root $\mathcal{T}$ of $G[R] - N^t(G[R])$, where a *bed_t^+-root* of G is a root $\mathcal{T}$ such that $\mathsf{bed}_t^+(G - V(\mathcal{T})) = \mathsf{bed}_t^+(G) - 1$. We point out that, unlike the case for treedepth or bridge-depth, computing such a root is not straightforward, as one cannot try the a priori exponentially many possible roots to find one that decreases bed_t^+. However, the algorithm of [11, Lemma 6.9] relies on the fact that it is possible to compute in polynomial time a set of size $\mathcal{O}(n)$ that contains a bed_t^+-root. Coming back to the kernel strategy, observe that there may be edges between some $C(v)$ and $N^t(G)$, but not between $C(v)$ and $C(u)$ for $u \neq v$, and that by definition of a bed_t^+-root, $\mathsf{bed}_t^+(G[R']) < \lambda$, where $R' = R - V(\mathcal{T})$. Then, we mark a small (polynomial in $|X|$) set $M(\mathcal{T}, N^t(G[R]), G, X)$ of vertices (cf. [11, Definition 5.8]) of $V(\mathcal{T})$ using the `mark` algorithm (cf. [11, Definition 5.5]). If there exists $v \in V(\mathcal{T}) \setminus M(\mathcal{T}, N^t(G[R]), G, X)$, then we can remove $C(v)$ and decrease k by $\mathsf{opt}(G[C(v)])$ (cf. [11, Lemma 5.14]). Otherwise, $|M(\mathcal{T}, N^t(G[R]), G, X)| = \mathcal{O}(|X|^{f(\lambda,t)})$ for some function f, and thus we can move $M(\mathcal{T}, N^t(G[R]), G, X)$ to the modulator and get a new modulator $X' = X \cup M(\mathcal{T}, N^t(G[R]), G, X)$ whose size is still polynomial in $|X|$. The key point is that $\mathsf{bed}_t^+(G - X') = \mathsf{bed}_t^+(G[R']) < \lambda$, and thus we use induction on λ and make a recursive call to (G, X'), which is an input of K_t-SHM$_\mathsf{p}^{\lambda-1}$, leading to a kernel polynomial in $|X'|$, and thus in $|X|$.

This idea of shrinking the "root" of a decomposition of $G - X$ to decrease the "triviality measure" (here, bed_t^+) and recurse originates in [27], and was used in [12] for treedepth. It was subsequently generalized in [10], where the triviality measure is a parameter called *bridge-depth* and the equivalent of a root is a so-called *tree of bridges* for each connected component of $G[R]$.

New challenges. With respect to the strategies followed in previous work on related topics [10, 12, 18, 29, 30, 32, 33], in our setting we encounter (at least) the following three orthogonal difficulties, for which we have to develop new ideas: dealing with the non-K_t-vertices, dealing with cliques K_t for arbitrary fixed t instead of $t = 2$, and proving that there exists a function β such that for every graph G, $\mathsf{mmbs}_t(G) \leq \beta(\mathsf{bed}_t^+(G))$.

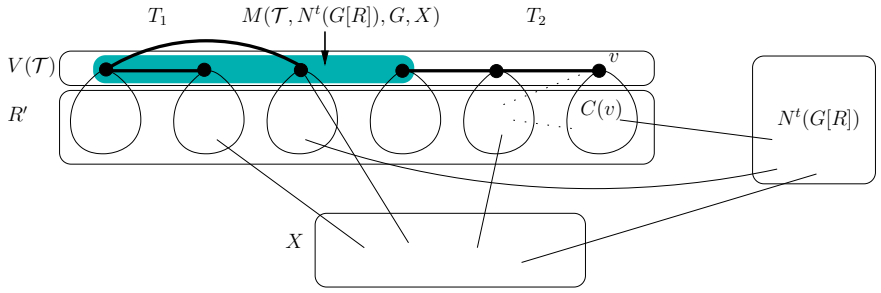


Figure 2 Main steps of the kernel. In this example $\mathcal{T} = \{T_1, T_2\}$ (edges inside T_i are in bold, and dotted edges cannot exist), and there exists a non-marked vertex v, implying that the pending component $C(v)$ will be removed.

The first difficulty is handling vertices of $N^t(G[R])$, which are vertices not belonging to a t-clique in $G[R]$. Indeed, observe that these non-K_t-vertices are "free" for bed_t^+, in the sense that $\mathsf{bed}_t^+(G[R]) = \mathsf{bed}_t^+(G[R] - N^t(G[R]))$. However, these vertices make the structure of $G[R]$ more complicated. Indeed, $\mathcal{T}$ being a root of $G[R] - N^t(G[R])$ implies that for any $T \in \mathcal{T}$ and $v \in T$, there are no edges between $C(v) \setminus \{v\}$ and other vertices in $C(u)$ for $u \in V(\mathcal{T}) \setminus \{v\}$, but there could be edges between $C(v)$ and $N^t(G[R])$. Thus, unlike in [10, 12], we cannot just bound the number of connected components of $G[R]$, and then assume that we have a single root T with simple properties (again, the root being a single vertex in [12], and a tree of bridges in [10]). Typically, here $G[R]$ could have only one connected component, but the nice structure given by $\mathcal{T}$ could be "polluted" by vertices of $N^t(G[R])$. We handle these vertices by considering a packing of "bidimensional" parts (V_i, N_i), where in particular $V_i \subseteq V(\mathcal{T})$ is a clique of size at most $t-1$ and $N_i \subseteq N^t(G[R])$, and we use a kind of generalized "sunflower-like" marking by first creating a maximal packing $\mathcal{P}$ of parts (V_i, N_i), of size at most $|X|+1$, and then recursively marking around each possible $g \in \bigcup N_i$ (see the last line of [11, Definition 5.5]).

The second difficulty is to handle t-cliques instead of edges. Indeed, assume that we just removed a pending component $C(v)$ for some $v \in V(\mathcal{T})$ and defined $k' = k - \mathsf{opt}(G[C(v)])$. Assume also that, given a solution Z' of (G', k'), we have the good starting point $\mathsf{conf}^t_{(X \cup N^t(G[R])) \setminus Z'}(C(v)) = 0$, implying, by the definition of conflict, that there exists a locally optimal solution $S_v^\star$ of $G[C(v)]$ that intersects all t-cliques of $G[C(v) \cup ((X \cup N^t(G[R])) \setminus Z')]$. However, there could also exist "spread" cliques K containing v and using vertices of $((X \cup N^t(G[R])) \setminus Z')$ and $M' \subseteq (V(\mathcal{T}) \setminus \{v\})$. These cliques may be spread across several vertices of $V(\mathcal{T})$, and by definition of a root they cannot use vertices in $C(u) \setminus \{u\}$ for any $u \in M' \cup \{v\}$ (according to Observation 2.3). To take into account the potential conflicts generated by these spread cliques, we perform $t-1$ marking steps (cf. [11, Definition 5.8]), where informally at each step we guess all possible subsets M', with $|M'| \leq t-1$, corresponding to a guessed intersection of a spread clique with previously marked vertices.

The last difficulty is to bound $\mathsf{mmbs}_t(G)$ as a function of $\mathsf{bed}_t^+(G)$ for any graph G. We first need to define the notion of blocking set adapted to our problem. Let EK_t-SH (for *Extended* K_t-Subgraph Hitting) be the problem where given $(G, \mathcal{F})$, where $\mathcal{F}$ is a set of subsets of $V(G)$ such that for any $Z \in \mathcal{F}, 1 \leq |Z| \leq t-1$ and $G[Z]$ is a clique, a solution must intersect all t-cliques of G and all $Z \in \mathcal{F}$. A *blocking set* $\mathcal{B}$ of G is a set of subsets of vertices of G such that $\mathsf{opt}(G, \mathcal{B}) > \mathsf{opt}(G)$, where $\mathsf{opt}(G, \mathcal{B})$ is the minimum size of a solution of the EK_t-SH problem with input $(G, \mathcal{B})$, meaning that any set hitting all t-cliques of G and all $Z \in \mathcal{B}$ cannot be an optimal solution of G for K_t-SH. Then, $\mathsf{mmbs}_t(G)$ is the

maximum size of an inclusion-wise minimal blocking set of G. The "max-min" taste of this definition makes it difficult to handle, but fortunately we will use [11, Property 1] stating that for any β, $\mathsf{mmbs}_t(G) \leq \beta$ is equivalent to the fact that for any blocking set $\mathcal{B}$ of G, there exists $\overline{\mathcal{B}} \subseteq \mathcal{B}$ such that $\overline{\mathcal{B}}$ is still a blocking set of G and $|\overline{\mathcal{B}}| \leq \beta$. We obtain the following upper bound, which requires a considerable amount of technical work.

▶ **Theorem 2.6.** *For any graph G and any integer $t \geq 3$, it holds that $\mathsf{mmbs}_t(G) \leq \beta(\mathsf{bed}_t^+(G), t)$, where $\beta(x,t) = \underbrace{2^{t2^{t2^{\cdots^t}}}}_{x \text{ times}}$ (i.e., $\beta(1) = 2^t, \beta(2) = 2^{t2^t}$, etc.)*

To explain the difficulty of proving the above theorem, let us sketch how such a bound is obtained for Vertex Cover as a function of treedepth, for example in the proof in [12] that, for every graph G, $\mathsf{mmbs}_2(G) \leq 2^{\mathsf{td}(G)}$.

Observe first that for the Vertex Cover problem, a blocking set $\mathcal{B}$ is a set of singletons, which we consider as a subset of vertices, such that any vertex cover S containing $\mathcal{B}$ is not optimal. Let us now use [11, Property 1] and consider a blocking set $\mathcal{B}$ of G, and let us show that there exists $\overline{\mathcal{B}} \subseteq \mathcal{B}$ such that $\overline{\mathcal{B}}$ is still a blocking set of G and $|\overline{\mathcal{B}}| \leq 2^{\mathsf{td}(G)}$. Consider a graph G and a "root" v of a treedepth decomposition of G, meaning that $\mathsf{td}(G-v) < \mathsf{td}(G)$. Let us consider the most complex case where there exists an optimal solution using v, another avoiding v, and $v \notin \mathcal{B}$. It is not difficult to prove that, as $\mathcal{B}$ is a blocking set of G, $\mathcal{B}_1 = \mathcal{B}$ is a blocking set of $G_1 := G - v$, and $\mathcal{B}_2 = \mathcal{B} \setminus N(v)$ is a blocking set of $G_2 := G - (\{v\} \cup N(v))$. Thus, as for any $i \in [2]$, $\mathsf{td}(G_i) < \mathsf{td}(G)$, by induction we get that there exists $\overline{\mathcal{B}}_i \subseteq \mathcal{B}_i$ such that $\overline{\mathcal{B}}_i$ is a blocking set of G_i and $|\overline{\mathcal{B}}_i| \leq 2^{\mathsf{td}(G)-1}$. As $\overline{\mathcal{B}}_1 \cup \overline{\mathcal{B}}_2$ is a blocking set of G, we get the desired bound. The problem when lifting this idea to K_t-Subgraph Hitting instead of Vertex Cover is that, when considering an optimal solution S that avoids a root v, we do not know which vertex the solution S will pick in $N(v)$. This is the reason for which we consider the more general version of the problem, namely EK_t-SH, to encode the fact that a solution of $(G, \emptyset)$ avoiding a root v must be a solution of $(G-v, \mathsf{pr}_v^t(V(G) \setminus \{v\}))$, where given two disjoint sets $A, B \subseteq V(G)$, $\mathsf{pr}_A^t(B) = \{K \cap B \mid K \text{ is a } t\text{-clique in } G[A \cup B] \text{ and } K \cap A \neq \emptyset \text{ and } K \cap B \neq \emptyset\}$. We also need to define the corresponding generalized notion of blocking set of an instance $(G, \mathcal{F})$ of EK_t-SH (cf. [11, Definition 4.1]), and not only of a graph G. Moreover, we have to keep track of the structure of $\mathcal{F}$, as there is no hope to bound $\mathsf{mmbs}_t(G, \mathcal{F})$ as a function of $\mathsf{bed}_t^+(G)$ for an arbitrary set $\mathcal{F}$. Indeed, for example, let G_ℓ be a chain of triangles of length ℓ, as depicted in Figure 4. We have $\mathsf{mmbs}_2(G_\ell) \geq \ell$, as if we let $\mathcal{B}$ be the set of top vertices of the ℓ triangles, then it can be easily seen that $\mathcal{B}$ is a minimal blocking set of G_ℓ with $|\mathcal{B}| = \ell$. Now, take $\mathcal{F} = E(G_\ell)$, $t = 4$, and observe that any solution of instance $(G_\ell, \mathcal{F})$ of EK_t-SH is a vertex cover of G_ℓ, and thus $\mathcal{B}$ is also a minimal blocking set of size ℓ for the input $(G_\ell, \mathcal{F})$ with $t = 4$, while $\mathsf{bed}_t^+(G_\ell) = 0$ as G_ℓ is K_4-free.

We resolve this problem by proving bounds on $\mathsf{mmbs}_t(G, \mathcal{F})$ only for a special type of instances that we call *clean*, which are pairs $(G, \mathcal{F})$ such that $\mathsf{opt}(G, \mathcal{F}) = \mathsf{opt}(G)$. The first main difficulty is that, when starting with a blocking set $\mathcal{B}$ of $(G, \mathcal{F})$, reducing to a graph G' with $\mathsf{bed}_t^+(G') < \mathsf{bed}_t^+(G)$ requires to remove the entire root $\mathcal{T}$ of $G - N^t(G)$, instead of just one vertex as in the treedepth case. As $|V(\mathcal{T})|$ may be arbitrarily large, we need to prove (see [11, Lemma 4.8]) that it is enough to "zoom in" on a small number of subgraphs (pending components here), allowing us to extract (by induction) a small blocking set only in each of these subgraphs. The second main difficulty is to ensure that we can reduce via recursion to smaller *clean* instances. Indeed, even if we initially consider a clean instance $(G, \emptyset)$, and even in the favorable case where $\mathcal{T}$ is just one vertex v (as in treedepth), we have to consider

the case where there exists an optimal solution of G using v, another avoiding v, and $v \notin \mathcal{B}$, where $\mathcal{B}$ is fixed blocking set from which we try to extract a small one. However, observe that optimal solutions avoiding v are optimal solutions of $(G-v, \mathsf{pr}_v^t(V(G)\setminus\{v\}))$, and that $\mathsf{opt}(G-v, \mathsf{pr}_v^t(V(G)\setminus\{v\})) = \mathsf{opt}(G) = 1+\mathsf{opt}(G-v)$ (the last equality holds since there are optimal solutions taking v). Thus, we observe that this situation leads to a *non*-clean instance $(G-v, \mathsf{pr}_v^t(V(G)\setminus\{v\}))$, but "almost clean" as $\mathsf{opt}(G-v, \mathsf{pr}_v^t(V(G)\setminus\{v\})) = \mathsf{opt}(G-v)+1$. We treat these almost clean instances in [11, Lemma 4.5], which is the main cause of the huge growth of function β in the final bound $\mathsf{mmbs}_t(G) \leq \beta(\mathsf{bed}_t^+(G), t)$ given in Theorem 2.6. As this bound directly reverberates both in the running time and the size of the kernel (see [11, Theorem 5.16], where $\delta(\lambda, t)$ is dominated by $\beta(\lambda, t)$ for an instance (G, X) of K_t-SHM where $\mathsf{bed}_t^+(G-X) \leq \lambda$), improving this bound is crucial in order to improve the kernel size. In this direction, we provide in [11, Lemma 4.12] a significantly better upper bound for minimal blocking sets of the K_t-SH problem as a function of td instead of bed_t^+. As the proof technique is also different, we believe that this result might be of independent interest.

Finally, let us mention that in several earlier papers on kernelization using structural parameterizations, it was also crucial to understand the maximum size of an inclusion-minimal set with additional requirements on the solution to a vertex-deletion problem, for which no optimal solution can satisfy all additional requirements; these correspond to variations on the notion of blocking sets. They were explored for the problems of hitting forbidden connected minors in graphs of bounded treedepth [33], and for hitting cycles in graphs of bounded elimination distance to a forest [18], both of which lead to super-exponential bounds in terms of the graph parameter.

3 Hardness results

In this section we present two reductions from CNF-SAT, and to transfer the non-existence of polynomial kernels (under reasonable complexity assumptions), we use the notion of *polynomial parameter transformation*, introduced by Bodlaender, Thomassé, and Yeo [9]. A polynomial parameter transformation from a parameterized problem P to a parameterized problem Q is an algorithm that, given an instance (x, k) of P, computes in polynomial time an equivalent instance (x', k') of Q such that k' is bounded by a polynomial depending only on k. It follows easily from the definition that if P does not admit a polynomial generalized[1] kernel under some complexity assumption, then the same holds for Q. The complexity hypothesis in the following proposition builds on the results by Fortnow and Santhanam [26].

▶ **Proposition 3.1** (Dell and van Melkebeek [19])**.** CNF-SAT *does not admit a polynomial generalized kernel parameterized by the number of variables of the input formula, unless* $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

We are ready to present our main hardness result, which is inspired by other reductions for related problems [10, 16, 18, 25, 32]. The crucial issue of this reduction, and its main conceptual novelty, is the following fact: when H is not a clique, the intersection of an occurrence of H with (a subgraph of) $G-X$ may be *disconnected*. We exploit this fact by

[1] A *generalized kernel* for a parameterized problem P, also called sometimes *compression* in the literature [15], is a polynomial-time algorithm reducing any instance (x, k) of P to an equivalent instance (x', k') with size bounded by a function $f(k)$ depending only on k of a fixed but potentially *different* parameterized problem Q. A generalized kernel is *polynomial* if $f(k)$ is a polynomial function.

creating clause gadgets with large minimal blocking sets whose elements are disconnected (in Figure 3, each pair of consecutive non-adjacent vertices u, v is an element of a blocking set), and this results in clause gadgets behaving as "chains" where the propagation of information (that is, the vertices picked by the solution) is done without needing edges connecting the elements of the chain (thus, in a "wireless" fashion), easily implying that the corresponding parameter in $G - X$ is bounded by a constant.

▶ **Theorem 3.2.** *Let H be a biconnected graph that is not a clique. The H-*Subgraph Hitting *(resp. H-*Induced Subgraph Hitting*) problem does not admit a polynomial kernel parameterized by the size of a given vertex set X of the input graph G such that $\mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}(G - X) \leq 1$ (resp. $\mathsf{ved}^+_{\mathcal{F}^{\mathsf{ind}}_{\bar{H}}}(G - X) \leq 1$), unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.*

Proof. We present a polynomial parameter transformation from the CNF-SAT problem parameterized by the number of variables, which does not admit a polynomial generalized kernel by Proposition 3.1, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. We present our reduction for the H-Subgraph Hitting problem, and at the end of the proof we observe that the same reduction applies to H-Induced Subgraph Hitting as well.

Given a CNF-SAT formula ϕ with n variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$, we proceed to construct in polynomial time an instance G of H-Subgraph Hitting, together with a set $X \subseteq V(G)$ with $\mathsf{ved}^+_H(G - X) \leq 1$ and $|X| = |V(H)| \cdot n$, such that ϕ is satisfiable if and only if G contains a solution of H-Subgraph Hitting of size at most $n - m + \sum_{j=1}^{m} c_j$, where c_j denotes the number of literals in clause C_j. Since $|V(H)|$ is a constant, this would indeed define a polynomial parameter transformation from CNF-SAT parameterized by the number of variables to H-Subgraph Hitting parameterized by the size of a given vertex set X of the input graph G such that $\mathsf{ved}^+_H(G - X) \leq 1$.

For each variable x_i, we add a disjoint copy of H to G. We call such a copy of H the *i-variable-copy* of H. For each clause C_j, we add $c_j - 1$ disjoint copies of H to G, and we order them arbitrarily from 1 to $c_j - 1$. Moreover, we add two new vertices u_j^0 and $v_j^{c_j}$ to G. We call each of these $c_j - 1$ copies of H a *j-clause-copy* of H. Note that, so far, we have introduced $n - m + \sum_{j=1}^{m} c_j$ disjoint copies of H in G.

We now proceed to interconnect these copies of H according to ϕ. Since H is a biconnected graph that is not a clique (hence, it is 2-connected), it follows that $|V(H)| \geq 4$. Thus, in particular there exist two *non-adjacent* vertices $u, v \in V(H)$ and another vertex $w \in V(H)$ distinct from u and v. Let $H' = H - \{u, v, w\}$. (Even if it is not critical for the proof, note that $|V(H')| \geq 1$.) Let also z^+ and z^- be two distinct vertices of H (not necessarily different from u, v, w). We will use the copies of these vertices in the variable-copies and clause-copies of H to interconnect them in G. To this end, for three distinct vertices $a, b, c \in V(G)$ and a subgraph F of G isomorphic to H' not containing any of a, b, c, by *adding an $(a, b, c, V(F))$-copy of H to G* we mean the operation of, starting from $G[\{a, b, c\} \cup V(F)]$, adding the missing edges to complete a copy of H, where vertex a (resp. b, c) of G plays the role of vertex w (resp. u, v) of H, and F plays the role of H', with a fixed isomorphism that we suppose to have at hand.

For each clause C_j of ϕ, consider an arbitrary ordering of its literals as $\ell_1, \ldots, \ell_{c_j}$, and recall that G contains $c_j - 1$ ordered disjoint j-clause-copies of H together with two extra vertices u_j^0 and $v_j^{c_j}$. For $i \in [c_j]$, we add a new copy of H' to G, which we denote by F_j^i. For $i \in [c_j - 1]$, let u_j^i and v_j^i be the copies of vertices u and v of H, respectively, in the i-th j-clause-copy of H. For $i \in [c_j]$, if literal ℓ_i of clause C_j corresponds to a positive (resp. negative) occurrence of a variable x_p, let z be the copy of vertex $z^+ \in V(H)$ (resp. $z^- \in V(H)$) in the p-variable-copy of H. Then we add a $(z, u_j^{i-1}, v_j^i, V(F_j^i))$-copy of H to G,

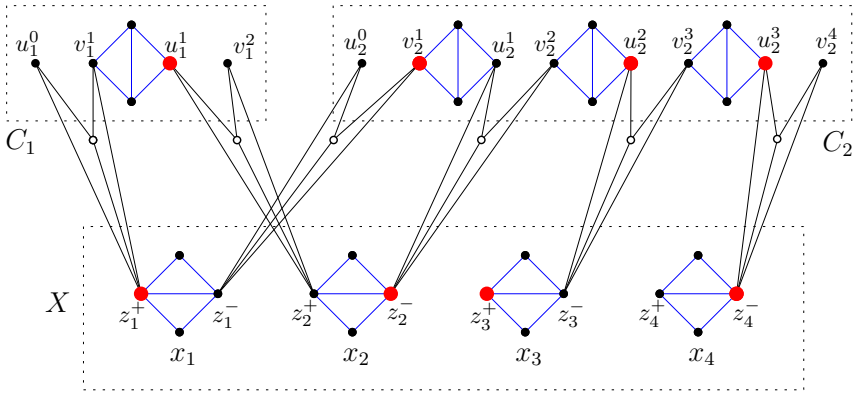


Figure 3 Example of the construction of graph G in the proof of Theorem 3.2 for H-Subgraph Hitting. In this example, H is the diamond (that is, K_4 minus one edge), u and v are the only pair of non-adjacent vertices in H, and w is any other vertex. The construction corresponds to a CNF-SAT formula ϕ consisting of two clauses $C_1 = (x_1 \vee x_2)$ and $C_2 = (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4)$, and the satisfying assignment $\alpha(x_1) = 1$, $\alpha(x_2) = 0$, $\alpha(x_3) = 1$, and $\alpha(x_4) = 0$. The variable-copies and clause-copies of H are depicted in blue, the vertices in the copies of $H' = H - \{u, v, w\}$ (which is a single vertex) are the white ones, and the vertices in the solution S are the large red ones. Note that clause C_2 is satisfied by both $\bar{x}_2$ and $\bar{x}_4$; in the example we have taken $\bar{x}_2$ as the satisfying literal.

and we call such a copy of H a *transversal-copy* of H, denoted by H_j^i. We define $X \subseteq V(G)$ to be the union of the vertex sets of all the variable-copies of H, and note that $|X| = |V(H)| \cdot n$. This completes the construction of G and X, which is illustrated in Figure 3.

In the next claim we prove one of the properties claimed in the statement of the theorem.

▷ Claim 3.3. $\mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}(G - X) = 1$ and $\mathsf{ved}^+_{\mathcal{F}^{\mathsf{ind}}_{\bar{H}}}(G - X) = 1$.

Proof. Note that each connected component of $G - X$ corresponds to the clause-copies of H associated with a clause C_j and isolated vertices, together with the copies of H' between those j-clause-copies of H and isolated vertices. By construction of G, each such a copy of H', say F_j^i, has at most two neighbors in $G - X$, namely vertices u_j^{i-1} and v_j^i. If an occurrence of H as a subgraph in $G - X$, say F, contained a vertex of F_j^i, since $|V(H')| = |V(H)| - 3$, necessarily F contains at least one of u_j^{i-1} and v_j^i, and at least one more vertex in the $(i-1)$-th or i-th j-clause-copies of H. Thus, u_j^{i-1} or v_j^i is a separator of size one of F, contradicting the hypothesis that H is biconnected.

That is, we have proved that no vertex in a copy F_j^i of H' in $G - X$ is contained in an occurrence of H as a subgraph, hence neither as an induced subgraph. Therefore, those vertices can be removed while preserving the value of $\mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}(G - X)$. Formally,

$$\mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}(G - X) = \mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}(G - X - \bigcup_{j=1}^{m} \bigcup_{i=1}^{c_j} V(F_j^i)),$$

and the same holds for $\mathsf{ved}^+_{\mathcal{F}^{\mathsf{ind}}_{\bar{H}}}(G - X)$. To conclude the proof of the claim, it suffices to note that $G - X - \bigcup_{j=1}^{m} \bigcup_{i=1}^{c_j} V(F_j^i)$ consists of a disjoint union of clause-copies of H and isolated vertices, and using the fact that $\mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}$ (resp. $\mathsf{ved}^+_{\mathcal{F}^{\mathsf{ind}}_{\bar{H}}}$) of a disconnected graph is the maximum of $\mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}$ (resp. $\mathsf{ved}^+_{\mathcal{F}^{\mathsf{ind}}_{\bar{H}}}$) over its connected components, by removing one arbitrary vertex from each such a copy of H we get that $\mathsf{ved}^+_{\mathcal{F}_{\bar{H}}}(G - X) = 1$ and $\mathsf{ved}^+_{\mathcal{F}^{\mathsf{ind}}_{\bar{H}}}(G - X) = 1$. ◁

We now claim that ϕ is satisfiable if and only if G contains a solution $S \subseteq V(G)$ of H-Subgraph Hitting of size at most $n - m + \sum_{j=1}^{m} c_j$.

Suppose first that ϕ is satisfiable and let $\alpha\colon \{x_1, \ldots, x_n\} \to \{0,1\}$ be a satisfying assignment of the variables. We define a set $S \subseteq V(G)$ of size $n - m + \sum_{j=1}^{m} c_j$ as follows (cf. the red vertices in Figure 3). For each variable x_i, if $\alpha(x_i) = 1$ (resp. $\alpha(x_i) = 0$), we add to S the copy of z^+ (resp. z^-) in the i-variable-copy of H, which we denote by z_i^+ (resp. z_i^-). For each clause C_j, let ℓ_{s_j} be a literal in C_j that is satisfied by the assignment α. Recall that the $c_j - 1$ j-clause-copies of H are ordered (arbitrarily) from 1 to $c_j - 1$. We add to S the vertex set

$$\{v_j^i \mid 1 \leq i \leq s_j - 1\} \cup \{u_j^i \mid s_j \leq i \leq c_j - 1\}.$$

In words, we add to S the copy of vertex v in all the j-clause-copies of H from 1 to $s_j - 1$, and the copy of vertex u in all the j-clause-copies of H from s_j to $c_j - 1$. Note that $|S| = n - m + \sum_{j=1}^{m} c_j$, and it remains to prove that $G - S$ does not contain H as a subgraph. Note that each variable-copy and clause-copy of H contains exactly two vertices that have neighbors outside of that copy – let us call these vertices *boundary vertices* of that copy – , and that S contains exactly one of these two boundary vertices for each of these copies of H. Hence, since H is biconnected, no occurrence of H in $G - S$ can contain a non-boundary vertex in a variable-copy or clause-copy of H.

Moreover, there do not exist two pairs of integers (i_1, j_1) and (i_2, j_2), with $i_1 \neq i_2$ or $j_1 \neq j_2$, such that there exists an occurrence F of H in $G - S$ with $F \cap (V(H_{j_1}^{i_1}) \setminus X) \neq \emptyset$ and $F \cap (V(H_{j_2}^{i_2} \setminus X) \neq \emptyset$. Indeed, if such (i_1, j_1) and (i_2, j_2) existed, then, as $|N(V(H_j^i) \setminus X) \cap X| = 1$ for any two indices i, j, and as F cannot contain a non-boundary vertex of a variable-copy of H, there would exist $z \in X$ such that $F \cap X = \{z\}$, implying that z is a separator of F, contradicting the 2-connectivity of H.

Thus, if an occurrence of H in $G - S$ existed, say F, then the above discussion and the construction of G imply that F should be one of the transversal-copies of H. But such an F cannot exist in $G - S$ by the choice of S: either S contains one of the boundary vertices in the two j-clause-copies intersected by F for some $j \in [m]$ or, if it is not the case, then S contains the vertex in a variable-copy of H corresponding to the literal that satisfies clause C_j.

Conversely, let $S \subseteq V(G)$ be a solution of H-Subgraph Hitting of size $n - m + \sum_{j=1}^{m} c_j$. Since G contains $|S|$ disjoint variable-copies and clause-copies of H, necessarily S consists of exactly one vertex in each of these copies. Since the boundary vertices in each of the variable-copies and clause-copies of H are the only vertices with neighbors outside of the corresponding copy, we may assume that all the vertices in S are boundary vertices. We define from S a satisfying assignment α of ϕ as follows. If S contains z_i^+ (resp. z_i^-) we set $\alpha(x_i) = 1$ (resp. $\alpha(x_i) = 0$). Let us verify that α indeed satisfies all the clauses of ϕ. Consider an arbitrary clause C_j with c_j literals, and note that S contains $c_j - 1$ vertices in the j-clause-copies of H. Therefore, since by construction no two transversal-copies intersect a clause-copy in a common vertex, there exists $s_j \in [c_j]$ such that the s_j-th transversal-copy of H associated with C_j, say F, is *not* hit by a vertex in a clause-copy of H. Thus, since $S \cap V(F) \neq \emptyset$, necessarily there exists an index $i \in [n]$ such that $S \cap V(F)$ is equal to either z_i^+ or z_i^-, and thus the defined assignment of variable x_i satisfies clause C_j.

To conclude the proof, we claim that the same reduction presented above proves the hardness result for the H-Induced Subgraph Hitting problem. Indeed, in the proof of the equivalence between the satisfiability of ϕ and the existence of a solution S of H-Subgraph Hitting with the appropriate size, all that is relevant to the proof are the variable-copies, clause-copies, and transversal-copies of H. As all these occurrences of H in G occur as induced subgraphs, the same reduction implies the non-existence of polynomial kernels for H-Induced Subgraph Hitting. ◀

In Theorem 1.3 we replace the condition "$\mathsf{ved}^+_H(G-X) \leq 1$" of Theorem 3.2 with the condition that $\mathsf{td}(G-X)$ is bounded by a constant. However, in the proof of Theorem 1.3 we need an extra condition on H stronger than biconnectivity, namely the non-existence of a stable cutset. The reduction in the proof of Theorem 1.3 follows essentially the same lines as the one described in Theorem 3.2, but in order to guarantee that $\mathsf{td}(G-X)$ is bounded by a constant (depending on H), we need to be more careful. Namely, in the interconnection among the variable and clause gadgets, now we cannot afford to add a distinct gadget for each literal in a clause, as it was the case for the copies of H' in the proof of Theorem 3.2 (cf. the white vertices in Figure 3). Indeed, these copies of H' can be removed "for free" when dealing with ved^+_H, but it is not the case anymore when dealing with treedepth, as they may blow up the value of $\mathsf{td}(G-X)$. In a nutshell, we overcome this issue by "reusing" these copies of a (now, carefully chosen) subgraph $H' \subseteq H$ for all the literals of the same clause. However, having a single common H' for each clause may create undesired occurrences of H (other than the variable-copies, clause-copies, and transversal-copies, as we wish), and preventing the existence of these undesired copies is the reason why we need an assumption on H stronger than biconnectivity. The proof of Theorem 1.3 can be found in the full version [11].

4 Further research

In this paper we studied the existence of polynomial kernels for the H-Subgraph Hitting and H-Induced Subgraph Hitting problems under structural parameterizations, namely parameterized by the size of a modulator to a graph class $\mathcal{C}$ that has a "simple structure". Our main achievement is the identification of two arguably natural graph parameters $\mathsf{ved}^+_{\mathcal{F}_H}$ and $\mathsf{bed}^+_{\mathcal{F}_H}$ (or $\mathsf{ved}^+_{\mathcal{F}^{\mathsf{ind}}_H}$ and $\mathsf{bed}^+_{\mathcal{F}^{\mathsf{ind}}_H}$ for the induced version) that allowed us to prove complexity dichotomies in terms of the forbidden graph H. Our results pave the way to a systematic investigation of this topic, where we identify the following avenues for further research.

Getting rid of the hypothesis on H. In our hardness results we need additional assumptions on H, mainly that H is biconnected in Theorem 3.2. Observe that the requirement that H is connected is necessary to obtain polynomial kernels. Indeed, when H is the union of a K_5 and a $K_{1,3}$, it is known [31] that H-Subgraph Hitting is para-NP-hard, even for $\mathsf{ed}_{\mathcal{H}} = 0$. Moreover, when H is a non-edge, H-Induced Subgraph Hitting parameterized by vertex cover (which is a larger parameter than $\mathsf{ed}_{\mathcal{H}}$) is equivalent to maximum clique parameterized by vertex cover, which does not admit a polynomial kernel under standard complexity assumptions [7]. Thus, it is natural to wonder whether the biconnectivity hypothesis could be replaced by just connectivity.

Improving the degree of the kernel. The degree of our polynomial kernel depends on the size t of the excluded clique and on the value λ of the promised upper bound $\mathsf{bed}^+_t(G-X) \leq \lambda$. Namely, as stated in Theorem 2.5, the kernel has size $\mathcal{O}_{\lambda,t}(|X|^{\delta(\lambda,t)})$, where function δ mainly depends on the upper bound on mmbs_t given in Theorem 2.6. This function behaves as a tower of exponents in t of height λ. Hence, improving the bound on mmbs_t directly translates to an improvement of the kernel size. We did obtain such an improvement if instead of assuming that $\mathsf{bed}^+_t(G-X) \leq \lambda$, one assumes that $\mathsf{td}(G-X) \leq \lambda$, namely with a function $\lambda^\lambda \cdot 2^{\lambda^2}$; see [11, Lemma 4.12]. We leave as an open problem to obtain improved upper bounds for mmbs_t in terms of ved^+_t and bed^+_t.

Computing the modulator. In our kernelization algorithm we assume that we are *given* a modulator, namely a set $X \subseteq V(G)$ such that $\mathsf{bed}_t^+(G - X) \leq \lambda$. Note that this hypothesis appears also in the related work dealing with Feedback Vertex Set [18]. Obtaining a constant factor or even $\mathsf{poly}(\mathsf{opt})$-approximation of the modulator in polynomial time for fixed t and λ, which will be enough for our kernelization algorithm (note that minimizing its size is NP-hard [38]), remains an interesting direction. One may start with the probably simpler cases of a modulator to bounded $\mathcal{F}_{\bar{H}}$-elimination distance or bounded ved_t^+.

Finding the right measure. The focus of this article is on obtaining kernelization dichotomies as a function of the forbidden (induced) subgraph H. Of course, it is also relevant to characterize, for a fixed graph H, which is the most general (monotone or hereditary) target family $\mathcal{C}_H$ such that H-(Induced) Subgraph Hitting admits a polynomial kernel parameterized by the size of a modulator to a graph in $\mathcal{C}_H$. Needless to say, solving this general problem seems quite challenging. Indeed, even the case of Vertex Cover, that is, $H = K_2$, is far from being well understood for monotone or hereditary target graph classes, as for instance the only known polynomial kernel for Vertex Cover parameterized by a modulator to a bipartite graph (i.e., an odd cycle transversal) is randomized and relies on quite powerful tools [36, 37]. One may hope that larger cliques allow for simpler characterizations, the natural first candidate being the case where H is a triangle. Let $\mathcal{C}_\Delta$ be the, say, hereditary target graph class that we want to characterize. Following the approach of [10] that characterized the target *minor-closed* graph classes for Vertex Cover, one may hope that K_3-Subgraph Hitting admits a polynomial kernel parameterized by a modulator to $\mathcal{C}_\Delta$ if and only if the graphs in $\mathcal{C}_\Delta$ have bounded mmbs_3. With no extra assumption on $\mathcal{C}_\Delta$, this property is probably false due to the results of Hols, Kratsch, and Pieterse [29], but we conjecture that it is true if we ask $\mathcal{C}_\Delta$ to be hereditary and closed under disjoint union, even for hitting K_t for every $t \geq 3$, replacing mmbs_3 by mmbs_t. Toward an eventual proof of this conjecture, having unbounded minimal blocking sets seems to permit a generic reduction to obtain the lower bound, in the spirit of the one of Theorem 3.2 or any similar one in previous work [10, 16, 18, 25, 32]. Indeed, Hols, Kratsch, and Pieterse [29, Thm 1.1] show that for Vertex Cover, lower bounds on kernel sizes directly follow from lower bounds on mmbs_2. However, the opposite direction seems way more challenging. In [10], this fact was established for Vertex Cover and minor-closed target classes via the notion of bridge-depth by proving, in particular, that there is a *single* minor-obstruction for having large maximum minimal blocking sets, namely the chains of triangles (cf. left part of Figure 4). Unfortunately, we cannot hope the same nice behavior for K_3-Subgraph Hitting and monotone or hereditary graph classes, as chains of triangles are still an obstruction in this setting, but there exist other incomparable ones, as depicted in Figure 4.

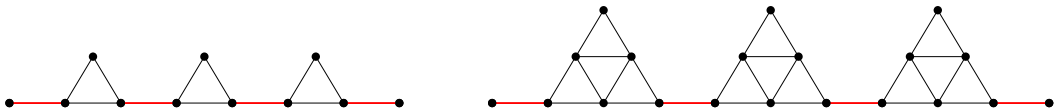


Figure 4 Two chains of length three incomparable with respect to the (induced) subgraph relation. In both graphs, it can be verified that the set of four red thicker edges is a minimal blocking set.

Finally, in the ambitious quest for finding the appropriate measures that characterize the hereditary or monotone classes $\mathcal{C}_t$ for which K_t-Subgraph Hitting admits a polynomial kernel parameterized by the size of a modulator X to $\mathcal{C}_t$, we hope that the techniques we developed to provide a polynomial kernel for the case $\mathsf{bed}_t^+(G - X) \leq \lambda$ will play an

important role. A natural attempt to generalize bed_t^+ to a more powerful measure is to relax, or even drop, the "weak attachment" condition on the sets to be removed in every round of the elimination process. This raises new challenges for obtaining a polynomial kernel that do not seem easy to overcome, at least with the existing tools in this area.

References

1 Faisal N. Abu-Khzam. A kernelization algorithm for d-Hitting Set. *Journal of Computer and System Sciences*, 76(7):524–531, 2010. doi:10.1016/j.jcss.2009.09.002.

2 Akanksha Agrawal, Lawqueen Kanesh, Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Deleting, Eliminating and Decomposing to Hereditary Classes Are All FPT-Equivalent. In *Proc. of the 32st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1976–2004, 2022. doi:10.1137/1.9781611977073.79.

3 Júlio Araújo, Marin Bougeret, Victor A. Campos, and Ignasi Sau. Parameterized complexity of computing maximum minimal blocking and hitting sets. *Algorithmica*, 85(2):444–491, 2023. doi:10.1007/s00453-022-01036-5.

4 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Hitting minors on bounded treewidth graphs. IV. an optimal algorithm. *SIAM Journal on Computing*, 52(4):865–912, 2023. doi:10.1137/21m140482x.

5 Hans L. Bodlaender. Kernelization, exponential lower bounds. In *Encyclopedia of Algorithms*, pages 1013–1017. Springer, 2016. doi:10.1007/978-1-4939-2864-4_521.

6 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.

7 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Cross-composition: A new technique for kernelization lower bounds. In *Proc. of the 28th 28th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 9 of *LIPIcs*, pages 165–176, 2011. doi:10.4230/LIPIcs.STACS.2011.165.

8 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014. doi:10.1137/120880240.

9 Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theoretical Computer Science*, 412(35):4570–4578, 2011. doi:10.1016/j.tcs.2011.04.039.

10 Marin Bougeret, Bart M. P. Jansen, and Ignasi Sau. Bridge-depth characterizes which minor-closed structural parameterizations of vertex cover admit a polynomial kernel. *SIAM Journal on Discrete Mathematics*, 36(4):2737–2773, 2022. doi:10.1137/21m1400766.

11 Marin Bougeret, Bart M. P. Jansen, and Ignasi Sau. Kernelization dichotomies for hitting subgraphs under structural parameterizations. *CoRR*, abs/2404.16695, 2024. arXiv:2404.16695.

12 Marin Bougeret and Ignasi Sau. How much does a treedepth modulator help to obtain polynomial kernels beyond sparse graphs? *Algorithmica*, 81(10):4043–4068, 2019. doi:10.1007/s00453-018-0468-8.

13 Jannis Bulian and Anuj Dawar. Graph isomorphism parameterized by elimination distance to bounded degree. *Algorithmica*, 75(2):363–382, 2016. doi:10.1007/s00453-015-0045-3.

14 Jannis Bulian and Anuj Dawar. Fixed-parameter tractable distances to sparse graph classes. *Algorithmica*, 79(1):139–158, 2017. doi:10.1007/s00453-016-0235-7.

15 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

16 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. On the hardness of losing width. *Theory of Computing Systems*, 54(1):73–82, 2014. doi:10.1007/s00224-013-9480-1.

17 Marek Cygan, Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. Hitting forbidden subgraphs in graphs of bounded treewidth. *Information and Computation*, 256:62–82, 2017. doi:10.1016/j.ic.2017.04.009.

18 David Dekker and Bart M. P. Jansen. Kernelization for Feedback Vertex Set via Elimination Distance to a Forest. In *Proc. of the 48th International Workshop Graph-Theoretic Concepts in Computer Science (WG)*, volume 13453 of *LNCS*, pages 158–172, 2022. doi:10.1007/978-3-031-15914-5_12.

19 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *Journal of the ACM*, 61(4):23:1–23:27, 2014. doi:10.1145/2629620.

20 Michael R. Fellows, Lars Jaffke, Aliz Izabella Király, Frances A. Rosamond, and Mathias Weller. What Is Known About Vertex Cover Kernelization? In *Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, volume 11011 of *LNCS*, pages 330–356, 2018. doi:10.1007/978-3-319-98355-4_19.

21 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006. doi:10.1007/3-540-29953-X.

22 Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. Parameterized complexity of elimination distance to first-order logic properties. *ACM Transactions on Computational Logic*, 23(3):17:1–17:35, 2022. doi:10.1145/3517129.

23 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar $\mathcal{F}$-Deletion: Approximation, Kernelization and Optimal FPT Algorithms. In *Proc. of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2012. doi:10.1109/FOCS.2012.62.

24 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.

25 Fedor V. Fomin and Torstein J. F. Strømme. Vertex cover structural parameterization revisited. In *Proc. of the 42nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 9941 of *LNCS*, pages 171–182, 2016. doi:10.1007/978-3-662-53536-3_15.

26 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *Journal of Computer and System Sciences*, 77(1):91–106, 2011. doi:10.1016/j.jcss.2010.06.007.

27 Jakub Gajarský, Petr Hlinený, Jan Obdrzálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. Kernelization using structural parameters on sparse graph classes. In *Proc. of the 21st Annual European Symposium on Algorithms (ESA)*, volume 8125 of *LNCS*, pages 529–540, 2013. doi:10.1007/978-3-642-40450-4_45.

28 Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Proc. of the 1st International Workshop on Parameterized and Exact Computation (IWPEC)*, volume 3162 of *LNCS*, pages 162–173, 2004. doi:10.1007/978-3-540-28639-4_15.

29 Eva-Maria C. Hols, Stefan Disc.tsch, and Astrid Pieterse. Elimination distances, blocking sets, and kernels for vertex cover. *SIAM Journal on Discrete Mathematics*, 36(3):1955–1990, 2022. doi:10.1137/20m1335285.

30 Bart M. P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited - upper and lower bounds for a refined parameter. *Theory of Computing Systems*, 53(2):263–299, 2013. doi:10.1007/s00224-012-9393-4.

31 Bart M. P. Jansen, Jari J. H. de Kroon, and Michal Włodarczyk. Vertex deletion parameterized by elimination distance and even less. In *Proc. of the 53rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 1757–1769, 2021. doi:10.1145/3406325.3451068.

32 Bart M. P. Jansen and Stefan Kratsch. On Polynomial Kernels for Structural Parameterizations of Odd Cycle Transversal. In *Proc. of the 6th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 7112 of *LNCS*, pages 132–144, 2011. doi:10.1007/978-3-642-28050-4_11.

33 Bart M. P. Jansen and Astrid Pieterse. Polynomial kernels for hitting forbidden minors under structural parameterizations. *Theoretical Computer Science*, 841:124–166, 2020. doi:10.1016/j.tcs.2020.07.009.

34 Bart M. P. Jansen and Michal Wlodarczyk. Lossy planarization: a constant-factor approximate kernelization for planar vertex deletion. In *Proc of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 900–913, 2022. doi:10.1145/3519935.3520021.

35 Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *LNCS*. Springer, 1994. doi:10.1007/BFb0045375.

36 Stefan Kratsch. A randomized polynomial kernelization for vertex cover with a smaller parameter. *SIAM Journal on Discrete Mathematics*, 32(3):1806–1839, 2018. doi:10.1137/16M1104585.

37 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *Journal of the ACM*, 67(3):16:1–16:50, 2020. doi:10.1145/3390887.

38 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

39 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Kernelization - Preprocessing with a guarantee. In *The Multivariate Algorithmic Revolution and Beyond*, pages 129–161, 2012. doi:10.1007/978-3-642-30891-8_10.

40 Diptapriyo Majumdar, Venkatesh Raman, and Saket Saurabh. Polynomial kernels for vertex cover parameterized by small degree modulators. *Theory of Computing Systems*, 62(8):1910–1951, 2018. doi:10.1007/s00224-018-9858-1.

41 Laure Morelle, Ignasi Sau, Giannos Stamoulis, and Dimitrios M. Thilikos. Faster parameterized algorithms for modification problems to minor-closed classes. In *Proc. of the 50th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 261 of *LIPIcs*, pages 93:1–93:19, 2023. doi:10.4230/LIPIcs.ICALP.2023.93.

42 Michal Pilipczuk, Nicole Schirrmacher, Sebastian Siebertz, Szymon Torunczyk, and Alexandre Vigny. Algorithms and data structures for first-order logic with connectivity under vertex failures. In *Proc. of the 49th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 229 of *LIPIcs*, pages 102:1–102:18, 2022. doi:10.4230/LIPICS.ICALP.2022.102.

43 Ignasi Sau and Uéverton dos Santos Souza. Hitting forbidden induced subgraphs on bounded treewidth graphs. *Information and Computation*, 281:104812, 2021. doi:10.1016/j.ic.2021.104812.

Fundamental Problems on Bounded-Treewidth Graphs: The Real Source of Hardness

Barış Can Esmer
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
Saarbrücken Graduate School of Computer Science, Saarland Informatics Campus, Germany

Jacob Focke
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Dániel Marx
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Paweł Rzążewski
Warsaw University of Technology, Poland
University of Warsaw, Poland

Abstract

It is known for many algorithmic problems that if a tree decomposition of width t is given in the input, then the problem can be solved with exponential dependence on t. A line of research initiated by Lokshtanov, Marx, and Saurabh [SODA 2011] produced lower bounds showing that in many cases known algorithms already achieve the best possible exponential dependence on t, assuming the Strong Exponential-Time Hypothesis (SETH). The main message of this paper is showing that the same lower bounds can already be obtained in a much more restricted setting: informally, a graph consisting of a block of t vertices connected to components of constant size already has the same hardness as a general tree decomposition of width t.

Formally, a (σ, δ)-*hub* is a set Q of vertices such that every component of Q has size at most σ and is adjacent to at most δ vertices of Q. We explore if the known tight lower bounds parameterized by the width of the given tree decomposition remain valid if we parameterize by the size of the given hub.

- For every $\varepsilon > 0$, there are $\sigma, \delta > 0$ such that Independent Set (equivalently Vertex Cover) cannot be solved in time $(2-\varepsilon)^p \cdot n$, even if a (σ, δ)-hub of size p is given in the input, assuming the SETH. This matches the earlier tight lower bounds parameterized by width of the tree decomposition. Similar tight bounds are obtained for Odd Cycle Transversal, Max Cut, q-Coloring, and edge/vertex deletions versions of q-Coloring.
- For every $\varepsilon > 0$, there are $\sigma, \delta > 0$ such that $\triangle$-Partition cannot be solved in time $(2-\varepsilon)^p \cdot n$, even if a (σ, δ)-hub of size p is given in the input, assuming the Set Cover Conjecture (SCC). In fact, we prove that this statement is *equivalent* to the SCC, thus it is unlikely that this could be proved assuming the SETH.
- For Dominating Set, we can prove a non-tight lower bound ruling out $(2-\varepsilon)^p \cdot n^{\mathcal{O}(1)}$ algorithms, assuming *either* the SETH or the SCC, but this does not match the $3^p \cdot n^{\mathcal{O}(1)}$ upper bound.

Thus our results reveal that, for many problems, the research on lower bounds on the dependence on tree width was never really about tree decompositions, but the real source of hardness comes from a much *simpler* structure.

Additionally, we study if the same lower bounds can be obtained if σ and δ are fixed universal constants (not depending on ε). We show that lower bounds of this form are possible for Max Cut and the edge-deletion version of q-Coloring, under the Max 3-Sat Hypothesis (M3SH). However, no such lower bounds are possible for Independent Set, Odd Cycle Transversal, and the vertex-deletion version of q-Coloring: better than brute force algorithms are possible for every fixed (σ, δ).

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 34; pp. 34:1–34:17

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Keywords and phrases Parameterized Complexity, Tight Bounds, Hub, Treewidth, Strong Exponential Time Hypothesis, Vertex Coloring, Vertex Deletion, Edge Deletion, Triangle Packing, Triangle Partition, Set Cover Hypothesis, Dominating Set

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.34

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2402.07331

1 Introduction

Starting with the work of Lokshtanov, Marx, and Saurabh [24], there is a line of research devoted to giving lower bounds on how the running time of parameterized algorithms can depend on treewidth (or more precisely, on the width of a given tree decomposition) [32, 31, 11, 7, 4, 19, 27, 8, 13, 28, 12]. The goal of this paper is to revisit the fundamental results from [24] to point out that previous work could have considered a *simpler* parameter to obtain *stronger* lower bounds in a more uniform way. Thus, in a sense, this line of research was never really about treewidth; a fact that future work should take into account.

Suppose we want to solve some algorithmic problem on a graph G given with a tree decomposition of width t. For many NP-hard problems, standard dynamic program techniques or meta theorems such as Courcelle's Theorem [5] show that the problem can be solved in time $f(t) \cdot n^{\mathcal{O}(1)}$ for some computable function f [10, Chapter 7]. In many cases, the running time is actually $c^t \cdot n^{\mathcal{O}(1)}$ for some constant $c > 1$, where it is an obvious goal to make the constant as small as possible. A line of work started by Lokshtanov, Marx, and Saurabh [24] provides tight conditional lower bounds for many problems with known $c^t \cdot n^{\mathcal{O}(1)}$-time algorithms. The lower bounds are based on the Strong Exponential-Time Hypothesis, formulated by Impagliazzo, Paturi, and Zane [16, 17].

▶ **Strong Exponential-Time Hypothesis (SETH).** *There is no $\varepsilon > 0$ such that for every k, every n-variable instance of k-SAT can be solved in time $(2-\varepsilon)^n \cdot n^{\mathcal{O}(1)}$.*

The goal of these results is to provide evidence that the base c of the exponent in the best known $c^t \cdot n^{\mathcal{O}(1)}$-time algorithm is optimal: if a $(c-\varepsilon)^t \cdot n^{\mathcal{O}(1)}$-time algorithm exists for any $\varepsilon > 0$, then SETH fails. The following theorem summarizes the basic results obtained by Lokshtanov, Marx, and Saurabh [24].

▶ **Theorem 1.1** ([24]). *If there exists an $\varepsilon > 0$ such that*
1. Independent Set *can be solved in time $(2-\varepsilon)^t \cdot n^{\mathcal{O}(1)}$, or*
2. Dominating Set *can be solved in time $(3-\varepsilon)^t \cdot n^{\mathcal{O}(1)}$, or*
3. Max Cut *can be solved in time $(2-\varepsilon)^t \cdot n^{\mathcal{O}(1)}$, or*
4. Odd Cycle Transversal *can be solved in time $(3-\varepsilon)^t \cdot n^{\mathcal{O}(1)}$, or*
5. q-Coloring *can be solved in time $(q-\varepsilon)^t \cdot n^{\mathcal{O}(1)}$ for some $q \geqslant 3$, or*
6. Triangle Partition *can be solved in time $(2-\varepsilon)^t \cdot n^{\mathcal{O}(1)}$,*

on input an n-vertex graph G together with a tree decomposition of width at most t, then the SETH fails.

Already in [24] it is pointed out that many of the lower bounds remain true even in the more restricted setting where the input is not a tree decomposition, but a path decomposition. This raises the following natural questions:

- How much further can we restrict the input and still obtain the same lower bounds?
- What is the *real* structural source of hardness in these results?

In this paper, we show that many of these lower bounds remain true in a much more restricted setting where a block of p vertices is connected to constant-size components. Additionally, we demonstrate that our results are very close to being best possible, as further restrictions of the structure of the graphs allow better algorithms.

We say that a set Q of vertices is a (σ, δ)-*hub* of G if every component of $G-Q$ has at most σ vertices and each such component is adjacent to at most δ vertices of Q in G[1]. Our goal is to prove lower bounds parameterized by the size of a (σ, δ)-hub given in the input, where σ and δ are treated as constants. One can observe that a (σ, δ)-hub of size p in G can be easily turned into a tree decomposition of width less than $p+\sigma$, hence the treewidth of G is at most $p+\sigma$. Therefore, any lower bound parameterized by the size p of hub immediately implies a lower bound parameterized by the width of the given tree decomposition. We systematically go through the list of problems investigated by Lokshtanov, Marx, and Saurabh [24], to see if the same lower bound can be obtained with this parameterization. Our results show that, in most cases, the results remain valid under parameterization by hub size. However, new insights, techniques and arguments are needed; in particular, we require different complexity assumptions for some of the statements.

1.1 Coloring Problems and Relatives

Let us first consider the q-Coloring problem: given a graph G, the task is to find a coloring of the vertices of G with q colors such that adjacent vertices receive different colors. Given a (σ, δ)-hub Q of size p, we can try all possible q-colorings on Q and check if they can be extented to every component of $G - Q$. Assuming σ and δ are constants, this leads to a $q^p \cdot n^{\mathcal{O}(1)}$ algorithm. Our first result shows that this is essentially best possible, assuming the SETH; note that this result immediately implies Theorem 1.1(5).

▶ **Theorem 1.2.** *Let $q \geqslant 3$ be an integer.*

1. *For every $\sigma, \delta \geqslant 1$, q-Coloring on n-vertex graphs can be solved in time $q^p \cdot n^{\mathcal{O}(1)}$ if a (σ, δ)-hub of size p is given in the input.*
2. *For every $\varepsilon > 0$, there exist integers $\sigma, \delta \geqslant 1$ such that if there is an algorithm solving in time $(q-\varepsilon)^p \cdot n^{\mathcal{O}(1)}$ every n-vertex instance of q-Coloring given with a (σ, δ)-hub of size at most p, then the SETH fails.*

The q-ColoringED problem is an edge-deletion optimization version of q-Coloring: given a graph G, the task is to find a set X of edges of minimum size such that $G \setminus X$ has a q-coloring. We show that $q^p \cdot n^{\mathcal{O}(1)}$ running time is essentially optimal for this problem as well.

▶ **Theorem 1.3.** *Let $q \geqslant 2$ be an integer.*

1. *For every $\sigma, \delta \geqslant 1$, q-ColoringED on n-vertex graphs can be solved in time $q^p \cdot n^{\mathcal{O}(1)}$ if a (σ, δ)-hub of size p is given in the input.*
2. *For every $\varepsilon > 0$, there exist integers $\sigma, \delta \geqslant 1$ such that if there is an algorithm solving in time $(q-\varepsilon)^p \cdot n^{\mathcal{O}(1)}$ every n-vertex instance of q-ColoringED given with a (σ, δ)-hub of size at most p, then the SETH fails.*

[1] This notion is related to *component order connectivity*, which is the size of the smallest set Q of vertices such that deleting Q leaves components of size not larger than some predefined constant σ [37, 33, 26, 3, 20, 22, 1, 14, 34, 6, 25]. Our definition has the additional constraint on the neighborhood size of each component. As we often refer to the set Q itself (not only its smallest possible size) and we want to make the constants σ, δ explicit, the terminology (σ, δ)-hub is grammatically more convenient than trying to express the same using component order connectivity.

For $q \geqslant 3$, the lower bound of Theorem 1.2 for q-Coloring immediately implies the same lower bound for the more general problem q-ColoringED. Observe that for $q = 2$, the q-ColoringED problem is equivalent to the Max Cut problem: deleting the minimum number of edges to make the graph bipartite is equivalent to finding a bipartition with the maximum number of edges going between the two classes. Thus the lower bound for Max Cut is needed to complete the proof of Theorem 1.3.

Let us consider now the vertex-deletion version q-ColoringVD, where given a graph G, the task is to find a set X of vertices of minimum size such that $G - X$ has a q-coloring (equivalently, we want to find a partial q-coloring on the maximum number of vertices). For this problem, a brute force approach would need to consider $(q+1)^p$ possibilities on a (σ, δ)-hub of size p: each vertex can receive either one of the q colors, or be deleted.

▶ **Theorem 1.4.** *Let $q \geqslant 1$ be an integer.*

1. *For every $\sigma, \delta \geqslant 1$, q-ColoringVD on n-vertex graphs can be solved in time $(q+1)^p \cdot n^{\mathcal{O}(1)}$ if a (σ, δ)-hub of size p is given in the input.*
2. *For every $\varepsilon > 0$, there exist integers $\sigma, \delta \geqslant 1$ such that if there is an algorithm solving in time $(q+1-\varepsilon)^p \cdot n^{\mathcal{O}(1)}$ every n-vertex instance of q-ColoringVD given with a (σ, δ)-hub of size at most p, then the SETH fails.*

Observe that Vertex Cover is equivalent to 1-ColoringVD and Odd Cycle Transversal is equivalent to 2-ColoringVD. Furthermore, Independent Set and Vertex Cover have the same time complexity (due to the well-known fact that minimum size of a vertex cover plus the maximum size of an independent set is always equal to the number of vertices). Thus the definition of q-ColoringVD gives a convenient unified formulation that includes these fundamental problems.

1.2 Packing Problems

Given a graph G, the Triangle Partition (denoted by $\triangle$-Partition for short) problem asks for a partition of the vertex set into triangles. Triangle Packing (denoted by $\triangle$-Packing) is the more general problem where the task is to find a maximum-size collection of vertex-disjoint triangles. Given a tree decomposition of width t, Theorem 1.1(6) shows that $2^t \cdot n^{\mathcal{O}(1)}$ is essentially the best possible running time. It seems that the same lower bound holds when parameterizing by the size of a hub, but the source of hardness is somehow different. Instead of assuming the SETH, we prove this lower bound under the *Set Cover Conjecture (SCC)* [9, 10]. In the d-Set Cover problem, we are given a universe U of size n and a collection $\mathcal{F}$ of subsets of U, each with size at most d. The task is to find a minimum-size collection of sets whose union covers the universe.

▶ **Set Cover Conjecture (SCC).** *For all $\varepsilon > 0$, there exists $d \geqslant 1$ such that there is no algorithm that solves every $\leqslant d$-Set Cover instance $(U, \mathcal{F})$ in time $(2-\varepsilon)^n \cdot n^{\mathcal{O}(1)}$ where $n = |U|$.*

We actually show that the lower bounds for $\triangle$-Partition/$\triangle$-Packing are *equivalent* to the SCC.

▶ **Theorem 1.5.** *The following three statements are equivalent:*

- *The SCC is true.*
- *For every $\varepsilon > 0$, there are $\sigma, \delta > 0$ such that $\triangle$-Partition on an n-vertex graph cannot be solved in time $(2-\varepsilon)^p \cdot n^{\mathcal{O}(1)}$, even if the input contains a (σ, δ)-hub of size p.*
- *For every $\varepsilon > 0$, there are $\sigma, \delta > 0$ such that $\triangle$-Packing on an n-vertex graph cannot be solved in time $(2-\varepsilon)^p \cdot n^{\mathcal{O}(1)}$, even if the input contains a (σ, δ)-hub of size p.*

Ideally, one would like to prove lower bounds under the more established conjecture: the SETH. However, Theorem 1.5 shows that it is no shortcoming of our technique that we prove the lower bound based on the SCC instead. If we proved statement 2 or 3 under the SETH, then this would prove that the SETH implies the SCC, resolving a longstanding open question.

1.3 Dominating Set

Given an n-vertex graph with a tree decomposition of width t, a minimum dominating set can be computed in time $3^t \cdot n^{\mathcal{O}(1)}$ using an algorithm based on fast subset convolution [35, 36]. By Theorem 1.1 (2), this running time cannot be improved to $(3-\varepsilon)^t \cdot n^{\mathcal{O}(1)}$ for any $\varepsilon > 0$, assuming the SETH. Can we get a $(3-\varepsilon)^p \cdot n^{\mathcal{O}(1)}$ algorithm if a hub of size p is given in the input? We currently have no answer to this question. In fact, we do not even have a good guess whether or not such algorithms should be possible. What we do have are two very simple weaker results that rule out $(2-\varepsilon)^p \cdot n^{\mathcal{O}(1)}$ algorithms, with one proof based on the SETH and the other proof based on the SCC.

▶ **Theorem 1.6.** *For every $\varepsilon > 0$, there are $\sigma, \delta > 0$ such that* Dominating Set *on an n-vertex graph with a (σ,δ)-hub of size p given in the input cannot be solved in time $(2-\varepsilon)^p \cdot n^{\mathcal{O}(1)}$, unless* both *the SETH and the SCC fail.*

Theorem 1.6 suggests that, if there is no $(3-\varepsilon)^p \cdot n^{\mathcal{O}(1)}$ time algorithm for Dominating Set, then perhaps the matching lower bound needs a complexity assumption that is stronger than both the SETH and the SCC.

1.4 Universal Constants for σ and δ?

The lower bounds in Theorems 1.2–1.6 are stated in a somewhat technical form: "for every $\varepsilon > 0$, there are σ and δ such that…". The statements would be simpler and more intuitive if they were formulated in a setting where σ and δ are universal constants, say, 100. Can we prove statements that show, for example, that there is no $(2-\varepsilon)^p \cdot n^{\mathcal{O}(1)}$ algorithm, where p is the size of a $(100, 100)$-hub given in the input? The answer to this question is complicated. For the vertex-deletion problem q-ColoringVD (which includes Vertex Cover and Odd Cycle Transversal) there are actually better than brute force algorithms for fixed constant values of σ and δ.

▶ **Theorem 1.7.** *For every $q \geqslant 3$ and $\sigma, \delta > 0$, there exists $\varepsilon > 0$ with the following property: every instance (G, L) of q-ColoringVD with n vertices, given with a (σ,δ)-hub of size p, can be solved in time $(q+1-\varepsilon)^p \cdot n^{\mathcal{O}(1)}$.*

Thus Theorem 1.7 explains why the formulation of Theorem 1.4 needs to quantify over σ and δ, and cannot be stated for a fixed pair (σ, δ).

On the other hand, for the edge-deletion problem q-ColoringED (which includes Max Cut), we can prove stronger lower bounds where σ and δ are universal constants. However, we need a complexity assumption different from the SETH.

An instance of Max 3-Sat is a CNF formula φ with at most three literals in each clause. We ask for the minimum number of clauses that need to be deleted in order to obtain a satisfiable formula. Equivalently, we look for a valuation of the variables which violates the minimum number of clauses. Clearly, an instance of Max 3-Sat with n variables can be solved in time $2^n \cdot n^{\mathcal{O}(1)}$ by exhaustive search. It is a notorious problem whether this running time can be significantly improved, i.e., whether there exists an $\varepsilon > 0$ such that every n-variable instance of Max 3-Sat can be solved in time $(2-\varepsilon)^n$.

▶ **Max 3-Sat Hypothesis (M3SH).** *There is no $\varepsilon > 0$ such that every n-variable instance of* Max 3-Sat *can be solved in time $(2-\varepsilon)^n \cdot n^{\mathcal{O}(1)}$.*

Under this assumption, we can prove a lower bound where $\delta = 6$ and σ is a constant (depending only on q).

▶ **Theorem 1.8.** *For every $q \geqslant 2$ there is an integer σ such that the following holds. For every $\varepsilon > 0$, no algorithm solves every n-vertex instance of q-ColoringED that is given with a $(\sigma, 6)$-hub of size p, in time $(q-\varepsilon)^p \cdot n^{\mathcal{O}(1)}$, unless the M3SH fails.*

For the case $q = 2$ we even show a slight improvement over Theorem 1.8 – in this case it suffices to consider instances with a constant σ and $\delta = 4$.

For $\triangle$-Partition, we do not know if the lower bound of Theorem 1.5, ruling out $(2-\varepsilon)^p \cdot n^{\mathcal{O}(1)}$ running time under the SCC, remains valid for some fixed universal σ and δ independent of ε. Note that the proof of Theorem 1.5 provides a reduction from $\triangle$-Partition to d-Set Packing for some d. It is known that d-Set Packing over a universe of size n can be solved in time $(2-\varepsilon)^n \cdot (n+m)^{\mathcal{O}(1)}$ with some $\varepsilon > 0$ depending on d [30, 21, 2]. However, our reduction from $\triangle$-Partition to d-Set Packing chooses d in a way that it cannot be used to reduce the case of a fixed σ and δ to a d-Set Packing problem with fixed d. It seems that we would need to understand if certain generalizations of d-Set Packing can also be solved in time $(2-\varepsilon)^n \cdot (n+m)^{\mathcal{O}(1)}$ for fixed d. The simplest such problem would be the generalization of d-Set Packing where the sets in the input are partitioned into pairs and the solution is allowed to use at most one set from each pair.

1.5 Discussion

Given the amount of attention to algorithms on tree decompositions and the number of nontrivial techniques that were developed to achieve the best known algorithms, it is a natural question to ask if these algorithms are optimal. Even though understanding treewidth is a very natural motivation for this line of research, the actual results turned out to be less related to treewidth than one would assume initially: the lower bounds remain valid even under more restricted conditions. Already the first paper on this topic [24] states the lower bounds in a stronger form, as parameterized by pathwidth or by feedback vertex set number (both of which are bounded below by treewidth). Some other results considered parameters such as the size of a set Q where every component of $G - Q$ is a path [18] or has bounded treewidth [15]. However, our results show that none of these lower bounds got to the fundamental reason why known algorithms on bounded-treewidth graphs cannot be improved: Theorems 1.2–1.5 highlight that these algorithms are best possible already if we consider a much more restricted problem setting where constant-sized gadgets are attached to a set of hub vertices. Moreover, Theorems 1.2 and 1.4 are likely to be best possible: as we have seen, for coloring and its vertex-deletion generalizations, σ and δ cannot be made a constant independent from ε (Theorem 1.7). Therefore, one additional conceptual message of our results is understanding where the hardness of solving problems on bounded-treewidth graphs really stems from, by reaching the arguably most restricted setting in which the lower bounds hold.

The success of Theorems 1.2–1.4 (for coloring problems and relatives) suggests that possibly all the treewidth optimality results could be revisited and the same methodology could be used to strengthen to parameterization by hub size. But the story is more complicated than that. For example, for $\triangle$-Packing, the ground truth appears to be that the lower bound parameterized by width of the tree decomposition can be strengthened to a lower

bound parameterized by hub size. However, proving the lower bound parameterized by hub size requires a different proof technique, and we can do it only by assuming the SCC – and for all we know this is an assumption orthogonal to the SETH. In fact, we showed that the lower bound for $\triangle$-Packing is equivalent to the SCC, making it unlikely that a simple proof based on the SETH exists. For Dominating Set, we currently do not know how to obtain tight bounds, highlighting that it is far from granted that all results parameterized by width of tree decomposition can be easily turned into lower bounds parameterized by hub size.

Another important aspect of our results is the delicate way they have to be formulated, with the values of σ and δ depending on ε. Theorem 1.8 shows that in some cases it is possible to prove a stronger bound where σ and δ are universal constants, but this comes at a cost of choosing a different complexity assumption (M3SH). Thus, there is a tradeoff between the choice of the complexity assumption and the strength of the lower bound. In general, it seems that the choice of complexity assumption can play a crucial role in these kind of lower bounds parameterized by hub size. This has to be contrasted with the case of parameterization by the width of the tree decomposition, where the known lower bounds are obtained from the SETH (or its counting version).

It would be natural to try to obtain lower bounds parameterized by hub size for other algorithmic problems as well. The lower bounds obtained in this paper for various fundamental problems can serve as a starting point for such further results. Concerning the problems studied in this paper, we leave two main open questions:

- For Dominating Set, can we improve the lower bound of Theorem 1.6 to rule out $(3-\varepsilon)^p \cdot n^{\mathcal{O}(1)}$ algorithms, under some reasonable assumption? Or is there perhaps an algorithm beating this bound?
- For $\triangle$-Partition/$\triangle$-Packing, can we improve the lower bound of Theorem 1.5 such that σ and δ are universal constants? Or is it true perhaps that for every fixed σ and δ, there is an algorithm solving these problems in time $(2-\varepsilon)^p \cdot n^{\mathcal{O}(1)}$ for some $\varepsilon > 0$?

2 Technical Overview

In this section, we overview some of the most important technical ideas in our results.

2.1 q-Coloring

The algorithmic statement in Theorem 1.2 is easily obtained via a simple branching procedure. For the hardness part, we use a lower bound of Lampis [23] for constraint satisfaction problems (CSP) as a starting point: for any $\varepsilon > 0$ and integer d, there is an integer r such that there is no algorithm solving CSP on n variables of domain size d and r-ary constraints in time $(d-\varepsilon)^n$. Therefore, to prove Theorem 1.2 (2), we give a reduction that, given an n-variable CSP instance where the variables are over $[q]$ and the arity of constraints is some constant r, creates an instance of q-Coloring having a hub of size roughly n.

First, we introduce a set of n main vertices in the hub, representing the variables of the CSP instance. We would like to represent each r-ary constraint with a gadget that is attached to a set S of r vertices. We will first allow our gadgets to use lists that specify to which colors certain vertices are allowed to be mapped. In a second step we then remove these lists.

A bit more formally, we say that an *r-ary q-gadget* is a graph J together with a list assignment $L\colon V(J) \to 2^{[q]}$ and r distinguished vertices $\mathbf{x} = (z_1, \ldots, z_r)$ from J. The vertices $z_1, \ldots, z_r$ are called *portals*. A *list coloring* of (J, L) is an assignment $\varphi\colon V(J) \to [q]$ that respects the lists L, i.e., with $\varphi(v) \in L(v)$ for all $v \in V(J)$.

A construction by Jaffke and Jansen [18] gives a gadget that enforces that a set of vertices forbids one prescribed coloring. We use this statement to construct a gadget extends precisely the set of colorings that are allowed according to some relation.

▶ **Proposition 2.1.** *Let $q \geqslant 3$ and $r \geqslant 1$ be integers, and let $R \subseteq [q]^r$ be a relation. Then there exists an r-ary q-gadget $\mathcal{F} = (F, L, (z_1, \ldots, z_r))$ such that*

- *the list of every vertex is contained in $[q]$,*
- *for each $i \in r$, it holds that $L(z_i) = [q]$,*
- *$\{z_1, \ldots, z_r\}$ is an independent set,*
- *for any $\psi : \{z_1, \ldots, z_r\} \to [q]$, coloring vertices $z_1, \ldots, z_r$ according to ψ can be extended to a list coloring of (F, L) if and only if $(\psi(z_1), \ldots, \psi(z_r)) \in R$.*

Then, by introducing one gadget per constraint and attaching it to the vertices of the hub, from the q^n possible behaviors of the hub vertices, only those can be extended to the gadgets that correspond to a satisfying assignment of the CSP instance. Note that gadgets are allowed to use lists and they model the relational constraints using list colorings. So the final step to obtain a reduction to q-Coloring is to remove these lists. This can be done using a standard construction, where a central clique of size q is used to model the q colors, and a vertex v of the graph is adjacent to the ith vertex of the clique, whenever $i \notin L(v)$.

2.2 Vertex Deletion to q-Coloring

Similarly to q-Coloring, the algorithmic statement in Theorem 1.4 is easily obtained via a simple branching procedure. However, for q-ColoringVD, we need to consider $q + 1$ possibilities at each vertex: assigning to it one of the q colors, or deleting it. This leads to the running time $(q + 1)^p \cdot n^{\mathcal{O}(1)}$. The hardness proof is also similar, but this time we have to give a reduction that, given an n-variable CSP instance where the variables are over $[q+1]$ and the arity of constraints is some constant r, creates an instance of q-ColoringVD having a hub of size roughly n. Intuitively, we are using deletion as the $(q + 1)$-st color: the $(q+1)^n$ possibilities for these vertices in the q-ColoringVD problem (coloring with q colors + deletion) correspond to the $(q + 1)^n$ possible assignments of the CSP instance. To enforce this interpretation, we attach to these vertices small gadgets representing each constraint. We attach a large number of copies of each such gadget, which means that it makes no sense for an optimum solution to delete vertices from these gadgets and hence deletions occur only in the hub. This means that we can treat the vertices of the gadgets as "undeletable".

We would like to use again the construction from Proposition 2.1 to create gadgets that enforce that a set of vertices has one of the prescribed colorings/deletions. A gadget can force the deletion of a vertex if its neighbors are colored using all q colors. However, there is a fundamental limitation of this technique: deleting a vertex is always better than coloring it. That is, a gadget cannot really force a set S of vertices to the color "red": from the viewpoint of the gadget, deleting some of them and coloring the rest red is equally good. In other words, it is not true that every relation $R \subseteq [q + 1]^r$ can be represented by a gadget that allows only these combinations of q colors + deletion on a set S of r vertices.

To get around this limitation, we use a grouping technique to have control over how many vertices are deleted. Let us divide the n variables into $M = n/b$ blocks $B_1, \ldots, B_M$ of size b each. Let us guess the number f_i of variables in B_i that receive the value $q + 1$ in a hypothetical solution; that is, we expect f_i deletions in block B_i of central vertices. Instead of just attaching a gadget to a set S of at most r vertices, now each gadget is attached to the at most r blocks containing S. Besides ensuring a combination of values on S that satisfies the constraint, the gadget also ensures that each block B_i it is attached to has at

least the guessed number f_i of deletions. This way, if we have a solution with exactly $\sum_{i=1}^{M} f_i$ deletions, then we know that it has exactly f_i deletions in the i-th block. Therefore, if a gadget forces the deletion of f_i vertices of B_i and forces a coloring on the remaining vertices of B_i, then we know that that block has exactly this behavior in the solution.

2.3 Edge Deletion to q-Coloring

Let us turn our attention to the edge-deletion version now. Similarly to the vertex-deletion version, the algorithmic results are simple, thus we discuss only the hardness proofs here. As starting point for all our reductions, we use a CSP problem with domain size q that naturally generalizes Max 3-Sat: the task is to find an assignment of variables that satisfies the maximum number of constraints. For $q = 2$, the hardness of this problem follows from the SETH and the M3SH. For $q \geqslant 3$, we prove a new tight lower bound based on M3SH.

For $q \geqslant 3$, the lower bound of Theorem 1.3 (hardness of q-ColoringED under the SETH) already follows from our result for *finding* a q-coloring *without deletions* (Theorem 1.2). So, in order to complete the proof of Theorem 1.3, we give a reduction from the CSP problem with $q = 2$ to 2-ColoringED (i.e., Max Cut), which shows hardness under SETH. As the gadgets of Proposition 2.1 work only for $q \geqslant 3$, we need to design new gadgets using only 2 colors for this case.

The same reduction can be used to establish the lower bound from Theorem 1.8 (hardness of q-ColoringED under the M3SH) in the $q = 2$ case. For the $q \geqslant 3$ case, we present a reduction from the CSP problem with domain size q to q-ColoringED. Here we can once again use the gadgets from Proposition 2.1.

In all cases, as the gadgets we design may use lists, we establish respective lower bounds for the list coloring problem on the way. In a second step, we then show how to remove the lists.

Max CSP – Hardness under the M3SH

For some positive integers d and r, we define Max (d,r)-CSP: Given v variables over a q-element domain and a set of n relational constraints of arity 3, the task is to find an assignment of the variables such that the maximum number of constraints are satisfied. The problem can be solved in time $q^v \cdot n^{\mathcal{O}(1)}$ by brute force. For $q = 2$, the problem is clearly a generalization of Max 3-Sat, hence the M3SH immediately implies that there is no $(q-\varepsilon)^v \cdot n^{\mathcal{O}(1)}$ algorithm for any $\varepsilon > 0$. We show that the M3SH actually implies this for any $q \geqslant 2$. This might also be a helpful tool for future work.

▶ **Theorem 2.2.** *For $d \geqslant 2$ and any $r \geqslant 3$, there is no algorithm solving every n-variable instance of Max (d,r)-CSP in time $(d-\varepsilon)^n \cdot n^{\mathcal{O}(1)}$ for $\varepsilon > 0$, unless the M3SH fails.*

In order to show Theorem 2.2, if q is a power of 2, then a simple grouping argument works: for example, if $q = 2^4 = 16$, then each variable of the CSP instance can represent 4 variables of the Max 3-Sat instance, and hence it is clear that a $(q-\varepsilon)^v$ algorithm would imply a $(2-\varepsilon)^{4v}$ algorithm for a Max 3-Sat instance with $4v$ variables.

The argument is not that simple if q is not a power of 2, say $q = 15$. Then a variable of that CSP instance cannot represent all 16 possibilities of 4 variables of the Max 3-Sat instance, and using it to represent only 3 variables would be wasteful. We cannot use the usual trick of grouping the CSP variables such that each group together represents a group of Max 3-Sat variables: then each constraint representing a clause would need to involve not only 3 variables, but 3 blocks of variables, making δ larger than 3. Instead, for each block of 4 variables of the Max 3-Sat instance, we randomly choose 15 out of the 16 possible

assignments, and use a single variable of the CSP instance to represent these possibilities. An optimum solution of a $4v$-variable Max 3-Sat instance "survives" this random selection with probability $(15/16)^v$. Thus a $(15-\varepsilon)^v \cdot n^{\mathcal{O}(1)}$ time algorithm for the CSP problem would give a randomized $(16/15)^v \cdot (15-\varepsilon)^v \cdot n^{\mathcal{O}(1)} = (16-\varepsilon')^v \cdot n^{\mathcal{O}(1)} = (2-\varepsilon'')^{4v} \cdot n^{\mathcal{O}(1)}$ time algorithm for Max 3-Sat, violating (a randomized version of) the M3SH. Furthermore, we show in the full version that the argument can be derandomized using the logarithmic integrality gap between integer and fractional covers in hypergraphs.

Realizing Relations using Lists

Recall that an r-ary q-gadget is a graph with lists in $[q]$ and r specified portal vertices. For our hardness proofs, we reduce from Max (q,r)-CSP, and we use gadgets to "model" the relations in $[q]^r$. We say that an r-ary q-gadget *realizes* a relation $R \in [q]^r$ if there is an integer k such that (1) for each $\mathbf{d} \in R$, if the portals are colored according to $\mathbf{d}$, then it requires precisely k edge deletions to extend this to a full list coloring of the gadget, and (2) extending a state that is *not* in R requires strictly more than k edge deletions. We say that such a gadget 1-realizes R, if for each state outside of R it takes precisely $k+1$ edge deletions to extend this state. So, this is a stronger notion in the sense that now the violation cost is the same for all tuples outside of R. Moreover, with a 1-realizer in hand, by identifying copies of this gadget with the same portal vertices one can freely adjust the precise violation cost – this works as long as the portals form an independent set and therefore no multiedges are introduced in the copying process.

For our treatment of the case $q \geqslant 3$, we again use Proposition 2.1 to show that arbitrary relations over a domain of size q can be realized. As Proposition 2.1 is for the decision problem without deletions, it does not help for the case $q = 2$, i.e., for Max Cut/2-Coloring. In this case, we need a different approach to show that every relation over a domain of size 2 can be realized. For 2-Coloring, a single edge is essentially a "Not Equals"-gadget as the endpoints have to take different colors or otherwise the edge needs to be deleted. Starting from this, we show how to model OR-relations of any arity. With these building blocks we then obtain the following result.

▶ **Theorem 2.3.** *For each $r \geqslant 1$, and $R \subseteq [2]^r$, there is an r-ary 2-gadget that 1-realizes R.*

Removing the Lists

Note that gadgets may use lists and therefore, on the way, we first obtain the following lower bounds for the respective list coloring problems.

▶ **Theorem 2.4.** *For every $q \geqslant 2$ and $\varepsilon > 0$, there are integers σ and δ such that if an algorithm solves in time $(q-\varepsilon)^p \cdot n^{\mathcal{O}(1)}$ every n-vertex instance of List-q-ColoringED that is given with a (σ,δ)-hub of size p, then the SETH fails.*

▶ **Theorem 2.5.** *For every $q \geqslant 2$, there is a constant σ_q such that, for every $\varepsilon > 0$, if an algorithm solves in time $(q-\varepsilon)^p \cdot n^{\mathcal{O}(1)}$ every n-vertex instance of List-q-ColoringED that is given with a $(\sigma_q,3)$-hub of size p, then the M3SH fails.*

In a second step, we show how to remove the lists by adding some additional object of size roughly q (a central vertex or a q-clique for $q = 2$ or $q \geqslant 3$, respectively). This addition is then considered to be part of the hub, thereby increasing the size of the hub by some constant. However, this modification means that for the other gadgets the number of neighbors in the hub increases slightly. This is irrelevant for the SETH-based lower bound, but it leads to a slight increase in the universal constant δ that we obtain for our M3SH-based lower bounds for the coloring problems without lists.

2.4 Covering, Packing, and Partitioning

Theorem 1.5 gives lower bounds for $\triangle$-Partition and $\triangle$-Packing based on the Set Cover Conjecture. This hypothesis was formulated in terms of the d-Set Cover problem. For our purposes, it is convenient to consider slightly different covering/partitioning problems. To facilitate our reductions and as a tool for future reductions of this type, we establish equivalences between eight different covering type problems. Before we make this more formal in Theorem 2.6, let us briefly introduce the corresponding problems.

First, we use $=d$-Set Cover and $\leqslant d$-Set Cover to distinguish between the problem for which the sets have size exactly d or at most d, respectively. For $\triangle$-Partition, it is more natural to start a reduction from the partitioning problems $=d$-Set Partition or $\leqslant d$-Set Partition, in which the task is to find pairwise disjoint sets that cover the universe. The $\leqslant d$-Set Partition problem can be considered as a decision problem. However, we can also consider the corresponding optimization problem in which the task is to minimize the number of selected sets, and we use $\leqslant d$-Set Partition (#Sets) to denote this problem. Further variants are the optimization problems $=d$-Set Packing (#Sets) and $\leqslant d$-Set Packing (#Sets), in which we need to select the maximum number of pairwise disjoint sets. For $\leqslant d$-Set Packing, an equally natural goal is to maximize the total size of the selected sets (for $= d$-Set Packing, this is of course equivalent to maximizing the number of selected sets). So we use $\leqslant d$-Set Packing (Union) to denote the packing problem in which the union/total size of the selected sets is maximized.

Given the large number of variants of d-Set Cover, one may wonder how they are related to each other. In particular, does the SCC imply lower bounds for these variants? There are obvious reductions between some of these problems (e.g., from $= d$-Set Cover to $\leqslant d$-Set Cover) and there are also reductions that are not so straightforward. We fully clarify this question by showing that choosing *any* of these problems in the definition of the SCC leads to an equivalent statement. Thus in our proofs to follow we can choose whichever form is most convenient for us. Knowing this equivalence could prove useful for future work as well.

▶ **Theorem 2.6.** *Suppose that for one of the problems below, it is true that for every $\varepsilon > 0$, there is an integer d such that the problem cannot be solved in time $(2-\varepsilon)^n \cdot n^{\mathcal{O}(1)}$, where n is the size of the universe. Then this holds for all the other problems as well. In particular, any of these statements is equivalent to the SCC.*

1. *$=d$-Set Cover*
2. *$=d$-Set Partition*
3. *$=d$-Set Packing (#Sets)*
4. *$\leqslant d$-Set Cover*
5. *$\leqslant d$-Set Partition*
6. *$\leqslant d$-Set Partition (#Sets)*
7. *$\leqslant d$-Set Packing (#Sets)*
8. *$\leqslant d$-Set Packing (Union)*

To make the statements about relationships between the problems from the list in Theorem 2.6 more concise, it will be convenient to introduce some shorthand notation. Let $\mathbb{A} = \{A_d\}_{d\geqslant 1}$ and $\mathbb{B} = \{B_d\}_{d\geqslant 1}$ be two families of problems where A_d and B_d belong to the list in Theorem 2.6. To shorten notation, we speak of an *n-element instance* if the universe U of an instance has size n. We say that $\mathbb{A}$ is 2^n-hard if the following lower bound holds

> For each $\varepsilon > 0$ there is some $d \geqslant 1$ such that no algorithm solves A_d on all n-element instances in time $(2-\varepsilon)^n \cdot n^{\mathcal{O}(1)}$.

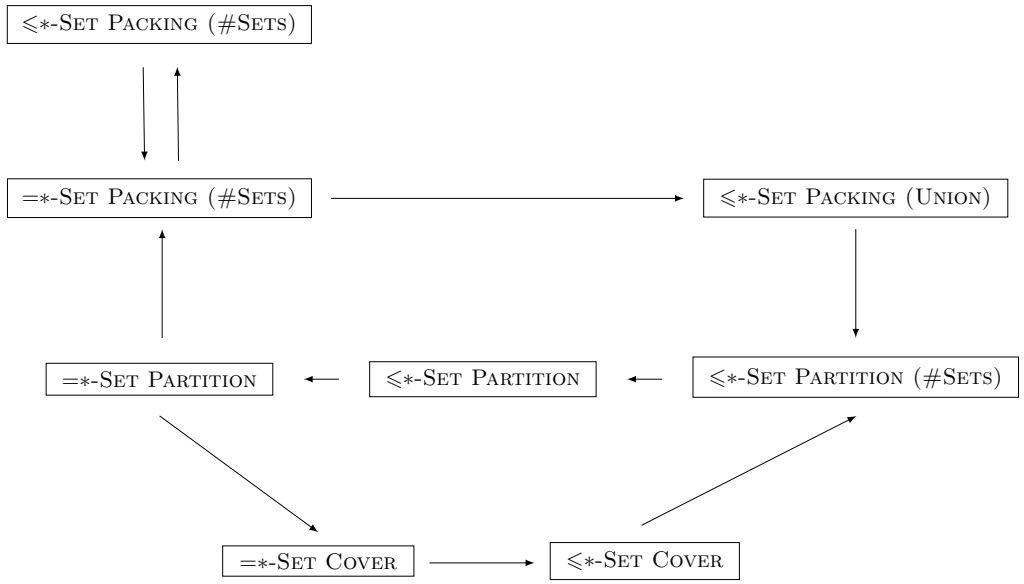


Figure 1 An overview of the proof of Theorem 2.6. An arrow from $\mathcal{A}$ to $\mathcal{B}$ indicates an implication stating that if $\mathcal{A}$ is 2^n-hard then $\mathcal{B}$ is 2^n-hard as well. The details are in the full version.

Using this language, the SCC states that $\{\leqslant d\text{-Set Cover}\}_{d\geqslant 1}$ is 2^n-hard. To establish Theorem 2.6, we show reductions, stating that if $\mathcal{A}$ is 2^n-hard then $\mathcal{B}$ is 2^n-hard as well. Spelled out this means:

> Suppose for each $\varepsilon > 0$ there is some $d \geqslant 1$ such that no algorithm solves A_d on all n-element instances in time $(2-\varepsilon)^n \cdot n^{\mathcal{O}(1)}$.
>
> Then, for each $\varepsilon > 0$ there is some $d' \geqslant 1$ such that no algorithm solves $B_{d'}$ on all n-element instances in time $(2-\varepsilon)^n \cdot n^{\mathcal{O}(1)}$.

This shows that this is really a relationship between two classes of problems, and not necessarily a relationship between A_d and B_d for the same value d. To make this distinction explicit, we write $=*$-Set Cover if we refer to the class of problems $\{=d\text{-Set Cover}\}_{d\geqslant 1}$. We use analogous notation for the other problems on the list. For example, a simple observation is that if $=*$-Set Cover is 2^n-hard then so is $\leqslant*$-Set Cover as the latter is a generalization of the former. The reductions we use to prove Theorem 2.6 are illustrated in Figure 1.

2.5 Triangle Partition and Triangle Packing

Now let us discuss the proof of Theorem 1.5 that can be found in the full version. The proof consists of two main steps: (1) a reduction from $=*$-Set Partition to $\triangle$-Partition, and (2) a reduction from $\triangle$-Packing to $\leqslant*$-Set Packing (#Sets) (see Figure 2). Recall that by Theorem 2.6, assuming the SCC, all of $=*$-Set Partition, $\leqslant*$-Set Packing (#Sets), and $\leqslant*$-Set Cover are 2^n-hard. Finally, $\triangle$-Partition trivially reduces to $\triangle$-Packing, so indeed, the statements in Theorem 1.5 are equivalent.

Reducing Set Partition to $\triangle$-Partition

We start with step (1), i.e., reducing an instance $(U, \mathcal{F})$ of $=d$-Set Partition to an equivalent instance G of $\triangle$-Partition. With a simple technical trick we can ensure that d is divisible by 3.

Figure 2 An overview of the reductions in the proof of Theorem 1.5. The two dashed arrows refer to 2^n-hardness reductions from Theorem 2.6. To establish these two connections, note that we actually utilize almost all the reductions shown in Figure 1. The arrows annotated with (1) and (2) refer to another two reductions proved in the full version.

The main building block used in the reduction is the so-called $\triangle$-eq gadget. For fixed d, it is a graph with d designated vertices called *portals*. The gadget essentially has exactly two triangle packings that cover all non-portal vertices:

- one that also covers *all portals* (i.e., is actually a triangle partition), and
- one that covers *no portal*.

Now the construction of G is simple: we introduce the set Q containing one vertex for each element of U, and for each set $S \in \mathcal{F}$ we introduce a copy of the $\triangle$-eq gadget whose portals represent elements of S and are identified with corresponding vertices from Q. It is straightforward to verify that there is $\mathcal{F}' \subseteq \mathcal{F}$ that partitions U if and only if G has a triangle partition: the sets from $\mathcal{F}'$ correspond to $\triangle$-eq gadgets whose non-portal vertices are covered in the first way. Note that Q is a (σ, d)-hub of G, where σ is the number of vertices of the $\triangle$-eq gadget, i.e., is a constant that depends only on d.

Reducing $\triangle$-Packing to Set Packing.

Now let us consider a graph G given with a (σ, δ)-hub Q of size p, and an integer t. We will show that a hypothetical fast algorithm for $\leqslant d$-Set Packing (#Sets) can be used to determine whether G has a triangle packing of size at least t.

For simplicity of exposition, assume that G has no triangles contained in Q; dealing with such triangles is not difficult but would complicate the notation. We say that a component C of $G - Q$ is *active* in some triangle packing Π if there is a triangle in Π that intersects both C and Q. Note that for any triangle packing there are at most p active components.

We would like to guess components that are active for some (unknown) solution Π. However, this results in too many branches. We deal with it by employing color-coding and reducing the problem to its auxiliary *precolored variant*. Suppose for a moment that we are given a coloring ψ of components of $G - Q$ into p/c colors, where c is a large constant, with a promise that at most c components in each color are active in Π.

For a color $i \in [p/c]$, let $\mathcal{C}_i$ denote the set of components of $G - Q$ colored i by ψ. The *contribution* of the color i to Π is the number of triangles that intersect vertices in components from $\mathcal{C}_i$. Note that the size of Π is the sum of contributions of all color (since we assumed that there are no triangles contained in Q). What can be said about the contribution of i? Certainly picking a maximum triangle packing in the graph consisting only of components from $\mathcal{C}_i$ is a lower bound. Let X_i denote the number of triangles in such a triangle packing and note that X_i can be computed in polynomial time as each component of $G - Q$ is of constant size. Moreover, for each active component $C \in \mathcal{C}_i$, there are at most σ triangles that

intersect both C and Q (as each of them has to use a distinct vertex from C). As, by the promise on ψ, there are at most c active components in $\mathcal{C}_i$, we observe that the contribution of i is at most $X_i + c\sigma$. We exhaustively guess the contribution of each color by guessing the offset q_i against X_i; it gives a constant number of options per color. We reject guesses where the total contribution of all colors, i.e., the number of all triangles packed, is less than t.

For each color i, we enumerate all sets $S \subseteq Q$ that are candidates for these vertices of Q that form triangles with vertices from components of $\mathcal{C}_i$; call such sets *i-valid*. An i-valid set S must satisfy the following two conditions. First, the size of S is at most $2c\sigma$, as there are at most $c\sigma$ vertices in active components from $\mathcal{C}_i$ and each such vertex belongs to a triangle with at most two vertices from Q. Second, there exists a triangle packing Π_S in the graph induced by S together with components of $\mathcal{C}_i$ such that

- at most c elements from $\mathcal{C}_i$ are active in Π_S (this follows from the promise on ψ), and
- the number of triangles in Π_S is at least $X_i + q_i$ (by our guess of q_i).

It is not difficult to verify that i-valid sets can be enumerated in polynomial time, where the degree of the polynomial depends on c and σ.

Now we are ready to construct an instance $(U, \mathcal{F}, p/c)$ of $\leqslant d$-Set Packing (#Sets). The universe U is $Q \cup \{a_i \mid i \in [p/c]\}$, i.e., it consists of the hub of G and one extra vertex per color. For each i-valid set S, we include in $\mathcal{F}$ the set $S \cup \{a_i\}$. Again, one can verify that $\mathcal{F}$ contains p/c pairwise disjoint sets if and only if G has a packing of t triangles that agree both with ψ and with the guessed values of q_i's.

By adjusting c, we can ensure that the whole algorithm works in time $(2-\varepsilon')^p \cdot |V(G)|^{\mathcal{O}(1)}$, for some $\varepsilon' > 0$, provided that we have a fast algorithm for $\leqslant d$-Set Packing (#Sets).

The only thing left is to argue how we obtain the coloring ψ satisfying the promise. Here we use *splitters* introduced by Naor, Schulman, and Srinivasan [29]. Informally, a splitter is a family of colorings of a "large set" $\mathcal{X}$, such that for each "small subset" $\mathcal{Y} \subseteq \mathcal{X}$ there is a coloring that splits $\mathcal{Y}$ evenly. In our setting, the "large set" $\mathcal{X}$ is the set of all components of $G - Q$ and the "small subset" $\mathcal{Y}$ is the set of all active components with respect to some fixed (but unknown) solution; recall that there are at most p such active components. Since our colorings use p/c colors, we are sure that there is some ψ for which at most $\frac{p}{p/c} = c$ components in each color are active. Calling the result of Naor, Schulman, and Srinivasan [29], we can find a small splitter Ψ, and then just exhaustively try every coloring $\psi \in \Psi$. Again, carefully adjusting the constants, we can ensure that the overall running time is $(2-\varepsilon)^p \cdot |V(G)|^{\mathcal{O}(1)}$, for some $\varepsilon > 0$.

References

1. Jørgen Bang-Jensen, Eduard Eiben, Gregory Z. Gutin, Magnus Wahlström, and Anders Yeo. Component order connectivity in directed graphs. *Algorithmica*, 84(9):2767–2784, 2022. doi:10.1007/s00453-022-01004-z.
2. Andreas Björklund. Exact Covers via Determinants. In Jean-Yves Marion and Thomas Schwentick, editors, *27th International Symposium on Theoretical Aspects of Computer Science*, volume 5 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 95–106, Dagstuhl, Germany, 2010. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2010.2447.
3. F. Boesch, D. Gross, and C. Suffel. Component order connectivity. In *Proceedings of the Twenty-ninth Southeastern International Conference on Combinatorics, Graph Theory and Computing (Boca Raton, FL, 1998)*, volume 131, pages 145–155, 1998.
4. Glencora Borradaile and Hung Le. Optimal dynamic program for r-domination problems over tree decompositions. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.8.

5 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

6 Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification. *Comput. Sci. Rev.*, 48:100556, 2023. doi:10.1016/j.cosrev.2023.100556.

7 Radu Curticapean, Nathan Lindzey, and Jesper Nederlof. A tight lower bound for counting Hamiltonian cycles via matrix rank. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1080–1099. SIAM, 2018. doi:10.1137/1.9781611975031.70.

8 Radu Curticapean and Dániel Marx. Tight conditional lower bounds for counting perfect matchings on graphs of bounded treewidth, cliquewidth, and genus. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1650–1669. SIAM, 2016. doi:10.1137/1.9781611974331.ch113.

9 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On Problems as Hard as CNF-SAT. *ACM Transactions on Algorithms*, 12(3):41:1–41:24, May 2016. doi:10.1145/2925416.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

11 László Egri, Dániel Marx, and Paweł Rzążewski. Finding list homomorphisms from bounded-treewidth graphs to reflexive graphs: a complete complexity characterization. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 27:1–27:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.27.

12 Jacob Focke, Dániel Marx, Fionn Mc Inerney, Daniel Neuen, Govind S. Sankar, Philipp Schepper, and Philip Wellnitz. Tight complexity bounds for counting generalized dominating sets in bounded-treewidth graphs. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3664–3683. SIAM, 2023. doi:10.1137/1.9781611977554.ch140.

13 Jacob Focke, Dániel Marx, and Paweł Rzążewski. Counting list homomorphisms from graphs of bounded treewidth: tight complexity bounds. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 431–458. SIAM, 2022. doi:10.1137/1.9781611977073.22.

14 Daniel Gross, L. William Kazmierczak, John T. Saccoman, Charles L. Suffel, and Antonius Suhartomo. On component order edge connectivity of a complete bipartite graph. *Ars Comb.*, 112:433–448, 2013.

15 Falko Hegerfeld and Stefan Kratsch. Towards exact structural thresholds for parameterized complexity. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPIcs*, pages 17:1–17:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.IPEC.2022.17.

16 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

17 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

18 Lars Jaffke and Bart M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. In Dimitris Fotakis, Aris Pagourtzis, and Vangelis Th. Paschos, editors, *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, volume 10236 of *Lecture Notes in Computer Science*, pages 345–356, 2017. doi:10.1007/978-3-319-57586-5_29.

19 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discret. Appl. Math.*, 264:90–117, 2019. doi:10.1016/j.dam.2018.11.002.

20 Lawrence William Kazmierczak. *On the relationship between connectivity and component order connectivity.* ProQuest LLC, Ann Arbor, MI, 2003. Thesis (Ph.D.)–Stevens Institute of Technology. URL: http://gateway.proquest.com/openurl?url_ver=Z39.88-2004&rft_val_fmt=info:ofi/fmt:kev:mtx:dissertation&res_dat=xri:pqdiss&rft_dat=xri:pqdiss:3088817.

21 Mikko Koivisto. Partitioning into sets of bounded cardinality. In Jianer Chen and Fedor V. Fomin, editors, *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*, volume 5917 of *Lecture Notes in Computer Science*, pages 258–263. Springer, 2009. doi:10.1007/978-3-642-11269-0_21.

22 Mithilesh Kumar and Daniel Lokshtanov. A 2lk kernel for l-component order connectivity. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 20:1–20:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.20.

23 Michael Lampis. Finer tight bounds for coloring on clique-width. *SIAM J. Discret. Math.*, 34(3):1538–1558, 2020. doi:10.1137/19M1280326.

24 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

25 Daniel Lokshtanov, Pranabendu Misra, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Fpt-approximation for FPT problems. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 199–218. SIAM, 2021. doi:10.1137/1.9781611976465.14.

26 Dániel Marx, Pranabendu Misra, Daniel Neuen, and Prafullkumar Tale. A framework for parameterized subexponential algorithms for generalized cycle hitting problems on planar graphs. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2085–2127. [Society for Industrial and Applied Mathematics (SIAM)], Philadelphia, PA, 2022. doi:10.1137/1.9781611977073.83.

27 Dániel Marx, Govind S. Sankar, and Philipp Schepper. Degrees and gaps: Tight complexity results of general factor problems parameterized by treewidth and cutwidth. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 95:1–95:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.95.

28 Dániel Marx, Govind S. Sankar, and Philipp Schepper. Anti-factor is FPT parameterized by treewidth and list size (but counting is hard). In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPIcs*, pages 22:1–22:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.IPEC.2022.22.

29 Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 182–191. IEEE Computer Society, 1995. doi:10.1109/SFCS.1995.492475.

30 Jesper Nederlof. Finding Large Set Covers Faster via the Representation Method. In Piotr Sankowski and Christos Zaroliagis, editors, *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 69:1–69:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2016.69.

31 Karolina Okrasa, Marta Piecyk, and Paweł Rzążewski. Full complexity classification of the list homomorphism problem for bounded-treewidth graphs. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 74:1–74:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.74.

32 Karolina Okrasa and Paweł Rzążewski. Fine-grained complexity of the graph homomorphism problem for bounded-treewidth graphs. *SIAM J. Comput.*, 50(2):487–508, 2021. doi:10.1137/20M1320146.

33 Masataka Shirahashi and Naoyuki Kamiyama. Kernelization algorithms for a generalization of the component order connectivity problem. *J. Oper. Res. Soc. Japan*, 66(2):112–129, 2023.

34 Dekel Tsur. Faster parameterized algorithms for two vertex deletion problems. *Theoretical Computer Science*, 940:112–123, 2023. doi:10.1016/j.tcs.2022.10.044.

35 Johan M. M. van Rooij. Fast algorithms for join operations on tree decompositions. In Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen, editors, *Treewidth, Kernels, and Algorithms - Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *Lecture Notes in Computer Science*, pages 262–297. Springer, 2020. doi:10.1007/978-3-030-42071-0_18.

36 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0_51.

37 M. Yatauro. Component order connectivity and vertex degrees. *Congr. Numer.*, 220:195–205, 2014.

A Spectral Approach to Approximately Counting Independent Sets in Dense Bipartite Graphs

Charlie Carlson
Department of Computer Science, University of California Santa Barbara, CA, USA

Ewan Davies[1]
Department of Computer Science, Colorado State University, Fort Collins, CO, USA

Alexandra Kolla
Computer Science and Engineering, University of California Santa Cruz, CA, USA

Aditya Potukuchi
Department of Electrical Engineering and Computer Science, York University, Toronto, Canada

Abstract

We give a randomized algorithm that approximates the number of independent sets in a dense, regular bipartite graph – in the language of approximate counting, we give an FPRAS for #BIS on the class of dense, regular bipartite graphs. Efficient counting algorithms typically apply to "high-temperature" problems on bounded-degree graphs, and our contribution is a notable exception as it applies to dense graphs in a low-temperature setting. Our methods give a counting-focused complement to the long line of work in combinatorial optimization showing that CSPs such as Max-Cut and Unique Games are easy on dense graphs via spectral arguments.

Our contributions include a novel extension of the method of graph containers that differs considerably from other recent low-temperature algorithms. The additional key insights come from spectral graph theory and have previously been successful in approximation algorithms. As a result, we can overcome some limitations that seem inherent to the aforementioned class of algorithms. In particular, we exploit the fact that dense, regular graphs exhibit a kind of small-set expansion (i.e., bounded threshold rank), which, via subspace enumeration, lets us enumerate small cuts efficiently.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Mathematics of computing → Approximation algorithms; Theory of computation → Algorithm design techniques

Keywords and phrases approximate counting, independent sets, bipartite graphs, graph containers

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.35

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2307.09533

Funding *Ewan Davies*: Supported in part by NSF grant CCF-2309707.
Aditya Potukuchi: Supported in part by NSERC Discovery grants RGPIN-2023-05087 & DGECR-2023-00408.

1 Introduction

Exactly computing the number $i(G)$ of independent sets in a graph G is #P-hard, even when restricted to bipartite graphs [41]. In the general case, approximating $i(G)$ (to within, say, a constant factor) is NP-hard, even when restricted to d-regular graphs with $d \geq 6$ [20, 46, 45]. Restricted to *bipartite* graphs the problem of counting independent sets is known as #BIS, and the prospect of hardness of approximation is less clear because finding a maximum

[1] Corresponding author

independent set can be done in polynomial time. Under polynomial-time approximation-preserving reductions, many natural counting problems are equivalent to #BIS [17], and the complexity of approximating #BIS has received a lot of attention. Existing approximation algorithms for #BIS include "high-temperature" algorithms that work when degrees on one side of the bipartition are small [38], "low-temperature" algorithms that require additional assumptions such as expansion [11, 29] or unbalanced degrees [8], and exponential-time algorithms that are nonetheless faster than algorithms for the general, non-bipartite case [24]. The description of these methods in terms of temperature is due to a common generalization in terms of weighted counting and strong connections to statistical physics, where counting (weighted) independent sets corresponds to computing the partition function of the hard-core model.

The idea that Max-CSP optimization problems such as Max-Cut and Unique Games should be easy to approximate on dense graphs – perhaps because they have good expansion properties – is well-established [3, 18, 19]. Many of the techniques that apply to dense or expanding graphs have been generalized in interesting directions. In particular, spectral methods give good results in both dense graphs and expanders, and in many cases can be extended to more refined structural properties such as small-set expansion and threshold rank to great effect. Most of the prominent approaches to Max-CSPs relevant to this work fall into three categories: algorithmic regularity lemmas which began with Frieze and Kannan [19] and were extended to threshold rank by Oveis Gharan and Trevisan [39]; convex hierarchies and correlation rounding [4, 6, 25]; and the spectral technique of subspace enumeration due to Kolla and Tulsiani [36, 37]. Prior to these developments were several algorithms demonstrating that counting problems on dense graphs admit efficient approximation algorithms [1, 16, 33], though these results do not apply to counting independent sets.

An analogous theme in approximate counting is to obtain algorithms on expander graphs or random graphs [7, 10, 21, 26, 29]. Despite superficial similarity to the aforementioned work on Max-CSPs in the sense that these works give algorithms for dense or expanding instances, there is relatively little work establishing any common underlying phenomenon that makes Max-CSP problems and counting problems easy on dense or expanding graphs. A notable exception is due to Risteski [42], who connected the work on correlation rounding and convex hierarchies [6] to the broad and well-studied problem of approximating partition functions. His approach is also known as the variational method. Regularity methods and correlation rounding do provide some evidence of structure common to these problems; for example, Coja–Oghlan and various coauthors have developed a range of regularity lemmas and applied them to both Max-CSPs and spin models on random graphs [5, 13, 14], and Coja–Oghlan and Perkins independently discovered correlation rounding in the context of Gibbs measures and partition functions [15]. Counting independent sets is not typically one of the examples studied, though occasionally this is more for convenience than for fundamental reasons.

In the specific context of #BIS, connections to Max-CSP research are even more scarce. The polymer approach of Jenssen, Keevash and Perkins [29] is a major algorithmic breakthrough for #BIS which shows that several prominent #BIS-hard problems can be approximated in polynomial time on bounded-degree expander graphs (and thus random d-regular graphs for $d = O(1)$). Further refinements of the method broaden the range of problems covered [21, 26], provide faster algorithms based on rapid mixing of Markov chains known as polymer dynamics [11], or weaken the structural properties required by applying container theorems to combinatorial enumeration problems that arise in the method [10, 32]. None of these developments give polynomial-time algorithms in dense graphs, however. Carlson, Davies, and Kolla [9] applied the polymer method to approximate the Potts model partition

function on (bounded-degree) graphs with bounded threshold rank, but the conditions their analysis requires are prohibitively restrictive, and it is unclear whether their techniques can be applied to #BIS. While Risteski's approach has been extended and improved [28, 35], results are stated for spin models with soft constraints such as the Ising and Potts models, and the approximation guarantees degrade in the presence of the hard constraints that are inherent to independent sets.

1.1 Main result

We specifically address the superficial similarities between algorithms for Max-CSPs and counting independent sets by giving an algorithm for approximately counting independent sets in dense, regular bipartite graphs which combines the highly successful techniques of polymer models, subspace enumeration, and container theorems for the enumeration of independent sets in bipartite graphs. Our approximation guarantee is of the strong type typically sought in approximate counting. We say that a relative ϵ-approximation of a real number x is a real number y such that $e^{-\epsilon} \leq x/y \leq e^{\epsilon}$, and a fully polynomial randomized approximation scheme (FPRAS) for a counting problem is an algorithm that with probability at least $3/4$ outputs a relative ϵ-approximation to the solution in time polynomial in the instance size and $1/\epsilon$.

▶ **Theorem 1.** *For each $\delta \in (0,1)$ there is an FPRAS for #BIS on the class of d-regular bipartite graphs G with $d = \lfloor \delta |V(G)|/2 \rfloor$.*

We use spectral methods and subspace enumeration to enumerate small cuts in d-regular bipartite graphs via an ϵ-net of the vector space spanned by small eigenvalues of the Laplacian matrix of the graph, influenced by the use of these methods in combinatorial optimization [2, 36, 37] and approximate counting. Some of our analysis builds upon the perturbative approach of [27, 29] and an important refinement of this method due to Jenssen and Perkins [30] (and with Potukuchi [31]) that uses graph container lemmas of the type developed by Sapozhenko [43, 44]. While container theorems for independent sets have been used to control enumeration problems that arise in establishing the convergence of the cluster expansion [30, 31, 32], and these have inspired container-like theorems for controlling analogous enumeration problems [10], our addition of subspace enumeration here has a different purpose.

In terms of running time, our result improves upon the dense case of an algorithm of Jenssen, Perkins, and Potukuchi [32] which runs in subexponential time on d-regular bipartite graphs for all $d \geq \omega(1)$. In the case $d = \Theta(n)$ their algorithm takes time $\exp(\Omega(\log^4 n))$, and our contribution works for any accuracy parameter ϵ, which is not given by the methods in [32]. The improvement stems from incorporating the spectral techniques mentioned above, which lets us sidestep algorithmic cluster expansion. That is, our spectral techniques overcome an obstacle in the algorithm of [32] related to polynomial accuracy: we can achieve arbitrary accuracy without resorting to a naive enumeration of polymers (which in this setting are connected subgraphs of the square of the instance).

An interesting question posed in [32] is whether #BIS admits a general subexponential-time algorithm. One of our technical contributions is to show that a perspective on graph spectra involving higher-order eigenvalues and eigenvectors advances our understanding of #BIS.

2 Overview

Fix $\delta > 0$ and for $d = \lfloor \delta n \rfloor$, a bipartite graph $G = (X \cup Y, E)$ on $2n$ vertices. Let $\epsilon > 0$ and note that we allow ϵ to depend on n.

Our proof begins with the well-known observation that to enumerate independent sets in a bipartite graph it suffices to enumerate deviations from the "ideal" independent set X. That is, we have the identity

$$i(G) = \sum_{A \subseteq X} 2^{|Y \setminus N(A)|} \tag{1}$$

because, for a fixed $A \subseteq X$, any vertex of $Y \setminus N(A)$ can be added to A without spanning an edge. There is a similar formula for $i(G)$ based on enumerating deviations from Y. An important achievement of [29] is to give a rigorous proof that in bipartite graphs with strong expansion, typical independent sets are small deviations from either X or Y. Intuitively, we see a hint of this idea in equation (1) as when G is an expander we expect that $N(A) \gg |A|$ and so the terms on the right-hand side are small unless $|A|$ is small. Given this, one might hope to obtain an algorithm provided one can solve the problem of efficiently enumerate the small deviations and quantifing their contributions to $i(G)$. This is done in [29] by approximating $i(G)$ with the sum of two polymer models, and brute force enumeration of terms in the cluster expansion for these models.

If the bipartite graph is not an expander, then large deviations from X and Y must be handled. For example, in a $2n$-vertex disjoint union of complete d-regular bipartite graphs, a significant number of independent sets intersect both X and Y on $\Omega(n)$ vertices. To extend the algorithm to all bipartite graphs, using an idea from [32] we can separate contributions from expanding and non-expanding pieces of the deviation A. The first step is to break $A \subseteq X$ in the sum in (1) into pieces with disjoint neighborhoods. We say that a subset $A \subseteq X$ is *polymer*[2] if it is connected in the square G^2 of G, and note that any $A \subseteq X$ admits a unique partition into polymers which have disjoint neighborhoods. We call the polymers in this partition the *components* of A and denote the set of components of A by $\mathcal{K}(A)$. We say that two polymers are *compatible* if their neighborhoods are disjoint, and that a set or tuple of polymers is *compatible* if the polymers in it are pairwise compatible. Thus, subsets $A \subseteq X$ correspond to compatible sets of polymers via the unique partition into polymers with disjoint neighborhoods.

▷ Claim 2.

$$i(G) = \sum_{k \geq 0} \frac{1}{k!} \sum_{\substack{(A_1,\ldots,A_k) \text{ s.t.} \\ \text{each } A_i \text{ is a polymer and} \\ (A_1,\ldots,A_k) \text{ compatible}}} 2^{\left|Y \setminus \bigcup_{j=1}^{k} N(A_j)\right|}. \tag{2}$$

Proof. The claim follows from the correspondence between subsets $A \subseteq X$ and sets of compatible polymers given by $A \mapsto \mathcal{K}(A)$. By convention, we sum over compatible tuples of polymers which leads to the term $1/k!$ to account for the permutations of each tuple. We use the fact that compatible polymers have disjoint neighborhoods for the correspondence of the summands. ◁

The *closure* $[A]$ of a subset $A \subseteq X$ is $[A] := \{x \in X : N(x) \subseteq N(A)\}$, and we say that A is *closed* if $A = [A]$. Note that A is closed if and only if each component of A is closed. A subset $A \subseteq X$ is called t-*expanding* if $|N(A)| = |[A]| + t$, and (in a slight abuse of terminology

[2] In related works the term "2-linked" is used for the property of being connected in G^2.

that we hope the reader permits) t-contracting if $|N(A)| < |[A]| + t$. For a fixed t_0 that we determine later, we split the sum over polymers in (2) according to t_0-contraction. To do this, for each subset $A \subseteq X$, let $X_A = X \setminus N(N(A))$ and $Y_A = Y \setminus N(A)$. Let $\mathcal{P}_A$ be the set of polymers which are subsets of X_A and define

$$\Xi(A) := \sum_{k \geq 0} \frac{1}{k!} \sum_{\substack{(B_1,\ldots,B_k) \text{ s.t.} \\ B_i \in \mathcal{P}_A \text{ is not } t_0\text{-contracting} \\ \text{and } (B_1,\ldots,B_k) \text{ compatible}}} 2^{-\sum_{i=1}^{k} |N(B_i)|},$$

where the inner sum is over compatible k-tuples of polymers, each of which is not t_0-contracting (equivalently, t-expanding for some $t \geq t_0$).

▷ Claim 3.

$$i(G) = \sum_{k \geq 0} \frac{1}{k!} \sum_{\substack{(A_1,\ldots,A_k) \text{ s.t.} \\ \text{each } A_i \text{ is a } t_0\text{-contracting polymer} \\ \text{and } (A_1,\ldots,A_k) \text{ compatible}}} 2^{\left|Y \setminus \bigcup_{j=1}^{k} N(A_j)\right|} \cdot \Xi\left(\bigcup_{j=1}^{k} A_j\right). \tag{3}$$

Proof. From Claim 2 we can split the sum over tuples of polymers into a sum over tuples of t_0-contracting polymers and tuples of non-t_0-contracting polymers. The idea is to first sum over tuples $(A_1, \ldots, A_k)$ of t_0-contracting polymers and then use the fact that for $A = \bigcup_{j=1}^{k} A_j$ the quantity $\Xi(A)$ contains a sum over the ways to extend this tuple to one containing non-t_0-contracting polymers and the summand is the additional contribution that each such extension makes. The definition of $\mathcal{P}_A$ means that any $B_i \in \mathcal{P}_A$ is compatible with each component of A. With a little care, one can check that the permutaions of the tuples are correctly taken into account and the claim follows. ◁

A further refinement of the expression for $i(G)$ groups t_0-contracting polymers according to their neighborhoods. The motivation for this is that two subsets $A \subseteq X$ and $B \subseteq X$ (each corresponding to the union of some compatible tuple of t_0-contracting polymers) have the same contribution in the sum if $N(A) = N(B)$ because this implies that $\Xi(A) = \Xi(B)$. For a subset $A \subseteq X$ write

$$\mathcal{D}(A) := \prod_{A' \in \mathcal{K}(\mathcal{A})} \left|\{B' \subseteq A' : B' \text{ is a polymer and } N(B') = N(A')\}\right|.$$

The quantity $\mathcal{D}(A)$ counts the number of subsets B of A such that $N(B) = N(A)$ and which are formed by choosing for each component A' of A, a subset $B' \subseteq A'$ which is a polymer. Note that if A' is t_0-contracting then so is any polymer $B' \subseteq A'$ with $N(B') = N(A')$. For convenience, we define $\mathcal{A}$ to be the set of all $A \subseteq X$ with closed, t_0-contracting components.

▷ Claim 4.

$$i(G) = \sum_{A \in \mathcal{A}} \mathcal{D}(A) \cdot 2^{|Y \setminus N(A)|} \cdot \Xi(A), \tag{4}$$

Proof. From Claim 3 we can restrict the sum over tuples of t_0-contracting polymers to closed t_0-contracting polymers provided, for each compatible tuple $(A_1, \ldots, A_k)$ of closed t_0-contracting polymers, we multiply their contribution to the sum by a term counting the number of ways of getting that contribution with polymers that are not necessarily closed. Identifying compatible tuples of closed polymers with their union, i.e. setting $A = \bigcup_{j=1}^{k} A_j$, the contribution to the sum from A is $2^{|Y \setminus N(A)|} \cdot \Xi(A)$. The term $\mathcal{D}(A)$ is exactly the number

of ways of getting this contribution. The claim follows from the conversion of the sum back into one over suitable subsets of X, namely those in $\mathcal{A}$, instead of a sum over compatible tuples of polymers. ◁

Now that we have a suitable expression (4) for $i(G)$, we can describe how our algorithm approximates $i(G)$. Our algorithm simply enumerates the sets $A \in \mathcal{A}$, approximates each $\mathcal{D}(A)$ term, and uses the fact (which we must prove) that 1 is a good approximation of each $\Xi(A)$ to approximate $i(G)$. Given these subroutines, computing the sum (4) is straightforward. The analysis of our algorithm thus splits into three separate components. Recall that the input is a d-regular bipartite graph G on $2n$ vertices such that for some constant $\delta > 0$ we have $d = \lfloor \delta n \rfloor$, and an approximation error ϵ. We set $t_0 = C \log(n/\epsilon)$, where $C = C(\delta)$ is large enough, and the correctness and running time of our algorithm follows from the results below. Note that for this choice of t_0 an exponential such as 4^{t_0} is polynomial in n and $1/\epsilon$.

▶ **Lemma 5.** *For $t_0 \leq 2^{-8}d$, the set $\mathcal{A} = \{A \subseteq X : A$ closed and t_0-contracting$\}$ has size at most $n^{O(1/\delta)} \cdot 4^{t_0}$ and can be enumerated in the same time.*

The proof of this lemma uses subspace enumeration to find small cuts in G, and then for each such small cut enumerates the sets $A \in \mathcal{A}$ which are close to the cut. See Section 4.

▶ **Lemma 6.** *Let $A \subseteq X$ be a closed t_0-contracting polymer. Then for $\epsilon', \rho' > 0$ there is a randomized algorithm running in time polynomial in n, $1/\epsilon'$ and $\log(1/\rho')$ that with probability at least $1 - \rho'$ outputs a relative ϵ'-approximation to the number of polymers $B \subseteq A$ such that $N(B) = N(A)$.*

This lemma uses straightforward estimation of an expectation by repeated sampling, and is very similar to the analogous result in [32]. The proof is in Section 5. We use the lemma in each component of the sets $A \in \mathcal{A}$ in the claim below. This claim requires an upper bound on t_0, but this is a small technical detail as the only way to violate this bound is to choose an error parameter ϵ so small that one has time for brute force because an FPRAS can take time polynomial in $1/\epsilon$, see Section 3 where we use the claim.

▷ Claim 7. Suppose that $t_0 \leq d/2$. Then each set $A \in \mathcal{A}$ has at most $2/\delta = O(1)$ components, and for $\epsilon', \rho' > 0$ there is a randomized algorithm running in time polynomial in n, $1/\epsilon'$ and $\log(1/\rho')$ that, given a set $A \in \mathcal{A}$ as input, with probability $1 - \rho'$ obtains an ϵ'-approximation of $\mathcal{D}(A)$.

Proof. Any t_0-contracting set must have size at least $d - t_0$, and in the case $t_0 \leq d/2$ we have $d - t_0 \geq d/2$ and hence each $A \in \mathcal{A}$ has at least $2n/d = 2/\delta$ components.

Observe that if $A \in \mathcal{A}$ has ℓ components then running the algorithm of Lemma 6 on each component with error parameter ϵ'/ℓ and probability parameter ρ'/ℓ yields, with probability at least $1 - \rho'$, a relative ϵ-approximation to $\mathcal{D}(A)$ in time polynomial in n, ℓ/ϵ' and $\log(\ell/\rho')$. When $t_0 \leq d/2$ we have the upper bound $\ell = O(1)$ from above and the claim follows. ◁

▶ **Lemma 8.** *Let $A \in \mathcal{A}$, then $1 \leq \Xi(A) \leq e^{\epsilon/2}$.*

This result means that 1 is a relative $\epsilon/2$-approximation for each of the $\Xi(A)$ terms appearing in (4). The proof is based on graph container methods due to Sapozhenko [43, 44], which have since been refined, [23, 22, 34, 40], and their application to algorithmic counting [30, 31, 32]. We give the proof in Section 6.

3 The algorithm and proof of Theorem 1

Input A $\lfloor \delta n \rfloor$-regular bipartite graph $G = (X \cup Y, E)$ on $2n$ vertices and an approximation error $\epsilon > 0$.

Output A relative ϵ-approximation i' of $i(G)$.

Recall that $C = C(\delta)$ is a large enough constant, and that $t_0 = C \log(n/\epsilon)$. In the following proof, implicit constants in the $O(\cdot)$ notation and implicit polynomials are allowed to depend on δ but not ϵ. If $\epsilon \leq n \exp(-d/(2^8 C))$ then we can afford to run a brute force algorithm that computes $i(G)$ exactly in time $e^{O(n)}$ and the running time is still polynomial in $1/\epsilon$. Otherwise, we note that for all large enough n we have $d - 2^7 t_0 \geq d/2$ and run the following algorithm. For convenience, we assume that $\epsilon \leq 1$ and simply run the algorithm for $\epsilon = 1$ if the given ϵ is larger.

First, construct the set $\mathcal{A}$, which can be done in time $(n/\epsilon)^{O(1)}$ by Lemma 5. Note also that $|\mathcal{A}|$ is polynomial in n and $1/\epsilon$. Then, for each $A \in \mathcal{A}$ compute an $\epsilon/2$-approximation $\tilde{\mathcal{D}}_A$ of $\mathcal{D}_A$ with the algorithm of Claim 7 and probability parameter $\rho' = 3/(4|\mathcal{A}|)$. Then $\log(1/\rho')$ is polynomial in $\log n$ and $\log(1/\epsilon)$ so the running time of this step is polynomial in n and $1/\epsilon$. By a union bound, with probability at least $3/4$ we get the desired approximation in each application of the claim, and thus a valid relative $\epsilon/2$-approximation $\tilde{\mathcal{D}}_A$ of each $\mathcal{D}_A$. Then output $i' = \sum_{A \in \mathcal{A}} \tilde{\mathcal{D}}_A 2^{|Y \setminus N(A)|}$. By Lemma 8 and the analysis above, the output is a valid ϵ-approximation of $i(G)$ obtained in time $(n/\epsilon)^{O(1)}$, thus proving Theorem 1.

4 Subspace enumeration and contracting sets

The proof of Lemma 5 has two parts. First, we show how to enumerate small cuts using subspace enumeration. For related results see [2, 36, 37]. We use the term *cut* to mean a subset of $V = X \cup Y$, and the *value* $|\nabla(C)|$ of a cut C is the number of edges with precisely one endpoint in C. Subspace enumeration involves what is commonly called an ϵ-net of a subset U' of a vector space, which is a collection of points such that U' is contained in the union of the balls of radius ϵ around each point. Since we reserve ϵ for the error parameter in our algorithm, our nets are ξ-nets.

▶ **Lemma 9.** *Let $G = (V, E)$ be a d-regular bipartite graph on $N = 2n$ vertices. There is a set $\mathcal{C}^{\text{cut}} \subseteq 2^V$ such that $|\mathcal{C}^{\text{cut}}| \leq n^{O(n/d)}$ and $\mathcal{C}^{\text{cut}}$ has the following property. For all $t \geq 1$ and cuts $S \subseteq V$ with value $|\nabla(S)| \leq td$, there is some $C \in \mathcal{C}^{\text{cut}}$ such that $|S \triangle C| \leq 32t$ and $|\nabla(C)| \leq 33td$. Moreover, the set $\mathcal{C}^{\text{cut}}$ can be constructed in time $n^{O(n/d)}$.*

Proof. Let $d = \lambda_1 \geq \cdots \geq \lambda_N = -d$ be the spectrum of the adjacency matrix A of G. The facts that $\lambda_1 = d = -\lambda_N$ and that the spectrum of A is symmetric about zero are standard, see e.g. [12]. Let k be such that A has precisely $2k$ eigenvalues of absolute value at least $d/2$. Counting closed walks of length two gives

$$\operatorname{Tr}(A^2) = Nd = \sum_{i=1}^{N} \lambda_i^2 \geq kd^2/2,$$

and hence $k \leq 4n/d$.

Let $L = dI - A$ be the Laplacian matrix of G and let $\mathbf{e}_1, \ldots, \mathbf{e}_N$ be an orthonormal basis of eigenvectors of L such that $\mathbf{e}_i$ has eigenvalue μ_i with $0 = \mu_1 \leq \cdots \leq \mu_N = 2d$. By the definition of k, it must be the case that $\mu_{k+1} > d/2$. Let U be the span of $\mathbf{e}_1, \ldots \mathbf{e}_k$, and $U^\perp$ be the orthogonal complement of U. For $\xi = \sqrt{2}$, we require an efficient construction of a ξ-net $\mathcal{E} \subseteq U$ covering all vectors of L^2-norm at most $\sqrt{n}$ in U. For example, we can take

$$\mathcal{E} := \left\{ \mathbf{p} = \sum_{i=1}^{k} x_i \mathbf{e}_i : x_1, \ldots, x_k \in (\xi/\sqrt{k}) \cdot \mathbb{Z},\ \|\mathbf{p}\| \leq \sqrt{n} \right\},$$

yielding $|\mathcal{E}| \leq (2\sqrt{nk}/\xi)^k$. Then every vector in U with L^2-norm at most $\sqrt{n}$ lies at most distance ξ from a vector in $\mathcal{E}$.

The algorithm to construct $\mathcal{C}^{\text{cut}}$ is as follows. Start with $\mathcal{C}^{\text{cut}} = \emptyset$ and for each point $\mathbf{p} \in \mathcal{E}$, form $\mathbf{p}'$ by rounding each coordinate of $\mathbf{p}$ to $\{0, 1\}$ (breaking ties with $1/2 \mapsto 1$) and add the vertex subset with indicator vector $\mathbf{p}'$ to $\mathcal{C}^{\text{cut}}$.

We now show that $\mathcal{C}^{\text{cut}}$ has the desired properties. By the construction of $\mathcal{C}^{\text{cut}}$ and $\mathcal{E}$ we have $|\mathcal{C}^{\text{cut}}| \leq |\mathcal{E}| \leq n^{O(n/d)}$. To establish the other property of $\mathcal{C}^{\text{cut}}$, let $t \geq 1$ and consider an arbitrary subset $S \subseteq V$ with $|\nabla(S)| \leq td$. Let $\mathbf{s}$ be the indicator vector of the set S and write this vector in the eigenbasis of L as $\mathbf{s} = \sum_{i=1}^{N} s_i \mathbf{e}_i$. Let $\mathbf{u} = \sum_{i=1}^{k} s_i \mathbf{e}_i$ be the projection of $\mathbf{s}$ onto U and let $\mathbf{p}$ be the point in $\mathcal{E}$ closest to $\mathbf{u}$. Indicator vectors of subsets of V have L^2-norm at most $\sqrt{n}$, and hence $\|\mathbf{u} - \mathbf{p}\| \leq \xi$.

Without considering our need for an efficient construction, the idea is that because $\nabla(S)$ is small we know that $\mathbf{s}$ is an indicator vector close to its projection $\mathbf{u}$ onto U. Thus, if we form $\mathcal{C}^{\text{cut}}$ as the union of all sets whose indicator vectors are close to vectors in U, each set S of interest has an indicator vector that lies within a distance twice the definition of "close" to a set in C.

To make the above sketch efficient, we replace U with the ξ-net $\mathcal{E}$. Note that

$$td \geq |\nabla(S)| = \mathbf{s}^{\mathrm{T}} L \mathbf{s} = \sum_{i=1}^{N} \mu_i s_i^2 \geq \frac{d}{2} \sum_{i=k+1}^{N} s_i^2.$$

But $\sum_{i=k+1}^{N} s_i^2 = \|\mathbf{s} - \mathbf{u}\|^2$, so we have the bound $\|\mathbf{s} - \mathbf{u}\| \leq \sqrt{2t}$. Then we immediately have $\|\mathbf{s} - \mathbf{p}\| \leq \sqrt{2t} + \xi$ from the triangle inequality. Let $\mathbf{p}'$ be obtained from $\mathbf{p}$ by rounding each coordinate to $\{0, 1\}$, breaking ties with $1/2 \mapsto 1$, and let $C \subseteq V$ be the set whose indicator vector is $\mathbf{p}'$. We have $|S \triangle C| = \|\mathbf{s} - \mathbf{p}'\|^2$ and we bound the latter with the triangle inequality. In particular, $\mathbf{s}$ is an indicator vector of distance at most $\sqrt{2t} + \xi$ from $\mathbf{p}$ and $\mathbf{p}'$ must be the closest indicator vector to $\mathbf{p}$, hence $\|\mathbf{p} - \mathbf{p}'\| \leq \sqrt{2t} + \xi$. Then $\|\mathbf{s} - \mathbf{p}\| \leq 2(\sqrt{2t} + \xi)$, and because $t \geq 1$ and $\xi = \sqrt{2}$ we have

$$|S \triangle C| \leq 4 \left(\sqrt{t} + \xi \right)^2 \leq 32t.$$

It remains to bound the value of the cut $|\nabla(C)|$, and the desired bound follows from the observation that

$$|\nabla(C)| \leq |\nabla(S)| + d|S \triangle C| \leq td + 32td = 33td. \qquad \blacktriangleleft$$

Lemma 9 tells us that there is an efficient construction of a collection $\mathcal{C}^{\text{cut}}$ of cuts such that any small cut S must be close to a cut in $\mathcal{C}^{\text{cut}}$ in Hamming distance. We now show that given a small cut S we can enumerate the sets $A \in \mathcal{A}$ which are close to S. For this to be useful, it must be that each $A \in \mathcal{A}$ is close to some small cut, and we give the details of this later.

▶ **Lemma 10.** *Fix any $c \geq 1$ and let $t \leq \frac{d}{8c}$. Given a cut C with value at most td, there are at most 4^t closed t-contracting subsets $A \subseteq X$ such that $|A \triangle (C \cap X)| \leq ct$ and $|N(A) \triangle (C \cap Y)| \leq ct$. Moreover, these sets A can be enumerated in time $4^t \cdot n^{O(1)}$.*

Proof. Let $A' := C \cap X$ and $W' := C \cap Y$. By the fact that G is d-regular, $|E(A', W')| \leq d\min\{|A'|, |W'|\}$ and hence $|\nabla(C)| \geq d\max\big\{|W'| - |A'|, |A'| - |W'|\big\}$. By assumption, we have $|\nabla(C)| \leq td$ and therefore $\big||W'| - |A'|\big| \leq t$.

Set

$$S_X := \big\{v \in X \setminus A' : |N(v) \setminus W'| \leq 3ct\big\}, \text{ and}$$
$$S_Y := \big\{v \in W' : |N(v) \cap A'| \leq ct\big\},$$

so that $S_X \subseteq X$ consists of vertices in $X \setminus A'$ with almost all of their neighbors in W' and $S_Y \subseteq Y$ consists of vertices in W' with almost all of their neighbors in $X \setminus A'$. We have the following claims.

▷ Claim 11. For any closed t-contracting subset $A \subseteq X$ such that $|A \triangle A'| \leq ct$, $A \setminus A' \subseteq S_X$.

Proof. Suppose for contradiction that there is a vertex $v \in A \setminus A'$ such that

$$|N(v) \setminus W'| > 3ct.$$

We derive the contradiction using the facts that $|\nabla(A \cap A')| = d|A \cap A'|$ and that any of the edges in $\nabla(A \cap A')$ not incident to W' contribute to the value of the cut C. These facts imply that $|E(A \cap A', W')| \geq d|A \cap A'| - t \cdot d$, and hence

$$|N(A \cap A') \cap W'| \geq |A \cap A'| - t.$$

Then because A is closed and non-expanding,

$$\begin{aligned} |A| + t &\geq |N(A)| \geq |N((A \cap A') \cup \{v\})| \\ &> |N((A \cap A') \cup \{v\}) \cap W'| + 3ct \\ &\geq |A \cap A'| + 2ct \geq |A| + ct, \end{aligned}$$

which is a contradiction because there is a strict inequality in the chain and $c \geq 1$. ◁

▷ Claim 12. For any t-contracting subset $A \subseteq X$ such that $|A \triangle A'| \leq ct$, $W' \setminus N(A \cap A') \subseteq S_Y$.

Proof. We note that for each vertex v in $W' \setminus N(A \cap A')$, we have that

$$N(v) \cap A' \subseteq A' \setminus A.$$

Since $|A' \setminus A| \leq ct$, it follows that $|N(v) \cap A'| \leq ct$. ◁

We can now complete the proof of the lemma. Using the degree constraints in the definitions of S_X and S_Y, we have

$$\begin{aligned} td &\geq |\nabla(C)| \\ &\geq |S_X|(d - 3ct) + |S_Y|(d - ct) \\ &\geq (d/2) \cdot (|S_X| + |S_Y|) \end{aligned}$$

where the last inequality uses $t < \frac{d}{8c}$. As a result, we have

$$|S_X| + |S_Y| \leq 2t.$$

Putting Claim 11 and Claim 12 together, we have that each closed t-contracting sets A with $|A \triangle A'|,\ |N(A) \triangle W'| \leq ct$ must be of the form

$$A = [(A' \setminus N(S'_Y)) \cup S'_X]$$

for some subsets $S'_Y \subseteq S_Y$ and $S'_X \subseteq S_X$. Thus, the total number of such A is at most $2^{|S_X|+|S_Y|} \leq 4^t$. Since we are given the cut C, S_X and S_Y can be found in time polynomial in n as required. ◀

With these ingredients we can proof Lemma 5, which we recall states that $\mathcal{A}$ can be enumerated in time $n^{O(1/\delta)}4^{t_0}$.

Proof of Lemma 5. Since $d = \lfloor \delta n \rfloor$, we construct $\mathcal{C}^{\text{cut}}$ as in Lemma 9 in time $n^{O(1/\delta)}$. We then choose $c = 32$ and enumerate for each $C \in \mathcal{C}^{\text{cut}}$, every closed t_0-contracting subset A with $|A \triangle (C \cap X)| \leq 32t_0$ and $|N(A) \triangle (C \cap Y)| \leq 32t_0$ using Lemma 10. We are done if every $A \in \mathcal{A}$ appears in this enumeration process, as the running times combine to give the required $n^{O(1/\delta)}4^{t_0}$. This holds because each $A \in \mathcal{A}$ is closed and t_0-contracting and hence setting $S_A = A \cup N(A)$, by a double counting argument, we have $|\nabla(S_A)| = d|N(A)| - d|A| \leq t_0 d$. So each $A \in \mathcal{A}$ corresponds to a cut of value at most $t_0 d$ and hence some $C \in \mathcal{C}^{\text{cut}}$ has $|S_A \triangle C| \leq 32t_0$ by Lemma 9. ◀

5 Approximating the number of covers

For convenience, we restate Lemma 6 here.

▶ **Lemma 6.** *Let $A \subseteq X$ be a closed t_0-contracting polymer. Then for $\epsilon', \rho' > 0$ there is a randomized algorithm running in time polynomial in n, $1/\epsilon'$ and $\log(1/\rho')$ that with probability at least $1 - \rho'$ outputs a relative ϵ'-approximation to the number of polymers $B \subseteq A$ such that $N(B) = N(A)$.*

Proof. The method is exactly the same as [32, Lem. 17], but in our setting with $d = \lfloor \delta n \rfloor$ the resulting algorithm runs in time polynomial in n.

Let $|A| = a$, $N(A) = W$ have size $|W| = w$, and let $W' = \{v \in W : |N(v) \cap A| \leq d/2\}$ have size $|W'| = w'$. Let

$$\mathcal{D} = \{B \subseteq A : N(B) = W \text{ and } B \text{ is a polymer}\}$$

be the set whose size we wish to estimate.

By [32, Cor. 10], there is a polymer $A' \subseteq A$ of size at most

$$\frac{2a}{d} \log d + \frac{2w}{d} + 2(w - a) \leq \frac{2}{\delta}(1 + \log n) + 2t_0$$

such that $N(A') = W$. Then $|\mathcal{D}| \geq 2^{a - \left(\frac{2}{\delta}(1+\log n) + 2t_0\right)}$, because any subset of A which contains A' is a polymer. Now $|\mathcal{D}|$ can be estimated to relative error ϵ' with probability at least $1 - \rho$ by sampling

$$\frac{1}{(\epsilon')^2} \log(1/\rho) n^{O(1/\delta)} 4^{t_0}$$

subsets of A uniformly at random, and this can be proved with a suitable application of the Chernoff bound. ◀

6 Enumerative lemmas

In this section we prove Lemma 8 which states that for $A \in \mathcal{A}$ we have $1 \leq \Xi_A \leq e^{\epsilon/2}$.

Proof of Lemma 8. For the proof, we fix an arbitrary $A \in \mathcal{A}$. The terms in the sum giving Ξ_A are non-negative, and the lower bound comes from the term $k = 0$ which contributes 1. For the upper bound, we use recent results on graph containers and adapt them to our purposes.

Recall that a polymer is a 2-linked subset $B \subseteq X$ and that the function Ξ_A involves a sum over tuples of non-t_0-contracting polymers. For convenience, we define $\mathcal{G}(w,t)$ to be the set of t-expanding polymers with neighborhood size w,

$$\mathcal{G}(w,t) = \{B \subseteq X,\ \text{polymer} : |N(B)| = w, |N(B)| - |[B]| = t\}.$$

In terms of this notation, we have

$$\Xi_A = \sum_{k\geq 0} \sum_{\substack{\{B_1,\ldots,B_k\}\in\mathcal{P}_A \text{ compatible} \\ \text{s.t. each } B_i \text{ not } t_0\text{-contracting}}} 2^{-\sum_{i=1}^k |N(B_i)|} \tag{5}$$

$$\leq \sum_{k\geq 0} \frac{1}{k!} \left(\sum_{t\geq t_0} \sum_{w\geq 0} |\mathcal{G}(w,t)| 2^{-w} \right)^k, \tag{6}$$

where we drop the requirement on the tuples of being compatible and relax the requirement that the B_i are subsets of X_A to being subsets of X, and hence have an upper bound. To proceed, we require upper bounds on $|\mathcal{G}(w,t)|$ and split into two cases according to t. The following result is proved in the rest of this section and Appendix A.

▶ **Lemma 13.** *There is an absolute constant $\gamma > 0$ such that for $t_0 \leq t$, and any integer w,*

$$|\mathcal{G}(w,t)| \leq 2^{w-\gamma t}.$$

With this lemma in hand, and because each neighborhood size w that we see is in $[1,n]$, there is an absolute constant $\gamma > 0$ such that

$$\Xi_A \leq \sum_{k\geq 0} \frac{1}{k!} \left(\sum_{t\geq t_0} n 2^{-\gamma t} \right)^k \tag{7}$$

$$= \sum_{k\geq 0} \frac{1}{k!} \left(n \frac{2^{-\gamma t_0}}{1-2^{-\gamma}} \right)^k = \exp\left(n \frac{2^{-\gamma t_0}}{1-2^{-\gamma}} \right). \tag{8}$$

This at most the required $e^{\epsilon/2}$ provided that

$$t_0 \geq \frac{1}{\gamma} \log_2 \left(\frac{2}{1-2^{-\gamma}} \frac{n}{\epsilon} \right),$$

which our choice $t_0 = C \log(n/\epsilon)$ satisfies for all large enough constants $C = C(\delta)$. ◀

Before we proceed with the proof of Lemma 13, we would like to remark out that one of the main contributions of this paper is to handle the case when t is small.

Proof of Lemma 13. We first take care of the case when $t \geq \log^4 n$. For each $v \in V$, let us define

$$\mathcal{G}'(v,w,t) = \{A \in \mathcal{G}(w,t) : v \in A\}.$$

First, we observe that $\log^2 d \cdot \frac{t}{d} \leq \log^2 n \cdot \frac{n}{\delta n} \ll \log^4 n$. Lemma 4 in [32] gives us that there is a constant c such that for each v, $\mathcal{G}'(v,w,t) \leq 2^{w-ct}$. Thus, we have

$$|\mathcal{G}(w,t)| \leq \sum_v |\mathcal{G}'(v,w,t)| \leq n \cdot 2^{n-ct} \leq 2^{n-ct/2}$$

for n large enough.

Before we address the case when $t_0 \leq t < \log^4 n$, let us set up some additional notation. Given a vertex $v \in V$ and a subset $S \subseteq V$, we write $d_S(v)$ for the number of neighbors of v in S.

▶ **Definition 14** (Essential subset). *For a subset $A \subseteq X$, we write $W = N(A)$ and $W_s = \{y \in W : d_A(y) \geq s\}$. We say that F is an* essential set *for A if $W \supseteq F \supseteq W_{d/2}$ and $N(F) \supseteq [A]$.*

It may be useful to consider such an F an approximation for the neighborhood $W = N(A)$.

▶ **Definition 15** (Container). *We call a tuple $(S', T') \in 2^X \times 2^Y$ a γ'-container for a subset $A \subseteq X$ that is t-contracting, with neighborhood $W = N(A)$ if*

1. *$S' \supseteq [A]$ and $W_{d/2} \subseteq T' \subseteq W$,*
2. *$d_{Y \setminus T'}(v) \leq \gamma' t$ for each $v \in S'$, and*
3. *$d_{S'}(v) \leq \gamma' t$ for each $v \in Y \setminus T'$.*

The following two results show the existence of containers and bound the number of sets for which a given container is a γ'-container.

▶ **Lemma 16.** *For any $\gamma' > 0$ and any set $F \subseteq Y$, there is a set $\mathcal{C}^{\text{ind}} \subseteq 2^X \times 2^Y$ of size at most $n^{O(1/\gamma')}$ such that any $A \subseteq X$ for which F is an essential set, has a γ'-container in $\mathcal{C}^{\text{ind}}$.*

▶ **Lemma 17.** *There is an absolute constant $\gamma'' > 0$ such that the following holds.*

For any $\gamma' > 0$, $w < n$, $t < \log^4 n$, and tuple $(S', T') \in \mathcal{C}^{\text{ind}}$, there are at most $2^{w - \gamma'' t}$ sets $A \in \mathcal{G}(w, t)$ such that (S', T'), and is a γ'-container for A.

Since the proofs of these results are small modifications of existing container results, e.g. [40], we defer their proofs to Appendix A. We are now ready to handle the case of small t as follows. Consider an integer $t \in [t_0, \log^4 n]$ and a set $A \in \mathcal{G}(w, t)$. Define $S_A := [A] \cup N(A)$. As in the proof of Lemma 5, we have that $|\nabla(S_A)| = d|N(A)| - d|[A]| = td$. By Lemma 9, there is a cut $C \in \mathcal{C}^{\text{cut}}$ such that $g := |S_A \triangle C| \leq O(t)$. Let $A' := C \cap X$ and $W' := C \cap Y$.

Consider the set $W'_g = \{u \in Y : d_{A'}(u) > g\}$. We have the following two claims.

▷ Claim 18. $W \supseteq W'_g \supseteq W_{d/2}$.

Proof. Consider a vertex $u \in W_{d/2}$. We have

$$d_{A'}(u) \geq d_A(u) - |A \setminus A'| \geq d/2 - |L \triangle L'| \geq d/2 - g > g,$$

where the last inequality holds since $d = \lfloor \delta n \rfloor$ and $g = O(\log^4 n)$. Therefore $u \in W'_g$. Moreover, consider a vertex $u \in W'_g$. We have

$$d_A(u) \geq d_{A'}(u) - |A' \setminus A| > g - |L \triangle L'| > 0,$$

and hence $u \in W$. ◁

▷ Claim 19. $A \subseteq N(W'_g)$.

Proof. Suppose otherwise, i.e. there is a vertex $u \in A$ such that for each vertex $v \in N(u)$ we have $d_{A'}(v) \leq g$. For any such v, we have

$$d_A(v) \leq d_{A'}(v) + |A \setminus A'| \leq d_{A'}(v) + |L \triangle L'| \leq 2g.$$

This gives us that

$$t \cdot d = |E(W, X \setminus A)| \geq |E(N(u), W \setminus A)| \geq d(d - 2g),$$

contradicting the assumptions that $d = \lfloor \delta n \rfloor$ and t and g are both $O(\log^4 n)$. ◁

Claims 18 and 19 show that W'_g is an essential set for A. The set $A \in \mathcal{G}(w,t)$ may be constructed by

1. choosing the appropriate cut C in the set $\mathcal{C}^{\text{cut}}$ constructed in Lemma 9,
2. constructing the essential subset W'_g for it as above,
3. using Lemma 16 to obtain a γ'-container of A, where γ' is the absolute constant of Lemma 17, and finally
4. reconstructing A from the γ'-container with Lemma 17.

There are $n^{O(1/\delta)}$ choices for L' in the first step, a unique construction of W'_g for the second, $n^{O(1/\gamma')}$ possible containers in the third step, and $2^{w-\gamma'' t}$ ways for the final step. In total there are

$$2^{w-\gamma'' t + O(1/\gamma' + 1/\delta)\log n} \leq 2^{w-\gamma'' t/2}$$

such sets $A \in \mathcal{G}(w,t)$. The last inequality comes from our assumption that $t \geq t_0$ for our choice of $t_0 = C(\delta)\log(n/\epsilon) \geq C\log(n)$ (because wlog $\epsilon \leq 1$) satisfying

$$t_0 \geq \Omega\left(\frac{\log n}{\gamma''}\left(\frac{1}{\gamma'} + \frac{1}{\delta}\right)\right). \qquad \blacktriangleleft$$

7 Concluding remarks and future directions

1. Naturally, a next goal is to understand the power and limitations of the methods presented, especially in conjunction with existing cluster expansion methods. More specifically, we are curious about the following two questions:
   a. Can this spectral point of view help with our understanding of independent sets in a larger class of bipartite graphs?
   b. To what extent do these methods help in reducing the computation needed to implement algorithmic cluster expansion?

   In this context, the problem of approximating the number of independent sets in *small-set expanders* feels within striking distance.
2. Our next remark concerns Lemma 9. As mentioned before, similar results have had other applications in optimization and Unique Games [2, 36, 37], though we take a subtly different viewpoint worth noting: we seek to approximate *all* cuts in the graph, not just small ones. In any case, we find the lemma interesting in its own right and conjecture something stronger.

   ▶ **Conjecture 20.** *Lemma 9 holds with* $|\mathcal{C}^{\text{cut}}| \leq 2^{O(n/d)}$.

   If true, this would be best possible, as evidenced by a disjoint union of $1/\delta$ components. Setting $t = 0$ in this case gives exactly $2^{1/\delta}$ cuts of size 0.
3. Finally, we leave open the problem of making our algorithm deterministic. At the moment, the only step where randomness is used is Lemma 6.

References

1 J. D. Annan. A Randomised Approximation Algorithm for Counting the Number of Forests in Dense Graphs. *Combinatorics, Probability and Computing*, 3(3):273–283, 1994. doi:10.1017/S0963548300001188.

2 Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential Algorithms for Unique Games and Related Problems. *Journal of the ACM*, 62(5):1–25, 2015. doi:10.1145/2775105.

3 Sanjeev Arora, David Karger, and Marek Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '95, pages 284–293, New York, NY, USA, 1995. Association for Computing Machinery. doi:10.1145/225058.225140.

4 Sanjeev Arora, Subhash A. Khot, Alexandra Kolla, David Steurer, Madhur Tulsiani, and Nisheeth K. Vishnoi. Unique games on expanding constraint graphs are easy: Extended abstract. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, pages 21–28, Victoria British Columbia Canada, 2008. ACM. doi:10.1145/1374376.1374380.

5 Victor Bapst and Amin Coja-Oghlan. Harnessing the Bethe free energy. *Random Structures & Algorithms*, 49(4):694–741, 2016. doi:10.1002/rsa.20692.

6 Boaz Barak, Prasad Raghavendra, and David Steurer. Rounding Semidefinite Programming Hierarchies via Global Correlation. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 472–481, 2011. doi:10.1109/FOCS.2011.95.

7 Antonio Blanca, Andreas Galanis, Leslie Ann Goldberg, Daniel Stefankovic, Eric Vigoda, and Kuan Yang. Sampling in Uniqueness from the Potts and Random-Cluster Models on Random Regular Graphs. In Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018)*, volume 116 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2018.33.

8 Sarah Cannon and Will Perkins. Counting independent sets in unbalanced bipartite graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1456–1466, 2020. doi:10.1137/1.9781611975994.88.

9 Charles Carlson, Ewan Davies, and Alexandra Kolla. Efficient algorithms for the Potts model on small-set expanders. *To appear in Chicago Journal of Theoretical Computer Science*, March 2020. arXiv:2003.01154.

10 Charlie Carlson, Ewan Davies, Nicolas Fraiman, Alexandra Kolla, Aditya Potukuchi, and Corrine Yap. Algorithms for the ferromagnetic Potts model on expanders. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 344–355, 2022. doi:10.1109/FOCS54457.2022.00040.

11 Zongchen Chen, Andreas Galanis, Leslie A. Goldberg, Will Perkins, James Stewart, and Eric Vigoda. Fast algorithms at low temperatures via Markov chains. *Random Structures & Algorithms*, 58(2):294–321, 2021. doi:10.1002/rsa.20968.

12 Fan Chung. *Spectral Graph Theory*, volume 92 of *CBMS Regional Conference Series in Mathematics*. American Mathematical Society, December 1996. doi:10.1090/cbms/092.

13 Amin Coja-Oghlan, Colin Cooper, and Alan Frieze. An Efficient Sparse Regularity Concept. *SIAM Journal on Discrete Mathematics*, 23(4):2000–2034, 2010. doi:10.1137/080730160.

14 Amin Coja-Oghlan and Will Perkins. Belief Propagation on Replica Symmetric Random Factor Graph Models. In Klaus Jansen, Claire Mathieu, José D. P. Rolim, and Chris Umans, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2016)*, volume 60 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2016.27.

15 Amin Coja-Oghlan and Will Perkins. Bethe States of Random Factor Graphs. *Communications in Mathematical Physics*, 366(1):173–201, February 2019. doi:10.1007/s00220-019-03387-7.

16 Martin Dyer, Alan Frieze, and Mark Jerrum. Approximately counting Hamilton cycles in dense graphs. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '94, pages 336–343, USA, 1994. Society for Industrial and Applied Mathematics. URL: https://dl.acm.org/doi/abs/10.5555/314464.314557.

17 Martin Dyer, Leslie Ann Goldberg, Catherine Greenhill, and Mark Jerrum. The Relative Complexity of Approximate Counting Problems. *Algorithmica*, 38(3):471–500, 2004. doi:10.1007/s00453-003-1073-y.

18 A. Frieze. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, FOCS '96, page 21, USA, 1996. IEEE Computer Society.

19 A. Frieze and R. Kannan. The regularity lemma and approximation schemes for dense problems. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 12–20, 1996. doi:10.1109/SFCS.1996.548459.

20 Andreas Galanis, Qi Ge, Daniel Štefankovič, Eric Vigoda, and Linji Yang. Improved inapproximability results for counting independent sets in the hard-core model. *Random Structures & Algorithms*, 45(1):78–110, 2014. doi:10.1002/rsa.20479.

21 Andreas Galanis, Leslie Ann Goldberg, and James Stewart. Fast Algorithms for General Spin Systems on Bipartite Expanders. *ACM Transactions on Computation Theory*, 13(4):25:1–25:18, 2021. doi:10.1145/3470865.

22 David Galvin. A Threshold Phenomenon for Random Independent Sets in the Discrete Hypercube. *Combinatorics, Probability and Computing*, 20(1):27–51, 2011. doi:10.1017/S0963548310000155.

23 David Galvin and Prasad Tetali. Slow mixing of Glauber dynamics for the hard-core model on regular bipartite graphs. *Random Structures & Algorithms*, 28(4):427–443, 2006. doi:10.1002/rsa.20094.

24 Leslie Ann Goldberg, John Lapinskas, and David Richerby. Faster exponential-time algorithms for approximately counting independent sets. *Theoretical Computer Science*, 892:48–84, November 2021. doi:10.1016/j.tcs.2021.09.009.

25 Venkatesan Guruswami and Ali Kemal Sinop. Lasserre Hierarchy, Higher Eigenvalues, and Approximation Schemes for Graph Partitioning and Quadratic Integer Programming with PSD Objectives. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 482–491. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.36.

26 Tyler Helmuth, Matthew Jenssen, and Will Perkins. Finite-size scaling, phase coexistence, and algorithms for the random cluster model on random graphs. *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, 59(2):817–848, 2023. doi:10.1214/22-AIHP1263.

27 Tyler Helmuth, Will Perkins, and Guus Regts. Algorithmic Pirogov–Sinai theory. *Probability Theory and Related Fields*, 2019. doi:10.1007/s00440-019-00928-y.

28 Vishesh Jain, Frederic Koehler, and Andrej Risteski. Mean-field approximation, convex hierarchies, and the optimality of correlation rounding: A unified perspective. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 1226–1236, Phoenix, AZ, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316299.

29 Matthew Jenssen, Peter Keevash, and Will Perkins. Algorithms for #BIS-Hard Problems on Expander Graphs. *SIAM Journal on Computing*, 49(4):681–710, 2020. doi:10.1137/19M1286669.

30 Matthew Jenssen and Will Perkins. Independent sets in the hypercube revisited. *Journal of the London Mathematical Society*, 102(2):645–669, 2020. doi:10.1112/jlms.12331.

31 Matthew Jenssen, Will Perkins, and Aditya Potukuchi. Independent sets of a given size and structure in the hypercube. *Combinatorics, Probability and Computing*, 31(4):702–720, July 2022. doi:10.1017/S0963548321000559.

32 Matthew Jenssen, Will Perkins, and Aditya Potukuchi. Approximately counting independent sets in bipartite graphs via graph containers. *Random Structures & Algorithms*, 63(1):215–241, 2023. doi:10.1002/rsa.21145.

33 Mark Jerrum and Alistair Sinclair. Approximating the Permanent. *SIAM Journal on Computing*, 18(6):1149–1178, 1989. doi:10.1137/0218077.

34 Jeff Kahn and Jinyoung Park. The Number of Maximal Independent Sets in the Hamming Cube. *Combinatorica*, 42(6):853–880, December 2022. doi:10.1007/s00493-021-4729-9.

35 Frederic Koehler, Holden Lee, and Andrej Risteski. Sampling Approximately Low-Rank Ising Models: MCMC meets Variational Methods. In *Proceedings of Thirty Fifth Conference on Learning Theory*, pages 4945–4988. PMLR, June 2022. URL: https://proceedings.mlr.press/v178/koehler22a.html.

36 Alexandra Kolla. Spectral Algorithms for Unique Games. In *2010 IEEE 25th Annual Conference on Computational Complexity*, pages 122–130, Cambridge, MA, USA, June 2010. IEEE. doi:10.1109/CCC.2010.20.

37 Alexandra Kolla and Madhur Tulsiani. Playing random and expanding unique games. Unpublished, 2007. URL: https://home.cs.colorado.edu/~alko5368/UGspec.pdf.

38 Jingcheng Liu and Pinyan Lu. FPTAS for #BIS with Degree Bounds on One Side. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 549–556, New York, NY, USA, June 2015. Association for Computing Machinery. doi:10.1145/2746539.2746598.

39 Shayan Oveis Gharan and Luca Trevisan. A New Regularity Lemma and Faster Approximation Algorithms for Low Threshold Rank Graphs. *Theory of Computing*, 11(1):241–256, 2015. doi:10.4086/toc.2015.v011a009.

40 Jinyoung Park. Note on the Number of Balanced Independent Sets in the Hamming Cube. *The Electronic Journal of Combinatorics*, page P2.34, 2022. doi:10.37236/10471.

41 J. Scott Provan and Michael O. Ball. The Complexity of Counting Cuts and of Computing the Probability that a Graph is Connected. *SIAM Journal on Computing*, 12(4):777–788, 1983. doi:10.1137/0212053.

42 Andrej Risteski. How to calculate partition functions using convex programming hierarchies: Provable bounds for variational methods. In *Conference on Learning Theory*, pages 1402–1416. PMLR, 2016. URL: https://proceedings.mlr.press/v49/risteski16.html.

43 A. A. Sapozhenko. On the number of connected subsets with given cardinality of the boundary in bipartite graphs. *Metody Diskretnogo Analiza*, (45):42–70, 96, 1987.

44 A. A. Sapozhenko. On the number of independent sets in extenders. *Diskretnaya Matematika*, 13(1):56–62, 2001. doi:10.1515/dma.2001.11.2.155.

45 A. Sly and N. Sun. The Computational Hardness of Counting in Two-Spin Models on d-Regular Graphs. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 361–369, 2012. doi:10.1109/FOCS.2012.56.

46 Allan Sly. Computational Transition at the Uniqueness Threshold. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 287–296, Las Vegas, NV, USA, 2010. IEEE. doi:10.1109/FOCS.2010.34.

A Deferred proofs

A.1 Proof of Lemma 16

We restate the result for convenience.

▶ **Lemma 16.** *For any $\gamma' > 0$ and any set $F \subseteq Y$, there is a set $\mathcal{C}^{\text{ind}} \subseteq 2^X \times 2^Y$ of size at most $n^{O(1/\gamma')}$ such that any $A \subseteq X$ for which F is an essential set, has a γ'-container in $\mathcal{C}^{\text{ind}}$.*

Proof. Let $A \subseteq X$ be a subset for which F is an essential set and let $W = N(A)$, $t := |N(A)| - |[A]|$. Consider the following algorithm

---

initialize $T' \leftarrow F$
while $\exists\ v \in [A]$ s.t. $d_{W \setminus T'}(v) > \gamma' t$, pick such a v **do**
  $T' \leftarrow T' \cup N(v)$
end while
initialize $S' \leftarrow \{v \in X : d_{Y \setminus T'}(v) \leq \gamma' t\}$
while $\exists\ v \in Y \setminus W$ s.t. $d_{S'}(v) > \gamma' t$, pick such a v **do**
  $S' \leftarrow S' \setminus N(v)$
end while
$T' \leftarrow T' \cup \{v \in Y : d_S(v) > \gamma' t\}$
return (S', T')

The lemma follows provided we can show that (S', T') as given by the algorithm above is a γ'-container for A by establishing properties 1–3, and provided we can show a good enough bound on the total number of outputs (S', T') which can occur for a fixed F as A varies.

To prove that the output (S', T') is a γ'-container of A, we first show that $S' \supseteq [A]$ and $W_{d/2} \subseteq T' \subseteq W$, establishing 1. Since F is an essential subset for A, we initialize $T' \leftarrow F$, and T' can then only grow, we have $W_{d/2} \subseteq T'$. Clearly, $T' \subseteq W$ at the end of the first while loop. After the second initialize statement, we have that each vertex $v \in [A]$ satisfies $d_{W \setminus T'}(v) \leq d_{Y \setminus T'}(v) \leq \gamma' t$. Therefore, $S' \supseteq A$ at the end of this line. This property is maintained during the second while loop since we only delete $N(v)$ from S for $v \notin W$. This also means that in the penultimate line, all vertices added to T' are from W. Thus $T' \subseteq W$ is also maintained at the end of the algorithm. Next, we prove 3. At the beginning of the second loop, every $v \in S'$ satisfies $d_{Y \setminus T'}(v) \leq \gamma' t$. Since vertices are only removed from S and added to T' after this point, this property is preserved till the end. Finally, to prove 2 note that the penultimate line of the algorithm ensures that every $v \in Y \setminus T$ satisfies $d_S(v) \leq \gamma' t$.

To bound the number of possible outputs for a fixed F, note that before the start of the first loop we have $|W \setminus T'| \leq O(t)$. Each step in the first loop of the algorithm removes γt vertices from $W \setminus T'$. Therefore, this loop runs at most $O(1/\gamma')$ times. Next, each step in the second loop removes at least $\gamma' t$ vertices from $S \setminus [A]$. Immediately after the second initialize statement, we have

$$dt \geq |E(S' \setminus [A], T')| \geq (d - \gamma' t)|S' \setminus [A]|.$$

As a result, $|S' \setminus [A]| = O(t)$. So the second loop runs for at most $1/\gamma'$ steps. The output is determined by the set of $O(1/\gamma')$ vertices chosen in both loops, so the number of possible outputs for the algorithm for a given F is at most $n^{O(1/\gamma')}$. ◀

A.2 Proof of Lemma 17

We restate the result for convenience.

▶ **Lemma 17.** *There is an absolute constant $\gamma'' > 0$ such that the following holds.*

For any $\gamma' > 0$, $w < n$, $t < \log^4 n$, and tuple $(S', T') \in \mathcal{C}^{\text{ind}}$, there are at most $2^{w - \gamma'' t}$ sets $A \in \mathcal{G}(w, t)$ such that (S', T'), and is a γ'-container for A.

We need the following lemma

▶ **Lemma 21.** *Let (S', T') be a γ'-container for a set $A \in \mathcal{G}(w, t)$. Then $|S'| \leq |T'|$.*

Proof. Let us denote $W = N(A)$. First, we observe that $|E(S', W)| \leq d|T'| + \gamma' t|W \setminus T'|$ by 3. We also have that $|E(S', W)| \geq d|[A]| + |S' \setminus [A]|(d - \gamma' t) = d|S'| - \gamma' t|S' \setminus [A]|$ by 1 and 2. Combining these inequalities, we have

$$|S'| \leq |T'| + \frac{\gamma' t(|S' \setminus [A]| + |W \setminus |T'||)}{d}. \tag{9}$$

Since $T' \supseteq W_{d/2}$, we have that $|W \setminus T'| \leq O(t)$ and

$$td = |E(W, X \setminus [A])| \geq \sum_{v \in S' \setminus [A]} d_{T'}(v) \geq |S' \setminus [A]|(d - \gamma' t)$$

which gives $|S' \setminus [A]| = O(t)$. So (9) implies

$$|S'| \leq |T'| + O\left(\frac{\gamma' t^2}{d}\right).$$

Since $t \leq \log^4 n$, $d = \lfloor \delta n \rfloor$, and $|S'|$ and $|T'|$ are both integers, we have that $|S'| \leq |T'|$. ◀

We finish the proof using the following lemma from [40], whose proof we reproduce for clarity.

▶ **Lemma 22** ([40], Lemma 11). *There is an absolute constant $\gamma'' > 0$ such that the following holds.*

For any tuple $(S', T') \in 2^X \times 2^Y$ such that $|S'| \leq |T'|$, there are at most $2^{w-\gamma''t}$ sets $A \in \mathcal{G}(w,t)$ such that $[A] \subseteq S'$ and $T' \subseteq N(A)$.

To be precise, in [40] the graph in question is the d-dimensional hypercube and additional hypotheses are stated, namely $w - t < n/4$ and $w > d^4$. These play no role in the proof, however, and it extends verbatim to the result stated above.

Proof. Throughout, we denote $W = N(A)$, and let $\alpha > 0$ be a constant that will be determined later.

If $|S'| < w - \alpha t$, then A is among the possible $2^{w-\alpha t}$ subsets of S'. Suppose otherwise, that $|S'| > w - \alpha t$. Let $A^* \in \mathcal{G}(w,t)$ such that (S', T') is a γ'-container for A^* and let $W^* = N(A^*)$. We have that $[A]$ is completely determined by $W \setminus W^*$ and $W^* \setminus W$. Since $W^* \setminus W \subseteq W^* \setminus T$, and

$$|W^* \setminus T'| \leq |W^*| - |T'| = |W| - |T'| \leq |W| - |S'| \leq \alpha t,$$

there are at most $2^{\alpha t}$ choices for $W^* \setminus W$. Next, for each vertex in $W \setminus W^*$, we choose a neighbor in $A \setminus A^* \subseteq S' \setminus A^*$. Observe that $W \setminus W^* = N(A \setminus A^*) \setminus W^*$. Since

$$|W \setminus W^*| \leq |W \setminus F| = |W| - |F| \leq |W| - |S'| \leq \alpha t,$$

and

$$|S' \setminus A^*| \leq |S'| - |A^*| = |S'| - |A| \leq |T'| - |A| \leq |W| - |A| = t.$$

Therefore, the number of choices for $W \setminus W^*$ is at most

$$\binom{t}{\alpha t} \leq 2^{H(\alpha)t}.$$

Once we have $[A]$, there are at most 2^{w-t} possibilities for A. Thus the total number of choices is at most

$$2^{w-t+t(\alpha+H(\alpha))}.$$

Choosing e.g., $\alpha = 0.17$ allows one to choose $\gamma'' = 0.17$. ◀

Vertex-Minor Universal Graphs for Generating Entangled Quantum Subsystems

Maxime Cautrès
Université Grenoble Alpes, CEA-Léti, F-38054 Grenoble, France
École Normale Supérieure de Lyon, F-69007 Lyon, France

Nathan Claudet
Inria Mocqua, LORIA, CNRS, Université de Lorraine, F-54000 Nancy, France

Mehdi Mhalla
Université Grenoble Alpes, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France

Simon Perdrix
Inria Mocqua, LORIA, CNRS, Université de Lorraine, F-54000 Nancy, France

Valentin Savin
Université Grenoble Alpes, CEA-Léti, F-38054 Grenoble, France

Stéphan Thomassé
Université de Lyon, École Normale Supérieure de Lyon, UCBL, CNRS, LIP, F-69007 Lyon, France

Abstract

We study the notion of k-stabilizer universal quantum state, that is, an n-qubit quantum state, such that it is possible to induce any stabilizer state on any k qubits, by using only local operations and classical communications. These states generalize the notion of k-pairable states introduced by Bravyi et al., and can be studied from a combinatorial perspective using graph states and k-vertex-minor universal graphs. First, we demonstrate the existence of k-stabilizer universal graph states that are optimal in size with $n = \Theta(k^2)$ qubits. We also provide parameters for which a random graph state on $\Theta(k^2)$ qubits is k-stabilizer universal with high probability. Our second contribution consists of two explicit constructions of k-stabilizer universal graph states on $n = O(k^4)$ qubits. Both rely upon the incidence graph of the projective plane over a finite field $\mathbb{F}_q$. This provides a major improvement over the previously known explicit construction of k-pairable graph states with $n = O(2^{3k})$, bringing forth a new and potentially powerful family of multipartite quantum resources.

2012 ACM Subject Classification Theory of computation → Quantum information theory; Theory of computation → Quantum communication complexity; Mathematics of computing → Graph theory

Keywords and phrases Quantum networks, graph states, vertex-minors, k-pairability

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.36

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2402.06260` [3]
This work is a follow up of a previous arXiv preprint: `https://arxiv.org/abs/2309.09956` [6]

Funding This work was supported by the PEPR integrated project EPiQ ANR-22-PETQ-0007 part of Plan France 2030, by the STIC-AmSud project Qapla' 21-STIC-10, by the QuantERA grant EQUIP ANR-22-QUA2-0005-01, and by the European projects NEASQC and HPCQS.

1 Introduction

Quantum communication networks often rely on classical communication along with pre-shared entanglement. In this context, a highly pertinent problem is to explore which resource states enable a group of n parties, equipped with the capability of employing Local Operations and Classical Communication (LOCC), to create entangled EPR pairs among any k pairs

of qubits. It is only recently that Bravyi et al. addressed this fundamental question and provided both upper and lower bounds for what they called the k-pairability of quantum states, in terms of the number of parties and the number of qubits per party needed for a quantum state to be k-pairable [2]. Formally, an n-party state $|\psi\rangle$ is said to be *k-pairable* if, for every k disjoint pairs of parties $\{a_1, b_1\}, \ldots, \{a_k, b_k\}$, there exists a LOCC protocol that starts with $|\psi\rangle$ and ends up with a state where each of those k pairs of parties shares an EPR pair. Bravyi et al. studied n-party states in the case where each party holds m qubits, with m ranging from 1 to $\log(n)$. In the case where each party holds at least $m = 10$ qubits, they showed the existence of k-pairable states where k is of the order of $n/\text{polylog}(n)$, which is nearly optimal when m is constant. They also showed that if one allows a logarithmic number of qubits per party, then there exist k-pairable states with $k = n/2$. Moreover, before their work, numerous variations of this problem had surfaced in the literature, some in the context of entanglement routing [16, 26, 27], and some about problems that can be described as variants of k-pairability [5, 7–9, 12, 13, 18, 24, 25].

The notion of k-pairability that we focus on in the present paper relates to the scenario that is both the most natural and challenging [2], when each party possesses precisely one qubit, *i.e.*, $m = 1$. Protocols with multi-qubit parties, require the use of quantum operations acting on two (or more) qubits, which are significantly harder to implement in all the known technologies. For instance in quantum optics, whose 'flying' qubits are well suited for pairability protocols, one-qubit operations are easy to perform using off-the-shelf standard devices, whereas two-qubit operations, like those required by the protocols using multi-qubit parties, can only be performed probabilistically with a non-negligible probability of failure [1,14,21,22]. Bravyi et al. provided some results in the setup where each party holds one single qubit, although arguably weaker than those obtained in the case where each party holds at least 10 qubits. Using Reed-Muller codes, they were able to construct a k-pairable state of size exponential in k, namely $n = 2^{3k}$, leaving the existence of a k-pairable states of size $n = poly(k)$ as an open problem. They also found a 2-pairable graph state of size 10 and proved that there exists no stabilizer state on less than 10 qubits that is 2-pairable using LOCC protocol based on Pauli measurements.

A natural generalization is to consider quantum states satisfying a stronger property: for some integer k, it is possible to induce any stabilizer state on any subset of k qubits, by means of LOCC protocols. We call these states k-stabilizer universal. Stabilizer states constitute a powerful resource for multipartite quantum protocols [17, 19, 23, 28], and can be described, up to local [1] unitaries, by the formalism of graph states: a subset of quantum states which are in one-to-one correspondence with (undirected, simple) graphs. $2k$-stabilizer universality is a stronger notion than k-pairability: any $2k$-stabilizer universal state is k-pairable, as EPR pairs are stabilizer states.

Our contributions rely on the graph state formalism and the ability to characterize properties of quantum states using tools from graph theory. In particular, we reformulate k-pairability as a property of a graph (rather than a property of a quantum state), such that the graph state corresponding to a k-pairable graph, is k-pairable. Furthermore, we relate pairability to the standard notion of vertex-minor (a complete and up-to-date survey on vertex-minors can be found in [20]). A graph H is a vertex-minor of G if one can transform G into H by means of local complementations[2] and vertex deletions. If H is a vertex-minor of G then the graph state $|H\rangle$ can be obtained from $|G\rangle$ using only local Clifford operations,

[1] As we consider one qubit per party, "local" is to be understood as "on each single qubit independently".
[2] Local complementation on a vertex u consists in complementing the subgraph induced by its neighbors.

local Pauli measurements and classical communications. Dahlberg, Helsen, and Wehner proved that the converse is also true when H has no isolated vertices [10]. In [9], they proved that it is NP-complete[3] to decide whether a graph state can be transformed into a set of EPR pairs on specific qubits using using only local Clifford operations, local Pauli measurements and classical communications. In [8], they showed that it is also NP-complete to decide whether a graph state can be transformed into another one using using only local Clifford operations, local Pauli measurements and classical communications.

The graphical counterpart of k-stabilizer universal graph states are called k-vertex-minor universal graphs, introduced by some of the authors in [6]: a graph is k-vertex minor universal if it has any graph defined on any k of its vertices as a vertex minor. If a graph is k-vertex-minor universal then the corresponding graph state is k-stabilizer universal. Stabilizer universal states (and thus k-vertex-minor universal graphs) are useful in themselves beyond the fact that they imply pairability, as they can serve as a primitive for quantum protocols using multipartite entanglement. For instance, in [6], it is shown that stabilizer universal states constitute a resource to perform a robust pairability protocol, in the sense that it allows some known parties to be malicious, while ensuring the correctness of the protocol. Furthermore, the notion of stabilizer universality is stronger than the notion of pairability. Nevertheless, while previous work [6] establishes the existence of k-stabilizer universal graph states of size $n = O(k^4 \ln(k))$ and of k-pairable graph states of size $n = O(k^3 \ln^3(k))$, there are no known graph states that are k-pairable but not $2k$-stabilizer universal.

In this work, we provide both probabilistic and explicit constructions of k-stabilizer universal graph states resulting from k-vertex-minor universal graphs. While the results are interesting in themselves from a combinatorial perspective, they allow one to explicitly define quantum communication protocols: if a k-stabilizer universal graph state is prepared, and each qubit is sent to a different party, then, with the assumption that each party can perform local quantum operations and that they can share classical information, any stabilizer state on any k qubits can be generated. Note that this includes any set of disjoint EPR pairs on less than k qubits. The local operations to perform in order to induce a given subgraph state derive directly from the proofs of our results.

The main contributions of the paper are as follows. In the first part of the paper, we prove the existence of k-vertex-minor universal graphs of order $n = \Theta(k^2)$, which is optimal as shown in [6]. We adopt a probabilistic approach, exhibiting a family of random bipartite graphs of quadratic order in k, which are k-vertex-minor universal with probability going to 1 exponentially fast in k. On the practical side, in the proof we introduce an efficient algorithm that tries to generate any induced graph of order k as a vertex-minor on any k vertices of a random bipartite graph, and the proof yields a bound on the probability of failure of the algorithm. The second part of the paper focuses on explicit constructions of k-vertex-minor universal graphs. We derive our constructions from the incidence graph of the projective plane over the finite field $\mathbb{F}_q$, where q is a prime power. It is a bipartite graph of order $n = 2(q^2 + q + 1)$, with the same number $(n/2)$ of left and right vertices, corresponding respectively to points and lines of the projective plane (equivalently, 1-dimensional and 2-dimensional linear subspaces of $\mathbb{F}_q^3$). We show it satisfies the k-vertex-minor universality property, with $k = \Theta(n^{1/4})$. Furthermore, we show that the graph on the points of the projective plane, with edges connecting points corresponding to orthogonal 1-dimensional linear subspaces of $\mathbb{F}_q^3$, is k-vertex-minor universal, again with $k = \Theta(n^{1/4})$. To the best of our knowledge, these are the first explicit constructions of k-vertex-minor universal graphs of order polynomial in k, significantly improving on the previous explicit construction of k-pairable states based on Reed-Muller codes from [2], with exponential overhead.

[3] Where the size of the input is the number of bits needed to describe the given graph state.

2 Vertex-minor and stabilizer universality

The goal of this section is to cover different notions related to k-pairability and k-vertex-minor universality properties. We first define the above properties for graphs, then we discuss their implications on the corresponding graph states.

We denote a graph as $G = (V, E)$, where V is the vertex set and E is the edge set. All graphs are assumed to be undirected and simple (without loops or multiple edges). A vertex subset $S \subseteq V$ is said to be **stable** if no two vertices in S are adjacent. Bipartite graphs are denoted as $G = (L, R, E)$, where $V = L \sqcup R$, with L and R disjoint stable sets referred to as **left and right vertex sets**, respectively. To avoid possible confusion, we may sometimes write $V(G), E(G), L(G)$, or $R(G)$. A **pairing** is a graph G such that any vertex is incident to exactly one edge. Given a vertex $v \in V$, we denote by $N_G(v)$ the **neighborhood** of v in G, consisting of vertices $v \in V$ adjacent to v.

A **local complementation** on a vertex v of a graph G consists in complementing the subgraph induced by the neighborhood of v, more precisely, it leads to a graph $G \star v$ such that $V(G \star v) = V(G)$ and $E(G \star v) = E(G) \oplus E(K_{N_G(v)})$ where K_S denotes the complete graph on the vertices in S, and $\oplus$ denotes the symmetric difference of two sets. We say that G' is a **vertex-minor** of G, if G' can be obtained from G by means of local complementations and vertex deletions. Here we consider $V(G') \subseteq V(G)$ and require G' to be obtained exactly (not up to an isomorphism of graphs), meaning that there exists a sequence of graph transformations consisting of local complementations and the deletions of the vertices of $V(G) \setminus V(G')$.

▶ **Definition 1.** *Given a graph G, a vertex subset $V' \subseteq V(G)$, and an integer $k > 0$, we say that:*

- *G is k-**vertex-minor universal** on V', if $k \leqslant |V'|$ and any graph on any k vertices in V' is a vertex-minor of G.*
- *G is k-**pairable** on V', if $k \leqslant |V'|/2$ and any pairing on any $2k$ vertices in V' is a vertex-minor of G.*

If any of the above properties is satisfied with $V' = V(G)$, we say that G is k-vertex-minor universal *or that G is* k-pairable*, respectively.*

▶ **Definition 2.** *We say that a bipartite graph $G = (L, R, E)$ is **left** (resp. **right**) k-**vertex-minor universal** or k-**pairable** if the corresponding condition from Definition 1 is satisfied for $V' = L$ (resp. $V' = R$). We say that G is **two-side** k-vertex-minor universal / k-pairable if it is both left and right k-vertex-minor universal / k-pairable.*

Graph states form a standard family of quantum states that can be represented using simple undirected graphs (Ref. [17] is an excellent introduction to graph states). Given a graph $G = (V, E)$, the corresponding **graph state** $|G\rangle$ is the $|V|$-qubit state:

$$|G\rangle = \frac{1}{\sqrt{2}^{|V|}} \sum_{x \in 2^V} (-1)^{|G[x]|} |x\rangle$$

where $|G[x]|$ is the size (number of edges) of the subgraph induced by x, and $|x\rangle$ is the corresponding base vector in the Hilbert space[4].

[4] With a slight abuse of notation we identify a subset (say $x = \{u_2, u_4\}$) of the set of qubits $V = \{u_1, \ldots, u_5\}$ with its characteristic binary word ($x = 01010$).

We shall alternatively refer to the vertex set V as qubit set. A graph state $|G\rangle$ can be prepared as follows: initialize every qubit in $|+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ then apply for each edge of the graph a CZ gate on the corresponding pair of qubits, where $CZ : |ab\rangle \mapsto (-1)^{ab}|ab\rangle$. The graph state $|G\rangle$ is the unique quantum state (up to a global phase) that, for every vertex $u \in V$, is a fixed point of the Pauli operator $X_u Z_{N_G(u)}$.[5] Hence, graph states form a subfamily of stabilizer states. Formally, an n-qubit **stabilizer state** [15] is a quantum state that is the simultaneous eigenvector with eigenvalue 1 of n commuting and independent Pauli operators. A useful property is that any stabilizer state is related to some graph state by the application of local Clifford unitaries, and these unitaries can be computed efficiently [11]. For instance, an EPR pair is equal to $|K_2\rangle$ up to local Clifford unitaries, where K_2 is the graph with two vertices and one edge. Thus, under LOCC protocols, generating any graph state on a given set of qubits is equivalent to generating any stabilizer state. We introduce below the notion of k-stabilizer universal states.

▶ **Definition 3.** *A quantum state $|\psi\rangle$ is k-**stabilizer universal** (resp., k-**pairable**) if any stabilizer state on any k qubits in V (resp., any k EPR pairs on any $2k$ qubits in V) can be induced by means of LOCC protocols.*

If H is a vertex-minor of a G then the graph state $|H\rangle$ can be obtained from $|G\rangle$ using only local Clifford operations, local Pauli measurements and classical communications, and the converse is true when H has no isolated vertices [10]. As a pairing on $2k$ vertices has no isolated vertices, we have the following:

▶ **Proposition 4.** *A graph G is k-pairable if and only if the corresponding graph state $|G\rangle$ is k-pairable using only local Clifford operations, local Pauli measurements, and classical communication.*

In the case of vertex-minor universality and stabilizer universality, the characterization from [10] does not apply directly, because of possible isolated vertices. For instance, K_2 is not 2-vertex-minor universal since no local complementation can turn it into an empty graph. However, $|K_2\rangle$ is 2-stabilizer universal: with *e.g.* an X-measurement on each qubit, one can map the corresponding graph state (a maximally entangled pair of qubits) to the graph state composed of a tensor product of two qubits. To be able to state a characterization, a solution is to introduce *destructive* measurements (*i.e.*, the measured qubit is removed from the system and can no longer be used).

▶ **Proposition 5.** *Given two graphs G and H such that $V(H) \subseteq V(G)$, H is a vertex-minor of G if and only if $|H\rangle$ can be obtained from $|G\rangle$ (on the qubits corresponding to $V(H)$) using only local Clifford operations, local destructive Pauli measurements, and classical communications.*

Proof. Notice that a similar statement – involving non-destructive measurements and only valid when H does not contain isolated vertices – has been proved in [10] (Theorem 2.2). We provide here a direct proof of Proposition 5 which is actually slightly simpler thanks to the use of destructive measurements. In the following proof all measurements are destructive. ($\Rightarrow$) Local complementations can be implemented by means of local Clifford unitaries, and vertex deletions by means of Z-measurements together with classical communications and Pauli corrections [11]. ($\Leftarrow$) We prove the property by induction on the number of measurements. If

[5] It consists in applying $X : |a\rangle \mapsto |1-a\rangle$ on u and $Z : |a\rangle \mapsto (-1)^a|a\rangle$ on each of its neighbors in G.

there are no measurements the property is true [11]. Otherwise, let u be the first qubit to be measured. Assume u is measured according to P and C_u is the Clifford operator applied on u before the measurement. $C_u^\dagger P C_u$ is proportional to some Pauli operator $P_0 \in \{X, Y, Z\}$:

(i) if $P_0 = Z$, then the measurement of u can be interpreted as a vertex deletion and leads to $|G \setminus u\rangle$ up to Pauli corrections. By the induction hypothesis, H is a vertex minor of $G \setminus u$, thus of G.

(ii) if $P_0 = Y$, then the measurement of u can be interpreted as a Z-measurement on $|G \star u\rangle$ (up to a Clifford operator on some other qubits), thus according to (i), H is a vertex minor of $G \star u$, so is of G.

(iii) if $P_0 = X$ and $N_G(u) \neq \emptyset$, then the measurement u can be interpreted as a Y-measurement on $|G \star v\rangle$ with $v \in N_G(u)$ (up to local Clifford operations on qubits different from u), thus according to (ii) H is a vertex minor of $G \star v$, so is of G.

(iv) if $P_0 = X$ and $N_G(u) = \emptyset$, then $|G\rangle = |G \setminus u\rangle \otimes |+\rangle_u$ so after the measurement of u the state is $|G \setminus u\rangle$, thus, by the induction hypothesis, H is a vertex minor of $G \setminus u$, so is of G. ◀

▶ **Corollary 6.** *A graph G is k-vertex-minor universal if and only if the corresponding graph state $|G\rangle$ is k-stabilizer universal using only local Clifford operations, local destructive Pauli measurements, and classical communication.*

Relations between pairability, vertex-minor universality and stabilizer universality of graph and graph states, are shown in Figure 1. To the best of our knowledge, all known examples of k-stabilizer universal (resp. k-pairable) graph states come from k-vertex-minor universal (resp. k-pairable) graphs. Furthermore, to date, it is not known whether there exist k-pairable states which are not $2k$-stabilizer universal. Throughout this paper, we will essentially focus on the existence and the explicit construction of k-vertex-minor universal graphs.

$$\begin{array}{ccc} G \text{ is } 2k\text{-vertex-minor universal} & \Longrightarrow & |G\rangle \text{ is } 2k\text{-stabilizer universal} \\ \Downarrow & & \Downarrow \\ G \text{ is } k\text{-pairable} & \Longrightarrow & |G\rangle \text{ is } k\text{-pairable} \end{array}$$

Figure 1 Implications between pairability, vertex-minor universality and stabilizer universality of graphs and graph states.

3 Existence of k-vertex-minor universal graphs of order quadratic in k

Given any k, a k-vertex-minor universal graph has at least a quadratic order in k:

▶ **Proposition 7** ([6]). *If a graph G of order n is k-vertex-minor universal then*

$$k < \sqrt{2n \log_2(3)} + 2.$$

In this section we prove that this bound is tight asymptotically, *i.e.* there exists k-vertex-minor universal graphs whose order grows quadratically with k. This greatly improves over the probabilistic construction obtained by some of us in [6], where the existence of k-vertex-minor universal graphs of order $O(k^4 \ln(k))$ was proven.

▶ **Theorem 8.** *For any constant $\alpha > 2$, there exists k_0 s.t. for any $k > k_0$, there exists a k-vertex-minor universal graph G of order at most αk^2.*

The remaining of this section is a proof of Theorem 8. First we bound the probability that some graph of order k is not a vertex-minor of a random bipartite graph G, in Lemma 10. Then we bound the probability that such a random bipartite graph is k-vertex-minor universal, in Lemma 11, by defining some algorithm that tries to generate any graph as a vertex-minor of G. Finally, we prove that there exists a k-vertex-minor universal bipartite graph of quadratic order in k. More precisely, the probability of a random bipartite graph of quadratic order being k-vertex-minor universal goes to 1 exponentially fast in k:

▶ **Proposition 9.** *Fix constants $\epsilon > 0$, $c > 2$, and $c' > \frac{1+\epsilon}{\ln(2)}$. There exists k_0 s.t. for any $k > k_0$, the random bipartite graph G (the probability of an edge existing between two vertices, one in $L(G)$ and one in $R(G)$, is $1/2$, independently of the other edges) with $|L(G)| = \lfloor c' k \ln(k) \rfloor$ and $|R(G)| = \lfloor ck^2 \rfloor$, is k-vertex-minor universal with probability at least $1 - e^{-\epsilon k \ln(k)}$.*

Proposition 9 will be proved alongside Theorem 8 in this section. Notation-wise, given a set A and an integer k, $\binom{A}{k}$ refers to $\{B \subseteq A \mid |B| = k\}$.

▶ **Lemma 10.** *Consider a random bipartite graph G with $|L(G)| \geqslant k$, $|R(G)| \geqslant 4\binom{k}{2} + 5$: the probability of an edge existing between two vertices (one in $L(G)$ and one in $R(G)$) is $1/2$, independently of the other edges. Take $k \in \mathbb{N}$ and consider a set of vertices $K \in \binom{L(G)}{k}$. The probability that there exists a graph defined on K which is not a vertex-minor of G is upper bounded by $e^{-\frac{\left(\frac{|R(G)|}{4} - \binom{k}{2} + 1\right)^2}{\left(\frac{7|R(G)|}{4} - \binom{k}{2} + 1\right)}}$.*

Proof. For some $j \in \mathbb{N} \setminus \{0\}$ and $X \in \binom{R(G)}{j}$, consider the incidence matrix M_X of size $j \times \binom{k}{2}$, whose column i represents the pairs of vertices of K that are in the neighborhood of the i^{th} vertex of X, in the sense that its entries are 1 if the pair of vertices u,v is in its neighborhood, 0 else. Note that if there exists some $X \in \binom{R(G)}{\binom{k}{2}}$ whose incidence matrix M_X is of full column-rank, then any $2^{\binom{k}{2}}$ graph defined on K is a vertex-minor of G. Indeed, column number i represents the edges (resp. non-edges) of K to be toggled by a local complementation on the i^{th} vertex of X. So now we will bound the probability of such a set X existing within $R(G)$.

For this purpose we will greedily try to construct the set $X \in \binom{R(G)}{\binom{k}{2}}$, one vertex after the other, by considering each vertex in $R(G)$ one by one, and we will lower bound the probability of the event "there exists some $X \in \binom{R(G)}{\binom{k}{2}}$ whose incidence matrix M_X is of full column-rank" by the probability of success of the algorithm. The algorithm works as follows. Arbitrarily order the vertices of $R(G)$. At each step (say that we have j vertices in X at some step), suppose the corresponding matrix of incidence (of size $j \times \binom{k}{2}$) full column-rank. We consider the next vertex $u \in R(G)$ in the list: if adding its corresponding vector to M_X increases its column-rank, then we add u to X, else we remove u from the vertices to consider. The algorithm stops (and succeeds) if M_X has $\binom{k}{2}$ columns and is full column-rank. Let us show that the probability of a vertex u increasing the column-rank of M_X (if $j < \binom{k}{2}$) is lower-bounded by $1/4$.

If M_X is of rank $j < \binom{k}{2}$, there exists a non-zero vector W (*i.e.* a set of pairs of vertices of K) which is orthogonal to all j first vectors. W can be seen as the characteristic function of the edges of some graph H on the vertices of $L(G)$. Adding a vertex u to X increases the

rank of M_X if the vector U of incidence of u in K is such that $U \cdot W = 1 \mod 2$ (because then U is not in the span of M_X). Note that, if H has exactly one edge, then there is exactly probability $\frac{1}{4}$ that $U \cdot W = 1 \mod 2$ (in this case the two ends of the unique edge of H are connected to u, which happens with probability $\frac{1}{2} \times \frac{1}{2}$). As H has at least one edge, it has at least one vertex of non-zero degree z. Let us draw randomly the neighborhood of u: first we draw among the vertices of $H \setminus \{z\}$, then we add z with probability $\frac{1}{2}$. The probability that an odd number of neighbors of z are neighbors of u is $1/2$, so drawing z changes the parity of the number of edges in H whose ends are both neighbors of u, with probability $1/2$. At the end of the day there is a probability of at least $\frac{1}{4}$ that $U \cdot W = 1 \mod 2$, so that u increases the column-rank of M_X.

Finally, the algorithm fails if we encounter more than $|R(G)| - \binom{k}{2} + 1$ vertices that did not increase the column-rank of M_X. Let us introduce a random variable T that follows the distribution $B(|R(G)|, 3/4)$. The probability that the algorithm fails is upper bounded by $\Pr(T \geqslant |R(G)| - \binom{k}{2} + 1)$. We will use the Chernoff bound: With $\mu = \mathbb{E}[T] = \frac{3|R(G)|}{4}$, for any $\delta > 0$, $\Pr(T \geqslant (1+\delta)\mu) \leqslant e^{-\frac{\delta^2}{2+\delta}\mu}$. As we need $(1+\delta)\mu = |R(G)| - \binom{k}{2} + 1$, we take $\delta = \frac{|R(G)| - \binom{k}{2} + 1 - \mu}{\mu}$. From $|R(G)| \geqslant 4\binom{k}{2} + 5$ it follows that $\delta > 0$. The Chernoff bound then gives

$$\Pr\left(T \geqslant |R(G)| - \binom{k}{2} + 1\right) \leqslant e^{-\frac{\left(\frac{|R(G)| - \binom{k}{2} + 1 - \mu}{\mu}\right)^2}{\left(\frac{|R(G)| - \binom{k}{2} + 1 + \mu}{\mu}\right)}\mu} = e^{-\frac{\left(|R(G)| - \binom{k}{2} + 1 - \mu\right)^2}{\left(|R(G)| - \binom{k}{2} + 1 + \mu\right)}} = e^{-\frac{\left(\frac{|R(G)|}{4} - \binom{k}{2} + 1\right)^2}{\left(\frac{7|R(G)|}{4} - \binom{k}{2} + 1\right)}}$$

So the probability of the existence of $X \subseteq \binom{R(G)}{k}$ whose incidence matrix M_X if of full column-rank is lower bounded by $1 - e^{-\frac{\left(\frac{|R(G)|}{4} - \binom{k}{2} + 1\right)^2}{\left(\frac{7|R(G)|}{4} - \binom{k}{2} + 1\right)}}$. ◀

▶ **Lemma 11.** *Consider a random bipartite graph G with $|L(G)| \geqslant k$, $|R(G)| \geqslant 4\binom{k}{2} + 5$: the probability of an edge existing between two vertices (one in $L(G)$ and one in $R(G)$) is $1/2$, independently of the other edges. The probability that G is k-vertex-minor universal is lower bounded by*

$$1 - \left(\frac{k}{2^{|L(G)|-k+1}} + e^{-\frac{\left(\frac{|R(G)|}{4} - \binom{k}{2} + 1\right)^2}{\left(\frac{7(|R(G)|-k)}{4} - \binom{k}{2} + 1\right)}} \right) \times \binom{|L(G)| + |R(G)|}{k}$$

The proof makes use of the union bound along with Lemma 10, and can be found in the extended version of this paper [3]. Roughly speaking, the proof introduces an algorithm that makes use of pivoting to get all k vertices of some set $K \subseteq V(G)$ on the left side of the bipartite graph, in order to use Lemma 10 properly.

▶ Remark 12. Lemma 11 has concrete applications on its own right: in particular for any integer k, it yields an integer n such that there exists a (bipartite) k-vertex-minor universal graph of order n. In general, one can infer a lower bound on the probability of generating a k-vertex-minor universal graph, for any choice of k and n, using the algorithm presented in the proof of Lemma 11. A table presenting orders for which some bipartite k-vertex-minor universal graph exists, as well as orders for with a randomly generated bipartite graph is k-vertex-minor universal with at least 99% probability, for particular values of k ranging from 3 to 100, can be found in Appendix A. Surprisingly enough, we observe that a small constant additive overheard in the order of the graph is sufficient to attain a high probability of generating a k-vertex-minor universal graph.

Now we are ready to conclude. Fix some constants $c > 2$ and $c' > \frac{1}{\ln(2)}$. Let G be a random bipartite graph G with $|L(G)| = \lfloor c'k\ln(k)\rfloor$ and $|R(G)| = \lfloor ck^2\rfloor$: the probability of an edge existing between two vertices (one in $L(G)$ and one in $R(G)$) is $1/2$, independently of the other edges.

Note $n = |V| = |L(G)| + |R(G)| = \lfloor c'k\ln(k)\rfloor + \lfloor ck^2\rfloor$. Using Lemma 11, the probability that G is k-vertex-minor universal is lower bounded by

$$1 - \left(\frac{k}{2^{|L(G)|-k+1}} + e^{-\frac{\left(\frac{|R(G)|}{4} - \binom{k}{2}+1\right)^2}{\left(\frac{7(|R(G)|-k)}{4} - \binom{k}{2}+1\right)}} \right) \times \binom{n}{k}$$

Let us prove that this probability is positive with our choice of parameters, for some big enough k. It is sufficient to have:

$$\textbf{(1)}\quad \frac{k}{2^{|L(G)|-k+1}}\binom{n}{k} < \frac{1}{2} \quad \text{and} \quad \textbf{(2)}\quad e^{-\frac{\left(\frac{|R(G)|}{4} - \binom{k}{2}+1\right)^2}{\left(\frac{7(|R(G)|-k)}{4} - \binom{k}{2}+1\right)}}\binom{n}{k} < \frac{1}{2}$$

Let us show that these equations are satisfied for any large enough k. Recall that $\binom{n}{k} \leqslant 2^{nH(k/n)}$ where $H(x) = -x\log_2(x) - (1-x)\log_2(1-x)$ is the binary entropy.

(1): It is sufficient that $\log_2(k) + nH(k/n) - |L(G)| + k - 1 < -1$.

$\log_2(k) + nH(k/n) - |L(G)| + k - 1 \sim_{k\to\infty} n\frac{k}{n}\log_2(\frac{k}{n}) - c'k\ln(k) = k(\log_2(k) - \log_2(n)) - c'k\ln(k) \sim_{k\to\infty} \frac{1}{\ln(2)}k\ln(k) - c'k\ln(k)$. The choice of c' guarantees that for any large enough k, **(1)** is satisfied.

(2): It is sufficient that $nH(k/n)\ln(2) - \frac{\left(\frac{|R(G)|}{4} - \binom{k}{2}+1\right)^2}{\left(\frac{7(|R(G)|-k)}{4} - \binom{k}{2}+1\right)} < -\ln(2)\cdot\frac{\left(\frac{|R(G)|}{4} - \binom{k}{2}+1\right)^2}{\left(\frac{7(|R(G)|-k)}{4} - \binom{k}{2}+1\right)}$ $\sim_{k\to\infty} \frac{\left(\frac{ck^2}{4} - \frac{k^2}{2}\right)^2}{\left(\frac{7ck^2}{4} - \frac{k^2}{2}\right)} = k^2\frac{(c-2)^2}{4(7c-2)}$. We saw above that $nH(k/n)\ln(2) \sim_{k\to\infty} k\ln(k)$. The choice of c guarantees that for any large enough k, **(1)** is satisfied.

This proves that, for any large enough k, G of order $\lfloor c'k\ln(k)\rfloor + \lfloor ck^2\rfloor$, is k-vertex-minor universal with non-zero probability. Taking $\alpha > c$, for any large enough k, $\lfloor c'k\ln(k)\rfloor + \lfloor ck^2\rfloor \leqslant \alpha k^2$, proving Theorem 8.

Furthermore, we just saw that side **(1)** of the equation dominates **(2)** asymptotically. Thus, the probability of G being k-vertex-minor universal is roughly lower bounded by $1 - 2^{\frac{1}{\ln(2)}k\ln(k) - c'k\ln(k)} = 1 - e^{-(\ln(2)c'-1)k\ln(k)}$ as k grows. Then, for any $\epsilon > 0$ such that $\epsilon < ln(2)c' - 1$, for any large enough k, G of order $\lfloor c'k\ln(k)\rfloor + \lfloor ck^2\rfloor$, is k-vertex-minor universal with probability at least $1 - e^{-\epsilon k\ln(k)}$, proving Proposition 9.

4 Vertex-minor universal graphs from projective planes

In this section, we provide explicit constructions of families of k-vertex-minor universal graphs, of order n proportional to k^4. Thus, the order of the constructed graphs scales as the square of the asymptotically optimal graph order from Section 3. We start in Section 4.1 with some preparatory lemmas. In Section 4.2, we introduce a family of *bipartite incidence graphs* of projective planes, and study their k-pairability and k-vertex-minor universality properties. In Section 4.3 we introduce a new family of so-called *reduced graphs* from projective planes, and investigate their k-vertex-minor universality properties.

4.1 Sufficient conditions for k-pairability and k-vertex-minor universality

Below and throughout Section 4, given a graph G, a vertex $v \in V(G)$, and a vertex subset $U \subseteq V(G)$, we shall use the **shorthand notation** $N_U(v) := N_G(v) \cap U$, that is, **the set of neighbors of v that belong to U** (in such a case, we shall always ensure that the context makes the choice of G unambiguous).

The following two lemmas give sufficient conditions for a bipartite graph G to be one-side (*i.e.*, left or right) k-pairable or k-vertex-minor universal. For simplicity, we state these conditions for the set of left vertices.

▶ **Lemma 13.** *Let G be a bipartite graph satisfying the following property:*

(P) *For any set of $2k$ vertices $K = \{u_1, v_1, u_2, v_2, \ldots, u_k, v_k\} \subseteq L(G)$, there exist:*

- **(i)** *a set of k vertices $C = \{c_1, c_2, \ldots, c_k\} \subseteq L(G)$, with $C \cap K = \emptyset$, and*
- **(ii)** *a set of $2k$ vertices $S = \{\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_k, \beta_k\} \subseteq R(G)$, such that $N_{K \cup C}(\alpha_i) = \{u_i, c_i\}$ and $N_{K \cup C}(\beta_i) = \{v_i, c_i\}$, for all $i = 1, \ldots, k$.*

Then G is left k-pairable.

Proof. We use first local complementation on vertices α_i and β_i to create edges (u_i, c_i) and (v_i, c_i), followed by local complementation on vertices c_i to create edges (u_i, v_i), as desired. It is easily seen that no edges are created between u_i and $K \setminus \{v_i\}$, or between v_i and $K \setminus \{u_i\}$. ◀

▶ **Lemma 14.** *Let G be a bipartite graph satisfying the following property:*

(VMU) *For any set of k vertices $K = \{u_1, u_2, \ldots, u_k\} \subseteq L(G)$, there exist:*

- **(i)** *a set of $k(k-1)/2$ vertices $C = \{c_{ij} \mid 1 \leqslant i < j \leqslant k\} \subseteq L(G)$, with $C \cap K = \emptyset$, and*
- **(ii)** *a set of $k(k-1)$ vertices $S = \{\alpha_{ij}, \beta_{ij} \mid 1 \leqslant i < j \leqslant k\} \subseteq R(G)$, such that $N_{K \cup C}(\alpha_{ij}) = \{u_i, c_{ij}\}$ and $N_{K \cup C}(\beta_{ij}) = \{u_j, c_{ij}\}$, for all $1 \leqslant i < j \leqslant k$.*

Then G is left k-vertex-minor universal.

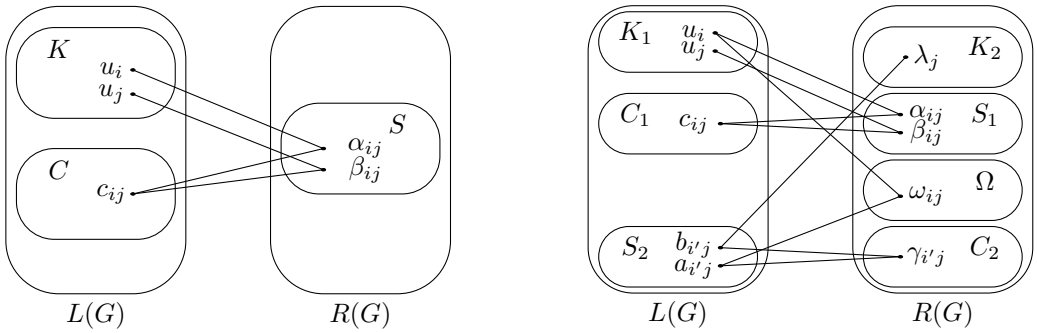
Proof. (See also Figure 2a.) The proof is similar to that of Lemma 13. To create an edge between u_i and u_j, we use first local complementation on vertices α_{ij} and β_{ij}, followed by local complementation on vertex c_{ij}. This procedure does not create any other edge between the vertices of K. ◀

Providing sufficient conditions for a bipartite graph G to be k-vertex-minor universal (on the entire vertex set) is more involved. To induce an arbitrary graph with vertex set $K = K_1 \sqcup K_2$, where $K_1 \subseteq L(G)$ and $K_2 \subseteq R(G)$, we may need to create edges with both endpoints in either K_1 or K_2, which can be dealt with by using conditions similar to those in Lemma 14, but also "toggle" (*i.e.*, either create or remove, as needed) edges between K_1 and K_2, which represents an additional difficulty. We give sufficient conditions for doing so, in the lemma below (see also Figure 2b).

▶ **Lemma 15.** *Let G be a bipartite graph satisfying the following property:*

(VMU*) *For any set of k vertices $K = K_1 \sqcup K_2$, with $K_1 = \{u_1, \ldots, u_{k_1}\} \subseteq L(G)$, and $K_2 = \{\lambda_1, \ldots, \lambda_{k_2}\} \subseteq R(G)$, there exist:*

- **(i)** *a subset $C_1 = \{c_{ij} \mid 1 \leqslant i < j \leqslant k_1\} \subseteq L(G)$, such that $C_1 \cap K_1 = \emptyset$ and $N_{K_2}(C_1) = \emptyset$,*
- **(ii)** *a subset $S_1 = \{\alpha_{ij}, \beta_{ij} \mid 1 \leqslant i < j \leqslant k_1\} \subseteq R(G)$, such that $S_1 \cap K_2 = \emptyset$ and for all $1 \leqslant i < j \leqslant k_1$, $N_{K_1 \sqcup C_1}(\alpha_{ij}) = \{u_i, c_{ij}\}$ and $N_{K_1 \sqcup C_1}(\beta_{ij}) = \{u_j, c_{ij}\}$,*
- **(iii)** *a subset $\Omega = \{\omega_{ij} \mid 1 \leqslant i \leqslant k_1, 1 \leqslant j \leqslant k_2\} \subseteq R(G)$ such that $\Omega \cap (K_2 \sqcup S_1) = \emptyset$ and for all $1 \leqslant i \leqslant k_1, 1 \leqslant j \leqslant k_2$, $N_{K_1 \sqcup C_1}(\omega_{ij}) = \{u_i\}$,*
- **(iv)** *a subset $C_2 = \{\gamma_{ij} \mid 1 \leqslant j \leqslant k_2, j < i \leqslant k_1 + k_2\} \subseteq R(G)$ such that $C_2 \cap (K_2 \sqcup S_1 \sqcup \Omega) = \emptyset$ and $N_{K_1 \sqcup C_1}(C_2) = \emptyset$,*



(a) (VMU) conditions.

(b) (VMU*) conditions where $i' := i + k_2$.

Figure 2 Illustration of the (VMU) and (VMU*) conditions from Lemma 14 and Lemma 15.

(v) *a subset* $S_2 = \{a_{ij}, b_{ij} \mid 1 \leqslant j \leqslant k_2, j < i \leqslant k_1+k_2\} \subseteq L(G)$ *such that* $S_2 \cap (K_1 \sqcup C_1) = \emptyset$ *and for all* $1 \leqslant j \leqslant k_2$, $j < i \leqslant k_1 + k_2$,
- $N_{K_2 \sqcup S_1 \sqcup \Omega \sqcup C_2}(a_{ij}) = \{\lambda_i, \gamma_{ij}\}$ *and* $N_{K_2 \sqcup S_1 \sqcup \Omega \sqcup C_2}(b_{ij}) = \{\lambda_j, \gamma_{ij}\}$, *if* $i \leqslant k_2$
- $N_{K_2 \sqcup S_1 \sqcup \Omega \sqcup C_2}(a_{ij}) = \{\omega_{(i-k_2)j}, \gamma_{ij}\}$ *and* $N_{K_2 \sqcup S_1 \sqcup \Omega \sqcup C_2}(b_{ij}) = \{\lambda_j, \gamma_{ij}\}$, *otherwise.*

Then G *is* k*-vertex-minor universal.*

Proof. We start by removing all vertices that are not in any set defined in (VMU*). Then we proceed in the following three steps.

1) In case we need to create an edge (u_i, u_j) for $1 \leqslant i < j \leqslant k_1$ between two vertices in K_1. We first use local complementations on α_{ij} and β_{ij} to create edges (u_i, c_{ij}) and (u_j, c_{ij}) (no other edges are created) and then remove α_{ij} and β_{ij}. Then, we use local complementation on c_{ij} to create the edge (u_i, u_j) (no other edges are created). Finally, we remove vertex c_{ij}, thus only the edge (u_i, u_j) has been constructed.
2) In case we need to create an edge (λ_i, λ_j) for $1 \leqslant j < i \leqslant k_2$ between two vertices in K_2. We first use local complementations on a_{ij} and b_{ij} to create edges (λ_i, γ_{ij}) and (λ_j, γ_{ij}) (no other edges are created) and then remove a_{ij} and b_{ij}. Then, we use local complementation on γ_{ij} to create the edge (λ_i, λ_j) (no other edges are created). Finally, we remove vertex γ_{ij}, thus only the edge (λ_i, λ_j) has been constructed.
3) In case we need to toggle an edge (u_i, λ_j) for $1 \leqslant i \leqslant k_1$ and $1 \leqslant j \leqslant k_2$ between two vertices in K_1 and K_2. We first use local complementations on $a_{(i+k_2)j}$ and $b_{(i+k_2)j}$ to create edges $(\omega_{ij}, \gamma_{(i+k_2)j})$ and $(\lambda_j, \gamma_{(i+k_2)j})$ (no other edges are created) and then remove $a_{(i+k_2)j}$ and $b_{(i+k_2)j}$. Then, we use local complementation on $\gamma_{(i+k_2)j}$ to create the edge (ω_{ij}, λ_j) (no other edges are created). After that, we remove vertex γ_{ij}, thus only the edge (ω_{ij}, λ_j) has been constructed. Finally, we use local complementation on ω_{ij} to create the edge (u_i, λ_j) (no other edges are created). Then, we remove vertex ω_{ij}, thus only the edge (u_i, λ_j) has been toggled. ◀

The following lemma is a generalization of Lemma 14 to the case of general (not necessarily bipartite) graphs.

▶ **Lemma 16.** *Let* G *be a graph satisfying the following property:*

(VMU$^\circ$) *For any set of* k *vertices* $K = \{u_1, u_2, \ldots, u_k\} \subseteq V(G)$*, there exist:*
- **(i)** *a set of* $k(k-1)/2$ *vertices* $C = \{c_{ij} \mid 1 \leqslant i < j \leqslant k\} \subseteq V(G)$*, such that* C *is stable,* $C \cap K = \emptyset$*, and* $N_K(c_{ij}) = \emptyset$*, for all* $1 \leqslant i < j \leqslant k$*, and*
- **(ii)** *a set of* $k(k-1)$ *vertices* $S = \{a_{ij}, b_{ij} \mid 1 \leqslant i < j \leqslant k\} \subseteq V(G)$*, such that* S *is stable,* $S \cap (K \cup C) = \emptyset$*,* $N_{K \cup C}(a_{ij}) = \{u_i, c_{ij}\}$ *and* $N_{K \cup C}(b_{ij}) = \{u_j, c_{ij}\}$*,* $\forall 1 \leqslant i < j \leqslant k$*.*

Then G *is* k*-vertex-minor universal.*

Proof. Whenever we need to create or to remove an edge between vertices $u_i, u_j \in K$, we use first local complementation on vertices a_{ij} and b_{ij} to create edges between u_i and c_{ij}, and between u_j and c_{ij}, and then we use local complementation on c_{ij}. ◀

4.2 Bipartite graphs from projective planes

Let $q > 0$ be a prime power, $\mathbb{F}_q$ be the finite field with q elements, and $\mathrm{PG}(2,q) := \left(\mathbb{F}_q^3\right)^* / \mathbb{F}_q^*$ be the *projective plane* over $\mathbb{F}_q$. *Points* and *lines* of $\mathrm{PG}(2,q)$ are identified respectively to 1-dimensional and 2-dimensional linear subspaces of $\mathbb{F}_q^3$. A line λ passes through a point a (we write $a \in \lambda$) if the 2-dimensional linear subspace of $\mathbb{F}_q^3$ corresponding to λ contains the 1-dimensional linear subspace corresponding to a. We will use the following properties of the projective plane:

- $\mathrm{PG}(2,q)$ has q^2+q+1 points and q^2+q+1 lines.
- Any line contains exactly $q+1$ points, and any point is contained in exactly $q+1$ lines.
- Any two distinct lines intersect in one point, and for any two distinct points there is one unique line containing them.

We denote by $\mathbb{G}_q$ the bipartite incidence graph of the projective plane $\mathrm{PG}(2,q)$. Precisely, the set of left vertices $L(\mathbb{G}_q)$ is the set of points of $\mathrm{PG}(2,q)$, the set of right vertices $R(\mathbb{G}_q)$ is the set of lines of $\mathrm{PG}(2,q)$, and the set of edges $E(\mathbb{G}_q)$ corresponds to incidences between points and lines, that is $E(\mathbb{G}_q) = \{(a,\lambda) \in L(\mathbb{G}_q) \times R(\mathbb{G}_q) \mid a \in \lambda\}$.

▶ **Theorem 17.** *Let k be such that $k \leqslant (q+4)/5$. Then $\mathbb{G}_q$ is two-side k-pairable.*

Proof. Due to the symmetry of $\mathbb{G}_q$, it is enough to prove it is left k-pairable. For this, we will use Lemma 13. Let $K = \{u_1, v_1, u_2, v_2, \ldots, u_k, v_k\} \subseteq L(\mathbb{G}_q)$ be a set of $2k$ points. To construct the sets $C = \{c_1, c_2, \ldots, c_k\} \subseteq L(\mathbb{G}_q)$ and $S = \{\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_k, \beta_k\} \subseteq R(\mathbb{G}_q)$ from the property (P) in Lemma 13, we will proceed by recursion.

First, since there are $q+1$ lines passing through u_1 and $|K \setminus \{u_1\}| = 2k-1 \leqslant q$, we may choose a line α_1 passing through u_1 and not passing through any other point in $K \setminus \{u_1\}$. Similarly, let β_1 be a line passing through v_1 and not passing through any other point in $K \setminus \{v_1\}$. We take c_1 to be the intersection point between α_1 and β_1.

For $1 \leqslant j < k$, assume that we have constructed a set of j points $C_j = \{c_1, \ldots, c_j\} \subseteq L(\mathbb{G}_q)$ and a set of $2j$ lines $S_j = \{\alpha_1, \beta_1, \ldots, \alpha_j, \beta_j\} \subseteq R(\mathbb{G}_q)$, satisfying the following conditions:

(i) $C_j \cap K = \emptyset$,

(ii) $N_{K \cup C_j}(\alpha_i) = \{u_i, c_i\}$ and $N_{K \cup C_j}(\beta_i) = \{v_i, c_i\}$, for all $i = 1, \ldots, j$.

To construct $\alpha_{j+1}, \beta_{j+1}$, and c_{j+1}, we proceed in the following steps (see also Figure 3).

- **We take α_{j+1} to be any line passing through u_{j+1} and not passing through any other point in $(K \setminus \{u_{j+1}\}) \cup C_j$.**
  This is possible since $|(K \setminus \{u_{j+1}\}) \cup C_j| = 2k-1+j \leqslant 3k-2 \leqslant q$. Moreover, $\alpha_{j+1} \notin S_j$, since by construction no line in S_j passes through u_{j+1}. We further denote by $I_{j+1} \subseteq L(\mathbb{G}_q)$ the set consisting of the intersection points between α_{j+1} and the $2j$ lines in S_j. Thus, $|I_{j+1}| \leqslant 2j$.
- **We take β_{j+1} to be any line passing through v_{j+1} and not passing through any other point in $(K \setminus \{v_{j+1}\}) \cup C_j \cup I_{j+1}$.**
  This is possible since $|(K \setminus \{v_{j+1}\}) \cup C_j \cup I_{j+1}| \leqslant 2k-1+3j \leqslant 5k-4 \leqslant q$. Clearly, $\beta_{j+1} \notin S_j \cup \{\alpha_{j+1}\}$, since no line in $S_j \cup \{\alpha_{j+1}\}$ passes through v_{j+1}.
- **We take c_{j+1} to be the intersection point between α_{j+1} and β_{j+1}.**
  Clearly, $c_{j+1} \notin C_j$, since neither one of α_{j+1} nor β_{j+1} passes through the points in C_j.

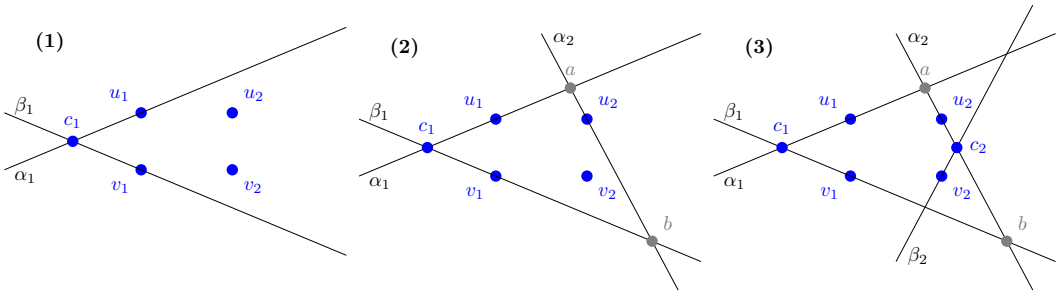


Figure 3 Recursive construction of sets C and S in the proof of of Theorem 17, for $k = 2$. (1) We chose α_1 any line passing through u_1, and not passing through v_1, u_2, v_2. Similarly, we choose β_1 passing through v_1, and not passing through u_1, u_2, v_2. We take c_1 the intersection point between α_1 and β_1. (2) We chose α_2 any line passing through u_2, and not passing through u_1, v_1, c_1, v_2. We determine the intersection points a and b of α_2 with α_1 and β_1. (3) We chose β_2 any line passing through v_2, and not passing through u_1, v_1, c_1, u_2, as well as a, b (to avoid α_2 and β_2 intersecting on these points). We take c_2 the intersection point between α_2 and β_2.

To complete our recursion, we need to prove:

(i) $C_{j+1} \cap K = \emptyset$. We only have to prove that $c_{j+1} \notin K$. This follows from the fact that each of α_{j+1} and β_{j+1} passes through only one point in K, namely u_{j+1} and v_{j+1}, respectively, and they are distinct.

(ii) $N_{K \cup C_{j+1}}(\alpha_i) = \{u_i, c_i\}$ and $N_{K \cup C_{j+1}}(\beta_i) = \{v_i, c_i\}$, for all $i = 1, \dots, j+1$.
For $i = j+1$, the above equalities follow by construction. Indeed, α_{j+1} passes through u_{j+1} and c_{j+1}, but it does not pass through any other point in $(K \setminus \{u_{j+1}\}) \cup C_j$, and similarly, β_{j+1} passes through v_{j+1} and c_{j+1}, but it does not pass through any other point in $(K \setminus \{v_{j+1}\}) \cup C_j$.
For $1 \leqslant i \leqslant j$, we only need to prove that neither α_i nor β_i passes through c_{j+1}. This follows from the fact that β_{j+1} does not pass through any point of I_{j+1}. Indeed, assuming that c_{j+1} belongs to either α_i or β_i, implies it belongs to I_{j+1}, the set of intersection points between α_{j+1} and the lines in S_j. This contradicts the fact that β_{j+1} does not pass through any point of I_{j+1}.

By recursion, we can construct sets $C := C_k$ and $S := S_k$ satisfying the property (P) from Lemma 13, and thus we conclude that $\mathbb{G}_q$ is left k-pairable. ◀

▶ **Theorem 18.** *Let k be such that $3k^2 - k - 8 \leqslant 2q$. Then $\mathbb{G}_q$ is two-side k-vertex-minor universal.*

Proof. Due to the symmetry of $\mathbb{G}_q$, it is enough to prove it is left k-vertex-minor universal. We prove $\mathbb{G}_q$ satisfies the property (VMU) from Lemma 14. Let $K = \{u_1, u_2, \dots, u_k\} \subseteq L(\mathbb{G}_q)$ be a set of k points. To construct the sets $C = \{c_{ij} \mid 1 \leqslant i < j \leqslant k\} \subseteq L(\mathbb{G}_q)$ and $S = \{\alpha_{ij}, \beta_{ij} \mid 1 \leqslant i < j \leqslant k\} \subseteq R(\mathbb{G}_q)$ from Lemma 14 we will proceed again by recursion, by running through pairs (u_i, u_j) in some particular order, say in lexicographical order with respect to indexes (i, j).

The recursion is similar to the one in the proof of Lemma 13. We construct recursively lines α_{ij} and β_{ij}, passing through u_i and u_j, respectively, and take $c_{ij} = \alpha_{ij} \cap \beta_{ij}$. In the recursion, we take α_{ij} to be any line passing through u_i and not passing through any other point in $(K \setminus \{u_i\}) \cup C_{ij}$, where $C_{ij} := \{c_{i'j'} \mid (i', j') < (i, j)\}$. Since $|(K \setminus \{u_i\}) \cup C_{ij}| \leqslant (k-1) + (k(k-1)/2 - 1) = \frac{1}{2}(k^2 + k - 4)$, such a choice of α_{ij} is possible if $k^2 + k - 4 \leqslant 2q$.

A stronger constraint on the value of k comes from the choice of β_{ij}. Indeed, for β_{ij} we take any line passing through u_j and not passing through any other point in $(K \setminus \{u_j\}) \cup C_{ij} \cup I_{ij}$, where I_{ij} is the set of intersection points between α_{ij} and the previously constructed lines $\alpha_{i'j'}$ and $\beta_{i'j'}$, with $(i', j') < (i, j)$. Since $|(K \setminus \{u_j\}) \cup C_{ij} \cup I_{ij}| \leqslant (k-1)+3(k(k-1)/2-1) = \frac{1}{2}(3k^2-k-8)$, we conclude that such a choice of β_{ij} is possible as long as $\frac{1}{2}(3k^2-k-8) \leqslant q$, as stated in the lemma. ◀

▶ **Theorem 19.** *Let k be such that $7k^2 - 16 \leqslant 4q$. Then $\mathbb{G}_q$ is k-vertex-minor universal.*

The proof is done by showing that $\mathbb{G}_q$ satisfies the property (VMU*) from Lemma 15, and can be found in the extended version of this paper [3].

4.3 Reduced graphs from projective planes

▶ **Definition 20.** *Let G be a bipartite graph and $\varphi : L(G) \to R(G)$. The φ-**reduction** of G is the graph G_φ such that:*

- *The vertex set of G_φ is the left vertex set of G, that is $V(G_\varphi) = L(G)$,*
- *There is an edge between $a, b \in V(G_\varphi)$, if $a \neq b$ and either (a and $\varphi(b)$) or (b and $\varphi(a)$) are neighbors in G, that is,*

$$E(G_\varphi) = \{(a, b) \mid a \neq b \text{ and } [\,(a, \varphi(b)) \in E(G) \text{ or } (b, \varphi(a)) \in E(G)\,]\,\}$$

*The reduction is said to be **bijective** if φ is bijective. It is said to be **symmetric** if φ is such that $(a, \varphi(b)) \in E(G) \Leftrightarrow (b, \varphi(a)) \in E(G), \forall a, b \in L(G)$.*

We enforce the condition $a \neq b$ in the definition of $E(G_\varphi)$, in order to avoid loops in case $(a, \varphi(a)) \in E(G)$ for some $a \in L(G)$.

Let G_φ be a bijective, symmetric reduction of G. For any vertex $a \in V(G_\varphi) = L(G)$, let $N_{G_\varphi}(a) \subseteq L(G)$ be the set of neighbors of a in G_φ, and $N_G(a) \subseteq R(G)$ be the set of neighbors of a in G. By definition, if $b \in N_{G_\varphi}(a)$ then $\varphi(b) \in N_G(a)$. The converse is also true, except if $\varphi(a) \in N_G(a)$, or equivalently, $(a, \varphi(a)) \in E(G)$. Hence, $N_{G_\varphi}(a) = \{b \mid \varphi(b) \in N_G(a)\} \setminus \{a\}$, and therefore:

- If $(a, \varphi(a)) \notin E(G)$, the map φ induces a bijection between $N_{G_\varphi}(a)$ and $N_G(a)$. In particular, $|N_{G_\varphi}(a)| = |N_G(a)|$.
- If $(a, \varphi(a)) \in E(G)$, the map φ induces a bijection between $N_{G_\varphi}(a)$ and $N_G \setminus \{\varphi(a)\}$. In particular, $|N_{G_\varphi}(a)| = |N_G(a)| - 1$.

In what follows, we take $\mathbb{G}_q$ to be the bipartite incidence graph of the projective plane $\mathrm{PG}(2, q)$ from the previous section. Let $\varphi : L(\mathbb{G}_q) \to R(\mathbb{G}_q)$ be defined as follows. Recall that a vertex $a \in L(\mathbb{G}_q)$ (that is, a point of the projective plane) corresponds to a 1-dimensional linear subspace of $\mathbb{F}_q^3$, while a vertex $\lambda \in R(\mathbb{G}_q)$ (that is, a line of the projective plane) corresponds to a 2-dimensional linear subspace of $\mathbb{F}_q^3$. Hence, for $a \in L(\mathbb{G}_q)$, we define $\varphi(a) \in R(\mathbb{G}_q)$ as the projective line corresponding to the 2-dimensional linear subspace orthogonal to a. Clearly, φ is bijective. It is also symmetric, since $a \in \varphi(b) \Leftrightarrow$ (a and b are orthogonal 1-dimensional linear subspaces) $\Leftrightarrow b \in \varphi(a)$. Note also that $(a, \varphi(a)) \in E(\mathbb{G}_q)$ if and only if a is self-orthogonal.

Let $\mathbb{G}_{q|\phi}$ be the bijective, symmetric reduction of $\mathbb{G}_q$ induced by φ. We will not use the explicit definition of φ, but only the fact it is bijective and symmetric. Note that $\mathbb{G}_{q|\phi}$ is a graph with $q^2 + q + 1$ vertices, and vertex degree equal to either q (vertices corresponding to self-orthogonal linear subspaces) or $q + 1$ (other vertices). The diameter of $\mathbb{G}_{q|\phi}$ is equal to 2, and for any two non-adjacent vertices $a, b \in V(\mathbb{G}_{q|\phi})$, there is a unique path of length 2 connecting them.

▶ **Theorem 21.** *Let k be such that $5k^2 - k - 10 \leqslant 2q$. Then $\mathbb{G}_{q|\phi}$ is k-vertex-minor universal.*

The proof is done by showing that $\mathbb{G}_{q|\phi}$ satisfies the property (VMU°) from Lemma 16, and can be found in the extended version of this paper [3]. Here, we briefly discuss the implications of the two constructions from Theorem 19 and Theorem 21. We denote by $\lceil x \rceil_{\mathrm{p}}$ the smallest prime power greater than or equal to a real number $x > 1$. For a given $k > 1$, let $q_2 := \left\lceil \frac{7}{4}k^2 - 4 \right\rceil_{\mathrm{p}}$ and $q_1 := \left\lceil \frac{5}{2}k^2 - \frac{1}{2}k - 5 \right\rceil_{\mathrm{p}}$ given by the inequalities in Theorem 19 (bipartite graph) and Theorem 21 (reduced graph), respectively. It follows that $\mathbb{G}_{q_2}$ is a k-vertex-minor universal of order $n_2 = 2(q_2^2 + q_2 + 1) \sim \frac{49}{8}k^4$, while $\mathbb{G}_{q_1|\phi}$ is a k-vertex-minor universal of order $n_1 = q_1^2 + q_1 + 1 \sim \frac{25}{4}k^4$ (where $\sim$ indicates asymptotic equivalence, as k goes to infinity). Thus, asymptotically, the bipartite graph construction yields k-vertex-minor universal graphs of slightly lower order than the reduced graph construction. Another interesting property of the bipartite graph is that the corresponding graph state $|\mathbb{G}_{q_2}\rangle$ is equivalent, up to local Clifford unitaries, to a Calderbank-Shor-Steane (CSS) state [4, Section IV]. However, to construct a desired graph on k-vertices of the bipartite-graph $\mathbb{G}_{q_2}$, we need to follow Lemma 15, thus to construct the sets $C_1, C_2, S_1, S_2, \Omega$ therein, which is done by following the steps highlighted in bold in the proof of Theorem 19. Note that this directly translates into a LOCC protocol to induce a desired graph state on k qubits of the state $|\mathbb{G}_{q_2}\rangle$, using Proposition 5. For the reduced graph the corresponding protocol is simpler, as we only have to construct the sets C, S from Lemma 16, which is again done by following the steps highlighted in bold in the proof of Theorem 21.

5 Conclusion

We showed the existence of k-vertex-minor universal graphs of order quadratic in k, which attain the optimum. This implies the existence of k-vertex-minor universal and thus k-pairable graph states with a quadratic number of qubits. Then, our study of the incidence graph of a finite projective plane exhibited two families of k-vertex-minor universal graphs of linear order in k^4. These two families being, to our knowledge, the first k-stabilizer universal quantum states, and so k-pairable quantum states, that can be constructed on a polynomial number of qubits in k.

This leaves open some questions for future work.

- The logical next step is the explicit, deterministic construction of an infinite family of k-vertex-minor universal graphs whose order is cubic, or even quadratic in k, asymptotically matching the order of the k-vertex-minor universal graphs which can be constructed in a probabilistic, non-deterministic way (although with arbitrarily high probability).
- Our probabilistic construction for k-vertex-minor universal graphs is asymptotically optimal. The graph states corresponding to $2k$-vertex-minor universal graphs are also k-pairable: however the only known lower bound on the size of k-pairable states (where one party holds only one qubit) is quasi-linear [2]. Does there exist k-pairable states with a quasi-linear number of qubits?
- Even though $2k$-stabilizer universality is a stronger requirement than k-pairability, it is not clear whether there exist k-pairable states which are not $2k$-stabilizer universal. A similar question can be asked for graphs: it is not clear whether there exist k-pairable graphs on more than 2 vertices which are not $2k$-vertex-minor universal.
- Bravyi et al. presented a construction of k-pairable states with an asymptotically optimal number of parties, in the case where each party holds at least 10 qubits [2]. How does k-stabilizer universality evolve when considering quantum communication networks where each party holds more than one qubit? Note that the construction of Bravyi et al. where each party holds at least 10 qubits does not translate well for k-stabilizer universality.

References

1 Sara Bartolucci, Patrick Birchall, Hector Bombin, Hugo Cable, Chris Dawson, Mercedes Gimeno-Segovia, Eric Johnston, Konrad Kieling, Naomi Nickerson, Mihir Pant, et al. Fusion-based quantum computation. *Nature Communications*, 14(1):912, 2023. doi:10.1038/s41467-023-36493-1.

2 Sergey Bravyi, Yash Sharma, Mario Szegedy, and Ronald de Wolf. Generating k EPR-pairs from an n-party resource state. *Quantum Information Processing*, 2023. arXiv:2211.06497.

3 Maxime Cautrès, Nathan Claudet, Mehdi Mhalla, Simon Perdrix, Valentin Savin, and Stéphan Thomassé. Vertex-minor universal graphs for generating entangled quantum subsystems, 2024. arXiv:2402.06260.

4 Kai Chen and Hoi-Kwong Lo. Multi-partite quantum cryptographic protocols with noisy GHZ states. *Quantum Information and Computation*, 7(8), November 2007. doi:10.26421/QIC7.8-1.

5 Matthias Christandl, Vladimir Lysikov, Vincent Steffan, Albert H Werner, and Freek Witteveen. The resource theory of tensor networks, 2023. arXiv:2307.07394.

6 Nathan Claudet, Mehdi Mhalla, and Simon Perdrix. Small k-pairable states, 2023. arXiv:2309.09956.

7 Patricia Contreras-Tejada, Carlos Palazuelos, and Julio I. de Vicente. Asymptotic survival of genuine multipartite entanglement in noisy quantum networks depends on the topology. *Physical Review Letters*, 128(22), 2022. doi:10.1103/physrevlett.128.220501.

8 Axel Dahlberg, Jonas Helsen, and Stephanie Wehner. How to transform graph states using single-qubit operations: computational complexity and algorithms. *Quantum Science and Technology*, 5(4):045016, September 2020. doi:10.1088/2058-9565/aba763.

9 Axel Dahlberg, Jonas Helsen, and Stephanie Wehner. Transforming graph states to Bell-pairs is NP-Complete. *Quantum*, 4:348, October 2020. doi:10.22331/q-2020-10-22-348.

10 Axel Dahlberg and Stephanie Wehner. Transforming graph states using single-qubit operations. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 376(2123):20170325, 2018. doi:10.1098/rsta.2017.0325.

11 Maarten Van den Nest, Jeroen Dehaene, and Bart De Moor. Graphical description of the action of local clifford transformations on graph states. *Physical Review A*, 69(2), February 2004. doi:10.1103/physreva.69.022316.

12 Gang Du, Tao Shang, and Jian-wei Liu. Quantum coordinated multi-point communication based on entanglement swapping. *Quantum Information Processing*, 16, March 2017. doi:10.1007/s11128-017-1558-2.

13 Alex Fischer and Don Towsley. Distributing graph states across quantum networks. In *IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 324–333, 2021. doi:10.1109/QCE52317.2021.00049.

14 Sobhan Ghanbari, Jie Lin, Benjamin MacLellan, Luc Robichaud, Piotr Roztocki, and Hoi-Kwong Lo. Optimization of deterministic photonic graph state generation via local operations, 2024. arXiv:2401.00635.

15 Daniel Gottesman. The heisenberg representation of quantum computers, 1998. arXiv:quant-ph/9807006.

16 Frederik Hahn, Anna Pappa, and Jens Eisert. Quantum network routing and local complementation. *npj Quantum Information*, 5(1):1–7, 2019. doi:10.1038/s41534-019-0191-6.

17 Marc Hein, Wolfgang Dür, Jens Eisert, Robert Raussendorf, Maarten Van den Nest, and Hans J. Briegel. Entanglement in graph states and its applications, 2006. arXiv:quant-ph/0602096.

18 Jessica Illiano, Michele Viscardi, Seid Koudia, Marcello Caleffi, and Angela Sara Cacciapuoti. Quantum internet: from medium access control to entanglement access control, 2022. arXiv:2205.11923.

19 Jérôme Javelle, Mehdi Mhalla, and Simon Perdrix. New protocols and lower bounds for quantum secret sharing with graph states. In *Conference on Quantum Computation, Communication, and Cryptography*, pages 1–12. Springer, 2012. arXiv:1109.1487.

20 Donggyu Kim and Sang-il Oum. Vertex-minors of graphs: A survey, October 2023. URL: https://dimag.ibs.re.kr/home/sangil/wp-content/uploads/sites/2/2023/10/2023vertexminors-survey-revised.pdf.

21 Seok-Hyung Lee and Hyunseok Jeong. Graph-theoretical optimization of fusion-based graph state generation. *Quantum*, 7:1212, December 2023. doi:10.22331/q-2023-12-20-1212.

22 Chao-Yang Lu, Xiao-Qi Zhou, Otfried Gühne, Wei-Bo Gao, Jin Zhang, Zhen-Sheng Yuan, Alexander Goebel, Tao Yang, and Jian-Wei Pan. Experimental entanglement of six photons in graph states. *Nature physics*, 3(2):91–95, 2007.

23 Damian Markham and Barry C. Sanders. Graph states for quantum secret sharing. *Physical Review A*, 78:042309, 2008. doi:10.1103/PhysRevA.78.042309.

24 Clément Meignant, Damian Markham, and Frédéric Grosshans. Distributing graph states over arbitrary quantum networks. *Physical Review A*, 100:052333, November 2019. doi:10.1103/PhysRevA.100.052333.

25 Jorge Miguel-Ramiro, Alexander Pirker, and Wolfgang Dür. Optimized quantum networks. *Quantum*, 7:919, February 2023. doi:10.22331/q-2023-02-09-919.

26 Mihir Pant, Hari Krovi, Don Towsley, Leandros Tassiulas, Liang Jiang, Prithwish Basu, Dirk Englund, and Saikat Guha. Routing entanglement in the quantum internet. *npj Quantum Information*, 5(1):1–9, 2019. doi:10.1038/s41534-019-0139-x.

27 Eddie Schoute, Laura Mancinska, Tanvirul Islam, Iordanis Kerenidis, and Stephanie Wehner. Shortcuts to quantum network routing, 2016. arXiv:1610.05238.

28 Péter Vrana and Matthias Christandl. Entanglement distillation from Greenberger–Horne–Zeilinger shares. *Communications in Mathematical Physics*, 352:621–627, 2017. arXiv:1603.03964.

A Some data on the size of the existence constraints

By Lemma 11, given some $k \in \mathbb{N} \setminus \{0\}$, there exists a k-vertex-minor universal bipartite graph G with $|L(G)| \geqslant k$, $|R(G)| \geqslant 4\binom{k}{2} + 5$ if

$$\left(\frac{k}{2^{|L(G)|-k+1}} + e^{-\frac{\left(\frac{|R(G)|}{4} - \binom{k}{2}+1\right)^2}{\left(\frac{7(|R(G)|-k)}{4} - \binom{k}{2}+1\right)}} \right) \times \binom{|L(G)| + |R(G)|}{k} < 1$$

In Table 1 we provide values for which there exists a k-vertex-minor universal bipartite graph of this order, for some particular values of k. In Table 2 we provide values for which a randomly generated bipartite graph is k-vertex-minor universal with at least 99% probability. Experimentally, adding a small, constant number of vertices to the randomly generated bipartite graph, greatly increases the probability of it to be k-vertex-minor universal.

Table 1 Parameters for which some k-vertex-minor universal bipartite graph exists.

k	3	4	5	6	7	8	9	10	11	12	13	14	15
|V(G)|	36	57	83	113	147	184	226	272	322	377	434	497	563
$\|L(G)\|$	18	24	32	40	48	55	63	72	80	90	97	107	115
$\|R(G)\|$	18	33	51	73	99	129	163	200	242	287	337	390	448

k	20	25	30	35	40	50	60	70	80	90	100
|V(G)|	955	1448	2041	2736	3531	5424	7718	10414	13512	17012	20912
$\|L(G)\|$	161	208	256	306	357	461	568	677	788	902	1016
$\|R(G)\|$	794	1240	1785	2430	3174	4963	7150	9737	12724	16110	19896

Table 2 Parameters for which a randomly generated bipartite graph is k-vertex-minor universal with at least 99% probability.

k	3	4	5	6	7	8	9	10	11	12	13	14	15
|V(G)|	47	68	93	123	156	194	235	281	331	385	443	505	571
$\|L(G)\|$	25	32	39	47	55	63	71	79	88	96	105	113	122
$\|R(G)\|$	22	36	54	76	101	131	164	202	243	289	338	392	449

k	20	25	30	35	40	50	60	70	80	90	100
|V(G)|	962	1456	2049	2743	3539	5431	7726	10422	13519	17019	20920
$\|L(G)\|$	167	215	263	313	364	468	575	684	795	908	1023
$\|R(G)\|$	795	1241	1786	2430	3175	4963	7151	9738	12724	16111	19897

Fast Approximate Counting of Cycles

Keren Censor-Hillel
Department of Computer Science, Technion, Haifa, Israel

Tomer Even
Department of Computer Science, Technion, Haifa, Israel

Virginia Vassilevska Williams
Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

We consider the problem of approximate counting of triangles and longer fixed length cycles in directed graphs. For triangles, Tětek [ICALP'22] gave an algorithm that returns a $(1 \pm \varepsilon)$-approximation in $\tilde{O}(n^\omega/t^{\omega-2})$ time, where t is the unknown number of triangles in the given n node graph and $\omega < 2.372$ is the matrix multiplication exponent. We obtain an improved algorithm whose running time is, within polylogarithmic factors the same as that for multiplying an $n \times n/t$ matrix by an $n/t \times n$ matrix. We then extend our framework to obtain the first nontrivial $(1 \pm \varepsilon)$-approximation algorithms for the number of h-cycles in a graph, for any constant $h \geq 3$. Our running time is

$$\tilde{O}\left(\mathsf{MM}\left(n, n/t^{1/(h-2)}, n\right)\right), \text{the time to multiply } n \times \frac{n}{t^{1/(h-2)}} \text{ by } \frac{n}{t^{1/(h-2)}} \times n \text{ matrices.}$$

Finally, we show that under popular fine-grained hypotheses, this running time is optimal.

2012 ACM Subject Classification Mathematics of computing → Approximation algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases Approximate triangle counting, Approximate cycle counting Fast matrix multiplication, Fast rectangular matrix multiplication

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.37

Category Track A: Algorithms, Complexity and Games

Funding *Keren Censor-Hillel*: The research is supported in part by the Israel Science Foundation (grant 529/23).
Virginia Vassilevska Williams: Supported by NSF Grant CCF-2330048, BSF Grant 2020356, and a Simons Investigator Award.

Acknowledgements We would like to thank the anonymous reviewers for their invaluable feedback and for identifying a technical issue in a previous version of our paper.

1 Introduction

Detecting small subgraph patterns inside a large graph is a fundamental computational task with many applications. Research in this domain has flourished, leading to fast algorithms for many tractable versions of the subgraph isomorphism problem: given a fixed (constant size) graph H, detect whether a large graph G contains H as a subgraph, list all copies of H in G, count the copies (exactly or approximately) and more.

The topic of this paper is the fast estimation of the number of copies of a pattern H in a graph G. One of the most studied patterns H is the *triangle* whose detection, listing and approximate counting has become a prime testing ground for ideas in classic graph algorithms [25, 3, 30, 8, 31], sublinear and distributed algorithms [24, 23, 20, 22, 21, 12, 15, 26, 19, 33, 11, 13, 14], streaming [10, 6, 5, 27, 28], parallel [7, 29, 32] algorithms and more. This is largely because triangles are arguably the simplest subgraph patterns and moreover, often algorithms for the triangle version of the (detection, counting or listing) problem formally lead to algorithms for other patterns as well (see Nešetril and Poljak [30]).


Detecting and finding a triangle, and counting the number of triangles in an n-vertex graph can all be reduced to fast matrix multiplication [25], and the fastest algorithm for these versions has running time $O(n^\omega)$, where ω is the exponent of square matrix multiplication, currently $\omega \leq 2.371552$ [35]. It is also believed that even detecting a triangle requires $n^{\omega-o(1)}$ time, due to known fine-grained reductions that show that Boolean matrix multiplication and triangle detection are equivalent, at least for combinatorial algorithms [34].

Obtaining an approximate count $\hat{t}$ to the number of triangles t in an n-node graph such that $(1-\varepsilon)t \leq \hat{t} \leq (1+\varepsilon)t$ for an arbitrarily small constant $\varepsilon > 0$ can be used to detect whether a graph has a triangle, as the algorithm would be able to distinguish between $t = 0$ and $t = 1$. Thus, it is plausible that when the number of triangles is $O(1)$, $n^{\omega-o(1)}$ time is needed to obtain an approximate triangle count.

When the number of triangles t in G is large, however, a simpler sampling approach can obtain a good estimate of t: repeatedly sample a triple of vertices and check whether they form a triangle; in expectation $O(n^3/t)$ samples are sufficient to get a constant factor approximation.

The best known algorithm for approximately counting triangles is by Tětek [33], with running time $\tilde{O}(n^\omega/t^{\omega-2})$. When t becomes constant, the running time becomes $\tilde{O}(n^\omega)$, which is believed to be optimal, as we mentioned earlier. When t becomes $\Theta(n)$, the running time is the same as the naïve sampling algorithm, $\tilde{O}(n^2)$. This quadratic running time is provably necessary even for randomized algorithms (see [21]). Nevertheless, it is unclear whether $\tilde{O}(n^\omega/t^{\omega-2})$ time is needed for all values of t between $O(1)$ and $\Omega(n)$.

Is there a faster algorithm for approximate triangle counting when the triangle count is in $[\Omega(1), O(n)]$?

As triangle counting is an important special case of fixed subgraph isomorphism counting, a natural question is, what is the fastest algorithm for approximately counting arbitrary subgraphs H?

Dell, Lapinskas and Meeks [18] provide a general reduction from approximate H counting to detecting a "colorful" H in an n-node, m-edge graph, so that a $T(n,m)$ time detection algorithm can be converted into an $\tilde{O}(\varepsilon^{-2}T(n,m))$ time $(1\pm\varepsilon)$-approximation algorithm. For many patterns [1] such as triangles and k-cliques or directed h-cycles, the colorful and normal versions of the detection problems are equivalent (e.g. via color-coding [2] and layering). While the detection running time is provably necessary to approximately count when the number of copies of the pattern is constant, similarly to the case of triangles, when the number of copies t is large, faster sampling algorithms are possible. Unfortunately, the reduction of [18] doesn't seem easy to extend to provide runtime savings that grow with t. Thus, we ask:

What is the best approximate counting algorithm for subgraph patterns H with running time depending on the number t of copies of H?

As triangles are also cycles, one special case of the above question is when H is a cycle on h vertices.

[1] This equivalence is not true in general: detecting cycles of fixed even length is believed to be computationally easier than detecting colorful even cycles which are known to be equivalent to the directed version of the problem.

1.1 Our Contribution

The main result of our paper is a new algorithm for approximating the number of h-cycles in a given *directed* graph, for any constant $h \geq 3$, together with a conditional lower bound from a fine-grained hypothesis, showing that the running time of our algorithm is likely tight.

Our main theorem is:

▶ **Theorem 1** (Approximating the Number of h-Cycles). *Let G be a given graph with n vertices and let $h \geq 3$ be a fixed integer. There is a randomized algorithm that outputs an approximation $\hat{t}$ for the number t of h-cycles in G such that $\Pr\left[(1-\varepsilon)t \leq \hat{t} \leq (1+\varepsilon)t\right] \geq 1 - 1/n^2$, for any constant $\varepsilon > 0$. The running time is bounded by $\tilde{\mathcal{O}}\left(\mathsf{MM}\left(n, n/t^{1/(h-2)}, n\right)\right)$, the fastest running time to multiply an $n \times n/t^{1/(h-2)}$ matrix by an $n/t^{1/(h-2)} \times n$ matrix.*

As long as $\omega > 2$, the running time in the theorem is always upper-bounded by $\tilde{\mathcal{O}}\left(n^\omega / t^{\frac{\omega-2}{(h-2)(1-\alpha)}} + n^2\right)$, where $\omega \leq 2.371552$ is the square matrix multiplication exponent mentioned earlier and $\alpha \geq 0.321334$ [35] is the largest real number such that one can multiply $n \times n^\alpha$ by $n^\alpha \times n$ matrices in $n^{2+o(1)}$ time.[2] It is easy to see that for any value $\omega > 2$, our $\tilde{\mathcal{O}}\left(n^\omega / t^{\frac{\omega-2}{1-\alpha}}\right)$ time for *triangles* is faster than the previous state of the art $\tilde{\mathcal{O}}\left(n^\omega / t^{\omega-2}\right)$ for approximate counting of triangles [33] for all t between $\Omega(1)$ and $O(n)$, answering our first question in the introduction. Figure 1 plots our two running times for approximate triangle counting together with Tětek's algorithm, naive sampling and the $O(n^\omega)$ time exact counting algorithm.

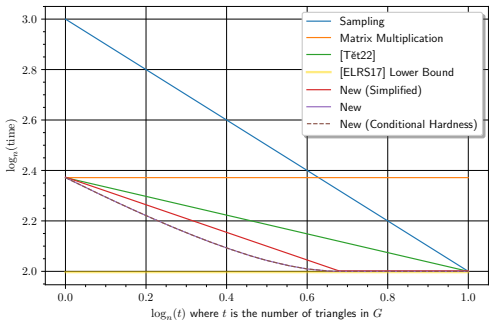


Figure 1 A comparison between our new running times for approximate triangle counting with prior work, together with the lower bounds, both conditional and unconditional.

Method	Runtime
Sampling	n^3/t
Matrix Multiplication	n^ω
[33]	$n^\omega/t^{\omega-2}$
[21] Lower Bound	n^2
New (Simplified)	$n^\omega/t^{(\omega-2)/(1-\alpha)}$
New	$\mathsf{MM}(n,n,n/t)$
New (Conditional Hardness)	$\mathsf{MM}(n,n,n/t)$

Figure 2 Comparative Runtime Analysis.

We obtain our algorithm via a simplification and generalization of Tětek's approach that allows us to both obtain an improved running time for triangles, but also to get faster algorithms for longer cycles. The approach can also extend to other patterns; we leave this as future work.

Our algorithm for longer cycles is arguably the first non-trivial algorithm for the problem with a negative dependence on the number of cycles t. To our knowledge, prior to our work the only approximate counting algorithms for h-cycles for $h > 3$ in directed graphs (or in

[2] We use $\mathsf{MM}(a,b,c)$ to denote the time complexity of multiplying two matrices with dimensions $a \times b$ and $b \times c$.

undirected graphs when h is odd[3]) were to either use naive random sampling resulting in an $O(n^4/t)$ time or to approximate the answer in the best h-cycle detection time, $O(n^\omega)$ (e.g. via [18]), a running time that does not depend on t.

We complement our algorithms for approximately counting h-cycles with a *tight* conditional lower bound under a popular fine-grained hypothesis. The k-Clique Hypothesis of fine-grained complexity (e.g. [1, 4, 9, 16]) postulates that the current fastest algorithms for detecting a k-clique in a graph (for constant $k \geq 3$) are optimal, up to $n^{o(1)}$ factors. We formulate a natural hypothesis about the complexity of *triangle detection* in unbalanced tripartite graphs that is motivated by and in part implied by the k-Clique Hypothesis. Then we show, under that hypothesis:

▶ **Theorem 2.** *Under fine-grained hypotheses, in the word-RAM model with $O(\log n)$ bit words, for any constant integer $h \geq 3$, any randomized algorithm that, when given an n node directed graph G, can distinguish between G being h-cycle-free and containing $\geq t$ h-cycles needs $\mathsf{MM}\left(n, n/t^{1/(h-2)}, n\right)^{1-o(1)}$ time. The same result holds for undirected graphs as well whenever h is odd.*

As any algorithm that can approximate the number t of h-cycles multiplicatively, can distinguish between 0 and t h-cycles, we get that our algorithm running times are essentially tight. We present our lower bound for triangles in Figure 1 as a dotted line. Together with the lower bound by [21], our lower bound shows that our algorithm is (conditionally) optimal for all values of t. Due to space considerations, the lower bound proof is deferred to the full version.

Similarly to Tětek's algorithm, our algorithms for approximate h-cycle counting can be used to obtain improved h-cycle counting algorithms for *sparse* graphs, where the running time is measured in terms of the number of edges m. In particular, for triangles, one can simply substitute our new algorithm in terms of n in Tětek's argument [33] to obtain an approximate counting algorithm that runs in time $\tilde{\mathcal{O}}\left(m^{2\omega/(\omega+1)}/t^{\frac{2(\omega-1)}{\omega+1}+\frac{\alpha(\omega-2)}{(1-\alpha)(\omega+1)}}\right)$. This running time is always faster than Tětek's $\tilde{\mathcal{O}}\left(m^{2\omega/(\omega+1)}/t^{\frac{2(\omega-1)}{\omega+1}}\right)$ for any $\omega > 2$. One can similarly adapt the algorithms of Yuster and Zwick [37] and their analysis in [17] to obtain approximate counting algorithms for longer cycles. We leave this to future work.

1.2 Technical Overview

To frame our technical contribution, we first briefly overview the approach of [33]. The latter gives a randomized approximate counting algorithm for triangles, in time $\tilde{\mathcal{O}}\left(n^\omega/t^{\omega-2}\right)$. In a nutshell, the algorithm finds a subset of vertices S that contains all Λ-heavy vertices and no $\Lambda/\mathrm{polylog}\,(n)$-light vertices – a vertex is called Λ-heavy if it participates in at least Λ triangles, and otherwise it is called Λ-light. Then, the algorithm approximately counts the number of Λ-heavy triangles, which are triangles with at least one heavy vertex. The algorithm then continues by sampling subsets of vertices from the set $V - S$, where each vertex is kept independently uniformly at random with some probability, and processing the sampled graphs by recursion.

[3] The detection problem for even h-cycles in undirected graphs is known to be much easier than that for odd cycles, and for directed graphs, as for every even constant integer h, an $O(n^2)$ time algorithm was developed by Yuster and Zwick [36]. Meanwhile, directed h-cycles and undirected odd h-cycles are believed to require $n^{\omega-o(1)}$ time to detect.

Our technical contribution consists of three parts: (I) We simplify the above recursive approach, (II) we improve upon the component that finds heavy vertices, and (III) we improve upon the component that counts the number of cycles that contain heavy vertices. A compelling aspect of our technique is that it applies to any constant-length cycle. In what follows, we overview each of these aspects.

I. The Recursive Template. In [33], each recursive invocation triggers seven further recursive calls and takes the median of their return values. As the depth of the recursion increases, the algorithm needs to use a precision parameter that becomes exponentially tighter.

In contrast, our algorithm initiates only a single recursive call. This allows us to avoid having to compute the median of several subcalls, which makes it easier to apply standard amplification tools. In particular, it allows us to *fix the precision parameter*.

In addition, by reducing the recursive tree to a "path", we *simplify the analysis* of the running time.

A prime feature of our simplification of the recursion is that it allows us to present it as a *template* for approximate subgraph counting for any fixed subgraph $\mathtt{H}$, provided one designs the two black boxes (one for finding a superset S of the Λ-heavy vertices with no $\Lambda/\text{polylog}(n)$-light vertices, and another for approximately counting the number of copies of $\mathtt{H}$ that intersect the set S).

For triangles, our improvement comes from simplifying the recursion and implementing the black box that finds heavy vertices faster, using rectangular matrix multiplication. Crucially, our implementations of these black boxes are general, in the sense that they apply to constant length h-cycle. Specifically, we find the set S in time $\tilde{\mathcal{O}}\left(\mathsf{MM}\left(n, n/\Lambda^{1/(h-2)}, n\right)\right)$. For such S, we find an $(1+\epsilon)$ approximation for the number of copies of $\mathtt{H}$ that intersect S in time $\tilde{\mathcal{O}}\left(n^2/\varepsilon^3\right)$, which is independent of Λ and of the cardinality of S.

II. Finding the Heavy Vertices. The algorithm in [33] finds Λ-heavy vertices by sampling a subset of vertices uniformly at random, and using matrix multiplication to detect triangles inside the induced sampled subgraph. This takes $\tilde{\mathcal{O}}\left(n^\omega/\Lambda^{\omega-2}\right)$ time, by $\tilde{\mathcal{O}}\left(\Lambda^2\right)$ repetitions of multiplying $n/\Lambda \times n/\Lambda$ matrices.

At the heart of our approach for finding the heavy vertices lies non-uniform sampling, and computing the product of rectangular matrices rather than square ones. We obtain a running time of $\tilde{\mathcal{O}}\left(\mathsf{MM}\left(n, n/\Lambda^{1/(h-2)}, n\right)\right) = \tilde{\mathcal{O}}\left(n^\omega/\Lambda^{\gamma_h}\right)$ for h-cycles, where $\gamma_h \triangleq \frac{\omega-2}{(1-\alpha)(h-2)}$. This comes from multiplying an $n \times n/\Lambda^{1/(h-2)}$ matrix by an $n/\Lambda^{1/(h-2)} \times n$ matrix. In [35] it was shown that $\alpha \geq 0.321334$, and therefore $\frac{\omega-2}{1-\alpha} \geq 1.47(\omega-2)$, which establishes that our algorithm is never slower, and is faster (if $\omega > 2$), where the gap increases with Λ (for sufficiently large Λ, the folklore naïve sampling algorithm is superior).

Our starting point for finding the heavy vertices is the *color-coding* technique of [2], which is widely employed for detecting h-cycles for $h = \mathcal{O}(1)$. This technique colors the vertices using h colors uniformly independently at random and looks for *colorful* h-cycles, which are h-cycles with exactly one vertex of each color. This restriction allows for faster detection but suffers some probability of missing h-cycles that are colored out of order, which can be overcome with sufficiently large probability by repeating this process.

To find colorful h-cycles, we utilize matrix multiplication. However, we do so in a refined manner. Rather than considering all vertices, we sample a subset of vertices from each color class in a nonuniform manner. To illustrate this, consider the task of finding Λ-heavy vertices w.r.t. triangles. We assign a random color to each vertex, and denote the three color classes by V_1, V_2, V_3. We focus on identifying the Λ-heavy vertices within V_1. Fix some $i \in [\log \Lambda]$.

We sample each vertex from V_2 with probability $2^i/\Lambda$, and we sample each vertex from V_3 with probability $1/2^i$. Let H_i denote the induced graph obtained by all vertices from V_1 and the sampled vertices from V_2 and V_3, where we also direct edges from V_j to $V_{j+1 \mod 3}$ and discard monochromatic edges. We show that for every Λ-heavy vertex v, there exists an index $i \in [\log \Lambda]$ such that v is in a triangle in H_i with some probability at least p_{heavy}, where $1/p_{\text{heavy}} = \tilde{\mathcal{O}}(1)$. On the other hand, for $\Lambda/\text{polylog}(n)$-light vertex u, we show that for every $i \in [\log \Lambda]$, the vertex v is in a triangle in H_i with probability at most $p_{\text{heavy}}/2$. Therefore, we can distinguish between these cases. Checking whether v is in a triangle in H_i can be done in $\tilde{\mathcal{O}}(\mathsf{MM}(n, n, n/\Lambda))$ time. Using amplification, we approximate the probability that v is in a triangle in H_i for every $v \in V_1$, and thus distinguish heavy vertices from lighter ones.

We generalize our approach for h-cycles by coloring the vertices with h colors, and directing edges and discarding monochromatic edges, as for triangles. We also discard edges between non-consecutive color classes. To find Λ-heavy vertices in the first color class, we sample in a nonuniform manner a subset of vertices from the j-th color class for $2 \leq j \leq h$, where the product of the sampling probabilities of the color classes should be at most $1/\Lambda$, as for triangles. Let H denote the obtained random induced subgraph. The running time of computing the exact number of h-cycles each vertex in H participates in, which is dominated by the size of the smallest color class in H, becomes $\tilde{\mathcal{O}}\left(\mathsf{MM}\left(n, n, n/\Lambda^{1/(h-2)}\right)\right)$. To see why, consider an h-partite graph G with n vertices in each part. Suppose G has a vertex $v \in V_1$ with a neighbor $u \in V_2$, such that all h-cycles that intersect v, also intersect the edge (u, v). Now, suppose each vertex set V_j, for $3 \leq j \leq h$ has a subset W_j of $\Lambda^{1/(h-2)}$ vertices, such that any h-tuple of the form $(v, u, w_3, w_4, \ldots, w_h)$ is an h-cycle in G, where $w_j \in W_j$ for $3 \leq j \leq h$. This implies that v is Λ-heavy. Note that if we keep each vertex from the j-th color class with a probability of $o(1/\Lambda^{1/(h-2)})$, we are unlikely to sample any vertex from W_j, and therefore we fail to learn that v is Λ-heavy. On the other hand, if we sample vertices from each class with probability $\Omega\left(1/\Lambda^{1/(h-2)}\right)$, the smallest color class is of size $\Omega\left(n/\Lambda^{1/(h-2)}\right)$.

III. Counting the Heavy Copies. Given a graph G and a subset of vertices S, where each vertex in S participates in at least a and at most b copies of h-cycles for $h = \mathcal{O}(1)$, we show how to compute a $(1+\epsilon)$ approximation for the number of h-cycles that intersect the set S, in time $\tilde{\mathcal{O}}\left(n^2 b/\varepsilon a\right)$. In particular, the runtime is independent of size of the set S.

Consider a naïve approach, which approximates the average number of h-cycles that a vertex from S intersects, and let us see why it fails to provide a good approximation for the total number of h-cycles intersecting S. Suppose $h = 3$ and $|S| = 3$ and each vertex $v \in S$ participates in exactly one triangle in G. Based solely on the number of triangles in which a vertex participates, it is impossible to distinguish the case where the set S intersects one triangle in G from the case in which it intersects three triangles in G. The issue here is double counting, as we did not avoid counting the same cycle more than once. For triangles, this obstacle can be avoided by replacing G with a tripartite graph G', where each of the three parts is a copy of V, and for each edge in G there are six edges in G', one for every ordered pair of parts. It is easy to see that every triangle in G corresponds to six triangles in G', and thus an estimate on G' directly gives an estimate on G. That is, we sample a subset F of vertices from S, and for each copy v' of $v \in F$ in the, say, first part of in the tri-partition G', we compute the number of triangles that go through it in G'. This avoids double counting, because each triangle in G' intersects copies of the set F from the first part at most once (as vertices in the same part form an independent set and hence cannot be in

the same triangle). To summarize, restricting to triangles would significantly simplify this part to a single Chernoff inequality for independent random variables. The source of this simplicity is that triangles are cliques. However, larger cycles are not cliques. If we simply repeat every vertex h times to create a new graph G' as in the triangle case, we could create h-cycles in G' that do not correspond to h-cycles in G: every *closed walk* of length h would become an h-cycle. As in [2], we use color-coding to overcome this, and this necessitates a more careful probabilistic analysis .

First, we sample a subset of vertices from S and for each sampled vertex v we approximate the number of h-cycles which go through v (and therefore intersect the set S). The crux of our algorithm is that in order to approximate the above, we approximate the number of h-cycles which go through v and intersect the set S exactly k times, for each $1 \leq k \leq h$. The summation of these approximations yields our final result, and it naturally avoids the pitfall of double-counting.

To approximate the number of h-cycles intersecting the set S exactly k times for some k, we color the graph with $h-1$ colors and color vertex v with the color h. This ensures that any colorful h-cycle intersects v. Then, we choose $k-1$ color classes from the first $h-1$ classes, and retain only vertices of S within those classes. For the remaining color classes, we keep only vertices that are not in S. The color class h is fixed and always contains only v. This promises that each colorful h-cycle with v in this auxiliary graph intersects S exactly k times. The number of ways to choose exactly $k-1$ color classes that keep only vertices from S is $\binom{h-1}{k-1}$. We compute the number of h-cycles in each such auxiliary graph. We prove that the expectation of this number is some fixed constant multiplicative factor off from the number of h-cycles intersecting v and S exactly k times. Finally, we prove that the variance of this random variable is suitably bounded. Therefore, conducting this process $\tilde{\mathcal{O}}\left(b/(a\varepsilon)\right)$ times enables us to obtain an $(1 \pm \varepsilon)$ approximation for its expectation by Chebyshev's inequality. We compute the number of h-cycles in this auxiliary graph using rectangular matrix multiplication. Since the auxiliary graph is h-partite and one part contains only a single vertex, we get a running time of $\mathsf{MM}(n, n, 1) = \tilde{\mathcal{O}}\left(n^2\right)$. Thus, we achieve our claimed running time of $\tilde{\mathcal{O}}\left(n^2 b/\varepsilon a\right)$.

We mention that we invoke this procedure on the set of vertices given by the previous component of finding heavy vertices, which is called upon in every recursion step. A crucial observation that we make is that not only does this set contain all Λ-heavy vertices and no $\Lambda/\mathrm{polylog}(n)$-light vertices, but rather we also know that it does not contain $(2^h\Lambda)$-heavy vertices, because those are handled during previous steps of the recursion. This means that we invoke this procedure for a, b that differ only by $\mathrm{polylog}(n)$ and ε^2 factors, and thus we effectively get a running time of $\tilde{\mathcal{O}}\left(n^2/\varepsilon^3\right)$ for counting h-cycles through Λ-heavy vertices.

Roadmap. Section 2 contains our template for the recursion, and proves its correctness for any graph $\mathtt{H}$ given implementations of two black boxes, one that finds heavy vertices and another that counts the copies of $\mathtt{H}$ that contain heavy vertices. Section 3 proves the running time that our template obtains for h-cycles, given the running times of implementations of the two black boxes. We implement our black boxes for h-cycles in Sections 4 and 5. Missing proofs, as well as our hardness result, appear in the full version.

1.3 Preliminaries

Let G be a graph on n vertices. Let $\mathtt{H}$ be a fixed graph with $h = \mathcal{O}(1)$ vertices. For a subgraph $G' \subseteq G$, and a subset of vertices S, denote by $t_{G'}(S)$ number of copies of $\mathtt{H}$ in G' which intersect S. Denote by $\tau = \tau_{G'}$ the maximal number of copies of $\mathtt{H}$ in G' in which a

vertex participates. We say that a vertex v is Λ-heavy (in G) if $t_G(v) \geq \Lambda$, and otherwise it is Λ-light. We say that a copy C of $\mathtt{H}$ is Λ-heavy if C contains at least one Λ-heavy vertex. Let G be a graph and p some parameter that could depend on G. We denote by $G[p]$ a random induced subgraph of G obtained by keeping each vertex from G independently with probability p. We use $t(1 \pm \varepsilon)$ to denote the closed interval $[t(1-\varepsilon), t(1+\varepsilon)]$. We say that a value $\hat{t} = t(1 \pm \varepsilon)$ if $\hat{t} \in [t(1-\varepsilon), t(1+\varepsilon)]$. We assume that $\varepsilon \in (0, 1/2]$, which might depend on n. If ε is bigger, our algorithm assumes $\varepsilon \leq 1/2$. Finally, all logarithms in this paper are base 2.

▶ **Definition 3** (Fast Matrix Multiplication Definitions). *We denote the time it takes to compute the product of two matrices of dimension $n^a \times n^b$ and $n^b \times n^c$ by either $\mathsf{MM}\left(n^a, n^b, n^c\right)$ or $n^{\omega(a,b,c)}$. We also abuse the notation and write $\omega = \omega(1,1,1)$, and $\omega(k) = \omega(1,k,1)$. Note that for any permutation $\pi : [3] \to [3]$ we have $\omega(x_1, x_2, x_3) = \omega(x_{\pi(1)}, x_{\pi(2)}, x_{\pi(3)})$. In addition to ω, we will also use α to be the largest real number such that n by n^α by n matrix multiplication can be done in $n^{2+o(1)}$ time.*

2 The Recursive Template

Organization. In this section, we present an algorithm for approximating the number of copies of a graph $\mathtt{H}$ in a graph G, denoted by t, which builds upon two black boxes. The first black box, called Find-Heavy, takes a graph H and a parameter Λ as input and computes a superset of the Λ-*heavy* vertices, excluding any $\Lambda/\mathrm{polylog}\,(n)$-light vertices. We denote the computed superset of heaviest vertices as S. The second black box, called Count-Heavy, is used to compute an approximation for the number of *heavy-copies* of $\mathtt{H}$ in G, which is the set of all copies that contain at least one vertex from S. Our algorithm for subgraph approximate counting that uses the specified black boxes consists of two parts: a doubling algorithm called Doubling-Template, and a recursive algorithm called Template, which is the main focus of this section.

The Template Algorithm. The $\mathsf{Template}_{\varepsilon'}(G, \Lambda)$ algorithm takes two parameters: a graph G and a heaviness threshold Λ. The output of the algorithm is a value $\hat{t}$, which, with a probability of at least $2/3$, is within the range $t \pm (t \cdot \varepsilon' + \Lambda \cdot \mathrm{polylog}\,(n)/\varepsilon')$, where ε' is the fixed precision parameter of the algorithm.

We next explain how the recursive Template algorithm works. The algorithm does the following. (1) Find the heaviest vertices using the Find-Heavy black box, and denote this set by V_Λ. (2) Compute an approximation to the number of heavy copies of $\mathtt{H}$, i.e., copies of $\mathtt{H}$ with at least one vertex from V_Λ, using the Count-Heavy black box, and denote the output by $\hat{t}_\Lambda$. (3) Let $H = G[V(G) - V_\Lambda]$, and let $F \leftarrow H[p]$. That is, F is an induced subgraph of H, where each vertex from H joins F independently with probability p. (4) Make a recursive call to $\mathsf{Template}_{\varepsilon'}\left(H, \Lambda \cdot p^{|\mathtt{H}|}\right)$ and let $\hat{t}_H$ denote its output. (5) Return $\hat{t}_\Lambda + \hat{t}_H/p^{|\mathtt{H}|}$. The analysis of the probability that the algorithm produces a good approximation appears in the proof of Lemma 7. The running time of the algorithm depends on the implementation of the black boxes. In the next section, we analyze the running time of the algorithm for the case where $\mathtt{H}$ is a cycle.

The Doubling-Template Algorithm. The Doubling-Template algorithm is a doubling algorithm, which starts with an initial guess for t, denoted by $\mathsf{W}_0 = n^{|\mathtt{H}|}$. This is the maximal number of copies of $\mathtt{H}$ an n vertex graph can contain ($h! \cdot \binom{n}{|\mathtt{H}|} \leq n^{|\mathtt{H}|}$). The algorithm then makes $\tilde{\mathcal{O}}(1)$ calls to $\mathsf{Template}_{\varepsilon'}(G, \Lambda_0)$, where $\Lambda_0 \leftarrow \mathsf{W} \cdot \varepsilon^2/8Q$, where $Q = 8\log^4(n)$, and

computes their median, which we denote by $\hat{t}_0$. If $\hat{t}_0 \geq \mathsf{W}_0$ the doubling algorithm stops and outputs $\hat{t}_0$ as its approximation for t. Otherwise, the guess for the value of t is decreased by a factor of 2. The main point here is that the smaller the value of Λ given to the recursive algorithm is, the better approximation we get, while simultaneously increasing the running time. The guess W is a guess for the highest heaviness threshold the algorithm can start with to output a good approximation, and not an actual guess for the value of t (although both quantities are related).

Formally, the black boxes that we assume are the following.

Find-Heavy(G, Λ)

Input: A graph G, some parameter Λ.
Output: A subset V_Λ of vertices, such that with probability at least $1 - \frac{1}{n^4}$, $\forall v \in V(G)$:
1. If $t_G(v) \geq \Lambda$, then $v \in V_\Lambda$.
2. If $t_G(v) \leq \Lambda/(\log n)^{h^2}$, then $v \notin V_\Lambda$.

Count-Heavy$_{\varepsilon'}(G, V_\Lambda, a, b)$

Input: A graph G, a precision parameter ε', a subset of vertices V_Λ, and two real numbers $0 < a \leq b$, such that $\forall v \in V_\Lambda$ we have $t_G(v) \in [a, b]$.
Output: $\hat{t}_\Lambda$ which satisfies $\Pr\left[\hat{t}_\Lambda = t_G(V_\Lambda)(1 \pm \varepsilon')\right] \geq 1 - \frac{1}{n^4}$.

It should be noted that Count-Heavy cannot be applied to the entire graph, as it might contain a vertex v with $t_G(v) = 0$. Moreover, even if all vertices have $t_G(v) > 0$, employing this black box on the entire graph might result in slower running time. Indeed, in our implementation of the black box, the runtime is contingent on the ratio b/a. Therefore, we only employ this black box with $b/a = \mathcal{O}(1/\varepsilon^2)$.

Algorithm 1 Template$_{\varepsilon'}(G, \Lambda)$.

Input: A graph $G = (V, E)$, a heaviness threshold Λ, and a precision parameter ε'.

1 $V_\Lambda \leftarrow$ Find-Heavy(G, Λ) ; ▷ V_Λ is a superset of Λ-heavy vertices in G with no $\Lambda/(\log n)^{h^2}$-light vertices
2 $a_\Lambda \leftarrow \frac{\Lambda}{(\log n)^{h^2}}$, $b_\Lambda \leftarrow \Lambda \cdot \frac{8Q}{\varepsilon^2}$; ▷ $Q = 8\log^4(n)$
3 $\hat{t}_\Lambda \leftarrow$ Count-Heavy$_{\varepsilon'}(G, V_\Lambda, a_\Lambda, b_\Lambda)$; ▷ $\hat{t}_\Lambda$ is a $(1 \pm \varepsilon')$ approximation for $t_G(V_\Lambda)$ (the number of copies of H intersecting V_Λ)
4 **if** $\Lambda \leq 1$ **then return** $\hat{t}_\Lambda$;
5 $H \leftarrow G[V - V_\Lambda]$;
6 $p \leftarrow 1/2$; ▷ We keep p instead of $1/2$ for readability
7 $F \leftarrow H[p]$; ▷ F is a random subgraph of H
8 $\hat{t}_F \leftarrow$ Template$_{\varepsilon'}\left(F, \Lambda \cdot p^h\right)$;
9 **return** $\hat{t}_\Lambda + \hat{t}_F/p^h$;

The depth of the recursion in Template$_{\varepsilon'}(G, \Lambda)$ is at most $\log_{1/p^h}(\Lambda) + 1$. Since we will only call this algorithm with $\Lambda \leq |V(G)|^h$, we can conclude that the depth of the recursion is at most $\log n + 1$. The guarantees for the template are given in the following lemma.

▶ **Lemma 4** (Guarantees for the Template Algorithm). *For every* $\varepsilon \in (0, 1/2]$ *and every* $\Lambda \geq \tau_G \cdot \frac{\varepsilon^2}{8Q}$, *we have* $\Pr\left[\mathsf{Template}_{\varepsilon'}(G, \Lambda) = t_G(1 \pm \frac{\varepsilon}{2}) \pm \Lambda \cdot \frac{\log^4(n)}{2\varepsilon}\right] \geq \frac{2}{3}$, *where* $\varepsilon' = \frac{\varepsilon}{4 \log n}$.

Algorithm 2 Doubling-Template (G, ε).

Input: A graph $G = (V, E)$ with t_G copies of $\mathtt{H}$ and a precision parameter $\varepsilon \leq 1/2$.
Output: $\hat{t}$, which is a $(1 \pm \varepsilon)$ approximation for t w.h.p.

$\varepsilon' \leftarrow \frac{\varepsilon}{4 \log n}$, $\mathsf{W} \leftarrow n^h$, $Q \leftarrow 8 \log^4(n)$, $\Lambda \leftarrow \mathsf{W} \cdot \varepsilon^2 / Q$
for $i = 0$ *to* $i = h \log n$ **do**
  $\hat{t}_i \leftarrow \mathsf{Median}\left[\mathsf{Template}_{\varepsilon'}\left(G, \Lambda/2^i\right), 400 \log n\right]$; ▷ $\hat{t}_i$ `is the median of` $400 \log n$ `independent executions of` $\mathsf{Template}_{\varepsilon'}\left(G, \Lambda/2^i\right)$.
  if $\hat{t}_i \geq \mathsf{W}/2^i$ **then return** $\hat{t}_i$;
return *Exact deterministic count of* t_G.

▶ **Lemma 5** (Guarantees for the Doubling-Template Algorithm). *Let* G *be a graph with* n *vertices and* t_G *copies of* $\mathtt{H}$. *Fix some* $\varepsilon > 0$ *that may depend on* n. *Let* $\hat{t}$ *denote the output of* Doubling-Template (G, ε) *(specified in Algorithm 2). Then,* $\Pr\left[\hat{t} = t(1 \pm \varepsilon)\right] \geq 1 - 1/n^2$.

The main result that we prove in this section is Lemma 5. We prove it using Lemma 4. First, we use amplification, to show that the event specified in Lemma 4 occurs with high probability, and not only with probability at least 2/3. The rest assumes that this event always occurs. We then use case analysis on $\Lambda/2^i$.

1. For i such that $\Lambda/2^i \geq 4t_G$, we show that $\hat{t}_i < \Lambda/2^i$ w.h.p., meaning the doubling algorithm does not stop for such i w.h.p., and makes another iteration with a refined initial heaviness threshold.
2. For i such that $\Lambda/2^i \leq 4t_G$, we show that $\hat{t}_i = t_G(1 \pm \varepsilon)$ w.h.p.
3. For i such that $\Lambda/2^i \leq t_G/2$, we show that $\hat{t}_i \geq \Lambda/2^i$ w.h.p., which means the algorithm stops as soon as $\Lambda/2^i \leq t_G/2$ w.h.p.

To summarize, the doubling algorithm always stops (by the third property). It does not stop when $\Lambda/2^i \geq 4t_G$ w.h.p. Therefore, when it does stop we have that $\Lambda/2^i \leq 4t_G$, and then it obtains a $(1 \pm \varepsilon)$ approximation for t_G.

To prove Lemma 4, we state and prove a more refined version of the guarantees of the template algorithm. We need the following definition to restate it.

▶ **Definition 6** ($\hat{\mathsf{D}}(\Lambda)$). *We define the depth of the call* $\mathsf{Template}_{\varepsilon'}(F, \Lambda)$ *as* $\hat{\mathsf{D}}(\Lambda) = \max\left\{0, \left\lceil \log_{1/p^h}(\Lambda) \right\rceil\right\}$.

We assume $\Lambda \leq (h!) \cdot \binom{n}{h} \leq n^h$, and therefore that $\hat{\mathsf{D}}(\Lambda) \leq \log n + 1$.

▶ **Lemma 7** (Induction Hypothesis). *Given a graph* G *and* ε, *we set* $p = 1/2$, *and* $\varepsilon' = \frac{\varepsilon}{4 \log n}$. *Then, for any* $\Lambda \geq \tau_G \cdot \frac{\varepsilon^2}{8Q}$, *and any* $\mathsf{K} = o(n)$ *that could depend on* $n, \hat{\mathsf{D}}(\Lambda)$ *and* p, *we have*

$$\Pr\left[\mathsf{Template}_{\varepsilon'}(G, \Lambda) = t_G(1 \pm \varepsilon')^{\hat{\mathsf{D}}(\Lambda)} \pm 2\hat{\mathsf{D}}(\Lambda)\,\mathsf{K} \cdot \Lambda/\varepsilon'\right] \geq 1 - 4h\hat{\mathsf{D}}(\Lambda)/(\mathsf{K}p^h) .$$

Proof Sketch. We prove using induction on the depth of the recursion. We skip the proof of the base case and state the induction hypothesis. The full proof appears in the full version.

Step. We first define some notation and events. We have three graphs $F \subseteq H \subseteq G$, where G is the input graph, H is an induced subgraph of G without the Λ-heavy vertices, and F is a random induced graph of H obtained by keeping each vertex of H independently with probability p. Let $\hat{t}_G$ denote the output of $\mathsf{Template}_{\varepsilon'}(G, \Lambda)$, and let $\hat{t}_F$ denote the output of $\mathsf{Template}_{\varepsilon'}\left(F, \Lambda \cdot p^h\right)$. We use t_Λ to denote $t_G(V_\Lambda)$, and let $\hat{t}_\Lambda$ denote the output of $\mathsf{Count\text{-}Heavy}_{\varepsilon'}(G, V_\Lambda)$. Note that $\hat{\mathsf{D}}(\Lambda) \geq \hat{\mathsf{D}}(\Lambda p^h) + 1$ (unless $\hat{\mathsf{D}}(\Lambda) = 0$, in which case $\hat{\mathsf{D}}(\Lambda) = \hat{\mathsf{D}}(\Lambda p^h) = 0$).

Intuition. We compute two values, $\hat{t}_\Lambda$ and $\hat{t}_F$. We then output $\hat{t} = \hat{t}_\Lambda + \hat{t}_F/p^h$ as an approximation for t_G. By the black box guarantees, we have that $\hat{t}_\Lambda$ is a good approximation for t_Λ. What is left is to show that $\hat{t}_F/p^h$ is a good approximation for t_H. We split this into two parts. First, we show that t_F/p^h is "close" to the value of t_H. Next, we use the induction hypothesis, to show that $\hat{t}_F$ is a good approximation of t_F. We need to show that the "composition" of these approximations is also good. Let $\mathcal{E}_{\text{Find-Heavy}}$ denote the event that all calls made to the $\mathsf{Find\text{-}Heavy}$ black box produce a valid output. That is, the event that V_Λ contains a superset of the Λ-heavy vertices, without any $\Lambda/(\log n)^{h^2}$ n-light vertices. **We prove the correctness of the algorithm under the assumption that $\mathcal{E}_{\text{Find-Heavy}}$ occurs**. We define:

1. $\mathcal{E}_1 \triangleq \{\hat{t}_\Lambda = t_\Lambda(1 \pm \varepsilon')\}$. The heavy copies of $\mathtt{H}$ are approximated correctly.
2. $\mathcal{E}_2 \triangleq \{\hat{t}_F = t_F(1 \pm \varepsilon')^{\hat{\mathsf{D}}(\Lambda p^h)} \pm 2\hat{\mathsf{D}}(\Lambda p^h) \cdot \mathsf{K} \cdot (\Lambda \cdot p^h)/\varepsilon'\}$. This is the event in the induction hypothesis.
3. $\mathcal{E}_3 \triangleq \{t_F/p^h = t_H(1 \pm \varepsilon') \pm \mathsf{K} \cdot \Lambda/\varepsilon'\}$. This is a concentration bound on t_F.
4. $\mathcal{E}_4 \triangleq \{\hat{t}_F/p^h = t_H(1 \pm \varepsilon')^{\hat{\mathsf{D}}(\Lambda)} \pm 2\hat{\mathsf{D}}(\Lambda)\,\mathsf{K} \cdot \Lambda/\varepsilon'\}$. This event contains $\mathcal{E}_2 \cap \mathcal{E}_3$.
5. $\mathcal{E}_5 \triangleq \{\hat{t}_G = t_G(1 \pm \varepsilon')^{\hat{\mathsf{D}}(\Lambda)} \pm 2\hat{\mathsf{D}}(\Lambda)\,\mathsf{K} \cdot \Lambda/\varepsilon'\}$. This is the event specified in Lemma 7.

Lemma 7 requires us to show that $\Pr[\mathcal{E}_5] \geq 1 - \frac{4h\hat{\mathsf{D}}(\Lambda)}{\mathsf{K}p^h}$. We show this by proving that $\mathcal{E}_1 \cap \mathcal{E}_2 \cap \mathcal{E}_3 \subseteq \mathcal{E}_5$, which implies that it is sufficient to prove that $\Pr[\mathcal{E}_{\text{Find-Heavy}} \cap \mathcal{E}_1 \cap \mathcal{E}_2 \cap \mathcal{E}_3] \geq 1 - \frac{4h\hat{\mathsf{D}}(\Lambda)}{\mathsf{K}p^h}$. To show the latter, we show that $\Pr[\mathcal{E}_{\text{Find-Heavy}} \cap \mathcal{E}_1 \cap \mathcal{E}_2] \geq 1 - \frac{4h\hat{\mathsf{D}}(\Lambda p^h)+2}{\mathsf{K}p^h}$ and that $\Pr[\mathcal{E}_3] \geq 1 - \frac{h+1}{\mathsf{K}p^h}$. Summing up the two error probabilities in the above expressions gives the desired result. ◀

3 Application: Approximating the Number of h-Cycles

In this section, we analyze the running time of the recursive and doubling algorithm, when the counted subgraph is an h-cycle for $h = \mathcal{O}(1)$. For this task, we assume the black boxes can be implemented in specific runtime, as stated in the following lemma which is proven in the next sections.

▶ **Lemma 8.** *Let G be a graph with n vertices, let $\mathtt{H}$ be an h-cycle for some $h = \mathcal{O}(1)$, and let ε' be some parameter. Then, each call to $\mathsf{Find\text{-}Heavy}(G, \Lambda)$ can be implemented in time $\tilde{\mathcal{O}}\left(\mathsf{MM}\left(n, n, \frac{n}{\Lambda^{1/(h-2)}}\right)\right)$ and each call to $\mathsf{Count\text{-}Heavy}_{\varepsilon'}(G, V_\Lambda, a_\Lambda, b_\Lambda)$ can be implemented in time $\tilde{\mathcal{O}}\left(\mathsf{MM}(n, n, 1) \cdot \frac{b_\Lambda}{a_\Lambda \cdot \varepsilon}\right) = \tilde{\mathcal{O}}\left(n^2 \cdot \frac{b_\Lambda}{a_\Lambda \cdot \varepsilon}\right)$.*

▶ **Theorem 1** (Approximating the Number of h-Cycles)**.** *Let G be a given graph with n vertices and let $h \geq 3$ be a fixed integer. There is a randomized algorithm that outputs an approximation $\hat{t}$ for the number t of h-cycles in G such that $\Pr\left[(1-\varepsilon)t \leq \hat{t} \leq (1+\varepsilon)t\right] \geq 1 - 1/n^2$, for any constant $\varepsilon > 0$. The running time is bounded by $\tilde{\mathcal{O}}\left(\mathsf{MM}\left(n, n/t^{1/(h-2)}, n\right)\right)$, the fastest running time to multiply an $n \times n/t^{1/(h-2)}$ matrix by an $n/t^{1/(h-2)} \times n$ matrix.*

Proof Sketch. Consider the Doubling-Template (G, ε). Its correctness follows from Lemma 5. We analyze its running time. We prove that given Lemma 8, the running time of the recursive algorithm Template$_{\varepsilon'}$ (G, Λ) is bounded by $\tilde{\mathcal{O}}\left(\mathsf{MM}\left(n, n, \frac{n}{\Lambda^{1/(h-2)}}\right) + \frac{n^2}{\varepsilon^3}\right)$ with probability at least $1 - \exp(-\log^2 n)$. Then, we show that the complexity of the doubling algorithm is bounded by $\tilde{\mathcal{O}}\left(\frac{1}{\varepsilon^3} \cdot \mathsf{MM}\left(n, n, \frac{n}{t^{1/(h-2)}}\right)\right)$ with probability at least $1 - \frac{1}{n^3}$. The full details appear in the full version. Here we give a sketch of why the running time of Template$_{\varepsilon'}$ (G, Λ) is bounded by $\tilde{\mathcal{O}}\left(\mathsf{MM}\left(n, n, \frac{n}{\Lambda^{1/(h-2)}}\right) + \frac{n^2}{\varepsilon^3}\right)$ with probability at least $1 - \exp(-\log^2 n)$.

We unroll the recursion of the algorithm: it makes a single call to Find-Heavy, then a single call to Count-Heavy$_{\varepsilon'}$, and then a recursive call. The algorithm's runtime can be analyzed by bounding $\{\mathsf{Find\text{-}Heavy}(G_k, \Lambda_k)\}_{k=0}^{r}$, and $\{\mathsf{Count\text{-}Heavy}_{\varepsilon'}(G_k, V_{\Lambda_k}, a_{\Lambda_k}, b_{\Lambda_k})\}_{k=0}^{r}$, where r is the recursion depth and G_k denotes the input graph for the k-th call of the recursive algorithm, where $G_0 = G$. Let $\Lambda_k = \Lambda_0 \cdot p^{kh}$ denote the heaviness threshold, and let V_{Λ_k} denote the set of Λ_k-heavy vertices in the k-th iteration. We bound the running time of the k-th call to each of the black boxes. We prove in the full version that the total running time for all calls to the Count-Heavy$_{\varepsilon'}$ black-box is $\tilde{\mathcal{O}}\left(n^2/\varepsilon^3\right)$. Next, we bound the running time of the k-th call to the Find-Heavy black-box. Note that for any k, the call $\mathsf{Find\text{-}Heavy}(G_k, \Lambda_k)$ takes $\tilde{\mathcal{O}}(\mathsf{MM}\left(|V(G_k)|, |V(G_k)|, |V(G_k)|/\Lambda_k^{1/(h-2)}\right))$ by Lemma 8. We use Chernoff's inequality to show that the number of vertices in G_k is $\tilde{\mathcal{O}}\left(\max\left\{1, np^k\right\}\right)$, thus we can replace $|V(G_k)|$ by np^k in the above expression.

The crux is that the running time of the first call to Find-Heavy also applies to subsequent calls. This is because $\mathsf{MM}\left(np^k, np^k, np^k/\left(\Lambda \cdot p^{hk}\right)^{1/(h-2)}\right) \leq \mathsf{MM}\left(n, n, \frac{n}{\Lambda^{1/(h-2)}} \cdot p^{k\cdot(3-h/(h-2))}\right) \leq \mathsf{MM}\left(n, n, n/\Lambda^{1/(h-2)}\right)$, where the first inequality is a simple observation that we prove in the full version, and the second inequality follows since $3 - h/(h-2) = 2\frac{h-3}{h-2}$ is non-negative for $h \geq 3$ and therefore $p^{k\cdot(3-h/(h-2))} \leq 1$. We conclude that the running time of each call in $\{\mathsf{Find\text{-}Heavy}(G_k, \Lambda_k)\}_{k=0}^{r}$ is at most $r \cdot \mathcal{O}\left(\mathsf{MM}\left(n, n, n/\Lambda^{1/(h-2)}\right)\right)$. As $r \leq \log n$, we get that all calls take a total of $\tilde{\mathcal{O}}\left(\mathsf{MM}\left(n, n, n/\Lambda^{1/(h-2)}\right)\right)$ time.

This completes the proof, as all calls to the Find-Heavy black box and the Count-Heavy$_{\varepsilon'}$ black box take at most $\tilde{\mathcal{O}}\left(\mathsf{MM}\left(n, n, n/\Lambda^{1/(h-2)}\right) + n^2/\varepsilon^3\right)$ time in total. ◀

4 Implementing the Black Box Count-Heavy$_\varepsilon$

In this section, we implement Count-Heavy$_\varepsilon$. We prove the second part of Lemma 8, stated next.

▶ **Theorem 9.** *There is an algorithm that implements the* Count-Heavy$_\varepsilon(G, S, a, b)$ *black box, when* H *is an* h*-cycles, for* $h = \mathcal{O}(1)$*, in time* $\tilde{\mathcal{O}}\left(n^2 \cdot \frac{b}{a\cdot\varepsilon}\right)$.

For this entire section, the graph G is fixed, and the set S is a fixed subset of vertices, where for every $v \in S$ we have $t_G(v) \in [a, b]$. We emphasize that a is only a lower bound on $\min_{v\in S} t_G(v)$ and b is only an upper bound on $\max_{v\in S} t_G(v)$. Denote $N_k \triangleq |S|/k$. As explained in the introduction, it is insufficient to sample a few vertices from S, estimate $t_G(v)$ for each one, and apply a concentration bound to compute $t_G(S)$. Formally, this approach fails because $\sum_{v\in S} t_G(v) \neq t_G(S)$ due to possible double counting. We overcome this issue by sampling a small subset of vertices $S' \subseteq S$, and then, for every $v \in S'$ we approximate the number of copies of H which intersect v and exactly i additional vertices from S for $0 \leq i \leq h-1$. This will allow us to estimate the number of multiple countings and therefore get an estimation of $t_G(S)$.

That is, the key ingredient of our approach for approximating $\hat{t}$ as required by $\mathsf{Count\text{-}Heavy}_\varepsilon$ is to approximate the number of cycles that intersect S in exactly k vertices. To this end, we define the following.

▶ **Definition 10.** *Let $\mathcal{C}$ denote the set of copies of $\mathtt{H}$ in G. For $U \subseteq V$, define $\mathcal{C}(U) \triangleq \{C \in \mathcal{C} \mid C \cap U \neq \emptyset\}$. Denote $t \triangleq |\mathcal{C}|$ and $t(U) \triangleq |\mathcal{C}(U)|$. In general, we replace the symbol $\mathcal{C}$ by t, to denote the cardinality of a set. Let $\mathcal{C}^k$ denote the set of copies $C \in \mathcal{C}$ with $|C \cap S| = k$. Let $\mathcal{C}^k(v) \triangleq \mathcal{C}^k \cap \mathcal{C}(v)$. Let $t^k = \left|\mathcal{C}^k\right|$.*

The following lemma shows that we can efficiently approximate t^k.

▶ **Lemma 11** (Algorithm Approx_{t^k})**.** *There exists a randomized algorithm Approx_{t^k} with the following characteristics. The input is a graph G, a set S, a precision parameter δ, a parameter $k \in [h]$, and a tuple (a, b), such that for every $v \in S$ we have $t_G(v) \in [a, b]$. The algorithm produces an output $\hat{t}^k$ which satisfies $\Pr\left[\hat{t}^k = t^k \pm t_G(S) \cdot \delta\right] \geq 1 - \frac{1}{n^4}$. The running time of the algorithm is bounded by $\tilde{\mathcal{O}}\left(n^2 \cdot \frac{b}{a} \cdot \frac{1}{\delta}\right)$.*

Approximating t^k directly leads to Theorem 9 because $\sum_{k\in[h]} t^k = t_G(S)$. A formal proof appears in the full version. To approximate t^k as required by Lemma 11, we find a value whose expectation is t^k and whose variance is at most $\mathcal{O}\left((N_k \cdot b)^2\right)$ (recall that $N_k = |S|/k$). We can do this efficiently, as follows.

▶ **Lemma 12** (Algorithm $\mathsf{Approx}_{\mathbb{E}[t^k]}$)**.** *There is a randomized algorithm $\mathsf{Approx}_{\mathbb{E}[t^k]}$ whose input is G, S, δ and k. Note that unlike Approx_{t^k}, the algorithm $\mathsf{Approx}_{\mathbb{E}[t^k]}$ does not require a and b as part of its input parameters. $\mathsf{Approx}_{\mathbb{E}[t^k]}$ computes a value X such that $\mathbb{E}[X] = t^k$ and $\mathsf{Var}[X] \leq C \cdot (N_k \cdot b)^2$, where C is a constant. The running time of the algorithm is bounded by $\tilde{\mathcal{O}}\left(n^2\right)$.*

The reason that Lemma 12 is helpful is that we can run the algorithm it provides r times and take the median of means of these invocations. A formal proof of Lemma 11 appears in the full version. To get a sample X with $\mathbb{E}[X] = t^k$ and $\mathsf{Var}[X] \leq C \cdot (b \cdot N_k)^2$ as needed by Lemma 12, we find samples Y_v with $\mathbb{E}[Y_v] = t^k(v)$ and $\mathsf{Var}[Y_v] \leq C \cdot b^2$. We can do this efficiently, as follows.

▶ **Lemma 13.** *There is a randomized algorithm $\mathsf{Approx}_{\mathbb{E}[\mathsf{vertex}\text{-}t^k]}$ whose input is G, S, δ, k and a vertex $v \in S$. Unlike $\mathsf{Approx}_{\mathbb{E}[t^k]}$, $\mathsf{Approx}_{\mathbb{E}[\mathsf{vertex}\text{-}t^k]}$ additionally takes a vertex $v \in S$ as input. $\mathsf{Approx}_{\mathbb{E}[\mathsf{vertex}\text{-}t^k]}$ computes a value Y_v such that $\mathbb{E}[Y_v] = t^k(v)$ and $\mathsf{Var}[Y_v] \leq C \cdot b^2$, where C is a constant. The running time of the algorithm is bounded by $\tilde{\mathcal{O}}\left(n^2\right)$.*

The reason that Lemma 13 is helpful is that we can sample a vertex v uniformly at random from S, and get, using Lemma 13, an unbiased estimator for the number of copies of $\mathtt{H}$ which contains v, and intersect S exactly k times, i.e., $t^k(v)$. By the law of total expectation, we get that the expected value of this quantity, is equal to $\frac{1}{|S|}\sum_{u\in S}\mathbb{E}\left[t^k(u)\right] = t^k/N_k$. Therefore, we get an unbiased estimator for t^k up to a known value N_k. A formal proof of Lemma 12 appears in the full version.

To prove Lemma 13, we utilize the color-coding technique introduced by [2]. The high level approach of the technique is to randomly color vertices with h colors and detect *colorful* h-cycles that are ordered by, say, increasing colors. This additional structure allows for faster detection, at the cost of some probability of missing h-cycles that are colored out of order, which is overcome by repeated experiments.

A pertinent question arises: why are the algorithms $\mathsf{Approx}_{\mathbb{E}[t^k]}, \mathsf{Approx}_{\mathbb{E}[\mathsf{vertex}\text{-}t^k]}$ necessary? Why not just choose a random coloring, compute the number of colorful copies of $\mathtt{H}$ intersecting a set S exactly k times, and apply Chebyshev's inequality to conclude that repeating this process $\tilde{\mathcal{O}}\left(n^2 \cdot \frac{b}{a\delta}\right)$ times suffices for a good approximation of t^k? The answer lies in the execution time of the matrix multiplication algorithm for counting colorful copies, which is dominated by the sizes of the largest, second largest, and smallest color classes. Roughly speaking, the smaller the product of these sizes, the faster the algorithm runs. Under random coloring, color classes each have a size of $\Omega(n)$ with high probability. Conversely, $\mathsf{Approx}_{\mathbb{E}[\mathsf{vertex}\text{-}t^k]}$ produces a color class containing just a single vertex, which significantly improves the running time in the worst case, compared to the approach which does not use the algorithms $\mathsf{Approx}_{\mathbb{E}[t^k]}, \mathsf{Approx}_{\mathbb{E}[\mathsf{vertex}\text{-}t^k]}$. We need the following definitions to explain how color-coding works, and how the algorithm $\mathsf{Approx}_{\mathbb{E}[\mathsf{vertex}\text{-}t^k]}$ uses rectangular matrix multiplication.

▶ **Definition 14.** *Fix some coloring $\varphi : V \to [\ell]$ for some $\ell \in \mathbb{N}$ (ℓ will usually be h). Let $\mathcal{C}_\varphi$ denote the set of all copies $C \in \mathcal{C}$, such that $\varphi(C) = [\ell]$. If $C \in \mathcal{C}_\varphi$, we say that C is* φ-colorful. *Also define $\mathcal{C}_\varphi(v) = \mathcal{C}_\varphi \cap \mathcal{C}(v)$, $t_\varphi \triangleq |\mathcal{C}_\varphi|$, and $t_\varphi(v) \triangleq |\mathcal{C}_\varphi(v)|$. Let $\mathcal{C}^k_\varphi \triangleq \mathcal{C}^k \cap \mathcal{C}_\varphi$. That is, $\mathcal{C}^k_\varphi$ is the set of copies of $\mathtt{H}$ in G, where each such copy is colorful w.r.t. φ, and additionally intersects the set S exactly k times. Let $t^k_\varphi = \left|\mathcal{C}^k_\varphi\right|$.*

Let A, B be two finite sets. We say that a function $\varphi : A \to B$ is a random coloring, *if the value of each $a \in A$ is set to some value $b \in B$, where b is chosen uniformly at random from B and independently of values chosen for other elements in A.*

Let $\varphi : V \to [h]$. For $i \in [h]$, we denote by $\varphi^{-1}(i)$ the set of all vertices v with $\varphi(v) = i$, and call this set the i-th color class. Assume without loss of generality that the color classes are sorted according their cardinalities, in a non-decreasing order. That is, for every $i < h$ we have $\left|\varphi^{-1}(i)\right| \geq \left|\varphi^{-1}(i+1)\right|$.

The last part of the above definition is used to quantify the complexity of computing t^k_φ as a function of the sizes of the color classes that the coloring φ induces, as follows.

▶ **Lemma 15.** *Let $(\varphi_1, \varphi_2, \varphi_h)$ denote the cardinality of the largest, second largest, and smallest color classes, respectively. For any fixed $k \in [h]$, there is a deterministic algorithm for computing $\left\{t^k_\varphi(v)\right\}_{v \in V}$ in time $\mathcal{O}\left((h!)^2 \cdot h^2 \cdot \mathsf{MM}\left(\varphi_1, \varphi_2, \varphi_h\right)\right)$.*

Next, we explain how to implement the algorithm $\mathsf{Approx}_{\mathbb{E}[\mathsf{vertex}\text{-}t^k]}$ given that we can compute the number t^k_φ of colorful copies of $\mathtt{H}$. The algorithm works as follows. It colors each vertex with a random color from the set $[h-1]$. It then recolors the input vertex by a new color h. Let φ denote this coloring. The algorithm then computes t^k_φ and outputs t^k_φ / q for some constant q such that $\mathbb{E}\left[t_\varphi / q\right] = t^k(v)$. We prove in the full version that the expectation and variance of this output satisfy the claimed requirements.

We are left with proving Lemma 15, which is the final step in the implementation of the algorithm $\mathsf{Approx}_{\mathbb{E}[\mathsf{vertex}\text{-}t^k]}$. To prove it, we reduce the problem of computing t^k_φ to the problem of computing t_φ on an auxiliary graph, in which every h-cycle is colorful and also intersects the set S exactly k times. We construct the auxiliary graph by randomly coloring the vertices with h colors, then selecting k color classes and keeping only vertices from S in them, while discarding the rest of the vertices in those classes. For the remaining $h-k$ color classes, we retain only vertices that are not part of S. We get a graph in which each color class is either contained in S or disjoint from S. This reduces the problem of computing t^k_φ on the auxiliary graph, to computing t_φ on it. The next claim addresses the running time of computing t_φ (on the auxiliary graph) instead of computing t^k_φ, and is the final missing piece for the proof of Theorem 9.

▷ Claim 16. Let G be a graph and let $\sigma \triangleq (U_1, \ldots, U_h)$ be an (ordered) sequence of disjoint subsets of vertices of $V(G)$. Let t_G^σ denote the number of copies $C \in \mathcal{C}_G$ where $C = (v_1, v_2, \ldots, v_h)$ and $v_i \in U_i$ for every $i \in [h]$. Let $U_{(1)}, U_{(2)}, U_{(h)}$ denote the cardinality of the largest, second largest, and smallest subset, respectively. Then, there is a deterministic algorithm that outputs $\{t_G^\sigma(v)\}_{v \in V}$ in time $\mathcal{O}\left(h^2 \cdot \mathsf{MM}\left(U_{(1)}, U_{(2)}, U_{(h)}\right)\right)$.

Proof of Claim 16. We assume without loss of generality that $|U_1| = U_{(h)}$, i.e., that U_1 is the smallest set. We first explain how to compute $\{t_G^\sigma(v)\}_{v \in U_1}$, and then we generalize this for U_j for any $j \in [h]$.

Let A denote the adjacency matrix of G. For $X, Y \subseteq V(G)$ let $A[X, Y]$ denote the submatrix containing all rows v for $v \in X$ and all columns u for $u \in Y$. Combinatorially, define a new directed graph H' with vertex set $X \cup Y$, and a directed edge (x, y) between a pair of vertices $x \in X$ and $y \in Y$ if and only if (x, y) is an edge in G. Note that $A[X, Y]$ is exactly the adjacency matrix of the new graph H'. Define $B_0 \triangleq I_{|U_1|}$, and for $0 \leq i < h$, define $A_i^\sigma \triangleq A[U_i, U_{i+1}]$ and $B_i^\sigma \triangleq B_{i-1} \cdot A_i$. We compute B_i^σ for $0 \leq i < h$. Note that $B_i^\sigma[x, y]$ denotes the number of paths with i edges between a vertex $x \in U_1$, and a vertex $y \in U_{i+1}$, which are of the form $(x, u_2, u_3, \ldots, u_{i-1}, y)$ where $u_j \in U_j$ for $2 \leq j < i$. Let $A_h \triangleq A[U_h, U_1]$. After computing B_h, we compute $M = B_h^\sigma \cdot A_h^\sigma$ and return all entries on the diagonal of M. Note that for $v \in U_1$, we have that $M[v, v] = t_G^\sigma(v)$.

The generalization to other values $j \in [h]$ has only a small modification and appears in the full version.

Running Time. In the i-th iteration, for $1 \leq i \leq h$, we compute the product of the matrices B_{i-1} with the matrix A_i. Let a', b' denote the dimensions of A_i. The dimensions of B_{i-1} are $U_{(h)}, a'$, and therefore the running time is $\mathsf{MM}\left(a', b', U_{(h)}\right)$. Without loss of generality, we can assume $b' \leq a'$, because $\mathsf{MM}(a', b', X) = \mathsf{MM}(X, b', a')$ for any a', b', X. We also have $a' \leq U_{(1)}$ and $b' \leq U_{(2)}$. This bounds the time for the i-th iteration by $\mathcal{O}\left(h^2 \cdot \mathsf{MM}\left(a', b', U_{(h)}\right)\right) \leq \mathcal{O}\left(h^2 \cdot \mathsf{MM}\left(U_{(1)}, U_{(2)}, U_{(h)}\right)\right)$, which proves the claim. ◁

5 Implementing the Find-Heavy Black Box

In this section, we prove the first part of Lemma 8, as is specified in the following theorem.

▶ **Theorem 17.** *There is an algorithm that implements the* $\mathsf{Find\text{-}Heavy}(G, \Lambda)$ *black box when* $\mathtt{H}$ *is an* h*-cycle with* $h = \mathcal{O}(1)$*, in time* $\tilde{\mathcal{O}}\left(\mathsf{MM}\left(n, n, n/\Lambda^{1/(h-2)}\right)\right)$.

An algorithm for Theorem 17 is given a graph G and a heaviness threshold Λ, and needs to output a superset of the Λ-heavy vertices, which contains no $\Lambda/(\log n)^{h^2}$-light vertices. Our algorithm works as follows. The algorithm selects a vector $P = (p_1, \ldots, p_h)$, where $p_i \in [0, 1]$ for $i \in [h]$, which we explain shortly how to select. The algorithm then samples a uniform coloring φ for the vertices, and keeps each vertex of the i-th color class with probability p_i. We emphasize that not all vertices are kept with the same probability. Let F denote the obtained graph. If v is in at least one φ-colorful cycle in F, we say that v is *P-discovered*. We can find all P-discovered vertices over F using Lemma 15. For every vertex $v \in V$, let $v[P]$ denote the probability that v is P-discovered. The randomness is taken over the choice of the coloring and the sampling of vertices. We call this experiment the *P-discovery experiment*. We repeat this P-discovery experiment k times. If v is P-discovered more than $k\tau$ times, where τ is a threshold we set later, then P adds v into the set of heavy vertices. In this case, we say that v is *P-added* to the set of heavy vertices.

The final step of our algorithm is choosing a vector P, or rather a set of vectors $\mathcal{P}$, and then performing the P-discovery experiment with each vector in the set k times. Before specifying how we should choose $\mathcal{P}$, k, and τ, we state the properties we hope to achieve.

(1) Each vector $P \in \mathcal{P}$ induces a graph F, such that invoking Lemma 15 for computing the set of P-discovered vertices, takes $\tilde{\mathcal{O}}\left(\mathsf{MM}\left(n, n, n/\Lambda^{1/(h-2)}\right)\right)$ time. (2) Each vertex with $t_G(v) \geq \Lambda$ has at least one $P \in \mathcal{P}$ that P-adds v to the set of heavy vertices, w.h.p. (3) Any vertex v with $t_G(v) \leq \Lambda/(\log n)^{h^2}$ is w.h.p. not P-added to the heavy vertex set for any $P \in \mathcal{P}$. (4) The set $\mathcal{P}$ has only $\tilde{\mathcal{O}}(1)$ vectors, allowing us to avoid repeating this experiment too many times.

The set of all vectors we will use is as follows. We take a (finite) subset of the all vectors $(p_1, \ldots, p_h) \in [0,1]^h$ which satisfy $\prod_{i\in[h]} p_i \leq \tilde{\mathcal{O}}(1/\Lambda)$. That is, all such vectors for which $p_i \in \{2^{-j} \mid 0 \leq j \leq \log(\Lambda) + 1\}$ for $i \in [h]$. We denote this set of vectors by $\mathsf{Product}_h(\Lambda)$. Note that $|\mathsf{Product}_h(\Lambda)| \leq (\log(\Lambda) + 2)^h = \tilde{\mathcal{O}}(1)$, where the last inequality follows because $\Lambda \leq n^h$, since no vertex in G participates in more than n^h copies of any h-vertex graph (that is, if the input was $\Lambda > n^h$, the algorithm could simply output an empty set). This means that $\log(\Lambda) \leq h \log n = \tilde{\mathcal{O}}(1)$. This proves Property (4) above.

The rest of the section proves that this set satisfies Properties (1)–(3). We first prove the first property, stating that for each $P \in \mathsf{Product}_h(\Lambda)$, the P-discovery experiment can be implemented in the desired time.

▶ **Lemma 18.** *Let G be a graph with n vertices and let Λ be some positive number. Let $P = (p_1, \ldots, p_h)$ be a vector in $[0,1]^h$ with $\prod_{i=1}^h p_i = \tilde{\mathcal{O}}\left(\frac{1}{\Lambda}\right)$. Let F be the (random) graph obtained in a P-discovery experiment. Then, we can find all the P-discovered vertices in time $\tilde{\mathcal{O}}\left(\mathsf{MM}\left(n, n, n \cdot \Lambda^{-1/(h-2)}\right)\right)$, w.h.p.*

Proof Sketch. The proof is included in the full version, and we provide the proof sketch here. Fix some $P \in [0,1]^h$ where $P = (p_1, \ldots, p_h)$, and $\prod_{i=1}^h p_i \leq 1/X$, for $X \geq 1$. Assume without loss of generality that $p_i \geq p_j$ for $i > j$. This implies that p_1 is the largest coordinate. Let F be the random graph, and let F_1, F_2, F_h denote the sizes of the first, second, and h color classes in F. First, note that we can use Lemma 15 to compute the set of discovered vertices in time $\mathcal{O}(\mathsf{MM}(F_1, F_2, F_h))$. We analyze the running time of the algorithm specified in Lemma 15 on the random graph F. Using a standard Chernoff's inequality, we can get that $\mathcal{O}(\mathsf{MM}(F_1, F_2, F_h)) = \tilde{\mathcal{O}}(\mathsf{MM}(np_1, np_2, np_h))$ w.h.p. The crux of the algorithm is the following inequalities $\mathsf{MM}(np_1, np_2, np_h) \leq \mathsf{MM}(n, n, n \cdot (p_1 p_2 p_h)) \leq \mathsf{MM}\left(n, n, n/X^{1/(h-2)}\right)$, which completes the proof by setting $X \leftarrow \Lambda$. The first inequality is proved in the full version. The second inequality reduces to solving the following optimization task. Maximize $p_1 \cdot p_2 \cdot p_h$, under the constraints $p_i \geq p_{i+1}$, and $\prod_{i\in[h]} p_i = 1/X$. The proof is also in the full version, where we show that an optimal value is obtained when $p_1 = p_2 = 1$ and $p_i = X^{1/(h-2)}$, which completes the proof sketch. ◀

It remains to prove Properties $(2) - (3)$. For this, we prove that each $\Lambda \cdot (2h \log(n))^{h+1}$-heavy vertex v, has a vector $P \in \mathsf{Product}_h(\Lambda)$ for which $v[P] = \Omega(1)$. On the other hand, for every $\Lambda/\log n$-light vertex v, and any vector $P \in \mathsf{Product}_h(\Lambda)$, we have $v[P] = \mathcal{O}(1/\log n)$, and therefore, we can distinguish between the two. The full details appear in the full version.

The upper bound on $v[P]$ for light vertex uses Markov's inequality. For the rest of this section we prove the statement on heavy vertices, by induction on h, for which the base case is $h = 3$. The intuition for the proof is as follows. Consider the base case of triangles. Let V_i denote the i-th color class. Consider finding the heavy vertices in the first color class. Consider all vectors $P_i = (1, 2^{-i}, 2^i/\Lambda)$, for $0 \leq i \leq \log(\Lambda) + 1$. These vectors form a subset of $\mathsf{Product}_h(\Lambda)$. Fix some Λ-heavy vertex $v \in V_1$. We want to prove that for at least one

$i \in [\log(\Lambda)]$ we have $v[P_i] \geq \Omega(1)$. Our choice for the vectors is designed to deal with the following two extreme cases. The first case is that every $C \in \mathcal{C}_\varphi(v)$ intersects one specific vertex $u \in V_2$. For this case, the vector $P_0 = (1, 1, \frac{1}{\Lambda})$ is the right choice for discovering v because it maximizes the probability we hit u and a common neighbor of v and u, under the constraint that the sampling probability $p_1 \cdot p_2 \cdot p_3 \leq 1/\Lambda$. The second case is that among each of $V_2 \cap N(v)$ and $V_3 \cap N(v)$, there are $\sqrt{\Lambda}$ vertices that are connected as a complete bipartite graph (contained in $V_2 \times V_3$). For this case, the vector P_i for $i = \log(\Lambda)/2$ is the right choice for discovering v, because $P_i = (1, \frac{1}{\sqrt{\Lambda}}, \frac{1}{\sqrt{\Lambda}})$. For a larger h, the "hard" case is the following generalization of the above. From each color class V_i for $i \in \{3, \ldots, h\}$ we take $k = \Lambda^{1/(h-2)}$ vertices and connect them by a complete $(h-2)$-partite graph. We then take $v \in V_1$ and $u \in V_2$, we add an edge between v and u, and we connect v to all the aforementioned k vertices from V_h and we connect u to all the aforementioned k vertices from V_3. The vector $(1, 1, \frac{1}{k}, \ldots, \frac{1}{k})$ is the right choice for this case.

Roughly speaking, we show the set $\mathsf{Product}_h(\Lambda)$ gradually shifts from handling one extreme case to another, and therefore "covers" all cases in between, which are the different ways to split vertices in h-cycles into h pieces whose product of sizes is Λ.

We need a final technical step before presenting the proof. First, we construct an h-partite graph which is easier to work with. For this, we sample uniform h coloring of the vertices of G. For $i \in [h]$ let V_i denote the set of vertices colored in the i-th color. For every $i \in [h]$ we keep only edges between the vertices of V_i and $V_{i+1 \mod h}$, and direct those edges from V_i to V_{i+1}. Let G_φ denote the obtained directed graph. We emphasize that $t_{G_\varphi}(v) \leq t_\varphi(v)$, as the latter counts all colorful cycles in which v participates in, whereas the former counts only colorful cycles with edges between V_i and V_{i+1} in which v participates in. We prove that a heavy vertex will be P discovered in G_φ, with probability at least $\Omega(1)$, by some $P \in \mathsf{Product}_h(\Lambda)$, whereas light vertices will only be discovered with probability $\mathcal{O}(1/\log n)$. In other words, since the gap between the heavy and light vertices is sufficiently large, we can still distinguish between the two in G_φ.

The rest of this section is dedicated to proving the following proposition.

▶ **Proposition 19.** *Fix a vertex v, and an h-coloring of the vertices of G. Suppose $t_{G_\varphi}(v) \geq \Lambda \cdot (2h\log(n))^{(h-1)^2}$. Then, there exists a vector $P \in \mathsf{Product}_h(\Lambda)$ such that the probability that v gets P-discovered in G_φ is at least $(1-1/e)^{h-1}$.*

Due to space considerations, we prove here only the base case where the induction step is deferred to the full version.

Proof of Proposition 19. We prove this by induction on h. We start with the base case, that is, $h = 3$. We fix some h-coloring φ. Recall that we denote the i-th color class by V_i. Consider a vertex $v \in V_1$ with $t_{G_\varphi(\pi)}(v) \geq \Lambda \cdot (2h\log(n))^4$ To simplify the notation, we use $G' = G_\varphi$. We will prove that there exists a vector $P \in \mathsf{Product}_h(\Lambda)$, such that the probability that the vertex v is P-discovered over G' is at least $(1-1/e)^2$. Let $K_0 = \log(\Lambda) + 1$. We consider a subset of vectors in $\mathsf{Product}_h(\Lambda)$ of the form $P_i = (1, p_i, q_i)$, for $i \in \{0, 1, \ldots, K_0\}$, where $p_i \triangleq 2^{-i}$, $q_i \triangleq \frac{2^i}{\Lambda}$. We partition V_2 into classes as follows. For $k \in \{1, 2, \ldots, K_0\}$, let $Q_k = \{u \in V_2 \mid |\mathcal{C}_{G'}(v) \cap \mathcal{C}_{G'}(u)| \in [1/q_k, 2/q_k)\}$, $Q_0 = \{u \in V_2 \mid |\mathcal{C}_{G'}(v) \cap \mathcal{C}_{G'}(u)| \geq 1/q_0\}$. In words, $u \in Q_k$ if and only if the number of 3-cycles in G' that contain both v and u is at least $1/q_k$ and less than $2/q_k$. The set Q_0 consists of all vertices $u \in V_2$, such that the number of 3-cycles in G' that contain both v and u is at least $1/q_0 = \Lambda$. We claim that there exists $k \in \{0, 1, \ldots, K_0\}$ such that $|Q_k| \geq 1/p_k$. The proof appears in the full version. We next show that for such k, we have $v[P_k] \geq (1-1/e)^{h-1}$ which completes the proof of the base case. Fix $k \in \{0, 1, \ldots, K_0\}$ where $|Q_k| \geq 1/p_k$. For any non-empty subset $S \subseteq Q_k$,

let $\mathcal{E}_1(S)$ denote the event that $\{Q_k \cap V_2[p_k] = S\}$. That is, $\mathcal{E}_1(S)$ denote the event that during the P_k-discovery experiment, the set of vertices that were sampled from $V_2 \cap Q_k$ is exactly the set S. Let $R(S)$ denote the subset of vertices in V_3 which participate in a 3-cycle (in G') that contains v and some additional vertex from S. Let $\mathcal{E}_2(S)$ denote the event that $\{V_3[q_k] \cap R(S) \neq \emptyset\}$. That is, $\mathcal{E}_2(S)$ denotes the event that during the P_k-discovery experiment, the set of vertices that were sampled from $V_3 \cap R(S)$ is not empty. Next, we show that

$$v[P_k] \geq \sum_{S:\emptyset \subsetneq S \subseteq Q_k} \Pr\left[\mathcal{E}_1(S) \cap \mathcal{E}_2(S)\right] \geq (1-1/e)^2 \ . \tag{1}$$

For every fixed $S \subseteq Q_k$, which is not empty, the events $\mathcal{E}_1(S)$ and $\mathcal{E}_2(S)$ are independent, since $\mathcal{E}_1(S)$ addresses sampling vertices from V_2, while $\mathcal{E}_2(S)$ addresses sampling vertices from V_3, where the two samples are independent of each other. Also note that $\mathcal{E}_1(S) \cap \mathcal{E}_2(S)$ is contained in the event that v is P_k-discovered, and since the events $\{\mathcal{E}_1(S)\}_{S \subseteq Q_k}$ are disjoint, so are the events $\{\mathcal{E}_1(S) \cap \mathcal{E}_2(S)\}_{S \subseteq Q_k}$. Therefore, the event that v is P_k-discovered, contains the union of the following disjoint events $\{\bigcup_{S:\emptyset \subsetneq S \subseteq Q_k} \mathcal{E}_1(S) \cap \mathcal{E}_2(S)\}$. We get

$$\begin{aligned} v[P_k] \geq \Pr[\bigcup_{S:\emptyset \subsetneq S \subseteq Q_k} \mathcal{E}_1(S) \cap \mathcal{E}_2(S)] &= \sum_{S:\emptyset \subsetneq S \subseteq Q_k} \Pr\left[\mathcal{E}_1(S) \cap \mathcal{E}_2(S)\right] \\ &= \sum_{S:\emptyset \subsetneq S \subseteq Q_k} \Pr\left[\mathcal{E}_1(S)\right] \Pr\left[\mathcal{E}_2(S)\right] \ . \end{aligned}$$

To complete the proof, we need to show that (1) $\sum_{S:\emptyset \subsetneq S \subseteq Q_k} \mathcal{E}_1(S) \geq 1-1/e$, and (2) that for every $S \subseteq Q_k$, which is not empty, we have $\mathcal{E}_2(S) \geq 1-1/e$. The first claim follows as

$$\sum_{S:\emptyset \subsetneq S \subseteq Q_k} \mathcal{E}_1(S) = \Pr\left[Q_k[p_k] \neq \emptyset\right] = 1-(1-p_k)^{|Q_k|} \geq 1-1/e \ ,$$

where the inequalities hold for the following reasons. The first two equalities follows from definition, and the last inequality follows from the assumption that $|Q_k| \geq 1/p_k$.

The second claim follows as every non-empty subset $S \subseteq Q_k$ satisfies $|R(S)| \geq 1/q_k$. To see this, fix some vertex $u \in S$. We have $R(u) \subseteq R(S)$, and $|R(u)| \geq 1/q_k$ because $u \in Q_k$. Therefore, for any such S, we have

$$\Pr\left[\mathcal{E}_2(S)\right] = \Pr\left[R(S)[q_k] \neq \emptyset\right] = 1-(1-q_k)^{|R(S)|} \geq 1-(1-q_k)^{1/q_k} \geq 1-1/e \ .$$

This completes the proof of Equation (1). The rest of the proof appears in the full version. ◀

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is valiant's parser. *SIAM J. Comput.*, 47(6):2527–2555, 2018.

2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.

3 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

4 Arturs Backurs and Christos Tzamos. Improving viterbi is hard: Better runtimes imply faster clique algorithms. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 311–321, 2017.

5 Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 623–632. ACM/SIAM, 2002.

6 Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient algorithms for large-scale local triangle counting. *ACM Trans. Knowl. Discov. Data*, 4(3):13:1–13:28, 2010.

7 Amartya Shankha Biswas, Talya Eden, Quanquan C. Liu, Ronitt Rubinfeld, and Slobodan Mitrovic. Massively parallel algorithms for small subgraph counting. In Amit Chakrabarti and Chaitanya Swamy, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, September 19-21, 2022, University of Illinois, Urbana-Champaign, USA (Virtual Conference)*, volume 245 of *LIPIcs*, pages 39:1–39:28. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.APPROX/RANDOM.2022.39.

8 Andreas Björklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. Listing triangles. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 223–234. Springer, 2014. doi:10.1007/978-3-662-43948-7_19.

9 Karl Bringmann and Philip Wellnitz. Clique-based lower bounds for parsing tree-adjoining grammars. In *Proceedings of the 28th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 12:1–12:14, 2017.

10 Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In Stijn Vansummeren, editor, *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*, pages 253–262. ACM, 2006.

11 Keren Censor-Hillel, Dean Leitersdorf, and Elia Turner. Sparse matrix multiplication and triangle listing in the congested clique model. *Theoretical Computer Science*, 809:45–60, 2020.

12 Keren Censor-Hillel, Dean Leitersdorf, and David Vulakh. Deterministic near-optimal distributed listing of cliques. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 271–280, 2022.

13 Yi-Jun Chang, Seth Pettie, Thatchaphol Saranurak, and Hengjie Zhang. Near-optimal distributed triangle enumeration via expander decompositions. *Journal of the ACM (JACM)*, 68(3):1–36, 2021.

14 Yi-Jun Chang and Thatchaphol Saranurak. Deterministic distributed expander decomposition and routing with applications in distributed derandomization. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 377–388. IEEE, 2020. doi:10.1109/FOCS46700.2020.00043.

15 Artur Czumaj and Christian Konrad. Detecting cliques in congest networks. *Distributed Computing*, 33(6):533–543, 2020.

16 Mina Dalirrooyfard, Surya Mathialagan, Virginia Vassilevska Williams, and Yinzhan Xu. Listing cliques from smaller cliques. *CoRR*, 2023. arXiv:2307.15871.

17 Mina Dalirrooyfard, Thuy-Duong Vuong, and Virginia Vassilevska Williams. Graph pattern detection: Hardness for all induced patterns and faster noninduced cycles. *SIAM J. Comput.*, 50(5):1627–1662, 2021.

18 Holger Dell, John Lapinskas, and Kitty Meeks. Approximately counting and sampling small witnesses using a colorful decision oracle. *SIAM J. Comput.*, 51(4):849–899, 2022.

19 Danny Dolev, Christoph Lenzen, and Shir Peled. "tri, tri again": finding triangles and small subgraphs in a distributed setting. In *Distributed Computing: 26th International Symposium, DISC 2012, Salvador, Brazil, October 16-18, 2012. Proceedings 26*, pages 195–209. Springer, 2012.

20 Talya Eden, Nimrod Fiat, Orr Fischer, Fabian Kuhn, and Rotem Oshman. Sublinear-time distributed algorithms for detecting small cliques and even cycles. *Distributed Computing*, pages 1–28, 2022.

21 Talya Eden, Amit Levi, Dana Ron, and C Seshadhri. Approximately counting triangles in sublinear time. *SIAM Journal on Computing*, 46(5):1603–1646, 2017.

22 Talya Eden, Dana Ron, and Will Rosenbaum. Almost optimal bounds for sublinear-time sampling of k-cliques in bounded arboricity graphs. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 56:1–56:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

23 Talya Eden, Dana Ron, and C. Seshadhri. On approximating the number of k-cliques in sublinear time. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 722–734. ACM, 2018.

24 Mira Gonen, Dana Ron, and Yuval Shavitt. Counting stars and other small subgraphs in sublinear-time. *SIAM J. Discret. Math.*, 25(3):1365–1411, 2011.

25 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 1–10, 1977.

26 Taisuke Izumi and François Le Gall. Triangle finding and listing in congest networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 381–389, 2017.

27 Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In Lusheng Wang, editor, *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings*, volume 3595 of *Lecture Notes in Computer Science*, pages 710–716. Springer, 2005.

28 Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming – 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 598–609. Springer, 2012.

29 Tamara G. Kolda, Ali Pinar, Todd D. Plantenga, C. Seshadhri, and Christine Task. Counting triangles in massive graphs with mapreduce. *SIAM J. Sci. Comput.*, 36(5), 2014.

30 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Comment. Math. Univ. Carol.*, 26(2):415–419, 1985.

31 Mihai Puatracscu. Towards polynomial lower bounds for dynamic problems. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 603–610. ACM, 2010.

32 Kanat Tangwongsan, A. Pavan, and Srikanta Tirthapura. Parallel triangle counting in massive streaming graphs. In Qi He, Arun Iyengar, Wolfgang Nejdl, Jian Pei, and Rajeev Rastogi, editors, *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 – November 1, 2013*, pages 781–786. ACM, 2013.

33 Jakub Tětek. Approximate triangle counting via sampling and fast matrix multiplication. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

34 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018.

35 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *Proc. SODA*, page to appear, 2024.

36 Raphael Yuster and Uri Zwick. Finding even cycles even faster. *SIAM J. Discret. Math.*, 10(2):209–222, 1997.

37 Raphael Yuster and Uri Zwick. Detecting short directed cycles using rectangular matrix multiplication and dynamic programming. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 254–260. SIAM, 2004.

The Group Access Bounds for Binary Search Trees

Parinya Chalermsook
Aalto University, Finland

Manoj Gupta
IIT Gandhinagar, India

Wanchote Jiamjitrak
University of Helsinki, Finland

Akash Pareek
IIT Gandhinagar, India

Sorrachai Yingchareonthawornchai
The Hebrew University of Jerusalem, Israel

Abstract

The access lemma (Sleator and Tarjan, JACM 1985) is a property of binary search trees (BSTs) that implies interesting consequences such as static optimality, static finger, and working set property on any access sequence $X = (x_1, x_2, \ldots, x_m)$. However, there are known corollaries of the dynamic optimality that cannot be derived via the access lemma, such as the dynamic finger, and any $o(\log n)$-competitive ratio to the optimal BST where n is the number of keys.

In this paper, we introduce the *group access bound* that can be defined with respect to a reference *group access tree*. Group access bounds generalize the access lemma and imply properties that are far stronger than those implied by the classical access lemma. For each of the following results, there is a group access tree whose group access bound

1. Is $O(\sqrt{\log n})$-competitive to the optimal BST.
2. Achieves the k-finger bound with an *additive* term of $O(m \log k \log\log n)$ (randomized) when the reference tree is an almost complete binary tree.
3. Satisfies the unified bound with an *additive* term of $O(m \log\log n)$.
4. Matches the unified bound with a time window k with an *additive* term of $O(m \log k \log\log n)$ (randomized).

Furthermore, we prove the simulation theorem: For every group access tree, there is an online BST algorithm that is $O(1)$-competitive with its group access bound. In particular, any new group access bound will automatically imply a new BST algorithm achieving the same bound. Thereby, we obtain an improved k-finger bound (reference tree is an almost complete binary tree), an improved unified bound with a time window k, and matching the best-known bound for Unified bound in the BST model. Since any dynamically optimal BST must achieve the group access bounds, we believe our results provide a new direction towards proving $o(\log n)$-competitiveness of the Splay tree and Greedy, two prime candidates for the dynamic optimality conjecture.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Dynamic Optimality, Binary Search Tree, Online Algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.38

Category Track A: Algorithms, Complexity and Games

Related Version *Full version:*: https://arxiv.org/abs/2312.15426

Funding This research was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 759557.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 38; pp. 38:1–38:18

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In the amortized analysis of a self-adjusting binary search tree (BST), the *access lemma* [32] is perhaps the most fundamental property of a BST algorithm that allows us to prove the competitiveness against many performance benchmarks, including temporal and spatial locality. If a BST algorithm satisfies the access lemma, then, by plugging in appropriate parameters, the algorithm also satisfies many interesting corollaries of the *dynamic optimality conjecture* including, for example, *balance theorem [32], static optimality [32, 20], static finger property [32], working set property [32], and key-independent optimality [19].* For example, Splay tree [32], Greedy [14, 30], and Multi-splay tree [34] are all known to satisfy the access lemma [32, 20, 34].

Despite these many applications, several strong BST properties cannot be implied via the access lemma, including *"non-trivial" competitiveness*, *k-finger property* (or even the (weaker) dynamic finger property [13]), and the *unified bound* [24, 17]. For completeness, we discuss each one of them in turn.

Competitiveness. Dynamic optimality conjecture [32] postulates that there is an online binary search tree (BST) on n keys whose cost to perform a search (access) sequence $(x_1, \ldots, x_m) \in \{1, 2, \ldots, n\}^m$ (including the cost to adjust the internal structure in between the sequence) is at most that of the offline optimum up to a constant factor. We say that a BST algorithm is $f(n)$-*competitive* if its total cost is, at most, the cost of the offline optimum up to a factor of $f(n)$. A BST algorithm is dynamically optimal if it is $O(1)$-competitive.

Splay tree [32] and Greedy [14] are widely regarded as the prime candidates for dynamic optimality conjecture. However, the best-known competitiveness of both the algorithms remains $O(\log n)$, which can be shown from the access lemma or (via static optimality) using any balanced trees. The access lemma cannot be used to prove $o(\log n)$-competitive (for example, we cannot even derive the *sequential access theorem* [33] via the access lemma). In contrast, there are BST algorithms with $O(\log \log n)$-competitiveness. In [15], the authors presented the first $O(\log \log n)$-competitive binary search tree, which they call Tango Trees. Several subsequent results provided alternate $O(\log \log n)$-competitive algorithms [21, 35, 3, 11]. Despite achieving the best-known competitive ratios, the fact that these algorithms do not satisfy many corollaries of the dynamic optimality conjecture makes them less promising than Splay and Greedy.

***k*-Finger Bound.** Several notions of finger bounds [32, 13, 26] have been introduced to capture the "locality" of input sequences. The strongest finger bound, called k-Finger bound, was motivated by the connection between BSTs and the k-server problem [8]. It has still remained unclear whether any online BST algorithm achieves this property.

We define k-Finger bound as follows. Assume we have an almost complete binary tree where each leaf represents a distinct key from $\{1, \ldots n\}$. This tree is called the *reference tree*. Assume that there are k-fingers stationed at k arbitrary leaves in the reference tree.

▶ **Definition 1 (k-Finger Bound).** *When a key x_t is searched at time t, we must move one finger from its current position to the node containing x_t. The cost of the search is the number of nodes on the unique path connecting the finger's source to its destination. We define $F^k(X)$ as the minimum, overall finger movement strategies distance traversed by the fingers to process the sequence X when the reference tree is an almost complete binary tree.*

The classical access lemma cannot imply the k-finger property even when $k = 1$ (called the lazy finger bound [26], which generalizes the dynamic finger bound [13]). Nonetheless, it is possible to prove a non-trivial bound w.r.t. k-Finger using different techniques. In [8],

the authors claimed the existence of an online BST algorithm with cost $O((\log k)^7 \mathrm{F}^k(X))$. However, this claim has an error since the algorithm employs Lee's [29] k-server result, which the author has retracted. Instead of Lee's result, we can use the k-server result of Koutsoupias and Papadimitriou [27], $(2k-1)$-competitive. By using [8], it implies that an online BST algorithm exists with a running time of $O(k\mathrm{F}^k(X))$. This is a relatively large gap when compared with the best achievable offline BST bound of $O(\log k)F^k(X)$ [8]; in fact, whether there exists a BST whose cost is additive in the k-finger bound, that is $O(F^k(X) + m \log k)$, has remained open.

Unified Bound. To describe the unified bound, we should first understand the *working set bound* [25, 32] and the *dynamic finger bound* [13, 32, 26]. If x_t is a search key, then the working set bound requires x_t to be searched in amortized time $O(\log \mathrm{Ws}(x_t))$, where $\mathrm{Ws}(x_t)$ is the number of distinct keys searched since the last search of x_t. The working set bound is based on temporal locality and implies many more bounds, such as the static finger bound, the static optimality bound, etc. The dynamic finger bound states that the amortized time to search x_t is $O(\log |x_t - x_{t-1}| + 2)$. The dynamic finger bound is based on spatial locality.

The *unified bound* [24] implies both the working set and the dynamic finger.

▶ **Definition 2.** *The unified bound can be defined as* $\mathrm{UB}(X) = \sum_{t=2}^{m} \log(\min_{t'<t}\{t - t' + |x_t - x_{t'}| + 2\})$.

The unified bound is stronger than both the working set and the dynamic finger, as it suggests that it is in-expensive to search a key that is close to a recently searched key. It is true that the access lemma cannot prove the unified bound, although there are data structures that satisfy the unified bound. Iacono [24] gave a comparison-based data structure called the unified structure that achieves the unified bound. In [1], the authors gave a dynamic comparison-based data structure that achieves the unified bound. Derryberry and Sleator [17] designed the first BST algorithm called Skip-Splay tree that nearly achieves the unified bound with its running time of $O(\mathrm{UB}(X) + m \log\log n)$. In [4], the authors modified Skip-Splay using layered working-set trees to get amortized $O(\mathrm{UB}(x_i) + \log\log n)$ time, where x_i is a search key at time i. In Derryberry's thesis [18], Cache trees were introduced as the first BST algorithm aiming to achieve the Unified bound. However, this assertion was later questioned by Sleator in [31]. The question of whether a BST algorithm can truly achieve the Unified bound remains open and intriguing.

Unified Bound with time window. In the unified bound, for each x_t, we find a key $x_{t'}$ that minimizes the term $(t - t' + |x_t - x_{t'}| + 2)$ where $t' < t$. We can add another condition that t' should be one of the last k searched keys before time t. Thus, t' comes from some time window. We call this variant *Unified Bound with a time window*. We formally define it as follows.

▶ **Definition 3.** *Given an integer* k*, the unified bound with a time window is*

$$UB^k(X) = \sum_{t=2}^{m} \log\Big(\min_{t' \in [t-k \ldots t-1]} \{t - t' + |x_t - x_{t'}| + 2\}\Big).$$

This bound can be seen as "interpolating" between the dynamic finger and the unified bounds: When $k = 1$, $\mathrm{UB}^k(X)$ is the dynamic finger bound, and when $k = m$, $\mathrm{UB}^m(X)$ is the unified bound. For $k = 1$, we know that both Splay tree and Greedy satisfy the dynamic finger bound. The authors of [8], showed a relation between $\mathrm{UB}^k(X)$ and $\mathrm{Opt}(X)$, i.e,

$$\textsc{Opt}(X) \leq \beta(k)\textsc{Ub}^k(X)$$

where $\beta(\cdot)$ is some fixed (super exponentially growing) function.

In sum, the access lemma is an intrinsic property of a BST which implies nice properties but cannot seem to imply any of the aforementioned properties.

1.1 Our New Concept: The Group Access Bounds

The main conceptual contribution of this paper is to introduce the *group access bounds* that generalize the bound from the access lemma. Informally, the access lemma states that the amortized cost to access x_t at time is $O\left(\log \frac{W}{w(x_t)}\right)$, where $w(x_t)$ is the weight of x_t and W is the sum of weights of all the keys in the BST.

We give an informal definition of the group access bounds (see Section 3 for the formal definitions). Denote $[n] = \{1, \ldots, n\}$. The key gadget to describe our bound is the notion of *group access tree*, which captures a hierarchical partition of $[n]$ until singletons are obtained. That is, in a group access tree $\mathcal{T}$, each node $v \in V(\mathcal{T})$ is associated with an "interval" $I_v \subseteq [n]$ (consecutive integers). The root $r \in V(\mathcal{T})$ has $I_r = [n]$, if a node v has children $v_1, \ldots, v_k$, then we have that $\{I_{v_1}, \ldots, I_{v_k}\}$ forms a partition of I_v. This process continues until each leaf $v \in V(\mathcal{T})$ is associated with a singleton. Note that the tree does not have to be binary. See Figure 1 for illustration.

Figure 1 Examples of two group access trees. Each represents a hierarchical partition of $[n]$ until singletons are obtained. When a group access tree is a star (LHS), our bound is simply the access lemma.

Let w be a positive weight function that assigns a real-valued weight to each node in $\mathcal{T}$. We define the cost to access the tree $\mathcal{T}$ w.r.t. the weight function w as follows. For any edge (u, v) where u is the parent of v, the cost on (u, v) is $\log \frac{W(u)}{w(v)}$ where $W(u)$ is the total weight of all the children of u. The access cost of a key $a \in [n]$, denoted as $\text{cost}_{\mathcal{T},w}(a)$, is defined as the total cost of all the edges in the path P_a from the root to the leaf containing a in the group access tree $\mathcal{T}$. That is,

$$\text{cost}_{\mathcal{T},w}(a) = \sum_{(u,v)\in P_a} \log \frac{W(u)}{w(v)}.$$

Some readers may have observed the similarity between our cost function and that of the access lemma. Indeed, one can show that it generalizes the access lemma.

▶ **Observation 4.** *If the group access tree $\mathcal{T}$ is a star, then the access $a \in [n]$ on $\mathcal{T}$ gives the same (amortized) cost as the access lemma on the same weight function.*

Intuitively, the group access bound offers a "search tree" (which is not necessarily binary) where the cost of searching a is the sum of the cost of the edges on the search path P_a.

Let $\mathcal{W} = (w^{(1)}, \ldots, w^{(m)})$ be a sequence of weight functions. The total cost of the group access tree $\mathcal{T}$ on an access sequence $X = (x_1, \ldots, x_m)$, where $x_t \in [n]$ for all t, is

$$\text{cost}_{\mathcal{T},\mathcal{W}}(X) = \sum_t \text{cost}_{\mathcal{T},w^{(t)}}(x_t).$$

Similar to the access lemma, the weight functions should change in a controllable manner in order to be meaningful (e.g., the weight functions for the working set bound are changed in a structured way in the access lemma). Here, we introduce the notion of *locally bounded* weight families. We say that a sequence of weight functions $\mathcal{W} = (w^{(1)}, \ldots, w^{(m)})$ is *locally bounded* if for all t, the weight increase from time t to $t+1$ can happen only at the nodes in the path from root to x_t in the group access tree $\mathcal{T}$.

Given an input sequence $X = (x_1, \ldots, x_m)$, *the group access bound* $\mathsf{GAB}(\mathcal{T}, X)$ w.r.t. a group access tree $\mathcal{T}$ is

$$\mathsf{GAB}(\mathcal{T}, X) = \min_{\mathcal{W} \text{ locally bounded}} \text{cost}_{\mathcal{T},\mathcal{W}}(X).$$

1.2 Our Technical Results: Deriving BST Bounds from GAB

We show that the cost of the group access tree (with respect to certain weight functions) is competitive against many strong BST bounds that are not known via the access lemma. We say a group access bound is *randomized* if the group access tree $\mathcal{T}$ is obtained by a random process that iteratively partitions $[n]$ until singletons are obtained.

▶ **Theorem 5.** *For each of the following bounds, there exists deterministic group access trees $\mathcal{T}_1, \mathcal{T}_3$ and randomized group access trees $\mathcal{T}_2, \mathcal{T}_4$ such that for all access sequences $X = (x_1, \ldots, x_m)$ where $x_t \in [n]$ for all t,*

1. $\mathsf{GAB}(\mathcal{T}_1, X) = O(\sqrt{\log n}) \cdot \textsc{Opt}(X)$ *where $\textsc{Opt}(X)$ is the cost of offline optimal BST on X.*
2. $\mathsf{GAB}(\mathcal{T}_2, X) = O(F^k(X) + m \log k \log\log n)$ *(randomized). That is, it is competitive with k-finger up to an additive term when the reference tree is an almost complete binary tree.*
3. $\mathsf{GAB}(\mathcal{T}_3, X) = O(\textsc{Ub}(X) + m \log\log n)$*. That is, it is competitive with the unified bound up to an additive term.*
4. $\mathsf{GAB}(\mathcal{T}_4, X) = O(\textsc{Ub}^k(X) + m \log k \log\log n)$ *(randomized). That is, it is competitive with unified bound with a time window up to an additive term.*

The group access tree for the second and the fourth bound can be efficiently constructed from a probability distribution.

It is not immediately clear that these group access bounds can be realized by BST algorithms. We show that every group access tree $\mathcal{T}$ can be simulated by an online BST algorithm. Let $\mathcal{A}$ be a BST algorithm. We denote $\text{cost}_{\mathcal{A}}(X)$ to be the cost of algorithm $\mathcal{A}$ running on a sequence $X = (x_1, \ldots, x_m)$.

▶ **Theorem 6** (Simulation Theorem)**.** *For any group access tree $\mathcal{T}$, there exists an online BST algorithm $\mathcal{A}$ such that for any sufficiently long access sequence X , $\text{cost}_{\mathcal{A}}(X) = O(\mathsf{GAB}(\mathcal{T}, X))$. Furthermore, if the group access tree $\mathcal{T}$ is randomized, then $\mathcal{A}$ is randomized, where the competitive ratio is measured in the oblivious adversary model.*

The Simulation Theorem (Theorem 6) signifies that one can prove new BST bounds by just proving the existence of a group access tree along with locally bounded weight families.

Table 1 Summary of the new bounds that can be derived via the group access bound. The first two columns show the best-known results prior to this paper. Asymptotic notations (Big-Oh and Big-Omega) are hidden for brevity.

	Offline Upper	**Online Upper**	**Our new bounds**
OPT	$\mathsf{OPT} \log\log n$ [15]	$\mathsf{OPT} \log\log n$ [15]	$\mathsf{OPT}\sqrt{\log n}$
F^k	$F^k(X) \log k$ [8]	$kF^k(X)$ [8]	$F^k(X) + m \log k \log\log n$
UB	$\mathsf{UB}(X) + m \log\log n$ [17]	$\mathsf{UB}(X) + m \log\log n$ [17]	$\mathsf{UB}(X) + m \log\log n$
UB^k	$\beta(k)\mathsf{UB}^k(X)$ [8]	$\beta(k)\mathsf{UB}^k(X)$ [8]	$\mathsf{UB}^k(X) + m \log k \log\log n$

1.3 Significance of our results

Beyond the access lemma, group access bounds serve as the first step towards providing a unified framework for proving binary search tree bounds systematically. To resolve the dynamic optimality conjecture, a candidate algorithm must satisfy all dynamic optimality corollaries simultaneously, and we believe group access bounds are the starting point for this.

Apart from the competitiveness result, the results we present here either match the best-known bounds (such as unified bound) or provide an improvement upon even the best-known offline algorithms (k-finger bound when reference tree is an almost complete binary tree and unified bound with bounded time window) in the BST model.

We remark that even though our competitive ratio does not match the $O(\log\log n)$ best-known factor, what we prove is more general where any BST algorithm satisfying the GAB will automatically be $O(\sqrt{\log n})$- competitive. The BST algorithm we present here is called GGREEDY, a very similar algorithm to GREEDY – a prime candidate for dynamic optimality. GGREEDY resembles Greedy and inherits its conceptual simplicity. Even after a lot of work in this area [20, 26, 5, 7, 6, 8, 11, 22, 23, 17, 28, 9], only the trivial bound for GREEDY is known: $\textsc{Greedy}(X) = O(\log n)\ \textsc{Opt}(X)$ for all possible X. Thus, GREEDY is $O(\log n)$-competitive. Our result raises some hope of proving $o(\log n)$-competitive ratio for Greedy by showing that Greedy satisfies the group access bound.

We illustrate the power of the group access bound by deriving two new BST bounds that have not been known for even offline BST algorithms.

▶ **Corollary 7.** *For each of the following items, there is a randomized online BST algorithm $\mathcal{A}$ such that the expected cost for any search sequence $X = (x_1, \ldots, x_m)$ where $x_t \in [n]$ for all t,*

- $cost_{\mathcal{A}}(X) = O(F^k(X) + m \log k \log\log n)$, *and*
- $cost_{\mathcal{A}}(X) = O(UB^k(X) + m \log k \log\log n)$.

The algorithms are randomized because the group access trees are chosen based on a probability distribution. The bound $O(F^k(X) + m \log k \log\log n)$ gives an improvement from the best known online BST that costs $O(kF^k(X))$ in [8] and improves upon the offline bound $O(\log k)F^k(X)$ for some range of parameter k when the reference tree is an almost complete binary tree.

From such an improvement, we can derive a new "pattern-avoiding" bound for BSTs. We say that a sequence *contains* another sequence (or pattern) π if it contains a subsequence that is order-isomorphic to π.

▶ **Corollary 8.** *Let $X \in [n]^m$ be a sequence that does not contain the pattern $(k, k-1, \ldots, 1)$. Then there exists a randomized BST that accesses X with cost $O(nk + m \log k \log\log n)$.*

This bound improves the best-known online algorithm [9] that gives a bound of $O(mk^2)$ and the best-known bound on the offline optimum $O(mk)$ [8] for some range of parameters when the reference tree used to calculate $F^k(X)$ is an almost complete binary tree.

The unified bound obtained in this paper matches the best-known bound of $O(\text{UB}(X) + m \log \log n)$ by Derryberry and Sleator [17]. In fact, we show that we can use the analysis of [17] as a black box once we obtain the group access tree for Unified bound.

We do expect more applications of the group access bounds in binary search trees since group access bounds are generic and yet (unlike the dynamic optimality conjecture) maintain a certain flavor of being "static" (since $\mathcal{T}$ is still fixed). We show that this static component can be algorithmically leveraged. We believe that our bound offers a "bridge" between the relatively static access lemma to the dynamic optimality conjecture, which requires a full understanding of dynamic BSTs.

Contribution to potential function analysis. Another interesting aspect of our work is that once the group access bound is formulated, the proof relies solely on the use of the standard Sum-of-logs potential function, which in general, does not seem sufficient to prove any strong bounds beyond the access lemma. We show that by "augmenting" the access lemma with a group access tree, a natural and standard sum-of-logs potential function immediately provides significantly stronger BST bounds.

In general, designing a potential function for analyzing a given algorithm is a highly innovative but rather ad-hoc task. Our work suggests that the sum-of-logs potential function on the group access tree might be a good candidate for proving that Greedy or Splay is $O(\sqrt{\log n})$-competitive.

Combining BSTs. In [16], the authors showed that different BSTs with well-known bounds can be combined into a single BST that achieves all the properties of the combined BSTs. For example, Tango trees and Skip-Splay trees can be combined to get a BST, which is $O(\log \log n)$-competitive and achieves the Unified bound with an additive term of $O(\log \log n)$. The combined BST from their approach, however, results in a different BST algorithm. Our group access bounds offer another way to combine known BST bounds, as we have illustrated in the above discussion, while retaining the original algorithm, e.g., proving that Splay satisfies the group access bound implies that Splay itself possesses all the nice properties derived from GAB.

1.4 Concluding Remarks

We propose the group access bound – a far-reaching extension of the standard access lemma and present applications in deriving new and unifying old bounds. Some of our bounds even improve the best-known upper bound on the offline optimum on some range of parameters.

An immediate (and perhaps most interesting) open question is whether Greedy or Splay satisfies the group access bound via the sum-of-logs potential function. We believe that this question is very concrete (since it involves a specific potential function), so proving or refuting it would not be beyond our reach.

Developing further understanding and finding more applications of our group access bounds are interesting directions. For instance, what are other BST bounds that can be implied by the group access bounds? Can we show that $\mathsf{GAB}(\mathcal{T}, X) \leq O(\mathsf{OPT}(X))$ for some (distribution of) group access tree $\mathcal{T}$? Can one derive a non-trivial result about more general pattern-avoiding bounds [6, 23, 10, 12]? It was shown that optimal BST takes linear time for a pattern of bounded length [2]. Can we design a BST algorithm with a running time $O(\text{F}^k(X) + m \log k)$ for any sequence X of length m where the k-finger bound is calculated on any arbitrary reference tree?

There are also open questions to settle the complexity of specific BST bounds both in the online and offline settings. Most notably, is there any BST data structure that satisfies the unified bound?

1.5 Organization

We introduce notation and terminology in Section 2. We formalize the description of the group access bounds in Section 3. We prove $O(\sqrt{\log n})$-competitiveness in Section 4. Owing to space limitations, the k-finger bound, unified bound and unified bound with a time window can be found in the full version of the paper. To prove the Simulation theorem (Theorem 6), we first define an algorithm, called GGREEDY, that simulates the group access tree in Section 5. We derive the group access lemma in Section 6 and finally prove the Simulation theorem (See to the full version of the paper). Certain proofs and sections are removed which can be found in the full version of the paper.

2 Preliminaries

Let $X = (x_1, x_2, \ldots, x_m)$ be a sequence of m accesses where each access is from the set $\{1, 2, ..., n\}$. This sequence can be represented as points in the plane, that is, $X_p = \{(x_t, t) : t \in [m]\} \subseteq \mathbb{R}^2$. Imagine these points on a plane with an origin and X-Y axis. Since both the coordinates of a point are positive, all points lie in the first quadrant. The positive x-axis represents the key space, and the y-axis represents time. For any two points p, q in a point-set P, if they are not in the same horizontal or vertical line, we can form a rectangle $\square pq$. A rectangle $\square pq$ is said to be *arborally satisfied* if $\exists r \in P \setminus \{p, q\}$ such that r lies in $\square pq$. [14] introduced us to the following beautiful problem:

▶ **Definition 9** (Arborally Satisfied Set). *Given a point set X_p, find a point set Y such that $|X_p \cup Y|$ is minimum and every pair of points in $X_p \cup Y$ is arborally satisfied.*

[14] showed that finding the best BST execution for a sequence X is equivalent to finding the minimum cardinality set Y such that $X_p \cup Y$ is arborally satisfied.

▶ **Lemma 10** (See Lemma 2.3 in [14]). *Let A be an online algorithm that outputs an arborally satisfied set on any input representing X. Then, there is an online BST algorithm A' such that the cost of A' is asymptotically equal to the cost of A, where the cost of A is the number of points added by A plus the size of X.*

At time t, we say that x_t is an *access key*. An algorithm adds (or touches) *points* while processing a key with the aim of making the final point set arborally satisfied. For a point p, denote $p.x$ and $p.y$ as its x-coordinate and y-coordinate, respectively. Note that $p.x$ denotes a key, and $p.y$ denotes the time when the point p was added.

Let q be a key. When we say that GREEDY (or any other algorithm) adds a *point at key* q at time t, it means that GREEDY adds a point at coordinates (q, t). Given two points p_1 and p_2, p_2 lies to the right of p_1 if $p_2.x > p_1.x$, else it lies to the left of p_1. For brevity, we will avoid using ceil and floor notation for various parameters used in this paper.

3 The Group Access Bound

In this section, we describe the *group access bound*, which generalizes the access lemma. The concept of group access bound consists of two ingredients:

- **Hierarchical partition**: An **interval partition** of $[n]$ is a partition π such that each set $S \in \pi$ is an interval (i.e. consecutive integers). Let π be an interval partition of $[n]$. We say that π is a refinement of another partition π' if for all $S \in \pi$ and $S' \in \pi'$, we have $S \subseteq S'$ or $S \cap S' = \emptyset$. For instance $\{\{1,2\},\{3,4\},\{5,6\}\}$ is a refinement of $\{\{1,2,3,4\},\{5,6\}\}$.
  A hierarchical partition is a sequence of partitions $\mathcal{P} = \{\pi_0, \pi_1, \ldots, \pi_k\}$ such that (i) for all i, π_{i+1} is a refinement of π_i, (ii) $\pi_0 = \{[n]\}$ and (iii) $\pi_k = \{\{i\}\}_{i\in[n]}$ (singletons). Given such $\mathcal{P}$, each interval (set) in π_i is referred to as a **group**. The sets in π_i are called level-i groups for $\mathcal{P}$.
  A hierarchical partition $\mathcal{P}$ has a natural corresponding tree $\mathcal{T}$ where each node in $V(\mathcal{T})$ corresponds to a group. The root of $\mathcal{T}$ is $[n]$ (the group in π_0). Level-i of the tree contains nodes that have 1-to-1 correspondence with sets in π_i. Moreover, there is an edge connecting $S \in \pi_i$ to $S' \in \pi_{i+1}$ if $S' \subseteq S$.
- **Weight functions**: Given a canonical hierarchical partition $\mathcal{P}$, a $\mathcal{P}$-weight function is an assignment of positive real values to nodes in $V(\mathcal{T})$, i.e., $w : V(\mathcal{T}) \to \mathbb{R}_{>0}$.

We use the term **group access tree** to denote a hierarchical partition $\mathcal{P}$ (as well as its corresponding tree $\mathcal{T}$). See Figure 2 for illustration.

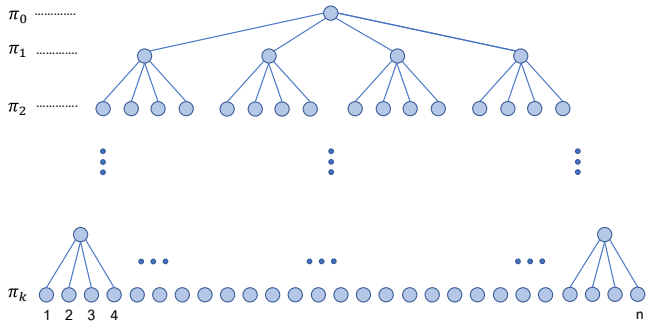


Figure 2 An illustration of group access tree.

▶ **Definition 11** (Group given a key). *Let $\mathcal{P}$ (or $\mathcal{T}$) be a group access tree and let $p \in [n]$ be a key. Then, for each j, we use $g_j(p)$ to denote the (unique) level-j group $S \in \pi_j$ in which p lies.*

▶ **Definition 12** (Level j groups of a group). *Given a level-$(j-1)$ group (interval) g, denote by $\mathsf{CG}(g)$ the set of children groups in level j that are contained in g, i.e., $\mathsf{CG}(g) = \{g' \in \pi_j : g' \subseteq g\}$. These are exactly the same as the groups that are children of node $g \in V(\mathcal{T})$.*

We are now ready to define the access cost in the group access tree. When accessing key $a \in [n]$ in the group access tree $\mathcal{T}$ and $\mathcal{P}$-weight w, denote by $\mathcal{T}(a)$ the path from root to a in the tree $\mathcal{T}$; in particular, this path contains $\mathcal{T}(a) = (g_0(a), g_1(a), \ldots, g_k(a) = \{a\})$. The cost incurred on edge $e = (g_{j-1}(a), g_j(a))$ is

$$c_e(\mathcal{T}, w, a) = \log\left(\frac{W^j}{w(g_j(a))}\right)$$

where $W^j = \sum_{g' \in \mathsf{CG}(g_{j-1}(a))} w(g')$. The total access cost is $c(\mathcal{T}, w, a) = \sum_{e\in\mathcal{T}(a)} c_e(\mathcal{T}, w, a)$. Notice that this access cost is very similar to the access lemma cost. In fact, one can show that it generalizes the access lemma:

▶ **Observation 13.** *The access lemma corresponds to the cost $c(\mathcal{T}, w, a)$ when $\mathcal{T}$ is a star.*

In this way, our group access bound can be seen as an attempt to strengthen the standard access lemma by introducing hierarchical partitioning. As in the access lemma, the interesting application to BSTs happens when the changes of weights are "controllable" (e.g., in the working set bound). We introduce the concept of *locally bounded* weight families $\mathcal{W}$ to capture this property.

▶ **Definition 14.** *The weight family $\mathcal{W}$ is locally bounded if for all time t, for every group $g \notin \mathcal{T}(x_t)$, we have $w_{t+1}(g) \leq w_t(g)$. This means that the weight can increase only when $g \in \mathcal{T}(x_t)$.*

We are interested in the total access cost on a sequence $X = (x_1, \ldots, x_m) \in [n]^m$ where the group access trees are allowed weight changes over time, that is, we are given a sequence of weight functions $\mathcal{W} = \{w_1, \ldots, w_m\}$ where w_t denotes the weight function at time t.

The **group access bound** w.r.t. $(\mathcal{T}, X)$ is:

$$\mathsf{GAB}(\mathcal{T}, X) = \min_{\mathcal{W} \text{ locally bounded}} \sum_{t \in [m]} c(\mathcal{T}, w_t, x_t).$$

Note that the use of "minimum" in the definition of group access bound serves to select the weight family with the lowest weight among multiple locally bounded weight families.

Our main contribution is in showing that the group access bound is competitive to many strong bounds in the binary search tree model. We say that the group access bound is (α, β)-**competitive** to function $f : [n]^m \to \mathbb{R}$ if there exists a group access tree $\mathcal{T}$ such that $\mathsf{GAB}(\mathcal{T}, X) \leq \alpha f(X) + \beta|X|$.

▶ **Theorem 15.** *The group access bound is $(O(\sqrt{\log n}), O(1))$-competitive to* OPT.

The group access bound can also be used by allowing randomization in choosing the hierarchical partition $\mathcal{P}$. We say that the group access bound is randomized (α, β)-**competitive** to function $f : [n]^m \to \mathbb{R}$ if there is an efficiently computable distribution $\mathcal{D}$ that samples $\mathcal{T}$ such that

$$\mathbb{E}_{\mathcal{T} \sim \mathcal{D}}[\mathsf{GAB}(\mathcal{T}, X)] \leq \alpha f(X) + \beta|X|.$$

▶ **Theorem 16.** *The group access bound is (randomized) $(O(1), O(\log k \log\log n))$-competitive to the k-finger bound when the reference tree is an almost complete binary tree.*

▶ **Theorem 17.** *The group access bound is $(O(1), O(\log\log n))$-competitive to the unified bound.*

▶ **Theorem 18.** *The group access bound is (randomized) $(O(1), O(\log k \log\log n))$-competitive to the unified bound with a time window of size k.*

We say that a BST algorithm $\mathcal{A}$ **satisfies the group access bound** w.r.t. group access tree $\mathcal{T}$ if the cost of the algorithm is at most $O(\mathsf{GAB}(\mathcal{T}, X))$ for all input sequence X.

▶ **Proposition 19.** *If a BST algorithm satisfies the group access bound and the group access bound is (α, β)-competitive to function f, then the cost of the BST algorithm on any sequence X is at most $O(\alpha \cdot f(X) + \beta|X|)$.*

We will later show that a family of BST algorithms (named Ggreedy) satisfies the group access bound and possesses all the aforementioned properties.

4 $(O(\sqrt{\log n}), O(1))$-competitiveness

In this section, we prove Theorem 15. We will define the appropriate group access tree so that the group access bound is upper bounded by $(O(\sqrt{\log n}), O(1)) \cdot \mathsf{OPT}(X)$.

Group access tree

Define the partition $\mathcal{P}$ inductively as follows. Let $M = 2^{\sqrt{\log n}}$. First $\pi_0 = \{[n]\}$. Given π_i, we define π_{i+1} by, for each interval $S \in \pi_i$, partitioning S into $S_1, S_2, \ldots, S_M$ equal-sized intervals and adding them into π_{i+1}. This would give us a group access tree where each non-leaf node has M children and its height is at most $h = O(\sqrt{\log n})$.

Weight function

Given a sequence $X = (x_1, x_2, \ldots, x_m)$, we define a weight function $\mathcal{W}$ which is locally bounded and such that $c(\mathcal{T}, \mathcal{W}, X) = \sum_t c(\mathcal{T}, \mathcal{W}, x_t) \leq O(\sqrt{\log n}) \cdot (\mathsf{OPT}(X) + m)$. This will give us the desired result. Our weight function uses the notion of *last access*.

▶ **Definition 20.** *Consider time t and the group $g = g_{j-1}(x_t)$. Let $t' < t$ be the last time before t at which g is on the search path $\mathcal{T}(x_t)$. We say that a child $g_1 \in \mathsf{CG}(g)$ is **last accessed (child) group** of g at time t if the edge (g, g_1) is on search path $\mathcal{T}(x_{t'})$.*

Remark that each group can have at most one child in $\mathcal{T}$ that is the last access group. Now, we are ready to define the weight function:

$$w_t(g) = \begin{cases} M & \text{if } g \text{ is the last accessed group of its parent} \\ 1 & \text{otherwise} \end{cases}$$

It is easy to verify that this family of weight functions $\mathcal{W}$ is locally bounded (there is only one key whose weight can increase between time t and $(t+1)$).

▶ **Lemma 21.** *Consider edge $e = (g_{j-1}(x_t), g_j(x_t))$. We have*

$$c_e(\mathcal{T}, \mathcal{W}, x_t) = \begin{cases} O(\log M) & \text{if } g_j(x_t) \text{ is not the last accessed group of } g_{j-1}(x_t) \\ O(1) & \text{otherwise} \end{cases}$$

Proof. By definition, $W^j = \sum_{g' \in \mathsf{CG}(g_{j-1}(x_t))} w_t(g') \leq 2M$ because there is only one group in $\mathsf{CG}(g_{j-1}(x_t))$ with weight M (the last accessed group) and there can be at most M groups with weight 1.

We analyze the cost in two cases: If $g = g_j(x_t)$ is the last accessed group, then we have $c_e(\mathcal{T}, \mathcal{W}, x_t) \leq \log(W^j / w_t(g)) \leq \log(2M/M) = O(1)$. Otherwise, if $g_j(x_t)$ is not the last accessed group, then we have $\log(2M) \leq O(\log M)$. ◀

Cost analysis

Consider the search path $\mathcal{T}(x_t)$. Denote by γ_t the number of groups on $\mathcal{T}(x_t)$ that are not the last accessed group of its parents. The cost $c(\mathcal{T}, w_t, x_t) = \sum_{e \in \mathcal{T}(x_t)} c_e(\mathcal{T}, w_t, x_t)$, and from the above lemma, the cost is at most $O(\log M) \cdot \gamma_t + O(h) \leq O(\sqrt{\log n})(\gamma_t + 1)$. Therefore, the total cost is $O(\sqrt{\log n}) \cdot (\sum_t \gamma_t + m)$. The sum of γ_t will be upper bounded by the Wilber bound.

The Wilber bound. Wilber [36] gave two lower bounds on the running time of any BST on a sequence X. These bounds are known as Wilber1 and Wilber2.[1] We now describe Wilber1(X). Let R be a leaf-oriented (keys at the leaves) binary search tree, and for each $a \in [n]$, denote by $R(a)$ the search path in R of key a. When searching a sequence X in R, for each node $v \in V(R)$, the **preferred child** of node v at time t (denoted by Preferred-Child$_t(v)$) is the child of v on the last search path in R at time t. If node v is not on the search path $R(x_t)$, we know that the preferred child cannot change, i.e., Preferred-Child$_t(v)$ = Preferred-Child$_{t-1}(v)$.

The Wilber bound with respect to R at time t and node v is:

$$\text{Wilber1}_R^t(v) = \begin{cases} 1 & \text{if Preferred-Child}_t(v) \neq \text{Preferred-Child}_{t-1}(v) \\ 0 & \text{otherwise} \end{cases}$$

The total Wilber bound of a sequence X is $\text{Wilber1}_R(X) = \sum_t \sum_v \text{Wilber1}_R^t(v)$. The Wilber bound is defined as the maximum, overall reference BST R, of Wilber1$_R(X)$.

▶ **Lemma 22.** *We have that* $\sum_t \gamma_t \leq O(\textit{Wilber1}(X))$

Proof. It is sufficient to define a reference tree R that allows us to charge the cost of $\sum_t \gamma_t$ to Wilber1$_R(X)$. Notice that our group access tree $\mathcal{T}$ is not a binary search tree. However, it can be naturally extended into a binary search tree R as follows: We process the non-leaf nodes in $V(\mathcal{T})$ in a non-decreasing order of distance from the root. When a group $g \in V(\mathcal{T})$ is processed, we remove the edges from g to its children. Let T_g be an arbitrary BST rooted at g and leave $\mathsf{CG}(g)$; we add the tree T_g in place of the deleted edges (See Figure 3). After all vertices are processed, it is straightforward to see that the resulting tree is a (leaf-oriented) BST.

Now we claim that $\sum_t \gamma_t$ can be upper bounded by Wilber1$_R(X)$. Recall that γ_t is the number of groups on the search path $\mathcal{T}(x_t)$ that are not last accessed. Let $G_t \subseteq V(\mathcal{T}(x_t))$ be those groups on the search path that are not last accessed. For each such group $g_1 \in G_t$, let g be its parent, so we know that the last time g was on the search path, some other group $g_2 \in \mathsf{CG}(g)$ was instead chosen. Notice that the search paths $R(g_1)$ and $R(g_2)$ also visit g but branch away at some vertex b inside T_g (it could be that $b = g$). This would imply that Preferred-Child$_t(b) \neq$ Preferred-Child$_{t-1}(b)$ and therefore $\text{Wilber1}_R^t(b) = 1$. This implies that $\gamma_t \leq \sum_v \text{Wilber1}_R^t(v)$ and hence the lemma. ◀

The lemma implies that the total group access bound is at most

$$O(\sqrt{\log n})(\text{Wilber1}(X) + m) \leq O(\sqrt{\log n}) \cdot (\mathsf{OPT}(X)).$$

See the full version of the paper for sections related to the *k-finger bound*, *Unified bound*, and *Unified bound with a time window*.

5 Ggreedy algorithm

This section will define a new BST algorithm named Ggreedy, which satisfies the *group access bound*. The Ggreedy algorithm is very similar to Greedy. Therefore, we first describe the Greedy algorithm and then provide some intuition about the Ggreedy algorithm before describing it formally.

[1] They can also be derived using an elegant geometric language of [14].

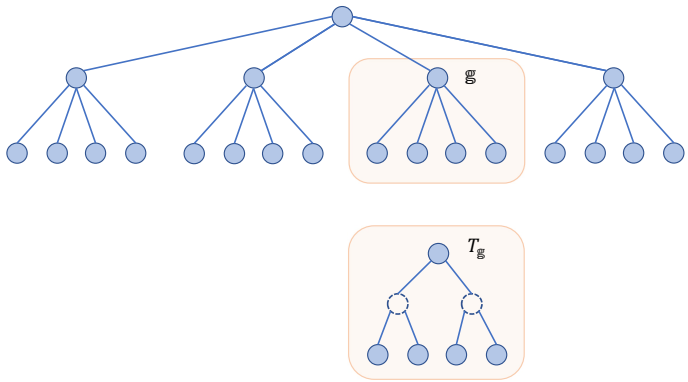


Figure 3 An example of g and T_g.

Given a search sequence $X = \{x_1, x_2, \ldots, x_m\}$, intuitively the GREEDY algorithm works as follows:
At time t, the GREEDY algorithm performs a horizontal line sweep of the points at $y = t$. By induction, we can assume that all the pairs of the points below the line $y = t$ are arborally satisfied. So, at time t, we find all the rectangles with one endpoint x_t and the other endpoint q, where q is a point below the sweep line. If the rectangle $\square x_t q$ is not arborally satisfied, then add a point at the corner of the rectangle $\square x_t q$ on the sweep line to make it arborally satisfied. See Algorithm 2 for the formal definition of GREEDY (Figure 4b illustrates the execution of GREEDY).

Algorithm 1 ADDPOINTS(A,x_t).

```
1 foreach □x_t q in A do
2 |   Add a point at (q.x, x_t.y)
3 end
```

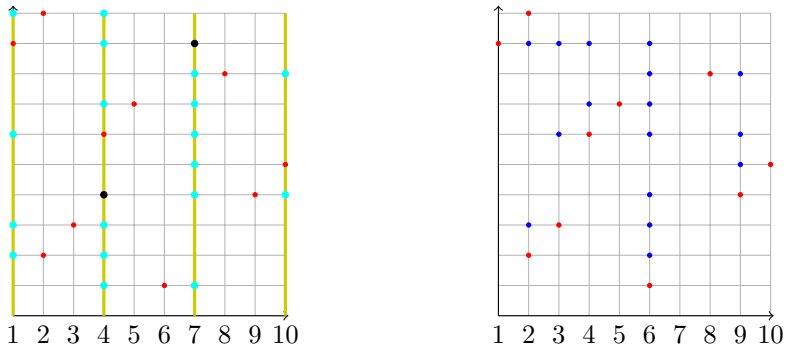
Algorithm 2 Processing GREEDY at time t.

```
1 Let A be the set of all the arborally
    unsatisfied rectangles with one endpoint as
    x_t and another endpoint below the line
    y = t;
2 AddPoints(A, x_t);
```

We will now try to provide some intuition (motivation) behind the GGREEDY algorithm. While analyzing the GREEDY algorithm, we observed that if we can partition the keys into groups (containing consecutive keys) and treat these groups as keys, then we can apply the GREEDY algorithm to this new set of keys. We can then recursively do this within each group, making the analysis easy.

As an example, we can divide the key space into three groups, say from $[1, \ldots, n/3]$, $[n/3+1, \ldots, 2n/3]$ and $[2n/3+1, \ldots, n]$ which we can represent as keys k_1, k_2 and k_3. We can now run the GREEDY algorithm on this set of keys. When a key in the range $[n/3+1, \ldots, 2n/3]$ is searched at time t, we can assume that the key k_2 is searched and perform the GREEDY algorithm accordingly on the keys k_1, k_2 and k_3. We can recursively do this procedure inside each group. To separate the groups and to ensure that the resultant point set is *arborally satisfied*, we add points at the boundaries of the groups.

One can observe that the recursive partition of the keys corresponds to a *group access tree*. An astute reader can see that GGREEDY is not a single algorithm but a family of BST algorithms, which depends on the group access tree.



(a) GGREEDY. **(b)** GREEDY.

Figure 4 GGREEDY(on level 1) vs GREEDY on a sequence $(6, 2, 3, 9, 10, 4, 5, 8, 1, 2)$. There are three groups $\{[1 \dots 4], [4 \dots 7], [7 \dots 10]\}$ in level 1, recursively we can define groups within each group. Red points are the searched keys. (a) Light blue points are added at the boundary of a group $g = g_1(x_t)$. Black points are added to make GGREEDY arborally satisfied with other groups in level 1 (b) Blue points are added to make GREEDY arborally satisfied.

We are now ready to define the GGREEDY algorithm. Pick a group access tree $\mathcal{T}$. Let g be a group at any level in the group access tree. The group g contains a set of consecutive keys at the leaves of $\mathcal{T}$. The group g has two end keys, which we denote as the boundary keys of the group.

▶ **Definition 23** (Boundary of a group)**.** *Let g be a group that contains consecutive keys $(a, a+1, a+2, \dots, b)$, then left$(g) = a$ is called the left boundary key of the group g. Similarly, right$(g) = b$ is called the right boundary key. Together, they are called the boundary keys of the group g.*

In GREEDY, we access a search key x_t at time t, but in GGREEDY, when a key is searched at time t, we access a group per level of the group access tree $\mathcal{T}$. We define the access of a group as follows:

▶ **Definition 24** (Accessed group at time t in level j)**.** *A group g is said to be accessed at time t in level j if $g = g_j(x_t)$.*

▶ **Definition 25** (Accessed boundary key at time t in level j)**.** *The boundary keys of a group are accessed at time t in level j when the group g is accessed at time t in level j. We denote the boundary key access by adding points at the two boundaries of the group g at time t in level j.*

While accessing a key in GREEDY at time t, the algorithm touches keys to form an arborally satisfied set. Similarly, in GGREEDY, when a group is accessed at time t, we might touch other groups. We define a touch group in GGREEDY as follows:

▶ **Definition 26** (Touched group at time t in level j)**.** *A group g is said to be touched at time t in level j if one of the boundary keys of g is touched.*

▶ Remark 27. An accessed group is also a touch group at time t in level j, where both the boundary keys are touched.

Similar to the definition of *unsatisfied rectangle* for GREEDY, we define an *unsatisfied group* in GGREEDY as follows:

▶ **Definition 28** (Unsatisfied group at time t in level j)**.** *A group g_1 is said to be unsatisfied at time t in level j if one of the boundary keys of g_1 forms an unsatisfied rectangle with the boundary keys of $g = g_j(x_t)$ at time t. We denote the unsatisfied group by $\square g_1 g$.*

Let g_1 be a group at time t in level j, which is unsatisfied when the group g is accessed. We touch g_1 at time t in level j to make it an *arborally satisfied* group.

Let us now informally describe the GGREEDY algorithm. When a key x_t is accessed at time t, we access one group per level in the group access tree $\mathcal{T}$. While accessing the group $g = g_j(x_t)$ in level j, we find all the unsatisfied groups in level j which lie inside the group $g_{j-1}(x_t)$ and make them arborally satisfied. We recursively do this for level $j+1$ and so on. The GGREEDY algorithm can be viewed as if we are applying the GREEDY algorithm on groups at each level of the group access tree $\mathcal{T}$. Hence the name, GREEDY on groups or GGREEDY.

We now describe the GGREEDY algorithm formally (See Algorithm 4).

Algorithm 3 Touchgroups(A,g,j).

```
1 Touch g in level j;
2 foreach □g1g in A do
3 |   Touch g1 in level j;
4 end
```

Algorithm 4 Processing of GGREEDY at time t.

```
1 foreach j = 1 to k do
2 |   Let A_j be the set of all arborally
  |     unsatisfied groups in level j when
  |     accessing g = g_j(x_t);
3 |   Touchgroups(A_j, g, j);
4 end
```

In the above algorithm, at iteration j, we first add points at left($g_j(x_t)$) and right($g_j(x_t)$) (same as touching group $g_j(x_t)$). Then, we process all the arborally unsatisfied groups with one endpoint as $g_j(x_t)$ and the other endpoint inside the group $g_{j-1}(x_t)$ (See Figure 4a for the execution of GGREEDY on level 1).

▶ **Observation 29.** *Although the GGREEDY algorithm is very similar to GREEDY, it is not a candidate for dynamic optimality conjecture because of the inherent cost of the groups at each level in the group access tree $\mathcal{T}$.*

We will now try to bound the number of groups touched by GGREEDY in iteration j of Algorithm 4. Let us give it a special notation:

▶ **Definition 30.** *Let $\mathcal{T}_j(t)$ be the set of groups touched by GGREEDY in the j-th iteration of Algorithm 4 at time t.*

Touching a group is the same as touching the group's boundary key(s). Therefore, we consider the boundary keys of the group for the rest of this section to provide better-detailed proofs. We now show some essential properties of $\mathcal{T}_j(t)$.

▶ **Lemma 31.** *$\mathcal{T}_j(t)$ only contains points that are at the boundaries of level j groups that lie inside $g_{j-1}(x_t)$.*

For proof, refer to the full version.

An immediate corollary of the above lemma is:

▶ **Corollary 32.** *Let p be a point that lies strictly inside $g_{j-1}(x_t)$. If GGREEDY adds p at time t then x_t lies in $g_j(p)$.*

We will now show that GGREEDY outputs an arborally satisfied set. Let us assume that we are processing x_t. We can visualize GGREEDY as running GREEDY first on the level 1 group. This execution is the same as GREEDY. Then, we add points at the boundary of $g_1(x_t)$. This is the first step when we deviate from the GREEDY algorithm. Once we have added the boundary points, we again run GREEDY on level 2 groups, and the process continues. One may feel that touching the boundary keys may create some unsatisfied rectangles. But in the following lemma, we show that this is not the case.

▶ **Lemma 33.** *GGREEDY outputs an arborally satisfied set on any input representing X.*

The proof can be found in the full version of the paper.

In the next section, we generalize the *access lemma*, which have been proved for the Splay tree and GREEDY [33, 20]. Let us define a notation before we move to the next section.

▶ **Definition 34.** *Let $\hat{\mathcal{T}}_j(t)$ denote the amortized number of groups touched by GGREEDY in the j-th iteration of Algorithm 4 at time t.*

6 The Group Access Lemma

In this section, we introduce the *group access lemma*, which is a generalization of the *access lemma* and show that the GGREEDY algorithm satisfies it.

Consider the group access tree $\mathcal{T}$. When a BST algorithm $\mathcal{A}$ accesses x_t at time t, after the access, it adjusts itself to $\mathcal{A}'$. Let $g_1 = g_j(x_t)$ be the group that contains x_t. Let Φ_t^j be a potential function that depends on the state of the algorithm with respect to the groups at level j in $\mathcal{T}$. Define $\Phi_t = \sum_j \Phi_t^j$. Define the group access lemma as follows:

▶ **Definition 35** (Group Access Lemma)**.** *A BST algorithm $\mathcal{A}$ satisfies the group access lemma if the amortized cost to access x_t at time t in level j is:*

$$\hat{\mathcal{T}}_j(t) \leq O\left(\log \frac{W^j}{w_{t-1}(g_1)}\right) + \Phi_{t-1}^j - \Phi_t^j$$

The amortized cost of the algorithm $\mathcal{A}$ at time t can be denoted as:

$$\hat{\mathcal{T}}(\mathcal{A}, x_t) = \sum_j \hat{\mathcal{T}}_j(t)$$

and the amortized cost of the algorithm $\mathcal{A}$ on the access sequence X can be defined as:

$$\hat{\mathcal{T}}(\mathcal{A}, X) = \sum_t \hat{\mathcal{T}}(\mathcal{A}, x_t).$$

With the definition of the group access lemma in hand, we will now show that GGREEDY satisfies the group access lemma.

Please refer to the full version of the paper for the proof that GGREEDY satisfies the group access lemma and the simulation theorem.

References

1 Mihai Bădoiu, Richard Cole, Erik D Demaine, and John Iacono. A unified access bound on comparison-based dynamic dictionaries. *Theoretical Computer Science*, 382(2):86–96, 2007.

2 Benjamin Aram Berendsohn, László Kozma, and Michal Opler. Optimization with pattern-avoiding input. *CoRR*, abs/2310.04236, 2023. arXiv:2310.04236.

3 Prosenjit Bose, Karim Douïeb, Vida Dujmović, and Rolf Fagerberg. An o (log log n)-competitive binary search tree with optimal worst-case access times. In *Scandinavian Workshop on Algorithm Theory*, pages 38–49. Springer, 2010.

4 Prosenjit Bose, Karim Douïeb, Vida Dujmović, and John Howat. Layered working-set trees. *Algorithmica*, 63(1-2):476–489, 2012.

5 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Greedy is an almost optimal deque. In *Workshop on Algorithms and Data Structures*, pages 152–165. Springer, 2015.

6 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Pattern-avoiding access in binary search trees. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 410–423. IEEE, 2015.

7 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Self-adjusting binary search trees: What makes them tick? In *Algorithms-ESA 2015*, pages 300–312. Springer, 2015.

8 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Multi-finger binary search trees. *arXiv preprint*, 2018. `arXiv:1809.01759`.

9 Parinya Chalermsook, Manoj Gupta, Wanchote Jiamjitrak, Nidia Obscura Acosta, Akash Pareek, and Sorrachai Yingchareonthawornchai. Improved pattern-avoidance bounds for greedy bsts via matrix decomposition. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 509–534. SIAM, 2023.

10 Parinya Chalermsook, Manoj Gupta, Wanchote Jiamjitrak, Nidia Obscura Acosta, Akash Pareek, and Sorrachai Yingchareonthawornchai. Improved pattern-avoidance bounds for greedy bsts via matrix decomposition. In *SODA*, pages 509–534. SIAM, 2023.

11 Parinya Chalermsook and Wanchote Po Jiamjitrak. New binary search tree bounds via geometric inversions. In *28th Annual European Symposium on Algorithms (ESA 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

12 Parinya Chalermsook, Seth Pettie, and Sorrachai Yingchareonthawornchai. Sorting pattern-avoiding permutations via 0-1 matrices forbidding product patterns. In *SODA*, pages 133–149. SIAM, 2024.

13 Richard Cole. On the dynamic finger conjecture for splay trees. part ii: The proof. *SIAM Journal on Computing*, 30(1):44–85, 2000.

14 Erik D Demaine, Dion Harmon, John Iacono, Daniel Kane, and Mihai Patraşcu. The geometry of binary search trees. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 496–505. SIAM, 2009.

15 Erik D Demaine, Dion Harmon, John Iacono, and Mihai Patraşcu. Dynamic optimality—almost. *SIAM Journal on Computing*, 37(1):240–251, 2007.

16 Erik D Demaine, John Iacono, Stefan Langerman, and Özgür Özkan. Combining binary search trees. In *International Colloquium on Automata, Languages, and Programming*, pages 388–399. Springer, 2013.

17 Jonathan C Derryberry and Daniel D Sleator. Skip-splay: Toward achieving the unified bound in the bst model. In *Workshop on Algorithms and Data Structures*, pages 194–205. Springer, 2009.

18 Jonathan Carlyle Derryberry. *Adaptive binary search trees*. PhD thesis, Carnegie Mellon University, 2009.

19 Amr Elmasry, Arash Farzan, and John Iacono. On the hierarchy of distribution-sensitive properties for data structures. *Acta informatica*, 50(4):289–295, 2013.

20 Kyle Fox. Upper bounds for maximally greedy binary search trees. In *Workshop on Algorithms and Data Structures*, pages 411–422, 2011.

21 George F Georgakopoulos. Chain-splay trees, or, how to achieve and prove loglogn-competitiveness by splaying. *Information Processing Letters*, 106(1):37–43, 2008.

22 Navin Goyal and Manoj Gupta. On dynamic optimality for binary search trees. *arXiv preprint*, 2011. `arXiv:1102.4523`.

23 Navin Goyal and Manoj Gupta. Better analysis of binary search tree on decomposable sequences. *Theoretical Computer Science*, 776:19–42, 2019.

24 John Iacono. Alternatives to splay trees with o (log n) worst-case access times. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 516–522. Society for Industrial and Applied Mathematics, 2001.

25 John Iacono. Key-independent optimality. *Algorithmica*, 42(1):3–10, 2005.

26 John Iacono and Stefan Langerman. Weighted dynamic finger in binary search trees. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 672–691. SIAM, 2016.

27 Elias Koutsoupias and Christos H Papadimitriou. On the k-server conjecture. *Journal of the ACM (JACM)*, 42(5):971–983, 1995.

28 László Kozma and Thatchaphol Saranurak. Smooth heaps and a dual view of self-adjusting data structures. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 801–814, 2018.

29 James R Lee. Fusible hsts and the randomized k-server conjecture. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 438–449. IEEE, 2018.

30 J Ian Munro. On the competitiveness of linear search. In *European Symposium on Algorithms*, pages 338–345. Springer, 2000.

31 Daniel Sleator. Achieving the unified bound in the best model. Talk, 2011.

32 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.

33 Robert Endre Tarjan. Sequential access in splay trees takes linear time. *Combinatorica*, 5(4):367–378, 1985.

34 Chengwen Chris Wang. *Multi-Splay Trees*. PhD thesis, Carnegie Mellon University, USA, 2006. AAI3253576.

35 Chengwen Chris Wang, Jonathan Derryberry, and Daniel Dominic Sleator. O (log log n)-competitive dynamic binary search trees. In *SODA*, volume 6, pages 374–383, 2006.

36 Robert Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM journal on Computing*, 18(1):56–67, 1989.

Optimal Bounds for Distinct Quartics

Panagiotis Charalampopoulos
School of Computing and Mathematical Sciences, Birkbeck, University of London, UK

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland

Samah Ghazawi
Department of Computer Science, University of Haifa, Israel
Department of Software Engineering, Braude, College of Engineering, Karmiel, Israel

Abstract

A fundamental concept related to strings is that of repetitions. It has been extensively studied in many versions, from both purely combinatorial and algorithmic angles. One of the most basic questions is how many distinct squares, i.e., distinct strings of the form UU, a string of length n can contain as fragments. It turns out that this is always $\mathcal{O}(n)$, and the bound cannot be improved to sublinear in n [Fraenkel and Simpson, JCTA 1998].

Several similar questions about repetitions in strings have been considered, and by now we seem to have a good understanding of their repetitive structure. For higher-dimensional strings, the basic concept of periodicity has been successfully extended and applied to design efficient algorithms – it is inherently more complex than for regular strings. Extending the notion of repetitions and understanding the repetitive structure of higher-dimensional strings is however far from complete.

Quartics were introduced by Apostolico and Brimkov [TCS 2000] as analogues of squares in two dimensions. Charalampopoulos, Radoszewski, Rytter, Waleń, and Zuba [ESA 2020] proved that the number of distinct quartics in an $n \times n$ 2D string is $\mathcal{O}(n^2 \log^2 n)$ and that they can be computed in $\mathcal{O}(n^2 \log^2 n)$ time. Gawrychowski, Ghazawi, and Landau [SPIRE 2021] constructed an infinite family of $n \times n$ 2D strings with $\Omega(n^2 \log n)$ distinct quartics. This brings the challenge of determining asymptotically tight bounds. Here, we settle both the combinatorial and the algorithmic aspects of this question: the number of distinct quartics in an $n \times n$ 2D string is $\mathcal{O}(n^2 \log n)$ and they can be computed in the worst-case optimal $\mathcal{O}(n^2 \log n)$ time.

As expected, our solution heavily exploits the periodic structure implied by occurrences of quartics. However, the two-dimensional nature of the problem introduces some technical challenges. Somewhat surprisingly, we overcome the final challenge for the combinatorial bound using a result of Marcus and Tardos [JCTA 2004] for permutation avoidance on matrices.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases 2D strings, quartics, repetitions, periodicity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.39

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2403.06667

1 Introduction

Repetitions are a staple topic of both combinatorics on words [22] and algorithms on strings [33]. In both areas, the classical objects of study are linear sequences of characters from a finite alphabet. Depending on whether we are more interested in their combinatorial properties or designing efficient algorithms for them, it is customary to call such sequences words or strings, respectively. In this paper, we use the latter convention.

Perhaps the most natural example of a repetition in a string is a square, that is, a string of the form UU, also known as a "tandem repeat" in the biological literature [53]. The basic question concerning squares is whether any of the fragments of a string of length n is a

square, and, if so, what is the number of such fragments. The origins of this question can be traced back to Thue [75], who constructed an infinite string over a ternary alphabet that contains no squares. Thus, we can construct arbitrarily long square-free strings over such alphabets. The next question is what is the largest possible number of fragments that are squares. However, any even-length fragment of $\mathtt{a}^n$ is a square. One way to make the question non-trivial is to only consider the primitively rooted squares, meaning that U is not a power of another string. This decreases the possible number of occurrences to $\mathcal{O}(n \log n)$, which is asymptotically tight [30]. Another way is to only consider distinct squares.

Fraenkel and Simpson [46] showed that any string of length n contains at most $2n$ distinct squares and constructed an infinite family of strings such that each string S in this family contains $|S| - \Theta(\sqrt{|S|})$ distinct squares. For many years, it was conjectured that the upper bound should be at most n. After a series of simplifications and improvements [41,57,58,65,74], the conjecture was finally proven by Brlek and Li [24], who showed an upper bound of $n - \sigma + 1$, where σ is the size of the alphabet. The same authors [25] also showed an upper bound of $n - \Theta(\log n)$. On the algorithmic side, Apostolico and Preparata [16], Main and Lorentz [67] and Crochemore [30] showed how to find a compact representation of all squares (in particular, test square-freeness) in a string of length n in $\mathcal{O}(n \log n)$ time. Specifically, such a representation stores all distinct squares. To obtain a faster algorithm for finding only the distinct squares, one needs to restrict the size of the alphabet. For constant alphabets, Gusfield and Stoye [54] designed an $\mathcal{O}(n)$ time algorithm. This was later generalized to the more general case of an integer alphabet (that can be sorted in linear time) [20, 36]. The complexity of testing square-freeness over general ordered and unordered alphabets of size σ was very recently settled by Ellert and Fischer [43] providing a linear time algorithm, and Ellert et al. [44] providing an $\mathcal{O}(n \log \sigma)$ time algorithm, respectively; this problem has been also studied in the parallel [13, 14, 37, 38] and the online settings [55, 63, 64, 66]. Thus, by now we seem to have obtained a rather good understanding of both the combinatorial and the algorithmic properties of distinct squares. We stress that, while these properties are interesting on their own, and naturally the combinatorial bound was used to design efficient enumeration algorithms [21, 54], they were also crucial in designing efficient algorithms and data structures for other problems. For example, the Maximal Augmented Suffix Tree (MAST) introduced by Apostolico and Preparata [17] which enables counting the maximum number of non-overlapping occurrences uses $\mathcal{O}(n)$ space due to the linear upper bound on the number of distinct squares (as observed by Brodal et al. [26]). The same applies to the construction time and the space of the Cover Suffix Tree (CST) [61, 71].

Arguably, linear sequences are not always best suited to model the objects that we would like to study. A natural extension is to consider rectangular arrays of characters from a finite alphabet, which can be seen as 2D strings. Possible applications in image processing [72] sparked interest in designing algorithms for searching in 2D strings already in the late 1970s [18, 23]. This turned out to be significantly more challenging than searching in 1D strings: both versions were studied already in the 70s, but while for 1D strings an alphabet-independent linear-time algorithm had been soon found [60], achieving the same goal for 2D strings took till the 90s [3, 39, 47]. Extensions of this basic problem such as approximate searching [9, 29], indexing [28, 49, 50], searching in smaller space [32], scaled searching [6, 7], searching in random 2D strings [59], dictionary searching [8, 56, 69], and searching in compressed 2D strings [1, 4, 11] have been also considered.

The combinatorial structure of 2D strings seems to be significantly more involved than that of 1D strings. As a prime example, the basic tool used in algorithms and combinatorics on 1D strings is periodicity. We say that p is a period of a 1D string $S[1\,..\,n]$ when $S[i] = S[i+p]$

for all $i = 1, 2, \ldots, n - p$. The set of all periods is very structured due to a classical result of Fine and Wilf [45] according to which for any two periods p, q such that $p + q \leq n$, the greatest common divisor of p and q is also a period. The natural way to extend this notion to 2D strings is to define (x, y) to be a period of a 2D string $S[1\,..\,n][1\,..\,n]$ when $S[i][j] = S[i + x][j + y]$ for all $i = 1, 2, \ldots, n - x$ and $j = 1, 2, \ldots, n - y$. This notion was introduced by Amir and Benson [2], who provided a detailed study based on classifying 2D strings into four periodicity classes. This classification was later crucial in designing solutions for pattern matching, namely, an alphabet-independent linear-time algorithms [39, 47] and alphabet-independent optimal parallel algorithms [5, 31].

The rich combinatorial structure of 2D strings brings the challenge of finding the right generalization of the concept of repetitions. Apostolico and Brimkov [12] introduced two notions of repetitions in 2D strings that can be seen as natural analogues of squares in 1D strings. A tandem $W^{1,2}$ (or $W^{2,1}$) consists of 2 occurrences of the same block W arranged in a 1×2 (or 2×1) pattern. Next, a quartic $W^{2,2}$ consists of 4 occurrences of the same block W arranged in a 2×2 pattern. Note that Apostolico and Brimkov [12] additionally required that W is primitive, meaning that it cannot be partitioned into non-overlapping copies of another block. However, it is more natural to call such tandems and quartics primitively rooted, as in [27]. Apostolico and Brimkov [12] showed asymptotically tight bounds of $\mathcal{O}(n^3 \log n)$ and $\mathcal{O}(n^2 \log^2 n)$ for the number of primitively rooted tandems and quartics, respectively. The former bound was later complemented with a worst-case optimal $\mathcal{O}(n^3 \log n)$-time algorithm [15]. Finally, Amir, Landau, Marcus, and Sokol [10] introduced the notion of maximal repetitions in 2D strings, as an analogue of so-called runs in 1D strings.

Two tandems $T = W^{1,2}$ and $T' = V^{1,2}$ are distinct when $W \neq V$. Similarly, two quartics $Q = W^{2,2}$ and $Q' = V^{2,2}$ are distinct when $W \neq V$. It is easy to see that an $n \times n$ 2D string contains $\mathcal{O}(n^3)$ distinct tandems by applying the bound on the number of 1D distinct squares on every horizontal slice of the 2D string. It is also not hard to show that this bound is asymptotically tight, even over a binary alphabet [48]. Thus, tandems do not seem to be the right generalization of squares, and we should rather focus on quartics.

Recently, Charalampopoulos, Radoszewski, Rytter, Waleń, and Zuba [27] showed a non-trivial upper bound of $\mathcal{O}(n^2 \log^2 n)$ on the number of distinct quartics in an $n \times n$ 2D string, and an algorithm that finds them in the same time complexity. At this point, it was quite unclear to what extent distinct quartics suffer from the "curse of dimensionality". Could it be that, similarly to the number of distinct squares, their number is also linear in the size of the input? Gawrychowski, Ghazawi, and Landau [48] very recently showed that this is not the case, by constructing an infinite family of $n \times n$ 2D strings over a binary alphabet containing $\Omega(n^2 \log n)$ distinct quartics. This shows that there is a qualitative difference between distinct squares and distinct quartics, but leaves a significant gap between the lower bound of $\Omega(n^2 \log n)$ and the upper bound of $\mathcal{O}(n^2 \log^2 n)$ [27].

Our Results. Our contribution is twofold. First, we show an asymptotically tight bound of $\mathcal{O}(n^2 \log n)$ on the number of distinct quartics in an $n \times n$ 2D string. Thus, the "curse of dimensionality" for this problem is a single logarithm for going from 1D to 2D. Second, we show how to find all distinct quartics in worst-case optimal $\mathcal{O}(n^2 \log n)$ time. We thus resolve both the combinatorial and algorithmic complexity of distinct quartics.

A notable difference of our algorithm from the previously fastest algorithm for computing distinct quartics [27] is that the algorithm of [27] first finds all 2D runs[1] of the 2D string, which are not even known to be $\mathcal{O}(n^2 \log n)$, and then infers the quartics from those. We manage to circumvent this, by focusing on some selected occurrences of 2D strings of the form $Q^{5,5}$, instead of considering all of them via 2D runs.

Overview of the Combinatorial Upper Bound. When bounding the number of distinct squares, one begins with fixing the rightmost occurrence of every distinct square [46]. In two dimensions, it is less clear what an extreme occurrence could mean. We simply say that it is an occurrence at a position (i, j) such that there is no other occurrence at a different position (i', j') such that $i' \geq i$ and $j' \geq j$. Next, a standard trick used when working with strings is to partition them into groups with length in $[2^a \,..\, 2^{a+1})$ for different integers a. Similarly to previous work [27], we partition quartics into groups $C_{(a,b)}$ with height in $[2^a \,..\, 2^{a+1})$ and width in $[2^b \,..\, 2^{b+1})$ for pairs of integers (a, b). We begin with proving that, for any position (i, j), the set of extreme occurrences at (i, j) may have a non-empty intersection with only $\mathcal{O}(\log n)$ such groups. Next, we partition all quartics into *thin* and *thick* (note that the meaning of thin and thick is slightly different than in the previous work [27]). More specifically, a quartic Q is thick if and only if it can be partitioned into $x \times y$ occurrences of a primitive 2D string R, i.e., $Q = R^{x,y}$ for some $x, y \geq 5$. Then, we show that for any position (i, j) and group $C_{(a,b)}$, there can be at most 10 extreme occurrences of thin quartics in $C_{(a,b)}$ at position (i, j). Overall, we thus have only $\mathcal{O}(n^2 \log n)$ distinct thin quartics.

The main part of our proof for the combinatorial upper bound is the analysis of the number of distinct thick quartics. Our starting point is the observation (already present in [27]) that this number can be upper bounded by the number of occurrences of 2D strings of the form $R^{5,5}$, for primitive R, that participate in the partition of an extreme occurrence of some quartic $R^{x,y}$. To bound the number of such occurrences, we assign an occurrence of $R^{5,5}$ at position (i, j) to position $(x, y) = (i + 2 \cdot \mathsf{height}(R), j + 2 \cdot \mathsf{width}(R))$ and say that this occurrence is *anchored* at position (x, y). Then, our goal is to show that the number of occurrences assigned to every position is only $\mathcal{O}(\log n)$. For a fixed position (i, j), this is done by first arguing that the pairs $(\lfloor \log(\mathsf{height}(R)) \rfloor, \lfloor \log(\mathsf{width}(R)) \rfloor)$ are pairwise distinct among occurrences of different $R^{5,5}$ assigned to (i, j). This requires a careful analysis of the implied periodic structure and allows us to focus on bounding the number of such pairs. What we do next is the main novelty of our approach for the combinatorial upper bound. We treat the pairs as a set of points $\mathcal{P} \subseteq [1 \,..\, m]^2$, where $m = \lfloor \log n \rfloor$, and argue that, for each $(a, b) \in \mathcal{P}$, the set of points of $\mathcal{P}$ that are strictly dominated by (a, b) can be partitioned into at most two chains. Next, our goal is to upper bound the size of any set $\mathcal{P}$ with this property by $\mathcal{O}(m)$. To this end, we leverage a result from extremal combinatorics, namely, the proof of the Füredi-Hajnal conjecture by Marcus and Tardos [68]. This result states that, if an $m \times m$ binary matrix M avoids a fixed permutation matrix P as a submatrix, i.e., if P cannot be obtained by deleting some rows and columns of M and changing some 1s to 0s, then M contains at most $c_P \cdot m$ 1s, where c_P is a constant if the size of P is a constant. We reformulate the constraint on $\mathcal{P}$ to avoid the permutation matrix shown below as a submatrix. Overall, this allows us to conclude that the number of extreme occurrences of thick quartics is also $\mathcal{O}(n^2 \log n)$.

[1] 2D runs are subarrays that are periodic both vertically and horizontally and cannot be extended without any of the periods changing.

		1	
	1		
1			
			1

A high-level description of our approach for the algorithmic part is provided in Section 4, as it is best read after the full proof of the combinatorial upper bound.

Open Problem. An interesting follow-up question on repetitions in 2D strings is that of settling the number of 2D runs that a 2D string can have. Charalampopoulos et al. [27] proved an $\mathcal{O}(n^2 \log^2 n)$ upper bound for the number of 2D runs that an $n \times n$ 2D string can contain, while Gawrychowski et al. [48] constructed an infinite family of $n \times n$ 2D strings (over a binary alphabet) with $\Omega(n^2 \log n)$ 2D runs. On the algorithms' side, Amir et al. [10] devised an algorithm that computes all 2D runs in an $n \times n$ 2D string in $\mathcal{O}(n^2 \log n + |\mathsf{output}|)$ time, and is thus optimal. For 1D strings, after a long line of results [34, 35, 51, 52, 62, 70, 73] the number of runs was shown to be less than n [19] and they can be computed in $\mathcal{O}(n)$ time for strings over ordered alphabets [43] (see [19, 62] for earlier algorithms for strings over linear-time sortable alphabets).

2 Preliminaries

For integers $i, j \in \mathbb{Z}$, we denote the set $\{k \in \mathbb{Z} : i \leq k \leq j\}$ by either of $[i\,..\,j]$, $(i-1\,..\,j+1)$, $[i\,..\,j+1)$, and $(i-1\,..\,j]$.

Let us consider a string $S = S[1]S[2]\cdots S[n]$ of length $|S| = n$. For integers $i \leq j$ in $[1\,..\,n]$, we denote the *fragment* $S[i]\cdots S[j]$ by $S[i\,..\,j]$. A positive integer $p \leq n$ is a *period* of S if and only if $S[i] = S[i+p]$ for all $i \in [1\,..\,n-p]$. The smallest period of S is called *the period* of S and is denoted by $\mathsf{per}(S)$. A string is called *periodic* if and only if its period is at most half its length. We will extensively use the following property of periods.

▶ **Lemma 2.1** (Periodicity Lemma [45]). *If p and q are periods of a string S and satisfy $p + q \leq |S|$, then $\gcd(p, q)$ is also a period of S.*

We denote the concatenation of two strings U and V by UV. Further, for $k \in \mathbb{Z}_+$, we denote the concatenation of k copies of U by U^k. A string V that cannot be written as U^k for a string U and an integer $k > 1$ is called *primitive*. A string of the form UU is called a *square*. A square UU is said to be primitively rooted if U is primitive. More generally, a string of the form U^k is called a k-th power, and it is said to be primitively rooted if U is primitive. We extensively use the following property of squares.

▶ **Lemma 2.2** (Three Squares Lemma [40][2]). *If squares U^2 and V^2 are proper prefixes of a square W^2, $|U| < |V|$, and U is primitive, then $|U| + |V| \leq |W|$.*

We next summarise some combinatorial properties of squares and higher powers.

▶ **Proposition 2.3.** *Consider a string S and an integer a. At most two prefixes of S with lengths from $[2^a\,..\,2^{a+1})$ can be primitively rooted squares.*

Proof. Assume that there are three such prefixes, and denote them by UU, VV, WW, where $|U| < |V| < |W|$. Since $|UU|, |VV|, |WW| \in [2^a\,..\,2^{a+1})$, we have $|U| + |V| > |W|$, which together with the primitivity of $|U|$ leads to a contradiction. ◀

[2] This formulation comes from [46].

W	W	W	W	W
W	W	W	W	W

Figure 1 2D string $W^{2,5}$ is shown for some 2D string W.

▶ **Proposition 2.4.** *Consider a string S and an integer a. All prefixes of S with lengths in $[2^a\,..\,2^{a+1})$ that are powers higher than 2 are of the form U^k for the same primitive string U.*

Proof. Assume that there are two such prefixes U^k and V^ℓ, where U is primitive, $k, \ell \geq 3$, and $|U| < |V|$. First, $|V|$ is a period of $Z := S[1\,..\,|U| + |V|]$. Second, $|V| < 2^{a+1}/3$ and consequently $|U^{k-1}| \geq |V|$, as otherwise we would have $|U^k| < k \cdot |V|/(k-1) \leq 3|V|/2 < 2^a$, hence $|U|$ is also a period of Z. We thus have that both $|U|$ and $|V|$ are periods of Z. Then, an application of Lemma 2.1 yields that $\gcd(|U|, |V|)$ is a period of Z. But U is primitive so $\gcd(|U|, |V|) = |U|$, and V hence is a power of U, a contradiction. ◀

A fragment $S[i\,..\,j]$ of a string S is a *run* if and only if it is periodic and it cannot be extended by a character in either direction with its period remaining unchanged.

An $m \times n$ 2D string A is simply a two-dimensional array with m rows and n columns, where $\mathsf{height}(A) = m$ and $\mathsf{width}(A) = n$. The position that lies on the i-th row and the j-th column of A is position (i, j). We regard the top-left position of a 2D string as position $(1, 1)$, and the bottom-right position as position (m, n). That is, we index rows from top to bottom and columns from left to right. We say that a 2D string P occurs at a position (i, j) of a 2D string T if and only if the *subarray* (also called *fragment*) $T[i\,..\,i+\mathsf{height}(P))[j\,..\,j+\mathsf{width}(P))$ of T equals P. We write $\Sigma^{*,*}$ to denote the set of all 2D strings over alphabet Σ.

A positive integer p is a *horizontal period* of a 2D string A such that $\mathsf{width}(A) \geq p$ if and only if the j-th column of A is equal to the $(j+p)$-th column of A for all $j \in [1\,..\,\mathsf{width}(A)-p]$. The smallest horizontal period of A is *the horizontal period* of A. An integer q is a (the) *vertical period* of A if and only if q is a (resp. the) horizontal period of the transpose of A.

It will be sometimes convenient to view a 2D string as a 1D metastring by viewing each column (or row) as a metacharacter such that metacharacters are equal if and only if the corresponding columns (resp. rows) are equal. Observe that the horizontal periods (resp. vertical periods) of a 2D string A are in one-to-one correspondence with the periods of the metastring obtained from A by viewing each column (resp. row) as a metacharacter.

For a 2D string W and $x, y \in \mathbb{Z}_+$, we denote by $W^{x,y}$ the 2D string that consists of $x \times y$ copies of W; see Figure 1 for an illustration. A 2D string W is *primitive* if it cannot be written as $Y^{a,b}$ for any 2D string Y and $a, b \in \mathbb{Z}_+$ that are not both equal to 1. The primitive root of a 2D string X is the unique primitive 2D string Y such that $X = Y^{a,b}$ for $a, b \in \mathbb{Z}_+$. Note that the primitive root is indeed unique by the periodicity lemma applied to the horizontal and vertical 1D metastrings obtained from X.

Model of computation. For our algorithm, we assume the standard word-RAM model of computation with word-size $\Omega(\log n)$, where n is the size of the input.

3 The Combinatorial Bound

We consider an $n \times n$ 2D string A, whose entries are over an arbitrary alphabet Σ. We say that a fragment $A[i\,..\,i')[j\,..\,j')$ is a quartic-fragment if and only if it equals some quartic Q; further, we say that it is an *extreme* or *bottom-right* quartic-fragment if Q does not have any occurrence at another position (i'', j'') with $i'' \geq i$ and $j'' \geq j$. We refer to such an

occurrence of Q as an extreme or bottom-right occurrence. We denote by $\mathsf{BR}(i,j)$ the set of extreme quartic-fragments with top-left corner (i,j). Further, we denote the union of all $\mathsf{BR}(i,j)$ by BR. Observe that the distinct quartics in BR are exactly the distinct quartics in A as every quartic that occurs in A has at least one extreme occurrence. Note that a quartic may have $\Theta(n)$ extreme occurrences; an example is provided in Figure 2.

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1
0	0	0	0	0	1	1	1
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	1	1	1	1	1	1

Figure 2 Consider an $n \times n$ 2D string A all of whose entries that lie weakly above the main diagonal are equal to 0 and all of whose entries that lie strictly below the main diagonal are equal to 1. The quartic that equals $0^{2,2}$ has $n-2$ extreme occurrences in A. This is illustrated for $n=8$: the bottom-right corners of extreme occurrences of said quartic are marked.

Let us consider a partition of the quartic-fragments of A into $\mathcal{O}(\log^2 n)$ *canonical sets*, such that, for each $(a,b) \in [1\,..\,\lfloor \log n \rfloor]^2$, the canonical set $C_{(a,b)}$ consists of all quartic-fragments of A whose height is in $[2^a\,..\,2^{a+1})$ and whose width is in $[2^b\,..\,2^{b+1})$.[3]

▶ **Lemma 3.1.** *For each position (i,j) of A, $\mathsf{BR}(i,j)$ has a non-empty intersection with $\mathcal{O}(\log n)$ canonical sets.*

Proof. We say that the *aspect ratio* of a quartic Q is equal to 2 raised to the power $\lfloor \log(\mathsf{height}(Q)) \rfloor - \lfloor \log(\mathsf{width}(Q)) \rfloor$. The aspect ratio of all quartic-fragments in a canonical set $C_{(a,b)}$ is 2^{a-b}. Observe, that there are $2 \cdot \lfloor \log n \rfloor - 1$ different possible values for the aspect ratio of a quartic. For each $d \in [-\lfloor \log n \rfloor + 1\,..\,\lfloor \log n \rfloor - 1]$, let $\mathsf{BR}_d(i,j)$ be the subset of $\mathsf{BR}(i,j)$ that contains exactly the elements of $\mathsf{BR}(i,j)$ with aspect ratio 2^d.

Next, we show that, for each d, we have at most two canonical sets contributing to $\mathsf{BR}_d(i,j)$. Let us suppose towards a contradiction that we have three canonical sets $C_{(a,b)}$, $C_{(a',b')}$, and $C_{(a'',b'')}$ that contribute to $\mathsf{BR}_d(i,j)$. In other words, there are quartic-fragments

- $Q \in \mathsf{BR}_d(i,j)$ with height in $[2^a\,..\,2^{a+1})$ and width in $[2^b\,..\,2^{b+1})$,
- $Q' \in \mathsf{BR}_d(i,j)$ with height in $[2^{a'}\,..\,2^{a'+1})$ and width in $[2^{b'}\,..\,2^{b'+1})$, and
- $Q'' \in \mathsf{BR}_d(i,j)$ with height in $[2^{a''}\,..\,2^{a''+1})$ and width in $[2^{b''}\,..\,2^{b''+1})$.

Since $a-b = a'-b' = a''-b'' = d$, we can assume without loss of generality that $a < a' < a''$ and $b < b' < b''$. We thus have that $a+1 < a''$ and $b+1 < b''$, which implies that Q is fully contained in the top left quarter of Q''. Thus, Q has an occurrence at position $(i + \mathsf{height}(Q'')/2, j + \mathsf{width}(Q'')/2)$; see Figure 3. This contradicts our assumption that the occurrence of Q at position (i,j) is an extreme occurrence.

Thus, $\mathcal{O}(\log n)$ canonical sets contribute to $\mathsf{BR}(i,j)$: at most two for each aspect ratio. ◀

Henceforth, we call a quartic Q with primitive root P *thick* if $Q = P^{x,y}$ for $x, y \geq 5$ and *thin* otherwise.

[3] Throughout this work, logarithms have base 2.

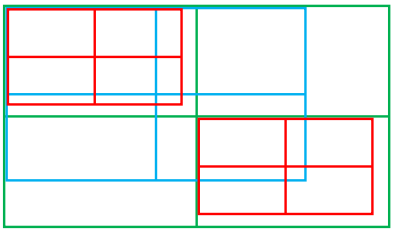


Figure 3 An illustration of the proof of Lemma 3.1 with quartics Q, Q', and Q'' drawn in red, blue, and green, respectively.

▶ **Lemma 3.2.** *For any position (i, j) of A and any pair $(a, b) \in [1 \,..\, \lfloor \log n \rfloor]^2$, $C_{(a,b)} \cap \mathit{BR}(i, j)$ can contain at most 10 thin quartics.*

Proof. The possible forms of *thin* quartics are $P^{2,2}$, $P^{2,2x}$, $P^{2y,2}$, $P^{4,2x}$, and $P^{2y,4}$ for $x > 1$ and $y > 1$. We will consider each form separately.

First, we consider quartics of the form $P^{2,2}$ in $C_{(a,b)} \cap \mathsf{BR}(i, j)$. We analyse the fragment $A[i \,..\, n][j \,..\, j + 2^b)$ and treat it as a metastring by viewing rows as metacharacters. We observe that each considered quartic defines a prefix of this string that is a square. Further, all those squares need to be primitive, as otherwise P could be written as $P = Q^{k,1}$, for some $k > 1$, in contradiction with the primitivity of P. Thus, by Proposition 2.3 we have at most two possible heights for the considered quartics. By a symmetric argument, we have at most two possible widths, and so at most 4 quartics.

Second, we consider quartics of the form $P^{2,2x}$ in $C_{(a,b)} \cap \mathsf{BR}(i, j)$. By the same reasoning as above, we have at most two possible heights for the considered quartics; let h by one of them. We analyse the fragment $A[i \,..\, i + h)[j \,..\, n]$ and treat it as a metastring by viewing columns as metacharacters. Each considered quartic with height h corresponds to a prefix that is a $(2x)$-th power, for some $x > 1$. By Proposition 2.4, all such prefixes are powers of the same U; let U^{2x} be the longest such prefix. Then, for any $x' < x$, the prefix $U^{2x'}$ also occurs at position $(i, j + |U|)$, so the occurrence at position (i, j) cannot be an extreme occurrence. Therefore, for every possible height, we have at most one quartic, so at most 2 in total.

Third, we consider quartics of the form $P^{4,2x}$ for $x > 1$ in $C_{(a,b)} \cap \mathsf{BR}(i, j)$. We (again) analyse the fragment $A[i \,..\, n][j \,..\, j + 2^b)$ and treat it as a metastring by viewing its rows as metacharacters. We observe that each considered quartic defines a prefix that is a primitively rooted fourth power there. Thus, by Proposition 2.4 we have at most one possible height, and, by the same reasoning, as above at most one quartic.

Symmetric arguments bound the number of quartics of the forms $P^{2y,2}$ and $P^{2y,4}$. ◀

▶ **Lemma 3.3.** *The number of distinct thin quartics in A is $\mathcal{O}(n^2 \log n)$.*

Proof. For each position (i, j) of A, $\mathsf{BR}(i, j)$ has a non-empty intersection with at most $\mathcal{O}(\log n)$ canonical sets due to Lemma 3.1. Further, by Lemma 3.2, there are at most 10 thin quartics in each such intersection. Since A has n^2 positions, the stated bound follows. ◀

3.1 Reduction to a Geometric Problem

We next partition the thick quartics by primitive root. For each primitive 2D string R, we choose any bottom-right occurrence of each distinct thick quartic with primitive root R. We denote the obtained set of quartic-fragments by Thick_R. Additionally, let us denote

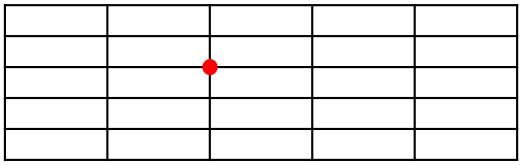


Figure 4 The red point corresponds to the anchor of the shown occurrence of $R^{5,5}$.

by $\mathsf{occ}_{5\times5}(R)$ the set of all positions (i,j) of A where $R^{5,5}$ occurs such that there is an element of Thick_R that fully contains this occurrence of $R^{5,5}$ and has top-left corner equal to $(i - x\cdot\mathsf{height}(R), j - y\cdot\mathsf{width}(R))$ for some non-negative integers x and y.

The proof of the following lemma proceeds almost exactly as the proof of Claim 18 in [27], except that we work with occurrences of $R^{5,5}$ instead of $R^{3,3}$ and do not need the notion of special points. We provide a detailed description for completeness.

▶ **Lemma 3.4** (cf. the proof of [27, Claim 18]). *For any 2D string R, $|\mathit{Thick}_R| \le |\mathit{occ}_{5\times5}(R)|$.*

Proof. We will map each $Q \in \mathsf{Thick}_R$ to an occurrence of $R^{5,5}$ in such a way that two distinct quartic-fragments $Q, Q' \in \mathsf{Thick}_R$ are mapped to distinct occurrences. This will imply that the number of occurrences of R is at least as large as the number of elements of Thick_R.

For each $x = 6, 8, \ldots$ in this order, we select $Q \in \mathsf{Thick}_R$ such that $\mathsf{height}(Q) = x\cdot\mathsf{height}(R)$ and $\mathsf{width}(Q) = y\cdot\mathsf{width}(R)$ is the largest among all $Q' \in \mathsf{Thick}_R$ with $\mathsf{height}(Q') = x\cdot\mathsf{height}(R)$. We note that the number of $Q' \in \mathsf{Thick}_R$ with $\mathsf{height}(Q') = x\cdot\mathsf{height}(R)$ is at most $y/2-2$, and our goal is to map them to occurrences of $R^{5,5}$ that have not been used so far. Additionally, we will ensure that those occurrences are all in the same row. Let (i,j) be the position of an extreme occurrence of Q. We observe that $R^{5,5}$ occurs at every position (i',j') with $i' = i + k\cdot\mathsf{height}(R)$ and $j' = j + \ell\cdot\mathsf{width}(R)$, for every $k \in [0\,..\,x-5]$ and $\ell \in [0\,..\,y-5]$. We choose $k \in [0\,..\,x-5]$ such that none of the occurrences of $R^{5,5}$ at positions $(i + k\cdot\mathsf{height}(R), j + \ell\cdot\mathsf{width}(R))$, for $\ell = 0, 1, \ldots, y-3$ have been used so far. This is possible because so far we have used occurrences of $R^{5,5}$ in only $x/2-3 < x-4$ rows. Then, we map every $Q' \in \mathsf{Thick}_R$ with $\mathsf{height}(Q') = x\cdot\mathsf{height}(R)$ to an occurrence of $R^{5,5}$ at position $(i + k\cdot\mathsf{height}(R), j + \ell\cdot\mathsf{width}(R))$, for some $\ell \in [0\,..\,y-5]$, which is possible due to $y/2-2 \le y-4$. ◀

Thus, it remains to upper bound $\sum_R |\mathsf{occ}_{5\times5}(R)|$, i.e., the sum, over all R, of the number of occurrences of $R^{5,5}$ which are contained in some element of Thick_R.

Consider an occurrence of a 2D string of the form $R^{5,5}$, for a primitive string R, at a position (i,j) of A. We call position $(i + 2\cdot\mathsf{height}(R), j + 2\cdot\mathsf{width}(R))$ the *anchor* of this occurrence; see Figure 4.

Now, for each primitive string R, for each element of $\mathsf{occ}_{5\times5}(R)$, we assign the corresponding occurrence of $R^{5,5}$ to its anchor. Let $\mathsf{assign}(i,j)$ be the set of primitive 2D strings R such that an occurrence of $R^{5,5}$ has been assigned to position (i,j). We have

$$\sum_R |\mathsf{occ}_{5\times5}(R)| = \sum_{i=1}^{n}\sum_{j=1}^{n} |\mathsf{assign}(i,j)|. \tag{1}$$

It now suffices to show that $\sum_{i=1}^{n}\sum_{j=1}^{n} |\mathsf{assign}(i,j)| = \mathcal{O}(n^2\log n)$. We will show that $|\mathsf{assign}(i,j)| = \mathcal{O}(\log n)$ for all i,j, which straightforwardly yields the desired bound.

Let us fix a position (i,j). By applying Proposition 2.4 horizontally and vertically one easily obtains the following fact.

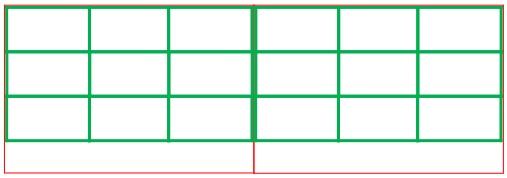


Figure 5 The red rectangles correspond to $T^{1,2}$ and the green rectangles correspond to $S^{3,6}$.

▶ **Fact 3.5** ([27, Corollary 13]). *Let a, b be non-negative integers and W, Z be different 2D strings with height in $[2^a\,..\,2^{a+1})$ and width in $[2^b\,..\,2^{b+1})$. If $W^{3,3}$ and $Z^{3,3}$ occur at some position of A, then at least one of W and Z is not primitive.*

This, together with the fact that, for each $R \in \mathsf{assign}(i,j)$, $R^{3,3}$ occurs at position (i,j), implies the following.

▶ **Fact 3.6.** *For each pair $(a,b) \in [1\,..\,\lfloor \log n \rfloor]^2$, for each position (i,j) of A, the set $\mathsf{assign}(i,j)$ contains at most one element with height in $[2^a\,..\,2^{a+1})$ and width in $[2^b\,..\,2^{b+1})$.*

Let us define a map $\Sigma^{*,*} \to [1\,..\,\lfloor \log n \rfloor]^2$ as $g(R) \mapsto (\lfloor \log(\mathsf{height}(R)) \rfloor, \lfloor \log(\mathsf{width}(R)) \rfloor)$. Let f be the restriction of g to the domain $\mathsf{assign}(i,j)$. Due to Fact 3.6, f is an injective function. We henceforth identify each element R of $\mathsf{assign}(i,j)$ with point $f(R)$. We denote the image of f by $\mathcal{P}$.

We say that a point $(a,b) \in \mathbb{Z}^2$ *dominates* or *weakly dominates* a point (a',b') if $a' \le a$ and $b' \le b$; the dominance is *strict* if $a' < a$ and $b' < b$. (If we require some dominance to be strict we explicitly say so; that is, whenever we refer to some dominance without explicitly mentioning whether it is weak or strict, we refer to weak dominance.) A set of points on which the domination relation forms a total order is called a *chain*. A set of points such that none dominates another is called an *antichain*. We are going to use Dilworth's theorem [42], which states that, in any finite partially ordered set, the size of the largest antichain is equal to the minimum number of chains in which the elements of the set can be decomposed.

For two primitive 2D strings S and T, with $3 \cdot \mathsf{height}(S) < \mathsf{height}(T)$ and $\mathsf{width}(S) < \mathsf{width}(T)$, we say that S *horizontally spans* T when the 2D string $\mathsf{row}_{3\cdot\mathsf{height}(S)}(T^{1,2})$, consisting of the $3 \cdot \mathsf{height}(S)$ topmost rows of $T^{1,2}$, equals $S^{3,y}$ for some even integer $y \ge 4$; see Figure 5. Similarly, when $\mathsf{width}(S) < \mathsf{width}(T)$ and $\mathsf{height}(S) < \mathsf{height}(T)$, we say that S *vertically spans* T when the 2D string $\mathsf{col}_{3\cdot\mathsf{width}(S)}(T^{2,1})$, consisting of the $3 \cdot \mathsf{width}(S)$ leftmost columns of $T^{2,1}$, equals $S^{x,3}$ for some even integer $x \ge 4$.

▶ **Fact 3.7.** *Let S and T be two primitive 2D strings. If S spans T horizontally, then the horizontal period of $\mathsf{row}_{3\cdot\mathsf{height}(S)}(T^{1,2})$ is $\mathsf{width}(S)$. Symmetrically, if S spans T vertically, then the vertical period of $\mathsf{col}_{3\cdot\mathsf{width}(S)}(T^{2,1})$ is $\mathsf{height}(S)$.*

Proof. We only prove the first statement as the second one follows by symmetry. Let us view $\mathsf{row}_{3\cdot\mathsf{height}(S)}(T^{1,2})$ as a metastring Z by viewing each of its columns as a metacharacter; the horizontal period of $\mathsf{row}_{3\cdot\mathsf{height}(S)}(T^{1,2})$ equals $p := \mathsf{per}(Z)$. Note that $\mathsf{width}(S)$ is a period of Z. Towards a contradiction, suppose that $p < \mathsf{width}(S)$. Then, an application of the periodicity lemma to Z implies that p must divide $\mathsf{width}(S)$. This fact contradicts the primitivity of S, as we would have that $S = (S[1\,..\,\mathsf{height}(S)][1\,..\,p])^{1,k}$ for $k = \mathsf{width}(S)/p$; see Figure 5. ◀

When reading the following lemma, one can think of M being in $\mathsf{assign}(i,j)$. However, the lemma is slightly more general, as needed for the algorithm that is presented in the full version.

▶ **Lemma 3.8.** *Let $R \in \mathsf{assign}(i,j)$ and M be a primitive 2D string such that:*
- *$M^{5,5}$ has an occurrence with anchor (i,j);*
- *$g(M)$ strictly dominates $g(R)$.*

Then, R spans M either horizontally or vertically (or both).

Proof. By the definition of $\mathsf{assign}(i,j)$, the occurrence of $R^{5,5}$ assigned to position (i,j) appears inside an element of Thick_R, which is necessarily a bottom-right occurrence of a thick quartic with primitive root R. Let us denote this quartic by Q. Observe that the considered occurrence of Q cannot be fully contained inside the occurrence of $M^{4,4}$ at position $(i-2\cdot\mathsf{height}(M), j-2\cdot\mathsf{width}(M))$ as this would contradict the fact that the considered occurrence of Q is bottom-right: there would be another occurrence $\mathsf{width}(M)$ positions to the right. Therefore, Q must contain at least one of the following four fragments of A, depicted in Figure 6: $A[i\,..\,i{+}3{\cdot}\mathsf{height}(R))[j\,..\,j{+}2{\cdot}\mathsf{width}(M))$, $A[i\,..\,i{+}3{\cdot}\mathsf{height}(R))[j{-}2{\cdot}\mathsf{width}(M)\,..\,j)$, $A[i\,..\,i+2\cdot\mathsf{height}(M))[j\,..\,j+3\cdot\mathsf{width}(R))$, $A[i-2\cdot\mathsf{height}(M)\,..\,i)[j\,..\,j+3\cdot\mathsf{width}(R))$.

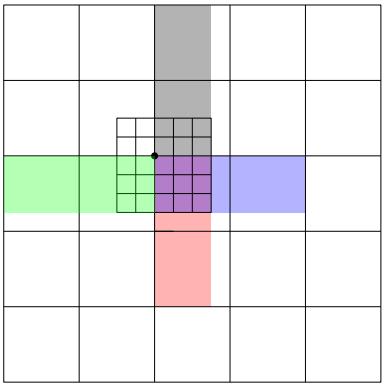


Figure 6 The considered occurrences of each of $M^{5,5}$ and $R^{5,5}$ are shown, together with the four specified fragments, at least one of which must be fully contained in Q.

We next show that in either of the first two cases, R horizontally spans M. In the remaining cases, a symmetric argument yields that R vertically spans M. First, observe that we have $\mathsf{width}(R) < \mathsf{width}(M)$ as a direct consequence of $g(M)$ strictly dominating $g(R)$. We next need to argue that $3\mathsf{height}(R) < \mathsf{height}(M)$. If this were not the case, $\mathsf{width}(R)$ would be a horizontal period of $M^{1,2}$, contradicting the primitivity of M. This completes the proof. ◀

▶ **Lemma 3.9.** *Two primitive 2D strings R_1 and R_2 such that $g(R_1)$ and $g(R_2)$ form an antichain cannot both horizontally span a 2D string R.*

Proof. Let $g(R_1) = (a_1, b_1)$ and $g(R_2) = (a_2, b_2)$. Without loss of generality, we can assume that $a_1 < a_2$ and $b_1 > b_2$. Suppose towards a contradiction that both R_1 and R_2 horizontally span R. By applying Fact 3.7, we obtain that
- $\mathsf{width}(R_1)$ is the horizontal period of $\mathsf{row}_{3\cdot\mathsf{height}(R_1)}(R^{1,2})$ and
- $\mathsf{width}(R_2)$ is the horizontal period of $\mathsf{row}_{3\cdot\mathsf{height}(R_2)}(R^{1,2})$.

Note that, for any $k \in [1\,..\,\mathsf{height}(R)]$, the horizontal period of the string comprised of the k topmost rows of $R^{1,2}$ equals the least common multiple of the periods of those k rows. Hence, the period cannot decrease as we increase the number of considered rows. We thus have $\mathsf{width}(R_1) \le \mathsf{width}(R_2)$ since our assumption that $a_2 > a_1$ implies that $\mathsf{height}(R_2) > \mathsf{height}(R_1)$. This is a contradiction to our assumption that $b_1 > b_2$, which implies that $\mathsf{width}(R_1) > \mathsf{width}(R_2)$. ◀

The above lemma, Fact 3.6, and Dilworth's theorem together imply the following.

▶ **Corollary 3.10.** *All primitive 2D strings that span a primitive 2D string R can be decomposed to two sets H and V, such that*
- *the elements of H span R horizontally;*
- *the elements of V span R vertically;*
- *the restriction of g to $H \cup V$ is an injective function;*
- *each of the sets $g(H)$ and $g(V)$ is a chain.*

Finally, as mentioned in the introduction, we need the following purely geometric lemma that follows from the result of Marcus and Tardos [68] on the number of 1s in an $m \times m$ binary matrix M that avoids a fixed permutation P as a submatrix.

▶ **Lemma 3.11.** *Consider a positive integer m and a set $\mathcal{P} \subseteq [1\,..\,m]^2$. If, for each $p \in \mathcal{P}$, the set of points of $\mathcal{P}$ that are strictly dominated by p can be partitioned into at most two chains, then $|\mathcal{P}| = \mathcal{O}(m)$.*

Proof. We think of $\mathcal{P}$ as an $m \times m$ matrix $M[1\,..\,m][1\,..\,m]$, where $M[a][b] = 1$ when $(a, b) \in \mathcal{P}$ and $M[a, b] = 0$ otherwise. Next, we say that M contains a matrix P as a submatrix when P can be obtained from M by removing rows, removing columns, and changing 1s into 0s. We claim that, by the assumptions in the lemma, M does not contain the following matrix P as a submatrix:

		1	
	1		
1			
			1

To establish this, assume otherwise towards a contradiction. Then, there exists $(a, b) \in \mathcal{P}$ and $(a_1, b_1), (a_2, b_2), (a_3, b_3) \in \mathcal{P}$ such that (a, b) strictly dominates $(a_1, b_1), (a_2, b_2), (a_3, b_3)$ and further $(a_1, b_1), (a_2, b_2), (a_3, b_3)$ create an antichain. By Dilworth's theorem, this implies that the points in $\mathcal{P}$ dominated by (a, b) cannot be partitioned into two chains, a contradiction. Thus, M indeed does not contain P as a submatrix. Because P is a permutation matrix, this implies $|\mathcal{P}| = \mathcal{O}(m)$. ◀

We now complete the proof of our main result with the aid of Lemma 3.11.

▶ **Theorem 3.12.** *An $n \times n$ 2D string has $\mathcal{O}(n^2 \log n)$ distinct quartics.*

Proof. The number of distinct thin quartics is $\mathcal{O}(n^2 \log n)$ by Lemmas 3.2 and 3.3. The number of distinct thick quartics is

$$\sum_R |\mathsf{Thick}_R| \le \sum_R |\mathsf{occ}_{5\times 5}(R)| \qquad \text{(Lemma 3.4)}$$

$$\le \sum_{i=1}^{n} \sum_{j=1}^{n} |\mathsf{assign}(i, j)|. \qquad (1)$$

To conclude the proof, it remains to show that $|\mathsf{assign}(i, j)| = \mathcal{O}(\log n)$ for all $(i, j) \in [1\,..\,n]^2$. Let $m = \lfloor \log n \rfloor$, and recall that $\mathcal{P} \subseteq [1\,..\,m]^2$ was defined as the image of f, which in turn was the restriction of g to the domain $\mathsf{assign}(i, j)$. By Fact 3.6, we only need to show that $|\mathcal{P}| = \mathcal{O}(m)$. By Lemma 3.8 and Corollary 3.10, for each $p \in \mathcal{P}$, the set of all points of $\mathcal{P}$ that are strictly dominated by p can be partitioned into at most two chains. Thus, by Lemma 3.11 we conclude that indeed $|\mathcal{P}| = \mathcal{O}(m)$, concluding the proof. ◀

4 Overview of the Algorithm

The detailed algorithm can be found in the full version of this work. After preprocessing the input 2D string, we compute thin and thick quartics separately. Here, we provide an overview of the main ideas of our $\mathcal{O}(n^2 \log n)$-time algorithm for computing distinct quartics in an $n \times n$ 2D string A over an ordered alphabet.

Computation of Thin Quartics

For thin quartics, our algorithm is quite similar to the combinatorial analysis. For each position (i, j), we compute an $\mathcal{O}(\log n)$-size superset $\mathcal{C}$ of the canonical sets that have a non-empty intersection with $\mathsf{BR}(i, j)$. We do this by relating extreme occurrences of quartics with occurrences of squares in metastrings obtained by viewing the columns of $A[i \,..\, i + 2^a)[1 \,..\, n]$, where $a \in [1 \,..\, \lfloor \log n \rfloor]$, as metacharacters. These squares can be efficiently computed and give us a handle on the sought thin quartics. Then, we compute the intersection of each canonical set in $\mathcal{C}$ with $\mathsf{BR}(i, j)$ in constant time using known tools that allow us to efficiently operate on the metastrings. We do this by fixing $(a, b) \in [1 \,..\, \lfloor \log n \rfloor]^2$ and computing all quartics with height in $[2^a \,..\, 2^{a+1})$ and width $[2^b \,..\, 2^{b+1})$ that occur at position (i, j) of A, of the form $P^{2y,2}$, $P^{2y,4}$, $P^{2,2x}$, and $P^{4,2x}$, for a primitive 2D string P and $x, y \geq 1$.

Computation of Thick Quartics

For thick quartics, our algorithmic approach follows our combinatorial approach in a more relaxed sense. The main technical challenge is to compute, for each position (i, j), an $\mathcal{O}(\log n)$-size set $\mathcal{R}$ of primitive 2D strings R, such that $\mathsf{assign}(i, j) \subseteq \mathcal{R}$. Then, those supersets can be postprocessed as in [27] in time linear in their total size to yield the sought distinct thick quartics. The computation of $\mathcal{R}$ is split into two major steps outlined next.

Skyline Computation. First, we compute a set $\mathcal{S}$ of *skyline primitive 2D strings* such that $S \in \mathcal{S}$ when (a) $S^{5,5}$ has an occurrence anchored at position (i, j), and (b) there is no other primitive 2D string T with $g(T) \geq g(S)$ such that $T^{5,5}$ has an occurrence anchored at position (i, j). This part of the proof is quite technical: it heavily relies on the analysis of periodicity for 1D (meta)strings and, roughly speaking, on the analysis of the evolution of the horizontal periodic structure of a 2D string as rows are appended to it. We show that, for each $a \in [1 \,..\, \lfloor \log n \rfloor]$, there is a single candidate $h \in [2^a \,..\, 2^{a+1})$ to be considered as the height of an element of $\mathcal{S}$. Then, using runs in 1D metastrings whose origins in A have sufficient overlap and bit-tricks, we can compute the widest 2D string S with height h such that an occurrence of $S^{5,5}$ is anchored at position (i, j) in constant time (using batched computations), if one exists.

Computation of Dominated 2D strings. This turned out to be the most challenging part of our approach. For this exposition, let us treat $\Sigma^{*,*}$ as a partially ordered set, in the order of decreasing widths. Let $\mathcal{S} = \{S_1, \ldots, S_\ell\}$ in accordance with this order. To obtain $\mathcal{R}$ from $\mathcal{S}$, we need to add to it the 2D strings $R \in \mathsf{assign}(i, j) \setminus \mathcal{S}$. By the construction of $\mathcal{S}$ we know that there exists $S \in \mathcal{S}$ such that $g(R) \leq g(S)$.

Our combinatorial analysis implies that each $R \in \mathsf{assign}(i, j)$ spans each element $S \in \mathcal{S}$ for which $g(S)$ strictly dominates $g(R)$ either vertically or horizontally. It turns out that if R spans all of these elements of $\mathcal{S}$ either vertically or horizontally, it is easy to compute it efficiently. This is, unfortunately, not the case in general. However, we observe that R spans

vertically (resp. horizontally) a contiguous subset of $\mathcal{S}$. We show that the problem boils down to computing the union, over all k, of sets I_k, where I_k contains exactly those primitive 2D strings that span S_{k-1} vertically and span S_k horizontally. Crucially, we observe that due to a strong form of transitivity of the spanning property, the intersection of any two such I_k and $I_{k'}$ consists of a number of the smallest elements of both (i.e., their longest common prefix if viewed as strings). Hence, by computing the elements of I_k from the largest to the smallest, we can stop whenever we encounter an element that has already been reported by this procedure. This allows us to reduce the computation of $\mathcal{R}$ to the problem of efficiently computing sets I_k. To this end, we prove that an $\mathcal{O}(\log n)$-bits representation of the evolution of the periodic structure of certain fragments of A as rows and columns are appended to them suffices for inferring I_k; we then use tabulation to infer it efficiently.

References

1 Amihood Amir and Gary Benson. Efficient two-dimensional compressed matching. In *Data Compression Conference*, pages 279–288. IEEE Computer Society, 1992. doi:10.1109/DCC.1992.227453.

2 Amihood Amir and Gary Benson. Two-dimensional periodicity in rectangular arrays. *SIAM J. Comput.*, 27(1):90–106, 1998. doi:10.1137/S0097539795298321.

3 Amihood Amir, Gary Benson, and Martin Farach. An alphabet independent approach to two-dimensional pattern matching. *SIAM Journal on Computing*, 23(2):313–323, 1994. doi:10.1137/S0097539792226321.

4 Amihood Amir, Gary Benson, and Martin Farach. Optimal two-dimensional compressed matching. *J. Algorithms*, 24(2):354–379, 1997. doi:10.1006/JAGM.1997.0860.

5 Amihood Amir, Gary Benson, and Martin Farach. Optimal parallel two dimensional text searching on a CREW PRAM. *Inf. Comput.*, 144(1):1–17, 1998. doi:10.1006/INCO.1998.2705.

6 Amihood Amir, Ayelet Butman, Moshe Lewenstein, and Ely Porat. Real two dimensional scaled matching. *Algorithmica*, 53(3):314–336, 2009. doi:10.1007/S00453-007-9021-X.

7 Amihood Amir and Eran Chencinski. Faster two dimensional scaled matching. *Algorithmica*, 56(2):214–234, 2010. doi:10.1007/S00453-008-9173-3.

8 Amihood Amir and Martin Farach. Two-dimensional dictionary matching. *Inf. Process. Lett.*, 44(5):233–239, 1992. doi:10.1016/0020-0190(92)90206-B.

9 Amihood Amir and Martin Farach. Efficient 2-dimensional approximate matching of half-rectangular figures. *Inf. Comput.*, 118(1):1–11, 1995. doi:10.1006/INCO.1995.1047.

10 Amihood Amir, Gad M. Landau, Shoshana Marcus, and Dina Sokol. Two-dimensional maximal repetitions. *Theoretical Computer Science*, 812:49–61, 2020. doi:10.1016/j.tcs.2019.07.006.

11 Amihood Amir, Gad M. Landau, and Dina Sokol. Inplace 2d matching in compressed images. *J. Algorithms*, 49(2):240–261, 2003. doi:10.1016/S0196-6774(03)00088-9.

12 A. Apostolico and V.E. Brimkov. Fibonacci arrays and their two-dimensional repetitions. *Theoretical Computer Science*, 237(1-2):263–273, 2000. doi:10.1016/S0304-3975(98)00182-0.

13 Alberto Apostolico. Optimal parallel detection of squares in strings. *Algorithmica*, 8(4):285–319, 1992. doi:10.1007/BF01758848.

14 Alberto Apostolico and Dany Breslauer. An optimal $\mathcal{O}(\log \log n)$-time parallel algorithm for detecting all squares in a string. *SIAM J. Comput.*, 25(6):1318–1331, 1996. doi:10.1137/S0097539793260404.

15 Alberto Apostolico and Valentin E. Brimkov. Optimal discovery of repetitions in 2D. *Discrete Applied Mathematics*, 151(1-3):5–20, 2005. doi:10.1016/j.dam.2005.02.019.

16 Alberto Apostolico and Franco P. Preparata. Optimal off-line detection of repetitions in a string. *Theor. Comput. Sci.*, 22:297–315, 1983. doi:10.1016/0304-3975(83)90109-3.

17 Alberto Apostolico and Franco P. Preparata. Data structures and algorithms for the string statistics problem. *Algorithmica*, 15(5):481–494, 1996.

18 Theodore P. Baker. A technique for extending rapid exact-match string matching to arrays of more than one dimension. *SIAM Journal on Computing*, 7(4):533–541, 1978. doi:10.1137/0207043.

19 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.

20 Hideo Bannai, Shunsuke Inenaga, and Dominik Köppl. Computing all distinct squares in linear time for integer alphabets. In *CPM*, pages 22:1–22:18, 2017. doi:10.4230/LIPICS.CPM.2017.22.

21 Hideo Bannai, Shunsuke Inenaga, and Dominik Köppl. Computing all distinct squares in linear time for integer alphabets. In *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, volume 78 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.22.

22 Jean Berstel and Dominique Perrin. The origins of combinatorics on words. *Eur. J. Comb.*, 28(3):996–1022, 2007. doi:10.1016/J.EJC.2005.07.019.

23 Richard S. Bird. Two dimensional pattern matching. *Inf. Process. Lett.*, 6(5):168–170, 1977. doi:10.1016/0020-0190(77)90017-5.

24 S. Brlek and S. Li. On the number of squares in a finite word. *arXiv*, 2022. arXiv:2204.10204.

25 Srecko Brlek and Shuo Li. On the number of distinct squares in finite sequences: Some old and new results. In *Combinatorics on Words – 14th International Conference, WORDS 2023*, pages 35–44, 2023. doi:10.1007/978-3-031-33180-0_3.

26 Gerth Stølting Brodal, Rune B. Lyngsø, Anna Östlin, and Christian N. S. Pedersen. Solving the string statistics problem in time o(n log n). In *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 728–739. Springer, 2002.

27 P. Charalampopoulos, J. Radoszewski, W. Rytter, T. Waleń, and W. Zuba. The number of repetitions in 2D-strings. In *28th Annual European Symposium on Algorithms, ESA 2020*, pages 1–18, 2020. doi:10.4230/LIPICS.ESA.2020.32.

28 Ying Choi and Tak Wah Lam. Dynamic suffix tree and two-dimensional texts management. *Inf. Process. Lett.*, 61(4):213–220, 1997. doi:10.1016/S0020-0190(97)00018-5.

29 Raphaël Clifford, Allyx Fontaine, Tatiana Starikovskaya, and Hjalte Wedel Vildhøj. Dynamic and approximate pattern matching in 2D. In *SPIRE*, pages 133–144, 2016. doi:10.1007/978-3-319-46049-9_13.

30 Maxime Crochemore. An optimal algorithm for computing the repetitions in a word. *Information Processing Letters*, 12(5):244–250, 1981.

31 Maxime Crochemore, Leszek Gasieniec, Ramesh Hariharan, S. Muthukrishnan, and Wojciech Rytter. A constant time optimal parallel algorithm for two-dimensional pattern matching. *SIAM J. Comput.*, 27(3):668–681, 1998. doi:10.1137/S0097539795280068.

32 Maxime Crochemore, Leszek Gasieniec, Wojciech Plandowski, and Wojciech Rytter. Two-dimensional pattern matching in linear time and small space. In *STACS 95, 12th Annual Symposium on Theoretical Aspects of Computer Science*, pages 181–192, 1995. doi:10.1007/3-540-59042-0_72.

33 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.

34 Maxime Crochemore and Lucian Ilie. Maximal repetitions in strings. *J. Comput. Syst. Sci.*, 74(5):796–807, 2008. doi:10.1016/j.jcss.2007.09.003.

35 Maxime Crochemore, Lucian Ilie, and Liviu Tinta. The "runs" conjecture. *Theor. Comput. Sci.*, 412(27):2931–2941, 2011. doi:10.1016/j.tcs.2010.06.019.

36 Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Extracting powers and periods in a word from its runs structure. *Theor. Comput. Sci.*, 521:29–41, 2014. doi:10.1016/J.TCS.2013.11.018.

37 Maxime Crochemore and Wojciech Rytter. Efficient parallel algorithms to test square-freeness and factorize strings. *Inf. Process. Lett.*, 38(2):57–60, 1991. doi:10.1016/0020-0190(91)90223-5.

38 Maxime Crochemore and Wojciech Rytter. Usefulness of the Karp-Miller-Rosenberg algorithm in parallel computations on strings and arrays. *Theor. Comput. Sci.*, 88(1):59–82, 1991. doi:10.1016/0304-3975(91)90073-B.

39 Maxime Crochemore and Wojciech Rytter. On linear-time alphabet-independent 2-dimensional pattern matching. In *LATIN '95: Theoretical Informatics*, pages 220–229, 1995. doi:10.1007/3-540-59175-3_91.

40 Maxime Crochemore and Wojciech Rytter. Squares, cubes, and time-space efficient string searching. *Algorithmica*, 13(5):405–425, 1995. doi:10.1007/BF01190846.

41 A. Deza, F. Franek, and A. Thierry. How many double squares can a string contain? *Discrete Applied Mathematics*, 180:52–69, 2015. doi:10.1016/J.DAM.2014.08.016.

42 R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950. URL: http://www.jstor.org/stable/1969503.

43 Jonas Ellert and Johannes Fischer. Linear Time Runs Over General Ordered Alphabets. *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, pages 63:1–63:16, 2021. doi:10.4230/LIPICS.ICALP.2021.63.

44 Jonas Ellert, Pawel Gawrychowski, and Garance Gourdel. Optimal square detection over general alphabets. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 5220–5242. SIAM, 2023. doi:10.1137/1.9781611977554.CH189.

45 Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. doi:10.2307/2034009.

46 Aviezri S. Fraenkel and Jamie Simpson. How many squares can a string contain? *Journal of Combinatorial Theory, Series A*, 82(1):112–120, 1998. doi:10.1006/jcta.1997.2843.

47 Zvi Galil and Kunsoo Park. Alphabet-independent two-dimensional witness computation. *SIAM J. Comput.*, 25(5):907–935, 1996. doi:10.1137/S0097539792241941.

48 P. Gawrychowski, S. Ghazawi, and Gad M. Landau. Lower bounds for the number of repetitions in 2D strings. In *SPIRE 2021*, pages 179–192, 2021. doi:10.1007/978-3-030-86692-1_15.

49 Raffaele Giancarlo. A generalization of the suffix tree to square matrices, with applications. *SIAM J. Comput.*, 24(3):520–562, 1995. doi:10.1137/S0097539792231982.

50 Raffaele Giancarlo and Roberto Grossi. On the construction of classes of suffix trees for square matrices: Algorithms and applications. *Inf. Comput.*, 130(2):151–182, 1996. doi:10.1006/INCO.1996.0087.

51 Mathieu Giraud. Not so many runs in strings. In *Language and Automata Theory and Applications, Second International Conference, LATA 2008*, volume 5196, pages 232–239. Springer, 2008. doi:10.1007/978-3-540-88282-4_22.

52 Mathieu Giraud. Asymptotic behavior of the numbers of runs and microruns. *Inf. Comput.*, 207(11):1221–1228, 2009. doi:10.1016/j.ic.2009.02.007.

53 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences – Computer Science and Computational Biology.* Cambridge University Press, 1997.

54 Dan Gusfield and Jens Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *J. Comput. Syst. Sci.*, 69(4):525–546, 2004. doi:10.1016/J.JCSS.2004.03.004.

55 Jin-Ju Hong and Gen-Huey Chen. Efficient on-line repetition detection. *Theor. Comput. Sci.*, 407(1-3):554–563, 2008. doi:10.1016/j.tcs.2008.08.038.

56 Ramana M. Idury and Alejandro A. Schäffer. Multiple matching of rectangular patterns. *Inf. Comput.*, 117(1):78–90, 1995. doi:10.1006/INCO.1995.1030.

57 L. Ilie. A simple proof that a word of length n has at most $2n$ distinct squares. *Journal of Combinatorial Theory, Series A*, 112(1):163–164, 2005. doi:10.1016/J.JCTA.2005.01.006.

58 L. Ilie. A note on the number of squares in a word. *Theoretical Computer Science*, 380(3):373–376, 2007. doi:10.1016/J.TCS.2007.03.025.

59 Juha Kärkkäinen and Esko Ukkonen. Two- and higher-dimensional pattern matching in optimal expected time. *SIAM J. Comput.*, 29(2):571–589, 1999. doi:10.1137/S0097539794275872.

60 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977. doi:10.1137/0206024.

61 Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Fast algorithm for partial covers in words. *Algorithmica*, 73(1):217–233, 2015.

62 Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *40th Annual Symposium on Foundations of Computer Science, FOCS 1999*, pages 596–604. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814634.

63 Dmitry Kosolobov. Online square detection. *CoRR*, abs/1411.2022, 2014. arXiv:1411.2022.

64 Dmitry Kosolobov. Online detection of repetitions with backtracking. In *Combinatorial Pattern Matching – 26th Annual Symposium, CPM 2015*, volume 9133, pages 295–306. Springer, 2015. doi:10.1007/978-3-319-19929-0_25.

65 N. H. Lam. On the number of squares in a string. *AdvOL-Report 2*, 2013.

66 Ho-fung Leung, Zeshan Peng, and Hing-Fung Ting. An efficient algorithm for online square detection. *Theor. Comput. Sci.*, 363(1):69–75, 2006. doi:10.1016/J.TCS.2006.06.011.

67 Michael G. Main and Richard J. Lorentz. An $\mathcal{O}(n \log n)$ algorithm for finding all repetitions in a string. *J. Algorithms*, 5(3):422–432, 1984. doi:10.1016/0196-6774(84)90021-X.

68 Adam Marcus and Gábor Tardos. Excluded permutation matrices and the stanley-wilf conjecture. *J. Comb. Theory, Ser. A*, 107(1):153–160, 2004. doi:10.1016/J.JCTA.2004.04.002.

69 Shoshana Neuburger and Dina Sokol. Succinct 2D dictionary matching. *Algorithmica*, 65(3):662–684, 2013. doi:10.1007/S00453-012-9615-9.

70 Simon J. Puglisi, Jamie Simpson, and William F. Smyth. How many runs can a string contain? *Theor. Comput. Sci.*, 401(1-3):165–171, 2008. doi:10.1016/J.TCS.2008.04.020.

71 Jakub Radoszewski. Linear time construction of cover suffix tree and applications. In *ESA*, volume 274 of *LIPIcs*, pages 89:1–89:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

72 Azriel Rosenfeld and Avinash C. Kak. *Digital Picture Processing: Volume 1 and 2*. Computer Science and Applied Mathematics. Academic Press, Orlando, FL, 2 edition, 1982.

73 Wojciech Rytter. The number of runs in a string: Improved analysis of the linear upper bound. In *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science*, pages 184–195, 2006. doi:10.1007/11672142_14.

74 A. Thierry. A proof that a word of length n has less than $1.5n$ distinct squares. *arXiv*, 2020. arXiv:2001.02996.

75 A. Thue. Über unendliche Zeichenreihen. *Norske Vid. Selsk. Skr., I Mat.–Nat. Kl., Christiania*, 7:1–22, 1906.

Streaming Edge Coloring with Subquadratic Palette Size

Shiri Chechik
Tel Aviv University, Israel

Doron Mukhtar
Tel Aviv University, Israel

Tianyi Zhang
Tel Aviv University, Israel

Abstract

In this paper, we study the problem of computing an edge-coloring in the (one-pass) W-streaming model. In this setting, the edges of an n-node graph arrive in an arbitrary order to a machine with a relatively small space, and the goal is to design an algorithm that outputs, as a stream, a proper coloring of the edges using the fewest possible number of colors.

Behnezhad et al. [Behnezhad et al., 2019] devised the first non-trivial algorithm for this problem, which computes in $\tilde{O}(n)$ space a proper $O(\Delta^2)$-coloring w.h.p. (here Δ is the maximum degree of the graph). Subsequent papers improved upon this result, where latest of them [Ansari et al., 2022] showed that it is possible to deterministically compute an $O(\Delta^2/s)$-coloring in $O(ns)$ space. However, none of the improvements succeeded in reducing the number of colors to $O(\Delta^{2-\epsilon})$ while keeping the same space bound of $\tilde{O}(n)$[1]. In particular, no progress was made on the question of whether computing an $O(\Delta)$-coloring is possible with roughly $O(n)$ space, which was stated in [Behnezhad et al., 2019] to be an interesting open problem.

In this paper we bypass the quadratic bound by presenting a new randomized $\tilde{O}(n)$-space algorithm that uses $\tilde{O}(\Delta^{1.5})$ colors.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases graph algorithms, streaming algorithms, edge coloring

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.40

Category Track A: Algorithms, Complexity and Games

Related Version *Previous Version*: https://arxiv.org/abs/2305.07090

Funding This publication is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803118 UncertainENV).

1 Introduction

The last few decades have witnessed significant technological advancements, which have led to an exponential increase in the volume of data sets and network traffic that require efficient processing. However, many of the devices that we use to perform these tasks lack sufficient storage capacity to store even a small fraction of the input data, which typically arrives in an unordered stream. Consequently, we often find ourselves processing this data using partial information. This scenario is commonplace, especially when attempting to receive information from remote servers over the internet or fetch data from an external memory unit.

[1] $\tilde{O}(f)$ hides $\log^{O(1)} f$ factors.

To gain a better understanding of computing capabilities in such scenarios, the *data stream model* has been introduced. This model involves receiving an arbitrary stream of tokens as input, and the objective is to compute a function of this input by performing one or more passes over the stream while using minimal working memory. The model has been primarily applied to problems on graphs. In this context, the data stream consists of a sequence of updates that defines the edge-set of a graph with a known number of vertices, and the aim is to compute various properties of this graph.

Clearly, our aim in such problems is to design algorithms whose space complexity is asymptotically much less than the number of edges - preferably linear in the number of vertices. However, in some cases, the size of the output may be as large as the size of the input, forcing us to use a very large space to store it. To get around this, we use a known variant of the streaming model, called the W-steaming model [5], which allows us to stream parts of the output along the computation. However, it forces the algorithm to commit to parts of the output without even seeing the entire input.

Many important problems have been studied in the W-streaming model. In this paper we focus on the problem of properly coloring the edges of a given graph with a small number of colors. This problem is considered to be one of the most fundamental and well-studied problems in graph theory, with many applications in scheduling and communications.

Behnezhad et al. [2] were the first to consider the problem of edge-coloring in the W-streaming model. They distinguished between two variants: random order streams – in which the edges arrive according to a random permutation that was chosen uniformly at random before the start of the algorithm, and adversarial order streams – in which the edges arrive in an arbitrary order. For the first variant, they provided a simple one-pass algorithm that uses $O(\Delta)$ colors and $\tilde{O}(n)$ space (where as usual Δ and n respectively denote the maximum degree of the input graph and the number of its vertices). For the second one, they provided a different one-pass algorithm that in $\tilde{O}(n)$ space computes w.h.p. a proper $O(\Delta^2)$-coloring. Charikar and Liu [7] improved the above results by devising a one-pass $\tilde{O}(n)$-space algorithm that uses $\Delta + o(\Delta)$ colors for random order streams, and a one-pass randomized algorithm that uses $\tilde{O}(ns)$ space and $(1+o(1))\Delta^2/s$ colors for adversarial order streams. More recently, Ansari et al. [1] provided two simple deterministic algorithms that in one pass and $O(ns)$ space compute a proper coloring that uses no more than $(1+o(1))\Delta^2/s$ colors for adversarial order streams, improving the result of [7].

Interestingly, for the more challenging model of adversarial order streams, all of the above results require $\tilde{\Omega}(\Delta^2)$ colors when the available space is $\tilde{O}(n)$. This raises the question of whether one can get asymptotically below this number of colors, while retaining the same space bound, or if $\tilde{\Omega}(\Delta^2)$ is an inherent limitation. In this paper we resolve this question, and show that it is possible to reduce the number of colors without compromising on increasing the given space.

1.1 Our result

We break the quadratic barrier for the first time by providing a new randomized algorithm that in one pass and $\tilde{O}(n)$-space (in expectation) computes a proper edge-coloring which uses, in expectation, no more than $\tilde{O}(\Delta^{1.5})$ colors (for adversarial order streams).

▶ **Theorem 1.** *For an undirected (multi-)graph $G = (V, E)$ on n vertices and maximum degree Δ, there is a single-pass randomized streaming algorithm using $\tilde{O}(n)$ space for edge coloring that uses $\tilde{O}(\Delta^{1.5})$ colors with high probability.*

1.2 Related work

The edge-coloring problem has been studied in several models of computation (see, e.g. [12, 10, 6, 2]). A closely related model to the W-Streaming is known as the Online model. In the online edge-coloring problem, we also assume that the edges of the input graph arrive as a data stream (which can be either random or arbitrary). There is no space limitation, but there is a requirement to instantly compute and output a color for each newly received edge, as opposed to the W-Streaming model in which we can delay the announcement of some of the edges by buffering them. See [15, 11, 13, 4, 8] for some latest results on this problem.

Independent work

In a concurrent work [3], Behnezhad and Saneian have obtained the same result as Theorem 1; besides, they have a general trade-off of $\tilde{O}(ns)$ space and $\tilde{O}(\Delta^{1.5}/s)$ colors, for any parameter s. In another concurrent work [9], Ghosh and Stoeckl have achieved a trade-off of $\tilde{O}(ns)$ space and $\tilde{O}(\Delta^2/s^2)$ colors, for any parameter s.

1.3 Technical Overview

Ansari et al. [1] used, in their second algorithm, a very simple buffering approach to color the graph's edges. Let $G = (V, E)$ be the input graph, n be its number of vertices and Δ be its maximum degree. Assuming that we have a working space of at least n words, we divide the edge-stream E into continuous intervals $E_1, ..., E_k$ of n edges each, buffer each one of them in order, and color each graph $G_i = (V, E_i)$ separately by using a different set of $O(\Delta)$ colors. This gives us a proper $O(\Delta^2)$-coloring as there can be at most Δ such intervals.

To reduce the number of colors in such an approach, we have to avoid somehow the allocation of a new $O(\Delta)$-color palette for each interval. Our main observation here is that we may not be able to edge-color the graph of a given interval G_i with much less than $\Omega(\Delta)$ colors (as it may contain vertices of high degree), but we can use fewer colors to color all the edges in G_i that are between vertices of a sufficiently low vertex degrees. We will still have to use palettes of $O(\Delta)$ colors to color the rest of the edges, but these edges are now adjacent to high degree vertices. As the number of these vertices in each of the intervals cannot be too large, we can afford to store in the machine more information about the colorings that were used to color these edges across several intervals (which we call a phase). This way, it will hopefully be sufficient to allocate a few palettes of $O(\Delta)$-colors per phase, instead of allocating a new one for each interval.

Let us develop this idea further. We divide the intervals $E_1, ..., E_k$ into contiguous phases $P_1, ..., P_{k'}$ of D intervals each. In each phase $i \in \{1, ..., k'\}$, we are going to process the intervals one by one, and color their edges. To do that, we distinguish, in each interval $E_j \in P_i$, between the "low" edges - those whose endpoints are of low degree (at most some parameter $t = \Theta(\sqrt{\Delta})$) in G_j, the "high" edges - those whose endpoints are of high degree (greater than t in G_j) and the "mixed" - all the rest. As we discussed above, in each interval $E_j \in P_i$, the number of nodes whose degree in G_j is greater than t, cannot be that large (less than $2n/t$). Thus, as long as $t \geq D$, we can store for the current phase, $O(1)$ words for each node and each interval such that this node is of degree greater than t in this interval.

With that in mind, consider the following procedure for coloring the high edges. We allocate at the start of the phase, c palettes $A_1, ..., A_c$ of $O(\Delta)$ colors each. In each interval, we choose uniformly at random a set A from the collection, and color all the high edges that are incident only to vertices that weren't previously incident to high edges that were colored using A. (that is, we store for each node that was incident to a high edge, the IDs of all the

palettes that were used to color its incident edges.) Note that each vertex may have a high degree in no more than Δ/t intervals. Therefore, the probability that a high edge will not be colored in a certain interval is at most $2\Delta/tc$, which for $c = C(\Delta/t)$ is $2/C$. This means that in expectation only a fraction of the edges remains uncolored. Let us ignore these edges for a moment, and discuss how we color the rest of the edges.

Coloring the low edges of each interval is simple as we don't need big palettes for them. So we just allocate a new set of $O(t)$ colors for each interval, and properly color the low edges of that interval with it.

We move now for the task of coloring the mixed edges. Note that this case is more complicated than the previous ones, as we need to use palettes of $K = O(\Delta)$-colors, but cannot store too much information for the nodes with low degree. As before, we start by assigning at the start of the phase a set of $c = C(\Delta/t)$ palettes $B_1, ..., B_c$ of $O(\Delta)$ colors each. In each interval, we choose uniformly at random a set B from the collection, and the idea now is that the vertex with the low degree of each mixed edge will choose a color from B to color this edge.

We illustrate the problems that could arise (when trying to color these edges) and the way we handle them, by focusing on two edges $\{u, v\}$ and $\{u, w\}$ that belong to two different intervals. Note that since these edges share a vertex, they cannot be colored with the same color. In the first case, u has a high degree in both of the intervals. A possible conflict can occur if we happen to chose the same set B in both of them. To solve this, we do the same thing that we did for the high edges (i.e. we store at the high degree vertices the IDs of the palettes with which previous incident edges were colored, and avoid coloring an edge if a conflict was detected). This will guarantee that in expectation only a fraction of them remains uncolored. In the second case, u has a low degree in both of the intervals. This means that both w and v have a high degree in their intervals, but they may be different, and so we cannot know the IDs of the previous palettes. To solve this, we maintain for u a counter c_u, which counts the total number of edges that connected u to high degree vertices when it had low degree. When u wants to choose a color from B it simply takes the c_u color in it, this will guarantee that no conflict can occur in such cases. In the third and last case, u has a high degree in one of the intervals and a low in the other. For resolving possible conflicts in this case, we make use of random offsets. At the start of the phase, we choose for each vertex v a uniformly random integer r_v between 0 and $K - 1$, and use it to permute the colors in the set (that is, v considers the first color in B to be the color in the position r_v, and so on). Before we color an edge $\{u, v\}$ we check whether the offsets r_v and r_u are far enough. If this is not the case we skip it (this happens with small probability).

To summarize, in each phase we use $O(c\Delta + tD)$ colors. As we have $O(\Delta/D)$ phases, we get that the total number of colors is $O(\Delta^3/(tD) + t\Delta)$. The edges that were left uncolored, we treat as a new virtual stream, and apply recursively the algorithm on it. As in expectation the number of uncolored edges reduces each time by a constant factor, we get that it only increases the space bound and the number of colors by a factor of $O(\log \Delta)$.

2 Edge coloring with subquadratic palette size

2.1 Algorithm description

Preparation

Our algorithm assumes prior knowledge of the maximum degree Δ. Without loss of generality, assume $\sqrt{\Delta}$ is an integer power of 2. If the data stream contains at most $O(n)$ edges, then the entire stream fits in the memory, so we can color the graph with $O(\Delta)$ colors straightforwardly (more precisely, $\lceil \frac{3\Delta}{2} \rceil$ colors for multi-graphs, and $\Delta + 1$ colors for simple graphs [14, 16]).

The algorithm divides the data stream into intervals, each interval consisting of n edges from E (the last interval may have less than n edges). In general, there are at most $|E|/n = O(\Delta)$ intervals in the data stream. Let $E_l \subseteq E$ be the set of edges in the l-th interval. Denote by $G_l = (V, E_l)$ the subgraph collected from this interval. We further partition all intervals into *phases*, each phase consisting of $\Delta^{1/2}$ consecutive intervals (the last phase may have less). Different phases will use different sets of edge colors and thus are independent. For the rest of this section, we will describe our algorithm for an individual phase.

Fix any degree threshold d be any integral power of 2. If $d < \sqrt{\Delta}$, we will create $O(\sqrt{\Delta})$ new colors for each interval which should be enough to color all edges (u, v) such that

$$\max\{\deg_{G_l}(u), \deg_{G_l}(v)\} < 2d$$

Otherwise if $d \geq \sqrt{\Delta}$, then for each interval E_l, we will describe an algorithm that assigns colors to all edges (u, v) such that

$$\max\{\deg_{G_l}(u), \deg_{G_l}(v)\} \in [d, 2d)$$

Ranging over all choices of d, we would be able to color the whole graph G_l, by blowing up the total number of colors by a factor of $O(\log \Delta)$.

A vertex $v \in V$ is called *high-degree* in G_l, if $\deg_{G_l}(v) \in [d, 2d)$, and if $\deg_{G_l}(v) < d$, then it is called *low-degree*; we will not consider any vertices whose degree in G_l is at least $2d$.

Let $\kappa \geq 32$ be a constant which is an integer power of 2. At the beginning of a single phase, the algorithm prepares three sequences of disjoint color palettes: $A_1, A_2, \cdots, A_{\kappa\Delta/d}$, and $B_1, B_2, \cdots, B_{\kappa\Delta/d}$, and $C_1, C_2, \cdots, C_{\kappa\Delta/d}$. All palettes A_i, B_i, C_i have size $K = 2\kappa d$.

For each palette X ($X \in \{A_i, B_i, C_i\}$), its colors will be indexed by $0, 1, 2 \cdots, K-1$. Each A_i will only be used to color edges between high-degree vertices in some intervals, and each $B_i \cup C_i$ will only be used to color edges between high-degree vertices and low-degree vertices in some intervals. For each vertex $v \in V$, take a uniformly random integer offset $r_v \in [0, K-1]$. Additionally, for each palette index i, maintain a counter $c_v[i] \in \{\perp\} \cup [0, K-1]$ which will bound the number of edges incident on v with colors from palette C_i; when $c_v[i] = \perp$, it means this counter $c_v[i]$ has not been initialized yet, so it does not take any space in the storage.

During this phase, each vertex v holds a set of indices $I_v \subseteq [\kappa\Delta/d]$, such that $i \in I_v$ if edges incident on v have been colored with colors from palette A_i in any previous intervals. Finally, the algorithm maintains a *virtual stream* of *leftover* edges, which are edges discarded temporarily but will be processed again recursively.

Processing intervals

Let l_0 be the index of the first interval of this phase. For each interval indexed by l, draw a random index $\sigma_l \in [\kappa\Delta/d]$ uniformly at random. For any i, keep a counter $p_l[i] = |\{\sigma_k = i \mid l_0 \leq k < l\}|$. Note that all these counters $\{p_l[i]\}_{1 \leq i \leq \kappa\Delta/d}$ can be stored using $O(\kappa\Delta/d) = O(n)$ space; for the l-th interval, we are only using counters $p_l[*]$, and previous counters will be discarded.

Partition the graph $G_l = (V, E_l)$ into two disjoint subgraphs: $G_l = H_l^1 \cup H_l^2$ defined below; edges in these subgraphs will be colored separately.

(1) H_l^1 consists of all edges between high-degree vertices in G_l.
To color edges in H_l^1, let U be the set of all high-degree vertices such that $\sigma_l \notin I_v$. Then, use the palette A_{σ_l} to color the subgraph $H_l^1[U]$ since $|A_{\sigma_l}| = 2\kappa d > 4d - 1$.

After that, for each high-degree vertex v, add σ_l to I_v by updating $I_v \leftarrow I_v \cup \{\sigma_l\}$. Finally, for each high-degree vertex $v \notin U$, insert all its incident edges in G_l to the virtual stream as leftover edges; we emphasize the point that this also includes its edges connecting to low-degree vertices.

(2) H_l^2 consists of all edges connecting high-degree vertices to low-degree vertices.
To color edges in H_l^2, go over all low-degree vertices u in the alphabetical order. For each low-degree vertex u, if $c_u[\sigma_l] = \perp$, then check if $\deg_{G_l}(u) > \sqrt{\Delta}$; if so, then initialize a counter $c_u[\sigma_l] \leftarrow 0$.
Enumerate its incident edges in H_l^2 in the alphabetical order. For the b-th edge $(u, v) \in E_l$ incident on u (starting with $b = 0$), consider several cases below.

- **(a)** If (u, v) was already inserted to the virtual stream in Step (1) due to the reason that $v \notin U$, then skip it and proceed to the next edge incident on u.
- **(b)** Define $\delta = r_v - r_u \mod K$, so $d \in [0, K-1]$. If $\delta < 2d$ or $\delta > K - 2d$, then insert edge (u, v) to the virtual stream as a leftover edge.
- **(c)** Otherwise, assume $2d \leq \delta \leq K - 2d$. There are several sub-cases.
  - If $c_u[\sigma_l] = 2d$, then simply insert the edge (u, v) to the virtual stream as a leftover edge.
  - If $c_u[\sigma_l] \neq \perp$ and $c_u[\sigma_l] < 2d$, then check if the $(r_u + c_u[\sigma_l])$-th color from palette C_{σ_l} has already been used in H_l^2 on any other edge incident on high-degree vertex v. If so, then insert the edge (u, v) to the virtual stream as a leftover edge; otherwise, assign the $(r_u + c_u[\sigma_l])$-th color from palette C_{σ_l} to edge (u, v).
  - Otherwise if $c_u[\sigma_l] = \perp$, check if $p_l[\sigma_l] \geq 2d/\sqrt{\Delta}$. If so, insert edge (u, v) to the virtual stream as a leftover edge.
    In the case that $c_u[\sigma_l] = \perp$ and $p_l[\sigma_l] < 2d/\sqrt{\Delta}$, check if the $\left(r_u + b + \sqrt{\Delta} \cdot p_l[\sigma_l]\right)$-th color from palette B_{σ_l} has already been used in H_l^2 on any other edge incident on high-degree vertex v. If so, then insert the edge (u, v) to the virtual stream as a leftover edge; otherwise, assign the $\left(r_u + b + \sqrt{\Delta} \cdot p_l[\sigma_l]\right)$-th color from palette B_{σ_l} to edge (u, v).

In all cases (a)(b)(c), if $c_u[\sigma_l] \neq \perp$, increment the counter by one $c_u[\sigma_l] \leftarrow c_u[\sigma_l] + 1$ afterwards (even if (u, v) is inserted to the virtual stream).

Coloring leftover edges

To color all leftover edges in the virtual stream, we apply the above algorithm recursively on this virtual stream as its input data stream, with fresh colors and independent randomness. Some of the notations used in the algorithm description are summarized in Table 1.

2.2 Proof of correctness

Let us first state some basic properties of the algorithm.

▶ **Lemma 2.** *The value of any counter $c_u[i]$ depends on the data stream and randomness of indices $\{\sigma_l\}$, but is independent of the random shifts $\{r_v \mid v \in V\}$.*

Proof. According to the description of the algorithm, each counter $c_u[i]$ is initialized to 0 in the first interval of edges E_k such that $\deg_{G_k}(u) > \sqrt{\Delta}$ and $\sigma_k = i$; this event is independent of random shifts $\{r_v \mid v \in V\}$. Later on, for any $l \geq k$ such that $\sigma_l = i$, $c_u[i]$ will increase by $|\deg_{H_l^2}(u)|$ if u is a low-degree vertex in G_l. Therefore, $c_u[i]$ is always independent of the random shifts $\{r_v \mid v \in V\}$. ◀

Table 1 Some of the notations used in the algorithm.

notation	definition
E_l	the set of $O(n)$ edges in the l-th interval in from the data stream
G_l	$G_l = (V, E_l)$ is the subgraph collected in the l-th interval
phase	a phase consists of $\Delta^{1/2}$ consecutive intervals
κ	a constant at least 32 which is also an integer power of 2
$A_i, B_i, C_i, 1 \le i \le \kappa\Delta/d$	three sequences of sets of palettes of size $K = 2\kappa d$
$c_v[i] \in \{\perp\} \cup [0, K-1]$	a counter bounding using colors from palette C_i
r_v	$r_v \in [0, K-1]$ is a random shift
I_v	$i \in I_v$ if v already has edges with colors from A_i
σ_l	a uniformly random index from $[1, \kappa\Delta/d]$ for the l-th interval
$p_l[i]$	a counter defined by $p_l[i] = \lvert\{\sigma_k = i \mid l_0 \le k < l\}\rvert$
high-degree vertex	high-degree if $\deg_{G_l} \in [d, 2d)$
low-degree vertex	high-degree if $\deg_{G_l} < d$
H_l^1	all edges between high-degree vertices in G_l
H_l^2	all edges connecting high-degree vertices and low-degree vertices

▶ **Lemma 3.** *At the beginning of the l-th interval, for any low-degree vertex u, if $c_u[\sigma_l] \neq \perp$, then all colors from any palette C_{σ_l} indexed by $r_u + c_u[\sigma_l], r_v + c_u[\sigma_l] + 1, \cdots, r_u + 2d - 1$ mod K haven't been used for any edge incident on u.*

Proof. This is because the counter $c_u[\sigma_l]$ increases by one each time we use a color from C_{σ_l} on edges incident on u, so the colors indexed by larger integers have not been used yet. ◀

▶ **Lemma 4.** *For each vertex u, at any moment during the current phase, there are at most $\frac{\Delta}{2d}$ different indices i such that*

$$\sum_{l,\sigma_l=i} \deg_{G_l}(u) \ge 2d$$

Proof. Since $\deg_G(u) \le \Delta$, the number of indices i such that $\sum_{l,\sigma_l=i} \deg_{G_l}(u) \ge 2d$ is bounded by $\frac{\Delta}{2d}$. ◀

Next, we show that our algorithm always produces a valid edge coloring.

▶ **Lemma 5.** *The algorithm always outputs a valid edge coloring of G.*

Proof. Consider any two adjacent edges $(u, v), (u, w) \in E$, and we will show that $(u, v), (u, w)$ must have different colors in the output stream. Note that v does not need to be different from w since G might not be a simple graph. If one of them appears in the virtual stream, then because the recursion for leftover edges uses fresh colors for each interval, we can make sure that $(u, v), (u, w)$ are assigned different colors. For the rest we assume that $(u, v), (u, w)$ are not leftover edges.

First, consider the case where both $(u, v), (u, w)$ appear in the same interval from the input stream. According to the algorithm, if they are colored in different Step (1) and (2) respectively, then they are picking colors from separate palettes; if they are colored in the same Step (1) or (2), then the algorithm must have colored them in a compatible way.

Secondly, consider the case where $(u, v), (u, w)$ appear in different intervals from the input stream. Assume (u, v) is in interval E_l and (u, w) is in interval $E_h, h \neq l$. Consider the following cases.

- One of $(u,v),(u,w)$ is colored in Step (1) and the other in Step (2).
  In this case, they are using different palettes, so their colors are always different.
- Both $(u,v),(u,w)$ are colored in Step (1) but in different intervals.
  In this case, since (u,w) is not a leftover edge, by the algorithm we know that $\sigma_l \neq \sigma_h$. Therefore, $(u,v),(u,w)$ use colors from different palettes $A_{\sigma_l}, A_{\sigma_h}$.
- Both $(u,v),(u,w)$ are colored in Step (2) but in different intervals.
  If u is high-degree in both intervals E_l, E_h, then since neither of $(u,v),(u,w)$ is a leftover edge, it must be $\sigma_l \neq \sigma_h$. Hence, $(u,v),(u,w)$ are taking colors from different palettes.
  Next, assume u is low-degree in both intervals, plus that $\sigma_l = \sigma_h$. If $(u,v),(u,w)$ were selecting colors from $B_{\sigma_l}, C_{\sigma_h}$, then their colors must be different. So, we only need to consider the case where both $(u,v),(u,w)$ were selecting colors from B_{σ_l}, or from C_{σ_l}. Let us analyze these two cases separately.
  - $(u,v),(u,w)$ were selecting colors from B_{σ_l}. Then, by the algorithm description, (u,v) could only take colors from B_{σ_l} taking indices from the range

    $$[r_u + \sqrt{\Delta} \cdot p_l[\sigma_l], r_u + \sqrt{\Delta} \cdot (p_l[\sigma_l]+1) - 1] \mod K$$

    and (u,w) could only take colors from B_{σ_l} taking indices from the range

    $$[r_u + \sqrt{\Delta} \cdot p_h[\sigma_h], r_u + \sqrt{\Delta} \cdot (p_h[\sigma_h]+1) - 1] \mod K$$

    As $l \neq h$ and $\sigma_l = \sigma_h$, we know that $p_l[\sigma_l] \neq p_h[\sigma_h]$, and therefore the two ranges

    $$[r_u + \sqrt{\Delta} \cdot p_l[\sigma_l], r_u + \sqrt{\Delta} \cdot (p_l[\sigma_l]+1) - 1] \mod K$$

    and

    $$[r_u + \sqrt{\Delta} \cdot p_h[\sigma_h], r_u + \sqrt{\Delta} \cdot (p_h[\sigma_h]+1) - 1] \mod K$$

    are disjoint, and thus the colors of the two edges must be different.
  - $(u,v),(u,w)$ were selecting colors from C_{σ_l}.
    Suppose the color indices of $(u,v),(u,w)$ were $r_u + c_u^{(1)}[\sigma_l]$ and $r_u + c_u^{(2)}[\sigma_l]$, where $c_u^{(1)}[\sigma_l] \neq c_u^{(2)}[\sigma_l]$ are the counter values of $c_u[\sigma_l]$ by the time when the color was selected. Hence, $(u,v),(u,w)$ are picking colors with different indices.

  Finally, assume u is high-degree in G_l and low-degree in G_h. Then, when coloring the edge (u,v), by the condition on Step (2)(b), we know that $\delta = r_u - r_v \mod K$ belongs to $[2d, K-2d]$. By Lemma 3, as the color index of (u,v) always belongs to $\{r_v, r_v+1, \cdots, r_v+2d-1\}$ and the color index of (u,w) always belongs to $\{r_u, r_u+1, \cdots, r_u+2d-1\}$, the colors of $(u,v),(u,w)$ should be different as the two index sets are disjoint. ◀

Next, we verify that the total space of each recursion level is at most $O(n)$.

▶ **Lemma 6.** *The space usage of each recursion level of the algorithm is $O(n)$.*

Proof. It is clear that in each phase, the total space of all offsets is at most $O(n)$. For the number of counters in a single phase, every time a new counter $c_u[\sigma_l]$ has been initialized, we must have $\deg_{G_l}(u) > \sqrt{\Delta}$ for some interval index l within this phase. Since the total sum of vertex degrees within this phase is bounded by $O(n\sqrt{\Delta})$, we know that the number of counters is bounded by $O(n)$.

As for the total space sets I_v's, in each interval, any set I_v adds one more element if v is high-degree. As the total number of edges collected from a single phase is $O(n\sqrt{\Delta})$, The total size of $\sum_{v \in V} |I_v|$ would be at most $O(n\sqrt{\Delta}/d) = O(n)$. ◀

To bound the total number different colors in G, we need to analyze the total number of leftover edges of each recursion level.

▶ **Lemma 7.** *Suppose there are m edges from the input stream (across all phases) in total. Then, the expected number of leftover edges to enter the virtual stream is at most $7m/\kappa$, over the randomness of shifts $\{r_v\}_{v\in V}$ and indices $\{\sigma_l\}$.*

Proof. Consider any single phase, and let $(u,v)\in E$ be an arbitrary edge that appears in interval E_l. There are several cases where (u,v) could possibly be a leftover edge that gets inserted to the virtual stream.

- Edge (u,v) becomes leftover on Step (1).
  The condition for (u,v) being cast as a leftover edge is that $\sigma_l \in I_u \cup I_v$; that is, palette A_{σ_l} was already used for edges incident on u or v in a previous interval. Since any vertex can be high-degree in at most Δ/d intervals, the number of non-zero entries of I_u, I_v is at most $2\Delta/d$. As $\sigma_l \in [\kappa\Delta/d]$ is uniformly random, the probability that $\sigma_l \in I_u \cup I_v$ is at most $2/\kappa$, over the randomness of σ_l.
- Edge (u,v) becomes leftover on Step (2) because $c_u[\sigma_l] = 2d$.
  By Lemma 4, there are at most $\frac{\Delta}{2d}$ different indices i such that $\sum_{k=l_0,\sigma_k=i}^{l-1} \deg_{G_k}(v) \geq 2d$; let I be the set of these indices. Then, since σ_l is picked uniformly at random, the probability that $\sigma_l \in I$ is at most $\frac{1}{2\kappa}$.
  As $c_u[i] \leq \sum_{k=l_0,\sigma_k=i}^{l-1} \deg_{G_k}(v)$ for any i, the probability that $c_u[\sigma_l] = 2d$ is at most $\frac{1}{2\kappa}$. Therefore, (u,v) becomes leftover on this step with probability at most $\frac{1}{2\kappa}$.
- Edge (u,v) becomes leftover on Step (2)(b).
  In this case, we have $\delta < 2d$ or $\delta > K - 2d$ for $\delta = r_v - r_u \mod K$. Since r_v, r_u were chosen uniformly at random from $[0, K-1]$, the probability of this event is at most $4d/K < 2/\kappa$, over the randomness of r_v.
- Edge (u,v) becomes leftover on Step (2)(c) due to $p_l[\sigma_l] \geq 2d/\sqrt{\Delta}$.
  Note that since each σ_k was picked uniformly at random, the expectation of $p_l[\sigma_l]$ which counts $|\{\sigma_k = \sigma_l \mid l_0 \leq k < l\}|$ is bounded by $\frac{d}{\kappa\sqrt{\Delta}}$, over the randomness of $\sigma_{l_0}, \sigma_{l_0+1}, \cdots, \sigma_{l-1}$. By Markov's inequality, the probability that $p_l[\sigma_l] \geq 2d/\sqrt{\Delta}$ is at most $\frac{1}{2\kappa}$.
- Edge (u,v) becomes leftover on Step (2)(c) and $p_l[\sigma_l] < 2d/\sqrt{\Delta}$.
  In this case, assume u is low-degree and v is high-degree in interval E_l. We need to consider two more sub-cases below.
  - $c_u[\sigma_l] \neq \perp$.
    In this case, edge (u,v) was attempting to use a color from C_{σ_l}. Right before (u,v) was enumerated, let S be the set of low-degree neighbors w of v whose alphabetical orders are before u, plus that $c_w[\sigma_l] \neq \perp$.
    For each vertex $w \in S$, suppose there are k_w parallel edges between v, w, and let b_w be the value of the counter $c_w[\sigma_l]$ when the first edge (u,w) was enumerated from the perspective of w. Define an index set

$$I = \bigcup_{w\in S} \{r_w + b_w, r_w + b_w + 1, \cdots, r_w + b_w + k_w - 1\}$$

    Note that $|I| < 2d$ since $\sum_{w\in S} k_w \leq \deg_{G_l}(v) < 2d$. As the algorithm enumerates vertices and edges on Step (2)(c) in the alphabetical order, we know that I is the set of all possible color indices of edges $(v,w), w \in S$ in palette C_{σ_l}.
    Now, on the one hand, by Lemma 2, all values of b_w, k_w are independent of the random shifts $\{r_z \mid z \in V\}$. Therefore, as $u \notin S$, we know that r_u is independent of I. On

the other hand, for (u, v) to be a leftover edge, $r_u + c_u[\sigma_l]$ must belong to I. As the value $c_u[\sigma_l]$ is also independent of r_u, the probability that $r_u + c_u[\sigma_l] \in I$ is at most $|I|/K < 1/\kappa$, over the random choice of r_u.

- $c_u[\sigma_l] = \perp$.
  In this case, edge (u, v) was attempting to use a color from B_{σ_l}. Similar to the previous sub-case, right before (u, v) was enumerated, let T be the set of low-degree neighbors w of v whose alphabetical orders are before u, plus that $c_w[\sigma_l] = \perp$.
  For each vertex $w \in T$, suppose there are k_w parallel edges between u, w, and (u, w) (the first copy) is the b_w-th edge incident on w. Define an index set

$$J = \bigcup_{w \in T} \{r_w + b_w + \sqrt{\Delta} \cdot p_l[\sigma_l], r_w + b_w + \sqrt{\Delta} \cdot p_l[\sigma_l] + 1, \cdots, r_w + b_w + \sqrt{\Delta} \cdot p_l[\sigma_l] + k_w - 1\}$$

  Note that $|J| < 2d$ since $\sum_{w \in T} k_w \leq \deg_{G_l}(v) < 2d$. As the algorithm enumerates vertices and edges on Step (2)(c) in the alphabetical order, we know that J is the set of all possible color indices of edges $(v, w), w \in T$ in palette B_{σ_l}.
  Suppose (u, v) is the b-th edge of u in G_l, so $r_u + b + \sqrt{\Delta} \cdot p_l[\sigma_l]$ is the color index that (u, v) attempted to use in B_{σ_l}. Since $u \notin T$, and that r_u is independent of $p_l[\sigma_l]$ and all values in $\{b_w \mid w \in T\}$, we know that the probability that $r_u + b + \sqrt{\Delta} \cdot p_l[\sigma_l] \in J$ is at most $|J|/K < 1/\kappa$, over the random choice of r_u.

Taking a summation of all the cases, the probability that (u, v) becomes a leftover edge is at most $7/\kappa$. Therefore, expected number of leftover edges is bounded by $7m/\kappa$. ◀

▶ **Lemma 8.** *The expected recursion depth of the main algorithm is $O(\log \Delta)$.*

Proof. Consider any recursion where the input stream contains m edges. By Markov's inequality and Lemma 7, at each recursion level, with probability at least $1/2$, the number of leftover edges is at most $14m/\kappa < m/2$. Since the original graph G contains $O(n\Delta)$ edges, after $O(\log \Delta)$ recursion levels in expectation, the input has at most $O(n)$ edges, so the algorithm would not recurse further. ◀

As a corollary, we can bound the total number of different colors and total memory, which concludes the proof of Theorem 1.

▶ **Corollary 9.** *The number of colors used by our algorithm is $\tilde{O}\left(\Delta^{1.5}\right)$, and the total memory is bounded by $O(n \log \Delta)$; both bounds hold in expectation.*

Proof. Consider any recursion level. According to the algorithm, in each phase and each choice of $d \geq \sqrt{\Delta}$ which is an integer power of 2, the total number of colors in the palettes $\{A_i\} \cup \{B_i\} \cup \{C_i\}$ is $O(\Delta)$. Also, each interval creates at most $O(\sqrt{\Delta})$ new colors when $d < \sqrt{\Delta}$. Since there are $O(\sqrt{\Delta})$ phases, the number of colors used for processing intervals is $\tilde{O}(\Delta^{1.5})$, summing over all choices of d. As the expected recursion depth is $O(\log \Delta)$, together with Lemma 6 we can finish the proof. ◀

2.3 Adaptation to an unknown Δ

So far we have assumed that the value of the maximum degree Δ of G is known to the algorithm as prior knowledge. We can adapt our algorithm to an unknown Δ by losing a constant factor in the total number of colors in the following way. Basically, we will maintain the value Δ_t which is the maximum degree of the subgraph containing the first t edges in the data stream. Whenever $\Delta_t \in (2^{k-1}, 2^k]$, we will apply Theorem 1 with $\Delta = 2^k$ to color

all the edges. If k increases at some point, we will restart a new instance of Theorem 1 with a new choice of $\Delta = 2^k$ and continue to color the edges with a separate palette; to clarify, when a new instance of Theorem 1 is restarted, we continue with the current pass of the data stream, not starting over with a new pass. In the end, the total number of colors will be $\tilde{O}(\sum_{k=1}^{\lceil \log \Delta \rceil} 2^{1.5k}) = \tilde{O}(\Delta^{1.5})$ in expectation.

2.4 From expectation to high probability

So far our bounds on memory and the number of colors only hold in expectation, not with high probability. These bounds can actually hold with high probability rather than in expectation. In fact, as shown in Lemma 8, the expected recursion depth can be bounded by $O(\log \Delta)$. We can also show that the depth is $O(\log n)$ with high probability, and therefore the total number of colors is at most $O(\Delta^{1.5} \cdot \log n)$. To get rid of the dependency on $\log n$, we can increase the size of intervals by a factor of $\log n$; that is, each interval now contains $O(n \log n)$ edges. This would decreases the total number of phases by a factor of $\Omega(\log n)$ and thus decreases the number of colors, which cancels out the extra $\log n$ factor in the color bound.

References

1 Mohammad Ansari, Mohammad Saneian, and Hamid Zarrabi-Zadeh. Simple streaming algorithms for edge coloring. In *Proccedings of the 30th Annual European Symposium on Algorithms (ESA '22)*, pages 8:1–8:4, 2022.

2 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Marina Knittel, and Hamed Saleh. Streaming and massively parallel algorithms for edge coloring. In *Proccedings of the 27th Annual European Symposium on Algorithms (ESA '19)*, pages 15:1–15:14, 2019.

3 Soheil Behnezhad and Mohammad Saneian. Streaming edge coloring with asymptotically optimal colors. *arXiv preprint*, 2023. `arXiv:2305.01714`.

4 Sayan Bhattacharya, Fabrizio Grandoni, and David Wajc. Online edge coloring algorithms via the nibble method. In *Proceedings of the 32nd Symposium on Discrete Algorithms (SODA '21)*, pages 2830–2842, 2021.

5 Andrea Ribichini Camil Demetrescu, Irene Finocchi. Trading off space for passes in graph streaming problems. In *Proccedings of the 17th Annual Symposium on Discrete Algorithms (SODA '06)*, pages 714–723, 2006.

6 Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. The complexity of distributed edge coloring with small palettes. In *Proceedings of the Twenty-Ninth Annual Symposium on Discrete Algorithms (SODA '18)*, pages 2633–2652, 2018.

7 Moses Charikar and Paul Liu. Improved algorithms for edge colouring in the W-streaming model. In *Proccedings of the 4th Symposium on Simplicity in Algorithms (SOSA '21)*, pages 181–183, 2021.

8 Ilan Reuven Cohen, Binghui Peng, and David Wajc. Tight bounds for online edge coloring. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1–25. IEEE, 2019.

9 Prantar Ghosh and Manuel Stoeckl. Low-memory algorithms for online and w-streaming edge coloring. *arXiv preprint*, 2023. `arXiv:2304.12285`.

10 David B Shmoys Howard J Karloff. Efficient parallel algorithms for edge coloring problems. *Journal of Algorithms*, 8(1):39–52, 1987.

11 Janardhan Kulkarni, Yang P Liu, Ashwin Sah, Mehtaab Sawhney, and Jakub Tarnawski. Online edge coloring via tree recurrences and correlation decay. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 104–116, 2022.

12 J. Misra and David Gries. A constructive proof of Vizing's theorem. *Information Processing Letters*, 41(3):131–133, 1992.

13 Amin Saberi and David Wajc. The greedy algorithm is not optimal for on-line edge coloring. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP '21)*, pages 109:1–109:18, 2021.

14 Claude E Shannon. A theorem on coloring the lines of a network. *Journal of Mathematics and Physics*, 28(1-4):148–152, 1949.

15 Aravind Srinivasan, David Wajc, et al. Online dependent rounding schemes. *arXiv preprint*, 2023. arXiv:2301.08680.

16 Vadim G Vizing. Critical graphs with given chromatic class (in russian). *Metody Discret. Analiz.*, 5:9–17, 1965.

Faster Algorithms for Dual-Failure Replacement Paths

Shiri Chechik
Tel Aviv University, Israel

Tianyi Zhang
Tel Aviv University, Israel

Abstract

Given a simple weighted directed graph $G = (V, E, \omega)$ on n vertices as well as two designated terminals $s, t \in V$, our goal is to compute the shortest path from s to t avoiding any pair of presumably failed edges $f_1, f_2 \in E$, which is a natural generalization of the classical replacement path problem which considers single edge failures only.

This dual failure replacement paths problem was recently studied by Vassilevska Williams, Woldeghebriel and Xu [FOCS 2022] who designed a cubic time algorithm for general weighted digraphs which is conditionally optimal; in the same paper, for unweighted graphs where $\omega \equiv 1$, the authors presented an algebraic algorithm with runtime $\tilde{O}(n^{2.9146})$, as well as a conditional lower bound of $n^{8/3-o(1)}$ against combinatorial algorithms. However, it was unknown in their work whether fast matrix multiplication is necessary for a subcubic runtime in unweighted digraphs.

As our primary result, we present the first truly subcubic combinatorial algorithm for dual failure replacement paths in unweighted digraphs. Our runtime is $\tilde{O}(n^{3-1/18})$. Besides, we also study algebraic algorithms for digraphs with small integer edge weights from $\{-M, -M+1, \cdots, M-1, M\}$. As our secondary result, we obtained a runtime of $\tilde{O}(Mn^{2.8716})$, which is faster than the previous bound of $\tilde{O}(M^{2/3}n^{2.9144} + Mn^{2.8716})$ from [Vassilevska Williams, Woldeghebriela and Xu, 2022].

2012 ACM Subject Classification Theory of computation → Shortest paths

Keywords and phrases graph algorithms, shortest paths, replacement paths

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.41

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2404.13907

Funding This work is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803118 UncertainENV).

1 Introduction

In the replacement path problem, we want to understand shortest paths in a directed graph that avoid presumably failed edges. More specifically, let $G = (V, E, \omega)$ be an edge-weighted simple digraph on n vertices and m edges. Fix a pair of source and terminal vertices $s, t \in V$, we want to compute the shortest path from s to t that avoids any designated set $F \subseteq E$ of failed edges.

The most classical setting is when the number of failures is at most one; namely, we want to compute all the values of $\mathsf{dist}(s, t, G \setminus \{f\})$ when f ranges over all edges on the shortest path from s to t in G. The complexity of single-failure replacement path is now well-understood. On the hardness side, it was proved that computing all single-failure replacement paths in weighted graphs requires at least $n^{3-o(1)}$ time [19] assuming the APSP conjecture. To breach the cubic barrier, we need to assume the input digraph has small integer edge weights, or allow approximation errors in the algorithm output. When the edge

weights are integers in the range $\{-M, -M+1, \cdots, M-1, M\}$, there is an algorithm with runtime $\tilde{O}(Mn^{\omega})$[1] [7, 17]. For the special case when the input digraph is unweighted ($\omega \equiv 1$), there is a combinatorial algorithm (algorithms not using fast matrix multiplication) with runtime $\tilde{O}(mn^{1/2})$ [16], which is optimal under the hardness of combinatorial boolean matrix multiplication [19].

A natural extension is to study replacement paths when there are two edge failures. We are interested in fast algorithms that compute for all pairs of edges $f_1, f_2 \in E$ the value of $\mathsf{dist}(s, t, G \setminus \{f_1, f_2\})$. This problem was first studied in [4] and recently revisited in [20]. For general weighted digraphs, the authors of [20] designed an algorithm with runtime $\tilde{O}(n^3)$, which is the same as the easier single failure replacement paths problem. When the graph has small edge weights from range $\{-M, -M+1, \cdots, M-1, M\}$, in the same paper the authors have shown subcubic runtime upper bound of $\tilde{O}(M^{2/3}n^{2.9144} + Mn^{2.8716})$ using fast matrix multiplication. Finally, as complementary to their algorithms, the authors showed a conditional lower bound of $n^{8/3-o(1)}$ against combinatorial algorithms for unweighted digraphs assuming the hardness of boolean matrix multiplication.

According to the results in [20], there is a gap in their understanding about dual-failure replacement paths in unweighted graphs. On the one hand, their algebraic algorithm computes dual-failure replacement paths in $\tilde{O}(n^{2.9144})$ by setting $M = 1$; on the other hand, their conditional lower bound against combinatorial algorithms is also subcubic. So, it is not clear whether combinatorial algorithm can achieve subcubic runtime as well, or the conditional lower bound can be improved to cubic.

1.1 Our results

In this paper, we first study fast combinatorial algorithms for unweighted digraphs and show that subcubic runtime can indeed be achieved without using fast matrix multiplication.

▶ **Theorem 1.** *Given a simple unweighted directed graph $G = (V, E)$ on n vertices, and fix any pair of vertices $s, t \in V$, the values of all dual-failure replacement path distances $\mathsf{dist}(s, t, G \setminus \{f_1, f_2\}), \forall f_1, f_2 \in E$ can be computed in $\tilde{O}(n^{3-1/18})$ time with high probability; most importantly, the algorithm does not use fast matrix multiplication.*

Here, a digraph is simple if it does not contain two edges between the same pair of vertices with the same direction. Secondly, we also study fast algebraic algorithms for dual-failure replacement paths when the edge weights are from the set $\{-M, -M+1, \cdots, M-1, M\}$. The proof of the following statement is presented in the full version.

▶ **Theorem 2.** *Given a simple weighted directed graph $G = (V, E, \omega)$ on n vertices along with integer edge weights $\omega : E \to \{-M, -M+1, \cdots, M-1, M\}$ without negative cycles, and fix any pair of vertices $s, t \in V$, the values of all dual-failure replacement path distances $\mathsf{dist}(s, t, G \setminus \{f_1, f_2\}), \forall f_1, f_2 \in E$ can be computed in time $\tilde{O}(Mn^{2.8716})$.*

1.2 Other related works

The replacement paths problem has also been studied in other settings, including the single-source setting [13, 6, 12, 11] and the approximation setting [8, 2, 15].

[1] $\omega \in [2, 2.371552]$ is the fast matrix multiplication exponent [21, 10, 1, 14, 18].

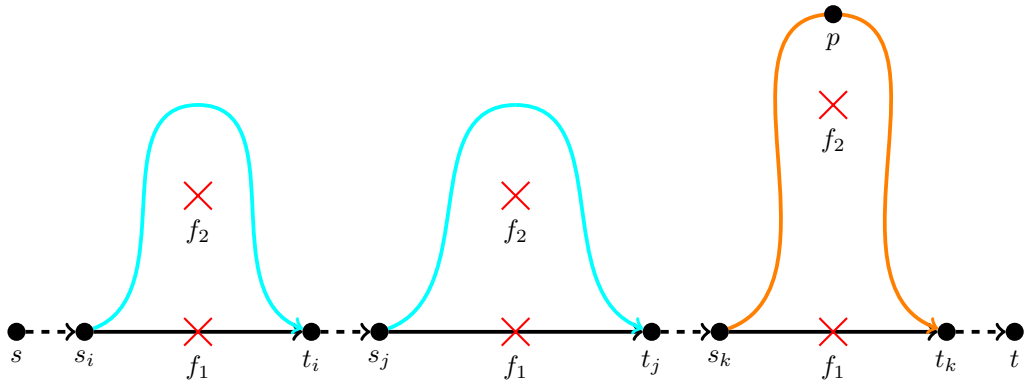


Figure 1 For simplicity, let us assume that when f_1 falls on the sub-paths $\pi[s_i, t_i]$, the dual-failure replacement path also passes through s_i, t_i. In this picture, the two cyan dual-failure replacement paths have length less than L, and we will show that they are vertex-disjoint. The orange dual-failure replacement path has length at least L, and so it hits a vertex $p \in U$ with high probability; in this case, we will compute single-source single-failure replacement paths to and from p in graph $G \setminus E(\pi)$ to help us compute dual-failure replacement paths.

1.3 Subcubic combinatorial algorithm for unweighted digraphs

1.3.1 One failure on a long *st*-path

Let us first consider the case where one edge failure f_1 lies on the shortest st-path π, while the other one f_2 does not. This case would be easy when we use fast matrix multiplication as did by [20], but it becomes complicated when we are restricted to purely combinatorial algorithms.

As a preliminary step, we first show how to deal with the case where $|\pi| > L$ for some parameter L slightly larger than $n^{0.5}$, say $L = n^{0.55}$. Partition the st-path π into sub-paths of length exactly $5L$ as $\pi = \gamma_1 \circ \gamma_2 \circ \cdots \circ \gamma_h, h \leq \lceil n/5L \rceil$, and let s_i, t_i be the endpoints of sub-path γ_i. Assume only one edge failure f_1 falls on sub-path γ_i, and let ρ be the optimal replacement path from s to t avoiding $\{f_1, f_2\}$.

Take a random set of vertices U of size $O(n \log n/L)$. If $|\rho \setminus \pi| > L$, then with high probability $\rho \setminus \pi$ would contain a vertex p from U. Then, we can compute single-source single-failure replacement paths to and from p in graph $G \setminus E(\pi)$ that takes runtime $\tilde{O}(n^{3.5}/L)$ [6], so that for each vertex $z \in V$, we know the shortest path between z, p in graph $G \setminus (E(\pi) \cup \{f_2\})$. Using this information, we will be able to recover ρ.

Now suppose $|\rho \setminus \pi| < L$. Then in this case, we will show that the detour parts of different dual-failure replacement paths ρ are vertex-disjoint from each other when the first edge failure f_1 comes from different choice of sub-paths γ_i. Then, we can partition G into vertex-disjoint subgraphs $G_1, G_2, \cdots, G_h$, such that the dual-failure replacement paths for failures on γ_i belong to subgragh G_i, and solve the dual-failure replacement paths problem for source-terminal pair (s_i, t_i) in graph G_i. See Figure 1 for an illustration.

1.3.2 One failure on a short *st*-path

By the previous subsection, we have reduced to the case that the st-path has length at most L. So, for the rest, let us rename the problem instance and assume that $|\pi| \leq L$. For the i-th edge e_i on π $(0 \leq i < L)$, we can compute the optimal replacement path from s to t avoiding e_i and let α_i be the corresponding detour whose endpoints are a_i and b_i. Again, through

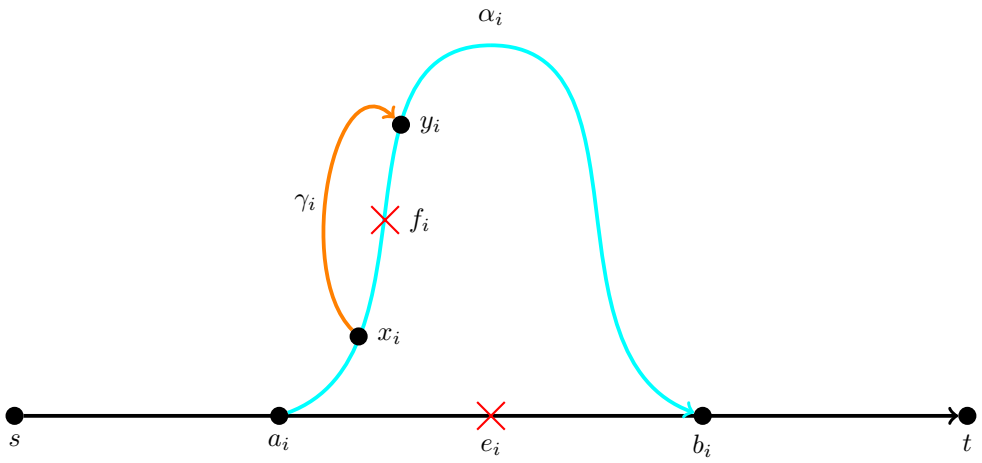


Figure 2 The cyan path is the detour α_i that avoids e_i, and the orange path is the detour γ_i that avoids f_i which lies on α_i. Via some case analysis, we will show the difficult case is that $|\alpha_i| < L$.

some case analysis, we can assume that the span of each detour α_i which is $|\pi[a_i, b_i]|$ is at most g for some parameter g slightly larger than $n^{1/3}$ (say $g = n^{0.35}$). Consider any edge failure $f_i \in E(\alpha_i)$, as a simplification let us assume that the optimal replacement path from s to t avoiding $\{e_i, f_i\}$ is a concatenation:

$$\rho = \pi[s, a_i] \circ \alpha_i[a_i, x_i] \circ \gamma_i \circ \alpha_i[y_i, b_i] \circ \pi[b_i, t]$$

where γ_i is a detour with respect to α_i in $G \setminus E(\pi)$. Via some case analysis, we can assume that $|\alpha_i| < L$. See Figure 2 for an illustration.

Let us first consider the case when $|\gamma_i| < g$. A wishful thought is that for two different choice of dual failures $\{e_i, f_i\}$ and $\{e_j, f_j\}$ where e_i and e_j are well-separated on the shortest path π ($|\pi(e_i, e_j)| \geq 10g$), we are guaranteed that the two dual-failure detours γ_i and γ_j are vertex-disjoint; if this is the case, then we can partition the graph G into $O(L/g)$ vertex-disjoint subgraphs and compute dual-failure replacement paths separately. However, this is generally not true. The key observation is that when f_i and f_j are roughly at the same height on the detour, namely $|\alpha_i[a_i, f_i)| \approx |\alpha_j[a_j, f_j)|$ up to an additive error of at most g, such a disjointness condition indeed holds. Therefore, our algorithm will further partition each detour α_i into sub-paths of length g as $\alpha_i = \beta_1^i \circ \beta_2^i \circ \cdots \circ \beta_l^i$. Then, fix any height index $1 \leq h \leq l$, we will deal with all the dual failures $\{e_i, f_i\}, \forall 1 \leq i \leq L/g, \forall f_i \in E(\beta_h^i)$ at the same time. See Figure 3 for an illustration.

Now, what happens if $|\gamma_i| \geq g$? We can take a random sample U of pivot vertices of size $O(n \log n/g)$. Then, with high probability, γ_i contains a vertex in U. For simplicity, assume both endpoints of the detour γ_i are lying within the sub-path β_h^i which contains the second edge failure f_i, then we could compute single-source shortest paths to and from each vertex $p \in U$ in the subgraph $G \setminus \big(E(\pi) \cup E(\beta_h^i)\big)$ which takes time $\tilde{O}(n^3/g)$, then we can compute each detour γ_i for each choice of f_i on β_h^i in time $\tilde{O}(g^2)$ by guessing the positions of x_i, y_i.

Unfortunately, there are L^2/g different choices for the subgraph $G \setminus \big(E(\pi) \cup E(\beta_h^i)\big)$, and thus we do not have enough time to compute single-source shortest paths for each vertex in U in all these subgraphs. In fact, we can only afford to compute single-source shortest paths for vertices in U in the subgraph $G_h = G \setminus \Big(E(\pi) \cup \bigcup_{i=0}^{L-1} E(\beta_h^i)\Big)$; that is, for each index h, remove all sub-paths $\beta_h^i, \forall 0 \leq i < L$ from $G \setminus E(\pi)$ simultaneously (which becomes G_h), and compute multi-source shortest paths to and from U in G_h. The key observation is that

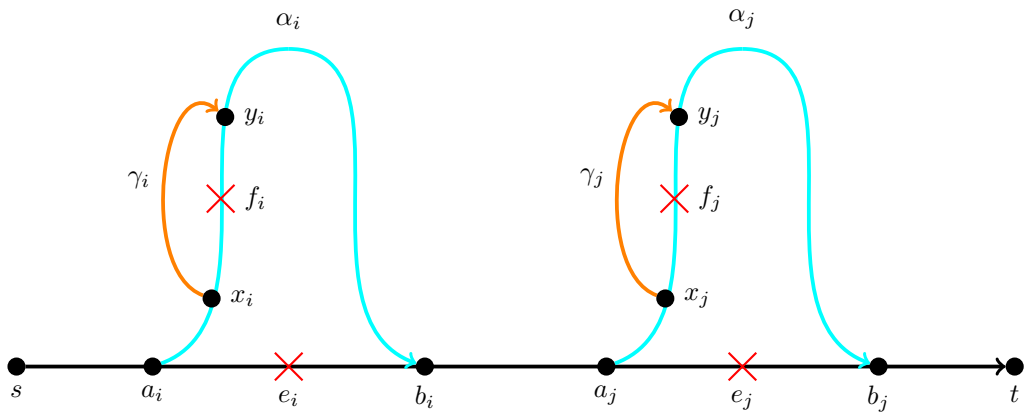


Figure 3 For two well-separated edges e_i, e_j such that $|\pi(e_i, e_j)| \geq 10g$, if both $|\gamma_i|, |\gamma_j|$ are less than g, and f_i, f_j are roughly at the same height (i.e., $|\alpha_i[a_i, f_i)| \approx |\alpha_j[a_j, f_j)|$), then we can show that the two dual-failure detours γ_i, γ_j are vertex-disjoint.

the detour γ_i cannot touch vertices on β_h^j if $|i - j| > 10g$. Therefore, to compute γ_i, we can build a small shortcut graph consisting of all vertices in $U \cup \bigcup_{j=i-10g}^{i+10g} V(\beta_h^j)$ which contains all edges in $\bigcup_{j=i-10g}^{i+10g} E(\beta_h^j) \setminus E(\beta_h^i)$ and all shortcut edges to and from $p \in U$ weighted by the single-source distances we have computed in G_h. See Figure 4 for an illustration.

1.3.3 Both failures on the st-path

Now let us assume both edge failures lie on π. As the same in [20], the main difficulty of dual-failure replacement path for this case comes from the backward paths. More specifically, given two edge failures $\{f_1, f_2\}$, in general the optimal dual replacement path ρ has three parts.

- A prefix of ρ that diverges from π before the first edge failure f_1 on $\pi[s, f_1)$, and then meets somewhere in the middle $y \in V(\pi(f_1, f_2))$ using edges in $G \setminus E(\pi)$.
- A sub-path of ρ that travels from y to another vertex $x \in V(\pi(f_1, y])$ using edges in $(G \setminus E(\pi)) \cup E(\pi(f_1, f_2))$; this sub-path is the so-called *backward* path of ρ which may converge and diverge multiple times with $\pi(f_1, f_2)$.
- A suffix of ρ that converges with π after the second edge failure f_2 on $\pi(f_2, t]$ starting from x using edges in $G \setminus E(\pi)$, and then reach t using the rest of π.

To compute the backward path, let us divide π into sub-paths of length L for some parameter L as $\pi = \gamma_1 \circ \gamma_2 \circ \cdots \circ \gamma_{n/L}$. Take a random pivot vertex set U of size $O(\frac{n \log n}{L})$. The main observation is that if f_1 and f_2 are in different sub-paths $\gamma_i, \gamma_j, j - i > 1$, plus that $x \in V(\gamma_i), y \in V(\gamma_j)$, then the prefix $\rho[s, y]$ must have length at least L and thus contain a pivot $p \in U$ with high probability. Therefore, if we compute single-source replacement paths [6] from p in graph $G \setminus E(\gamma_i)$, then it would provide useful information about the backward path of ρ from y to x. See Figure 5 for an illustration.

1.4 Faster algebraic algorithm for weighted digraphs

Let us divide the shortest path π from s to t into sub-paths of hops at most L; that is, $\pi = \gamma_1 \circ \gamma_2 \circ \cdots \circ \gamma_{\lceil n/L \rceil}$. Given a pair of edge failures $\{f_1, f_2\}$, suppose f_1 and f_2 belong to γ_{l_1} and γ_{l_2} respectively. In the previous paper [20], the difficult case is when f_1, f_2 come from different sub-paths, say $l_1 < l_2$. To compute $\mathsf{dist}(s, t, G \setminus \{f_1, f_2\})$ efficiently, their approach

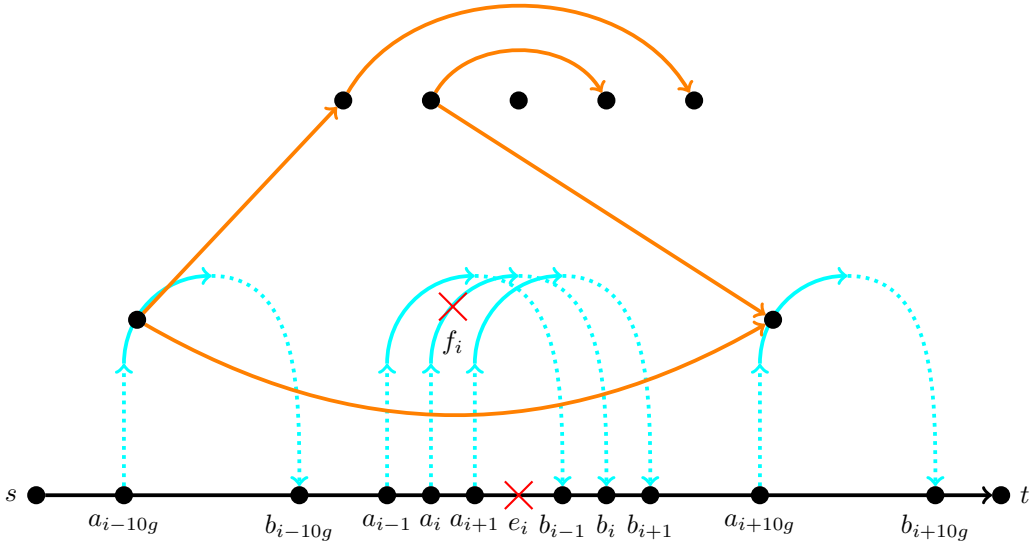


Figure 4 A shortcut graph that helps computing the dual-failure detour γ_i avoiding $\{f_i, g_i\}$, which consists of vertices in $U \cup \bigcup_{j=i-10g}^{i+10g} V(\beta_h^j)$. This shortcut graph includes all edges in $\bigcup_{j=i-10g}^{i+10g} E(\beta_h^j) \setminus E(\beta_h^i)$, and some shortcut edges representing distances to and from U in G_h; actually, it will also contain some shortcut edges between vertices in $\bigcup_{j=i-10g}^{i+10g} V(\beta_h^j)$ which we have not discussed in the overview.

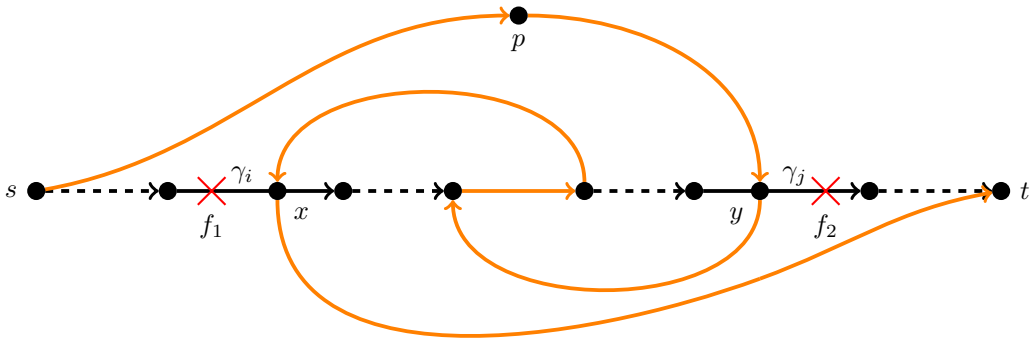


Figure 5 When f_1, f_2 lie in different sub-paths γ_i, γ_j such that $j - i > 1$, we can show that ρ must contain a vertex in U with high probability. Then we can apply the algorithm from [6] to compute single-source replacement path from p in graph $G \setminus E(\gamma_i)$ to learn about the sub-path $\rho[p, x]$ which contains the backward path in the middle.

was to build a sketch graph H_{f_1,f_2} on vertices $\{s,t\} \cup V(\gamma_{l_1}) \cup V(\gamma_{l_2})$ which encodes an optimal replacement path, and then run s-t shortest path in H_{f_1,f_2} which takes $\tilde{O}(L^2)$ time for each $\{f_1, f_2\}$, leading to a total runtime overhead of $\tilde{O}(n^2L^2)$.

Divide each sub-path γ_l into segments $\gamma_l = \alpha_1^l \circ \alpha_2^l \circ \cdots \circ \alpha_{\lceil L/g \rceil}^l$ of hops at most g (for some parameter $g < L$). Assume f_1 lies on $\alpha_{h_1}^{l_1}$ and f_2 lies on $\alpha_{h_2}^{l_2}$, and let ρ be the optimal replacement path from s to t avoiding $\{f_1, f_2\}$. Our main observation is that, if ρ intersects both $\alpha_{h_1}^{l_1}, \alpha_{h_2}^{l_2}$, then we can build a smaller sketch graph H_{f_1,f_2} on $\{s,t\} \cup V\left(\alpha_{h_1}^{l_1}\right) \cup V\left(\alpha_{h_2}^{l_2}\right)$ which still encodes ρ, and so the runtime would be reduced to $\tilde{O}(g^2)$. Otherwise, if ρ skips over $\alpha_{h_2}^{l_2}$ entirely, we will build a sketch graph $H_{f_1,(l_2,h_2)}$ on the vertex set $\{s,t\} \cup V(\gamma_{l_1}) \cup V(\gamma_{l_2})$ which only depends on f_1 and $\alpha_{h_2}^{l_2}$ and not on f_2. As the number of such graphs $H_{f_1,(l_2,h_2)}$ is at most $O(n^2/g)$, the runtime can be bounded by $\tilde{O}(n^2L^2/g)$. Overall, the runtime would be $\tilde{O}(n^2g^2 + n^2L^2/g)$ which is always better than the previous bound of $\tilde{O}(n^2L^2)$.

2 Preliminaries

Throughout the paper, logarithm $\log(*)$ will have base 2, and we assume the number of vertices n in the input graph is an integral power of 2. In any weighted digraph $G = (V, E, \omega)$, for any vertex $u \in V$, let $\deg(u, G)$ be its vertex degree (counting both in-edges and out-edges); for any pair of vertices $u, v \in V$, let $\mathsf{dist}(u, v, G)$ be the weighted length of the shortest path from u to v in G. Throughout the algorithm, we will maintain a distance estimation function $\mathsf{est}(*,*,*)$, such that the value $\mathsf{est}(u, v, G) \geq \mathsf{dist}(u, v, G)$ is always an upper bound. All values of $\mathsf{est}(*,*,*)$ are initially infinity and are non-increasing throughout the algorithms. When we update the value of an estimation $\mathsf{est}(u, v, G)$ with a distance value D, we mean $\mathsf{est}(u, v, G) \leftarrow \{D, \mathsf{est}(u, v, G)\}$.

Given any directed path or walk ρ in G, let $|\rho|$ be the number of edges on ρ, and let $\omega(\rho)$ be its total edge weight. For any two vertices $u, v \in V(\rho)$ where u comes before v on ρ, let $\rho[u, v] \subseteq \rho$ be the sub-path of ρ from u to v. In addition, let $\rho[*, v]$ and $\rho[u, *]$ be the prefix and suffix sub-path of ρ; this notation will be useful when we don't have variable names for the endpoints of path ρ.

For any edge $f \in E(\rho)$ and vertex $u \in V(\rho)$ which comes before edge f, let $\rho[u, f)$ be the sub-path from u to f (excluding edge f); similarly we can define notations $\rho(f, v]$ and $\rho(f_1, f_2)$ in the natural way.

Borrowing a terminology from [20], let us define the notion of canonical paths.

▶ **Definition 3** ([20]). *Let $s, t \in V$ be two vertices, and let π be a shortest path from s to t. For any edge set $F \subseteq E$, a path ρ from s to t in $G \setminus F$ is called* canonical *with respect to π and F, if for any $u, v \in V(\pi) \cap V(\rho)$ such that u appears before v in both π, ρ and $E(\pi[u, v]) \cap F = \emptyset$, then $\rho[u, v]$ is the same as $\pi[u, v]$.*

2.1 Unweighted digraphs

For tie-breaking among shortest paths, we can randomly perturb all unit edge weights slightly so that all replacement shortest paths for at most two edge failures are unique under the perturbed weights. We can show that, under the weight perturbation, all replacement shortest paths are canonical. Next, let us state a basic property regarding shortest replacement paths for one edge failure.

▶ **Lemma 4.** *Consider any edge $f \in E(\pi)$, any canonical shortest path ρ from s to t avoiding f can be decomposed as $\rho = \pi[s, a] \circ \alpha \circ \pi[b, t]$, where $a, b \in V(\pi)$ and α is a shortest path from a to b in $G \setminus E(\pi)$. For convenience, α will be called the* detour *of the replacement path, and a, b are called the* divergence *and* convergence *vertex, respectively.*

It is known that shortest replacement paths for one failure can be computed efficiently.

▶ **Lemma 5** ([16]). *Given an unweighted digraph $G = (V, E)$ on n vertices and m edges. Fixing any source-terminal pair $s, t \in V$, we can compute all canonical shortest replacement paths from s to t avoiding f with high probability in time $\tilde{O}(mn^{1/2} + n^2)$, where f ranges over all edges in E.*

We also need a basic property about replacement paths for dual edge failures where only one edge falls on the st-path π.

▶ **Lemma 6.** *Consider any edge $f_1 \in E(\pi)$ and let α be the detour of the shortest replacement path that avoids f_1. Consider any edge $f_2 \in E(\alpha)$. Then, there is a shortest replacement path ρ avoiding $\{f_1, f_2\}$ such that:*

- *ρ is a canonical shortest path avoiding f_1 in graph $G \setminus \{f_1\}$.*
- *ρ diverges and converges with π for once.*

We will be applying the following algorithm for single-source replacement paths algorithm from [6] as a black-box.

▶ **Lemma 7** ([6]). *Given an unweighted digraph $G = (V, E)$ on n vertices and m edges. Fixing any vertex $s \in V$, we can compute all shortest replacement paths from s to t avoiding f with high probability in time $\tilde{O}(mn^{1/2} + n^2)$, where t ranges over all vertices in $V \setminus \{s\}$ and f ranges over all edges in E.*

To be consistent with our perturbation-based tie-braking, we should also impose the same edge weight perturbation on the graph on which Lemma 7 is applied; although Lemma 7 is only stated for unweighted digraphs, it also works with edge weight perturbation (for example, all hitting set arguments still work, since perturbation only changes path lengths negligibly).

In the original paper [6], they only claim to compute all the length of shortest replacement paths, but here we need the actual shortest paths tree in each graph $G \setminus \{f\}$. To achieve such an augmentation, during the execution of the algorithm in [6], we can keep track of the last edge of each replacement path, and by uniqueness of shortest paths under weight perturbation, the set of these last edges form the shortest paths tree.

Finally, one of our basic tools is a *truncated* version of Dijkstra's algorithm, which is stated below.

▶ **Lemma 8** ([9]). *Given an unweighted digraph $G = (V, E)$ on n vertices. For any vertex $s \in V$ and any threshold L, let $U = \{u \mid \mathsf{dist}(s, u, G) \leq L\}$. Then we can compute shortest paths from s to all vertices in U in time $O(\sum_{u \in U} \deg(u, G) + n \log n)$.*

2.2 Weighted digraphs

When the input graph contains negative edge weights, we assume it does not contain any negative cycles. For weighted graphs, since fast matrix multiplication algorithms only work with small integer edge weights, we will not assume unique shortest paths by perturbing the edge weights.

Our algorithm relies on fast algorithms which computes shortest paths in digraphs with negative edge weights.

▶ **Lemma 9** ([3]). *Given a weighted digraph $G = (V, E, \omega)$ with edge weights $\omega : E \to \{-M, -M + 1, \cdots, M - 1, M\}$ without negative cycles, and fix any source vertex $s \in V$. Then, single-source shortest path from s can be computed in time $\tilde{O}(m \log M)$ with high probability.*

▶ **Lemma 10** ([22]). *Given a weighted digraph $G = (V, E, \omega)$ with edge weights $\omega : E \to \{-M, -M+1, \cdots, M-1, M\}$ without negative cycles, all-pairs shortest paths in G can be computed in time $\tilde{O}(M^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}})$ with high probability.*

We will apply single-source multi-terminal replacement paths algorithms as a black-box; the currently best known upper bound is stated below.

▶ **Lemma 11** ([13]). *Given a weighted digraph $G = (V, E, \omega)$ with edge weights $\omega : E \to \{-M, -M+1, \cdots, M-1, M\}$ without negative cycles, and fix any source vertex $s \in V$ and a terminal set $T \subseteq V$. Then, the value of all $\mathsf{dist}(s, t, G \setminus \{f\}), \forall t \in T, f \in E$ can be computed in time $\tilde{O}(Mn^{\omega} + M^{\frac{1}{4-\omega}} n^{1+\frac{1}{4-\omega}} \cdot |T|)$ with high probability.*

We will use the following fast algorithm for distance sensitivity oracles in weighted digraphs. In a weighted digraph $G = (V, E, \omega)$ with edge weights $\omega : E \to \{-M, -M+1, \cdots, M-1, M\}$, a distance sensitivity oracle is an efficient data structure that answers $\mathsf{dist}(u, v, G \setminus \{f\})$ for any $u, v \in V, f \in E$.

▶ **Lemma 12** ([5]). *Given a weighted digraph $G = (V, E, \omega)$ with edge weights $\omega : E \to \{-M, -M+1, \cdots, M-1, M\}$ without negative cycles, a distance sensitivity oracle with $\tilde{O}(1)$ query time can be constructed in time $\tilde{O}(Mn^{2.8719})$.*

3 One failure on a short st-path

We use two parameters g and L which will be determined in the end such that $g < n^{1/2} < L$. In this section, we study the case where only one edge failure lies on a short st-path, plus that the input graph is sparse and $\mathsf{dist}(s, t, G) \leq L$. More specifically, let $G = (V, E)$ be an unweighted digraph with n vertices and m edges, and consider a pair of vertices s, t with a shortest st-path $E(\pi)$ of length at most L. The task is to compute for any pairs of edges f_1, f_2 the value of $\mathsf{dist}(s, t, G \setminus \{f_1, f_2\})$, where $f_1 \in E(\pi), f_2 \notin E(\pi)$. The purpose of this section is to prove the following lemma.

▶ **Lemma 13.** *Let $G = (V, E)$ be an unweighted digraph with n vertices and m edges. Fix a pair of source and terminal $s, t \in V$ such that $\mathsf{dist}(s, t, G) \leq L$. Then, all values of $\mathsf{dist}(s, t, G \setminus \{f_1, f_2\})$ can be computed with high probability in time:*

$$\tilde{O}\left(\frac{mn^{1.5} + n^3}{L} + mn^{1/2}L/g + n^2L/g + mL^2/g + mnL^2/g^3 + mLg + L^2g^4 + n^2L^2/g^2\right)$$

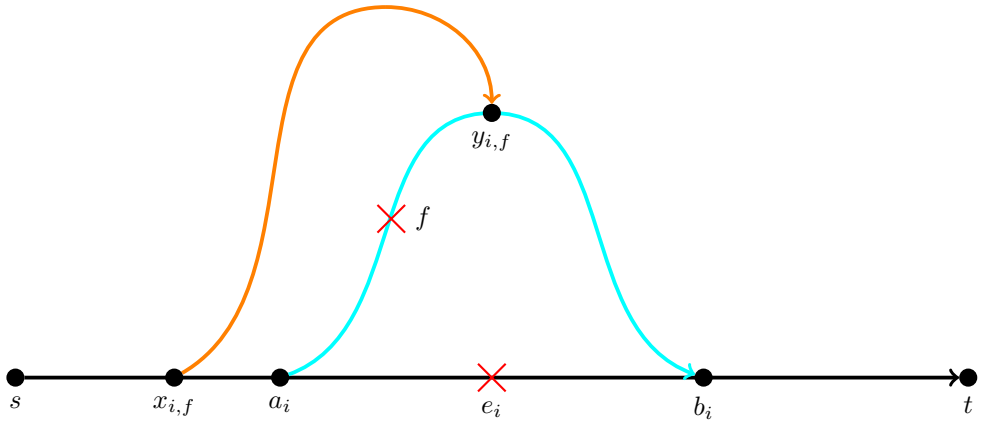
when $f_1 \in E(\pi)$ while $f_2 \notin E(\pi)$; here g is an arbitrary parameter to be determined later.

Let $\pi = \langle s = u_0, u_1, \cdots, u_{|\pi|} = t \rangle$. First, we use Lemma 5 to compute for each $(u_i, u_{i+1}) \in E(\pi)$ the replacement path from s to t that avoids $e_i = (u_i, u_{i+1})$, and let α_i be the corresponding detour avoiding e_i which starts at a_i and ends at b_i.

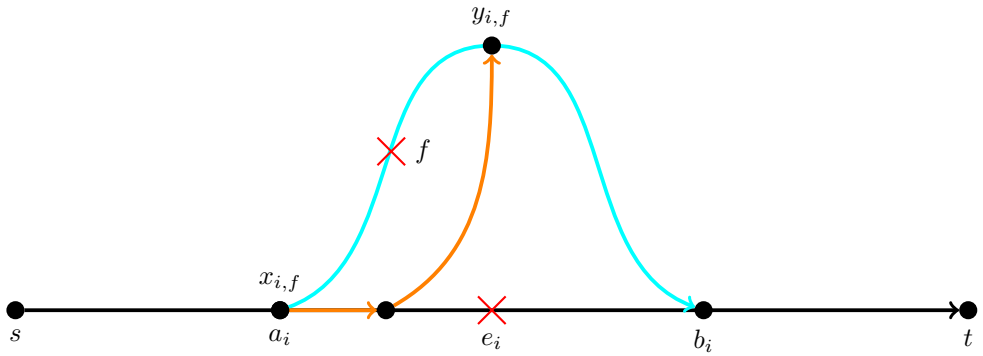
For each detour α_i, divide α_i into sub-paths of length g (except for the last sub-path) and list them as $\beta_1^i, \cdots, \beta_{l_i}^i$, and assume $f \in E(\beta_l^i)$; later on, when we refer to β_l^i, if $l \leq 0$ or $l > l_i$, then β_l^i would simply be an empty path.

Throughout the algorithm, for each $0 \leq i < |\pi|$ and every edge $f \in E(\alpha_i)$, we will maintain a distance estimation $\mathsf{est}(s, t, G \setminus \{e_i, f\}) \geq \mathsf{dist}(s, t, G \setminus \{e_i, f\})$, and in the end it will be guaranteed that $\mathsf{est}(s, t, G \setminus \{e_i, f\}) = \mathsf{dist}(s, t, G \setminus \{e_i, f\})$.

Let $\rho_{i,f}$ be a canonical shortest replacement path for $\{e_i, f\}$. Suppose $\rho_{i,f}$ departs from $\pi[s, a_i] \circ \alpha_i \circ \pi[b_i, t]$ at vertex $x_{i,f}$ and converges with $\pi[s, a_i] \circ \alpha_i \circ \pi[b_i, t]$ at vertex $y_{i,f}$. For the rest, we address different cases depending on the properties regarding $\alpha_i, \rho_{i,f}$; note that our algorithm does not need to know which case it is in advance.



(a) In this case, $x_{i,f} \neq a_i$ and $\rho_{i,f}[x_{i,f}, *]$ diverges from π immediately.



(b) In this case, $x_{i,f} = a_i$ and $\rho_{i,f}[x_{i,f}, *]$ uses some edges on $\pi[a_i, b_i]$ before it diverges.

Figure 6 In this picture, the cyan path is detour α_i, and the orange path is $\rho_{i,f}[x_{i,f}, y_{i,f}]$.

Case 1: $x_{i,f} \in V(\pi[s, a_i])$ or $y_{i,f} \in V(\pi[b_i, t])$

This is an easy case of our algorithm, and the runtime of this part can be bounded by $O(mL)$. See Figure 6 for an illustration. Check the full version for more details.

Case 2: $x_{i,f}, y_{i,f} \in V(\alpha_i)$ and $|\alpha_i| \geq L$

This is an easy case of our algorithm, and the runtime of this part can be bounded by $\tilde{O}(\frac{mn^{1.5}+n^3}{L})$. See Figure 7 for an illustration. Check the full version for more details.

Case 3: $x_{i,f}, y_{i,f} \in V(\alpha_i)$, plus that $|\pi[a_i, u_i]| > g$ or $|\pi[u_{i+1}, b_i]| > g$

This is an easy case of our algorithm, and the runtime of this part can be bounded by

$$\tilde{O}\left(m\sqrt{n}L/g + n^2L/g + L^2\right)$$

See Figure 8 for an illustration. Check the full version for more details.

Case 4: $x_{i,f}, y_{i,f} \in V(\alpha_i)$, plus that $x_{i,f} \notin V(\beta_l^i)$ or $y_{i,f} \notin V(\beta_l^i)$, and $|\alpha_i| < L$

This is an easy case of our algorithm, and the runtime of this part can be bounded by $\tilde{O}(mL^2/g)$. See Figure 9 for an illustration. Check the full version for more details.

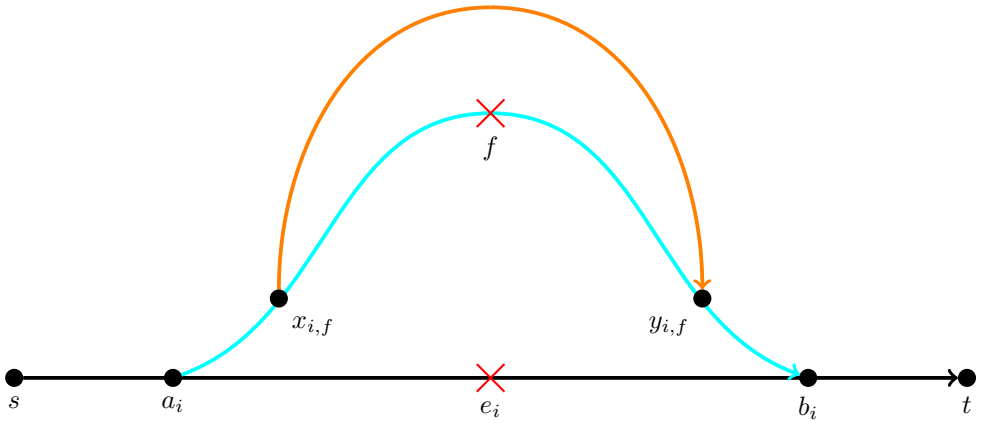


Figure 7 The sub-path $\rho_{i,f}[a_i, b_i]$ has length greater than L.

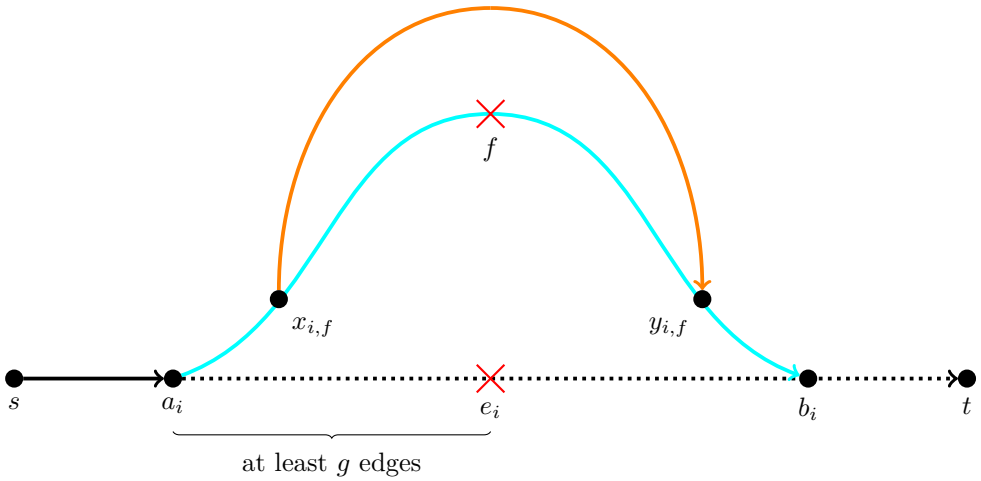


Figure 8 In this picture, $x_{i,f}, y_{i,f} \in V(\alpha_i)$ plus that $|\pi[a_i, u_i]| > g$.

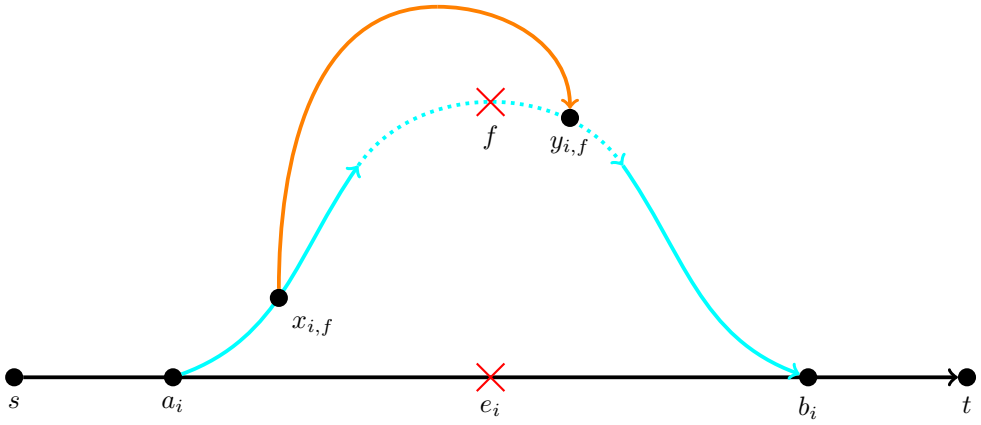


Figure 9 The dotted cyan path is β_l^i.

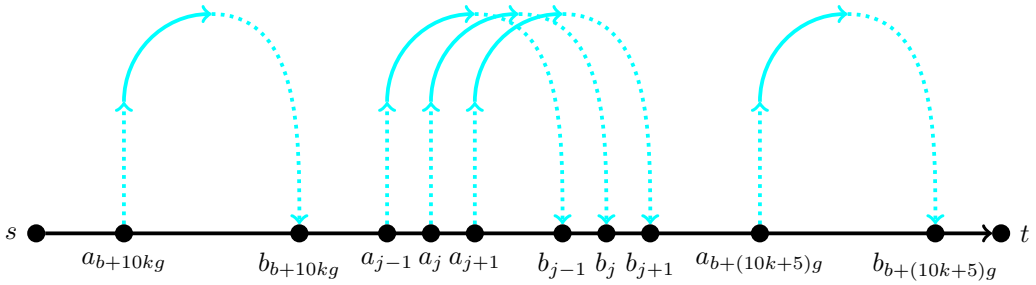


Figure 10 Graph X_l^{b+10kg} excludes all edges in $E(\pi) \cup \bigcup_{j=b+10kg, j\in I}^{b+(10k+5)g} E(\beta_l^j)$ which are drawn and black and cyan solid paths.

Main case: $|\pi[a_i, u_i]|, |\pi[u_{i+1}, b_i]| \leq g$, and $x_{i,f}, y_{i,f} \in V(\beta_l^i)$, and $|\alpha_i| < L$

Algorithm Main case.

Let $I \subseteq [L]$ be the set of all indices such that $|\pi[a_i, u_i]|, |\pi[u_{i+1}, b_i]| \leq g$ and $|\alpha_i| < L$. For each pair of indices $(p, q) \in [L/g] \times [L/g]$, define the set of sub-paths:

$$\mathcal{P}_{p,q} = \{\beta_l^i \mid (l, l_i - l + 1) = (p, q), i \in I\}$$

(1) Let $U \subseteq V$ be the uniformly random subset of size $\frac{10n \log n}{g}$. Then, for each pair of indices $p, q \in [L/g]$, define the graph:

$$G_{p,q} = G \setminus \left(E(\pi) \cup \bigcup_{\beta \in \mathcal{P}_{p,q}} E(\beta) \right)$$

(2) Then, for each vertex $u \in U$, compute single-source shortest paths to and from u in $G_{p,q}$. For any index $1 \leq l \leq \lceil L/g \rceil$, and for any offset $1 \leq b \leq 10g$, initialize two sets of vertices $A_{b,l}, B_{b,l} \leftarrow V$. Then, go over the sequence of all sub-paths $\beta_l^b, \beta_l^{b+10g}, \cdots, \beta_l^{b+10hg}$, where $h \leq \lceil \frac{L}{10g} \rceil$.

Next, for each sub-path β_l^{b+10kg}, define the following two graphs

$$X_l^{b+10kg} = G \setminus \left(E(\pi) \cup \bigcup_{j \in [b+10kg, b+(10k+5)g] \cap I} E(\beta_l^j) \right)$$

$$Y_l^{b+10kg} = G \setminus \left(E(\pi) \cup \bigcup_{j \in [b+(10k-5)g, b+10kg] \cap I} E(\beta_l^j) \right)$$

See Figure 10 for an illustration.

(a) Then, for each vertex $v \in V\left(\beta_l^{b+10kg}\right)$, perform a truncated Dijkstra at v in the induced subgraph $X_l^{b+10kg}[A_{b,l}]$ up to depth g. For each $z \in A_{b,l}$, record the distance value

$$\mu_X(v, z) \leftarrow \mathsf{dist}\left(v, z, X_l^{b+10kg}[A_{b,l}]\right)$$

if $\mathsf{dist}\left(v, z, X_l^{b+10kg}[A_{b,l}]\right) \leq g$.

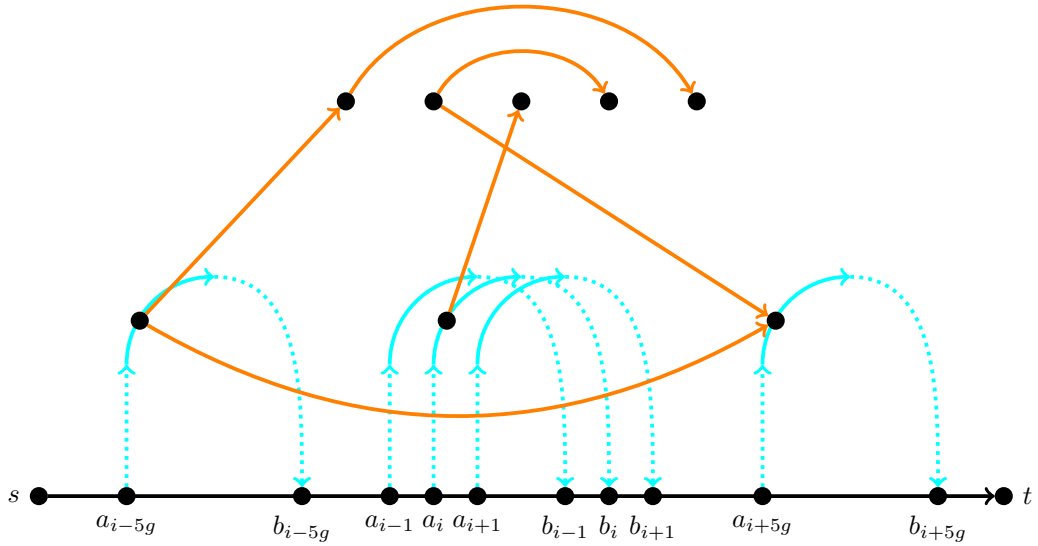


Figure 11 Vertices on the topmost are in U, and orange edges represent shortcut edges in H_l^i.

After we have visited all vertices $v \in V\left(\beta_l^{b+10kg}\right)$, let $P_{b,l}^k \subseteq A_{b,l}$ be the set of all vertices searched by truncated Dijkstra of any $v \in V\left(\beta_l^{b+10kg}\right)$. Then, prune the vertex set

$$A_{b,l} \leftarrow A_{b,l} \setminus P_{b,l}^k$$

(b) Symmetrically, for each vertex $v \in V\left(\beta_l^{b+10kg}\right)$, perform a truncated Dijkstra at v in the induced subgraph $Y_l^{b+10kg}[B_{b,l}]$ up to depth g. For each $z \in B_{b,l}$, record the distance value

$$\mu_Y(v,z) \leftarrow \mathsf{dist}\left(v, z, Y_l^{b+10kg}[B_{b,l}]\right)$$

if $\mathsf{dist}\left(v, z, Y_l^{b+10kg}[B_{b,l}]\right) \leq g$.

After we have visited all vertices $v \in V\left(\beta_l^{b+10kg}\right)$, let $Q_{b,l}^k \subseteq B_{b,l}$ be the set of all vertices searched by truncated Dijkstra of any $v \in V\left(\beta_l^{b+10kg}\right)$. Then, prune the vertex set

$$B_{b,l} \leftarrow B_{b,l} \setminus Q_{b,l}^k$$

(3) For any index $i \in I$ and index $1 \leq l \leq l_i/g$, let us build a shortcut digraph H_l^i with edge weight function ω in the following manner. See Figure 11 for an illustration.

- **Vertices.** Add all vertices in sub-paths $V(\beta_l^j), \forall j \in [i-5g, i+5g] \cap I$, as well as all pivot vertices in U to H_l^i.
- **Edges.** First, add to $E(H_l^i)$ all the sub-paths:

$$\left(\bigcup_{j \in [i-5g, i+5g] \cap I} E\left(\beta_l^j\right)\right) \setminus E\left(\beta_l^i\right)$$

Then, add the following three types of weighted edges.

(a) For any $u \in U$ and any vertex $v \in V(H_l^i)$, add an edge (u, v) with edge weight:

$$\omega(u, v) = \mathsf{dist}(u, v, G_{l,l_i-l+1})$$

and edge (v, u) with edge weight:

$$\omega(v, u) = \mathsf{dist}(v, u, G_{l,l_i-l+1})$$

(b) For any pair of vertices $u, v \in V(H_l^i)$ where $u \in V(\beta_l^j), j \in [i-5g, i]$, add an edge (u, v), and assign a weight:

$$\omega(u, v) = \mu_X(u, v)$$

If $\mu_X(u, v)$ was not assigned in Step (2), then it is infinity by default.

(c) For any pair of vertices $u, v \in V(H_l^i)$ where $u \in V(\beta_l^j), j \in [i, i+5g]$, add an edge (u, v), and assign a weight:

$$\omega(u, v) = \mu_Y(u, v)$$

If $\mu_Y(u, v)$ was not assigned in Step (2), then it is infinity by default.

After that, for each vertex $z \in V\left(\beta_l^i\right)$, apply single-source shortest path on z in H_l^i. In this way, for every pair of vertices $x, y \in V\left(\beta_l^i\right)$, we have computed a distance $\mathsf{dist}\left(x, y, H_l^i\right)$.

Finally, for every edge $f \in E\left(\beta_l^i\right)$, update the estimation $\mathsf{est}(s, t, G \setminus \{e_i, f\})$ with:

$$\min_{x \in V(\beta_l^i[*,f)), y \in V(\beta_l^i(f,*])} \left\{\mathsf{dist}(s, x, G \setminus \{e_i\}) + \mathsf{dist}\left(x, y, H_l^i\right) + \mathsf{dist}(y, t, G \setminus \{e_i\})\right\}$$

Runtime. The runtime of Step (1) is bounded by $\tilde{O}(|U| \cdot mL^2/g^2) = \tilde{O}(\frac{mnL^2}{g^3})$. As for the runtime of Step (2), consider any offset $1 \le b \le 10g$. We argue that the Dijkstra procedure for all vertices on sub-paths $\beta_l^b, \beta_l^{b+10g}, \cdots, \beta_l^{b+10hg}$ has runtime at most $O(mg)$; this is because every vertex in V is visited by at most $O(g)$ instances of Dijkstra rooted at vertices from some sub-paths β_l^{b+10kg}. Therefore, the overall runtime of Step (2) is bounded by $O(mLg)$ summing over all $1 \le b \le 10g$ and $1 \le l \le L/g$.

Finally, let us estimate the runtime of Step (3). For each index $i \in I$, there are at most L/g different sub-paths β_l^i as $|\alpha_i| \le L$. By definition, the shortcut digraph H_i contains at most $O(g^2 + \frac{n \log n}{g})$ vertices, and so the number of edges within is bounded by $\tilde{O}(g^4 + \frac{n^2}{g^2})$, and hence the runtime of multi-source shortest paths computation for all vertices in $V(\beta_l^i)$ in H_l^i is $\tilde{O}(g^5 + \frac{n^2}{g})$. Then, for each $f \in E(\beta_l^i)$, the time of calculating the formula

$$\min_{x \in V(\beta_l^i[*,f)), y \in V(\beta_l^i(f,*])} \{\mathsf{dist}(s, x, G \setminus \{e_i\}) + \mathsf{dist}(x, y, H_i) + \mathsf{dist}(y, t, G \setminus \{e_i\})\}$$

is bounded by $O(g^2)$. Summing over all i, l and $f \in E(\beta_l^i)$, the runtime of this part is bounded by $\tilde{O}(L^2 g^4 + \frac{n^2 L^2}{g^2})$.

Taking a summation, the overall runtime for this case is at most:

$$\tilde{O}\left(mnL^2/g^3 + mLg + L^2 g^4 + n^2 L^2/g^2\right)$$

Correctness. To prove that our algorithm computes the correct value for $|\rho_{i,f}|$, it suffices to prove that $\mathsf{dist}\left(x_{i,f}, y_{i,f}, H_l^i\right) = |\rho_{i,f}[x_{i,f}, y_{i,f}]|$. First we argue that $\mathsf{dist}\left(x_{i,f}, y_{i,f}, H_l^i\right) \geq |\rho_{i,f}[x_{i,f}, y_{i,f}]|$ due to the following reason.

▷ Claim 14. Any weighted edge (u, v) in H_l^i corresponds to a path from u to v in G that does not contain any edge in $E(\pi) \cup E(\beta_l^i)$.

Proof. If the weighted edge (u, v) was defined on Step (3)(a), then it corresponds to a shortest path in G_{l,l_i-l+1} which excludes all edges in $E(\pi) \cup E(\beta_l^i)$.

If the weighted edge (u, v) was defined on Step (3)(b), suppose $u \in V(\beta_l^j)$ where $j \in [i-5g, i]$, then by definition of $\omega(u, v) = \mu_X(u, v)$, which is equal to the length of a path in graph X_l^j which excludes all edges in $E(\pi) \cup E(\beta_l^i)$.

Symmetrically, if the weighted edge (u, v) was defined on Step (3)(c), suppose $u \in V(\beta_l^j)$ where $j \in [i, i+5g]$, then by definition of $\omega(u, v) = \mu_Y(u, v)$, it is equal to the length of a path in graph Y_l^j which excludes all edges in $E(\pi) \cup E(\beta_l^i)$. ◁

For the rest, let us prove that $\mathsf{dist}\left(x_{i,f}, y_{i,f}, H_l^i\right) \leq |\rho_{i,f}[x_{i,f}, y_{i,f}]|$. To do this, we will find a path in H_l^i from $x_{i,f}$ to $y_{i,f}$ with weighted length at most $|\rho_{i,f}[x_{i,f}, y_{i,f}]|$.

▷ Claim 15. $\rho_{i,f}[x_{i,f}, y_{i,f}]$ does not contain any vertices in the following vertex subset:

$$\bigcup_{j=0}^{i-5g-1} V\left(\beta_l^j\right) \cup \bigcup_{j=i+5g+1}^{L} V\left(\beta_{l_j-l_i+l}^j\right)$$

Proof. Suppose otherwise that the sub-path $\rho_{i,f}[x_{i,f}, y_{i,f}]$ contains a vertex $v \in \bigcup_{j=0}^{i-5g-1} V(\beta_l^j) \cup \bigcup_{j=i+5g+1}^{L} V(\beta_{l_j-l_i+l}^j)$. Let us assume that $v \in V(\beta_l^j)$ for some index $0 \leq j < i-5g$; if $v \in V(\beta_{l_j-l_i+l}^j)$ for some $j > i+5g$, the proof will be similar.

By the assumption that $i \in I$, we know that $|\pi[a_i, u_i]| \leq g$. Since $j < i-5g$, we know that vertex u_j lies between s, a_i, and consequently $|\pi[s, a_j]| \leq |\pi[s, u_j]| < |\pi[s, a_i]| - 4g$.

Consider the path $\pi[s, a_j] \circ \alpha_j[a_j, v]$. We first argue that this path does not contain edges from $\{e_i, f\}$. Clearly, $\pi[s, a_j] \circ \alpha_j[a_j, v]$ does not contain the edge e_i, since a_j lies between s and u_i, and $E(\alpha_j[a_j, v]) \cap E(\pi) = \emptyset$. As for the position of f, if $f \in E(\alpha_j[a_j, v])$, then there must be a vertex $z \in V(\alpha_j[a_j, v]) \cap V(\beta_l^i)$. Then $\pi[s, a_j] \circ \alpha_j[a_j, z] \circ \alpha_i[z, b_i] \circ \alpha_i[b_i, t]$ is a replacement path that avoids edge e_i, with length at most:

$$\begin{aligned}
&|\pi[s, a_j]| + |\alpha_j[a_j, z]| + |\alpha_i[z, b_i]| + |\alpha_i[b_i, t]| \\
&< (|\pi[s, a_i]| - 4g) + |\alpha_j[a_j, v]| + |\alpha_i[z, b_i]| + |\alpha_i[b_i, t]| \\
&\leq (|\pi[s, a_i]| - 4g) + l \cdot g + |\alpha_i[z, b_i]| + |\alpha_i[b_i, t]| \\
&\leq (|\pi[s, a_i]| - 4g) + (|\alpha_i[a_i, z]| + g) + |\alpha_i[z, b_i]| + |\alpha_i[b_i, t]| \\
&\leq |\pi[s, a_i] + |\alpha_i[a_i, z]| + |\alpha_i[z, b_i]| + |\alpha_i[b_i, t]| - 3g = |\pi[s, a_i] \circ \alpha_i \circ \pi[b_i, t]| - 3g
\end{aligned}$$

which is a contradiction that $\pi[s, a_i] \circ \alpha_i \circ \pi[b_i, t]$ is a shortest replacement path avoiding e_i.

Next, we argue that $|\pi[s, a_j] \circ \alpha_j[a_j, v]| < |\rho_{i,f}[s, x_{i,f}]| < |\rho_{i,f}[s, v]|$. In fact, by $|\pi[s, a_j]| < |\pi[s, a_i]| - 4g$, we have:

$$\begin{aligned}
&|\pi[s, a_j]| + |\alpha_j[a_j, v]| < (|\pi[s, a_i]| - 4g) + l \cdot g \\
&\leq (|\pi[s, a_i]| - 4g) + (|\alpha_i[a_i, x_{i,f}]| + g) = |\rho_{i,f}[s, x_{i,f}]| - 3g
\end{aligned}$$

As we have proved, $\pi[s, a_j] \circ \alpha_j[a_j, v]$ is a path avoiding $\{e_i, f\}$ of length less than $|\rho_{i,f}[s, v]|$. So, if we replace the prefix $\rho_{i,f}[s, v]$ with $\pi[s, a_j] \circ \alpha_j[a_j, v]$ and consider a new path:

$$\rho' = \pi[s, a_j] \circ \alpha_j[a_j, v] \circ \rho_{i,f}[v, t]$$

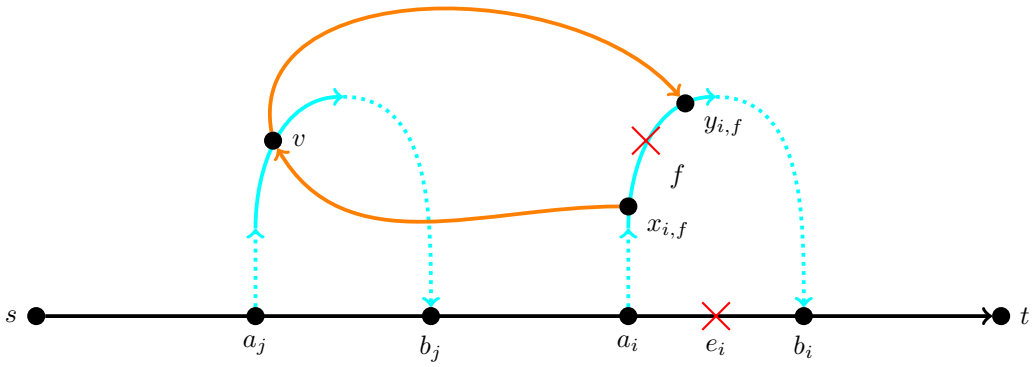


Figure 12 The sub-path $\rho_{i,f}[x_{i,f}, y_{i,f}]$ is drawn as the orange curve. If $\rho_{i,f}[x_{i,f}, y_{i,f}]$ contains a vertex $v \in V\left(\beta_l^j\right)$, then the alternate path $\pi[s, a_j] \circ \alpha_j[*, v] \circ \rho_{i,f}[v, y_{i,f}] \circ \alpha_i[y_{i,f}, b_i] \circ \pi[b_i, t]$ be a shorter replacement path than $\rho_{i,f}$ avoiding $\{e_i, f\}$.

then we have a new replacement path ρ' avoiding $\{e_i, f\}$ from s to t with a strictly better distance, a contradiction. See Figure 12 for an illustration.

◁

Decompose the path $\rho_{i,f}[x_{i,f}, y_{i,f}]$ into minimal sub-paths whose endpoints are belonging to $V(H_l^i)$; this is achievable because both $x_{i,f}, y_{i,f} \in V(H_l^i)$. To prove the upper bound:

$$\mathsf{dist}(x_{i,f}, y_{i,f}, H_l^i) \leq |\rho_{i,f}[x_{i,f}, y_{i,f}]|$$

it suffices to show that for any two consecutive vertices $u, v \in V(H_l^i)$ on path $\rho_{i,f}[x_{i,f}, y_{i,f}]$, we have $\omega(u, v) \leq |\rho_{i,f}[u, v]|$. Consider several cases below.

- One of u, v belongs to the pivot set U.
  Without loss of generality, assume that $u \in U$. It suffices to show that the shortest path from u to v in graph G_{l,l_i-l+1} is has the same length as $\rho_{i,f}[u, v]$.
  Since v is the next vertex in $V(H_l^i)$ after u on the path $\rho_{i,f}$, the sub-path $\rho_{i,f}[u, v]$ does not contain any vertices from $V(H_l^i)$ except for its endpoints; in other words, $\rho_{i,f}[u, v]$ does not contain vertices from $U \cup \bigcup_{j=i-5g}^{i+5g} V(\beta_l^j)$. Additionally, according to Claim 15, $\rho_{i,f}[u, v]$ does not contain (internally) any vertices in

$$\bigcup_{j=0}^{i-5g-1} V\left(\beta_l^j\right) \cup \bigcup_{j=i+5g+1}^{L} V\left(\beta_{l_j-l_i+l}^j\right)$$

  Therefore, $\rho_{i,f}[u, v]$ does not contain (internally) any vertices from the set:

$$U \cup \bigcup_{j \in [i-5g, i+5g] \cap I} V\left(\beta_l^j\right) \cup \bigcup_{j=0}^{i-5g-1} V\left(\beta_l^j\right) \cup \bigcup_{j=i+5g+1}^{L} V\left(\beta_{l_j-l_i+l}^j\right) \supseteq U \cup \bigcup_{\beta \in \mathcal{P}_{l,l_i-l+1}} V(\beta)$$

  By definition of graph G_{l,l_i-l+1}, we know that $\mathsf{dist}(u, v, G_{l,l_i-l+1}) \leq |\rho_{i,f}[u, v]|$.
- Both of u, v are not in U.
  Assume that $u \in V(\beta_l^c)$ for some $c \in [i - 5g, i + 5g]$, and $v \in V(\beta_l^d)$ for some $d \in [i - 5g, i + 5g]$. Without loss of generality, assume that $c \leq i$; if $c \geq i$, a symmetric argument would work.

Since v is the next vertex in $V(H_l^i)$ after u on the path $\rho_{i,f}$ and that U is a uniformly random vertex set of size $\frac{10n\log n}{g}$, with high probability over the randomness of U, we must have $|\rho_{i,f}[u,v]| \leq g$.

Similar to the previous case, we know that $\rho_{i,f}[u,v]$ does not contain (internally) vertices from $\bigcup_{j=i-5g}^{i+5g} V\left(\beta_l^j\right)$. Therefore, by definition of X_l^c, we know that $\rho_{i,f}[u,v] \subseteq X_l^c$; namely, the sub-path $\rho_{i,f}[u,v]$ belongs to graph X_l^c. Therefore, to prove that $\omega(u,v) = \mu_X(u,v) \leq |\rho_{i,f}[u,v]|$, it suffices to show that the truncated Dijkstra procedure correctly computes the value $\mu_X(u,v) = \mathsf{dist}(u,v,X_l^c)$.

Decompose the integer $c = b + 10kg$, where $1 \leq b \leq 10g$, $k \geq 0$. To prove that the truncated Dijkstra procedure correctly computes the value $\mu_X(u,v) = \mathsf{dist}(u,v,X_l^c)$, it suffices to show that the vertex set $A_{b,l}$ contains all vertices on $\rho_{i,f}[u,v]$ when u is performing a truncated Dijkstra in the induced subgraph $X_l^c[A_{b,l}]$; in other words, we need to show that any vertex on $\rho_{i,f}[u,v]$ has not been pruned from $A_{b,l}$ by truncated Dijkstraes from vertices on previous sub-paths $\beta_l^b, \beta_l^{b+10g}, \cdots, \beta_l^{b+10(k-1)g}$.

Assume otherwise there is a vertex $z \in V(\rho_{i,f}[u,v])$ which was also visited by the truncated Dijkstra of some vertices $w \in V(\beta_l^{b+10jg})$ for some $j < k$. As all Dijkstra searches are truncated up to depth g, we know that there is a path γ from w to z in X_l^{b+10jg} of length at most g. Consider the path

$$\theta = \pi[s, a_{b+10jg}] \circ \alpha_{b+10jg}[*, w] \circ \gamma \circ \rho_{i,f}[z,v] \circ \alpha_d[v, b_d] \circ \pi[b_d, t]$$

and claim two properties of it.

▷ Claim 16. θ is a path from s to t that avoids the edge e_d.

Proof. It is clear that path θ departs from π at vertex a_{b+10jg} and converges with π at vertex b_d. So it suffices to show that a_{b+10jg} lies between s and vertex u_d. This is straightforward since a_{b+10jg} lies on path $\pi[s, u_{b+10jg}]$ which is strictly a prefix of $\pi[s,u_d]$, as $d \geq i - 5g \geq b + (10k-5)g > b + 10jg$. ◁

To reach a contradiction, we show that $|\theta|$ is a strictly better replacement path from s to t that avoids e_d than path $\pi[s,a_d] \circ \alpha_d \circ \pi[b_d,t]$. In fact, on the one hand, since $d \in I$, we know that $|\pi[a_d, u_d]| \leq g$. Therefore,

$$|\pi[s, a_{b+10jg}]| \leq |\pi[s, u_{b+10jg}]| = b + 10jg \leq 10(k-1)g \leq i - 10g \leq d - 5g \leq |\pi[s,a_d]| - 4g$$

On the other hand, since $v \in V(\beta_l^d)$ and $w \in V(\beta_l^{b+10jg})$, we know that

$$|\alpha_d[*,v]| \geq (l-1)g + 1 > |\alpha_{b+10jg}[*,w]| - g$$

Finally, as $|\gamma|, |\rho_{i,f}[z,v]| \leq g$, we have:

$$\begin{aligned}
|\theta| &\leq (|\pi[s,a_d]| - 4g) + (|\alpha_d[*,v]| + g) + |\gamma| + |\rho_{i,f}[z,v]| + |\alpha_d[v,b_d]| + |\pi[b_d,t]| \\
&\leq |\pi[s,a_d]| + |\alpha_d[*,v]| + |\alpha_d[v,b_d]| + |\pi[b_d,t]| - g \\
&= |\pi[s,a_d] \circ \alpha_d \circ \pi[b_d,t]| - g
\end{aligned}$$

which contradicts the fact that $\pi[s,a_d] \circ \alpha_d \circ \pi[b_d,t]$ is the shortest replacement path avoiding e_d. See Figure 13 for an illustration.

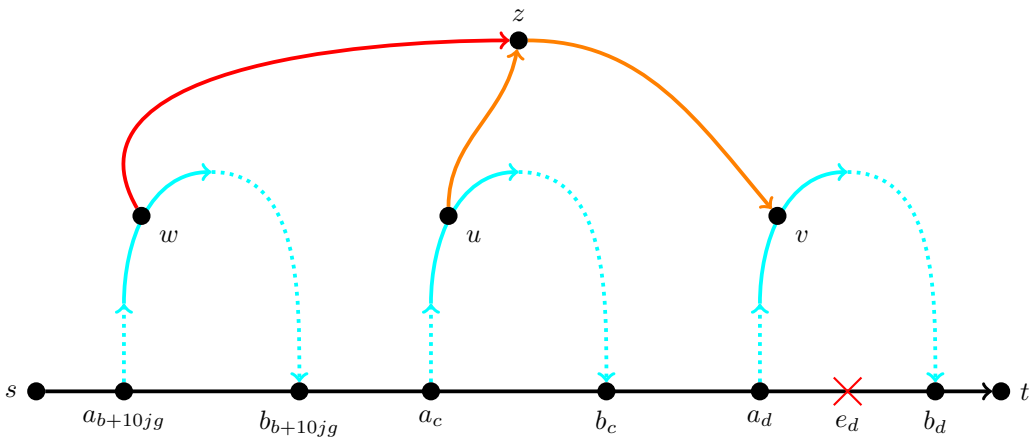


Figure 13 If $\mu_X(u,v)$ does not capture the orange path $\rho_{i,f}[u,v]$, then a previous Dijkstra search from vertex w must have intercepted $\rho_{i,f}[u,v]$ at a vertex z through a path γ drawn as the red curve. In this case, $\alpha_{b+10jg}[*,w] \circ \gamma \circ \rho_{i,f}[z,v] \circ \alpha_d[v,b_d]$ would be a better detour than α_d for avoiding e_d.

Proof of Lemma 13

Summarizing the total runtime of all five cases, the overall runtime is bounded by as following:

$$\tilde{O}\left(\frac{mn^{1.5}+n^3}{L} + mn^{1/2}L/g + n^2L/g + mL^2/g + mnL^2/g^3 + mLg + L^2g^4 + n^2L^2/g^2\right)$$

4 One failure on a long st-path

In this section, we study the case where only one edge failure lies on the shortest path, plus that the input graph is dense and $\mathsf{dist}(s,t,G)$ could be as large as $O(n)$. Let $G=(V,E)$ be a digraph with n vertices, and consider a pair of vertices s,t as well as an st-shortest path $\pi = \langle s = u_0, u_1, u_2, \cdots, u_{|\pi|} = t\rangle$. The task is to compute for any pairs of edges f_1, f_2, the value of $\mathsf{dist}(s,t,G\setminus\{f_1,f_2\})$ where $f_1 \in E(\pi), f_2 \notin E(\pi)$. The following lemma is proved in the full version.

▶ **Lemma 17.** *All values of* $\mathsf{dist}(s,t,G\setminus\{f_1,f_2\})$ *can be computed with high probability in time* $\tilde{O}\left(n^{3-1/18}\right)$ *when* $f_1 \in E(\pi)$ *while* $f_2 \notin E(\pi)$.

5 Both failures on the st-path

In this section, we study the case where both edge failures f_1, f_2 are lying the shortest path. Let $G=(V,E)$ be a digraph with n vertices, and consider a pair of vertices s,t as well as an st-shortest path $\pi = \langle s = u_0, u_1, u_2, \cdots, u_{|\pi|} = t\rangle$. For convenience, define $H = G\setminus E(\pi)$. The task is to compute for any pairs of edges $f_1, f_2 \in V(\pi)$, the value of $\mathsf{dist}(s,t,G\setminus\{f_1,f_2\})$. The following lemma is proved in the full version.

▶ **Lemma 18.** *All values of* $\mathsf{dist}(s,t,G\setminus\{f_1,f_2\})$ *can be computed in time* $\tilde{O}(n^{3-1/7})$ *when both edges* f_1, f_2 *are on* π.

Proof of Theorem 1. This is a direct combination of Lemma 17 and Lemma 18. ◀

References

1. Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.
2. Aaron Bernstein. A nearly optimal algorithm for approximating replacement paths and k shortest simple paths in general graphs. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 742–755. SIAM, 2010.
3. Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. Negative-weight single-source shortest paths in near-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 600–611. IEEE, 2022.
4. Amit M Bhosle and Teofilo F Gonzalez. Replacement paths for pairs of shortest path edges in directed graphs. In *Proceedings of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems*. Citeseer, 2004.
5. Shiri Chechik and Sarel Cohen. Distance sensitivity oracles with subcubic preprocessing time and fast query time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1375–1388, 2020.
6. Shiri Chechik and Ofer Magen. Near optimal algorithm for the directed single source replacement paths problem. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.
7. Shiri Chechik and Moran Nechushtan. Simplifying and unifying replacement paths algorithms in weighted directed graphs. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.
8. Shiri Chechik and Tianyi Zhang. Nearly optimal approximate dual-failure replacement paths. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2568–2596. SIAM, 2024.
9. Edsger W Dijkstra. A note on two problems in connexion with graphs. In *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, pages 287–290. 2022.
10. Ran Duan, Hongxun Wu, and Renfei Zhou. Faster matrix multiplication via asymmetric hashing. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2129–2138. IEEE, 2023.
11. Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 748–757. IEEE, 2012.
12. Fabrizio Grandoni and Virginia Vassilevska Williams. Faster replacement paths and distance sensitivity oracles. *ACM Transactions on Algorithms (TALG)*, 16(1):1–25, 2019.
13. Yuzhou Gu, Adam Polak, Virginia Vassilevska Williams, and Yinzhan Xu. Faster monotone min-plus product, range mode, and single source replacement paths. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
14. François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*, pages 296–303, 2014.
15. Liam Roditty. On the k-simple shortest paths problem in weighted directed graphs. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 920–928. Citeseer, 2007.
16. Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. In *Automata, Languages and Programming: 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005. Proceedings 32*, pages 249–260. Springer, 2005.
17. Virginia Vassilevska Williams. Faster replacement paths. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1337–1346. SIAM, 2011.

18 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 887–898, 2012.

19 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 645–654. IEEE, 2010.

20 Virginia Vassilevska Williams, Eyob Woldeghebriel, and Yinzhan Xu. Algorithms and lower bounds for replacement paths under multiple edge failure. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 907–918. IEEE, 2022.

21 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3792–3835. SIAM, 2024.

22 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM (JACM)*, 49(3):289–317, 2002.

Path-Reporting Distance Oracles with Logarithmic Stretch and Linear Size

Shiri Chechik
Tel Aviv University, Israel

Tianyi Zhang
Tel Aviv University, Israel

Abstract

Given an undirected graph $G = (V, E, \mathbf{w})$ on n vertices with positive edge weights, a distance oracle is a space-efficient data structure that answers pairwise distance queries in fast runtime. The quality of a distance oracle is measured by three parameters: space, query time, and stretch. In a landmark paper by [Thorup and Zwick, 2001], they showed that for any integer parameter $k \geq 1$, there exists a distance oracle with size $O(kn^{1+1/k})$, $O(k)$ query time, and $(2k-1)$-stretch error on the approximate distances. After that, there has been a line of subsequent improvements which culminated in the optimal trade-off of $O(n^{1+1/k})$ space, $O(1)$ query time, and $(2k-1)$-stretch [Chechik, 2015].

However, these line of constructions did not require that the distance oracle is capable of printing an actual path besides an approximate distance estimate, and there has been a performance gap between path-reporting distance oracles and ones that are not path-reporting. It is known that the earliest construction by [Thorup and Zwick, 2001] is path-reporting, but the parameters are worse by a factor of k. In a later construction by [Wulff-Nilsen, 2013], the query time was improved from $O(k)$ to $O(\log k)$. Better trade-offs were discovered in [Elkin and Pettie, 2015] where the authors broke the $O(kn^{1+1/k})$ space barrier and achieved $O(n^{1+1/k} \log k)$ space with $O(\log k)$ query time, but their stretch was blown up to a polynomial $O(k^{\log_{4/3} 7})$; they also gave an alternative choice of $O(n^{1+1/k})$ space which is optimal, and $O(k)$-stretch which is also optimal up to a constant factor, but their query time rose exponentially to $O(n^{\epsilon})$. In a recent work [Elkin and Shabat, 2023], the authors obtained significant improvements of $O(n^{1+1/k} \log k)$ space, $O(k)$-stretch, and $O(\log \log k)$ query time, or $O(n^{1+1/k})$ space, $O(k \log k)$-stretch, and $O(\log \log k)$ query time.

All the above constructions of path-reporting distance oracles share a common barrier; that is, they could not achieve optimal space $O(n^{1+1/k})$ and stretch $O(k)$ simultaneously within logarithmic query time; for example, in the natural regime where $k = \lceil \log n \rceil$, previous distance oracles had to pay an extra factor of $\log \log n$ either in the space or stretch. As our result, we bypass this barrier by a new construction of path-reporting distance oracles with $O(n^{1+1/k})$ space and $O(k)$-stretch and $O(\log \log k)$ query time.

2012 ACM Subject Classification Theory of computation → Shortest paths

Keywords and phrases graph algorithms, shortest paths, distance oracles

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.42

Category Track A: Algorithms, Complexity and Games

Funding This publication is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803118 UncertainENV).

1 Introduction

Given an undirected graph $G = (V, E, \mathbf{w})$ on n vertices with positive edge weights , distance oracles are space-efficient data structures that process distance queries with approximate answers in fast runtime. More specifically, given any pair of vertices $s, t \in V$, the distance oracle needs to return a distance estimate $\mathbf{est}(s, t)$ in the range $[\mathbf{dist}_G(s, t), \beta \cdot \mathbf{dist}_G(s, t)]$,

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 42; pp. 42:1–42:18

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

where β is an upper bound on the stretch error. Intuitively, when the distance oracle has smaller size, the stretch error grows larger, and the general goal is to understand the optimal trade-off between space and stretch with a fast query efficiency.

In a pioneering work by Thorup and Zwick [10], they constructed distance oracles for any integer parameter $k \geq 1$ with $O(kn^{1+1/k})$ space, $(2k-1)$-stretch and $O(k)$ query time. This balance between size and stretch is optimal when $k \leq \frac{\log n}{\log \log n}$ because the famous Erdős girth conjecture implies an $\Omega(n^{1+1/k})$ space lower bound for any distance oracle with $(2k-1)$-stretch, and it was left open in [10] if anything better can be achieved for larger choices of parameter k. In a subsequent work by Mendel and Naor [9], using Ramsey partitions and tree embeddings, the authors diverged from the techniques of [10] and devised a new distance oracle with optimal space $O(n^{1+1/k})$ and $O(1)$ query time, but its stretch becomes $O(k)$ which is worse than the optimal $2k-1$ by a constant factor. Later on, still following the old construction of [10], Wulff-Nilsen [11] improved the query time from $O(k)$ to $O(\log k)$ while retaining $O(kn^{1+1/k})$ space and $(2k-1)$-stretch; in the same paper, he could also obtain constant query time $O(1/\epsilon)$ by allowing a slightly larger stretch of $(2+\epsilon)k$. This line of works culminated in an optimal construction of distance oracles that achieve $O(n^{1+1/k})$ space, $(2k-1)$-stretch and $O(1)$ query time simultaneously [5, 4].

However, these works mentioned above did not require that the distance oracles should be capable of printing an actual path between the queried vertex pair that realizes the approximate distance estimate; distance oracles with this extra function are called *path-reporting* distance oracles. More specifically, in this setting, the query efficiency consists of two parts: the query time which is the amount of time to compute a distance estimate $\mathbf{est}(s,t)$, and after that we need to report a path π between the vertex pair with total weight $\mathbf{w}(\pi) \leq \mathbf{est}(s,t)$ in time $O(|\pi|)$.

So far there has been a performance gap between path-reporting distance oracles and ones that do not support reporting paths. It was known that the earliest construction of Thorup and Zwick [10] is path-reporting, but some later improvements including [9, 11, 4, 5] are not, except one by [11] with $O(kn^{1+1/k})$ space, $(2k-1)$-stretch and $O(\log k)$ query time. In a subsequent work [7], the authors achieved optimal $O(n^{1+1/k})$ space by tolerating a constant blowup in the stretch $O\left(k \cdot (1/\epsilon)^{\log_{4/3} 7}\right)$ and a sublinear query time $O(n^{\epsilon})$, or alternatively, a sub-optimal space complexity of $O(n^{1+1/k} \log k)$ and faster query time $O(\log k)$, but a polynomial blowup in the stretch $O\left(k^{\log_{4/3} 7}\right)$.

In a very recent work [8], the authors devised new constructions of path-reporting distance oracles with multiple choices of trade-offs which are significant improvements over prior works: (1) with space $O\left(n^{1+1/k} \cdot \left\lceil \frac{k \log \log n \cdot \log^{(3)} n}{\log n} \right\rceil\right)$, they could achieve $(4+\epsilon)k$-stretch and $O(\log k)$ query time, and (2) with space $O\left(n^{1+1/k} \cdot \left\lceil \frac{k \log \log n}{\log n} \right\rceil\right)$, they could achieve $(24+\epsilon)k$-stretch and $O(\log \log k)$ time, and (3) with optimal space $O(n^{1+1/k})$ they could achieve $O(k \log k)$-stretch and $O(\log \log k)$ query time; here we have not listed all the trade-offs, but only the representative ones in [8].

One common quantitative barrier of all the above constructions of path-reporting distance oracles is that they could not achieve optimal space $O(n^{1+1/k})$ and linear stretch $O(k)$ simultaneously using a fast query time. For example, when k gets close to $\log n$, the trade-offs in [8] needs to pay $O(\log \log n)$ factor either in the space or in the stretch. As for the results from [7], although could achieve $O(n^{1+1/k})$ space and $O\left(k \cdot (1/\epsilon)^{\log_{4/3} 7}\right)$-stretch at the same time, their query time blows up exponentially to $O(n^{\epsilon})$; besides, their stretch upper bound has a large constant coefficient of $O\left((1/\epsilon)^{\log_{4/3} 7}\right)$ in front of k. This leads to the following natural question even for $k = \lceil \log n \rceil$.

▶ **Question 1.** *How to design a path-reporting distance with $O(n)$ space and $O(\log n)$-stretch that supports $O(\log n)$ query time?*

As our result, we give a positive answer to the above question, which is formalized in the statement below.

▶ **Theorem 2.** *Given an undirected graph $G = (V, E, \mathbf{w})$ with positive edge weights on n vertices, for any integer $k \geq 1$, there is a path-reporting distance oracle with $12k$-stretch, $O(n^{1+1/k})$ space, and $O(\log \log k)$ query time; furthermore, an approximate shortest path π can be retrieved in time $O(\log \log k + |\pi|)$ time.*

1.1 Technical overview

To support the function of path-reporting, prior works adopted different strategies: in [7] the authors were heavily exploiting the power of distance preservers [6], and in [8] the authors were utilizing the connection to hopsets. In our construction, we employed another different strategy of using *tree covers* which is adapted from the notion of cluster covers in the design of the optimal but non-path-reporting distance oracles [5].

The approach of [5]

Similar to [10], let us take a hierarchy of random sets $V = A_0 \supseteq A_1 \supseteq A_2 \supseteq \cdots \supseteq A_{k-1} \supseteq A_k$, where A_{i+1} includes each vertex in A_i with probability $n^{-1/k}$ independently. For any A_i, a *cluster cover* is a collection of (not necessarily disjoint) subsets $\mathcal{C}^i$ with the following properties.

- **Weak radius.** Each vertex $v \in V$ is assigned to a unique cluster $\mathcal{C}^i_{\mathsf{home}}(v) \in \mathcal{C}^i$ which contains v. Plus, $\mathcal{C}^i_{\mathsf{home}}(v)$ has weak radius of roughly $\mathbf{dist}_G(v, A_i)$.
- **Packing.** The total size of clusters in $\mathcal{C}$ is bounded by $O(n^{1+1/k})$.
- **Covering.** For any pair of vertices s, t, if $\mathbf{dist}_G(s,t) \leq \frac{1}{i} \cdot \mathbf{dist}_G(s, A_i)$, then $\mathcal{C}^i_{\mathsf{home}}(s)$ contains both s, t.

The idea of using cluster covers to compute approximate distances is to build cluster covers $\mathcal{C}^{(i)}$ for a constant number of indices $i \in \{k/2, k/4, k/8, \ldots, k/128\}$ (assuming k is an integer multiple of 128), and check if $t \in \mathcal{C}^i_{\mathsf{home}}(s)$ for each such i. If we can locate two consecutive indices $i, i/2$ such that $t \in \mathcal{C}^i_{\mathsf{home}}(s)$ and $t \notin \mathcal{C}^{i/2}_{\mathsf{home}}(s)$, then we will estimate $\mathbf{dist}_G(s,t)$ through the weak radius of $\mathcal{C}^i_{\mathsf{home}}(s)$. The reason why this would be a good estimate is because $t \notin \mathcal{C}^{i/2}_{\mathsf{home}}(s)$ and so $\mathbf{dist}_G(s,t) > \frac{2}{i} \cdot \mathbf{dist}_G(s, A_{i/2})$, and therefore $\mathbf{dist}_G(s,t)$ cannot be too much smaller than the weak radius of $\mathcal{C}^i_{\mathsf{home}}(s)$ when $\mathbf{dist}_G(s, A_{i/2})$ and $\mathbf{dist}_G(s, A_i)$ are not too different; if $\mathbf{dist}_G(s, A_{i/2})$ is also way smaller than $\mathbf{dist}_G(s, A_i)$, we will use some other shortcuts between pivots in $A_{i/2}$ to find a better distance estimate.

If we cannot locate any pair of consecutive indices $i, i/2$ such that $t \in \mathcal{C}^i_{\mathsf{home}}(s)$ and $t \notin \mathcal{C}^{i/2}_{\mathsf{home}}(s)$, then this means that both s, t are actually a small cluster $\mathcal{C}^{k/128}_{\mathsf{home}}(s)$. In this case, we should expect that the same distance oracle construction by Thorup and Zwick [10] to give a much better distance estimate with roughly $k/64$-stretch. To take advantage of this slack, we can apply the construction by Mendel and Naor [9] to quickly calculate an estimation.

Tree covers

One of the reasons why the distance oracle of [5] is not path-reporting is that cluster covers only have an upper bound on weak radius rather than strong radius. Therefore, for the cause of reporting paths, we need to ensure that each cluster has a spanning tree whose radius is small; in other words, we are looking for a tree cover $\mathcal{C}^i$ with the following revised properties.

- **Strong radius.** Each vertex $v \in V$ is assigned to a unique cluster $\mathcal{C}^i_{\mathsf{home}}(v) \in \mathcal{C}^i$ which contains v. Plus, $\mathcal{C}^i_{\mathsf{home}}(v)$ has a spanning tree of radius at most $\mathbf{dist}_G(v, A_i)$.

To find such a tree cover, let us start with the basic approach of ball-carving (which is similar to the approach for cluster covers [5]): starting with $W \leftarrow V$, while W is not empty, pick an arbitrary vertex $v \in W$ and grow a tree T in graph $G[W]$ to a radius $r \cdot \rho = \frac{r}{i} \cdot \mathbf{dist}_G(v, A_i)$ such that:

$$\left|\mathbf{Ball}_{G[W]}(v, (r-1)\rho)\right| \geq n^{-1/k} \left|\mathbf{Ball}_{G[W]}(v, r\rho)\right|$$

Then, we add this tree T to $\mathcal{C}^i$ and remove all vertices in $\mathbf{Ball}_{G[W]}(v, (r-1)\rho)$ from W; for those vertices we have just removed, assign their home trees to be T. By doing this we can guarantee the strong radius and the packing property. However, this could damage the covering property. Imagine that for some pair of vertices s, t which are close, some vertices on the shortest path between s, t were removed after adding a spanning tree T to $\mathcal{C}^i$, then the home tree of s might not be able to reach t. To fix this issue, the idea is to assign s to this earlier tree T which might not have removed s at the time it was created.

Graph contraction

Another obstacle of path-reporting is that the construction of Chechik [5] incorporates the construction of Mendel and Naor [9]. To avoid using the distance oracle by Mendel and Naor, we could continue to apply tree covers for smaller indices $i < k/128$, but this would raise the space complexity from $O(n^{1+1/k})$ to $O(n^{1+1/k} \log k)$. Notice that on lower levels, we are obtaining much better stretch than $O(k)$, which gives us some slack when designing our data structures. To take advantage of such slack, the idea is to apply the stretch-friendly partition of a recent work [3] which roughly contracts the graph G into n/τ clusters while blowup the stretch by $O(\tau)$, and then build our tree cover structures on the contracted graph which takes only $O(n^{1+1/k}/\tau)$ space.

2 Preliminaries

Let $\epsilon = 0.01$ be a fixed constant (independent of n, k), and assume $k > 100$. Logarithms are of base 2. For any graph $G = (V, E, \mathbf{w})$ and any vertex subset $S \subseteq V$, let $G[S]$ be the induced subgraph of G on S.

▶ **Definition 3** (bunches and balls). *Let $G = (V, E, \mathbf{w})$ be an undirected weighted graph. For each vertex $v \in V$ and vertex subset $S \subseteq V$, let $\mathbf{piv}_G(v, S) \in S$ be the closest vertex to v, and define the bunch around v with respect to S as:*

$$\mathbf{Bun}_G(v, S) \stackrel{\text{def}}{=} \{u \mid \mathbf{dist}_G(u, v) < \mathbf{dist}_G(u, S)\}$$

Furthermore, for any $\delta > 0$, define

$$\mathbf{Bun}^{\delta}_G(v, S) \stackrel{\text{def}}{=} \{u \mid \mathbf{dist}_G(u, v) < \delta \cdot \mathbf{dist}_G(u, S)\}$$

Balls are similar to bunches, except that the radius is specified by a value, rather than a vertex set. More specifically, for any value $\rho \geq 0$, define the ball around v to be:

$$\mathbf{Ball}_G(v, \rho) = \{u \mid \mathbf{dist}_G(u, v) \leq \rho\}$$

2.1 Stretch-friendly partitions

We need the notion of *stretch-friendly partition* from [3].

▶ **Definition 4** ([3]). *Let $G = (V, E, \mathbf{w})$ be an undirected weighted graph, and fix an integer $\tau \geq 1$. A* stretch-friendly τ-partition *of G is a partition $\mathcal{C} \subset 2^V$ of V with the following guarantees for every $U \in \mathcal{C}$.*

(1) *There is a spanning tree $T[U]$ of $G[U]$ rooted at the cluster center vertex $\mathbf{cntr}[U] \in U$, such that for every $v \in U$, the unique tree path between $\mathbf{cntr}[U], v$ has at most τ edges.*

(2) *If $(x, y) \in E$, $x \in U, y \notin U$, then the weight of every edge on the tree path in $T[U]$ between $x, \mathbf{cntr}[U]$ is at most $\mathbf{w}(x, y)$.*

(3) *If $(x, y) \in E$, $x, y \in U$, then the weight of every edge on the tree path in $T[U]$ between x, y is at most $\mathbf{w}(x, y)$.*

For any stretch-friendly partition $\mathcal{C}$, define $G/\mathcal{C}$ to be the *quotient graph* where each cluster in $\mathcal{C}$ is contracted to a single node. Here are some basic properties of stretch-friendly partitions.

▶ **Lemma 5.** *Let $G = (V, E, \mathbf{w})$ be an undirected weighted graph, and let $\mathcal{C}$ be a stretch friendly τ-partition of G. Consider any pair of vertices $s, t \in V$, then we can find a path π between s, t using edges of $G/\mathcal{C}$ and tree edges of $T[C], C \in \mathcal{C}$, such that:*

$$\mathbf{w}(\pi) \leq 4\tau \cdot \mathbf{dist}_G(s, t)$$

Furthermore, given the shortest path π' between $\mathcal{C}(s), \mathcal{C}(t)$ in $G/\mathcal{C}$, the above path π can be computed in time $O(|\pi|)$.

Proof. If $\mathcal{C}(s) = \mathcal{C}(t)$, then we can take the tree path between s, t in $T[\mathcal{C}(s)]$ which has length at most $2\tau \cdot \mathbf{dist}_G(s, t)$. Otherwise, take the shortest path π_0 between $\mathcal{C}(s), \mathcal{C}(t)$ and suppose it visits contracted nodes $\mathcal{C}(s) = U_0, U_1, \ldots, U_l = \mathcal{C}(t)$. Let $(u_i, v_i) \in E$ be the edge connecting $U_i, U_{i+1}, \forall 0 \leq i < l$. By definition of stretch-friendly τ-partitions, we have:

$$\mathbf{dist}_G(s, t) \geq \frac{1}{\tau} \cdot \min\left\{\mathbf{dist}_{T[U_0]}(s, \mathbf{cntr}[U_0]), \mathbf{dist}_{T[U_l]}(t, \mathbf{cntr}[U_l])\right\}$$

$$\mathbf{w}(u_i, v_i) \geq \frac{1}{2\tau} \cdot \left(\mathbf{dist}_{T[U_i]}(u_i, \mathbf{cntr}[U_i]) + \mathbf{dist}_{T[U_{i+1}]}(v_i, \mathbf{cntr}[U_{i+1}])\right)$$

To find a path between s, t, let π be a concatenation of these paths: tree path between s, u_0 in $T[U_0]$, edge (u_0, v_0), tree path between v_0, u_1 in $T[U_1]$, ... , tree path between v_{l-1}, t in $T[U_l]$. Taking a summation of the above inequalities, we have:

$$\mathbf{w}(\pi) \leq 2\tau \cdot \mathbf{dist}_G(s, t) + 2\tau \cdot \mathbf{dist}_{G/\mathcal{C}}(\mathcal{C}(s), \mathcal{C}(t)) \leq 4\tau \cdot \mathbf{dist}_G(s, t)$$

As for the runtime to compute π, using the tree structure of $T[C]$, it is straightforward to compute π by its definition in time $O(|\pi|)$. ◀

The next statement shows how to construct a hierarchy of stretch-friendly clusters.

▶ **Lemma 6** ([3]). *For any $0 \leq i \leq \lceil \log n \rceil$, there exists a stretch-friendly $(3 \cdot 2^i - 1)$-partition $\mathcal{C}_i$ and each cluster in $\mathcal{C}_i$ has size at least 2^i. Moreover, $\mathcal{C}_i$ is a refinement of $\mathcal{C}_{i+1}$; that is, for each cluster $U \in \mathcal{C}_{i+1}$, U can be partitioned into sub-clusters $U = U_1 \cup U_2 \cup \cdots \cup U_k$, such that $U_j \in \mathcal{C}_i, \forall 1 \leq j \leq k$, and each tree $T[U_j]$ is a sub-tree of $T[U]$.*

2.2 Distance preservers

Throughout this paper, we assume shortest paths are unique by breaking ties using alphabetic ordering. We need a path-reporting distance preserver from [7].

▶ **Definition 7** (branching events, [7]). *For a collection of paths Π in an undirected graph G, a pair of paths together with common vertex (π_1, π_2, x) is called a* branching event*, if $x \in V(\pi_1) \cap V(\pi_2)$, and that the edges incident on x of π_1 and π_2 are different.*

▶ **Lemma 8** ([7]). *Given an undirected weighted graph $G = (V, E, \mathbf{w})$ and a collection of paths Π, there is a distance-preserving path-reporting data structure that reports any path $\pi \in \Pi$ in time $O(|\pi|)$. The data structure has size $O(n + |\Pi| + |\mathbf{br}(\Pi)|)$ where $\mathbf{br}(\Pi)$ is the set of branching events of Π.*

As a by-product, though it is not necessary for our main result, we slightly improve the above lemma as following, thus answering a small open question in [7]. This matches the sparsity bound of distance-preservers which do not require the function of path-reporting [6].

▶ **Lemma 9.** *Given an undirected weighted graph $G = (V, E, \mathbf{w})$ and a collection of shortest paths Π, there is a distance-preserving path-reporting data structure that reports any path $\pi \in \Pi$ in time $O(|\pi|)$. The data structure has size $O\left(n + |\Pi| + \sqrt{n|\mathbf{br}(\Pi)|}\right)$ where $\mathbf{br}(\Pi)$ is the set of branching events of Π.*

3 Tree covers

Before describing our path-reporting distance oracles, we will first need a building block which is adapted from [5]. Our stretch guarantee is worse than [5] roughly by a factor of 3/2 because our data structure is path-reporting, while the data structure from [5] could only report distance values, not any real paths in the graph.

▶ **Lemma 10.** *Let $G = (V, E, \mathbf{w})$ be an undirected weighted graph and let $S \subseteq V$ be a vertex subset. Suppose that there exists an integer parameter $r \geq \log^2 k$ such $|\mathbf{Bun}_G(v, S)| \leq O(n^{\frac{r}{k}} \log^2 n)$ for each $v \in V$. Then, there exists a collection of trees $\mathcal{T}$ of G with the following properties.*

- *The size of all trees in $\mathcal{T}$ is bounded by $O(n^{1+1/k})$.*
- *For any pair of vertices u, v, if $\mathbf{dist}_G(u, v) \leq \frac{2}{3(1+\epsilon)^2 r} \cdot \mathbf{dist}_G(u, S)$, then there exists a tree $T \in \mathcal{T}$ such that T contains both u, v, and more importantly:*

$$\mathbf{dist}_T(u, v) \leq \mathbf{dist}_T(u, \mathbf{rt}[T]) + \mathbf{dist}_T(v, \mathbf{rt}[T]) \leq 2(1+\epsilon) \cdot \mathbf{dist}_G(u, S)$$

where $\mathbf{rt}[T]$ is a fixed root of T; plus, such a tree T can be found in constant time.

Proof. For any integer $0 \leq b \leq \lceil \log_{1+\epsilon} 3 \rceil$, consider the sequence of values $(1+\epsilon)^b, 3(1+\epsilon)^b, 9(1+\epsilon)^b, \cdots, 3^i(1+\epsilon)^b, \cdots$. For any $i \geq 0$, let $V_{i,b} \subseteq V$ be the set of vertices v such that $\mathbf{dist}_G(v, S) \in \left[3^i(1+\epsilon)^b, 3^i(1+\epsilon)^{b+1}\right]$; by definition, we have $V = \bigcup_{i,b} V_{i,b}$.

Constructing a tree cover. Fix any offset $0 \leq b \leq \lceil \log_{1+\epsilon} 3 \rceil$, we will build a collection of trees $\mathcal{T}_b$ by a greedy ball-carving scheme on G; in the end, we will take the union of all trees $\mathcal{T} = \bigcup_b \mathcal{T}_b$. Initially, set $W \leftarrow V$, and go over all integers $i = 0, 1, 2, \cdots$. As long as$V_{i,b} \setminus W \neq \emptyset$, enumerate all vertices $v \in V_{i,b}$. Define a distance value $\rho_v = \frac{1}{(1+\epsilon)r} \cdot \mathbf{dist}_G(v, S)$. Therefore, as $r > \log^2 k$, we have

$$\left|\mathbf{Ball}_{G[W]}(v, (1+\epsilon) r \rho_v)\right| \leq |\mathbf{Bun}_G(v, S)| < O(n^{r/k} \log^2 n) < n^{\left\lfloor \frac{r}{(1-\epsilon)} \right\rfloor \cdot \frac{1}{k}}$$

▷ Claim 11. There must exist an integer $0 \leq r_v < \lfloor (1+\epsilon)r \rfloor$ such that:

$$\left|\mathbf{Ball}_{G[W]}\left(v, (r_v+1)\cdot\rho_v\right)\right| \leq n^{1/k}\left|\mathbf{Ball}_{G[W]}\left(v, r_v\cdot\rho_v\right)\right|$$

Proof of claim. Suppose otherwise, then for any integer $0 \leq l < \lfloor (1+\epsilon)r \rfloor$, we have:

$$\left|\mathbf{Ball}_{G[W]}\left(v, (l+1)\cdot\rho_v\right)\right| > n^{1/k}\left|\mathbf{Ball}_{G[W]}\left(v, l\cdot\rho_v\right)\right|$$

Taking a product of these inequalities over all integers $0 \leq l < \lfloor (1+\epsilon)r \rfloor$, we have:

$$\left|\mathbf{Ball}_{G[W]}\left(v, \lfloor (1+\epsilon)r \rfloor\cdot\rho_v\right)\right| > n^{\left\lfloor \frac{r}{1-\epsilon}\right\rfloor \cdot \frac{1}{k}}$$

Which is a contradiction. ◁

Then, let T be the single-source shortest paths tree rooted at $\mathbf{rt}[T] \leftarrow v$ that spans the vertex set $\mathbf{Ball}_{G[W]}\left(v, (r_v+1)\cdot\rho_v\right)$. All vertices in $\mathbf{Ball}_{G[W]}\left(v, r_v\cdot\rho_v\right) \cap V(T)$ will be called core vertices of T denoted as $\mathbf{core}(T)$, and vertices in $\mathbf{Ball}_{G[W]}\left(v, (r_v+1)\cdot\rho_v\right)$ will be called peripheral vertices of T. After that, add T to $\mathcal{T}_b$, and then remove all the core vertices $\mathbf{Ball}_{G[W]}\left(v, r_v\cdot\rho_v\right)$ from W by updating the vertex set:

$$W \leftarrow W \setminus \mathbf{Ball}_{G[W]}\left(v, r_v\cdot\rho_v\right)$$

After that, move on to the next vertex in $V_{i,b} \setminus W$.

Assigning home trees. To find a tree that meets the requirement in the lemma statement in constant time, we need each vertex to remember a constant number of trees that contain itself, which are called *home trees*. Specifically, for each vertex $v \in V$, we will associate a tree $\mathcal{T}^b_{\mathsf{home}}(v) \in \mathcal{T}_b$ with v which is defined to be the **first tree** T created in $\mathcal{T}_b$ containing v such that:

$$\mathbf{dist}_T(v, \mathbf{core}(T)) \leq \frac{1}{3(1+\epsilon)^2}\rho_v$$

Size of the tree cover. By the algorithm, each time we add a new tree T to $\mathcal{T}_b$, $|T|$ is at most $n^{1/k}|\mathbf{core}(T)|$. As core vertices are removed from W right afterwards, the total size of $\mathcal{T}_b$ is bounded by $O\left(n^{1+1/k}\right)$.

The covering property. Consider any pair of vertices u, v such that $\mathbf{dist}_G(u,v) \leq \frac{2}{3(1+\epsilon)^2 r} \cdot \mathbf{dist}_G(u, S)$. Let us first state an elementary inequality.

▷ Claim 12. $\rho_u \leq (1+\epsilon)\cdot\rho_v$.

Proof of claim. By triangle inequality, we have:

$$\begin{aligned}\mathbf{dist}_G(u,S) &\leq \mathbf{dist}_G(v,S) + \mathbf{dist}_G(u,v)\\ &\leq \mathbf{dist}_G(v,S) + \frac{2}{3(1+\epsilon)^2 r}\cdot\mathbf{dist}_G(u,S)\\ &< \mathbf{dist}_G(v,S) + \frac{\epsilon}{1+\epsilon}\cdot\mathbf{dist}_G(u,S)\end{aligned}$$

The last inequality holds as $r \geq \log^2 k$ and ϵ is a constant. Therefore, by definition of ρ_u, ρ_v, we can conclude the proof. ◁

Let π be the shortest path between u, v in G. Assume $u \in V_{i,b}$ for some $i, b \geq 0$.

▷ Claim 13. Vertices on π cannot be core vertices of trees rooted at any vertices from $V_{j,b}$ for some $j < i$.

Proof of claim. Suppose otherwise that there was a vertex $w \in V_{j,b}$ which grew a tree $T \in \mathcal{T}_b$ whose core includes a vertex $z \in V(\pi)$. Then, by triangle inequality, we would have:

$$\begin{aligned}\mathbf{dist}_G(u,S) &\leq \mathbf{dist}_G(u,z) + \mathbf{dist}_G(z,w) + \mathbf{dist}_G(w,S)\\ &\leq \frac{2}{3(1+\epsilon)r} \cdot 3^i \cdot (1+\epsilon)^{b+1} + 2 \cdot 3^{i-1} \cdot (1+\epsilon)^{b+1}\\ &< \left(\frac{1}{32} + 2\right) \cdot 3^{i-1} \cdot (1+\epsilon)^{b+1}\\ &< 3^i \cdot (1+\epsilon)^b\end{aligned}$$

This contradicts the definition that $u \in V_{i,b}$. ◁

As for the tree covering property, consider the first tree T created whose core intersected with $V(\pi)$, and consider the moment right before T was created. Since $z \in V(\pi)$, it must be:

$$\min\{\mathbf{dist}_{G[W]}(z,u), \mathbf{dist}_{G[W]}(z,v)\} \leq 0.5 \cdot \mathbf{dist}_G(u,v) \leq \frac{\rho_u}{3(1+\epsilon)^2}$$

Consider two possibilities.

- $\mathbf{dist}_{G[W]}(z,u) \leq 0.5 \cdot \mathbf{dist}_G(u,v) \leq \frac{\rho_u}{3(1+\epsilon)^2}$.
  In this case, we have $\mathbf{dist}_T(u, \mathbf{core}(T)) \leq \frac{\rho_u}{3(1+\epsilon)^2}$. Hence, by definition of $\mathcal{T}^b_{\mathsf{home}}(u)$, $\mathcal{T}^b_{\mathsf{home}}(u)$ must have been created no later than T. Therefore, when $\mathcal{T}^b_{\mathsf{home}}(u)$ was being created with root $z \in V_{i,b}$ (Claim 13), all vertices on π was still present in W. Therefore, at the moment, we have:
  $$\begin{aligned}\mathbf{dist}_{G[W]}\left(v, \mathbf{core}\left(\mathcal{T}^b_{\mathsf{home}}(u)\right)\right) &\leq \mathbf{dist}_{G[W]}(u,v) + \mathbf{dist}_{\mathcal{T}^b_{\mathsf{home}}(u)}\left(u, \mathbf{core}\left(\mathcal{T}^b_{\mathsf{home}}(u)\right)\right)\\ &\leq \frac{\rho_u}{(1+\epsilon)^2} < \rho_z\end{aligned}$$
  Hence, v was included in $\mathcal{T}^b_{\mathsf{home}}(u)$. As for their distance in the tree, by our algorithm, the radius of $\mathcal{T}^b_{\mathsf{home}}(u)$ is at most $3^i \cdot (1+\epsilon)^{b+1} \leq (1+\epsilon)\mathbf{dist}_G(u,S)$. Hence, we have:
  $$\mathbf{dist}_{\mathcal{T}^b_{\mathsf{home}}(u)}(u,v) \leq 2(1+\epsilon) \cdot \mathbf{dist}_G(u,S)$$
- $\mathbf{dist}_{G[W]}(z,v) \leq 0.5 \cdot \mathbf{dist}_G(u,v) \leq \frac{1}{3(1+\epsilon)^2}\rho_u$.
  In this case, we have $\mathbf{dist}_T(v, \mathbf{core}(T)) \leq \frac{1}{3(1+\epsilon)^2}\rho_u \leq \frac{1}{3(1+\epsilon)}\rho_v$. Hence, by definition of $\mathcal{T}^b_{\mathsf{home}}(v)$, $\mathcal{T}^b_{\mathsf{home}}(v)$ must have been created no later than T. Therefore, when $\mathcal{T}^b_{\mathsf{home}}(v)$ was being created with root $z \in V_{i,b}$ (Claim 13), all vertices on π was still present in W. Therefore, at the moment, we have:
  $$\begin{aligned}\mathbf{dist}_{G[W]}\left(u, \mathbf{core}\left(\mathcal{T}^b_{\mathsf{home}}(v)\right)\right) &\leq \mathbf{dist}_{G[W]}(u,v) + \mathbf{dist}_{\mathcal{T}^b_{\mathsf{home}}(v)}\left(v, \mathbf{core}\left(\mathcal{T}^b_{\mathsf{home}}(v)\right)\right)\\ &\leq \frac{1}{1+\epsilon}\rho_v \leq \rho_z\end{aligned}$$
  Hence, u was included in $\mathcal{T}^b_{\mathsf{home}}(v)$. As for their distance in the tree, by our algorithm, the radius of $\mathcal{T}^b_{\mathsf{home}}(v)$ is at most $3^i \cdot (1+\epsilon)^{b+1} \leq (1+\epsilon)\mathbf{dist}_G(u,S)$. Hence, we have:
  $$\mathbf{dist}_{\mathcal{T}^b_{\mathsf{home}}(u)}(u,v) \leq 2(1+\epsilon) \cdot \mathbf{dist}_G(u,S)$$ ◀

Given such a collection of trees $\mathcal{T}$ satisfying the conditions of Lemma 10, let us state a subroutine $\mathbf{TreeCover}(s,t,\mathcal{T})$ with constant runtime that checks if the tree cover data structure $\mathcal{T}$ can provide a distance estimation for $\mathbf{dist}_G(s,t)$ as long with the a path.

Algorithm 1 **TreeCover**$(s, t, \mathcal{T})$.

```
Assume s ∈ V_{i,a}, t ∈ V_{j,b};
for v ∈ {s,t}, c ∈ {a,b} do
    if T^c_home(v) contains both {s,t} then
        return dist_{T^c_home(v)}(s,t), along with the tree path in T^c_home(v) between s,t if
          required;
return ⊥;
```

4 Path-reporting distance oracles

4.1 Data structures

Define $\alpha = 3/4 + \epsilon$. Apply Lemma 6 on G to create a sequence of stretch-friendly partitions $\{\mathcal{C}_i\}_{0 \le i \le \lceil \log n \rceil}$. Build a level ancestor data structure with $O(n)$ space so that for any $u \in V$ and any index $0 \le i \le \lceil \log n \rceil$, we can access the cluster $\mathcal{C}_i(u)$ in constant time [2].

Take a hierarchy of vertex sets $V = A_0 \supseteq A_1 \supseteq A_2 \supseteq \cdots \supseteq A_k$, where A_{i+1} includes each vertex in A_i independently with probability $n^{-1/k}$, $\forall 0 \le i < k$. Therefore, with high probability, size of A_i is at most $O(n^{1-\frac{i}{k}} \log n)$.

Define a sequence of integers $h_0, h_1, h_2, \ldots, h_\iota$ satisfying $h_0 = k$, $h_1 = \lceil \alpha k \rceil$, and for $l \ge 1$ define $h_l = \max\left\{\lceil \alpha^l k \rceil, 2\lceil \log^2 k \rceil\right\}$, the sequence stops until it reaches $h_\iota = 2\lceil \log^2 k \rceil$.

Data structures for high levels

Let $\kappa = 50$ be an integer threshold. For every integer $0 \le l \le \iota$, we will build some data structures separately. When $l > \kappa$, let $\mathcal{C}^{(l)} \in \{\mathcal{C}_i\}_{0 \le i \le \lceil \log n \rceil}$ which is a stretch-friendly τ_l-partition where $\tau_l \in [\alpha^{-l/5}, 2\alpha^{-l/5}]$; we can make sure that the sequence $\{\tau_l\}_{\kappa \le l \le \iota}$ is non-decreasing. Plus, if $0 \le l \le \kappa$, then set $\mathcal{C}^{(l)}$ to be singletons (that is, $\tau_l = 1$). For each vertex $v \in V$, let $\mathcal{C}^{(l)}(v) \in \mathcal{C}^{(l)}$ be the unique cluster that contains v. Let $G/\mathcal{C}^{(l)}$ be the quotient graph of G where each cluster in $\mathcal{C}^{(l)}$ is contracted to a single node, and the edges of $G/\mathcal{C}^{(l)}$ are edges in G between different clusters in $\mathcal{C}^{(l)}$.

For the rest, we will build some graph data structures. For each integer $1 \le l \le \iota$, let B_{h_l} be the set of contracted nodes in $G/\mathcal{C}^{(l)}$ containing at least one vertex from the random set A_{h_l}, and let $C_{h_{l-1}}$ be the set of contracted nodes in $G/\mathcal{C}^{(l)}$ containing at least one vertex from the random set $A_{h_{l-1}}$. Build the following data structures.

(i) For each integer $1 \le l \le \iota$, apply Lemma 10 on the quotient graph $G/\mathcal{C}^{(l)}$ to build a tree cover $\mathcal{T}_l$ with respect to the terminal set B_{h_l} by setting the parameter $r = h_l$; we will prove that this parameter r satisfies the requirement of Lemma 10 with high probability over the random choices of $A_1, \ldots, A_k$.

For the special case when $l = 0$, simply set $\mathcal{T}_0$ to be single-source shortest paths trees rooted at each vertex in A_k in G, and each vertex in $u \in V$ has a pointer to the vertex in A_k that is the closest one to u. It is straightforward to see that $\mathcal{T}_0$ is also a valid tree cover for A_k using the definition of Lemma 10 (by setting $r \leftarrow k, S \leftarrow A_k$), and **TreeCover**$(s, t, \mathcal{T}_0)$ can always return a nonempty value whether or not the inequality $\mathbf{dist}_G(u, v) \le \frac{2(1-\epsilon)^2}{3k} \cdot \mathbf{dist}_G(u, A_k)$ is satisfied.

(ii) For each integer $0 \le l \le \iota$ and for each contracted node x in $G/\mathcal{C}^{(l)}$, store a shortest path from x to its nearest node in B_{h_l} denoted as $\mathbf{piv}_{G/\mathcal{C}^{(l)}}(x, B_{h_l})$.

(iii) For each integer $1 \leq l \leq \iota$, we will build a set of paths Π_l in $G/\mathcal{C}^{(l)}$ in the following manner.
For each pair of nodes $x \in B_{h_l}$ and $y \in B_{h_l} \cap \mathbf{Bun}^{1/2}_{G/\mathcal{C}^{(l)}}\left(x, C_{h_{l-1}}\right)$, let $\pi_{x,y}$ the shortest path between x, y in graph $G/\mathcal{C}^{(l)}$, and add $\pi_{x,y}$ to Π_l. After building path set Π_l, apply Lemma 8 to build a path-reporting distance preserver data structure in the quotient graph $G/\mathcal{C}^{(l)}$ with respect to Π_l.

Data structures for low levels

Let $\mathcal{C}^{(\iota)}$ be a stretch-friendly $\Theta(\log^2 k)$-partition of G, and let $G/\mathcal{C}^{(\iota)}$ be the quotient graph. For each $0 \leq i \leq 2\left\lceil \log^2 k \right\rceil$, let B_i be the set of nodes in $G/\mathcal{C}^{(\iota)}$ containing at least one vertex in A_i. Store the following data structures for each $0 \leq i \leq 2\left\lceil \log^2 k \right\rceil$.

(i) For each node x in graph $G/\mathcal{C}^{(\iota)}$, store the shortest paths to all nodes $y \in B_i$ in $G/\mathcal{C}^{(\iota)}$ such that $\mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(x, y) < \mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(x, B_{i+1})$, as well as the shortest path from x to $\mathbf{piv}_{G/\mathcal{C}^{(\iota)}}(x, B_i)$. We assume shortest paths are unique by breaking ties alphabetically.

(ii) For each node x in graph $G/\mathcal{C}^{(\iota)}$ and each even index $0 \leq i < 2\left\lceil \log^2 k \right\rceil$, store the difference:

$$\Delta(x, i) \stackrel{\text{def}}{=} \mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(x, B_{i+2}) - \mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(x, B_i)$$

Then, for each such x, store a range minimum query data structure for all entries $\Delta(x, i), i = 0, 2, \ldots, 2\left\lceil \log^2 k \right\rceil - 2$ with linear space and constant query time [1].

4.2 Query algorithm

Given any query $(s, t) \in V^2$, the query algorithm consists of two phases, one for high levels (in range $\left[2\left\lceil \log^2 k \right\rceil, k\right]$), and one for low levels (in range $\left[0, 2\left\lceil \log^2 k \right\rceil\right]$).

High-level phase

By the design of the data structure, $\mathbf{TreeCover}(s, t, \mathcal{T}_0)$ always returns nonempty value because each tree in $\mathcal{T}_0$ is a spanning tree of G. For each $1 \leq l \leq \iota$, define $u_l = \mathcal{C}^{(l)}(s), v_l = \mathcal{C}^{(l)}(t)$. Next, run the subroutine $\mathbf{TreeCover}(u_\iota, v_\iota, \mathcal{T}_\iota)$; if it successfully returns a nonempty value, then move on to the next phase. Otherwise, the algorithm maintains two indices $l_1 \leftarrow 0, l_2 \leftarrow \iota$ and performs the following binary search procedure; the goal of the binary search procedure is to find a consecutive pair of indices $l-1, l$ such that $\mathbf{TreeCover}(u_l, u_l, \mathcal{T}_l)$ returns an empty value, while $\mathbf{TreeCover}(u_{l-1}, v_{l-1}, \mathcal{T}_{l-1})$ returns successfully a nonempty value.

(1) If $l_1 < l_2 - 1$, define $l_3 = \lfloor (l_1 + l_2)/2 \rfloor$, and run the subroutine $\mathbf{TreeCover}(u_{l_3}, v_{l_3}, \mathcal{T}_{l_3})$. If the subroutine returns an empty value, then assign $l_2 \leftarrow l_3$; otherwise, assign $l_1 \leftarrow l_3$. Then, continue with the new pair (l_1, l_2).

(2) Now suppose $l_1 + 1 = l_2 = l$ and $\mathbf{TreeCover}(u_{l-1}, v_{l-1}, \mathcal{T}_{l-1})$ returns a nonempty value, but the procedure $\mathbf{TreeCover}(u_l, v_l, \mathcal{T}_l)$ returns null. Then, query part (ii) of the data structure in the storage to find the vertices $x = \mathbf{piv}_{G/\mathcal{C}^{(l)}}(u_l, B_{h_l})$ and $y = \mathbf{piv}_{G/\mathcal{C}^{(l)}}(v_l, B_{h_l})$, and check two possibilities below.

(a) The shortest path between nodes x, y in $G/\mathcal{C}^{(l)}$ is preserved in the distance-preserver; that is, there is a path between x, y in Π_l
In this case, query the shortest path π'_1 from u_l to x using part (ii), and the shortest path π'_2 from x to y using the path-reporting distance preserver in part (iii), and the shortest path π'_3 from y to v_l using part (ii). Then, define the concatenation $\pi' = \pi'_1 \circ \pi'_2 \circ \pi'_3$ and then recover a path π between s, t in G by unpacking the contracted nodes in $\mathcal{C}^{(l)}$ using π' by Lemma 5 with runtime $O(|\pi|)$.

(b) The shortest path between nodes x, y in $G/\mathcal{C}^{(l)}$ is not preserved in the distance-preserver.
In this case, as **TreeCover** $(u_{l-1}, v_{l-1}, \mathcal{T}_{l-1})$ returned a nonempty value, we can use the tree in $\mathcal{T}_{l-1}$ that covers both u_{l-1}, v_{l-1} to get a path π' between them. Then, by Lemma 5 we can obtain a path between s, t in G which unpacks the contracted nodes in $\mathcal{C}^{(l-1)}$.

Low-level phase

This part is pretty much the same as the query algorithm from [11]. The difference is that we are making the queries in the contracted graph $G/\mathcal{C}^{(\iota)}$, and total number of levels is at most $2\lceil\log^2 k\rceil$ rather than k. For convenience, rename the variables by $u = \mathcal{C}^{(\iota)}(s)$ and $v = \mathcal{C}^{(\iota)}(t)$. We first check if $u = v$; if it is the case, we can directly retrieve a path using the spanning trees of the stretch-friendly τ_ι-partition $\mathcal{C}^{(\iota)}$ which has stretch error at most $2\tau_\iota < k$. For the rest, let us assume that $u \neq v$.

▶ **Definition 14** ([11]). *For a pair of contracted nodes u, v in $G/\mathcal{C}^{(\iota)}$, and even index $j \in [0, 2\log^2 k]$ is called* (u, v)-terminal *if (1) $j = 2\lceil\log^2 k\rceil$ or (2) $j < 2\lceil\log^2 k\rceil$ and one of the following conditions holds:*

$$\mathbf{piv}_{G/\mathcal{C}^{(\iota)}}(u, B_j) \in \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(v, B_{j+1})$$

$$\mathbf{piv}_{G/\mathcal{C}^{(\iota)}}(v, B_{j+1}) \in \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(u, B_{j+2})$$

Next, we will perform a binary search on the interval $[0, 2\lceil\log^2 k\rceil]$ as did in [11]. Initially, set $i_1 \leftarrow 0, i_2 \leftarrow 2\lceil\log^2 k\rceil$. In each iteration we perform the following steps while guaranteeing that i_2 is always (u, v)-terminal.

- If $i_1 = i_2$, since i_2 is (u, v)-terminal, we can consider three different possibilities.
  - $i_2 = 2\lceil\log^2 k\rceil$.
    Since **TreeCover**$(u, v, \mathcal{T}_\iota)$ returns successfully, we can use the tree in $\mathcal{T}_\iota$ that covers both u, v to get a path π' between them. Then, by Lemma 5 we can obtain a path between s, t in G which unpacks the contracted nodes in $\mathcal{C}^{(\iota)}$.
  - $\mathbf{piv}_{G/\mathcal{C}^{(\iota)}}(u, B_{i_2}) \in \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(v, B_{i_2+1})$.
    According to part (i) of our low-level data structures, we have stored the shortest path π_1 from v to $\mathbf{piv}_{G/\mathcal{C}^{(\iota)}}(u, B_{i_2})$ in $G/\mathcal{C}^{(\iota)}$, as well as the shortest path π_2 from u to $\mathbf{piv}_{G/\mathcal{C}^{(\iota)}}(u, B_{i_2})$. Then, we can retrieve a path between s, t by applying Lemma 5 on $\pi_1 \circ \pi_2$.
  - $\mathbf{piv}_{G/\mathcal{C}^{(\iota)}}(v, B_{i_2+1}) \in \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(u, B_{i_2+2})$.
    Similarly, according to part (i) of our low-level data structures, we have stored the shortest path π_1 from u to $\mathbf{piv}_{G/\mathcal{C}^{(\iota)}}(v, B_{i_2+1})$ in $G/\mathcal{C}^{(\iota)}$, as well as the shortest path π_2 from v to $\mathbf{piv}_{G/\mathcal{C}^{(\iota)}}(u, B_{i_2+2})$. Then, we can retrieve a path between s, t by applying Lemma 5 on $\pi_1 \circ \pi_2$.
- Otherwise, let j be the middle even index among $i_1, i_1 + 2, \cdots, i_2 - 2$, and let i_3 be the even index in $i_1, i_1 + 2, \cdots, j$ maximizing $\Delta(u, i_3)$ by querying part (ii) of our low-level data structure. If i_3 is not (u, v)-terminal, then recurse on the index pair $(j + 2, i_2)$; otherwise, recurse on (i_1, i_3).

4.3 Space analysis

Size of high-level data structures

Let us analyze the size of our data structures part by part for each index $0 \leq l < \iota$.

▷ Claim 15. Let $N = O(n \cdot \alpha^{l/5})$ be the number of contracted nodes in $G/\mathcal{C}^{(l)}$. With high probability over the randomness of A_{h_l}, for any contracted node $u \in G/\mathcal{C}^{(l)}$, we have the following bounds.

- $\left|\mathbf{Bun}_{G/\mathcal{C}^{(l)}}(u, B_{h_l})\right| \leq O\left(N^{\frac{h_l}{k}} \log^2 N\right)$.
- $\left|B_{h_l} \cap \mathbf{Bun}_{G/\mathcal{C}^{(l)}}\left(u, C_{h_{l-1}}\right)\right| \leq O\left(n^{\frac{h_{l-1}-h_l}{k}} \log n\right)$.

Proof. Let S be the set of nodes in $G/\mathcal{C}^{(l)}$ which are the nearest $10n^{\frac{h_l}{k}} \log n$ ones to u. Then, since A_{h_l} samples each vertices in V independently with probability $n^{-\frac{h_l}{k}}$, with high probability, A_{h_l} contains at least one vertex contracted in some nodes in S. Therefore, in this case we have $\left|\mathbf{Bun}_{G/\mathcal{C}^{(l)}}(u, B_{h_l})\right| \leq O\left(n^{\frac{h_l}{k}} \log n\right) \leq O\left(N^{\frac{h_l}{k}} \log^2 N\right)$.

As for the second inequality bound, let $u_1, u_2, \ldots, u_j \in B_{h_l}$ be the nearest nodes to u in $G/\mathcal{C}^{(l)}$, for some $j = O\left(n^{\frac{h_{l-1}-h_l}{k}} \log n\right)$. Then, since each u_i belongs to $C_{h_{l-1}}$ with probability at least $n^{-\frac{h_{l-1}-h_l}{k}}$, at least one node $u_i, 1 \leq i \leq j$ should belong to $C_{h_{l-1}}$ with high probability. Therefore, the size of $\left|B_{h_l} \cap \mathbf{Bun}_{G/\mathcal{C}^{(l)}}\left(u, C_{h_{l-1}}\right)\right|$ should be $\leq O\left(n^{\frac{h_{l-1}-h_l}{k}} \log n\right)$. ◁

Using Lemma 10 and Claim 15, we know that the total size of the tree cover data structure for the quotient graph $G/\mathcal{C}^{(l)}$ is bounded by $O\left(n^{1+\frac{1}{k}} \cdot \alpha^{l/5}\right)$. Then, taking a summation over all indices $0 \leq l < \iota$, the size bound becomes $O\left(n^{1+\frac{1}{k}}\right)$. Similarly, we can also bound the total size of shortest paths from all the nodes x to $\mathbf{piv}_{G/\mathcal{C}^{(l)}}(x, B_{h_l})$.

To bound the total size of our path-reporting distance preservers, we need to bound the total number of branching events of Π_l.

▷ Claim 16. For each $0 \leq l \leq \iota$, the expectation of $|\mathbf{br}(\Pi_l)| + |\Pi_l|$ over the randomness of $A_1, \ldots, A_k$ is bounded by $O(n/\log n)$.

Proof. Let $(\pi_1, \pi_2, *) \in \mathbf{br}(\Pi_l)$ be any branching event, and assume π_1, π_2 are shortest paths in $G/\mathcal{C}^{(l)}$ between nodes u_1, v_1 and u_2, v_2 such that $v_b \in \mathbf{Bun}^{1/2}_{G/\mathcal{C}^{(l)}}\left(u_b, C_{h_{l-1}}\right), \forall b \in \{1, 2\}$. Without loss of generality, assume that $\mathbf{w}(\pi_1) \geq \mathbf{w}(\pi_2)$. Then, we have:

$$\mathbf{dist}_{G/\mathcal{C}^{(l)}}(u_1, x) \leq \mathbf{w}(\pi_1) + \mathbf{w}(\pi_2) < \mathbf{dist}_{G/\mathcal{C}^{(l)}}\left(u_1, C_{h_{l-1}}\right), \forall x \in \{v_1, u_2, v_2\}$$

Therefore, $v_1, u_2, v_2 \in B_{h_l} \cap \mathbf{Bun}_{G/\mathcal{C}^{(l)}}\left(u_1, C_{h_{l-1}}\right)$. Therefore, using Claim 15, with high probability, $|\mathbf{br}(\Pi_l)|$ is bounded by the following up to a constant factor (recall that $\alpha = 3/4 + \epsilon$):

$$\begin{aligned}
\sum_{u \in B_{h_l}} \left|B_{h_l} \cap \mathbf{Bun}_{G/\mathcal{C}^{(l)}}\left(u, C_{h_{l-1}}\right)\right|^3 &\leq |B_{h_l}| \cdot O\left(n^{\frac{3h_{l-1}-3h_l}{k}} \log^3 n\right) \\
&\leq |A_{h_l}| \cdot O\left(n^{\frac{3h_{l-1}-3h_l}{k}} \log^3 n\right) \\
&\leq O\left(n^{1-\frac{h_l}{k}+\frac{3h_{l-1}-3h_l}{k}} \log^4 n\right) \\
&< O\left(n^{1-\frac{4\left\lceil \alpha^l k\right\rceil - 3\left\lceil \alpha^{l-1} k\right\rceil}{k}} \log^4 n\right) \\
&\leq O\left(n^{1-\frac{3\epsilon \cdot \alpha^{l-1} k - 3}{k}} \log^4 n\right) \\
&< O\left(n^{1-\epsilon \cdot \log^2 k} \log^4 n\right) \\
&< n/\log n
\end{aligned}$$

As for the size of Π_l, by definition, it is bounded by $\sum_{u \in B_{h_l}} \left| B_{h_l} \cap \mathbf{Bun}^{1/2}_{G/\mathcal{C}^{(l)}}(u, C_{h_{l-1}}) \right|$ which is also at most $O(n/\log n)$ according to the above calculation. ◁

Applying Definition 7 and Claim 16, we can bound the expected size of part-(iii) of our data structures by $O(n)$.

Size of low-level data structures

It is clear that part (ii) only takes space $O(n \log^2 k/\tau_\iota) \leq O(n)$. As for part (i), for each node x, we have stored the shortest path from x to every node $y \in B_i \cap \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(x, B_{i+1})$ and $\mathbf{piv}_{G/\mathcal{C}^{(\iota)}}(x, B_i)$. It is clear that the total size of the shortest paths to pivots is at most $O(n/\tau_\iota)$, so it is at most $O(n)$ by taking a summation over all indices $0 \leq i \leq 2\lceil \log^2 k \rceil$. Next, let us focus on shortest paths from x to nodes in $B_i \cap \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(x, B_{i+1})$.

▷ Claim 17. For each node x in $G/\mathcal{C}^{(\iota)}$, the expected size of $B_i \cap \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(x, B_{i+1})$ is at most $n^{1/k}$ over the randomness of $A_1, A_2, \ldots, A_k$.

Proof. Fix any node x, order all nodes in $G/\mathcal{C}^{(\iota)}$ in an increasing order of distances as $y_0, y_1, \ldots, y_l, \ldots$. For each y_j, let $|y_j|$ be the number of vertices in V contracted within. Conditioning on $\mathbf{piv}_{G/\mathcal{C}^{(\iota)}}(x, B_{i+1})$ being y_l, the expected total number vertices contracted in nodes from $B_i \cap \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(x, B_{i+1})$ is equal to $n^{-i/k} \sum_{j=0}^{l-1} |y_j|$. Therefore, the overall expected total number vertices contracted in nodes from $B_i \cap \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(x, B_{i+1})$ would be:

$$
\begin{aligned}
& n^{-i/k} \sum_{l \geq 1} \sum_{j=0}^{l-1} |y_j| \cdot \Pr\left[\mathbf{piv}_{G/\mathcal{C}^{(\iota)}}(x, B_{i+1}) = y_l\right] \\
&= n^{-i/k} \sum_{l \geq 1} \sum_{j=0}^{l-1} |y_j| \cdot \left(1 - n^{-(i+1)/k}\right)^{\sum_{j=0}^{l-1} |y_j|} \cdot \left(1 - \left(1 - n^{-(i+1)/k}\right)^{|y_l|}\right) \\
&\leq n^{-i/k} \cdot n^{(i+1)/k} = n^{1/k}
\end{aligned}
$$

The inequality holds as the summation is maximized when $|y_j| = 1, \forall j \geq 0$. Therefore, the expected size of $B_i \cap \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(x, B_{i+1})$ is also at most $n^{1/k}$. ◁

To bound the total size of shortest paths from x to $B_i \cap \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(x, B_{i+1})$, we need the one more statement below.

▷ Claim 18. Fix any node x, and for any $y \in B_i \cap \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(x, B_{i+1})$ and node z on the shortest path from x to y in graph $G/\mathcal{C}^{(\iota)}$, we have $y \in \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(z, B_{i+1})$.

Proof. By triangle inequality, we have:

$$
\begin{aligned}
\mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(z, y) &= \mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(x, y) - \mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(x, z) \\
&< \mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(x, B_{i+1}) - \mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(x, z) \\
&< \mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(z, B_{i+1})
\end{aligned}
$$

So by definition of bunches, we have $y \in \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(z, B_{i+1})$. ◁

By Claim 18, if the data structure needs to store the shortest path $\gamma_{x,y}$ from x to $y \in B_i \cap \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(x, B_{i+1})$, then it also needs to store each shortest path $\gamma_{z,y}$ for any $z \in \gamma_{x,y}$. Therefore, to store all the path $\{\gamma_{x,y}\}$ in a space-efficient manner, we only need to store the first edge of $\gamma_{x,y}$ (which is incident on x) for all x in $G/\mathcal{C}^{(\iota)}$ and $y \in B_i \cap \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(x, B_{i+1})$; this also allows us to retrieve the path $\gamma_{x,y}$ in $|\gamma_{x,y}|$ time. Then, by Claim 17, the total size of this shortest path data structure over all node x in $G/\mathcal{C}^{(\iota)}$ would be bounded by $O(n^{1+1/k}/\tau_\iota)$, which is $O(n^{1+1/k})$ summing over all $0 \leq i \leq 2\lceil \log^2 k \rceil$.

4.4 Stretch analysis

Let us analyze the stretch of the high-level phase and the low-level phase separately.

High-level phase

By the algorithm description, suppose that the subroutine $\mathbf{TreeCover}(u_\iota, v_\iota, \mathcal{T}_\iota)$ does not return any nonempty value. Then, by the binary search procedure, in the end we will find an index $1 \le l \le \iota$ such that $\mathbf{TreeCover}(u_l, v_l, \mathcal{T}_l)$ returns null and $\mathbf{TreeCover}(u_{l-1}, v_{l-1}, \mathcal{T}_{l-1})$ returns a nonempty value. Define $x = \mathbf{piv}_{G/\mathcal{C}^{(l)}}(u_l, B_{h_l})$ and $y = \mathbf{piv}_{G/\mathcal{C}^{(l)}}(v_l, B_{h_l})$.

▷ Claim 19. If $1 \le l \le \kappa$, then:

$$\mathbf{dist}_G(x,y) \le \left(3(1+\epsilon)^2 h_l + 1\right) \cdot \mathbf{dist}_G(s,t)$$

If $\kappa < l \le \iota$, then:

$$\mathbf{dist}_{G/\mathcal{C}^{(l)}}(x,y) \le \left(3(1+\epsilon)^2 h_l + 1\right) \cdot \mathbf{dist}_{G/\mathcal{C}^{(l)}}(u_l, v_l)$$

Proof. For the first inequality, since $\mathbf{TreeCover}(s,t,\mathcal{T}_l)$ does not return anything, by the guarantee of Lemma 10, we know that:

$$\mathbf{dist}_G(s,t) > \frac{2}{3(1+\epsilon)^2 h_l} \cdot \max\left\{\mathbf{dist}_G(s, A_{h_l}), \mathbf{dist}_G(t, A_{h_l})\right\}$$

Thus, by triangle inequality, we have:

$$\mathbf{dist}_G(x,y) \le \mathbf{dist}_G(x,s) + \mathbf{dist}_G(s,t) + \mathbf{dist}_G(t,y) \le \left(3(1+\epsilon)^2 h_l + 1\right) \cdot \mathbf{dist}_G(s,t)$$

Similarly, for the second inequality, since $\mathbf{TreeCover}(u_l, v_l, \mathcal{T}_l)$ does not return anything, by the guarantee of Lemma 10, we know that:

$$\mathbf{dist}_{G/\mathcal{C}^{(l)}}(u_l, v_l) > \frac{2}{3(1+\epsilon)^2 h_l} \cdot \max\{\mathbf{dist}_{G/\mathcal{C}^{(l)}}(u_l, B_{h_l}), \mathbf{dist}_{G/\mathcal{C}^{(l)}}(v_l, B_{h_l})\}$$

Thus, by triangle inequality, we have:

$$\begin{aligned}\mathbf{dist}_{G/\mathcal{C}^{(l)}}(x,y) &\le \mathbf{dist}_{G/\mathcal{C}^{(l)}}(x,u_l) + \mathbf{dist}_{G/\mathcal{C}^{(l)}}(u_l,v_l) + \mathbf{dist}_{G/\mathcal{C}^{(l)}}(v_l,y)\\ &\le \left(3(1+\epsilon)^2 h_l + 1\right) \cdot \mathbf{dist}_{G/\mathcal{C}^{(l)}}(u_l,v_l)\end{aligned}$$ ◁

Next, let us study two possibilities. First, assume that the shortest path between nodes x, y in $G/\mathcal{C}^{(l)}$ belongs to Π_l.

- $1 \le l \le \kappa$.
  In this case, using Claim 19, the path reported by our distance oracle has length at most:
  $$\mathbf{dist}_G(s,x) + \mathbf{dist}_G(x,y) + \mathbf{dist}_G(y,t) \le \left(6(1+\epsilon)^2 h_l + 1\right) \cdot \mathbf{dist}_G(s,t) \le 5k \cdot \mathbf{dist}_G(s,t)$$
- $\kappa < l \le \iota$.
  In this case, let π' be the concatenation of shortest paths between u_l, x, and x, y, and y, v_l in graph $G/\mathcal{C}^{(l)}$. Using Claim 19 and triangle inequality, we have:
  $$\begin{aligned}\mathbf{w}(\pi') &= \mathbf{dist}_{G/\mathcal{C}^{(l)}}(u_l,x) + \mathbf{dist}_{G/\mathcal{C}^{(l)}}(x,y) + \mathbf{dist}_{G/\mathcal{C}^{(l)}}(y,v_l)\\ &\le \left(6(1+\epsilon)^2 h_l + 1\right) \cdot \mathbf{dist}_{G/\mathcal{C}^{(l)}}(u_l,v_l)\end{aligned}$$

Applying Lemma 5 on π' and then we can obtain a path between s,t with length at most:

$$\begin{aligned}4\tau_l \cdot \left(6(1+\epsilon)^2 h_l + 1\right) \cdot \mathbf{dist}_{G/\mathcal{C}^{(l)}}(u_l, v_l) &\leq 4\tau_l \cdot \left(6(1+\epsilon)^2 h_l + 1\right) \cdot \mathbf{dist}_G(s,t) \\ &\leq 8\alpha^{-l/5} \cdot \left(6(1+\epsilon)^2 \left\lceil \alpha^l k \right\rceil + 1\right) \cdot \mathbf{dist}_G(s,t) \\ &< 50\alpha^{4l/5} k \cdot \mathbf{dist}_G(s,t) \\ &< 10k \cdot \mathbf{dist}_G(s,t)\end{aligned}$$

Next, consider the case where the shortest path between nodes x, y in $G/\mathcal{C}^{(l)}$ does not belong to Π_l. Consider two possibilities.

- $1 \leq l \leq \kappa$.
  In this case, using Claim 19, we know $\mathbf{dist}_G(x,y) \leq \left(3(1+\epsilon)^2 h_l + 1\right) \cdot \mathbf{dist}_G(s,t)$. Also, in the proof of Claim 19, we have also shown:

  $$\mathbf{dist}_G(s,t) > \frac{2}{3(1+\epsilon)^2 h_l} \cdot \max\left\{\mathbf{dist}_G\left(s,x\right), \mathbf{dist}_G\left(t,y\right)\right\}$$

  Now, since $y \notin \mathbf{Bun}_G^{1/2}\left(x, A_{h_{l-1}}\right)$ and $x \notin \mathbf{Bun}_G^{1/2}\left(y, A_{h_{l-1}}\right)$, we know that:

  $$\max\left\{\mathbf{dist}_G\left(x, A_{h_{l-1}}\right), \mathbf{dist}_G\left(y, A_{h_{l-1}}\right)\right\} \leq 2\mathbf{dist}_G(x,y)$$

  Therefore, we have:

  $$\begin{aligned}&\max\left\{\mathbf{dist}_G\left(s, A_{h_{l-1}}\right), \mathbf{dist}_G\left(t, A_{h_{l-1}}\right)\right\} \\ &\leq \max\left\{\mathbf{dist}_G(s,x) + \mathbf{dist}_G\left(x, A_{h_{l-1}}\right), \mathbf{dist}_G(t,y) + \mathbf{dist}_G\left(y, A_{h_{l-1}}\right)\right\} \\ &\leq \left(\frac{15(1+\epsilon)^2 h_l}{2} + 2\right) \cdot \mathbf{dist}_G(s,t)\end{aligned}$$

  As $\mathbf{TreeCover}\left(s,t,\mathcal{T}_l\right)$ successfully returns a nonempty value, by Lemma 10, the path retrieved between s,t has length at most ($k > 100, \epsilon < 0.1$):

  $$\begin{aligned}&\left(\frac{15(1+\epsilon)h_l}{(1-\epsilon)} + 4(1+\epsilon)\right) \cdot \mathbf{dist}_G(s,t) \\ &< \left(\frac{15(1+\epsilon)\left\lceil (0.75+\epsilon)k \right\rceil}{(1-\epsilon)} + 4(1+\epsilon)\right) \cdot \mathbf{dist}_G(s,t) \\ &< 12k \cdot \mathbf{dist}_G(s,t)\end{aligned}$$

- $l > \kappa$.
  In this case, using Claim 19, we know that $\mathbf{dist}_{G/\mathcal{C}^{(l)}}(x,y) \leq \left(3(1+\epsilon)^2 h_l + 1\right) \cdot \mathbf{dist}_{G/\mathcal{C}^{(l)}}(u_l, v_l)$. Also, in the proof of Claim 19, we have also show:

  $$\mathbf{dist}_{G/\mathcal{C}^{(l)}}(u_l, v_l) > \frac{2}{3(1+\epsilon)^2 h_l} \cdot \max\left\{\mathbf{dist}_{G/\mathcal{C}^{(l)}}\left(u_l, x\right), \mathbf{dist}_{G/\mathcal{C}^{(l)}}\left(v_l, y\right)\right\}$$

  Now, since $x \notin \mathbf{Bun}_{G/\mathcal{C}^{(l)}}(y, C_{h_{l-1}})$ and $y \notin \mathbf{Bun}_{G/\mathcal{C}^{(l)}}(x, C_{h_{l-1}})$, we know that:

  $$\max\left\{\mathbf{dist}_{G/\mathcal{C}^{(l)}}\left(x, C_{h_{l-1}}\right), \mathbf{dist}_{G/\mathcal{C}^{(l)}}\left(y, C_{h_{l-1}}\right)\right\} \leq 2\mathbf{dist}_{G/\mathcal{C}^{(l)}}(x,y)$$

  Therefore, we have:

  $$\begin{aligned}&\max\left\{\mathbf{dist}_{G/\mathcal{C}^{(l)}}\left(u_l, C_{h_{l-1}}\right), \mathbf{dist}_{G/\mathcal{C}^{(l)}}\left(v_l, C_{h_{l-1}}\right)\right\} \\ &\leq \max\left\{\mathbf{dist}_{G/\mathcal{C}^{(l)}}(u_l, x) + \mathbf{dist}_{G/\mathcal{C}^{(l)}}\left(x, C_{h_{l-1}}\right), \mathbf{dist}_{G/\mathcal{C}^{(l)}}(v_l, y) + \mathbf{dist}_{G/\mathcal{C}^{(l)}}\left(y, C_{h_{l-1}}\right)\right\} \\ &\leq \left(\frac{15(1+\epsilon)^2 h_l}{2} + 2\right) \cdot \mathbf{dist}_{G/\mathcal{C}^{(l)}}(u_l, v_l)\end{aligned}$$

Take an arbitrary vertex $z \in V$ contracted in the node $\mathbf{piv}_{G/\mathcal{C}^{(l)}}(u_l, C_{h_{l-1}})$. Then, applying Lemma 5, we can bound $\mathbf{dist}_G(s, A_{h_{l-1}})$ as:

$$\mathbf{dist}_G(s, A_{h_{l-1}}) \le \mathbf{dist}_G(s,z) \le 4\tau_l \cdot \left(\frac{15(1+\epsilon)^2 h_l}{2} + 2\right) \cdot \mathbf{dist}_{G/\mathcal{C}^{(l)}}(u_l, v_l)$$
$$< 8\alpha^{-l/5} \cdot 8\alpha^l k \cdot \mathbf{dist}_G(s,t) = 64\alpha^{4l/5} k \cdot \mathbf{dist}_G(s,t)$$

Similarly we can prove that $\mathbf{dist}_G(t, A_{h_{l-1}}) < 64\alpha^{4l/5} k \cdot \mathbf{dist}_G(s,t)$.
As $\mathbf{TreeCover}\,(u_{l-1}, v_{l-1}, \mathcal{T}_{l-1})$ successfully returns a nonempty value, by Lemma 10, the path retrieved between u_{l-1}, v_{l-1} in $G/\mathcal{C}^{(l-1)}$ has length at most ($k > 100, \epsilon < 0.1$):

$$2(1+\epsilon)\max\left\{\mathbf{dist}_{G/\mathcal{C}^{(l-1)}}\left(u_{l-1}, B_{h_{l-1}}\right), \mathbf{dist}_{G/\mathcal{C}^{(l-1)}}\left(v_{l-1}, B_{h_{l-1}}\right)\right\}$$
$$\le 2(1+\epsilon) \cdot \max\left\{\mathbf{dist}_G(s, A_{h_{l-1}}), \mathbf{dist}_G(t, A_{h_{l-1}})\right\}$$
$$< 130 \cdot \alpha^{4l/5} k \cdot \mathbf{dist}_G(s,t)$$

Then, applying Lemma 5 again, we can unpack this path and retrieve a path between s, t with weight at most (note that $l > \kappa = 50$ and $\alpha = \frac{3}{4} + \epsilon$):

$$4\tau_l \cdot 130 \cdot \alpha^{4l/5} k \cdot \mathbf{dist}_G(s,t) < 1040 \cdot \alpha^{3l/5} k \cdot \mathbf{dist}_G(s,t) < 10k \cdot \mathbf{dist}_G(s,t)$$

Low-level phase

Next, let us turn to the stretch if the query procedure is in the low-level phase. By the algorithm description, we should assume that the subroutine $\mathbf{TreeCover}\,(u_\iota, v_\iota, \mathcal{T}_\iota)$ returns a nonempty value.

▷ Claim 20. During the binary search procedure on the index pair (i_1, i_2), it always holds that $\mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(u, B_{i_1}) \le i_1 \cdot \mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(u, v)$. Plus, i_2 is always (u,v)-terminal.

Proof. The second half of the statement is straightforward. So, let us prove the first half by an induction. At the beginning of the algorithm, $i_1 = 0$, and thus $\mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(u, B_{i_1}) = 0$. In each iteration, if i_3 is (u,v)-terminal, then i_1 does not change, so the induction holds. Otherwise, since $\mathbf{piv}_{G/\mathcal{C}^{(\iota)}}(u, B_{i_3}) \notin \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(v, B_{i_3+1})$ and $\mathbf{piv}_{G/\mathcal{C}^{(\iota)}}(v, B_{i_3+1}) \notin \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(u, B_{i_3+2})$, we know that by definition of bunches:

$$\mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(v, B_{i_3+1}) \le \mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(u,v) + \mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(u, B_{i_3})$$

$$\mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(u, B_{i_3+2}) \le \mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(u,v) + \mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(v, B_{i_3+1})$$

Hence, $\Delta(u, i_3) \le 2\mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(u,v)$. Since $\Delta(u, i_3)$ is the maximum among $\Delta(u,i), i_1 \le i \le j-2$, we know that $\mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(u, B_j) \le j \cdot \mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(u,v)$. ◁

In the end, when the algorithm terminates, we have $i_1 = i_2 = i$. By the above claim, we know that $\mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(u, B_i) \le i \cdot \mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(u,v)$, and either $i = 2\left\lceil \log^2 k \right\rceil$ or i is (u,v)-terminal. Consider two possibilities.

- $i = 2\left\lceil \log^2 k \right\rceil$.
  In this case, as $\mathbf{TreeCover}(u, v, \mathcal{T}_\iota)$ successfully returned a nonempty value, by Lemma 10, the path length between u, v in $G/\mathcal{C}^{(\iota)}$ reported by our distance oracle is at most:

  $$2(1+\epsilon) \cdot \mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(u, B_i) < 5\log^2 k \cdot \mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(u,v)$$

  If we unpack the clusters in $\mathcal{C}^{(\iota)}$, according to Lemma 5, this path has length at most:

  $$20\tau_\iota \cdot \log^2 k \cdot \mathbf{dist}_G(u,v) < 40k^{1/5}\log^2 k \cdot \mathbf{dist}_G(u,v) < 10k \cdot \mathbf{dist}_G(u,v)$$

- $i < 2\lceil \log^2 k \rceil$.
  In this case, as i is (u,v)-terminal, we know that either $\mathbf{piv}_{G/\mathcal{C}^{(\iota)}}(u, B_i) \in \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(v, B_{i+1})$ or $\mathbf{piv}_{G/\mathcal{C}^{(\iota)}}(v, B_{i+1}) \in \mathbf{Bun}_{G/\mathcal{C}^{(\iota)}}(u, B_{i+1})$. Therefore, our data structure can report a path in $G/\mathcal{C}^{(\iota)}$ of length at most $(2i+1) \cdot \mathbf{dist}_{G/\mathcal{C}^{(\iota)}}(u,v)$ between u, v. If we unpack the clusters in $\mathcal{C}^{(\iota)}$, according to Lemma 5, this path has length at most:

$$4\tau_\iota \cdot (2i+1) \cdot \mathbf{dist}_G(u,v) < 10k \cdot \mathbf{dist}_G(u,v)$$

References

1. Michael A Bender and Martin Farach-Colton. The LCA problem revisited. In *LATIN 2000: Theoretical Informatics: 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000 Proceedings 4*, pages 88–94. Springer, 2000.
2. Michael A Bender and Martın Farach-Colton. The level ancestor problem simplified. *Theoretical Computer Science*, 321(1):5–12, 2004.
3. Marcel Bezdrighin, Michael Elkin, Mohsen Ghaffari, Christoph Grunau, Bernhard Haeupler, Saeed Ilchi, and Václav Rozhoň. Deterministic distributed sparse and ultra-sparse spanners and connectivity certificates. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 1–10, 2022.
4. Shiri Chechik. Approximate distance oracles with constant query time. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 654–663, 2014.
5. Shiri Chechik. Approximate distance oracles with improved bounds. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 1–10, 2015.
6. Don Coppersmith and Michael Elkin. Sparse sourcewise and pairwise distance preservers. *SIAM Journal on Discrete Mathematics*, 20(2):463–501, 2006.
7. Michael Elkin and Seth Pettie. A linear-size logarithmic stretch path-reporting distance oracle for general graphs. *ACM Transactions on Algorithms (TALG)*, 12(4):1–31, 2016.
8. Michael Elkin and Idan Shabat. Path-Reporting Distance Oracles with Logarithmic Stretch and Size $O(n \log\log n)$. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2278–2311. IEEE, 2023.
9. Manor Mendel and Assaf Naor. Ramsey partitions and proximity data structures. *Journal of the European Mathematical Society*, 9(2):253–275, 2007.
10. Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005.
11. Christian Wulff-Nilsen. Approximate distance oracles with improved query time. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 539–549. SIAM, 2013.

A Missing proofs

▶ **Lemma 21** (restate Lemma 9). *Given an undirected weighted graph $G = (V, E, \mathbf{w})$ and a collection of shortest paths Π, there is a distance-preserving path-reporting data structure that reports any path $\pi \in \Pi$ in time $|\pi|$. The data structure has size $O\left(n + |\Pi| + \sqrt{n|\mathbf{br}(\Pi)|}\right)$ where $\mathbf{br}(\Pi)$ is the set of branching events of Π.*

Proof. First, for any path $\pi \in \Pi$, store both of endpoints in V as well as its first and last edges. Since all paths in Π are shortest paths, there are no two paths sharing the same pair of endpoints. For any path $\pi \in \Pi$, we can identify π with its starting and ending vertex which is denoted by $\mathsf{ID}(\pi)$. This part takes space $O(n + |\Pi|)$.

We will build a routing table data structure $\mathcal{D}_u$ at any vertex $u \in V$. That is, for any path $\pi \in \Pi$ that passes through u through a pair of edges $(v,u),(u,w)$, given the ID of π, the routing data structure $\mathcal{D}_u$ is able to answer the edges $(v,u),(u,w)$ in constant time. Once this is goal is fulfilled, as we have also stored starting and ending edges of each path, our path-reporting distance preserver is complete. For the rest, let us focus on the design of $\mathcal{D}_u$.

For any vertex $u \in V$, let Π_u be the set of paths in Π that pass through u, and for any unordered pairs of different neighbors v, w of u in G, let $\Pi_u^{\{v,w\}}$ be the set of paths that use edges $(v,u),(u,w)$ to go through u, and so $|\Pi_u| = \sum_{v \neq w} \left|\Pi_u^{\{v,w\}}\right|$, and also by definition of branching events, the number of branchings at x is equal to:

$$|\mathbf{br}(\Pi_u)| = \sum_{\{v,w\} \neq \{v',w'\}} \left|\Pi_u^{\{v,w\}}\right| \cdot \left|\Pi_u^{\{v',w'\}}\right| = \frac{1}{2}\left(|\Pi_u|^2 - \sum_{v \neq w}\left|\Pi_u^{\{v,w\}}\right|^2\right)$$

Consider two different cases.

- For any vertex pair v, w, we have $\left|\Pi_u^{\{v,w\}}\right| \leq \frac{1}{2} \cdot |\Pi_u|$.
  In this case, for each path $\pi \in \Pi_u$ using edges $(v,u),(u,w)$, store a triple $(\mathsf{ID}(\pi), v, w)$; all such triples will be stored in a hash table that supports constant-time queries using path IDs.
  As for the space of $\mathcal{D}_u$, on the one hand, it is $O(|\Pi_u|)$. On the other hand, under this condition we have:
  $$|\mathbf{br}(\Pi_u)| = \frac{1}{2}\left(|\Pi_u|^2 - \sum_{v \neq w}\left|\Pi_u^{\{v,w\}}\right|^2\right) \geq \frac{1}{2}\left(|\Pi_u|^2 - \frac{1}{2}|\Pi_u|^2\right) = \frac{1}{4}|\Pi_u|^2$$
  Therefore, $|\mathcal{D}_u| = O\left(\sqrt{|\mathbf{br}(\Pi_u)|}\right)$.
- There exists a vertex pair v_*, w_* such that $\left|\Pi_u^{\{v_*,w_*\}}\right| > \frac{1}{2} \cdot |\Pi_u|$.
  In this case, for each path $\pi \in \Pi_u$ using edges $(v,u),(u,w)$ such that $\{v,w\} \neq \{v_*,w_*\}$, store a triple $(\mathsf{ID}(\pi), v, w)$; all these triples will be stored in a hash table that supports constant-time queries using path IDs. For those paths π which pass through u using edges $(v_*,u),(u,w_*)$, we can query this hash table with $\mathsf{ID}(\pi)$ which returns nothing, and then answer the query with $\{(v_*,u),(u,w_*)\}$.
  As for the size of $\mathcal{D}_u$, on the one hand, its space is $O\left(\Pi_u \setminus \Pi_u^{\{v_*,w_*\}}\right)$. On the other hand, we have:
  $$\begin{aligned}|\mathbf{br}(\Pi_u)| &= \frac{1}{2}\left(|\Pi_u|^2 - \sum_{v \neq w}\left|\Pi_u^{\{v,w\}}\right|^2\right) \geq \frac{1}{2} \cdot \left(|\Pi_u|^2 - \left|\Pi_u^{\{v_*,w_*\}}\right|^2 - \left|\Pi_u \setminus \Pi_u^{\{v_*,w_*\}}\right|^2\right) \\ &\geq \left|\Pi_u^{\{v_*,w_*\}}\right| \cdot \left|\Pi_u \setminus \Pi_u^{\{v_*,w_*\}}\right| \geq \left|\Pi_u \setminus \Pi_u^{\{v_*,w_*\}}\right|^2\end{aligned}$$
  Therefore, we also have $|\mathcal{D}_u| = O\left(\sqrt{\mathbf{br}(\Pi_u)}\right)$

In either case, we are able to show $|\mathcal{D}_u| = O\left(\sqrt{\mathbf{br}(\Pi_u)}\right)$, and so the total size can be bounded by:

$$\sum_{u \in V} |\mathcal{D}_u| \leq \sqrt{n \sum_{u \in V} |\mathcal{D}_u|^2} = O\left(\sqrt{n|\mathbf{br}(\Pi)|}\right)$$ ◀

Robot Positioning Using Torus Packing for Multisets

Chung Shue Chen[1]
Nokia Bell Labs, Nozay, France

Peter Keevash
Mathematical Institute, University of Oxford, UK

Sean Kennedy
Nokia Bell Labs, Ottawa, Canada

Élie de Panafieu
Nokia Bell Labs, Nozay, France

Adrian Vetta
McGill University, Montreal, Canada

Abstract

We consider the design of a positioning system where a robot determines its position from local observations. This is a well-studied problem of considerable practical importance and mathematical interest. The dominant paradigm derives from the classical theory of de Bruijn sequences, where the robot has access to a window within a larger code and can determine its position if these windows are distinct. We propose an alternative model in which the robot has more limited observational powers, which we argue is more realistic in terms of engineering: the robot does not have access to the full pattern of colours (or letters) in the window, but only to the intensity of each colour (or the number of occurrences of each letter). This leads to a mathematically interesting problem with a different flavour to that arising in the classical paradigm, requiring new construction techniques. The parameters of our construction are optimal up to a constant factor, and computing the position requires only a constant number of arithmetic operations.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms

Keywords and phrases Universal cycles, positioning systems, de Bruijn sequences

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.43

Category Track A: Algorithms, Complexity and Games

Related Version *Extended Version*: `https://arxiv.org/abs/2404.09981` [8]

Supplementary Material *Software*: `https://gitlab.com/depanafieuelie/torus-packing-for-multisets` [13], archived at `swh:1:dir:2eb591ca09fb8739e2135e17b5f595b7f25ed924`

Funding *Peter Keevash*: Supported by ERC Advanced Grant 883810.
Adrian Vetta: NSERC Discovery Grant 04191

Acknowledgements Part of the work was carried out at the Laboratory for Information, Networking and Communication Sciences (`https://www.lincs.fr`). We thank Dr. Paolo Baracca, Siu-Wai Ho, Kenneth Shum and many colleagues for their great support and discussions.

[1] Authors presented in alphabetical order.

1 Introduction

Consider a robot located on a grid of coloured squares that must determine its position after observing part of the grid through a fixed viewing window. The dominant paradigm for this problem derives from the mathematical theory of de Bruijn sequences, i.e. binary (or bi-coloured) cyclic sequences of length 2^n in which each binary sequence of length n appears exactly once as a subsequence of consecutive entries. Such a sequence can be used for positioning a robot in one dimension: the robot sees a viewing window of length n; this window induces a subsequence with a unique colour pattern; from this colour pattern the robot can then reconstruct its position in the sequence (if we disregard issues of computational efficiency and error correction). Generalisations of this idea to higher dimensions and related combinatorial structures have led to a rich mathematical theory; see Section 1.2.

However, while this theory is mathematically pleasing, we will argue in Section 1.3 that engineering constraints support a model in which the robot does not have access to the full colour pattern in the window, but must infer its position only knowing the intensity of each colour, that is, the multiset of colours. More precisely, given an $n \times n$ grid, a robot with an $m \times m$ viewing window, and a palette of k colours, our task is to colour the grid so that each possible location of the viewing windows produces a different multiset of colours. Furthermore, it will be mathematically more natural to undertake this task for a torus of side n rather than a grid, and also to generalise to an arbitrary dimension d.

► **Example 1.** The grid colouring illustrated in Figure 1 has dimension $d = 2$, size $n = 8$, window size $m = 4$, and $k = 3$ colours (red 0, green 1, and blue 2). No two 4×4 subsquares contain the same multiset of colours. For example, two viewing windows are shown with multisets that have multiplicities for $(0, 1, 2)$ equal to $(6, 2, 8)$ and $(3, 5, 8)$, respectively. Observe that the second window "*wraps around*" because the grid is considered as a torus.

2	2	2	2	2	1	2	1
2	2	2	2	1	2	1	2
2	2	2	2	2	1	2	2
2	2	2	2	1	2	1	2
0	2	0	2	0	1	0	1
2	0	2	0	1	0	1	0
0	2	0	2	0	1	0	2
2	0	2	2	1	0	1	2

2	2	2	2	2	1	2	1
2	2	2	2	1	2	1	2
2	2	2	2	2	1	2	2
2	2	2	2	1	2	1	2
0	2	0	2	0	1	0	1
2	0	2	0	1	0	1	0
0	2	0	2	0	1	0	2
2	0	2	2	1	0	1	2

2	2	2	2	2	1	2	1
2	2	2	2	1	2	1	2
2	2	2	2	2	1	2	2
2	2	2	2	1	2	1	2
0	2	0	2	0	1	0	1
2	0	2	0	1	0	1	0
0	2	0	2	0	1	0	2
2	0	2	2	1	0	1	2

Figure 1 A grid coloring of dimension $d = 2$, size $n = 8$, window size $m = 4$. Observe that the two windows depicted contain distinct multisets of colours.

Our question of study is, given m, d, k, what is the largest grid size n for which position reconstruction is possible? Unfortunately, even the one-dimensional version of this question is the subject of several unsolved problems, discussed in Section 1.2. However, for practical purposes one can be content to relax from finding the optimal n given m, d, k to a value that is optimal up to a constant factor, where we think of d and k as fixed and consider the asymptotics for large m and n. There is a clear information theoretic barrier (see Observation 10) at $n = \Theta_{k,d}(m^{k-1})$, where the subscripts indicate that k and d are constants. The main contribution of this paper is a construction that achieves this theoretical optimum up to a constant factor, and moreover has optimal computational efficiency, in that only a constant number of arithmetic operations are required to compute the location of the window from its multiset of colours. We implemented and tested this construction in Python [13].

1.1 Definitions and Results

Notations. All vectors, tuples and sequences are indexed starting at 0. Integer intervals are denoted with square brackets, such as $[0, n-1]$. Given a length d, the vector $\boldsymbol{e}_i$ has all coordinates equal to 0, except the ith, which is equal to 1. The value of $\mathbb{1}_{\text{condition}}$ is 1 if *condition* is satisfied, and 0 otherwise. We write $i \equiv j \bmod n$ when the integer $i - j$ is divisible by n. Tables are represented with row indices increasing from top to bottom and column indices increasing from left to right, both starting at 0.

We first present general definitions that will be useful for discussing the literature and presenting our construction, starting with *cycle packings* for the one-dimensional case, and continuing with their higher dimensional generalisation, *torus packings*. Then the central objects of this article, *grid colouring*, are formally defined as particular cases of torus packings. Our main result, Theorem 9, is a near-optimal construction for grid colourings. It will make use of *vector sum packings* (Definition 11), which are particular cases of cycle packings.

▶ **Definition 2.** *Given an alphabet $\mathcal{A}$, a size n, a window size m, and a function f on $\mathcal{A}^m$, an $(\mathcal{A}, f, n, m)$-**cycle packing** is a function $W : \mathbb{Z} \mapsto \mathcal{A}$ that satisfies*

- `Periodicity.` *For all $x \in \mathbb{Z}$ we have $W_x = W_{x+n}$.*
- `Injectivity.` *If $f(W_x, W_{x+1}, \ldots, W_{x+m-1}) = f(W_y, W_{y+1}, \ldots, W_{y+m-1})$, for any integers x and y, then $x \equiv y \bmod n$.*

Let us explain this formal definition. Consider a circle composed of n squares, each receiving a letter from $\mathcal{A}$. A robot located on this circle wants to recover its position. It makes a local measurement, function f of the letters in a window of size m around it. The injectivity condition ensures that two positions of the robot (and thus of the window) correspond to two distinct values of f. Thus the robot is always able to deduce its position. Observe that the form of function f matters as different functions induce different types of information that robot can extract from the viewing window. To illustrate this point, consider the following three examples of $(\mathcal{A}, f, n, m)$-cycle packings. These examples all have $\mathcal{A} = \{0, 1, 2\}$ and $m = 3$ but differ in their functions f (which in turn lead to varying sizes of n).

▶ **Example 3.** Take the basic case where f is the identity function. The robot thus receives the entire colour pattern in its viewing window. Thus a cycle packing corresponds to a sequence of length n where all the contiguous subsequences of length m have distinct colour patterns. With $\mathcal{A} = \{0, 1, 2\}$ and window size $m = 3$, we have that $(0, 1, 1, 1, 2, 1, 0, 1, 2, 0, 1, 0, 2, 1, 1, 0, 0, 2, 2, 1, 2, 2, 2, 0, 2, 0, 0)$ is a cycle-packing of size $n = 27$ as each possible string of length 3 appears at most once; thus, the injectivity property holds. In fact, each such string appears *exactly* once (*e.g.* the factor $(0, 0, 0) = f(0, 0, 0)$ appears by wrapping around), making it a de Bruijn sequence. This exactness property is not required for cycle packings, but when it is satisfied the cycle packing is called a *universal cycle*.

▶ **Example 4.** Recall our motivation is that the robot extracts the colour intensities rather than the entire colour pattern. So instead of the identity function, assume that f is the multiset counting function which simply counts the number of appearances of each colour in the viewing window. Now, for the alphabet $\mathcal{A} = \{0, 1, 2\}$ with window size $m = 3$, the sequence $(0, 1, 1, 1, 2, 2, 2, 0, 0)$ is a cycle packing of size $n = 9$ as every multiset appears at most once (*e.g.* the multiset $(3, 0, 0) = f(0, 0, 0)$ arises by wrapping around) and the injectivity property holds. However, this cycle packing is not a universal cycle as no viewing window contains all three colours, so the multiset $(1, 1, 1)$ does not appear.

▶ **Example 5.** Again take $\mathcal{A} = \{0, 1, 2\}$ with window size $m = 3$, but now let f be the summation function. We claim the sequence $(0, 0, 0, 2, 2, 2, 1)$ is a cycle packing of size $n = 7$. To see this, observe that $f(0, 0, 0) = 0 + 0 + 0 = 0$, $f(0, 0, 2) = 0 + 0 + 2 = 2$, etc. Continuing these calculations, the outputted sequence of sums is $(0, 2, 4, 6, 5, 3, 1)$. This has distinct entries, so injectivity is satisfied. The reader may query the relevance of the summation function. In fact, this example is a special case of vector sum packings (Definition 11) which, in turn, will play a critical role in the construction underlying our main theorem.

Of course, our interest lies in dimension $d > 1$, so we now extend the definition of cycle packings to higher dimensions.

▶ **Definition 6.** *Given an alphabet $\mathcal{A}$, a size n, a window size m, a dimension d, and a function f on $\mathcal{A}^{m^d}$, an $(\mathcal{A}, f, n, m, d)$-**torus packing** is a function $W : \mathbb{Z}^d \mapsto \mathcal{A}$ that satisfies*

- `Periodicity`. *For all $\boldsymbol{x} \in \mathbb{Z}^d$ and $j \in [0, d-1]$, we have $W_{\boldsymbol{x}} = W_{\boldsymbol{x}+n\boldsymbol{e}_j}$ where $\boldsymbol{e}_j$ denotes the vector with a 1 in position j and 0 elsewhere.*
- `Injectivity`. *For any $\boldsymbol{x}$ and $\boldsymbol{y}$ in $\mathbb{Z}^d$, if $f\left((W_{\boldsymbol{x}+\boldsymbol{c}})_{\boldsymbol{c}\in[0,m-1]^d}\right) = f\left((W_{\boldsymbol{y}+\boldsymbol{c}})_{\boldsymbol{c}\in[0,m-1]^d}\right)$, then $x_j \equiv y_j \bmod n$ for all j (where x_j denote the jth coordinate of the vector $\boldsymbol{x}$).*

Note the key distinction in this definition is that the viewing window shares the same higher dimension as the torus. Again, by the periodicity property, we can identify a torus packing W with its pattern $(W_{\boldsymbol{x}})_{\boldsymbol{x}\in[0,n-1]^d}$. We are now ready to define the central objects of this paper. These are grid colourings, a particular case of torus packing that correspond exactly to the robot position reconstruction problem.

▶ **Definition 7.** *An (n, m, d, k)-**grid colouring** is an $(\mathcal{A}, f, n, m, d)$-torus packing with $|\mathcal{A}| = k$ and f mapping any sequence to the multiset of its entries.*

▶ **Example 8.** Consider, again, our example in Figure 1. This is an $(\mathcal{A}, f, n, m, d)$-torus packing with $\mathcal{A} = \{0, 1, 2\}$, $n = 8$, $m = 4$, $d = 2$ and where f is the multiset counting function. Consequently, since $|\mathcal{A}| = 3$, this is an $(8, 4, 2, 3)$-grid colouring.

The following is our main result on grid colourings, implemented and tested in Python [13].

▶ **Theorem 9.** *Fix a dimension $d \geq 2$ and a number of colours k of the form $bd+1$ for some $b \geq 1$. For any window size m multiple of $2(k-1)$, there is an (n, m, d, k)-grid colouring W (explicitly constructed in the proof) with*

$$n \sim C_k^{1/d} \cdot m^{k-1} \qquad \text{where} \qquad C_k = \left(\frac{2}{k-1}\right)^{k-1}.$$

Furthermore, for any $\boldsymbol{x}$ in $\mathbb{Z}^d$, given the multiset of colours in $(W_{\boldsymbol{x}+\boldsymbol{c}})_{\boldsymbol{c}\in[0,m-1]^d}$ one can compute $\boldsymbol{x}$ mod n with $O_{k,d}(1)$ arithmetic operations.

Our construction in Theorem 9 of size $n = \Omega_{k,d}(m^{k-1})$ is optimal up to a multiplicative constant. This fact follows from the following observation, which is immediate from counting considerations (injectivity requires the number of possible colour multisets to be at least the number of windows that they must distinguish).

▶ **Observation 10.** *The parameters of any (n, m, d, k)-grid colouring satisfy the inequality*

$$n^d \leq \binom{m^d + k - 1}{k - 1}.$$

In particular, for fixed dimension d and number of colours k, as the window size m tends to infinity, we have

$$n \leq C_k'^{1/d} \cdot m^{k-1}(1 + \mathcal{O}(m^{-1})) \qquad \text{with} \qquad C_k' = \frac{1}{(k-1)!}.$$

We conclude this section by discussing related work and providing technological justifications for our positioning model.

1.2 Related Work

The combinatorial structures associated with torus packings (Definition 6) have a rich mathematical literature, starting from a problem solved in the 19th century, independently rediscovered by de Bruijn and now known as *de Bruijn sequences* (see [12]). The generalisation to higher dimensions was independently considered in several papers starting from the 1960's (see [24]) and has developed an extensive literature (see *e.g.* [17, 18, 20]) under the name of *de Bruijn tori*. The extension to general combinatorial structures encoded by sequences as in Definition 6 was proposed by Chung, Graham and Diaconis [9], who considered the one-dimensional problem ("*universal cycles*") for a variety of combinatorial structures. These early formulations of the problem generally asked for optimal solutions in which every object in a given combinatorial class is realised exactly once; in the context of Definition 6 this corresponds to strengthening injectivity to bijectivity. However, for practical purposes one can be satisfied with approximately optimal solutions, and given the difficulty of finding optimal solutions there is also a substantial literature (see *e.g.* [3, 10, 11]) finding approximate solutions from the perspective of packing (injectivity) and covering (surjectivity).

For the multiset encoding problem considered in this paper, finding the exact optimum is an open problem even for the one-dimensional setting of universal cycles, considered by Knuth [21, Fascicle 3, Section 7.2.1.3, Problem 109]. Hurlbert *et al.* [19] construct universal cycles for multisets with particular parameters, and Blanca and Godbole [3] consider the problem when the multisets have bounded multiplicities. Furthermore, the known results only consider the opposite regime of parameters from those needed for our application: we consider small palettes of colours (*a.k.a.* alphabets) and a large window, whereas previous techniques in the literature only apply to small windows and large alphabets. This comment also applies to the problem replacing "multiset" by "set", which is perhaps even more natural mathematically, given that it can be viewed as a hypergraph version of the Euler tour problem (introduced by Euler in the 18th century). Following many partial results, an exact solution to this problem (for small windows and large alphabets) was found by Glock *et al.* [14], solving a conjecture from Chung *et al.* [9]. We are unaware of any results in the literature on multiset packing in one dimension (meaning maximizing the number of multisets that appear in a sequence, rather than looking for sequences that contain all possible multisets).

Moving on from the abstract mathematical problem, we now consider some of the computer science literature aimed towards the specific application of indoor positioning. Much of the early work was surveyed by Burns and Mitchell [5]. More recent literature (see *e.g.* [2, 4, 6, 25]) has emphasised two further conditions that do not appear in the mathematical formulation but are naturally desirable for practical implementations: namely (a) computational efficiency, and (b) robustness against measurement errors. These works bring a variety of techniques from coding theory to bear on the positioning problem, providing efficient positioning algorithms with error correction. However, while these algorithms make natural use of an existing toolkit and are conceptually pleasing, we will argue that they are not addressing the most appropriate formulation of the problem from the point of view of the target application of robot positioning, for which a new paradigm is needed.

1.3 Motivation and Engineering Aspects

The focus of this paper is theoretical, concerning near-optimal designs for elegant combinatorial structures, namely torus packings. However, as stated, our motivation derives from the problem of position reconstruction (where the dimension is 2 and the torus of is considered a square). The vast number of applications of localization and positioning have prompted the design of a plethora of systems; see [31] and references therein.

Given the great practical importance of positioning systems, it behoves us to justify our claim that the approporiate way to model them is with multiset counting functions. We do this in this section by summarising two systems where our code (and variants of it) will be useful.

Light-Based Positioning. Recently, visible light based positioning systems have gained much attention [23, 26, 31] for two key reasons: (i) there is a strong demand for accurate but low-cost solutions, and (ii) there have been breakthroughs in the energy consumption and life expectancy of light-emitting diodes (LEDs). Contrary to systems requiring several light emitters and relying on triangulation [26], systems based on universal torus packing (such as de Bruijn torus [1, 27, 28] and our construction) rely on only one light source, reducing cost and energy consumption. Consider a room lit by a light-emitting diode (LED). A robot moving on the floor wears a light sensor or camera. A printed film is placed on top of the robot, above this sensor. The film is printed with a coloured grid, distorted so that depending on the position of the robot in the room, the light coming from the LED projects a square window of the coloured grid on the sensor [16]. If the coloured grid code depends on the respective positions of the colours (such as de Bruijn torus), the light detector must be a camera [22, 30] to recognize the pattern. If a torus packing for multisets is used instead, the robot can wear a simple light intensity sensor to recover the multiset of colours: the pattern of the colours is not needed, and we side-step the problematic issue of resolving the image. Such a device is considerably less expensive and has much shorter response time (lower latency) than a camera. It should be noted that the total light intensity (number of coloured rays) received by the detector depends on the position and also the orientation of the robot. Thus, for practical application of the construction presented in this article some redundancy should be added to allow for a unique position decoding in all cases. Positioning systems based on a de Bruijn torus also require redundancy to correct errors [6] and allow different orientations of the receiver [29]. We plan to explore redundancy in universal torus packing for multisets in future work.

Ambient Backscatters. An ambient backscatter is a small and inexpensive device that, upon reception of a radio wave, turns it into electricity and sends back data through a radio signal. Consider a warehouse where ambient backscatter devices are regularly placed, forming a grid. A robot with a radio emitter and receptor needs to locate itself in the warehouse. Each backscatter device, when in reach of the emitted radio wave, sends back its identification number (ID). If all devices have distinct IDs, the robot can determine its position based on the signals it receives. However, in the interest of energy consumption, IDs composed of few bits are preferred [7]. The same ID can be reused by several devices, provided that at any position, the multiset of IDs detected by the robot is unique. The relative position of the robot to its neighbouring backscatter devices is unknown, so the IDs should be chosen following a universal torus packing for multisets (a de Bruijn torus would require the robot to have directional antennas, making the system less practical and more expensive). For this application, our construction should be adapted to account for radial symmetry and decay of signal power in the detection window.

2 Overview

We have reduced the position reconstruction problem to the design of grid colourings. Consequently our main result, Theorem 9, is based upon an near-optimal grid colouring construction. A key building block in this construction will be the following family of cycle packings, which we call vector sum packings.

▶ **Definition 11.** *Given positive integers n, m, b and s, an (n, m, b, s)-**vector sum packing** is an $(\mathcal{A}, f, n, m)$-cycle packing with $\mathcal{A} = [0, s]^b$ and $f(\boldsymbol{z}_0, \ldots, \boldsymbol{z}_{m-1}) = \boldsymbol{z}_0 + \cdots + \boldsymbol{z}_{m-1}$.*

▶ **Example 12.** For the special case of *vector dimension* $b = 1$ we have already encountered a $(8, 3, 1, 2)$-vector sum packing in Example 5.

▶ **Example 13.** Let us see an example of a vector sum packing with vector dimension $b = 3$. Set $m = 2$, $s = 1$, $n = 8$ and $(\boldsymbol{z}_0, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_7) = \left(\begin{pmatrix}0\\0\\0\end{pmatrix}, \begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix}, \begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\1\end{pmatrix}, \begin{pmatrix}0\\0\\0\end{pmatrix}\right)$. Then $f(\boldsymbol{z}_0, \boldsymbol{z}_1) = \begin{pmatrix}1\\0\\0\end{pmatrix}$, $f(\boldsymbol{z}_1, \boldsymbol{z}_2) = \begin{pmatrix}1\\1\\0\end{pmatrix}$, $f(\boldsymbol{z}_2, \boldsymbol{z}_3) = \begin{pmatrix}0\\1\\0\end{pmatrix}$, $f(\boldsymbol{z}_3, \boldsymbol{z}_4) = \begin{pmatrix}0\\0\\1\end{pmatrix}$, $f(\boldsymbol{z}_4, \boldsymbol{z}_5) = \begin{pmatrix}1\\0\\1\end{pmatrix}$, $f(\boldsymbol{z}_5, \boldsymbol{z}_6) = \begin{pmatrix}1\\1\\1\end{pmatrix}$, $f(\boldsymbol{z}_6, \boldsymbol{z}_7) = \begin{pmatrix}0\\1\\1\end{pmatrix}$, and $f(\boldsymbol{z}_7, \boldsymbol{z}_0) = \begin{pmatrix}0\\0\\0\end{pmatrix}$. As these sums all differ, the injectivity property holds and we have a vector sum packing.

The strategy of our construction is encapsulated in the following two key lemmas (proven in Section 3 and Section 4, respectively). The first reduces the construction of a grid colouring to the construction of a vector sum packing.

▶ **Lemma 14.** *Consider a dimension d, size n, window size m, and number of colours k, and assume the existence of integers b and s satisfying $k = bd + 1$ and $s = \frac{m^{d-1}}{bd}$. Then the existence of an (n, m, b, s)-vector sum packing implies the existence of a (n, m, d, k)-grid colouring.*

Given Lemma 14, the final piece in the puzzle is a construction of vector sum packings.

▶ **Lemma 15.** *For any $s \geq 1$, $m \geq 2$ and $b \geq 1$, there exists an $(n_b, m, b, 2s)$-vector sum packing with*

$$n_b = (2ms + 1)^{b-1}\left(2ms - \frac{1}{s}\right) + \frac{1}{s}.$$

To deduce the existence of the construction for Theorem 9, we combine Lemmas 14 and 15, choosing m as a large multiple of $2bd$, with $k = bd + 1$ and $s = \frac{m^{d-1}}{2bd}$. Then the value of n_b in Lemma 15 satisfies $n_b \sim \left(\frac{2}{k-1}\right)^{(k-1)/d} m^{k-1}$ for large m. We will prove in Corollary 22 that the position can be computed with a constant number of arithmetic operations.

We remark that the vector sum packing problem is related to the combinatorial theory of *antimagic labellings* (a natural variant on the classic topic of *magic labellings*) in which one is required to label the edges (or vertices) of a graph (or hypergraph) from a given set of integers (or vectors) so that vertices (or edges) are uniquely determined by the sum of their incident labels. A well-known conjecture of Ringel (cited in [15]) states that for any connected graph with $m > 1$ edges there is an antimagic labelling of its edges by $\{1, \ldots, m\}$. It appears that this connection has not previously been exploited and may be fruitful for further research.

3 From Vector Sum Packing to Grid Colouring

The aim of this section is to prove Lemma 14, which reduces the grid colouring problem to the vector sum packing problem.

3.1 The Separation Property

In this section, we consider a fixed dimension $d \geq 2$, window size $m \geq 2$ and grid size $n \geq m$. For $\boldsymbol{x} \in \mathbb{Z}^d$, let us define the *window* of corner $\boldsymbol{x}$ as the set of points

$$\text{Window}(\boldsymbol{x}) = \{\boldsymbol{x} + \boldsymbol{c} \mid \boldsymbol{c} \in [0, m-1]^d\}.$$

For the grid colouring problem, we require that the coordinates of each point $\boldsymbol{x}$ are uniquely determined modulo n by the multiset of colours of the points from $\text{Window}(\boldsymbol{x})$. This multiset is denoted by $\text{colourMultiset}(\boldsymbol{x})$, and the colour of the point $\boldsymbol{x}$ is denoted by $\text{colour}(\boldsymbol{x})$.

We now present a sufficient condition, called *separation*, that guarantees this property. Each colour is represented as a pair (pigment, shade). We divide the set of colours into d pigment classes, one pigment class C_i for each dimension $i \in [0, d-1]$. Each pigment class contains b shades. Thus $C_i = \{c_{i,0}, c_{i,1}, \ldots, c_{i,b-1}\}$ where if i is the green pigment, say, then $c_{i,0}$ represents very light green and $c_{i,b-1}$ represents very dark green. The idea now is that for each square $\boldsymbol{x} \in \mathbb{Z}^d$, the coordinate x_i modulo n is uniquely determined by the pigment class C_i, for each $i \in [0, d-1]$, precisely by $\text{colourMultiset}(\boldsymbol{x}) \cap C_i$. In particular, x_i is independent of any other pigment class $j \neq i$ on $\text{Window}(\boldsymbol{x})$, for example, shades of red. Evidently, this separation property implies that if $\boldsymbol{x}$ and $\boldsymbol{y}$ have the same colour multiset for every pigment class then $x_i \equiv y_i \bmod n$ for each $i \in [0, d-1]$ and, hence, $\boldsymbol{x} \equiv \boldsymbol{y} \bmod n$. Thus the separation property implies that we have a proper grid colouring.

Rather than working directly with separation, it will be convenient to consider the following two further conditions that together clearly imply separation.

`Dimensional Inconsistency`: Given $\boldsymbol{x}$ and $\boldsymbol{y}$, if $x_i \neq y_i \bmod n$ then

$$\text{colourMultiset}(\boldsymbol{x}) \cap C_i \neq \text{colourMultiset}(\boldsymbol{y}) \cap C_i.$$

`Anti-Dimensional Consistency`: Given $\boldsymbol{x}$ and $\boldsymbol{y}$, if $x_i = y_i \bmod n$ then

$$\text{colourMultiset}(\boldsymbol{x}) \cap C_i = \text{colourMultiset}(\boldsymbol{y}) \cap C_i.$$

To ensure anti-dimensional consistency it will be convenient to focus on the following condition called *quasi-periodicity*, which states that if some $\boldsymbol{x}$ is coloured a shade of pigment i then any translate $\boldsymbol{y}$ of $\boldsymbol{x}$ by distance m in any dimension other than i has the same colour as $\boldsymbol{x}$

$$\forall i \neq j, \text{ if } \text{colour}(\boldsymbol{x}) \in C_i \text{ then } \text{colour}(\boldsymbol{x} + m\,\boldsymbol{e}_j) = \text{colour}(\boldsymbol{x}).$$

▶ **Lemma 16.** *Any quasi-periodic grid colouring satisfies anti-dimensional consistency.*

Proof. Take a quasi-periodic colouring. It suffices to show, given x_i, that $\text{colourMultiset}(\boldsymbol{x}) \cap C_i$ is fixed. This will hold if the number of squares of colour $c_{i,\ell}$ in a window does not change if we translate by one square in any dimension j other than i. Let us define

$$A = \text{Window}(\boldsymbol{x}) \setminus \text{Window}(\boldsymbol{x} + \boldsymbol{e}_j), \qquad B = \text{Window}(\boldsymbol{x} + \boldsymbol{e}_j) \setminus \text{Window}(\boldsymbol{x}).$$

Since B is obtained from A by translation by $m\boldsymbol{e}_j$, quasi-periodicity implies that the number of points of colour $c_{i,\ell}$ in A and B are equal. Thus, the number of points of colour $c_{i,\ell}$ in $\text{Window}(\boldsymbol{x})$ and $\text{Window}(\boldsymbol{x} + \boldsymbol{e}_j)$ are equal as well. As this argument applies for any $j \neq i$, the anti-dimensional consistency property holds. Thus, quasi-periodicity implies anti-dimensional consistency. ◀

3.2 Separation via Vector Sum Packing

Assume we have an (n, m, b, s)-vector sum packing $(\boldsymbol{z}_j)_{j\in\mathbb{Z}}$ of size n and window size m, containing vectors in $[0, s]^b$, where s is a positive integer equal to $\frac{m^{d-1}}{bd}$ for some dimension $d \geq 2$. We will now construct a (n, m, d, k)-grid colouring with number of colours $k = bd + 1$. We detail our algorithm below and illustrate it in Figure 2. As we saw in the previous section, to ensure the separation condition, it is sufficient that our grid colouring satisfies dimensional inconsistency and quasi-periodicity.

(a) We begin with an initial colouring of the grid using only d pigments, each coming in b different shades. Take a pigment $i \in [0, d-1]$ and shade $h \in [0, b-1]$. Then the point $\boldsymbol{x} = (x_0, \ldots, x_{d-1})$ has pigment i and shade h if and only if

$$\sum_{j=0}^{d-1} x_j \equiv i + hd \bmod bd.$$

This initial colouring is quasi-periodic. But, of course, it does not satisfy dimensional inconsistency. To rectify this, we will apply the last unused colour, which we call *blank*, to erase some colours from the initial colouring.

(b) Consider a dimension $i \in [0, d-1]$. We associate to it the pigment i and the set C_i of the corresponding b shades. For each $j \in \mathbb{Z}$, let

$$B_{i,j} = \{\boldsymbol{x} \mid x_i = j \text{ and } \forall \ell \neq i,\ x_\ell \in [0, m-1]\}.$$

We apply the blank colour to erase some of the colours from $B_{i,j}$, so that the number of points of shade $c_{i,\ell}$ is equal to the ℓth component of $\boldsymbol{z}_j$. This is always possible, because in the initial colouring, $B_{i,j}$ contains $s = \frac{m^{d-1}}{bd}$ occurrences of shade $c_{i,\ell}$, and the values of the vectors from the vector sum packing are all in $[0, s]$.

(c) We apply quasi-periodicity to reproduce this construction on the rest of the grid. Specifically, point $\boldsymbol{x} = (x_0, \ldots, x_{d-1})$ of pigment i and shade h in the initial colouring is erased if and only if the point $\boldsymbol{y}$ with $y_i = x_i$ and for all $j \neq i$, $y_j = x_j \bmod m$ was erased at step (b).

The injectivity of the vector sum packing implies the dimensional inconsistency of our grid colouring. By construction, our grid colouring is quasi-periodic, so by Lemma 16, it satisfies the separation property, concluding the proof of Lemma 14.

3.3 An Example

An illustration of the proof of Lemma 14 is given in Figure 2 on a grid of dimension $d = 2$, size $n = 8$, window size $m = 4$, number of colours $k = 3$. To create the grid-colouring we use the vector sum packing $(\boldsymbol{z}_0, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_7) = (0, 0, 0, 0, 2, 2, 2, 1)$ (note each vector has dimension 1, so is represented by its content). In particular, $s = 2$ and $b = 1$; note that $k = bd + 1$ and $s = \frac{m^{d-1}}{bd}$. Recall, the aim is that number of occurrences of 0 (*resp.* 1) in a 4 by 4 square characterizes its row (*resp.* column) number. Following the proof of Lemma 14, we start with a periodic colouring, represented in (a). We now use the blank colour 2 to erase some of the 0. First, in (b), we erase entries in the first 4 columns so the number of occurrences of 0 in these columns for the eight rows are $(0, 0, 0, 0, 2, 2, 2, 1)$. Second in (c), we apply quasi-periodicity on the rest of the grid. Next, in (d) and (e), we apply the same approach to erase some of the 1. The final result is (e). No two 4×4 subsquares contain the same multiset of colours.

Let us now illustrate how this grid is used for localization. Recall that our convention is to number the rows from top to botton, and column from left to right, both starting at 0. Assume we are measuring in a window (*i.e.* a 4×4 subsquare) the multiset of colors

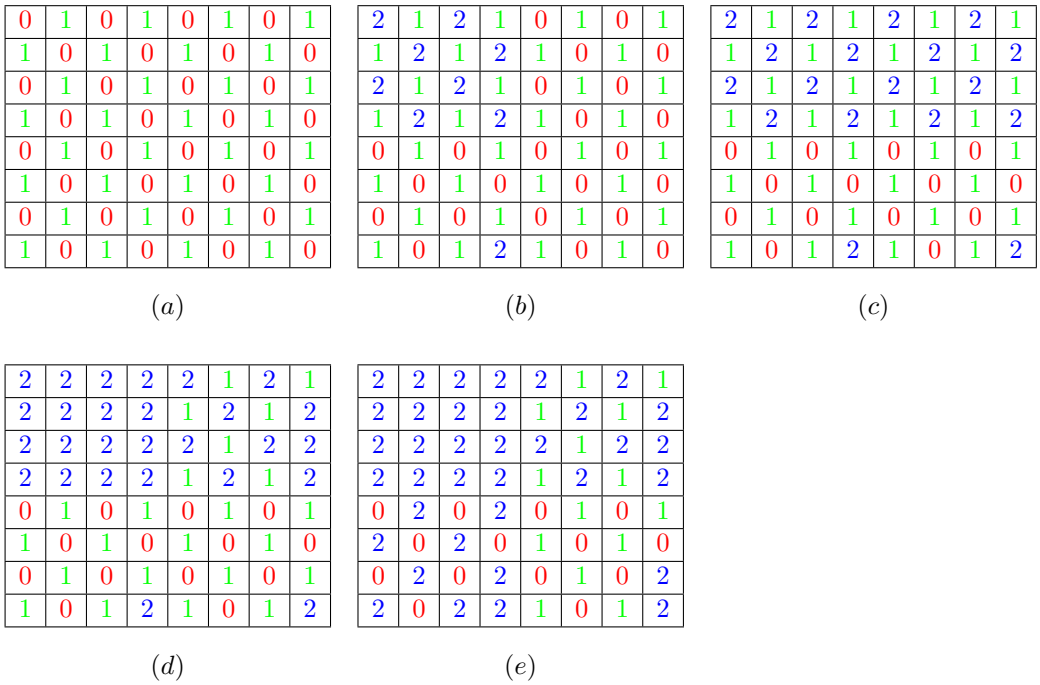


Figure 2 Illustration of the steps of the proof of Lemma 14 on a grid of dimension $d = 2$, size $n = 8$, window size $m = 4$, number of colours $k = 3$.

containing 5 occurrences of 0, 3 occurrences of 1 and 8 occurrences of 2. We wish to locate this window in the grid (e). The naive algorithm is to consider each possible window in the grid and compare the multisets of colors. This becomes costly for large grids, so we present a more efficient algorithm. By convention, color 0 is used to determine the row. Looking at the vector sum packing $(0, 0, 0, 0, 2, 2, 2, 1)$, we observe that the sequence whose jth element is the sum of $m = 4$ consecutive elements starting at position j, is $(0, 2, 4, 6, 7, 5, 3, 1)$. The number 5 is located at position 5 in this sequence, so the upper-left corner of the window we are seeking has row number 5. To determine the column, we consider the color 1. Its number of occurrence 3 has position 6 in $(0, 2, 4, 6, 7, 5, 3, 1)$, so the column number is 6. On the torus (e), the 4×4 subsquare with top left corner in row 5 and column 6 indeed contains 5 occurrences of 0, 3 occurrences of 1 and 8 occurrences of 2.

Observe that for any fixed dimension, the localization problem in the grid reduces in constant complexity to the problem of computing the position of a vector in a vector sum packing. We call this second problem *decoding*. In the next section, we will present our construction for vector sum packings, as well as a decoding algorithm with constant complexity (in the number of arithmetic operations, as the dimension d and parameter b, defined there, are fixed).

A grid colouring of size 256, window size 8, and 5 colours is presented in the appendix.

4 Vector Sum Packing

This section presents our construction of vector sum packings, thus proving Lemma 15, which is the last missing ingredient for the proof of our main result Theorem 9.

4.1 Profiles and Duals

To build vector sum packings (see Definition 11), we introduce certain integer sequences that we call *profiles*. Profiles with different parameters will be used to fill the coordinates of the vector sum packing, in Lemma 21.

The *m-dual* of a profile w is an integer sequence of same length $|w|$, defined as the sum of the elements of w on a cycling window of length m

$$\mathrm{Dual}_m(w) = \left(\sum_{j=i}^{i+m-1} w_{i \bmod |w|} \right)_{i \in [0,|w|-1]}.$$

Consider integers $s \geq 1$ and $m \geq 2$. Let us write sequences of length 1 as (σ) and sequences of length m as $(\sigma_0, \sigma_1, \ldots, \sigma_{m-1})$. For a finite sequence L and a nonnegative integer T, the sequence obtained by concatenating T copies of L one after the other is denoted by L^T. Let $\emptyset$ denote the emptysequence, and let $a \cdot b$ denote the concatenation of the sequences a and b, so $(1,2,3) \cdot (4)$ is equal to $(1,2,3,4)$. In the following tables, the rows and columns are numbered starting at 0, in red. Straight lines have been added between some of the rows to distinguish parts of the tables following different rules.

Let us define the sequence $\mathrm{Profile}(s,m,0)$ as the concatenation of the cells from the following table, read line by line from top left to bottom right.

	0	1	2	$\cdots$	$s-1$
0	$(0,\ldots,0)$	$(2,\ldots,2)$	$(4,\ldots,4)$	$\cdots$	$(2s-2,\ldots,2s-2)$
1	$(2s,\ldots,2s,2s-1)$	$(2s-2,\ldots,2s-2,2s-3)$	$(2s-4,\ldots,2s-4,2s-5)$	$\cdots$	$(2,\ldots,2,1)$

For example, we have $\mathrm{Profile}(1,3,0) = (0,0,0,2,2,1)$ and $\mathrm{Profile}(2,2,0) = (0,0,2,2,4,3,2,1)$.

▶ **Lemma 17.** $\mathrm{Profile}(s,m,0)$ *has length* $2ms$*. Furthermore, its* m*-dual is*

$$\mathrm{Dual}_m(\mathrm{Profile}(s,m,0)) = (0,2,4,\ldots,2ms-2,2ms-1,2ms-3,2ms-5,\ldots,1).$$

Proof. $\mathrm{Profile}(s,m,0)$ has length $2ms$ because each entry in the table is a sequence of cardinality m and there are $2s$ entries in the table. The reader may easily verify that the m-dual begins with non-negative even numbers increasing up to $2ms-2$ followed by positive odd numbers decreasing down from $2ms-2$. Thus, the m-dual contains every integer in $[0, 2m-1]$ exactly once. ◀

Let us define the $(s,m,0)$-decoding function as the function that associates to an integer v its smallest index in the m-dual of $\mathrm{Profile}(s,m,0)$. For example, the $(2,2,0)$-decoding function sends 6 to 3, because the 2-dual of $\mathrm{Profile}(2,2,0) = (0,0,2,2,4,3,2,1)$ is $(0,2,4,6,7,5,3,1)$, where the first (and only) occurrence of 6 is at position 3.

▶ **Corollary 18.** *The* $(s,m,0)$*-decoding function is*

$$v \mapsto \begin{cases} \frac{v}{2} & \text{if } v \text{ is even,} \\ 2ms - 1 - \frac{v-1}{2} & \text{if } v \text{ is odd.} \end{cases}$$

It is computable in a constant number of arithmetic operations.

For any positive integer T and $m \geq 2$, let us define the sequence Profile(s, m, T) as the concatenation of the cells from the following table.

	0	1	2	$\cdots$	$s-1$
0	$(0,\dots,0,0)^T$	$(0,\dots,0,2)^T$	$(0,\dots,0,4)^T$	$\cdots$	$(0,\dots,0,2s-2)^T$
1	$(0,\dots,0,0,2s)^T$	$(0,\dots,0,2,2s)^T$	$(0,\dots,0,4,2s)^T$	$\cdots$	$(0,\dots,0,2s-2,2s)^T$
$\vdots$	$\vdots$	$\vdots$			$\vdots$
$m-1$	$(0,2s,\dots,2s,2s)^T$	$(2,2s,\dots,2s,2s)^T$	$\cdots$	$\cdots$	$(2s-2,2s,\dots,2s)^T$
m	$(2s)^{mT-1}$	$(2s-1,2s,\dots,2s)^T$	$(2s-3,2s,\dots,2s)^T$	$\cdots$	$(3,2s,\dots,2s)^T$
$m+1$	$(1,2s,2s,\dots,2s)^{T-1}$	$(1,2s,\dots,2s,2s-2)^T$	$(1,2s,\dots,2s,2s-4)^T$	$\cdots$	$(1,2s,\dots,2s,2)^T$
$m+2$	$(1,2s,\dots,2s,0)^T$	$(1,2s,\dots,2s-2,0)^T$	$(1,2s,\dots,2s,2s-4,0)^T$	$\cdots$	$(1,2s,\dots,2s,2,0)^T$
$\vdots$	$\vdots$	$\vdots$			$\vdots$
$2m-1$	$(1,2s,0,\dots,0)^T$	$(1,2s-2,0,\dots,0)^T$	$\cdots$	$\cdots$	$(1,2,0,\dots,0)^T$
$2m$	$(1,0,\dots,0)^{T-1}\cdot(1)$	$\emptyset$	$\emptyset$	$\cdots$	$\emptyset$

▶ **Lemma 19.** *For any $m \geq 2$,* Profile(s, m, T) *has length $m((2ms+1)T-2)$. Furthermore, its m-dual is obtained by concatenation of the cells of the following table*

	0	1	2	$\cdots$	$s-1$
0	$(0)^{mT}$	$(2)^{mT}$	$(4)^{mT}$	$\cdots$	$(2s-2)^{mT}$
1	$(2s)^{mT-1}$	$(2s+2)^{mT}$	$(2s+4)^{mT}$	$\cdots$	$(4s-2)^{mT}$
2	$(4s)^{mT-1}$	$(4s+2)^{mT}$	$(4s+4)^{mT}$	$\cdots$	$(6s-2)^{mT}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
$m-1$	$(2(m-1)s)^{mT-1}$	$(2(m-1)s+2)^{mT}$	$(2(m-1)s+4)^{mT}$	$\cdots$	$(2ms-2)^{mT}$
m	$(2ms)^{mT-1}$	$(2ms-1)^{mT}$	$(2ms-3)^{mT}$	$\cdots$	$(2(m-1)s+3)^{mT}$
$m+1$	$(2(m-1)s+1)^{mT-1}$	$(2(m-1)s-1)^{mT}$	$(2(m-1)s-3)^{mT}$	$\cdots$	$(2(m-2)s+3)^{mT}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
$2m-1$	$(2s+1)^{mT-1}$	$(2s-1)^{mT}$	$(2s-3)^{mT}$	$\cdots$	$(3)^{mT}$
$2m$	$(1)^{mT-1}$	$\emptyset$	$\emptyset$	$\cdots$	$\emptyset$

Proof. In the table, there are $2ms-2$ cells containing sequences of length mT, one cell containing a sequence of length $mT-1$, one cell containing a sequence of length $m(T-1)$ and one celle containing a sequence of length $m(T-1)+1$, so Profile(s, m, T) has length

$$(2ms-2)mT + mT - 1 + m(T-1) + m(T-1) + 1 = mT(2ms+1) - 2m$$

as desired. Again, the reader may verify that the m-dual begins with non-negative even numbers increasing up to $2ms$ followed by positive odd numbers decreasing down from $2ms-1$ (repeated in the quantities specified). ◀

We define the (s, m, T)-decoding function as the function that associates to an integer v its smallest index in the m-dual of Profile(s, m, T).

▶ **Corollary 20.** *The (s, m, T)-decoding function output on the input v is computed using the following algorithm. If v is even, we define $r = \lfloor \frac{v}{2s} \rfloor$ and $c = \frac{v}{2} \bmod [s]$. They represent the row and column in the m-dual from Lemma 19. Then the output of the decoding function is*

$$(r\,m\,T\,s - r + 1)\mathbb{1}_{r>0} + (c\,m\,T - 1 + \mathbb{1}_{r=0})\mathbb{1}_{c>0}.$$

Otherwise, v is odd and we define $r = \lfloor \frac{2\,m\,s-v+1}{2s} \rfloor$ and $c = \frac{2\,m\,s-v+1}{2} \bmod s$. The output is then

$$1 + m(m\,T\,s - 1) + (m\,T\,s\,r - r)\mathbb{1}_{r>0} + (c\,m\,T\ - 1)\mathbb{1}_{c>0}.$$

It is computable in a constant number of arithmetic operations.

4.2 Generating a Vector Sum Packing

We will explicitly construct the vector sum packing by combining profiles, thus proving Lemma 15.

Recall that an (n, m, b, s)-vector sum packing is characterised by its pattern $(\boldsymbol{z}_0, \ldots, \boldsymbol{z}_{n-1})$ where each $\boldsymbol{z}_i$ is a vector in $[0, s]^b$. These vectors will be defined as the columns of a matrix $M^{(b)}$. Furthermore, the rows of this matrix will be constructed using sequences given by the Profile(s, m, T).

The matrix $M^{(b)}$ will have b rows and $m \cdot T_b = n_b$ columns. Specifically, given $s \geq 1$ and $m \geq 2$, we set $T_0 = 0$ and $T_1 = 2s$. Then, for each $b \geq 1$, we recursively set

$$T_{b+1} = (2ms + 1)T_b - 2.$$

Thus we obtain the dimensions of our $M^{(b)}$ matrices. To fill in the entries of the matrices we again apply a recursive construction.

- For $b = 1$, the matrix $M^{(1)}$ has only one row, which is identical to the sequence Profile$(s, m, 0)$. We remark that $M^{(1)}$ does indeed have $n_1 = m \cdot T_1 = 2ms$ columns, as required by Lemma 17.
- Next consider the case $b \geq 2$. The basic idea is that $M^{(b)}$ should simply be the concatenation on $2ms + 1$ copies of $M^{(b-1)}$, plus an additional row identical to the sequence Profile(s, m, T_{b-1}), which will be used to distinguish between the different copies.

However, this basic idea does not scale correctly, so instead of concatenating identical copies of $M^{(b-1)}$, we also concatenate truncated copies of $M^{(b-1)}$. Specifically, we allow for the truncated matrix $M^{(b-1,\star)}$ which is identical to $M^{(b-1)}$ except that its first column is removed.

We now set $M^{(b)} = M^{(b-1,0)} \circ M^{(b-1,2)} \circ \cdots \circ M^{(b-1,2ms)} \circ M^{(b-1,2ms-1)} \circ \cdots \circ M^{(b-1,1)})$, where each $M^{(b-1,\ell)}$ is either $M^{(b-1)}$ or $M^{(b-1,\star)}$ and where $\circ$ denotes the concatenation operation. Thus, it remains, to prescribe, for each for $\ell \in [0, 2ms]$ whether $M^{(b-1,\ell)}$ is set equal to $M^{(b-1)}$ or $M^{(b-1,\star)}$. To do this, we use the m-dual of the Profile(s, m, T_{b-1}). In particular, define I_ℓ to be the set of indices where the m-dual of the profile takes value ℓ. That is,

$$I_\ell = \{i \mid \text{Dual}_m(\text{Profile}(s, m, T_{b-1}))_i = \ell\} \tag{1}$$

Recall that $n_b = mT_b = m \cdot ((2ms + 1) \cdot T_{b-1} - 2)$. Observe then, from Lemma 19, that the (I_ℓ)s are disjoint integer intervals of length either n_{b-1} or $n_{b-1} - 1$ whose union is $[0, n_b - 1]$. For each $\ell \in [0, 2ms]$, we now define

$$M^{(b-1,\ell)} = \begin{cases} M^{(b-1)} & \text{if } |I_\ell| = n_{b-1}, \\ M^{(b-1,\star)} & \text{if } |I_\ell| = n_{b-1} - 1. \end{cases}$$

The resultant construction of $M^{(b)}$ is then illustrated in Figure 3. For example, for $s = 1$, $m = 2$ and $b = 2$, we have $T_0 = 0$ and $T_1 = 2$, so

$$\begin{aligned} \text{Profile}(s, m, T_{b-1}) &= \text{Profile}(1, 2, 2) = (0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 1, 2, 1, 0, 1), \\ \text{Dual}_m(\text{Profile}(s, m, T_{b-1})) &= (0, 0, 0, 0, 2, 2, 2, 4, 4, 4, 3, 3, 3, 1, 1, 1), \\ \text{Profile}(s, m, T_{b-2}) &= \text{Profile}(1, 2, 0) = (0, 0, 2, 1), \end{aligned}$$

so

$$M^{(2)} = \begin{pmatrix} 0 & 0 & 2 & 1 & 0 & 2 & 1 & 0 & 2 & 1 & 0 & 2 & 1 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 2 & 2 & 2 & 2 & 1 & 2 & 1 & 0 & 1 \end{pmatrix}.$$

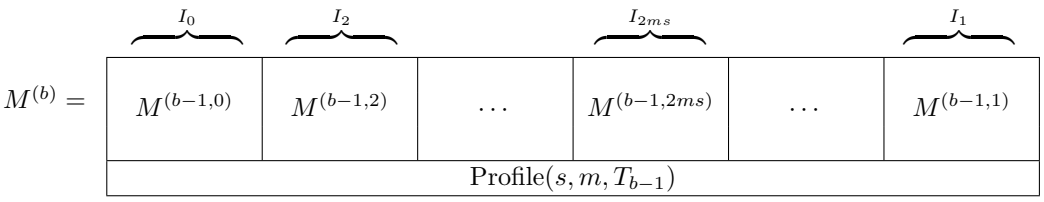


Figure 3 The recursive construction of the matrix $M^{(b)}$. The indices j for the sets I_j on top of the figure are first increasing even numbers, then decreasing odd numbers.

As stated, we will take our vectors $(\boldsymbol{z}_0, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n_b-1})$ to be the columns of $M^{(b)}$. That is, let $V_{i,j} = M^{(b)}_{i \bmod n_b, j}$. Then, for any $i \in \mathbb{Z}$, we have $\boldsymbol{z}_i = (V_{i,j})_{j \in [0,b-1]}$.

▶ **Lemma 21.** *The sequence of vectors $(\boldsymbol{z}_0, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n_b-1})$ is the pattern of an $(n_b, m, b, 2s)$-vector sum packing.*

Proof. Given $(\boldsymbol{z}_0, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n_b-1})$ recall that $f(\boldsymbol{z}_j, \boldsymbol{z}_{j+1}, \ldots, \boldsymbol{z}_{j+m-1}) = \boldsymbol{z}_j + \boldsymbol{z}_{j+1} + \cdots + \boldsymbol{z}_{j+m-1}$, where the indices are taken modulo n_b. Our task is to prove that f is injective.

We will show this by extending the definition of an m-dual to matrices: we define the m-dual of a matrix M with n columns to be the matrix D of the same dimensions where for all i, the ith column of D is equal to the sum of the columns of M of all indices in $[i, i+m-1]$ modulo n.

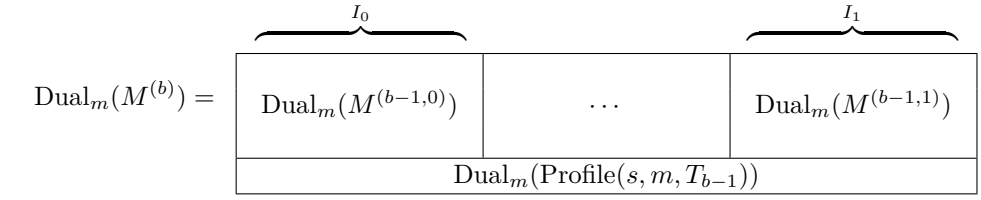
Proving $(\boldsymbol{z}_0, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n_b-1})$ is an $(n_b, m, b, 2s)$-vector sum packing is then equivalent to proving that the m-dual of $M^{(b)}$ does not contain any identical columns. We will do so by induction on b.

Initialization. For the base case $b = 1$, recall that $M^{(b)} = \text{Profile}(s, m, 0)$. Hence, by Lemma 17, $\text{Dual}_m(M^{(b)})$ does not contain two identical columns.

Induction Step. For the induction hypothesis, assume that $\text{Dual}_m(M^{(b-1)})$ does not contain two identical columns. Now define $\text{Profile}^*(s, m, T)$ to be equal to $\text{Profile}(s, m, T)$ except with the first 0 removed. By construction, each row $j \in [0, b-1]$ of $M^{(b)}$ is a concatenation of the form $P_0 \circ P_1 \circ \cdots \circ P_r$ where each P_i is equal either to $\text{Profile}(s, m, T_j)$ or to $\text{Profile}^*(s, m, T_j)$. Next observe that both $\text{Profile}(s, m, T_j)$ and $\text{Profile}^*(s, m, T_j)$ start with $m-1$ occurrences of 0. This implies that m-dual and concatenation commutes as

$$\text{Dual}_m(P_0 \circ P_1 \circ \cdots \circ P_r) = \text{Dual}_m(P_0) \circ \text{Dual}_m(P_1) \circ \cdots \circ \text{Dual}_m(P_r).$$

Therefore the m-dual of the matrix $M^{(b)}$ is



(2)

Assume that two columns c and c' with column indices i and i' in $\text{Dual}_m(M^{(b)})$ are equal. Let $\ell \in [0, 2ms]$ denote the value of c in its final row $b-1$. Thus, c' also has entry ℓ in its last row. But the last row $b-1$ of $\text{Dual}_m(M^{(b)})$ is defined to equal $\text{Dual}_m(\text{Profile}(s, m, T_{b-1}))$. Hence, by definition of I_ℓ from Equation (1), we have $i \in I_\ell$ and $i' \in I_\ell$. Let d and d' be

obtained from c and c' by removing their last row $b-1$. By Equation (2), both d and d' belong to $\text{Dual}_m(M^{(b-1,\ell)})$. Thus, they both belong to $\text{Dual}_m(M^{(b-1)})$. By the induction hypothesis, this implies $i = i'$ and concludes the proof. ◀

Let the (n, m, b, s)-decoding function for vector sum packing be defined as the function that inputs an integer vector $\boldsymbol{x}$ and outputs its index in the m-dual of the (n, m, b, s)-vector sum packing we defined.

▶ **Corollary 22.** *For fixed b, the $(n_b, m, b, 2s)$-decoding function for our vector sum packing is computable in a constant number of arithmetic operations.*

Proof. This decoding function is computed recursively, following the recursive construction of matrix $\text{Dual}_m(M^{(b)})$ from Equation (2). It inputs a vector $\boldsymbol{x}$ and an auxiliary Boolean parameter B, initialized at *False*, that indicates if we are looking for localization in a vector sum packing where the first vector has been removed.

We first use the decoding function for profiles from Corollaries 18 and 20 on the last coordinate of $\boldsymbol{x}$ to compute an integer c, and set $p = c$ if B is equal to *False*, and $p = c - 1$ if B is equal to *True*. For $b = 1$, as $\boldsymbol{x}$ is a vector of dimension 1, the algorithm stops and p is returned. Otherwise, by construction, p is the smallest possible index for any vector whose last coordinate is equal to the last coordinate of $\boldsymbol{x}$. Now, in Equation (2), we want to determine whether the matrix $\text{Dual}_m(M^{b-1,j})$ on top of p is equal to $\text{Dual}_m(M^{b-1})$ or to $\text{Dual}_m(M^{b-1,\star})$. As explained in the construction, this is decided by looking at $\text{Dual}_m(\text{Profile}(s, m, T_{b-1}))$ from Lemma 19. In this sequence, let ℓ denote the number of repetition of the element at position p.

- If $\ell = mT_{b-1}$, then we are working with $\text{Dual}_m(M^{b-1})$. In that case, we call recursively the $(n_{b-1}, m, b-1, 2s)$-decoding function on the vector $\boldsymbol{x}$ without its last coordinate, with auxiliary parameter equal to *False*. The output is added to p and returned.
- Otherwise, we have $\ell = mT_{b-1} - 1$, and we are working with $\text{Dual}_m(M^{b-1,\star})$. We call recursively the $(n_{b-1}, m, b-1, 2s)$-decoding function on the vector $\boldsymbol{x}$ without its last coordinate, with auxiliary parameter equal to *True*. The output is added to p and returned.

Deciding whether $\ell = mT_{b-1}$ or $\ell = mT_{b-1} - 1$ is achieved by looking at the parity of p and its value modulo s. This recursive construction has depth b, so for b fixed, it requires only a constant number of arithmetic operations. Python code for this algorithm is provided in [13]. ◀

4.3 Summary and Complexity

Let us summarize our algorithm constructing a grid coloring. It inputs a dimension $d \geq 2$ and two other parameters $b \geq 1$ and $t \geq 1$. The first step is to compute the window size $m = 2bdt$, the number of colors $k = bd + 1$, the parameter $s = (2bd)^{d-2}t^{d-1}$ to ensure $2s = \frac{m^{d-1}}{bd}$, and the grid size $n = (2ms + 1)^{b-1}(2ms - 1/s) + 1/s$. The second step is to construct an $(n, m, b, 2s)$-vector sum packing, as described in Section 4.2, using the *profile* sequences defined in Section 4.1. In the third step, we finally colour the points of our grid of side n and dimension d. Our set of colours is $\{0, 1, \ldots, k-1\}$. Here the last colour $k-1$ is a "*blank*" color used to erase the other colours. The other colours are divided into d sets, called pigment classes. Each pigment class contains b colours, which we call its *shades*. To each dimension of the grid is associated a unique pigment. The vector sum packing is then used, as explained in Section 3, to colour the points of the grid.

Next consider the localization problem. Localization means, given a multiset S of colours, the recovery of the unique window of size m in the grid that contains this multiset of colors (if it exists). To achieve localization, we proceed dimension by dimension. We count in S the colours from the pigment class corresponding to the dimension considered and make a vector out of it. For example, if the dimension corresponds to the colours $4, 5, 6$ and S contains three occurrences of the color 4, zero occurrences of the colour 5 and two occurrences of the colour 6, the vector is $(3, 0, 2)$. We use the decoding algorithm for vector sum packing from Corollary 22 to translate this vector into a coordinate. Having achieved this for every coordinate, we deduce the position of the window whose multiset of colours is equal to S.

The construction of the grid and localization procedure are illustrated in Section 3.3. We measure complexity as the number of arithmetic operations for b and d fixed, while t goes to infinity. The construction of the grid has complexity proportional to the size of its output, which is $\mathcal{O}(n^d) = \mathcal{O}(t^{b\,d^2})$. Corollary 22 implies that the complexity of the localization algorithm is constant.

5 Conclusion

Many interesting directions for future research remain, both theoretical and practical. One nice extension would be to make our grid colouring robust by allowing for error detection and correction. This has been achieved for other cycle and torus packing problems (see, for example, [2, 4, 6, 25]) and is a necessary step for practical applications. Another valuable contribution would be to study disc-like windows rather than the square windows examined in this article. This would match the natural shape of the domain where an emission emitted at a point is detectable.

References

1 E. Aboufadel, T. Armstrong, and E. Smietana. Position coding. *arXiv preprint*, 2007. arXiv:0706.0869.

2 R. Berkowitz and S. Kopparty. Robust positioning patterns. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1937–1951, 2016.

3 A. Blanca and A. Godbole. On universal cycles for new classes of combinatorial structures. *SIAM Journal on Discrete Mathematics*, 25(4):1832–1842, 2011. Publisher: SIAM.

4 A. Bruckstein, T. Etzion, R. Giryes, N. Gordon, R. Holt, and D. Shuldiner. Simple and robust binary self-location patterns. *IEEE Transactions on Information Theory*, 58(7):4884–4889, 2012.

5 J. Burns and C. Mitchell. *Coding schemes for two-dimensional position sensing.* Hewlett-Packard Laboratories, Technical Publications Department, 1992.

6 Y. Chee, D. Dao, H. Kiah, S. Ling, and H. Wei. Binary robust positioning patterns with low redundancy and efficient locating algorithms. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2171–2184, 2019.

7 C. S. Chen, P. Baracca, E. de Panafieu, and D. Michalopoulos. A positioning method exploiting ambient IoT deployments in controlled environments, Nokia Patent Application, 2023.

8 Chung Shue Chen, Peter Keevash, Sean Kennedy, Élie de Panafieu, and Adrian Vetta. Robot positioning using torus packing for multisets, 2024. arXiv:2404.09981.

9 F. Chung, P. Diaconis, and R. Graham. Universal cycles for combinatorial structures. *Discrete Mathematics*, 110(1-3):43–59, 1992. Publisher: North-Holland.

10 D. Curtis, T. Hines, G. Hurlbert, and T. Moyer. Near-universal cycles for subsets exist. *SIAM Journal on Discrete Mathematics*, 23(3):1441–1449, 2009.

11 M. Dębski and Z. Lonc. Universal cycle packings and coverings for k-subsets of an n-set. *Graphs and Combinatorics*, 32:2323–2337, 2016. Publisher: Springer.

12 N. de Bruijn. Acknowledgement of priority to C. Flye Sainte-Marie on the counting of circular arrangements of 2^n zeros and ones that show each n-letter word exactly once. *EUT report. WSK, Dept. of Mathematics and Computing Science*, 75-WSK-06, 1975.

13 É. de Panafieu. Torus packing for multisets. Software, version 1.0., swhId: `swh:1:dir:2eb591ca09fb8739e2135e17b5f595b7f25ed924` (visited on 2024-06-17). URL: `https://gitlab.com/depanafieuelie/torus-packing-for-multisets`.

14 S. Glock, F. Joos, D. Kühn, and D. Osthus. Euler tours in hypergraphs. *arXiv preprint*, 2020. `arXiv:1808.07720`.

15 N. Hartsfield and G. Ringel. *Pearls in graph theory: A comprehensive introduction*. Courier Corporation, 2013.

16 S.-W. Ho and C. S. Chen. Visible light communication based positioning using color sensor. In *IEEE 8th Optoelectronics Global Conference (OGC)*, pages 1–5, 2023.

17 G. Hurlbert and G. Isaak. On the de Bruijn Torus problem. *Journal of Combinatorial Theory, Series A*, 64(1):50–62, 1993. Publisher: Elsevier.

18 G. Hurlbert and G. Isaak. New constructions for de Bruijn tori. *Designs, Codes and Cryptography*, 6(1):47–56, 1995.

19 G. Hurlbert, T. Johnson, and J. Zahl. On universal cycles for multisets. *Discrete Mathematics*, 309(17):5321–5327, 2009. `doi:10.1016/j.disc.2008.04.050`.

20 G. Hurlbert, C. Mitchell, and K. Paterson. On the existence of de Bruijn tori with two by two windows. *Journal of Combinatorial Theory, Series A*, 76(2):213–230, 1996. Publisher: Elsevier.

21 D. Knuth. *Art of Computer Programming, Combinatorial Algorithms*, volume 4A. Addison-Wesley Professional, 2011.

22 H. Li, C. Zhou, S. Wang, Y. Lu, and X. Xiang. Two-dimensional gold matrix method for encoding two-dimensional optical arbitrary positions. *Optics Express*, 26(10):12742–12754, 2018.

23 J. Luo, L. Fan, and H. Li. Indoor positioning systems based on visible light communication: State of the art. *IEEE Commun. Surveys Tuts.*, 19(4):2871–2893, 2017.

24 J. MacWilliams and N. Sloane. Pseudo-random sequences and arrays. *Proceedings of the IEEE*, 64(12):1715–1729, 1976.

25 D. Makarov and A. Yashunsky. On a construction of easily decodable sub-de Bruijn arrays. *Journal of Applied and Industrial Mathematics*, 13:280–289, 2019. Publisher: Springer.

26 M. Rahman, T. Li, and Y. Wang. Recent advances in indoor localization via visible lights: A survey. *Sensors*, 20(5), 2020.

27 D. Schüsselbauer, A. Schmid, and R. Wimmer. Dothraki: Tracking tangibles atop tabletops through de-Bruijn tori. In *Proceedings of the 15th International Conference on Tangible, Embedded, and Embodied Interaction*, pages 1–10, 2021.

28 F. Sinden. Sliding window codes. *AT&T Bell Labs Technical Memorandum*, 1985.

29 I. Szentandrasi, M. Zachariáš, J. Havel, A. Herout, M. Dubska, and R. Kajan. Uniform marker fields: Camera localization by orientable de Bruijn tori. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 319–320, 2012.

30 Z. Yang, Z. Wang, J. Zhang, C. Huang, and Q. Zhang. Wearables can afford: Light-weight indoor positioning with visible light. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 317–330, 2015.

31 F. Zafari, A. Gkelias, and K. Leung. A survey of indoor localization systems and technologies. *IEEE Commun. Surveys Tuts.*, 2019.

A Software Implementation

We implemented our algorithm constructing the grid colouring as well as the decoding algorithm in Python. The code is available at [13]. An example of grid colouring computed with this code is presented in Figure 4.

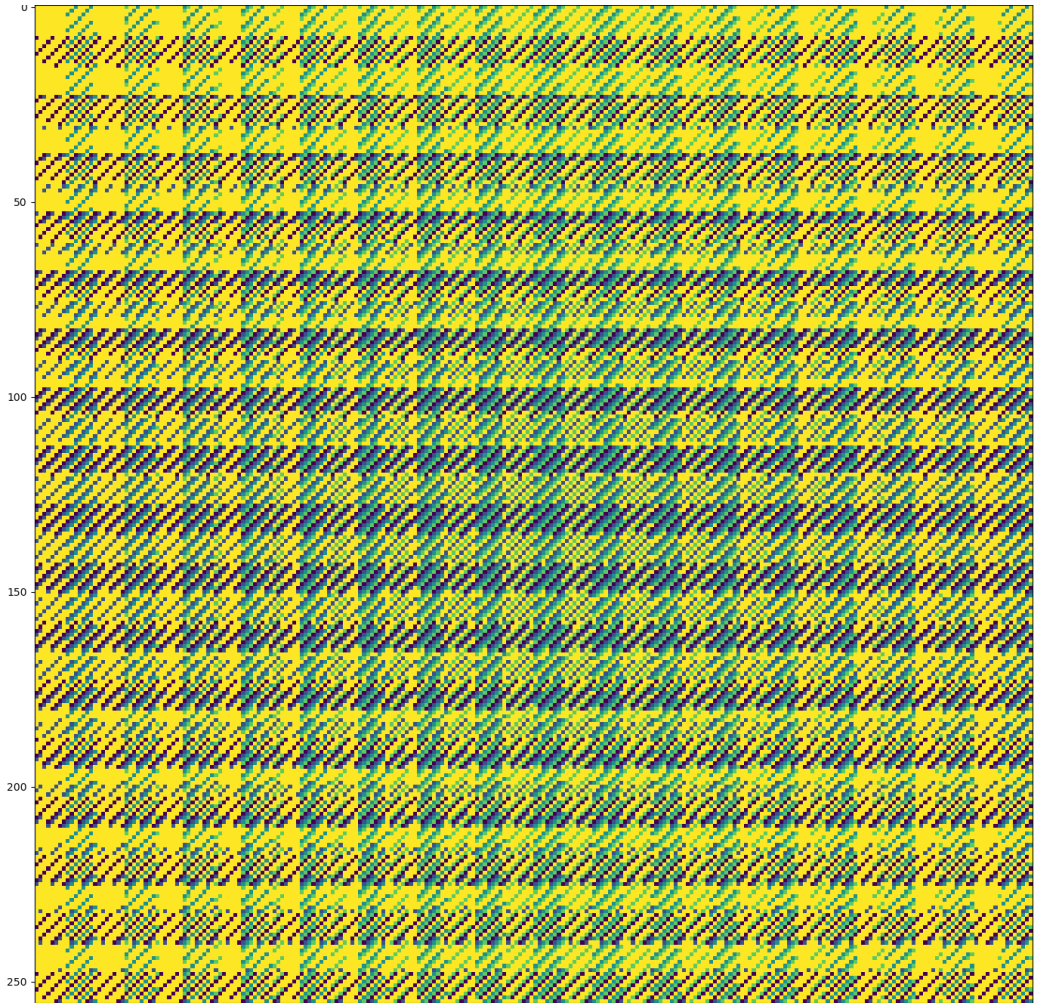


Figure 4 Grid colouring of size 256, window size 8 and number of colours 5. It corresponds to the parameters $d = 2$, $b = 2$ and $t = 1$.

Bayesian Calibrated Click-Through Auctions

Junjie Chen
City University of Hong Kong, Hong Kong, China

Minming Li
City University of Hong Kong, Hong Kong, China

Haifeng Xu
University of Chicago, IL, USA

Song Zuo
Google Research, New York, NY, USA

Abstract

We study information design in *click-through auctions*, in which the bidders/advertisers bid for winning an opportunity to show their ads but only pay for realized clicks. The payment may or may not happen, and its probability is called the *click-through rate* (CTR). This auction format is widely used in the industry of online advertising. Bidders have private values, whereas the seller has private information about each bidder's CTRs. We are interested in the seller's problem of partially revealing CTR information to maximize revenue. Information design in click-through auctions turns out to be intriguingly different from almost all previous studies in this space since any revealed information about CTRs will never affect bidders' bidding behaviors – they will always bid their true value per click – but only affect the auction's *allocation* and *payment* rule. In some sense, this makes information design effectively a constrained mechanism design problem.

Our first result is an FPTAS to compute an approximately optimal mechanism under a constant number of bidders. The design of this algorithm leverages Bayesian bidder values which help to "smooth" the seller's revenue function and lead to better tractability. The design of this FPTAS is complex and primarily algorithmic. Our second main result pursues the design of "simple" mechanisms that are approximately optimal yet more practical. We primarily focus on the two-bidder situation, which is already notoriously challenging as demonstrated in recent works. When bidders' CTR distribution is symmetric, we develop a simple *prior-free* signaling scheme, whose construction relies on a parameter termed *optimal signal ratio*. The constructed scheme provably obtains a good approximation as long as the maximum and minimum of bidders' value density functions do not differ much.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory and mechanism design

Keywords and phrases information design, ad auctions, online advertising, mechanism design

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.44

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version (with missing Proofs and Appendices)*: `https://arxiv.org/abs/2306.06554`

Funding *Haifeng Xu*: Supported by NSF Award CCF-2303372, Army Research Office Award W911NF-23-1-0030, Office of Naval Research Award N00014-23-1-2802 and the AI2050 program at Schmidt Sciences (Grant G-24-66104).

1 Introduction

Many of the phenomenal Internet Technology companies are powered by online advertising [25]. When an Internet user browses a webpage, an ad auction may be run to determine which ads to be displayed to this user. Such ad auctions can be either done by the webpage owner

or through ad exchange platforms. Large website owners (sellers) sometimes may know the users much better than individual advertisers (bidders) do. It is thus natural for the seller to utilize such information advantage to improve its ad revenue. As an example, the seller's knowledge about the user can be used to better predict an Internet user's *click-through rate* (CTR), which is then used in ad auctions to deliver ad impressions more efficiently.

Interestingly, a recent line of works, starting from Bro Miltersen and Sheffet [14], Emek et al. [26], comes to realize that it may not always be the seller's best interest to use as much information as possible. That is, partially use certain amount – and the right type – of information may be more beneficial for the seller's revenue. This insight has motivated many studies on the algorithmic problem of optimal *information design* – i.e., determining what information to be shared with which bidders – in order to maximize revenue in different auction formats [14, 26, 12, 38, 30, 5].

While there are two main types of bidding strategies in online advertising: autobidding [20, 7, 6] and manual bidding, to better understand the optimal design of information in an auction environment, we follow the rich literature and turn to the cleaner manual bidding where the platform incentivizes buyers to tell the truth. However, different from almost all previous works focusing on stylized auction formats such as the second price auction for independent-value bidders [14, 26, 12, 5] and common-value bidders [38] and Myerson's optimal auction [30], we focus on a different auction format, i.e., the *click-through auction*, which is the main auction format employed by the current online advertising industry [27]. The click-through auction is also widely known as the generalized second-price auction [25] or position auctions [44]. We use the term "click-through auction" as coined by Bergemann et al. [11] to emphasize the sale of *clicks*, since this factor turns out to make information design in click-through auctions significantly different from that in almost all other previously studied auctions. In a click-through auction for selling a single ad position, each bidder i submits a bid b_i expressing his value for each *click* of his ad. Additionally, the seller will estimate a *click through rate* (CTR) r_i for bidder i. The auction runs by ranking bidders according to the *product score* $b_i r_i$, denoted as $b_{(1)}r_{(1)} \geq b_{(2)}r_{(2)} \geq \cdots$, and allocates the item to the bidder with the highest product score $b_{(1)}r_{(1)}$. Crucially, since the auction only sells clicks, the winner does not need to pay *unless a click truly happens* in which case the winner pays the second highest score divided by his own CTR, i.e., $b_{(2)}r_{(2)}/r_{(1)}$. When there is only a single ad slot to allocate, it is straightforward to verify that this auction is truthful, *regardless of the CTR values (even mistakenly estimated)*.[1] As a result, the seller's information advantage of knowing the CTRs *cannot* be exploited to influence the advertisers' bidding behaviors since it is always their best interest to bid the true value per click v_i. In some sense, information design here is effectively a naturally restricted format of mechanism design for multiple items with additive bidder values (which is generally a very difficult question [19]). This crucially differs from the information design task in all previously studied auction formats, in which the seller's information about the item (e.g., the CTR) can be revealed to alter bidders' bidding behaviors. This is intrinsically because bidders pay for realized clicks only thus the probability of receiving a click will not matter in click-through auctions. Note that, even in per-impression ad auctions as studied in [5], bidders pay for impressions but care about clicks or conversions, thus their values do depend on the probability of receiving a click, i.e., the CTR.

[1] Such truthfulness holds only when there is a single slot, which is the case we focus on in this paper. Click-through auctions are well-known to be non-truthful and difficult to analyze when there are multiple slots [25].

Most relevant to ours is the recent work by Bergemann et al. [11], who study the same information design question of optimally revealing the CTR information in click-through auctions. They focused on a simplified setup with *complete information* about the bidders, i.e., the bidders' values are assumed to be fully known to the seller. A natural *calibration* constraint, originating from Foster and Vohra [29], is imposed on their information design, which simply means the disclosed CTR estimation to each bidder has to be consistent with (i.e., equal in expectation) the true CTR of the bidder. They thus call the new model *calibrated click-through auctions*. Notably, the seller is allowed to privately communicate information with each bidder, which is known as private signaling. The main contribution of this paper is to extend the setup of [11] to the more natural and also more widely studied setup of *Bayesian* bidders with independent values. For this reason, We call our model the *Bayesian calibrated click-through auction*.

To our knowledge, Bergemann et al. [11] is the first study of information design in auctions in which the revealed information *cannot* influence bidders' behaviors, but only affect the mechanism itself. This gives rise to an intriguing information design problem – in some sense, it is even in contrast to one's first impression about information design, also known as *persuasion* [33], which seeks to exploit information advantage to influence others' behaviors. In contrast, information design in click-through auctions only affects the final allocation and payment of the mechanism but has no effect on bidders' bidding behaviors. From this perspective, information design here is effectively a form of mechanism design. Indeed, a similar phenomenon was observed by Daskalakis et al. [19], who study the *co-design* of information structure and the auction mechanism. They observe that at the optimal co-design, there is no signaling to bidders whatsoever, and the seller will only use the underlying state to decide the item allocation and payment. Our problem is similar, except that we restrict ourselves to the class of click-through auctions. This restriction is motivated by its practical applications.

Before proceeding to describe our results, we briefly highlight the challenges of information design in our problem. Indeed, the difficulty of information design in strategic games is well documented in previous works [22, 13]. Even just for auctions, Emek et al. [26] shows that computing an optimal information design in a second price auction is NP-hard in general for Bayesian bidder values, though it does become polynomial-time solvable in cases with complete information of bidder values. Unfortunately, such tractability does not transfer to the click-through auctions: in a general environment with complete information about bidder values, even the problem of optimal information design in *two*-bidder click-through auctions is left as an open question in [11]. Bergemann et al. [11] show that when the distributions of CTRs are symmetric, an optimal signaling policy can be characterized. However, back to our Bayesian generalization of their setup, their optimal design for the symmetric information environment is no longer applicable because the design of their signaling scheme relies crucially on knowing the exact identity of the winner for any signal realization, which unfortunately becomes uncertain in our Bayesian setup with random bidder values. This barrier brings challenges, but also brings opportunities to adopt more algorithmic approaches. Next, we elaborate on our findings.

1.1 Results

We study the theoretical aspects of Bayesian calibrated click-through auctions and develop two encouraging positive results.

Our first result, Theorem 3.1, exhibits a Fully Polynomial Time Approximation Scheme (FPTAS) for computing an approximately revenue-optimal signaling scheme in an *arbitrary* information environment, assuming no point mass in bidders' value distributions. This

FPTAS applies to any constant number of bidders. Interestingly, our FPTAS bypasses the notorious challenge of the complete-information general setup of [11] by relaxing it to the more natural Bayesian valuation setup with smooth value distributions. We use signals of the form $k\epsilon$ for some integer k. The key challenge is to enforce the calibration constraint – i.e., the posterior mean of the CTR conditioned on the signal has to equal the signal itself – in the rounding process. We develop a nontrivial technique to address the challenge. Notably, this FPTAS applies to any auction mechanism, as long as its revenue as a function of signals is Lipschitz continuous.

Our second main result, Theorem 4.2, examines the popular recent computational model of unknown value distributions, also known as *prior-free* setups [21]. We give explicit and efficient construction of simple *prior-free* signaling schemes for *symmetric* information environments (as studied also by Bergemann et al. [11]) with strong approximation guarantees. The constructed signaling scheme is based on a parameter coined the *optimal signal ratio*, which we find is at most 1. This allows us to send larger signals (i.e., larger CTR estimations) to the bidder with a high click-through rate than the bidder with a low click-through rate. The proof mainly consists of (i) establishing a connection between the optimal signaling scheme under the unknown value distribution and the optimal signaling scheme under a uniform value distribution and (ii) providing the approximation ratio of the constructed signaling scheme under a uniform value distribution, which is proved to be 0.995. En route to proving (ii), we prove a more general result (Proposition 4.11) which shows that, if the optimal signal ratio is convex in CTR (which is true for various commonly used distributions), then a signaling scheme with better approximation can be devised. This result may be of independent interest.

1.2 Additional Related Works

Due to the space limit, we briefly discuss the related works here, while additional discussions in detail are in Appendix A. The most relevant literature to our work is information design in auctions. Information design has been adopted to second price auctions [14, 26, 18, 5] and Myerson auctions [30] in various setups. Recently, Bergemann et al. [11] provided characterizations for a symmetric calibrated click-through auction but with complete information of bidder values. Another line of related literature is the sale of information, which selectively reveals information for revenue improvement. There are a series of works in this line [2, 17, 9, 15, 39, 46, 37, 10]. Our work is also related to Bayesian persuasion [34, 1, 4, 24, 3, 45, 31]. We also refer interested readers to a comprehensive survey by Dughmi [23]. Finally, our work is partly related to the literature on automated bidding in auctions [36, 35, 28, 40, 8].

2 Preliminaries

We consider the (arguably more natural) Bayesian version of the *calibrated click-through auction* of [11], which is directly motivated by advertising auctions. At a high level, bidders in this auction are ranked by the products of their *private values* (i.e., the willingness-to-pay per click) and the seller's prediction of *click-through-rate* (*abbr.* CTR). Bergemann et al. [11] consider a basic setup in which bidders' private values are perfectly known to the seller. In this paper, we generalize their problem to the Bayesian setup by assuming that bidders' values are independently drawn from distributions. While each bidder knows his own private value, the seller only knows the distributions of bidder values (though our second main result further removes this assumption). This generalization is more aligned with the rich literature

of Bayesian auction design, starting from the seminal work of [41]. Similar to [11], we also aim to design a revenue-maximizing auction. Different from classic auction design, the seller in a click-through auction has an additional knob to tune, i.e., informing bidders about their CTRs through partial information disclosure (also known as *information design* [33]). So the optimal auction requires designs of both *incentives* and *information*.

We now describe the general auction setting. There are n bidders. The private value v_i of bidder (she) $i = 1, 2, 3, \ldots, n$ is independently drawn from some known distribution $F_i(v_i)$ (with a density function $f_i(v_i)$), over some bounded interval $[a, b]$ with $a, b \in R_{\geq 0} \cup \{\infty\}$ and $a < b$. Note that the density functions $f_1(v_1), f_2(v_2), \ldots, f_n(v_n)$ are not necessarily identical. As seen, our problem is a natural Bayesian variant of [11], whose settings can be obtained by allowing $f_i(v_i)$ to be some Dirac δ-function. i.e., a point distribution.

When auctions start, the seller privately observes a CTR vector $r = (r_1, r_2, \ldots, r_n) \in [\underline{r}, 1]^n$ for n bidders, which is drawn from a commonly known joint distribution $\lambda(r)$. $\underline{r} > 0$ is a small positive value close to 0. The CTR r_i represents the probability of bidder i's ad being clicked, which may be estimated by the seller with some machine learning method. For bidder i, assume the marginal probability $\lambda(r_i) \geq \xi$, where ξ is a small positive value since it does not make sense to consider a CTR with a marginal probability arbitrarily close to 0 in practice.

To maximize his revenue, the seller may privately disclose some information about r_i to bidder i. Suppose the seller has the power of commitment[2], and he designs a signaling scheme denoted as π, which is also observed by the bidders. Hence, conditioning on observing CTR vector r, the seller sends signal $s = (s_1, s_2, \ldots, s_n)$ with probability $\pi(s|r)$, where s_i is the signal privately sent to bidder i. In the rest of the paper, we assume that $r_i \in \mathcal{R}$ with $\mathcal{R}$ being a discrete and finite set of CTR values.

Calibrated Click-Through Auction. The calibrated click-through auction was previously studied by Bergemann et al. [11]. Initially, each bidder privately observes her value v_i and the seller privately observes the CTR vector r. The seller then sends a signal s_i to each bidder i privately. Upon receiving the signal s_i, bidder i submits her bid b_i to the auction and the auction determines the winner i^* by selecting the one with the highest product, i.e., $b_{i^*} s_{i^*} = \max_j b_j s_j$, and charges the winner for each realized click by

$$p_{i^*} = \max_{j \neq i^*} \frac{b_j s_j}{s_{i^*}}.$$

The winner only pays when a click is received[2]. Hence, the seller's expected revenue is $r_{i^*} p_{i^*}$. One can easily prove that the auction is truthful[2] for any $s_1, \ldots, s_n$, as it follows the "minimum-bid-to-win" payment rule. Therefore, without loss of generality, we will assume $b_i \equiv v_i$ in the rest of the paper.

Following [11], we adopt the concept *calibration* to information design. That is, given any private signal $s_i = s_i'$, bidder i's posterior estimation of CTR r_i should be equal to s_i', i.e.,

$$E[r_i | s_i = s_i'] = s_i'. \tag{1}$$

The calibration constraint is a consequence of the revelation principle. Given any signal realization $s_i = s_i'$, bidder i will interpret s_i' through Bayes updates by computing the expected CTR value $E[r_i | s_i = s_i']$. The calibration constraint simply requires that the signal

[2] More discussions about *pay-per-click*, commitment power and truthfulness are in Appendix B.

itself directly reflects this expected value so that bidders do not need to compute the expected CTR value by themselves. Such revelation-principle-style simplification is widely adopted in information design [34, 24, 33], where signals can without loss of generality be *persuasive* action recommendations. Analogously, in our auction setup, such direct information scheme will directly signal the posterior mean of the signal, i.e., obeying the calibration constraint (1). Hence, any signal s_i must be within $[\underline{r}, 1] \subseteq [0, 1]$ by (1). One important observation in [11] is that if the CTRs and signals are discrete, we can rewrite (1) as

$$\sum_{(r,s):s_i=s_i'} \lambda(r)\pi(s|r)(r_i - s_i') = 0. \tag{2}$$

Bayesian Calibrated Click-Through Auction. We focus on the seller's information design problem, i.e., to design a signaling scheme π in order to maximize the seller's expected revenue. The problem is formulated as below,

$$\begin{aligned} \max_{\pi,s} \quad & \sum_r \lambda(r) \sum_s \pi(s|r) \int_v f(v)R(r,v,s)dv \\ \text{subject to} \quad & E[r_i|s_i = s_i'] = s_i' \ \ \forall s_i' \in [0,1], \forall i \\ & \pi(s|r) \in [0,1], \sum_s \pi(s|r) = 1, \ \forall r, s \in [0,1]^n, \end{aligned} \tag{3}$$

where $f(v) = f_1(v_1)f_2(v_2)\cdots f_n(v_n)$ is the density function of $v = (v_1, v_2, \ldots, v_n)$, and the revenue $R(r, v, s) = r_{i^*}\frac{\max_{j\neq i^*} v_j s_j}{s_{i^*}}$. Since both the signals s and the probabilities of sending signals $\pi(s|r)$ are variables, the objective of (3) is non-convex.

Given CTR vector r and signal s, denote the seller's expected revenue as $R(r, s) = \int_v f(v)R(r, v, s)dv$. Specifically, $R(r, s)$ for 2 bidders is

$$\begin{aligned} R(r,s) &= \int_v f(v)R(r,v,s)dv \\ &= \int_a^b r_1 \frac{v_2 s_2}{s_1} \mathbf{Pr}\left(v_1 s_1 \geq v_2 s_2 | v_2\right) f_2(v_2)dv_2 + \int_a^b r_2 \frac{v_1 s_1}{s_2} \mathbf{Pr}\left(v_2 s_2 \geq v_1 s_1 | v_1\right) f_1(v_1)dv_1, \end{aligned} \tag{4}$$

where $\mathbf{Pr}\left(v_1 s_1 \geq v_2 s_2 | v_2\right)$ is the probability of bidder 1 winning given that bidder 2's value is v_2 (similarly for $\mathbf{Pr}\left(v_2 s_2 \geq v_1 s_1 | v_1\right)$).

By (4), we can see that our problem differs from [11] in a crucial way: With any fixed s, the winner is also fixed in [11] since they have complete information on v, while in our case, either bidder can be the winner with some probability due to the uncertainty in their valuations. This difference somewhat "smooths" our objective function while at the same time brings new challenges.

3 Click-Through Auctions in General Environments

In this section, we present an FPTAS for (3) achieving $1 - O(\epsilon)$ approximation when the number of bidders is a constant. This result hinges on a minor continuity assumption on bidders' value distribution. That is, we assume every distribution F_i has no point mass[3] and has finite second moment, i.e., $\mathbb{E}_{v_i\sim F_i}[v_i^2] < \infty$. Under these two mild technical assumptions, we prove the following theorem.

[3] The no-point-mass assumption alleviates the tie-breaking problem arising in the case of deterministic bidders' values. A discussion on this assumption is in Appendix D.

▶ **Theorem 3.1.** *For any small $\epsilon > 0$, there is an algorithm that computes a (multiplicative) $1 - O(\epsilon)$ approximate signaling scheme in $\text{poly}(|\mathcal{R}|^n, (\frac{n}{\epsilon})^n)$ time where n is the number of bidders and $|\mathcal{R}|$ is the size of set $\mathcal{R}$ of CTR values.*

To prove Theorem 3.1, we resort to discretizing the strategy space. Our starting point is a reformulation of Program (3) as a linear program with infinite dimension, and then convert the infinite-dimensional program into a finite-dimensional one by discretizing the signal space. The main technical challenge is to prove that the solution to the discrete program approximates the optimal solution to the original infinite-dimensional one. This is more involved than typical rounding approaches and hinges on the following two key properties:

a). For any signaling scheme, the value of the discrete objective is sufficiently close to the original objective. This is a consequence of the following Lipschitz continuity of revenue function $R(r, s)$, whose proof is deferred to Appendix C.

▶ **Lemma 3.2.** *$R(r, s)$ is Lipschitz continuous in $s \in [\underline{r}, 1]^n$ with some constant C_n, for any given CTR $r = (r_1, r_2, \ldots, r_n)$.*

b). For any solution to the original infinite dimensional program, there exists a corresponding feasible solution to the discrete program that is sufficiently close to the original solution.

These two properties together can ensure the existence of a solution to the discrete program yielding the revenue that approximates the optimal solution of the original infinite dimensional program. Thus by solving the discrete program, we can get an approximately optimal solution.

The major challenge here is to prove property b), as any naïve rounding of a signaling scheme from the continuous signal space to a discrete space would break the calibration constraints and hence end up with an infeasible rounded signaling scheme to the discrete program. To circumvent the difficulty, we introduce a novel technique to retain all the calibration constraints.

At a high level, we will reserve a small amount of probability mass from the given signaling scheme at the beginning and only apply rounding to the remaining probability mass. After the rounding step, we will redistribute the reserved probability mass to carefully fix all the calibration constraints broken by the rounding step. In particular, one has to be extremely careful to avoid the straightforward discretization structure of the signal space, because otherwise a large fraction of probability mass reservation will be needed for the fixing stage, which leads to a significant revenue loss and fails the $1 - O(\epsilon)$ approximation. The detailed proof of Theorem 3.1 is involved and relegated to Appendix C.2.

It is worthwhile to compare our FPTAS in Theorem 3.1 with a recent FPTAS for multi-channel Bayesian persuasion by Babichenko et al. [4], and highlight their key differences. While both rely on the continuity of the sender's utility functions, their techniques are significantly different. Specifically, the FPTAS of [4] applies to a constant number of states but many receivers (i.e., bidders) whereas our FPTAS applies to many states but a constant number of bidders. Their FPTAS discretizes the space of distributions over the states (thus requires a constant number of states) whereas our FPTAS discretizes the space of posterior CTR mean, which is why we have to adjust the scheme to satisfy the calibration constraints. Our choice of discretizing posterior means is due to its direct usage in click-through auctions. Such a special structure and the resultant challenges of fixing calibration constraint violation – which is the core difficulty in our proof – is not present in the setup of [4].

We remark that the discretization technique applied to Theorem 3.1 is in fact regardless of the expected revenue $R(r, s)$. To retain the calibration constraint and preserve the approximation simultaneously, one basic requirement is that $R(r, s)$ needs to be Lipschitz-continuous in signal s. Therefore, our algorithm can be treated as a framework that can accommodate different auctions (with Lipschitz continuous expected revenue $R(r, s)$) and

maintain the calibration of signals. Based on this observation, we consider one simple modified auction: a click-through auction with a reserved price. Reserve prices are widely studied in the literature and commonly used in the industry [43, 42]. Similarly, we only need to show $R(r, s)$ of this modified auction is Lipschitz-continuous for Corollary 3.3 to be true, which is deferred to Appendix C.3.

▶ **Corollary 3.3.** *In a click-through auction where winner pays at least some fixed reserve price p, for any constant number of bidders and any small $\epsilon > 0$, there is an algorithm that computes (multiplicatively) $1 - O(\epsilon)$ approximate signaling scheme with time complexity $poly(|\mathcal{R}|, \frac{1}{\epsilon})$.*

4 Click-Through Auctions in Symmetric Environments

While Theorem 3.1 provides an algorithm that can compute an approximately optimal signaling scheme in fairly general setups, the algorithm relies on solving large-scale linear programs thus may be too costly in reality and also lacks interpretability. In this section, we pursue the design of "simple" signaling schemes through the approximation lens. Given the challenge of the problem in general, we shall restrict our attention to a fundamental special case in this section – i.e., 2 bidders and symmetric environments – which is also a major focus of our preceding work by Bergemann et al. [11]. For this basic case, we first characterize that without calibration constraint, the optimal information design is governed by a single parameter called *optimal signal ratio*, which is at most 1. Such an observation motivates an explicit construction of a simple and *prior-free* signaling scheme, which is close to the optimal when bidder values' probability density function does not fluctuate much.

An environment is *symmetric* [11] if 1) the distribution of the products $v_1 r_1, v_2 r_2, \ldots, v_n r_n$ are exchangeable, and 2) the values v_i's are i.i.d. drawn from a distribution $F(v)$. Hence, the symmetric environment further implies that the distribution of $r_1, r_2, \ldots, r_n$ are also exchangeable, i.e., $\lambda(r_1, r_2, \ldots, r_n) = \lambda(perm(r_1, r_2, \ldots, r_n))$ where $perm(\cdot)$ is any permutation function. For the purpose of easy exposition, we allow r_i to be drawn from a slightly enlarged interval $[0, 1]$, with which the results obtained hold for $r_i \in [\underline{r}, 1]$ as in our original setting.

▶ **Definition 4.1.** *The distribution $F(v)$ (with density $f(v)$) has a monotone hazard rate (MHR) if the "hazard rate" of the distribution $\frac{f(v)}{1-F(v)}$ is increasing in v.*

MHR is a standard assumption widely used in economics [5, 32, 16]. Many distributions are MHR, such as uniform distribution, exponential distribution, gamma distribution, etc. Note that MHR implies regularity proposed by Myerson [41].

▶ **Theorem 4.2.** *There exists a prior-free signaling scheme that can be found in polynomial time and guarantees a (multiplicative) $0.995 \cdot (\underline{f}/\overline{f})^2$-approximation for any continuous MHR distribution $f(v)$, where $\overline{f}$ and $\underline{f}$ are the respective maximum and minimum values of $f(v)$ for $v \in [0, c]$ with any $c > 0$.*

The approximation ratio in Theorem 4.2 achieves its best ratio 0.995 for uniform distribution. More generally, the approximation ratio depends on the term $\underline{f}/\overline{f}$ which intuitively captures how much the distribution deviates from being uniform. Notably, both the algorithm and the signaling scheme require no knowledge about the value distribution F, i.e., F could be unknown to the seller (though Section 4.4 provides a better guarantee when F is known and satisfy certain properties). We remark that the signaling scheme in Theorem 4.2 is *simple* in the sense that its construction is parameterized by a single parameter called *optimal signal*

ratio. The information design problem thus can be easily optimized by empirically selecting a value that gives the best performance. More importantly, Theorem 4.2 also provides a robust solution to the problem, which can be very useful when it is hard or costly to accurately estimate the distributions of bidder values.

4.1 Key Primitive of the Construction: the Optimal Signal Ratio

Before presenting the proof, we discuss the main ingredient for the construction of the signaling scheme, the *optimal signal ratio*. Given a CTR vector $r = (h, l)$ (without loss of generality, assume $h \geq l$) and a signal pair $s = (s_1, s_2)$, the expected revenue by (4) can be rewritten as (Recall $f(v) = 0, \forall v \notin [0, c]$. Note that although the seller requires no knowledge about $F(v)$ to construct the scheme, the bidders' value should follow some prior $F(v)$.)

$$R(r, s) = \int_0^c (h\frac{v_2 s_2}{s_1}) \int_{\frac{v_2 s_2}{s_1}}^c f(v_1)dv_1 f(v_2)dv_2 + \int_0^c (l\frac{v_1 s_1}{s_2}) \int_{\frac{v_1 s_1}{s_2}}^c f(v_2)dv_2 f(v_1)dv_1. \tag{5}$$

Clearly, from (5), the expected revenue is determined by the signals through their ratio $x = \frac{s_2}{s_1}$, which we call the *signal ratio*. We define the *optimal signal ratio* to be the signal ratio maximizing (5). In the rest of the paper, without loss of generality, we assume $h = 1$. Then the optimal signal ratio only depends on the smaller CTR l, denoted as $x(l)$ with $l \in [0, 1]$.

▶ **Lemma 4.3.** *The optimal signal ratio $l < x(l) \leq 1$ for $l \in [0, 1)$ and $x(1) = 1$.*

Lemma 4.3 is directly implied by Lemma E.1 and E.2 in Appendix E, where more discussion about $x(l)$ can be found. There are two interesting implications from Lemma 4.3. The first interesting implication is that without the calibration constraint, it would be optimal for the seller to only send pairs of signals with the optimal signal ratio where a larger (*resp.* smaller) signal is observed by the bidder with a higher (*resp.* lower) CTR, which motivates our construction of the signaling scheme. The second implication is that by $l < x(l)$ in Lemma 4.3, information design induces more intensive competition between buyers by partial-information revelation (with signal ratio $x(l)$) than full revelation (with signal ratio l). This again explains the observations in [5] that revenue extraction in auctions needs competition and too "fine-grained" targeting information may lead to a thin market. More detailed discussions on the second implication are in Appendix F.

As an application of Lemma 4.3, we observed that $x(l) = 1$ for $l \in [0, 1]$ for any *exponential distribution*. It turns out that in this special case, the revenue-maximizing calibrated signaling scheme is simply to reveal no information, i.e., always sending one calibrated signal pair with $s_1 = s_2 = E[r_i]$. This leads to the following proposition, whose proof is deferred to Appendix G.5.

▶ **Proposition 4.4.** *If the value distribution is an exponential distribution with density $f(v) = \lambda e^{-\lambda v}, v \geq 0, \lambda > 0$, the signaling scheme revealing no information is revenue-optimal.*

4.2 Construction of the *Simple* Signaling Scheme

In this part, we present the construction of a simple signaling scheme. Recall that given the smaller CTR l, the maximum expected revenue is achieved when $\frac{s_2}{s_1} = x(l)$, i.e., the optimal signal ratio. In other words, one can construct an approximately optimal signaling scheme by sending signal pairs of the optimal signal ratio $x(l)$ as frequently as possible, while maintaining the calibration constraints.

Algorithm 1 Construction of *Simple* Signaling Scheme.

1. Given CTR vector $r = (1, l)$, we compute a list of candidate signals $\sigma_0, \sigma_1, \ldots, \sigma_{K-1}$, where $K = \lfloor \log_{x(l)} l \rfloor$ such that $x(l)^{K+1} \leq l \leq x(l)^K$. The signal σ_i is computed as

$$\sigma_K = 1, \sigma_0 = x(l)^K, \sigma_i = \sigma_{i-1} \cdot x(l), \forall i \in [K]. \tag{6}$$

2. Define $p(r, s) = \lambda(r)\pi(s|r)$ as the probability mass of sending signal s conditioning on CTR r. Let $(l, 1)$ and $(1, l)$ send signals (σ_k, σ_{k+1}) and (σ_{k+1}, σ_k), respectively, with the same probability mass

$$p\Big((l,1),(\sigma_k,\sigma_{k+1})\Big) = p\Big((1,l),(\sigma_{k+1},\sigma_k)\Big) = p_k, \forall k\{1,2,\ldots,K-1\}$$

where p_k is computed as

$$p_k = p_{k-1} \cdot \frac{1-\sigma_k}{\sigma_k - l} = p_0 \prod_{i=1}^{k} \frac{1-\sigma_i}{\sigma_i - l}, \forall k \in \{1, 2, \ldots, K-1\}. \tag{7}$$

3. By the above construction, we know $p\Big((l,1),(\sigma_{K-1},1)\Big) = p\Big((1,l),(1,\sigma_{K-1})\Big) = p_{K-1}$. To maintain calibration constraint, the seller will additionally send signal (σ_0, σ_0) with $p\Big((l,1),(\sigma_0,\sigma_0)\Big) = p\Big((1,l),(\sigma_0,\sigma_0)\Big) = z$, where

$$z = p_0 \cdot \frac{\sigma_0 - l}{l + 1 - 2\sigma_0}. \tag{8}$$

4. Choose a proper p_0 (other parameters z, p_i are then determined) so that

$$z + p_0 + p_1 + \cdots + p_{K-1} = \lambda(r) = \frac{1}{2}$$

Based on the above intuition, we then present the construction of the simple signaling scheme π depicted in Algorithm 1. Since the CTRs of bidders are exchangeable, we can separately consider the signaling scheme for each pair of CTR vectors. Without loss of generality, consider a pair of CTR vectors $(l, 1)$ and $(1, l)$ such that $\lambda(r = (l, 1)) = \lambda(r = (1, l)) = \frac{1}{2}$. In particular, we assume $x(l) < 1$ (otherwise, optimality can be easily achieved by revealing no information). With properly chosen parameter p_0, the constructed signaling scheme is calibrated and valid (see Appendix G.1). Note that by (8), the probability mass $z = 0$ if $\sigma_0 = l$, which implies all the signals sent are of the optimal signal ratio. Therefore, the optimal expected revenue is achieved.

▶ Remark 4.5. As mentioned previously, the construction only depends on one parameter, the optimal signal ratio $x(l)$. Besides, the scheme has a clear structure, i.e., computing a geometric series of signals $\{\sigma_i\}$ and then assigning probabilities, both of which can be done efficiently.

We present the following concrete example that describes the constructed signaling scheme.

▶ **Example 4.6.** Let $f(v)$ be the uniform distribution over $[0, 1]$. The click-through rates are $h = 1$ and $l = 0.6$, with $\lambda((h, l)) = \lambda((l, h)) = 0.5$. By solving (5), we obtain the optimal signal ratio $x = \frac{9}{10}$. We consider $k = 4$ levels of signals within $[0.6, 1)$ as $\sigma_0 = (\frac{9}{10})^4, \sigma_1 =$

Table 1 Example of a signaling scheme. Each entry corresponds to $\lambda(r)\pi(s|r)$, i.e., the probability mass of sending signals $s = (\sigma_i, \sigma_j)$ when observing the CTR vector r.

r \ s	(σ_0, σ_0)	(σ_0, σ_1)	(σ_1, σ_2)	(σ_2, σ_3)	(σ_3, σ_4)	(σ_1, σ_0)	(σ_2, σ_1)	(σ_3, σ_2)	(σ_4, σ_3)
(0.6, 1)	z	p_0	p_1	p_2	p_3	0	0	0	0
(1, 0.6)	z	0	0	0	0	p_0	p_1	p_2	p_3

$(\frac{9}{10})^3, \sigma_2 = (\frac{9}{10})^2, \sigma_3 = \frac{9}{10}, \sigma_4 = (\frac{9}{10})^0 = 1$. The signaling scheme is given by Table 1, where the probabilities z and p_0, p_1, p_2, p_3 are determined according to (7) and (8) to keep the construction a calibrated signaling scheme.

Note that all signal pairs except (σ_0, σ_0) follows the optimal signal ratio $x = \frac{9}{10}$. Hence the revenue suboptimality only happens when sending the signal $s = (\sigma_0, \sigma_0)$, of which the probability mass is $z \approx 0.016$. It turns out that the expected revenue of this calibrated signaling scheme is 0.2698. In contrast, by relaxing the calibration constraint, the maximum revenue one can achieve is 0.27 (i.e., always sending signals of the optimal signal ratio 9/10). Since 0.27 is clearly an upper bound of the optimal revenue for any calibrated signaling scheme, the revenue loss of our construction is less than 0.075%.

We remark that the revenue loss decreases very quickly as the number of signals k increases. In this example, if $l = 0.2$ instead, then the optimal signal ratio $x(l) = \frac{4}{5}$ and we can choose $k = 7$, yielding a much smaller probability mass of sending suboptimal signals $z \approx 1.5 \times 10^{-5}$.

4.3 Proof of Theorem 4.2

We first show a special case of Theorem 4.2, which is also a cornerstone for the main proof. That is, if the value distribution is uniform (i.e., $\underline{f}/\overline{f} = 1$), we can design a 0.995-approximate scheme.

▶ **Proposition 4.7.** *Given a uniform distribution supported on $[0, c]$, the constructed signaling scheme can achieve at least (multiplicative) 0.995-approximation.*

Simple calculation leads to the optimal signal ratio $x(l) = \frac{3+l}{4}$ under the uniform distribution. The proof of Proposition 4.7 is a combination of the following two lemmas. Recall that the constructed signaling scheme sends signal pairs of the optimal signal ratio $x(l)$ with probability $1 - z$ and with the remaining probability z sends the suboptimal signal pair (σ_0, σ_0). Lemma 4.8 bounds the probability z, while Lemma 4.9 lower bounds the revenue approximation when sending out (σ_0, σ_0). We will discuss the proof of Lemma 4.8 in Section 4.4, where a more general result is proved, and defer the proof of Lemma 4.9 to Appendix G.6.

▶ **Lemma 4.8.** *For optimal signal ratio function $x(l) = \frac{3+l}{4}$, there exists a signaling scheme whose worst-case approximation is $1 - z^*$, where $z^* \le 0.04$.*

▶ **Lemma 4.9.** *Sending signal (σ_0, σ_0), i.e., the signal ratio is 1, is (multiplicatively) $\frac{8}{9}$-approximation.*

Proof of Proposition 4.7. Combining Lemma 4.8 and Lemma 4.9, the approximation of the constructed signaling scheme is at least $(1 - z^*) \cdot 1 + z^* \cdot \frac{8}{9} \ge 224/225 \approx 0.995$. ◀

We prove Theorem 4.2 by showing that the signaling scheme we constructed for the uniform distribution is a $0.995\cdot(\underline{f}/\overline{f})^2$-approximation for any value distribution. In particular, sending signal pairs of signal ratio $\frac{3+l}{4}$ is $(\underline{f}/\overline{f})^2$-approximate and sending (σ_0,σ_0) is $\frac{8}{9}\cdot(\underline{f}/\overline{f})^2$-approximate. An analysis similar to Proposition 4.7 finally leads to $0.995\cdot(\underline{f}/\overline{f})^2$ approximation.

Proof of Theorem 4.2. We first derive the upper and lower bound of the expected revenue $R(r,s)$ by connecting to the uniform-distribution case. Given a CTR $r=(1,l)$ and any signal pair $s=(s_1,s_2)$ with ratio $\frac{s_2}{s_1}\le 1$ (by Lemma 4.3, $\frac{s_2}{s_1}\le 1$ is without loss of generality), the upper bound by (5) is computed as

$$\begin{aligned}
R(r,s)=&\int_0^c \frac{v_2s_2}{s_1}\int_{\frac{v_2s_2}{s_1}}^c f(v_1)dv_1f(v_2)dv_2+\int_0^{c\cdot\frac{s_2}{s_1}}(l\cdot\frac{v_1s_1}{s_2})\int_{\frac{v_1s_1}{s_2}}^c f(v_2)dv_2f(v_1)dv_1\\
\le&\int_0^c \frac{v_2s_2}{s_1}\int_{\frac{v_2s_2}{s_1}}^c \overline{f}dv_1\overline{f}dv_2+\int_0^{c\cdot\frac{s_2}{s_1}}(l\cdot\frac{v_1s_1}{s_2})\int_{\frac{v_1s_1}{s_2}}^c \overline{f}dv_2\overline{f}dv_1\\
=&(\overline{f}c)^2\cdot\Big(\int_0^c \frac{v_2s_2}{s_1}\int_{\frac{v_2s_2}{s_1}}^c \frac{1}{c}dv_1\frac{1}{c}dv_2+\int_0^{c\frac{s_2}{s_1}}(l\cdot\frac{v_1s_1}{s_2})\int_{\frac{v_1s_1}{s_2}}^c \frac{1}{c}dv_2\frac{1}{c}dv_1\Big)\\
=&(\overline{f}c)^2\cdot\overline{R}(r,s)
\end{aligned}$$

where $\overline{R}(r,s)$ computes the expected revenue under a uniform value distribution given CTR vector r and signal pair s. Similarly, the lower bound is $R(r,s)\ge(\underline{f}c)^2\cdot\overline{R}(r,s)$. Since $R(r,s)$ is only related to the signal ratio $x=\frac{s_2}{s_1}$, we rewrite $R(r,s)$ as $R(r,x)$. Similarly, rewrite $\overline{R}(r,s)$ as $\overline{R}(r,x)$. Let $x^*\le 1$ and $\overline{x}\le 1$ (by Lemma 4.3) be the optimal signal ratio for $R(r,x)$ and $\overline{R}(r,x)$, respectively. Then,

$$\begin{aligned}
&(\underline{f}c)^2\cdot\overline{R}(r,x^*)\le R(r,x^*)\le(\overline{f}c)^2\cdot\overline{R}(r,x^*)\\
&(\underline{f}c)^2\cdot\overline{R}(r,\overline{x})\le R(r,\overline{x})\le(\overline{f}c)^2\cdot\overline{R}(r,\overline{x})
\end{aligned}$$

Simple calculation leads to $(\underline{f}/\overline{f})^2\cdot R(r,x^*)\le(\underline{f}c)^2\cdot\overline{R}(r,x^*)\le(\underline{f}c)^2\cdot\overline{R}(r,\overline{x})\le R(r,\overline{x})$. The second inequality is due to the optimality of $\overline{x}$ to $\overline{R}(r,x)$. The whole inequality implies that if the seller sends signal pair whose ratio is optimal under a uniform value distribution, the approximation ratio is at least $(\underline{f}/\overline{f})^2$. Notice that Lemma 4.9 shows that under a uniform value distribution, sending a signal pair with signal ratio $\frac{s_2}{s_1}=1$ achieves $\frac{8}{9}$ approximation, i.e., $\frac{8}{9}\overline{R}(r,\overline{x})\le\overline{R}(r,1)$. Hence, we have $\frac{8}{9}(\underline{f}/\overline{f})^2\cdot R(r,x^*)\le\frac{8}{9}(\underline{f}c)^2\cdot\overline{R}(r,\overline{x})\le(\underline{f}c)^2\cdot\overline{R}(r,1)\le R(r,1)$, implying that if the seller sends signal pair with ratio equal to 1, then the approximation is $\frac{8}{9}\cdot(\underline{f}/\overline{f})^2$.

Now, we present the $0.995\cdot(\underline{f}/\overline{f})^2$ approximation. Lemma 4.8 shows that under a uniform distribution, the probability mass z of sending signal (σ_0,σ_0) is upper bounded as $z<0.04$, while $1-z>0.96$ probability mass is for sending signal pairs of the optimal signal ratio $\overline{x}$. Hence, if we construct a signaling scheme by assuming the unknown value distribution to be a uniform distribution, it can achieve at least $0.995\cdot(\underline{f}/\overline{f})^2$-approximation, where $0.995\approx 0.04\times\frac{8}{9}+0.96$. ◀

The above proof implies that the *prior-free* signaling scheme is constructed with signal ratio $x(l)=\frac{3+l}{4}$ obtained by assuming the unknown value distribution to be a uniform distribution. One direct result from the proof is that an easier but loose scheme is revealing no information which gives $\frac{8}{9}\cdot(\underline{f}/\overline{f})^2$ approximation. The improved ratio in Theorem 4.2 demonstrates the benefits of strategic information revelation.

4.4 Completing the Last Piece – Approximation Guarantee with Known Distributions

The only missing piece for completing the proof of Theorem 4.2 is the proof of Lemma 4.8, which is for a special linear optimal signal ratio function $x(l) = \frac{3+l}{4}$. In this part, we prove a more general result for any convex $x(l)$, which directly implies Lemma 4.8 with linear $x(l)$. To formally state the general proposition for convex $x(l)$, we need the following definitions.

▶ **Definition 4.10.** *Given that the optimal signal ratio $x(l)$ is a convex function in $l \in [0,1]$, define the following notations.*

1. *Initial number K_0: it satisfies that i) $x = x(l)^{K_0} \geq l$, and ii) $x = x(l)^{K_0+1}$ crosses the line $x = l$ and intersects at some point (l, l) with $l < 1$.*
2. *Define $S(k, l)$ with $\sigma_0 = x(l)^k$, $\sigma_i = x(l)^{k-i}$ as*

$$S(k,l) = 1 + \frac{l+1-2\sigma_0}{\sigma_0 - l} + \frac{l+1-2\sigma_0}{\sigma_0 - l} \cdot \frac{1-\sigma_1}{\sigma_1 - l} + \cdots + \frac{l+1-2\sigma_0}{\sigma_0 - l} \cdot \prod_{i=1}^{k-1} \frac{1-\sigma_i}{\sigma_i - l} \tag{9}$$

3. *Intersection point $l[k]$: given some $k > K_0$, $l[k] \neq 1$ is a solution to the equation $x(l)^k = l$. In another word, $x = x(l)^k$ crosses the line $x = l$ and intersects at the point $(l[k], l[k])$.*

Recall in the construction of the signaling scheme that given a CTR vector $r = (l, 1)$, at most $K = \lfloor \log_{x(l)} l \rfloor$ signals (i.e., σ_i) are within the interval $[l, 1]$. In fact, the initial number K_0 specifies the minimum number of signals constructed within $[l, 1]$ for any $l < 1$. Also, the probability mass z of sending (σ_0, σ_0) can be computed with the defined $S(k, l)$,

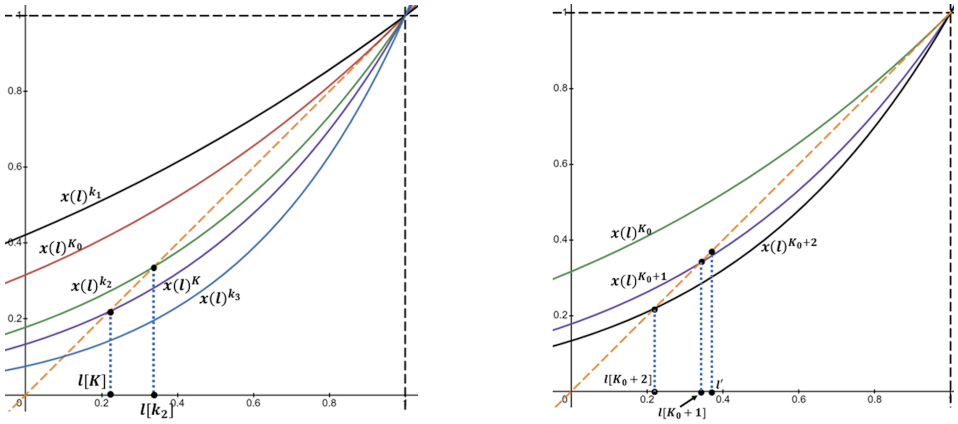
$$z + p_0 + p_1 + \cdots + p_{K-1} = \frac{1}{2} \quad \Longleftrightarrow \quad z \cdot S(K, l) = \frac{1}{2} \tag{10}$$

With the above definition, we present the general result.

▶ **Proposition 4.11.** *Assume $x(l)$ is a convex function. There exists a signaling scheme whose worst-case approximation is $1 - z^*$, where $z^* = 1/S(K_0, l[K_0+1])$ and $S(K_0, l[K_0+1])$ is computed as (9).*

Proposition 4.11 provides a *worst-case* approximation bound for the constructed signaling scheme and shows how it depends on the intrinsic properties of the optimal signal ratio function. Intuitively, by Equation (9), two situations lead to a small z^*: 1) The initial number K_0 is large. In this case, it implies that the seller can send many signal pairs with the optimal signal ratio and thus $S(K_0, l[K_0 + 1])$ grows exponentially; 2) $l[K_0 + 1]$ is close to σ_0 in (9). Later, we will show it is equivalent to that $l[K_0 + 1]$ is close to $x(l[K_0 + 1])^{K_0}$. This case will lead to a large $S(K_0, l[K_0 + 1])$ and thus small z^*. In fact, given the number of signals $k \geq K_0$ and the CTR l, the approximation of the constructed signaling scheme can be calculated similarly as $1 - z$ with $z = 1/S(k, l)$. As we will prove later, the worst-case approximation is obtained only when l is approaching $l[K_0 + 1]$ from above (see Figure 1b for an example). Hence, when l varies, the actual approximation of the constructed scheme may be (much) better than $1 - z^*$.

Proposition 4.11 may be of independent interest. If an estimate for the bidders' value distribution is available, the seller then can construct a signaling scheme as in Section 4.2 that has an approximation guarantee indicated by Proposition 4.11. Note that Proposition 4.11 only requires $x(l)$ to be convex. This condition applies to various classic distributions, e.g., (1) the uniform distribution admits a 0.995-approximation, (2) the exponential distribution can have the exact optimal mechanism, (3) the standard Weibull distribution (truncated on



(a) (b)

Figure 1 K_0 is initial number. $l[k]$ is the intersection point. $k_1 < K_0 < k_2 < K < k_3$. $x(l)$ is convex.

$[0,1]$) with parameter $\gamma \geq 2$ (specifically we let $\gamma = 10$) admits approximation ratio ≈ 0.75. In some cases, it may be difficult to write down the explicit formula of $x(l)$ and verify its convexity. We find it relatively easy to compute $l(x)$, the inverse of $x(l)$. Hence, to verify the convexity of $x(l)$, we only need to verify the concavity of $l(x)$. An example of this idea is in Appendix H.

Before presenting the proof of Proposition 4.11, we show some properties for convex $x(l)$.

▶ **Observation 4.12.** *The function $x = x(l)^k$, for any integer $k > 1$ and $l \in [0,1]$, intersects with line $x = l$ at most twice: one point intersecting is $(1,1)$, and the other one (if exists) is $(l[k], l[k])$ (called intersection point as in Definition 4.10). Furthermore, when $k > K_0$ increases, $l[k]$ decreases and $\lim_{k\to\infty} l[k] = 0$.*

The idea of Observation 4.12 is depicted in Figure 1a. Given an initial number K_0 and $K = \lfloor \log_{x(l')} l' \rfloor$ for some l', we can observe from the figure that: i) If $k \leq K_0$, the curve $x(l)^k$ will be above line $x = l$; ii) If $K_0 \leq k \leq K$, then $l' \leq l[K]$ and $x(l')^k$ will be above point (l', l'). If $l' = l[K]$, then there are exactly K signals constructed within $[l', 1]$; and iii) If $k > K$, $x(l')^k$ will be below (l', l').

The following key lemma characterizes the monotonicity of convex $x(l)$.

▶ **Lemma 4.13.** *$x(l)$ is a monotone increasing function.*

Now we are ready to show the proof of Proposition 4.11. The high-level idea of the proof is by upper bounding the probability mass z of sending (at most one) *non-optimal* signal pair (σ_0, σ_0), because when sending other signal pairs (σ_i, σ_{i+1}) as constructed in the signaling scheme, the signaling scheme achieves the maximum of expected revenue expressed in (5).

Proof of Proposition 4.11. Without loss of generality, we consider the case where there is only one pair of CTR vectors, $(l, 1)$ and $(1, l)$ such that $\lambda((l,1)) = \lambda(1,l)) = \frac{1}{2}$. By the definition of symmetric environments, the analysis can be easily generalized to the case of more than two CTR vectors.

The following lemma segments $[0,1]$ by the intersection points $l[k]$'s, and shows how the probability mass z changes within each segment. Note that starting from $k = K_0 + 1$, $x(l)^k$ crosses the line $x = l$ (see Figure 1b). Then, we define $l[K_0] = 1$.

▶ **Lemma 4.14.** *Given $k \geq K_0 \geq 1$, the probability mass $z > 0$ is decreasing in $l \in (l[k+1], l[k]]$.*

The proof of Lemma 4.14 is in Appendix G.4.The reason we only care about $(l[k+1], l[k]]$ instead of the closed interval $[l[k+1], l[k]]$ is that i) for $l[k+1] < l \leq l[k]$, there will be k signals constructed in the interval $[l, 1]$, and ii) if $l = l[k+1]$, the seller can construct a signaling scheme with a list of $k+1$ signals where all the signal pairs sent have the optimal signal ratio $x(l[k+1])$ and the probability mass $z = 0$.

One implication of Lemma 4.14 is that when there are at most k signals constructed, the probability mass z achieves its maximum when l approaches $l[k+1]$ from above, i.e., $l \downarrow l[k+1]$. Therefore, at the limit $l[k+1]$, the probability mass z is computed with $S(k, l[k+1])$ where $\sigma_0 = x(l[k+1])^k, \sigma_1 = x(l[k+1])^{k-1}, \cdots, \sigma_i = x(l[k+1])^{k-i}$. In another word, when $l = l[k+1]$, instead of constructing a scheme with a list of $k+1$ signals where all the signal pairs sent have signal ratio $x(l[k+1])$, the seller constructs a signaling scheme with a list of k signals starting from $\sigma_0 = x(l[k+1])^k$, which sends signal pair (σ_{i-1}, σ_i) of signal ratio $x(l[k+1])$ and one signal pair (σ_0, σ_0).

By the above analysis, to upper bound z, we only need to compare its values at these limit points $l[k]$'s. Alternatively, we compare $S(k, l[k+1])$ for different k. The following lemma shows that $S(k, l[k+1])$ is increasing in k, whose proof is in Appendix G.7.

▶ **Lemma 4.15.** *Given $k - 1 \geq K_0 \geq 1$, we have $S(k, l[k+1]) \geq S(k-1, l[k])$.*

Lemma 4.15 implies that $S(K_0, l[K_0+1])$ is the minimum compared with other $S(k, l[k+1])$ for $k > K_0$. The quantity $S(K_0, l[K_0+1])$ is computed as (9) with

$$\sigma_0 = x(l[K_0+1])^{K_0}, \sigma_1 = x(l[K_0+1])^{K_0-1}, \ldots, \sigma_i = x(l[K_0+1])^{K_0-i}$$

Since both CTR vectors $(1, l)$ and $(l, 1)$ will send signal (σ_0, σ_0), the upper bound of the probability mass of sending signal pair (σ_0, σ_0) in the constructed signaling scheme is $z^* = 1/S(K_0, l[K_0+1])$.

The above analysis generalizes to the case of more than two CTR vectors. Note that given any CTR vector (h, l), in the symmetric environment, we can find one CTR vector (l, h) with equal probability, i.e., $\lambda((h, l)) = \lambda((l, h))$. Hence, we can separately consider these pairs of CTR vectors when designing a signaling scheme and each of them achieves at least $(1 - z^*)$ (multiplicatively) approximation. Therefore, the constructed signaling scheme can achieve at least $(1 - z^*)$ (multiplicatively) approximation. ◀

5 Conclusions and Open Problems

This paper studies the natural Bayesian variant of the calibrated click-through auction of [11]. We focus on the seller's information design problem to maximize the expected revenue. In general environments, we develop an FPTAS to compute an approximately optimal signaling scheme. In a symmetric environment, we give a simple and prior-free signaling scheme with a constant approximation guarantee for not-too-fluctuating value distributions.

Our results raise many interesting questions for future research. Below we discuss some of them. The first is to develop a simple signaling scheme for more than two bidders in symmetric environments. More generally, it is interesting to study more efficient algorithms for optimal signaling. Though our FPTAS for the general model enriches our understanding of the tractability of the problem, such an algorithm may not be ideal from a practitioner's perspective. The second is to consider the worst-case approximation as in Proposition 4.11 but without the convexity assumption. In Appendix I,we provide more discussions about these two open problems.

References

1 Itai Arieli and Yakov Babichenko. Private bayesian persuasion. *Journal of Economic Theory*, 182:185–217, 2019.

2 Moshe Babaioff, Robert Kleinberg, and Renato Paes Leme. Optimal mechanisms for selling information. In *Proceedings of the 13th ACM Conference on Electronic Commerce (EC 2012)*, pages 92–109, 2012.

3 Yakov Babichenko and Siddharth Barman. Algorithmic aspects of private bayesian persuasion. In *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

4 Yakov Babichenko, Inbal Talgam-Cohen, Haifeng Xu, and Konstantin Zabarnyi. Multi-channel bayesian persuasion. In *Proceedings of 13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215, page 11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

5 Ashwinkumar Badanidiyuru, Kshipra Bhawalkar, and Haifeng Xu. Targeting and signaling in ad auctions. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*, pages 2545–2563, 2018.

6 Santiago Balseiro, Yuan Deng, Jieming Mao, Vahab Mirrokni, and Song Zuo. Robust auction design in the auto-bidding world. *Advances in Neural Information Processing Systems*, 34:17777–17788, 2021.

7 Santiago R Balseiro, Yuan Deng, Jieming Mao, Vahab S Mirrokni, and Song Zuo. The landscape of auto-bidding auctions: Value versus utility maximization. In *Proceedings of the 22nd ACM Conference on Economics and Computation*, pages 132–133, 2021.

8 Santiago R Balseiro and Yonatan Gur. Learning in repeated auctions with budgets: Regret minimization and equilibrium. *Management Science*, 65(9):3952–3968, 2019.

9 Dirk Bergemann, Alessandro Bonatti, and Alex Smolin. The design and price of information. *American economic review*, 108(1):1–48, 2018.

10 Dirk Bergemann, Yang Cai, Grigoris Velegkas, and Mingfei Zhao. Is selling complete information (approximately) optimal? In *Proceedings of the 23rd ACM Conference on Economics and Computation, (EC 2022)*, 2022.

11 Dirk Bergemann, Paul Dütting, Renato Paes Leme, and Song Zuo. Calibrated click-through auctions. In *Proceedings of the ACM Web Conference 2022*, pages 47–57, 2022.

12 Dirk Bergemann, Tibor Heumann, Stephen Morris, Constantine Sorokin, and Eyal Winter. Optimal information disclosure in auctions. *American Economic Review: Insights (forthcoming)*, 2022.

13 Umang Bhaskar, Yu Cheng, Young Kun Ko, and Chaitanya Swamy. Hardness results for signaling in bayesian zero-sum and network routing games. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, pages 479–496, 2016.

14 Peter Bro Miltersen and Or Sheffet. Send mixed signals: earn more, work less. In *Proceedings of the 13th ACM Conference on Electronic Commerce (EC 2012)*, pages 234–247, 2012.

15 Yang Cai and Grigoris Velegkas. How to sell information optimally: An algorithmic study. In *Proceedings of the 12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*, volume 185, 2021.

16 Shuchi Chawla, Jason D Hartline, and Robert Kleinberg. Algorithmic pricing via virtual valuations. In *Proceedings of the 8th ACM Conference on Electronic Commerce*, pages 243–251, 2007.

17 Yiling Chen, Haifeng Xu, and Shuran Zheng. Selling information through consulting. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2020)*, pages 2412–2431. SIAM, 2020.

18 Yu Cheng, Ho Yee Cheung, Shaddin Dughmi, Ehsan Emamjomeh-Zadeh, Li Han, and Shang-Hua Teng. Mixture selection, mechanism design, and signaling. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS 2015)*, pages 1426–1445. IEEE, 2015.

19 Constantinos Daskalakis, Christos Papadimitriou, and Christos Tzamos. Does information revelation improve revenue? In *Proceedings of the 17th ACM Conference on Economics and Computation (EC 2016)*, pages 233–250, 2016.

20 Yuan Deng, Negin Golrezaei, Patrick Jaillet, Jason Cheuk Nam Liang, and Vahab Mirrokni. Fairness in the autobidding world with machine-learned advice. *arXiv preprint*, 2022. `arXiv:2209.04748`.

21 Nikhil R Devanur, Bach Q Ha, and Jason D Hartline. Prior-free auctions for budgeted agents. In *Proceedings of the fourteenth ACM conference on Electronic commerce*, pages 287–304, 2013.

22 Shaddin Dughmi. On the hardness of signaling. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 354–363. IEEE, 2014.

23 Shaddin Dughmi. Algorithmic information structure design: a survey. *ACM SIGecom Exchanges*, 15(2):2–24, 2017.

24 Shaddin Dughmi and Haifeng Xu. Algorithmic persuasion with no externalities. In *Proceedings of the 18th ACM Conference on Economics and Computation (EC 2017)*, pages 351–368, 2017.

25 Benjamin Edelman, Michael Ostrovsky, and Michael Schwarz. Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. *American economic review*, 97(1):242–259, 2007.

26 Yuval Emek, Michal Feldman, Iftah Gamzu, Renato PaesLeme, and Moshe Tennenholtz. Signaling schemes for revenue maximization. *ACM Transactions on Economics and Computation (TEAC)*, 2(2):1–19, 2014.

27 Seyf Emen. A beginners guide to generalized second-price auctions, 2022. URL: `https://www.topsort.com/post/generalized-second-price-auctions-how#:~:text=The%20second%2Dprice%20auction%20is,bid%20to%20win%20the%20auction.`

28 Yiding Feng, Brendan Lucier, and Aleksandrs Slivkins. Strategic budget selection in a competitive autobidding world. *arXiv preprint*, 2023. `arXiv:2307.07374`.

29 Dean P Foster and Rakesh V Vohra. Calibrated learning and correlated equilibrium. *Games and Economic Behavior*, 21(1-2):40, 1997.

30 Hu Fu, Patrick Jordan, Mohammad Mahdian, Uri Nadav, Inbal Talgam-Cohen, and Sergei Vassilvitskii. Ad auctions with data. In *International Symposium on Algorithmic Game Theory (SAGT 2012)*, pages 168–179. Springer, 2012.

31 Ronen Gradwohl, Niklas Hahn, Martin Hoefer, and Rann Smorodinsky. Algorithms for persuasion with limited communication. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)*, pages 637–652, 2021.

32 Jason D Hartline and Tim Roughgarden. Simple versus optimal mechanisms. In *Proceedings of the 10th ACM conference on Electronic commerce (EC 2009)*, pages 225–234, 2009.

33 Emir Kamenica. Bayesian persuasion and information design. *Annual Review of Economics*, 11:249–272, 2019.

34 Emir Kamenica and Matthew Gentzkow. Bayesian persuasion. *American Economic Review*, 101(6):2590–2615, 2011.

35 Yoav Kolumbus and Noam Nisan. Auctions between regret-minimizing agents. In *Proceedings of the ACM Web Conference 2022*, pages 100–111, 2022.

36 Yoav Kolumbus and Noam Nisan. How and why to manipulate your own agent: On the incentives of users of learning agents. *Advances in Neural Information Processing Systems*, 35:28080–28094, 2022.

37 Yingkai Li. Selling data to an agent with endogenous information. In *Proceedings of the 23rd ACM Conference on Economics and Computation (EC 2022)*, 2022.

38 Zhuoshu Li and Sanmay Das. Revenue enhancement via asymmetric signaling in interdependent-value auctions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2093–2100, 2019.

39 Shuze Liu, Weiran Shen, and Haifeng Xu. Optimal pricing of information. In *Proceedings of the 22nd ACM Conference on Economics and Computation (EC 2021)*, pages 693–693, 2021.

40 Aranyak Mehta and Andres Perlroth. Auctions without commitment in the auto-bidding world. *arXiv preprint*, 2023. arXiv:2301.07312.

41 Roger B Myerson. Optimal auction design. *Mathematics of operations research*, 6(1):58–73, 1981.

42 Michael Ostrovsky and Michael Schwarz. Reserve prices in internet advertising auctions: A field experiment. In *Proceedings of the 12th ACM conference on Electronic commerce*, pages 59–60, 2011.

43 Renato Paes Leme, Martin Pal, and Sergei Vassilvitskii. A field guide to personalized reserve prices. In *Proceedings of the 25th international conference on world wide web*, pages 1093–1102, 2016.

44 Hal R Varian. Position auctions. *international Journal of industrial Organization*, 25(6):1163–1178, 2007.

45 Haifeng Xu. On the tractability of public persuasion with no externalities. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2020)*, pages 2708–2727. SIAM, 2020.

46 Shuran Zheng and Yiling Chen. Optimal advertising for information products. In *Proceedings of the 22nd ACM Conference on Economics and Computation (EC 2021)*, pages 888–906, 2021.

High-Accuracy Multicommodity Flows via Iterative Refinement

Li Chen
Carnegie Mellon University, Pittsburgh, PA, USA

Mingquan Ye
University of Illinois at Chicago, IL, USA

Abstract

The multicommodity flow problem is a classic problem in network flow and combinatorial optimization, with applications in transportation, communication, logistics, and supply chain management, etc. Existing algorithms often focus on low-accuracy approximate solutions, while high-accuracy algorithms typically rely on general linear program solvers. In this paper, we present efficient high-accuracy algorithms for a broad family of multicommodity flow problems on undirected graphs, demonstrating improved running times compared to general linear program solvers. Our main result shows that we can solve the $\ell_{q,p}$-norm multicommodity flow problem to a $(1+\varepsilon)$ approximation in time $O_{q,p}(m^{1+o(1)}k^2\log(1/\varepsilon))$, where k is the number of commodities, and $O_{q,p}(\cdot)$ hides constants depending only on q or p. As q and p approach to 1 and ∞ respectively, $\ell_{q,p}$-norm flow tends to maximum concurrent flow.

We introduce the first iterative refinement framework for $\ell_{q,p}$-norm minimization problems, which reduces the problem to solving a series of decomposable residual problems. In the case of k-commodity flow, each residual problem can be decomposed into k single commodity convex flow problems, each of which can be solved in almost-linear time. As many classical variants of multicommodity flows were shown to be complete for linear programs in the high-accuracy regime [Ding-Kyng-Zhang, ICALP'22], our result provides new directions for studying more efficient high-accuracy multicommodity flow algorithms.

2012 ACM Subject Classification Theory of computation → Continuous optimization

Keywords and phrases High-accuracy multicommodity flow, Iterative refinement framework, Convex flow solver

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.45

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/pdf/2304.11252`

Funding *Li Chen*: Supported by NSF grants CCF-2106444 and CCF-2330255.
Mingquan Ye: Supported by NSF grant CCF-2240024.

Acknowledgements We thank Richard Peng and Xiaorui Sun for helpful discussions.

1 Introduction

The multicommodity flow problem is a classic challenge in network flow and combinatorial optimization, where the goal is to optimally route multiple commodities through a network from their respective sources to their respective sinks, subject to flow conservation constraints. This problem has significant applications in various fields such as transportation, communication, logistics, and supply chain management [43, 6, 57, 9, 71]. Currently, the fastest algorithms for computing high-accuracy solutions involve formulating these problems as linear programs and employing generic linear program solvers [44, 39, 62, 22]. Notably, linear programs can be reduced to multicommodity flow problems with near-linear overhead [35, 26].

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 45; pp. 45:1–45:19


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Existing research predominantly focuses on obtaining $(1+\varepsilon)$-approximate solutions for maximum concurrent k-commodity flows [49, 48, 73, 27, 37, 30, 56, 51], as summarized in Table 1. However, these low-accuracy algorithms feature running times that are polynomial in $1/\varepsilon$ for computing $(1+\varepsilon)$-approximate solutions. In contrast, high-accuracy algorithms exhibit running times that are polynomial in $\log(1/\varepsilon)$. Importantly, [26] demonstrated that any enhancement in the high-accuracy algorithm for the 2-commodity flow problem would result in a faster general linear program solver.

In this paper, we investigate multicommodity flow problems on undirected graphs, which possess more structure than their directed counterparts. Prior work has shown that maximum concurrent 2-commodity flow on undirected graphs can be reduced to two instances of maximum flow problems, both solvable in almost-linear time [63, 15]. More generally, maximum concurrent k-commodity flows can be reduced to 2^{k-1} maximum flows[1]. Additionally, researchers have discovered that $(1+\varepsilon)$-approximate algorithms for undirected graphs are considerably faster than those for directed graphs [42, 41, 66]. Nevertheless, the fastest high-accuracy algorithms still rely on general linear program solvers. Given these advancements, we pose the following natural question:

> *Is it possible to solve multicommodity flow problems on undirected graphs to high accuracy more efficiently than with general linear program solvers?*

This paper gives an affirmative answer and presents high-accuracy algorithms for a large family of multicommodity flow problems that run in time $m^{1+o(1)}\mathrm{poly}(k, \log(1/\varepsilon))$. Our main result is an algorithm that, for any $1 \le q \le 2 \le p$ with $p = \widetilde{O}(1)$[2] and $\frac{1}{q-1} = O(1)$, given edge weights $\mathbf{w} \in \mathbb{R}^E_+$ and k vertex demands $\mathbf{D} = [\mathbf{d}_1, \cdots, \mathbf{d}_k] \in \mathbb{R}^{V \times k}$, solves the following optimization problem to a $(1+\varepsilon)$-approximation with high probability[3] in time $O_{q,p}(m^{1+o(1)}k^2 \log(1/\varepsilon))$:

$$\min_{k\text{-commodity flow } \mathbf{F} \text{ with residue } \mathbf{D}} \|\mathbf{WF}\|_{q,p}^{pq} \stackrel{\text{def}}{=} \sum_{e \in E} w_e^{pq} \left(\sum_{j=1}^{k} |F_{ej}|^q \right)^p .$$

The problem generalizes the maximum concurrent flow problem by setting the edge weights w_e as the reciprocal of edge capacities and letting $q \to 1, p \to \infty$. Therefore, $\ell_{q,p}$-norm flows are natural relaxations of the combinatorial maximum concurrent flows. However, unlike the typical relaxations using the exponential function (exp(congestion)) in previously efficient approximation schemes, we show that $\ell_{q,p}$-norm problems themselves admit high-accuracy solutions. Thus, we provide a large family of multicommodity flows that admit high-accuracy solutions in almost linear time.

From a technical standpoint, our exploration of multicommodity flows reflects the research trajectory of ℓ_p-norm single commodity flows in recent years [46, 1]. This line of study has led to the development of several novel algorithmic components, some of which have proven beneficial for classical single-commodity flow as well [40, 7]. More significantly, examining ℓ_p-norm flows, particularly their weighted variants, has directed attention towards the core challenges of flow problems. We posit that further investigation of $\ell_{q,p}$-norm flows is likely to yield similar insights, potentially for variants of multicommodity flows not known to be hard, such as unit-capacity maximum concurrent flows on undirected graphs.

---

[1] To see this, the edge capacity constraint $\sum_j |F_{ej}| \le u_e$ is equivalent to $|\sum_j \varepsilon_j F_{ej}| \le u_e$ for any $\varepsilon \in \{\pm 1\}^k$.

[2] We use $\widetilde{O}(f(n))$ to hide $\mathrm{poly}\log f(n)$ factors.

[3] We use "with high probability" (w.h.p.) throughout to say that an event happens with probability at least $1 - n^{-C}$ for any constant $C > 0$.

To summarize, this paper introduces the first iterative refinement framework for solving $\ell_{q,p}$-norm minimization problems. The proposed framework reduces the problem to approximately resolving $O_{p,q}(k\log(1/\varepsilon))$ instances of decomposable residual problems. For the k-commodity flow case, each residual problem can be divided into k single commodity flow problems and solved in $km^{1+o(1)}$-time using the almost-linear time convex flow solver [15]. We are the first to combine the iterative refinement framework [2, 3] with the convex flow solver [15], and give a non-trivial application to $\ell_{q,p}$-norm multicommodity flow problem. As many classical variants of multicommodity flows were shown to be complete for linear programs in the high-accuracy regime [26], our result provides new directions for studying more efficient high-accuracy multicommodity flow algorithms.

1.1 Main Result

Given an undirected graph $G=(V,E)$ and parameters q and p, the *$\ell_{q,p}$-norm multicommodity flow* problem asks for a multicommodity flow $\mathbf{F}=[\mathbf{f}_1,\cdots,\mathbf{f}_k]\in\mathbb{R}^{E\times k}$ that routes the given demands while minimizing the following objective:

$$\min_{\mathbf{B}^\top\mathbf{F}=\mathbf{D}}\|\mathbf{WF}\|_{q,p}^{pq}\stackrel{\text{def}}{=}\sum_{e\in E}w_e^{pq}\left(\sum_{j=1}^{k}|F_{ej}|^q\right)^p. \tag{1}$$

Here, we have

1. $\mathbf{B}\in\mathbb{R}^{E\times V}$, the edge-vertex incidence matrix of G;
2. $\mathbf{D}=[\mathbf{d}_1,\cdots,\mathbf{d}_k]\in\mathbb{R}^{V\times k}$, representing the set of k vertex demands, where $\langle\mathbf{d}_j,\mathbf{1}_V\rangle=0$ for each $j\in[k]$;
3. $\mathbf{w}\in\mathbb{R}_+^E$, the vector of edge weights, and $\mathbf{W}=\operatorname{diag}(\mathbf{w})$.

This problem can be seen as a generalization of the classical maximum concurrent flow problem, which aims to find a feasible flow $\mathbf{F}$ that minimizes edge congestion:

$$\min_{\mathbf{B}^\top\mathbf{F}=\mathbf{D}}\max_{e\in E}\frac{1}{c_e}\sum_{j\in[k]}|F_{ej}|=\left\|\mathbf{C}^{-1}\mathbf{F}\right\|_{1,\infty}. \tag{2}$$

Here, $\mathbf{c}\in\mathbb{R}_+^E$ denotes the vector of edge capacities and $\mathbf{C}=\operatorname{diag}(\mathbf{c})$. Our generalization allows for fractional vertex demands, meaning that for each commodity j, the demand $\mathbf{d}_j$ is not restricted to source-sink pairs.

The main technical result of this paper is presented below.

▶ **Theorem 1.** *Given any $1<q\le 2\le p$ with $p=\widetilde{O}(1)$ and $\frac{1}{q-1}=O(1)$ and an error parameter $\varepsilon>\exp(-\widetilde{O}(1))$, let* OPT *be the optimal value to Problem* (1)*. There is a randomized algorithm that computes a feasible flow $\mathbf{F}$ to Problem* (1) *such that*

$$\|\mathbf{WF}\|_{q,p}^{pq}\le(1+\varepsilon)\,\mathrm{OPT}+\varepsilon.$$

The algorithm runs in time

$$O\left(p^2\cdot\left(\frac{1}{q-1}\right)^{\frac{1}{q-1}}m^{1+o(1)}\cdot k^2\cdot\log\frac{1}{\varepsilon}\right)$$

with high probability.

Table 1 A summary of algorithms for the max concurrent k-commodity flow problem.

Year	References	Time	Directed?
1990	[64]	$O(nm^7/\varepsilon^5)$	Directed
1991	[49]	$\widetilde{O}(mnk/\varepsilon^3)$	Directed
1996	[34]	$\widetilde{O}(mnk/\varepsilon^2)$	Directed
1996	[61]	$\widetilde{O}(mnk/\varepsilon^2)$	Directed
2009	[56]	$\widetilde{O}(k^2m^2/\varepsilon)$	Directed
2010	[51]	$\widetilde{O}((m+k)n/\varepsilon^2)$	Directed
2012	[42]	$\widetilde{O}(m^{4/3}\text{poly}(k, \frac{1}{\varepsilon}))$	Undirected
2014	[41]	$m^{1+o(1)}k^2/\varepsilon^2$	Undirected
2017	[66]	$\widetilde{O}(mk/\varepsilon)$	Undirected
2019	[21]	$(mk)^{\omega+o(1)} \log \frac{1}{\varepsilon}$	Directed

1.2 Related Works

In this section, we discuss some work related to the problem and the techniques we use.

Multicommodity Flow

The multicommodity flow problem is a classic problem in network flow and combinatorial optimization. Multicommodity flow has a wide range of applications in various fields which are addressed in numerous surveys [43, 6, 57, 9, 71]. These problems can be formulated as linear programs and solved using generic linear program solvers which remain the fastest algorithms for computing high-accuracy solutions [44, 39, 62, 22]. On the other hand, linear programs can be reduced to 2-commodity flow efficiently [35, 26]. Specifically, any linear program can be reduced to a maximum throughput 2-commodity flow on sparse graphs with a near-linear overhead [26]. Recently, Brand and Zhang [13] proposed a faster algorithm for 2-commodity flow on non-sparse graphs with running time $\widetilde{O}(\sqrt{m}n^{\omega-1/2})$ that outperforms the existing time complexities $\widetilde{O}(m^\omega)$ [22] and $\widetilde{O}(\sqrt{m}n^2)$ [36]. For the general k-commodity flow, they proposed an algorithm with time complexity $\widetilde{O}(k^{2.5}\sqrt{m}n^{\omega-1/2})$.

Much of the existing works focus on finding $(1+\varepsilon)$-approximate solutions. [64] gave the first FPTAS to the maximum concurrent flow problem with unit capacity (Problem (2)). Subsequently, a series of work [49, 48, 32, 33, 45, 38, 60, 34, 61, 67] based on Lagrangian relaxation and linear program decomposition gave algorithms with improved running times for various versions of the problem with arbitrary edge capacity. These algorithms iteratively update the current flow and make progress by computing a series of either shortest paths [64, 45, 60, 67], or single commodity minimum cost flows [49, 48, 32, 33, 38, 34, 61]. In particular, [61] and [34] showed that finding $(1+\varepsilon)$-approximate solutions can be reduced to $\widetilde{O}(k/\varepsilon^2)$ min-cost flow computations which take $\widetilde{O}(kmn/\varepsilon^2)$-time in total using the fastest min-cost flow algorithm at the time. Combining with the almost-linear time min-cost flow algorithm yields a randomized $m^{1+o(1)}k/\varepsilon^2$-time max concurrent flow algorithm.

Later, a series of works based on multiplicative weight updates (MWU) gave conceptually simpler and faster algorithms at their time [73, 27, 37, 30, 51]. These methods build the solution from scratch without re-routing the current flow. At each step, they augment the current flow via a shortest path computation that favors relatively uncongested paths. [51] used dynamic APSP data structure to speed up these computations and resulted in an

$\widetilde{O}((m+k)n/\varepsilon^2)$-time max concurrent flow algorithm. At the time, the fastest max concurrent flow runs in $\widetilde{O}(m^{1.5}k/\varepsilon^2)$-time due to the $\widetilde{O}(m^{1.5})$-time min-cost flow algorithm by [24]. The result of [51] was a significant improvement in the case when k is large.

Another line of work focused on improving the $1/\varepsilon^2$ term. [11] found an FPTAS that only has $O(\frac{1}{\varepsilon \log 1/\varepsilon})$ dependence. Later, [56] gave an FPTAS with only $O(1/\varepsilon)$ dependence. In particular, the algorithm by Nesterov runs in $\widetilde{O}(k^2m^2/\varepsilon)$-time.

Similar to the situation of single commodity flows, researchers have discovered approximate algorithms with $m^{1.5-O(1)}$ dependence on undirected graphs. [42] gave the first $\widetilde{O}(m^{4/3}\mathrm{poly}(k, 1/\varepsilon))$ algorithm for max concurrent flow. The algorithm implements the method of [49, 48] using *electrical capacity-constrained flows* instead of min-cost flows. Each electrical capacity-constrained flow can be reduced to, using width-reduction MWU [17], $\widetilde{O}(m^{1/3}\mathrm{poly}(k, 1/\varepsilon))$ graph Laplacian systems. In the breakthrough result of [41], they improved the running time to $m^{1+o(1)}k^2/\varepsilon^2$ based on non-Euclidean gradient descent and fast oblivious routing. Specifically, the algorithm computes an oblivious routing of congestion $m^{o(1)}$ and uses it to reduce the number of gradient descent iterations to $m^{o(1)} \cdot k/\varepsilon^2$. Later, [66] introduced the idea of *area convexity* and improved the iteration count to $\widetilde{O}(1/\varepsilon)$. This resulted in the first $\widetilde{O}(mk/\varepsilon)$-time max concurrent flow algorithm.

ℓ_p-Norm Regression

The ℓ_p-norm regression problem seeks to find a vector $\mathbf{x}$ that minimizes $\|\mathbf{Ax} - \mathbf{b}\|_p$, where $\mathbf{A} \in \mathbb{R}^{d\times n}$ and $\mathbf{b} \in \mathbb{R}^d$. Varying in p interpolates between linear regression ($p = 2$) and linear program ($p \in \{1, \infty\}$). ℓ_p-norm regression has gained significant attention in the past decade due to its wide range of applications and its implications for other convex optimization problems [54, 72]. Many works have focused on low-accuracy algorithms for overconstrained matrices, i.e., $d \gg n$ [25, 54, 72, 19, 20, 18]. These results show various ways to find another matrix $\widetilde{\mathbf{A}}$ with fewer rows such that $\|\widetilde{\mathbf{A}}\mathbf{x}\|_p \approx \|\mathbf{Ax}\|_p$ for any $\mathbf{x}$. Then, approximate ℓ_p-norm regression can be reduced to $\widetilde{\mathbf{A}}$ and solved in time $O(\mathsf{nnz}(\mathbf{A}) + \mathrm{poly}(d, 1/\varepsilon))$ when p is a constant.

High-accuracy solutions can be found using interior point methods (IPM) in $\widetilde{O}(\sqrt{n})$ or $\widetilde{O}(\sqrt{d})$ iterations [47]. [14] showed a homotopy method that finds high accuarcy solution in $\widetilde{O}(n^{|1/2-1/p|})$ iterations of linear system solvers. Based on the idea of iterative refinement and width reduction, a series of works [2, 4, 5, 3] obtained improved iteration complexities of $\widetilde{O}_p(n^{(p-2)/(3p-2)})$ for $p \geq 2$. Motivated by the success of sparse linear system solver [59] and linear program in matrix multiplication time [22], [31] gave a high-accuracy algorithm that runs in time $\widetilde{O}_p(n^\theta)$ for some $\theta < \omega - \Omega(1)$, where ω is the matrix multiplication time exponent.

ℓ_p-Norm Flows

The $\ell_{q,p}$-norm formulation can be viewed as a multicommodity extension of the ℓ_p-norm flows, which seeks to find a flow $\mathbf{f} \in \mathbb{R}^E$ that routes the given demand and minimizes $\|\mathrm{diag}(\mathbf{w})\mathbf{f}\|_p$ where $\mathbf{w} \in \mathbb{R}^E_+$. Varying in p interpolates between the transshipment ($p = 1$), the electrical flow ($p = 2$), and the maximum flow problem ($p = \infty$). Combining the result on ℓ_p-norm regressions and Laplacian solvers, [2] gave an $\widetilde{O}_p(m^{1+|p-2|/(2p+|p-2|)})$-time ℓ_p-norm flow algorithm. Opening up the black box of the Spielman-Teng Laplacian solver [68], [46] gave the first $\widetilde{O}_p(m^{1+O(1/\sqrt{p})})$-time high-accuracy ℓ_p-norm flow algorithm for unweighted graphs, i.e., $\mathbf{w} = \mathbf{1}$. The runtime is almost linear for $p = \omega(1)$ and this was the first almost-linear time high-accuracy algorithm for a large family of single commodity flow problems. In the weighted case, [1] gave a $p(m^{1+o(1)} + n^{4/3+o(1)})$-time high-accuracy ℓ_p-norm flow algorithm

by combining [46] with the idea of sparsification. The study of the ℓ_p-norm flow algorithms has been proved useful for single commodity flow problems such as unit-capacity maximum flows, bipartite matchings, and min-cost flows [40, 7].

Continuous Optimization on Graphs

From a technical point of view, our $\ell_{q,p}$-norm multicommodity flow algorithm is inspired by the recent trend of applying continuous optimization techniques to solve graph problems. As one of the earlier results in this direction, [24] combined interior point methods with fast graph Laplacian system solver [68] and gave an $\widetilde{O}(m^{3/2})$-time min-cost flow algorithm. Later, the idea culminated in a decade of works improving max flow and min-cost flow algorithms [17, 52, 65, 41, 53, 58, 23, 7, 12, 40, 70, 8, 10, 15, 29, 69].

Beyond classical flow problems, the idea of combining continuous and combinatorial techniques gives improved algorithms for approximate shortest paths in parallel/distributed setting [50, 74], faster network flow algorithms in distributed setting [28], flow diffusion [16], ℓ_p-norm flows [46, 1], and more.

1.3 Our Approach

For the clarity of the presentation, we focus on the unweighted version of Problem (1), i.e., $w_e = 1$ for each edge e, which is shown as follows:

$$\min_{\mathbf{B}^\top \mathbf{F} = \mathbf{D}} \|\mathbf{F}\|_{q,p}^{pq} = \sum_{e \in E} \left(\sum_{j=1}^{k} |F_{ej}|^q \right)^p .$$

The algorithm follows an overall *iterative refinement* framework. That is, given the current flow $\mathbf{F}$, we want to find an update direction $\mathbf{\Delta}$ so that $\|\mathbf{F} + \mathbf{\Delta}\|_{q,p}^{pq}$ is smaller than $\|\mathbf{F}\|_{q,p}^{pq}$. However, finding the optimal $\mathbf{\Delta}$ is equivalent to the original problem. The idea of iterative refinement is to find a proxy *residual* function $\mathcal{R}(\mathbf{\Delta}; \mathbf{F})$ that approximates the Bregman divergence of $\|\mathbf{F}\|_{q,p}^{pq}$, i.e.,

$$\mathcal{R}(\mathbf{\Delta}; \mathbf{F}) \approx \|\mathbf{F} + \mathbf{\Delta}\|_{q,p}^{pq} - \|\mathbf{F}\|_{q,p}^{pq} - \langle \mathbf{G}, \mathbf{\Delta} \rangle ,$$

where $\mathbf{G} \stackrel{\text{def}}{=} \frac{\mathrm{d}}{\mathrm{d}\mathbf{F}} \|\mathbf{F}\|_{q,p}^{pq} \in \mathbb{R}^{E \times k}$ is the gradient. Then we can compute the direction $\mathbf{\Delta}$ by solving the *residual problem*:

$$\min_{\mathbf{B}^\top \mathbf{\Delta} = \mathbf{0}} \langle \mathbf{G}, \mathbf{\Delta} \rangle + \mathcal{R}(\mathbf{\Delta}; \mathbf{G}) \approx \|\mathbf{F} + \mathbf{\Delta}\|_{q,p}^{pq} - \|\mathbf{F}\|_{q,p}^{pq}.$$

If $\mathcal{R}(\mathbf{\Delta}; \mathbf{F})$ is a good approximation to the Bregman divergence, i.e., has a "condition number" of κ, we would obtain a $(1+\varepsilon)$-approximate solution in $O(\kappa \log(1/\varepsilon))$ iterations. On the other hand, $\mathcal{R}(\mathbf{\Delta}; \mathbf{F})$ should be computationally easier to minimize so that we can implement each iteration efficiently.

For any $p > 1$, [2] shows that $|x + \delta|^p$ can be locally approximated by a linear term plus an error term $\gamma_p(\delta; |x|)$, which behaves quadratically in δ when $|\delta| \leq |x|$ and as $|\delta|^p$ otherwise. Our key lemma (Lemma 8) extends the observation to approximating $\|\mathbf{F} + \mathbf{\Delta}\|_{q,p}^{pq}$ and gives

$$\begin{aligned} \|\mathbf{F} + \mathbf{\Delta}\|_{q,p}^{pq} - \|\mathbf{F}\|_{q,p}^{pq} - \langle \mathbf{G}, \mathbf{\Delta} \rangle \approx{}& O_{p,q}(k) \sum_{e,j} \|\mathbf{f}^e\|_q^{q(p-1)} \gamma_q(\Delta_{ej}; F_{ej}) \\ &+ O_{p,q}(k^p) \sum_{e,j} |\Delta_{ej}|^{pq}. \end{aligned} \tag{3}$$

Intuitively, the Bregman divergence can be approximated by a decomposable function on each coordinate (e, j). The contribution of each coordinate (e, j) behaves differently depending on the absolute value of Δ_{ej}. When $|\Delta_{ej}| \leq |F_{ej}|$, it behaves quadratically in $|\Delta_{ej}|^q$; when $|\Delta_{ej}| > |F_{ej}|$ but smaller than $\|\mathbf{f}^e\|_q^q$, the summation of k flow values to the power of q on edge $e \in E$, it is dominated by the term $|\Delta_{ej}|^{pq}$. The factors k and k^p in Equation (3) come from that given any k-dimensional vector $\mathbf{x} \in \mathbb{R}^k$, we have

$$\|\mathbf{x}\|_p^p \leq \|\mathbf{x}\|_1^p \leq k^{p-1} \|\mathbf{x}\|_p^p .$$

Surprisingly, this approximation has a conditioner number of $O_{q,p}(k)$ and is decomposable for each commodity $j \in [k]$. To obtain a $(1+\varepsilon)$-approximate solution, we only need to solve $O_{q,p}(k \log(1/\varepsilon))$ iterations of residual problems of the form

$$\min_{\mathbf{B}^\top \boldsymbol{\Delta} = \mathbf{0}} \langle \mathbf{G}, \boldsymbol{\Delta} \rangle + O_{p,q}(k) \sum_{e,j} \|\mathbf{f}^e\|_q^{q(p-1)} \gamma_q(\Delta_{ej}; F_{ej}) + O_{p,q}(k^p) \sum_{e,j} |\Delta_{ej}|^{pq}.$$

The decomposability allows us to use the convex flow solver from [15] and solve the residual problem to high accuracy for each commodity in almost-linear time. Thus, each iteration takes $km^{1+o(1)}$-time and the final running time is $O_{q,p}(k^2 m^{1+o(1)} \log(1/\varepsilon))$.

1.4 Paper Organization

In Section 2, we introduce some preliminaries before presenting the technical parts, including the convex flow solver [15] that is the core tool for solving the residual problem and the iterative refinement framework shown in [2, 3]. We formally present the proposed $\ell_{q,p}$-norm k-commodity flow algorithm in Section 3, and then solve the residual problem in Section 4.

2 Preliminaries

2.1 General Notations

We denote vectors (resp. matrices) by boldface lowercase (reps. uppercase) letters. For a vector $\mathbf{x} \in \mathbb{R}^n$, the scalar x_i, $i \in [n]$ represents the i-th entry of $\mathbf{x}$. For two vectors $\mathbf{x}$ and $\mathbf{y}$, the vector $\mathbf{x} \cdot \mathbf{y}$ represents the entrywise product, i.e., $(\mathbf{x} \cdot \mathbf{y})_i = x_i y_i$. Besides, for a vector $\mathbf{x}$, let $|\mathbf{x}|$ and $\mathbf{x}^p$ denote the entrywise absolute value and entrywise power of $\mathbf{x}$ respectively, that is, $|\mathbf{x}|_i = |x_i|$ and $(\mathbf{x}^p)_i = (x_i)^p$. We use $\langle \mathbf{x}, \mathbf{y} \rangle$ to denote the inner product of $\mathbf{x}, \mathbf{y}$, i.e., $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top \mathbf{y}$. For a vector $\mathbf{x}$, let $\mathrm{diag}(\mathbf{x})$ represent a diagonal matrix whose i-th entry is equal to x_i.

Graphs

In this paper, we consider multi-graph G, with edge set $E(G)$ and vertex set $V(G)$. When the graph is clear from context, we use the short-hand E for $E(G)$ and V for $V(G)$ with $|E| = m$ and $|V| = n$. Assume that each edge $e \in E$ has an implicit direction, used to define its edge-vertex incidence matrix $\mathbf{B} \in \mathbb{R}^{E \times V}$. Abusing notation slightly, we often write $e = (u, v) \in E$, where u and v are the tail and head of e respectively (note that technically multi-graph does not allow for edges to be specified by their endpoints). We say a flow $\mathbf{f} \in \mathbb{R}^E$ routes a demand $\mathbf{d} \in \mathbb{R}^V$ if $\mathbf{B}^\top \mathbf{f} = \mathbf{d}$. Given a k-commodity flow $\mathbf{F} \in \mathbb{R}^{E \times k}$, let $\mathbf{f}^e \in \mathbb{R}^k$ denote $\mathbf{F}$'s e-th row vector, a vector of k flow values through edge $e \in E$, and $\mathbf{f}_j \in \mathbb{R}^E$ denote $\mathbf{F}$'s j-th column vector, the flow vector of commodity $j \in [k]$; let $F_{i,j}$ denote the entry at the i-th row and j-th column of $\mathbf{F}$.

Model of Computation

In this paper, for problem instances encoded with z bits, all algorithms work in fixed-point arithmetic where words have $O(\log^{O(1)} z)$ bits, i.e., we prove that all numbers stored are in the interval $[\exp(-\log^{O(1)} z), \exp(\log^{O(1)} z)]$.

2.2 Convex Flow Solver

In this paper, we utilize the almost-linear time convex flow algorithm from [15]. Given a set of *computationally efficient* convex cost functions on edges $\{c_e(\cdot)\}_e$, the algorithm finds a single commodity flow $\mathbf{f}$ that routes the given demand $\mathbf{d}$ and minimizes $\sum_e c_e(f_e)$ up to a small $\exp(-\log^{O(1)} m)$ additive error.

▶ **Assumption 1** (Definition 10.1 and Assumption 10.2, [15]). *Let $K = \widetilde{O}(1)$ be a parameter fixed throughout. Given a convex cost function $c : \mathbb{R} \to \mathbb{R} \cup \{+\infty\}$, c is* computationally efficient *if there is a barrier function $\psi_c(f, y)$ defined on the domain $\mathcal{D}_c \stackrel{\text{def}}{=} \{(f, y) |\ c(f) \le y\}$ such that*

1. *The cost is quasi-polynomially bounded, i.e., $|c(f)| = O(m^K + |f|^K)$ for all $f \in \mathbb{R}$.*
2. *ψ_c is a ν-*self-concordant barrier *for some $\nu = O(1)$, that is, the following holds*

$$\begin{aligned} \psi_c(\mathbf{x}) &\to \infty, \text{ as } \mathbf{x} \text{ approaches the boundary of } \mathcal{D}_c, \\ \left|\nabla^3 \psi_c(\mathbf{x})[\mathbf{v}, \mathbf{v}, \mathbf{v}]\right| &\le 2\left(\nabla^2 \psi_c(\mathbf{x})[\mathbf{v}, \mathbf{v}]\right)^{3/2}, \forall \mathbf{x} \in \mathcal{D}_c, \mathbf{v} \in \mathbb{R}^2, \\ \langle \nabla \psi_c(\mathbf{x}), \mathbf{v}\rangle^2 &\le \nu \cdot \nabla^2 \psi_c(\mathbf{x})[\mathbf{v}, \mathbf{v}]. \end{aligned}$$

3. *The Hessian is quasi-polynomially bounded as long as the function value is $\widetilde{O}(1)$ bounded, i.e., for all points $|f|, |y| \le m^K$ with $\psi_c(f, y) \le \widetilde{O}(1)$, $\nabla^2 \psi_c(f, y) \preceq \exp(\log^{O(1)} m)\mathbf{I}$.*
4. *Both $\nabla \psi_c$ and $\nabla^2 \psi_c$ can be computed and accessed in $\widetilde{O}(1)$-time.*

▶ **Theorem 2** (Theorem 10.13, [15]). *Let G be a graph, and $\mathbf{d} \in \mathbb{R}^V$ be a demand vector. Given a collection of computationally efficient cost functions on edges $\{c_e(\cdot)\}_e$ and their barriers $\{\psi_e(\cdot)\}_e$ (Assumption 1), there is an algorithm that runs in $m^{1+o(1)}$ time and outputs a flow $\mathbf{f} \in \mathbb{R}^E$ that routes $\mathbf{d}$ and for any fixed constant $C > 0$,*

$$c(\mathbf{f}) \le \min_{\mathbf{B}^\top \mathbf{f}^* = \mathbf{d}} c(\mathbf{f}^*) + \exp(-\log^C m),$$

where $c(\mathbf{f}) \stackrel{\text{def}}{=} \sum_{e \in E} c_e(f_e)$.

2.3 Iterative Refinement

At a high level, the iterative refinement framework introduced by [2] approximates the Bregman Divergence of the function $|x|^p$ with something simpler.

▶ **Definition 3** (Bregman divergence). *Given a differentiable convex function $g(\cdot)$ and any two points $\mathbf{x}, \mathbf{y}$ in its domain, we define its* Bregman divergence *as*

$$D_g(\mathbf{y}, \mathbf{x}) \stackrel{\text{def}}{=} g(\mathbf{y}) - g(\mathbf{x}) - \langle \nabla g(\mathbf{x}), \boldsymbol{\Delta}\rangle = \int_0^1 \int_0^t \langle \boldsymbol{\Delta}, \nabla^2 g(\mathbf{x} + u\boldsymbol{\Delta})\boldsymbol{\Delta}\rangle \mathrm{d}u\mathrm{d}t,$$

where $\boldsymbol{\Delta} \stackrel{\text{def}}{=} \mathbf{y} - \mathbf{x}$.

For any $p > 1$, [2] shows that the Bregman Divergence of $|x + \delta|^p$ can be locally approximated by an error term $\gamma_p(\delta; |x|)$, which behaves quadratically in δ when $|\delta| \le |x|$ and as $|\delta|^p$ otherwise.

▶ **Definition 4** ([14]). *For $x \in \mathbb{R}$, $f > 0$, and $p > 1$, we define*

$$\gamma_p(x, f) \stackrel{\text{def}}{=} \begin{cases} \frac{p}{2} f^{p-2} x^2 & |x| \leq f, \\ |x|^p - \left(1 - \frac{p}{2}\right) f^p & |x| > f. \end{cases}$$

For $1 < q \leq 2$, [3] approximates the Bregman divergence of $|x|^q$ using the γ_q function we just defined.

▶ **Lemma 5** (Lemma 2.14, [3]). *For $q \in (1, 2]$ and $f, x \in \mathbb{R}$, it holds that*

$$\frac{q-1}{q 2^q} \cdot \gamma_q(x, |f|) \leq |f + x|^q - |f|^q - q|f|^{q-2} fx \leq 2^q \cdot \gamma_q(x, |f|).$$

For $p \geq 2$, the Bregman divergence of $|x|^p$ can be approximated by an x^2 and an $|x|^p$ term.

▶ **Lemma 6** (Lemma 2.5, [3]). *For $p \geq 2$ and $f, x \in \mathbb{R}$, it holds that*

$$\frac{p}{8}|f|^{p-2}x^2 + \frac{1}{2^{p+1}}|x|^p \leq |f + x|^p - |f|^p - p|f|^{p-2} fx \leq 2p^2|f|^{p-2}x^2 + p^p|x|^p.$$

Here we present some facts about γ_q functions that are helpful.

▶ **Lemma 7** (Lemma 3.3, [2]). *For $q \in (1, 2]$, $f, x \in \mathbb{R}$, and $t \geq 1$, it holds that*

$$t^q \cdot \gamma_q(x, |f|) \leq \gamma_q(tx, |f|) \leq t^2 \cdot \gamma_q(x, |f|).$$

3 Iterative Refinement Algorithm

In this section, we present the $\ell_{q,p}$-norm k-commodity flow algorithm based on iterative refinement and prove Theorem 1.

▶ **Theorem 1.** *Given any $1 < q \leq 2 \leq p$ with $p = \widetilde{O}(1)$ and $\frac{1}{q-1} = O(1)$ and an error parameter $\varepsilon > \exp(-\widetilde{O}(1))$, let* OPT *be the optimal value to Problem* (1)*. There is a randomized algorithm that computes a feasible flow $\mathbf{F}$ to Problem* (1) *such that*

$$\|\mathbf{WF}\|_{q,p}^{pq} \leq (1 + \varepsilon)\,\mathrm{OPT} + \varepsilon.$$

The algorithm runs in time

$$O\left(p^2 \cdot \left(\frac{1}{q-1}\right)^{\frac{1}{q-1}} m^{1+o(1)} \cdot k^2 \cdot \log \frac{1}{\varepsilon}\right)$$

with high probability.

In the rest of the paper, we use $\mathcal{E}(\mathbf{F})$ to denote the objective of Problem (1), that is, $\mathcal{E}(\mathbf{F}) = \|\mathbf{WF}\|_{q,p}^{pq}$.

Our iterative refinement algorithm is based on an approximation to the Bregman divergence of the objective $\|\mathbf{WF}\|_{q,p}^{pq}$. Since the objective can be decomposed into a summation of m seperate terms, i.e., $\sum_{e \in E} w_e^{pq} \|\mathbf{f}^e\|_q^{pq}$, we approximate the Bregman divergence for each edge separately. Consequently, we present the following lemma that approximates $\|\mathbf{f}^e + \mathbf{x}^e\|_q^{pq}$ for each edge e. This is the key technical lemma of this paper and its proof can be found in the full version.

▶ **Lemma 8.** *[$\ell_{q,p}$-Norm Iterative Refinement] Given $1 < q \le 2 \le p$, and any $\mathbf{f}, \mathbf{x} \in \mathbb{R}^k$, we have*

$$\|\mathbf{f}+\mathbf{x}\|_q^{pq} - \|\mathbf{f}\|_q^{pq} - pq\|\mathbf{f}\|_q^{q(p-1)}\langle|\mathbf{f}|^{q-2}\cdot\mathbf{f},\mathbf{x}\rangle \ge \frac{p(q-1)}{16}\|\mathbf{f}\|_q^{q(p-1)}\cdot\gamma_q(\mathbf{x},|\mathbf{f}|) + \frac{q-1}{pq-1}\frac{1}{2^{pq+2}}\|\mathbf{x}\|_{pq}^{pq}, \tag{4}$$

$$\|\mathbf{f}+\mathbf{x}\|_q^{pq} - \|\mathbf{f}\|_q^{pq} - pq\|\mathbf{f}\|_q^{q(p-1)}\langle|\mathbf{f}|^{q-2}\cdot\mathbf{f},\mathbf{x}\rangle \le \frac{7}{k}\|\mathbf{f}\|_q^{q(p-1)}\cdot\gamma_q(6kp\mathbf{x},|\mathbf{f}|) + \frac{3(6pk)^{pq}}{k}\|\mathbf{x}\|_{pq}^{pq}, \tag{5}$$

where we write $\gamma_q(\mathbf{x},\mathbf{y}) \stackrel{\text{def}}{=} \sum_j \gamma_q(x_j,y_j)$ for vectors $\mathbf{x},\mathbf{y} \in \mathbb{R}^k$.

Using Lemma 8, we can define the *residual function* $\mathcal{R}(\mathbf{x};\mathbf{f})$ that upper bounds the difference in the objective, i.e., $\mathcal{R}(\mathbf{x};\mathbf{f}) \ge \|\mathbf{f}+\mathbf{x}\|_q^{pq} - \|\mathbf{f}\|_q^{pq}$.

▶ **Definition 9** (Residual Problem)**.** *Given two vectors $\mathbf{f},\mathbf{x} \in \mathbb{R}^k$, we define $\mathcal{R}(\mathbf{x};\mathbf{f})$ as*

$$\mathcal{R}(\mathbf{x};\mathbf{f}) \stackrel{\text{def}}{=} pq\|\mathbf{f}\|_q^{q(p-1)}\langle|\mathbf{f}|^{q-2}\cdot\mathbf{f},\mathbf{x}\rangle + \frac{7}{k}\|\mathbf{f}\|_q^{q(p-1)}\cdot\gamma_q(6kp\mathbf{x},|\mathbf{f}|) + \frac{3(6pk)^{pq}}{k}\|\mathbf{x}\|_{pq}^{pq}.$$

In the setting of Problem 1, given a feasible flow $\mathbf{F} \in \mathbb{R}^{E\times k}$, we define the residual problem *w.r.t. $\mathbf{F}$ as follows*

$$\min_{\mathbf{X}\in\mathbb{R}^{E\times k}} \mathcal{R}(\mathbf{X};\mathbf{F}) \stackrel{\text{def}}{=} \sum_{e\in E} w_e^{pq}\mathcal{R}(\mathbf{x}^e;\mathbf{f}^e) \quad \text{s.t. } \mathbf{B}^\top\mathbf{X} = \mathbf{0}. \tag{6}$$

Note that in the residual problem, the objective is decomposable for each coordinate (e,j). The decomposability allows us to use the almost-linear time convex flow solver (Theorem 2) to solve the residual problem to high accuracy. The following lemma summarizes the algorithm and the proof is deferred to Section 4.

▶ **Lemma 10.** *Given any feasible flow $\mathbf{F} \in \mathbb{R}^{E\times k}$ to Problem (1), there is a randomized algorithm that runs in time $km^{1+o(1)}$ and outputs, for any $C > 0$, a k-commodity circulation $\mathbf{X}$ such that*

$$\mathcal{R}(\mathbf{X};\mathbf{F}) \le \min_{\mathbf{B}^\top\mathbf{X}^*=\mathbf{0}} \mathcal{R}(\mathbf{X}^*;\mathbf{F}) + \exp\left(-\log^C m\right).$$

Now, we are ready to prove Theorem 1 with the following Algorithm 1. The algorithm starts with some initial flow and runs in T iterations. Each iteration, the algorithm updates the flow $\mathbf{F}^{(t+1)} \leftarrow \mathbf{F}^{(t)} + \mathbf{X}^{(t)}$ with a near-optimal solution to the residual problem. In other words, given a current flow $\mathbf{F}$, the algorithm updates the flow by solving a simpler residual problem which is an upper bound to the change in objective value.

To analyze Algorithm 1 and prove Theorem 1, we first relate the value of $\mathcal{R}(\mathbf{X};\mathbf{F})$ to the change in objective value when updating $\mathbf{F}$ with $\mathbf{X}$.

▶ **Lemma 11.** *For any $\mathbf{F},\mathbf{X} \in \mathbb{R}^{E\times k}$, we have*

$$\mathcal{E}(\mathbf{F}+\mathbf{X}) - \mathcal{E}(\mathbf{F}) \le \mathcal{R}(\mathbf{X};\mathbf{F}) \text{ and} \tag{7}$$

$$\mathcal{E}(\mathbf{F}+\lambda\mathbf{X}) - \mathcal{E}(\mathbf{F}) \ge \lambda\mathcal{R}(\mathbf{X};\mathbf{F}), \text{ where } \lambda \stackrel{\text{def}}{=} O\left(kp\left(\frac{4032}{q-1}\right)^{\frac{1}{q-1}}\right). \tag{8}$$

Algorithm 1 High-Accuracy $\ell_{q,p}$-norm Multicommodity Flow.

procedure *LqpNormFlow*$(G, \mathbf{w}, \mathbf{D}, q, p)$
  Initialize the flow $\mathbf{F}^{(0)} \in \mathbb{R}^{E\times k}$ such that $\mathbf{B}^\top \mathbf{F}^{(0)} = \mathbf{D}$ via Lemma 13.
  $T \leftarrow O(p\lambda \log(m/\varepsilon))$, where λ comes from Lemma 11
  for $t = 0$ **to** $T-1$ **do**
    Solve the residual problem (6) w.r.t. $\mathbf{F}^{(t)}$ via Lemma 10 and obtain the solution $\mathbf{X}^{(t)}$.
    $\mathbf{F}^{(t+1)} \leftarrow \mathbf{F}^{(t)} + \mathbf{X}^{(t)}$
  end
  return $\mathbf{F}^{(T)}$

Proof. By the definition of $\mathcal{R}(\mathbf{x})$ and Lemma 8, the Inequality (7) holds trivially. We now focus on proving Inequality (8). In particular, we show that for any vector $\mathbf{f}, \mathbf{x} \in \mathbb{R}^k$, the following

$$\|\mathbf{f}+\lambda\mathbf{x}\|_q^{pq} - \|\mathbf{f}\|_q^{pq} \geq \lambda\mathcal{R}(\mathbf{f};\mathbf{x}). \tag{9}$$

Inequality (9) implies Inequality (8) by taking the summation over all edges.

Lemma 8 gives

$$\begin{aligned}&\|\mathbf{f}+\lambda\mathbf{x}\|_q^{pq} - \|\mathbf{f}\|_q^{pq}\\ &\geq pq\|\mathbf{f}\|_q^{(p-1)q}\langle|\mathbf{f}|^{q-2}\cdot\mathbf{f}, \lambda\mathbf{x}\rangle + \frac{p(q-1)}{16}\|\mathbf{f}\|_q^{(p-1)q}\cdot\gamma_q(\lambda\mathbf{x}, |\mathbf{f}|) + \frac{q-1}{pq-1}\frac{1}{2^{pq+2}}\|\lambda\mathbf{x}\|_{pq}^{pq}.\end{aligned}$$

In addition, we have

$$\lambda\mathcal{R}(\mathbf{x}) = \lambda pq\|\mathbf{f}\|_q^{(p-1)q}\langle|\mathbf{f}|^{q-2}\cdot\mathbf{f}, \mathbf{x}\rangle + \frac{7\lambda}{k}\|\mathbf{f}\|_q^{(p-1)q}\cdot\gamma_q(6kp\mathbf{x}, |\mathbf{f}|) + \frac{3\lambda(6pk)^{pq}}{k}\|\mathbf{x}\|_{pq}^{pq}.$$

In order to prove (9), it suffices to ensure that

$$\begin{aligned}\frac{p(q-1)}{16}\|\mathbf{f}\|_q^{(p-1)q}\cdot\gamma_q(\lambda\mathbf{x}, |\mathbf{f}|) &\geq \frac{7\lambda}{k}\|\mathbf{f}\|_q^{(p-1)q}\cdot\gamma_q(6kp\mathbf{x}, |\mathbf{f}|),\\ \frac{q-1}{pq-1}\frac{1}{2^{pq+2}}\|\lambda\mathbf{x}\|_{pq}^{pq} &\geq \frac{3\lambda(6pk)^{pq}}{k}\|\mathbf{x}\|_{pq}^{pq}.\end{aligned}$$

We can consider each entry separately and (9) follows if for any $f, x \in \mathbb{R}$, we have

$$\frac{p(q-1)}{16}\cdot\gamma_q(\lambda x, |f|) \geq \frac{7\lambda}{k}\cdot\gamma_q(6kpx, |f|), \tag{10}$$

$$\frac{q-1}{pq-1}\frac{1}{2^{pq+2}}|\lambda x|^{pq} \geq \frac{3\lambda(6pk)^{pq}}{k}|x|^{pq}. \tag{11}$$

Inequality (10) follows from Lemma 7

$$\frac{p(q-1)}{16}\cdot\gamma_q(\lambda x, |f|) \geq \frac{p(q-1)}{16}\left(\frac{\lambda}{6kp}\right)^q\cdot\gamma_q(6kpx, |f|) \geq \frac{7\lambda}{k}\cdot\gamma_q(6kpx, |f|),$$

if we set

$$\lambda \geq kp\left(\frac{4032}{q-1}\right)^{\frac{1}{q-1}} \geq 6kp. \tag{12}$$

For Inequality (11) to hold, we need to set

$$\lambda \geq 1728k \left(\frac{pq-1}{q-1}\right)^{\frac{1}{pq-1}} p^{\frac{pq}{pq-1}}. \tag{13}$$

Observe that

$$\left(\frac{pq-1}{q-1}\right)^{\frac{1}{pq-1}} = (pq-1)^{\frac{1}{pq-1}} \left(\frac{1}{q-1}\right)^{\frac{1}{pq-1}} \leq e \left(\frac{1}{q-1}\right)^{\frac{1}{q-1}}, \text{ and}$$
$$p^{\frac{pq}{pq-1}} = p^{1+\frac{1}{pq-1}} \leq p^{1+\frac{1}{p-1}} \leq 2p.$$

Combining the observation along with (12) and (13), Inequality (9) follows if we set

$$\lambda = O\left(kp\left(\frac{4032}{q-1}\right)^{\frac{1}{q-1}}\right).$$

This completes the proof. ◀

Using Lemma 11, we now show that each iteration decreases the objective exponentially. This is summarized by the following lemma

▶ **Lemma 12** (Convergence Rate). *Let $\mathbf{F}^*$ be the optimal solution to Problem (1). At any iteration t of Algorithm 1, we have*

$$\mathcal{E}(\mathbf{F}^{(t+1)}) - \mathcal{E}(\mathbf{F}^*) \leq \left(1 - \frac{1}{\lambda}\right)\left(\mathcal{E}(\mathbf{F}^{(t)}) - \mathcal{E}(\mathbf{F}^*)\right) + \exp(-\log^C m).$$

Proof. Recall that $\mathbf{F}^{(t+1)} = \mathbf{F}^{(t)} + \mathbf{X}^{(t)}$ and $\mathbf{X}^{(t)}$ is a high-accuracy solution to the residual problem w.r.t. $\mathbf{F}^{(t)}$. Lemma 11 and the optimality of $\mathbf{X}^{(t)}$ yields

$$\begin{aligned}\mathcal{E}(\mathbf{F}^{(t+1)}) - \mathcal{E}(\mathbf{F}^{(t)}) &\leq \mathcal{R}(\mathbf{X}^{(t)}; \mathbf{F}^{(t)}) \\ &\leq \mathcal{R}\left(\lambda^{-1}\left(\mathbf{F}^* - \mathbf{F}^{(t)}\right); \mathbf{F}^{(t)}\right) + \exp(-\log^C m) \\ &\leq \frac{1}{\lambda}\left(\mathcal{E}(\mathbf{F}^*) - \mathcal{E}(\mathbf{F}^{(t)})\right) + \exp(-\log^C m).\end{aligned} \tag{14}$$

We can conclude the lemma as follows:

$$\begin{aligned}\mathcal{E}(\mathbf{F}^{(t+1)}) - \mathcal{E}(\mathbf{F}^*) &= \mathcal{E}(\mathbf{F}^{(t+1)}) - \mathcal{E}(\mathbf{F}^{(t)}) + \mathcal{E}(\mathbf{F}^{(t)}) - \mathcal{E}(\mathbf{F}^*) \\ &\leq \frac{1}{\lambda}\left(\mathcal{E}(\mathbf{F}^*) - \mathcal{E}(\mathbf{F}^{(t)})\right) + \exp(-\log^C m) + \mathcal{E}(\mathbf{F}^{(t)}) - \mathcal{E}(\mathbf{F}^*) \\ &= \left(1 - \frac{1}{\lambda}\right)\left(\mathcal{E}(\mathbf{F}^{(t)}) - \mathcal{E}(\mathbf{F}^*)\right) + \exp(-\log^C m).\end{aligned}$$ ◀

Then, we show how to find an initial solution efficiently.

▶ **Lemma 13** (Initial Flow). *Given an instance of Problem (1), there is an algorithm that runs in $\widetilde{O}(pmk)$ time and finds a feasible flow $\mathbf{F}^{(0)} \in \mathbb{R}^{E\times k}$ such that $\mathcal{E}(\mathbf{F}^{(0)}) \leq 4m^{p+1}\mathcal{E}(\mathbf{F}^*)$, where $\mathbf{F}^*$ is the optimal solution.*

Proof. Let flow $\mathbf{F}^{(0)}$ be a $(1+1/pq)$-approximate maximum concurrent flow on $(G, \mathbf{w})$ that routes the demand $\mathbf{D}$. That is, $\mathbf{F}^{(0)}$ is a $(1+1/pq)$-approximate solution to the following problem

$$\min_{\mathbf{B}^\top \mathbf{F} = \mathbf{D}} \max_{e\in E} w_e \sum_{j=1}^{k} |F_{ej}|.$$

Moreover, $\mathbf{F}^{(0)}$ can be computed in $\widetilde{O}(pmk)$ time using the algorithm from [66]. Set $\mathbf{F} = \mathbf{F}^{(0)}$ and let $\mathbf{G} = \mathbf{F}^*$ be the optimal solution to Problem (1). We can view $\mathbf{G}$ as a collection of k single commodity flows and the approximation guarantee of $\mathbf{F}$ yields

$$\max_{e\in E} w_e \left\|\mathbf{f}^e\right\|_1 \leq \left(1+\frac{1}{pq}\right) \max_{e\in E} w_e \left\|\mathbf{g}^e\right\|_1 . \tag{15}$$

We now analyze the approximation ratio of $\mathbf{F}$. Recall the fact that for any vector $\mathbf{x} \in \mathbb{R}^k$, we have $\|\mathbf{x}\|_1 \leq m^{1-1/q} \|\mathbf{x}\|_q$. Combining the observation with Inequality (15), we have

$$\begin{aligned}
\mathcal{E}(\mathbf{G}) \leq \mathcal{E}(\mathbf{F}) &= \sum_{e\in E} w_e^{pq} \left\|\mathbf{f}^e\right\|_q^{pq} \\
&\leq \sum_{e\in E} w_e^{pq} \left\|\mathbf{f}^e\right\|_1^{pq} \leq m \left(\max_{e\in E} w_e \left\|\mathbf{f}^e\right\|_1\right)^{pq} \\
&\leq m\left(1+\frac{1}{pq}\right)^{pq} \left(\max_{e\in E} w_e \left\|\mathbf{g}^e\right\|_1\right)^{pq} \\
&\leq 4m \sum_{e\in E} w_e^{pq} \left\|\mathbf{g}^e\right\|_1^{pq} \leq 4m^{1+pq-p} \sum_{e\in E} w_e^{pq} \left\|\mathbf{g}^e\right\|_q^{pq} \\
&\leq 4m^{p+1}\mathcal{E}(\mathbf{G}).
\end{aligned}$$

◀

We use Lemma 12 to analyze the correctness of the algorithm and prove Theorem 1.

Proof of Theorem 1. After $T = O(p\lambda \log(m/\varepsilon))$ iterations, we use Lemma 12 to bound the objective of the final flow $\mathbf{F}^{(T)}$ output by Algorithm 1 as follows:

$$\begin{aligned}
\mathcal{E}(\mathbf{F}^{(T)}) - \mathcal{E}(\mathbf{F}^*) &\leq \left(1-\frac{1}{\lambda}\right)^T \left(\mathcal{E}(\mathbf{F}^{(0)}) - \mathcal{E}(\mathbf{F}^*)\right) + \exp(-\log^C m) \cdot \sum_{t=0}^{T-1} \left(1-\frac{1}{\lambda}\right)^t \\
&\leq \exp\left(-\frac{T}{\lambda}\right) \left(\mathcal{E}(\mathbf{F}^{(0)}) - \mathcal{E}(\mathbf{F}^*)\right) + \lambda \cdot \exp(-\log^C m) \\
&\leq \varepsilon \mathcal{E}(\mathbf{F}^*) + \varepsilon,
\end{aligned}$$

where the last inequality comes from $\mathcal{E}(\mathbf{F}^{(0)}) - \mathcal{E}(\mathbf{F}^*) \leq 4m^{p+1}\mathcal{E}(\mathbf{F}^*)$ and the sufficiently small constant C in Lemma 10. Rearrangement yields that $\mathcal{E}(\mathbf{F}^{(T)})$ is at most $(1+\varepsilon)\mathcal{E}(\mathbf{F}^*) + \varepsilon$.

We now analyze the runtime. Initialization of $\mathbf{F}^{(0)}$ takes $\widetilde{O}(pmk)$ time by Lemma 13. Each iteration takes $km^{1+o(1)}$ time due to Lemma 10 and there are $T = \widetilde{O}(p\lambda \log(1/\varepsilon))$ iterations. Then the runtime bound follows. ◀

4 Solving the Residual Problem

In this section, we prove Lemma 10.

▶ **Lemma 10.** *Given any feasible flow $\mathbf{F} \in \mathbb{R}^{E\times k}$ to Problem (1), there is a randomized algorithm that runs in time $km^{1+o(1)}$ and outputs, for any $C > 0$, a k-commodity circulation $\mathbf{X}$ such that*

$$\mathcal{R}(\mathbf{X};\mathbf{F}) \leq \min_{\mathbf{B}^\top \mathbf{X}^* = \mathbf{0}} \mathcal{R}(\mathbf{X}^*;\mathbf{F}) + \exp\left(-\log^C m\right).$$

At a high level, we will show that solving the residual problem is equivalent to solving k instances of single commodity convex flow problems. Each of them can be solved in $m^{1+o(1)}$ time via Theorem 2. In particular, we need to construct computationally efficient barriers for the edge cost functions.

Proof of Lemma 10. Recall the residual problem

$$\begin{aligned} &\min_{\mathbf{X}\in\mathbb{R}^{E\times k}} \mathcal{R}(\mathbf{X};\mathbf{F}) \\ &\quad \text{s.t. } \mathbf{B}^\top\mathbf{X} = \mathbf{0}. \end{aligned} \tag{16}$$

From Definition 9, we know

$$\mathcal{R}(\mathbf{X};\mathbf{F}) = \sum_{e\in E}\sum_{j=1}^{k} w_e^{pq}\cdot\left(pq\|\mathbf{f}^e\|_q^{q(p-1)}|F_{ej}|^{q-2}F_{ej}X_{ej} + \frac{7}{k}\|\mathbf{f}^e\|_q^{q(p-1)}\gamma_q(6kpX_{ej},|F_{ej}|) + \frac{3(6pk)^{pq}}{k}|X_{ej}|^{pq}\right).$$

For each commodity $j\in[k]$, we write $\mathbf{x}_j$ and $\mathbf{f}_j$ to be the j-th column of $\mathbf{X}$ and $\mathbf{F}$ respectively, i.e., the flow that routes the commodity j. The constraint $\mathbf{B}^\top\mathbf{X}=\mathbf{0}$ is equivalent to $\mathbf{B}^\top\mathbf{x}_j=\mathbf{0}$ for each j. Thus, (16) is equivalent to solving, for each commodity j, the following single commodity flow problem:

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbb{R}^{E}} \sum_{e\in E} w_e^{pq}\left(pq\|\mathbf{f}^e\|_q^{q(p-1)}|F_{ej}|^{q-2}F_{ej}x_e + \frac{7}{k}\|\mathbf{f}^e\|_q^{q(p-1)}\gamma_q(6kpx_e,|F_{ej}|) + \frac{3(6pk)^{pq}}{k}|x_e|^{pq}\right) \\ &\quad \text{s.t. } \mathbf{B}^\top\mathbf{x} = \mathbf{0}. \end{aligned} \tag{17}$$

To use Theorem 2 to solve (17), we need to show that the objective is a sum of convex edge costs with efficient barriers. For each edge e and commodity j, we define the cost $c_{ej}(x)$ to be

$$\begin{aligned} &c_{ej}(x) = A_{ej}x + B_{ej}\gamma_q(6kpx,|F_{ej}|) + C_{ej}|x|^{pq}, \text{ where} \\ &\quad A_{ej} \stackrel{\text{def}}{=} w_e^{pq}\cdot pq\|\mathbf{f}^e\|_q^{q(p-1)}|F_{ej}|^{q-2}F_{ej}, \\ &\quad B_{ej} \stackrel{\text{def}}{=} w_e^{pq}\cdot\frac{7}{k}\|\mathbf{f}^e\|_q^{q(p-1)}, \\ &\quad C_{ej} \stackrel{\text{def}}{=} w_e^{pq}\cdot\frac{3(6pk)^{pq}}{k}. \end{aligned}$$

Clearly $c_{ej}(x)$ is a convex function and the objective of (17) is exactly $\sum_e c_{ej}(x_e)$.

We now present computationally efficient barriers for each $c_{ej}(x)$ and show that $c_{ej}(x)$ satisfies Assumption 1. For clarity, we ignore both subscripts e and j and use f to denote $|F_{ej}|$. That is, we will show that the following function satisfies Assumption 1 for any positive $A,B,C,f>0$,

$$c(x) = A\cdot x + B\cdot\gamma_q(6kpx;f) + C\cdot|x|^{pq}.$$

Item 1 holds because $p=\widetilde{O}(1)$.

We now show Item 2 using Theorem 9.1.1 from [55]. It says that the function $\psi_c(x,y) = -\ln(y-c(x))$ is a 1-self-concordance barrier for the epigraph $\mathcal{D}_c \stackrel{\text{def}}{=} \{(x,y)|c(x)\le y\}$ if there is some $\beta>0$ such that $|c'''(x)|\le 3\beta|c''(x)/x|$ holds for all $x\in\mathbb{R}$. The derivatives of $c(x)$ have different forms depending on the value of x due to the γ_q function.

- If $|6kpx|\le f$, we know $\gamma_q(6kpx;f) = \frac{q}{2}f^{q-2}(6kp)^2x^2$ and

$$\begin{aligned} c'(x) &= A + Bqf^{q-2}(6kp)^2x + Cpq|x|^{pq-1}\mathrm{sgn}(x), \\ c''(x) &= Bqf^{q-2}(6kp)^2 + Cpq(pq-1)|x|^{pq-2}, \\ c'''(x) &= Cpq(pq-1)(pq-2)|x|^{pq-3}\mathrm{sgn}(x). \end{aligned}$$

In this case, we have

$$\left|\frac{c''(x)}{x}\right| = \frac{Bqf^{q-2}(6kp)^2}{|x|} + Cpq(pq-1)|x|^{pq-3} \geq \frac{1}{pq-2}|c'''(x)|.$$

It suffices to set $\beta \geq (pq-2)/3$.

- Otherwise, $|6kpx| > f$. We know $\gamma_q(6kpx; f) = |6kpx|^q - (1-\frac{q}{2})f^q$, and

$$\begin{aligned}
c'(x) &= A + B(6kp)^q q|x|^{q-1}\mathrm{sgn}(x) + Cpq|x|^{pq-1}\mathrm{sgn}(x),\\
c''(x) &= B(6kp)^q q(q-1)|x|^{q-2} + Cpq(pq-1)|x|^{pq-2},\\
c'''(x) &= B(6kp)^q q(q-1)(q-2)|x|^{q-3}\mathrm{sgn}(x) + Cpq(pq-1)(pq-2)|x|^{pq-3}\mathrm{sgn}(x).
\end{aligned}$$

In this case, notice that $q-2<0$ and we have

$$\begin{aligned}
\left|\frac{c''(x)}{x}\right| &= B(6kp)^q q(q-1)|x|^{q-3} + Cpq(pq-1)|x|^{pq-3}, \text{ and}\\
|c'''(x)| &= \left|B(6kp)^q q(q-1)(q-2)|x|^{q-3} + Cpq(pq-1)(pq-2)|x|^{pq-3}\right|\\
&\leq B(6kp)^q q(q-1)(2-q)|x|^{q-3} + Cpq(pq-1)(pq-2)|x|^{pq-3}.
\end{aligned}$$

Setting $\beta \geq \max\{2-q, pq-2\}/3$ yields that

$$3\beta\left|\frac{c''(x)}{x}\right| \geq B(6kp)^q q(q-1)(2-q)|x|^{q-3} + Cpq(pq-1)(pq-2)|x|^{pq-3} \geq |c'''(x)|.$$

We conclude Item 2 with the barrier function $\psi_c(y,x) = -\ln(y-c(x))$. Item 3 follows directly from Item 1, and Item 4 follows using the explicit formula for $c(x), c'(x)$, and $c''(x)$.

Now we can apply Theorem 2 to compute, for each j, a flow $\mathbf{x}_j$ in time $m^{1+o(1)}$ such that

$$c(\mathbf{x}_j) \leq \min_{\mathbf{B}^\top \mathbf{x}^* = \mathbf{0}} c(\mathbf{x}^*) + \exp(-\log^C m).$$

Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k]$, then we have $\mathbf{B}^\top \mathbf{X} = \mathbf{0}$ and $\mathbf{X}$ is optimal to (16) up to a $k\exp(-\log^C m)$-additive error, i.e.,

$$\mathcal{R}(\mathbf{X}; \mathbf{F}) \leq \min_{\mathbf{B}^\top \mathbf{X}^* = \mathbf{0}} \mathcal{R}(\mathbf{X}^*; \mathbf{F}) + k\exp(-\log^C m).$$

The total runtime is $k \cdot m^{1+o(1)}$ since we compute $\mathbf{x}_j$ for each $j \in [k]$. ◀

References

1. Deeksha Adil, Brian Bullins, Rasmus Kyng, and Sushant Sachdeva. Almost-linear-time weighted ℓ_p-norm solvers in slightly dense graphs via sparsification. *arXiv preprint*, 2021. arXiv:2102.06977.
2. Deeksha Adil, Rasmus Kyng, Richard Peng, and Sushant Sachdeva. Iterative refinement for ℓ_p-norm regression. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1405–1424, 2019.
3. Deeksha Adil, Rasmus Kyng, Richard Peng, and Sushant Sachdeva. Fast algorithms for ℓ_p-regression. *arXiv preprint*, 2022. arXiv:2211.03963.
4. Deeksha Adil, Richard Peng, and Sushant Sachdeva. Fast, provably convergent IRLS algorithm for p-norm linear regression. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

5 Deeksha Adil and Sushant Sachdeva. Faster p-norm minimizing flows, via smoothed q-norm problems. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 892–910, 2020.

6 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice-Hall, Inc., USA, 1993.

7 Kyriakos Axiotis, Aleksander Mądry, and Adrian Vladu. Circulation control for faster minimum cost flow in unit-capacity graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 93–104, 2020.

8 Kyriakos Axiotis, Aleksander Mądry, and Adrian Vladu. Faster sparse minimum cost flow by electrical flow localization. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 528–539, 2022.

9 Cynthia Barnhart, Niranjan Krishnan, and Pamela H Vance. Multicommodity flow problems. *Encyclopedia of Optimization*, 14:2354–2362, 2009.

10 Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental SSSP and approximate min-cost flow in almost-linear time. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1000–1008, 2022.

11 Daniel Bienstock and Garud Iyengar. Solving fractional packing problems in $\widetilde{O}(1/\varepsilon)$ iterations. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing (STOC)*, pages 146–155, 2004.

12 Jan van den Brand, Yin-Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 919–930, 2020.

13 Jan van den Brand and Daniel Zhang. Faster high accuracy multi-commodity flow from single-commodity techniques. *arXiv preprint*, 2023. `arXiv:2304.12992`.

14 Sébastien Bubeck, Michael B Cohen, Yin Tat Lee, and Yuanzhi Li. An homotopy method for lp regression provably beyond self-concordance and in input-sparsity time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1130–1137, 2018.

15 Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623, 2022.

16 Li Chen, Richard Peng, and Di Wang. 2-norm flow diffusion in near-linear time. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 540–549, 2022.

17 Paul Christiano, Jonathan A Kelner, Aleksander Madry, Daniel A Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the forty-third annual ACM symposium on Theory of computing (STOC)*, pages 273–282, 2011.

18 Kenneth Clarkson, Ruosong Wang, and David Woodruff. Dimensionality reduction for tukey regression. In *International Conference on Machine Learning (ICML)*, pages 1262–1271, 2019.

19 Kenneth L Clarkson, Petros Drineas, Malik Magdon-Ismail, Michael W Mahoney, Xiangrui Meng, and David P Woodruff. The fast cauchy transform and faster robust linear regression. *SIAM Journal on Computing*, 45(3):763–810, 2016.

20 Kenneth L Clarkson and David P Woodruff. Low-rank approximation and regression in input sparsity time. *Journal of the ACM (JACM)*, 63(6):1–45, 2017.

21 Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 938–942, 2019.

22 Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *Journal of the ACM (JACM)*, 68(1):1–39, 2021.

23 Michael B Cohen, Aleksander Mądry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in $\widetilde{O}(m^{10/7} \log w)$ time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 752–771, 2017.

24 Samuel I Daitch and Daniel A Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the fortieth annual ACM symposium on Theory of computing (STOC)*, pages 451–460, 2008.

25 Anirban Dasgupta, Petros Drineas, Boulos Harb, Ravi Kumar, and Michael W Mahoney. Sampling algorithms and coresets for ℓ_p regression. *SIAM Journal on Computing*, 38(5):2060–2078, 2009.

26 Ming Ding, Rasmus Kyng, and Peng Zhang. Two-commodity flow is equivalent to linear programming under nearly-linear time reductions. In *49th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 54:1–54:19, 2022.

27 Lisa K Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics*, 13(4):505–520, 2000.

28 Sebastian Forster, Gramoz Goranci, Yang P Liu, Richard Peng, Xiaorui Sun, and Mingquan Ye. Minor sparsifiers and the distributed laplacian paradigm. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 989–999, 2022.

29 Yu Gao, Yang P Liu, and Richard Peng. Fully dynamic electrical flows: Sparse maxflow faster than goldberg-rao. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 516–527, 2022.

30 Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007.

31 Mehrdad Ghadiri, Richard Peng, and Santosh S Vempala. Faster p-norm regression using sparsity. *arXiv preprint arXiv:2109.11537*, 2021.

32 Andrew V Goldberg. A natural randomization strategy for multicommodity flow and related algorithms. *Information Processing Letters*, 42(5):249–256, 1992.

33 Michael D Grigoriadis and Leonid G Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. *SIAM Journal on Optimization*, 4(1):86–107, 1994.

34 Michael D Grigoriadis and Leonid G Khachiyan. Approximate minimum-cost multicommodity flows in $\widetilde{O}(\varepsilon^{-2}knm)$ time. *Mathematical Programming*, 75(3):477–482, 1996.

35 Alon Itai. Two-commodity flow. *Journal of the ACM (JACM)*, 25(4):596–611, 1978.

36 Sanjiv Kapoor and Pravin M Vaidya. Speeding up karmarkar's algorithm for multicommodity flows. *Mathematical programming*, 73:111–127, 1996.

37 George Karakostas. Faster approximation schemes for fractional multicommodity flow problems. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 166–173, 2002.

38 David Karger and Serge Plotkin. Adding multiple cost constraints to combinatorial optimization problems, with applications to multicommodity flows. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing (STOC)*, pages 18–25, 1995.

39 Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing (STOC)*, pages 302–311, 1984.

40 Tarun Kathuria, Yang P. Liu, and Aaron Sidford. Unit capacity maxflow in almost $O(m^{4/3})$ time. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 119–130, 2020.

41 Jonathan A Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 217–226, 2014.

42 Jonathan A Kelner, Gary L Miller, and Richard Peng. Faster approximate multicommodity flow using quadratically coupled flows. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing (STOC)*, pages 1–18, 2012.

43 Jeff L Kennington. A survey of linear cost multicommodity network flows. *Operations Research*, 26(2):209–236, 1978.

44 Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

45 Philip Klein, Serge Plotkin, Clifford Stein, and Eva Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM Journal on Computing*, 23(3):466–487, 1994.

46 Rasmus Kyng, Richard Peng, Sushant Sachdeva, and Di Wang. Flows in almost linear time via adaptive preconditioning. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 902–913, 2019.

47 Yin Tat Lee and Aaron Sidford. Solving linear programs with sqrt (rank) linear system solves. *arXiv preprint*, 2019. `arXiv:1910.08033`.

48 T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, and S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. *Journal of Computer and System Sciences*, 50(2):228–243, 1995.

49 Tom Leighton, Clifford Stein, Fillia Makedon, Éva Tardos, Serge Plotkin, and Spyros Tragoudas. Fast approximation algorithms for multicommodity flow problems. In *Proceedings of the twenty-third annual ACM symposium on Theory of Computing (STOC)*, pages 101–111, 1991.

50 Jason Li. Faster parallel algorithm for approximate shortest path. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 308–321, 2020.

51 Aleksander Madry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the forty-second ACM symposium on Theory of computing (STOC)*, pages 121–130, 2010.

52 Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 253–262, 2013.

53 Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 593–602, 2016.

54 Xiangrui Meng and Michael Mahoney. Robust regression on mapreduce. In *International Conference on Machine Learning (ICML)*, pages 888–896, 2013.

55 Arkadi Nemirovski. Interior point polynomial time methods in convex programming. *Lecture notes*, 42(16):3215–3224, 2004.

56 Yurii Nesterov. Fast gradient methods for network flow problems. In *Proceeding of the 20th International Symposium of Mathematical Programming (ISMP)*, 2009.

57 Adamou Ouorou, Philippe Mahey, and J-Ph Vial. A survey of algorithms for convex multicommodity flow problems. *Management science*, 46(1):126–147, 2000.

58 Richard Peng. Approximate undirected maximum flows in $O(m\text{poly}\log(n))$ time. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 1862–1867, 2016.

59 Richard Peng and Santosh Vempala. Solving sparse linear systems faster than matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 504–521, 2021.

60 Serge A Plotkin, David B Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301, 1995.

61 Tomasz Radzik. Fast deterministic approximation for the multicommodity flow problem. *Mathematical Programming*, 78:43–58, 1996.

62 James Renegar. A polynomial-time algorithm, based on newton's method, for linear programming. *Mathematical programming*, 40(1-3):59–93, 1988.

63 Bruce Rothschild and Andrew Whinston. Feasibility of two commodity network flows. *Operations Research*, 14(6):1121–1129, 1966.

64 Farhad Shahrokhi and David W Matula. The maximum concurrent flow problem. *Journal of the ACM (JACM)*, 37(2):318–334, 1990.

65 Jonah Sherman. Nearly maximum flows in nearly linear time. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 263–269, 2013.

66 Jonah Sherman. Area-convexity, ℓ_∞ regularization, and undirected multicommodity flow. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 452–460, 2017.

67 David B Shmoys. Cut problems and their application to divide-and-conquer. *Approximation algorithms for NP-hard problems*, pages 192–235, 1997.

68 D. Spielman and S. Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Applications*, 35(3):835–885, 2014. Available at http://arxiv.org/abs/cs/0607105.

69 Jan van den Brand, Yu Gao, Arun Jambulapati, Yin Tat Lee, Yang P Liu, Richard Peng, and Aaron Sidford. Faster maxflow via improved dynamic spectral vertex sparsifiers. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 543–556, 2022.

70 Jan van den Brand, Yin Tat Lee, Yang P Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, MDPs, and ℓ_1-regression in nearly linear time for dense instances. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 859–869, 2021.

71 I-Lin Wang. Multicommodity network flows: A survey, part i: Applications and formulations. *International Journal of Operations Research*, 15(4):145–153, 2018.

72 David Woodruff and Qin Zhang. Subspace embeddings and ℓ_p-regression using exponential random variables. In *Conference on Learning Theory (COLT)*, pages 546–567, 2013.

73 Neal E. Young. Randomized rounding without solving the linear program. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 170–178, 1995.

74 Goran Zuzic, Gramoz Goranci, Mingquan Ye, Bernhard Haeupler, and Xiaorui Sun. Universally-optimal distributed shortest paths and transshipment via graph-based ℓ_1-oblivious routing. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2549–2579, 2022.

On the Streaming Complexity of Expander Decomposition

Yu Chen
EPFL, Lausanne, Switzerland

Michael Kapralov
EPFL, Lausanne, Switzerland

Mikhail Makarov
EPFL, Lausanne, Switzerland

Davide Mazzali
EPFL, Lausanne, Switzerland

Abstract

In this paper we study the problem of finding (ϵ, ϕ)-expander decompositions of a graph in the streaming model, in particular for dynamic streams of edge insertions and deletions. The goal is to partition the vertex set so that every component induces a ϕ-expander, while the number of inter-cluster edges is only an ϵ fraction of the total volume. It was recently shown that there exists a simple algorithm to construct a $(O(\phi \log n), \phi)$-expander decomposition of an n-vertex graph using $\widetilde{O}(n/\phi^2)$ bits of space [Filtser, Kapralov, Makarov, ITCS'23]. This result calls for understanding the extent to which a dependence in space on the sparsity parameter ϕ is inherent. We move towards answering this question on two fronts.

We prove that a $(O(\phi \log n), \phi)$-expander decomposition can be found using $\widetilde{O}(n)$ space, for every ϕ. At the core of our result is the first streaming algorithm for computing boundary-linked expander decompositions, a recently introduced strengthening of the classical notion [Goranci et al., SODA'21]. The key advantage is that a classical sparsifier [Fung et al., STOC'11], with size independent of ϕ, preserves the cuts inside the clusters of a boundary-linked expander decomposition within a multiplicative error.

Notable algorithmic applications use sequences of expander decompositions, in particular one often repeatedly computes a decomposition of the subgraph induced by the inter-cluster edges (e.g., the seminal work of Spielman and Teng on spectral sparsifiers [Spielman, Teng, SIAM Journal of Computing 40(4)], or the recent maximum flow breakthrough [Chen et al., FOCS'22], among others). We prove that any streaming algorithm that computes a sequence of $(O(\phi \log n), \phi)$-expander decompositions requires $\widetilde{\Omega}(n/\phi)$ bits of space, even in insertion only streams.

2012 ACM Subject Classification Theory of computation → Streaming models; Theory of computation → Sparsification and spanners; Theory of computation → Sketching and sampling; Theory of computation → Lower bounds and information complexity

Keywords and phrases Graph Sketching, Dynamic Streaming, Expander Decomposition

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.46

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2404.16701` [12]

1 Introduction

Expander graphs are known to represent a class of easier instances for many problems. Therefore, breaking down the input into disjoint expanders can allow to conveniently solve the task on each of them separately, before combining the partial results into a global solution. This approach is enabled by (ϵ, ϕ)-expander decompositions (for short, (ϵ, ϕ)-

ED). For an undirected graph $G = (V, E)$, this is a partition $\mathcal{U}$ of the vertex set V, such that there are at most $\epsilon|E|$ inter-cluster edges, while every cluster $U \in \mathcal{U}$ induces a ϕ-expander. The list of successful applications of this framework is long, including Laplacian system solvers [31], deterministic algorithms for minimum cut [28], graph and hyper-graph sparsification [2, 14, 23, 32], dynamic algorithms for cut and connectivity problems [18, 19], fast max flow algorithms [11], distributed triangle enumeration [9], polynomial time algorithms for semirandom planted CSPs [20], and many more.

We refer to ϵ and ϕ as the *sparsity* parameters: the former controls how sparsely connected the clusters need to be, and the latter determines how expanding (i.e. non-sparse) the cuts within clusters are. The reason for using the same term for both is that they are in fact very closely related. One can show that any n-vertex graph has an (ϵ, ϕ)-ED with $\epsilon = O(\phi \log n)$. To see this, consider the following constructive argument: if the graph has no ϕ-sparse cut then $\{V\}$ is a valid ED of G, otherwise recurse on the two sides $(S, V \setminus S)$ of a ϕ-sparse cut and union the results to get an ED for G. Every cluster in this decomposition is then an expander, and a charging argument allows to bound the number of inter-cluster edges by $O(\phi|E| \log n)$ [30]. One can also observe that no better asymptotic trade-off between ϵ and ϕ is possible in general [6]. Therefore, this sets the benchmark for algorithmic constructions of EDs.

At a high level, many ED algorithms follow the approach suggested by the existential argument. As a naive implementation would take exponential time, the crux often lies in efficient algorithms that either certify that a large portion of the input is an expander, or find a balanced sparse cut. This would result in a small depth recursion, thanks to balancedness, where each level requires little computational resources. There are several successful examples of this approach. In the sequential setting, a recent algorithm constructs a $(O(\phi \log^3 n), \phi)$-ED in $\widetilde{O}(|E|)$ time [29], based on the previous best algorithm which runs in $\widetilde{O}(|E|/\phi)$ time [30]. There is also a deterministic counterpart, which outputs a $(\phi \cdot n^{o(1)}, \phi)$-ED in almost linear time [15]. For the CONGEST model of distributed computing, it is possible to obtain, for instance, a $(\phi^{1/\sqrt{\log n}} \cdot n^{o(1)}, \phi)$-ED in $n^{o(1)}/\phi$ rounds [9]. It is also possible to maintain a $(\phi \cdot n^{o(1)}, \phi)$-ED for a graph undergoing edge updates in $n^{o(1)}/\phi^2$ amortized update time [19, 21].

1.1 Previous Work

In the streaming setting, the problem of finding EDs was open until the recent work of [16]. They obtain a dynamic stream algorithm which outputs a $(O(\phi \log n), \phi)$-ED and takes $\widetilde{O}(n/\phi^2)$ space. While being optimal in the quality of the decomposition, decoding the sketch to actually output an ED takes exponential time. The authors also give a polynomial time version: one can produce a $(\phi \cdot n^{o(1)}, \phi)$-ED using space $\widetilde{O}(n/\phi^2) + n^{1+o(1)}/\phi^{1-o(1)}$, with a post-processing that takes $\text{poly}(n)$ time (where the $o(1)$'s can be tuned, allowing for a quality-space trade-off).

These streaming algorithms also adopt the recursive approach based on finding balanced ϕ-sparse cuts, but the streaming model poses a challenge that we now illustrate. Sparsification for graph streams has been extensively studied [3, 4, 5, 24, 25], so a natural attempt would consist of maintaining a cut sparsifier as the stream comes and later run the recursive partitioning on it. However, it must be noted that these cuts are to be found in the subgraphs induced by the two sides of a previously made cut. This is not a problem in a classical computational setting, but it actually constitutes the main obstacle for the streaming model: at sketching time, the algorithm does not know which subgraphs it will need to access. Unfortunately, it is impossible to preserve cut sizes in arbitrary subgraphs with multiplicative

precision. The natural work-around is to introduce an additive error term: in [16], the authors introduce the concept of power-cut sparsifiers. For any given partition $\mathcal{U}$ of V, these sparsifiers preserve the cuts $|E(S, U \setminus S)|$ of each cluster $U \in \mathcal{U}$ in the partition up to an error of $\delta \cdot |E(S, U \setminus S)| + \psi \cdot \mathrm{vol}(S)$. They further show that maintaining $\widetilde{O}(n/\delta\psi)$ random linear measurements of the incidence matrix of the input graph is enough to obtain such sparsifiers. One can see that setting $\delta \ll 1$ and $\psi \approx \phi$ preserves the sparsity of cuts to within an additive error of roughly ϕ, using $\widetilde{O}(n/\phi)$ linear measurements. This is enough to find a balanced ϕ-sparse cut in every subgraph induced by a given partition of the vertex set. There is another caveat, though: power-cut sparsifiers give a high probability guarantee for any fixed partition, but we cannot expect them to work for all partitions simultaneously. Therefore, in order not to use the sparsifiers adaptively, we need one power-cut sparsifier for every recursion level. The authors show that the depth of the procedure cannot exceed $\widetilde{O}(1/\epsilon) = \widetilde{O}(1/\phi)$, thus obtaining the space complexity stated before. A few more details are involved in the polynomial time algorithm, but the underlying framework is the same.

1.2 Our Contribution

The work of [16] initiated the study of expander decompositions in the streaming setting, and consequently raised the question of whether a dependence in space on the sparsity parameter ϕ is inherent. In this paper, we move towards settling the streaming complexity of expander decompositions by attacking the problem on two fronts: (1) we give a nearly optimal algorithm for "one-level" expander decomposition that avoids the sparsity dependence, and (2) we show that computing a "repeated" expander decomposition, commonly used in applications, cannot avoid such dependence.

Upper Bound

We give an $\widetilde{O}(n)$ space algorithm for computing a $(O(\phi \log n), \phi)$-ED in dynamic streams. Specifically, we show that a "universal" sketch consisting of $\widetilde{O}(n)$ random linear measurements of the incidence matrix can be decoded into a $(O(\phi \log n), \phi)$-ED for any ϕ: the sketch is independent of the sparsity ϕ.

▶ **Theorem 1** (ED algorithm – exponential time decoding). *Let $G = (V, E)$ be a graph given in a dynamic stream. Then, there is an algorithm that maintains a linear sketch of G in $\widetilde{O}(n)$ space. For any $\phi \in (0, 1)$, the algorithm decodes the sketch to compute a $(O(\phi \log n), \phi)$-ED of G with high probability, in $\widetilde{O}(n)$ space and $2^{O(n)}$ time.*

We note that at least $\Omega(n \log n)$ space is needed for any small enough ϕ: for example, if the input graph is a matching of size $n/10$, say, its ED gives a $1 - O(\phi \log n)$ fraction of the matching edges.

The decoding time of our sketch can be made polynomial, at the expense of some loss in the quality of the expander decomposition (similarly to [16]), but keeping the space independent of the sparsity ϕ.

▶ **Theorem 2** (ED algorithm – polynomial time decoding). *Let $G = (V, E)$ be a graph given in a dynamic stream. Then, there is an algorithm that maintains a linear sketch of G in $n^{1+o(1)}$ space. For any $\phi \in (0, 1)$, the algorithm decodes the sketch to compute a $(\phi \cdot n^{o(1)}, \phi)$-ED of G with high probability, in $n^{1+o(1)}$ space and $\mathrm{poly}(n)$ time.*

In this case we are off by subpolynomial factors in both quality and space complexity as compared to the optimal ones. The actual theorem that we prove allows one to trade-off the loss in quality and increase in space.

Lower Bound

Most algorithmic applications of EDs, including the ones mentioned above, do not use just one ED of the input graph. Rather, they use an ED sequence obtained by repeatedly computing an ED of the inter-cluster edges from the previous level. This can be done in two natural ways: by contracting the clusters of an ED (we call this variant CED, for "contraction"), or by removing the intra-cluster edges without changing the vertex set (we call this variant RED, for "removal", see Section 1.3). These approaches lead to different results and are used for different applications (e.g., [19, 28] contract the clusters, and [32, 11] recurse on inter-cluster edges). In the sequential setting, both CEDs and REDs can be obtained straightforwardly given an ED algorithm. However, this is not so obvious in the streaming model.

One the one hand, one should be able get a sparsity-independent algorithm for computing CEDs in dynamic streams via Theorem 1 or Theorem 2: observe that contracting the vertices of a sparsifier of the input graph G gives a sparsifier of the graph obtained by contracting vertices in G, so the idea would be to maintain an independent copy of our algorithm for each of the $O(\log n)$ levels and contracting vertices in the sparsifier based on the decomposition of the previous level. On the other hand, we show a space lower bound for computing REDs, even in insertion only streams, showing that a dependence on $1/\phi$ is necessary.

▶ **Theorem 3** (RED lower bound). *Let $\epsilon, \phi \in (0,1)$ such that $1/n \ll \phi \ll 1/\operatorname{polylog} n$ and $\epsilon = \widetilde{O}(\phi)$. Any streaming algorithm that with constant probability computes at least two levels of an (ϵ, ϕ)-RED requires $\widetilde{\Omega}(n/\phi)$ bits of space.*

The result seems to challenge the intuition that EDs become weaker as ϕ and ϵ decrease (note that when ϕ is, say, $1/n^2$, an ED can simply consist of connected components). The questions of (1) whether this bound can be improved, (2) how it scales with the number of levels of RED we compute, and (3) whether there are algorithms matching such bounds, remain open.

1.3 Preliminaries

Graph Streaming

In this paper, we will be mostly working in the dynamic graph streaming model, where we know the vertex set $V = [n]$, and we receive a stream of insertions and deletions for undirected edges over V. In insertion-only graph streams, the only difference is that previously inserted edges cannot be deleted. At the end of the stream, the graph $G = (V, E)$ consists of the edges that have been inserted and not deleted, and we say that G is given in a (dynamic) stream. When we consider a graph given in a (dynamic) stream, we are implicitly assuming it to have n vertices, without introducing the parameter n explicitly. Also, when we refer to G and n without reintroducing them we are implicitly considering the graph resulting from the input stream and its number of vertices.

A powerful tool for dynamic graph streams is linear sketching, introduced in the seminal work of Ahn, Guha, and McGregor [4]. The idea is to left-multiply the $\binom{n}{2} \times n$ incidence matrix of the graph by a random $k \times \binom{n}{2}$ matrix for $k \ll \binom{n}{2}$. Since the sketch consists of linear measurements, it automatically handles the case of dynamic streams. We will not be using sketching techniques directly, but rather employ existing algorithms that do. In this paper we restrain ourselves to unweighted edge streams. One may study the same problem in general turnstile graph streams [13], but we do not do this here.

Cuts, Volumes, and Expanders

Given an unweighted graph $G = (V, E)$, possibly with multiple self-loops on the vertices, we will operate with weighted graphs that approximate G in an appropriate sense. We write $G' = (V', E', w)$ to denote a weighted graph, possibly with multiple weighted self-loops on the vertices. We describe next some notation for such weighted graph G'. The same notation carries over to the unweighted graph G by implicitly setting w to assign a weight of 1 to all edges and self-loops.

For any $A, B \subseteq V'$ we denote by $E'(A, B)$ the edges in E' with one endpoint in A and one in B, and by $w(A, B)$ the total weight of edges in $E'(A, B)$. The volume of a cut $S \subseteq V'$ is the sum of the (weighted) degrees, including self-loops, of its vertices. We denote it by $\mathrm{vol}_{G'}(S)$. The sparsity of a cut $\emptyset \neq S \subsetneq V'$ is defined as

$$\Phi_{G'}(S) = \frac{w(S, V' \setminus S)}{\min\{\mathrm{vol}_{G'}(S), \mathrm{vol}_{G'}(V' \setminus S)\}} .$$

For $\psi \in (0, 1)$, we make a distinction between cuts having sparsity less than ψ, which we call ψ-sparse, and cuts having sparsity at least ψ, which we call ψ-expanding. The sparsity of G' is then defined as

$$\Phi_{G'} = \min_{\emptyset \neq S \subsetneq V'} \Phi_{G'}(S) ,$$

and we call G' a ψ-expander if all its cuts are ψ-expanding, i.e. $\Phi_{G'} \geq \psi$.

Expander Decomposition

As we will treat expander decompositions for the input graph G only, we conveniently use the following additional notation. For a cluster $U \subseteq V$, i.e. a subset of the vertices, and a cut $S \subseteq U$, which is also a subset of the vertices, we denote the number of edges crossing S in U by $\partial_U S$, i.e. $\partial_U S = |E(S, U \setminus S)|$. This is the *local* cut of S in U. If $U = V$, we use a shorthand notation $\partial S = \partial_V S$. We call such a cut the *global* cut of S, since for $S \subseteq U$ we will be interested in both $\partial_U S$ and ∂S. Moreover, we drop the subscript from the volume and simply write $\mathrm{vol}(\cdot)$ instead of $\mathrm{vol}_G(\cdot)$. Then, EDs can be defined as follow.

▶ **Definition 4** (Expander decomposition). *Let $G = (V, E)$ and let $\epsilon, \phi \in (0, 1)$. A partition $\mathcal{U}$ of V is an (ϵ, ϕ)-expander decomposition (for short, (ϵ, ϕ)-ED) of G if*

1. $\frac{1}{2} \sum_{U \in \mathcal{U}} \partial U \leq \epsilon |E|$, *and*
2. *for every $U \in \mathcal{U}$, one has that $G[U]$ is a ϕ-expander.*

Boundary-Linked Expander Decomposition

A boundary-linked ED [19] is the same as a classical ED except that Property 2 of Definition 4 is strengthened. For a cut S in a cluster U, the boundary $\mathrm{bnd}_U(S)$ of S with respect to U is the number of edges that go from S to the outside of U, i.e. $\mathrm{bnd}_U(S) = |E(S, V \setminus U)|$. For $U \subseteq V$ and $\tau \geq 0$, the τ-boundary linked subgraph of G on U, denoted by $G[U]^\tau$, is the subgraph of G induced by U with additional $\tau \cdot \mathrm{bnd}_U(\{u\})$ self-loops attached to every $u \in U$. Then a boundary-linked ED can be defined as follows.

▶ **Definition 5** (Boundary-linked expander decomposition [19]). *Let $G = (V, E)$, let $b, \epsilon, \phi \in (0, 1)$ be parameters such that $b \geq \phi$, and let $\gamma \geq 1$ be an error parameter. A partition $\mathcal{U}$ of V is a $(b, \epsilon, \phi, \gamma)$-boundary-linked expander decomposition (for short, $(b, \epsilon, \phi, \gamma)$-BLD) of G if*

1. $\frac{1}{2} \sum_{U \in \mathcal{U}} \partial U \leq \epsilon |E|$, *and*
2. *for every $U \in \mathcal{U}$, $G[U]^{b/\phi}$ is a ϕ/γ-expander.*

Algorithm 1 DECOMPOSE: recursive procedure for computing a $(O(\phi \log n), \phi)$-ED of G.

```
// G = (V, E) is the input graph
// φ ∈ (0, 1) is the sparsity parameter
procedure DECOMPOSE(G):
    if G is a φ-expander then
        return {V}
    else
        S ← a φ-sparse cut of G
        return DECOMPOSE(G[S]) ∪ DECOMPOSE(G[V \ S])
    end
```

Expander Decomposition Sequence

For a partition $\mathcal{U}$ of V (think of $\mathcal{U}$ as an ED of G), we denote by $E \setminus \mathcal{U}$ the set of inter-cluster (or crossing) edges with respect to $\mathcal{U}$, i.e.

$$E \setminus \mathcal{U} = E \setminus \bigcup_{U \in \mathcal{U}} \binom{U}{2}.$$

Analogously, we let $G \setminus \mathcal{U} = (V, E \setminus \mathcal{U})$ be the subgraph of G obtained by removing the intra-cluster edges in $\mathcal{U}$. For a sequence of partitions $\mathcal{U}_1, \ldots, \mathcal{U}_\ell$ of V and $i \in [\ell]$, we define $G_1^{\mathrm{R}} = G$ and denote by $G_{i+1}^{\mathrm{R}} = G_i^{\mathrm{R}} \setminus \mathcal{U}_i$ the subgraph of G obtained by removing the intra-cluster edges of the first i partitions.

▶ **Definition 6** (Removal-based ED sequence). *Let $G = (V, E)$, let $\epsilon, \phi \in (0, 1)$, let $\ell \geq 1$, and let $\mathcal{U}_1, \ldots, \mathcal{U}_\ell$ be a sequence of partitions of V. The sequence $\mathcal{U}_1, \ldots, \mathcal{U}_\ell$ is an ℓ-level removal-based (ϵ, ϕ)-ED sequence (for short, ℓ-level (ϵ, ϕ)-RED or (ϵ, ϕ, ℓ)-RED) of G if, for all $i \in [\ell]$, $\mathcal{U}_i$ is an (ϵ, ϕ)-ED of the graph G_i^{R}.*

2 Sparsity-Independent One-Level Expander Decomposition

Given a graph $G = (V, E)$ in a dynamic stream, and a parameter $\phi \in (0, 1)$, we consider the problem of computing an (ϵ, ϕ)-ED of G for $\epsilon = O(\phi \log n)$. We show that one can do this in $\widetilde{O}(n)$ bits of space, without any dependence on ϕ. In this section, we sketch our approach.

Let us begin by recalling the standard recursive framework for constructing expander decompositions [22, 32, 33], concisely summarized in Algorithm 1. For any parameter $\phi \leq 10^{-1}/\log n$, this algorithm produces a $(O(\phi \log n), \phi)$-ED by recursively partitioning G along ϕ-sparse cuts until no more such cuts are found.

Many algorithmic constructions of EDs are essentially efficient implementations of Algorithm 1. Adapting Algorithm 1 to dynamic streams comes with its own set of challenges. When the input is given in a dynamic stream, one can only afford to store a limited amount of information about the input graph. Since Algorithm 1 only needs to measure the sparsity of cuts, it seems enough to have access to cut sizes and volumes. Both these quantities are preserved by cut sparsifiers. According to the classical definition [7], a δ-cut sparsifier is a weighted subgraph $H = (V, E', w)$ of the input $G = (V, E)$, where for every cut $S \subseteq V$ one has

$$(1 - \delta) \cdot \partial S \leq \partial^w S \leq (1 + \delta) \cdot \partial S.$$

It is known that such a sparsifier can be constructed in dynamic streams using $\widetilde{O}(n/\delta^2)$ bits of space [5]. With the same space requirement we can also measure the sparsity of cuts up to a $(1 \pm \delta)$ multiplicative error. Having this in mind, it is natural to consider the following algorithm: first, read the stream and construct a cut sparsifier, then run Algorithm 1 on it and output the resulting clustering as the expander decomposition. However, this approach does not work as is. As it turns out, more information is needed about the graph than what is captured by the regular notion of a cut sparsifier.

The problem with this approach becomes immediately apparent when one considers the second recursion level. As the process recurses on the two sides of a sparse cut S, it will repeat the procedure on the subgraphs $G[S]$ and $G[V \setminus S]$. Unfortunately, the cut preservation property of the sparsifier does not carry over to those subgraphs, making it inadequate for estimating the sparsity of the cuts in those graphs. In fact, the notion of expander decomposition itself already operates on the subgraphs, as it is required that each cut in each cluster of the decomposition is expanding.

2.1 Testing Expansion of Subgraphs

The reasoning from above gives rise to the following sketching problem: produce a sparsifier that can be used to check that any subgraph is a ϕ-expander. To solve it, the authors of [16] introduced the concept of a (δ, ψ)-power-cut sparsifier. Its property is that, for any cluster $U \subseteq V$, with high probability all cuts $S \subseteq U$ are preserved within an additive-multiplicative error. Recall that $\partial_U S$ is equal to the size of the cut S inside the induced subgraph $G[U]$. When talking about sparsifiers in this section, we will slightly abuse notation by assuming that there is some instance $H = (V, E', w)$ of it and denote by $\partial_U^w S$ the size of the cut S in the subgraph $H[U]$ of this sparsifier H. Then we can write the guarantee of a (δ, ψ)-power-cut sparsifier as follows[1]:

$$\forall\, S \subseteq U, \quad (1-\delta) \cdot \partial_U S - \psi \cdot \mathrm{vol}(S) \leq \partial_U^w S \leq (1+\delta) \cdot \partial_U S + \psi \cdot \mathrm{vol}(S)\,.$$

The authors also give a dynamic stream construction, which uses $\widetilde{O}(n/\delta\psi)$ bits of space by sampling edges proportionally to the degrees of their endpoints.

To check the ϕ-expansion of subgraphs up to a small constant multiplicative error, it is enough to set δ to be a small constant. However, the multiplicative error parameter ψ must be $\lesssim \phi$. This is a significant downside of this construction, as it was shown in [16] that a (δ, ϕ)-power-cut sparsifier must have at least $\Omega(n/\phi)$ edges. In fact, their lower bound is more general, and holds for general subgraphs[2]. In other words, $\Omega(n/\phi)$ space is necessary in order to test ϕ-expansion of general subgraphs. Consequently, new tools must be used to have any hope of getting an algorithm independent of $1/\phi$.

Our Contribution: Sparsification of Boundary-Linked Subgraphs

As it turns out, solving the expansion testing problem, as it was stated, is not necessary. Recently, in the breakthrough work of [19], it was shown that one could demand additional properties from the expander decomposition, and it will still exist at the price of increasing the number of inter-cluster edges by a small multiplicative factor.

[1] A similar sparsifier construction was proposed by [1]. Their construction has an additive error of $\psi|S|$, so the dependence is on the number of vertices instead of the volume.

[2] The lower bound instance is a $1/\phi$-regular graph. There, each edge by itself forms a ϕ-expander, while any pair of vertices without an edge between them is not. Being able to test the ϕ-expansion of the majority of those small subgraphs would imply being able to recover the majority of edges in the graph.

More formally, let $U \subseteq V$ and $\tau > 0$. Note that if the τ-boundary-linked subgraph $G[U]^\tau$ (see Section 1.3 for the definition) is a ϕ-expander, then so necessarily is $G[U]$, but not the other way around. Then it is possible, for a given ϕ, and $\epsilon = \widetilde{O}(\phi)$, $\tau \approx 1/\phi$, to construct an (ϵ, ϕ)-expander decomposition $\mathcal{U}$ where for each cluster $U \in \mathcal{U}$, $G[U]^\tau$ is a $\widetilde{\Omega}(\phi)$-expander [19]. Such a decomposition is called a boundary-linked expander decomposition.

A key observation is that if in our algorithm we were aiming to construct a boundary-linked expander decomposition, the testing problem would only involve checking that any given boundary-linked subgraph is ϕ-expanding. Indeed, this problem is much easier than the original one and can be solved in space $\widetilde{O}(n)$. We will show how to do it in two steps: first, we will discuss how to strengthen the power-cut sparsifier, and then prove that this strengthening is enough to resolve the problem.

Achieving Additive Error in the Global Cut

As was noted in [16], the idea behind constructing a power-cut sparsifier was to reanalyse the guarantee given by the construction of [32] for sparsifying expanders. In other words, a power-cut sparsifier results from a more rigorous analysis of an existing sparsifier. The problem with this approach is that the sparsifier of [32] is relatively weak to begin with: it only preserves cuts in expanders, while other constructions can preserve them in all graphs [7, 17]. To strengthen the guarantee, one can give the same treatment to the construction of [17] (see the full version of this paper). For the sake of simplicity and to gain an intuition for why this kind of sparsification is at all possible, we discuss here how to do that with the classical construction of [27] by closely following the original proof.

We show that, given a graph $G = (V, E)$ with a minimum cut of size k, it is possible to construct a sparsifier H of G such that every cut inside any given subgraph $G[U]$ of G is preserved with high probability with the following guarantee:

$$\forall\, S \subseteq U, \quad \partial_U S - \delta \cdot \partial S \leq \partial_U^w S \leq \partial_U S + \delta \cdot \partial S\,. \tag{1}$$

In this paper, a sparsifier with the property of Equation (1) is called a cluster sparsifier. To achieve the guarantee of Equation (1), consider using the same process as in [27]: sample each edge with the same probability $p \approx \delta^{-2}/k$.

To see why this works, fix a subgraph $U \subseteq V$, and consider any cut S in U. We wish to show that the size of the cut S inside U concentrates well after sampling. In the original proof, this is done by simply applying a Chernoff bound. In our case, this bound would look like this:

$$\Pr[|\partial_U^w S - \partial_U S| \geq \delta \cdot \partial_U S] \leq \exp\left(-\frac{1}{3}\delta^2 p \cdot \partial_U S\right)\,.$$

However, this is insufficient, as the probability depends on the cut size inside $G[U]$. As we have no lower bound on its size, unlike with the sizes of global cuts, the second part of the argument of [27] cannot be applied. Instead, we apply an additive-multiplicative version of the Chernoff bound (see for example [16]), that allows us to compare the approximation error with a bigger value than its expectation. This gives us

$$\Pr[|\partial_U^w S - \partial_U S| \geq \delta \cdot \partial S] \leq 2\exp\left(-\frac{1}{100}\delta^2 p \cdot \partial S\right)\,.$$

Expressing the probability in terms of the global cut allows us to use the cut counting lemma [26], which bounds the number of global cuts of size at most αk by $n^{2\alpha}$, for $\alpha \geq 1$. The proof is concluded by associating each global cut with a local cut in $G[U]$ and taking the union bound over them.

In order to get the guarantee of Equation (1) for all graphs, not only those with a minimum cut of size k, one can sample edges proportionally to the inverse of their edge connectivity (as in the work of [17]) as opposed to uniformly. Moreover, one can implement such sampling scheme in dynamic streams, using the approach of Ahn, Guha, and McGregor [5]. We defer the details of these reanalyses to the full version of this paper, where we show how to construct cluster sparsifiers with the property of Equation (1) in dynamic streams.

Benefits of Boundary-Linked Subgraphs

To see why the cluster sparsifier is enough to solve the boundary-linked ϕ-expansion testing problem, consider the following reasoning. Set the self-loop parameter τ equal to b/ϕ, for some $b \gg \phi$ and $b \ll 1$. Fix a cluster U, for which $G[U]^{b/\phi}$ is an $\widetilde{\Omega}(\phi)$-expander. The crucial fact is that the size of any cut inside $G[U]^{b/\phi}$ is lower bounded by its size in the global graph up to a small polylogarithmic factor in the following way:

$$\partial_U S \geq \widetilde{\Omega}(b) \cdot \partial S \,. \tag{2}$$

We will now explain the derivation of the above equation in detail. For a cut $\emptyset \neq S \subsetneq U$, recall that $\mathrm{bnd}_U(S)$ is the number of edges going from S to $V \setminus U$. Note that by the definition of $G[U]^{b/\phi}$, the volume of any cut S inside of it is equal to

$$\mathrm{vol}_{G[U]^{b/\phi}}(S) = \mathrm{vol}(S) + \left(\frac{b}{\phi} - 1\right) \mathrm{bnd}_U(S) \,.$$

First, because we assume that $G[U]^{b/\phi}$ is an $\widetilde{\Omega}(\phi)$-expander, we have

$$\partial_U S \geq \widetilde{\Omega}(\phi) \cdot \mathrm{vol}_{G[U]^{b/\phi}}(S) \,.$$

Then, applying the aforementioned formula for $\mathrm{vol}_{G[U]^{b/\phi}}(S)$, we have

$$\partial_U S \geq \widetilde{\Omega}(\phi)\,\mathrm{vol}(S) + \widetilde{\Omega}(\phi)\left(\frac{b}{\phi} - 1\right) \mathrm{bnd}_U(S) \,.$$

Since $b \gg \phi$ and dropping the first summand, the above simplifies to

$$\partial_U S \geq \widetilde{\Omega}(b)\,\mathrm{bnd}_U(S) \,.$$

Finally, applying $\partial_U S + \mathrm{bnd}_U(S) = \partial S$ and $b \ll 1$, we arrive back at Equation (2).

A consequence of Equation (2) is that setting $\delta \approx 1/b$ in the cluster sparsifier produces the following guarantee for $\widetilde{\Omega}(\phi)$-expander subgraphs $G[U]^{b/\phi}$:

$$\forall\, S \subseteq U, \quad \partial_U S - O(1) \cdot \partial_U S \leq \partial_U^w S \leq \partial_U S + O(1) \cdot \partial_U S \,,$$

where the $O(1)$ can be made arbitrarily small. In other words, we achieve a multiplicative approximation guarantee on subgraphs of interest, which is enough to solve the testing problem. Since b can be set to be $1/\operatorname{polylog} n$, the final sparsifier size does not depend on $1/\phi$. In this sense, such sparsifier can be thought of as a "universal" sketch of the graph that allows to solve the testing problem for all ϕ simultaneously.

Even though we now know how to solve the testing problem, an implementation of Algorithm 1 would need to actually find a sparse cut when the test fails (and in particular, it needs to find a *balanced* sparse cut, see the next section). In the full version of this paper, we show that cluster sparsifiers enable to solve this harder task too: consider an offline algorithm that either correctly determines a graph to be a boundary-linked expander or finds

a (balanced) sparse cut; one can prove that we can run such algorithm on a subgraph of the sparsifier as a black box and obtain essentially the same result as if the algorithm was run on the corresponding subgraph of the original graph. In this sense, cluster sparsifiers serve as small error proxies to the original graph for expander-vs-sparse-cut type of queries on vertex-induced subgraphs.

2.2 Low Depth Recursion

Another problem that arises when using the recursive approach is that the subgraph sparsification guarantee of both power-cut sparsifiers and cluster sparsifiers is only probabilistic, and it can be shown that it cannot be made deterministic [16]. This means that after finding a sparse cut S in a subgraph $G[U]^{b/\phi}$ of a sparsifier, we cannot claim that the same sparsifier would preserve the cuts inside the new graph $G[S]^{b/\phi}$ with high probability, as it is dependent on another cut that we have already found inside the sparsifier. This means we cannot use the same sparsifier for two different calls to Algorithm 1 inside the same execution path. However, we can share a sparsifier among all the calls at the same recursion level since these will operate on independent portions of G. Therefore, we want to have a separate sparsifier for every recursion level. This means that in order to minimize space requirements, it is crucial to have a small recursion depth.

To have small recursion depth, the algorithmic approach of [8, 9, 16] enforces the sparse cut S from line 5 of Algorithm 1 to be balanced, i.e. none of S and $V \setminus S$ is much larger than the other. In particular, they only recurse on the two sides of a cut if $\mathrm{vol}(S) \gtrsim \epsilon \,\mathrm{vol}(V \setminus S)$. When there is no balanced sparse cut and yet the input is not an expander, a lemma of Spielman and Teng [32] suggests there should be an $\Omega(\phi)$-expander $G[S']$ that accounts for a $(1 - O(\epsilon))$-fraction of the total volume. An algorithmic version of this structural result allows us to iteratively trim off a small piece of the graph until such S' is found. At this point, the algorithm of [16] can simply return $\{S'\} \cup (\cup_{u \in V \setminus S'} \{u\})$ as an ED, with at most $O(\epsilon|E|)$ inter-cluster edges between singletons. As the volume of the cluster multiplicatively decreases by $1 - O(\epsilon)$ after each call, this gives recursion depth at most $\widetilde{O}(1/\epsilon) \approx \widetilde{O}(1/\phi)$.

Our Contribution: Adaptation of Trimming

We show that a simple refinement of this approach allows us to adapt the framework of [30], which leads to an algorithm with recursion depth independent of ϕ. We run the same algorithm, but instead of separating each vertex in $V \setminus S'$ into its own singleton cluster, we recurse with Algorithm 1 on the whole set $V \setminus S'$. Conceptually, this implements an analogue of the trimming step of [30].

This means that at the end, fewer edges in $V \setminus S$ become inter-cluster edges. Because of that, we can strengthen the balancedness requirement: we recurse on the two sides of a sparse cut only if $\mathrm{vol}(S) \gtrsim \frac{1}{C} \mathrm{vol}(V \setminus S)$ for a large constant C. This allows us to trim more vertices each time, resulting in $\widetilde{O}(1)$ depth of the trimming step. On the other hand, because in each call to Algorithm 1 the volume of clusters passed to recursive calls is decreased by at least a constant factor, the total recursion depth becomes at most $\widetilde{O}(1)$.

Other than the refinement discussed above, our space efficient implementation of Algorithm 1, as well as the iterative procedure to find the large expander S', are almost the same as the ones of [16] (which in turn are inspired by the one of [9]). The details of the algorithms are given in the full version of this paper.

2.3 Putting It All Together

Combining the ideas illustrated in the two sections above, neither the sparsifier's size nor the recursion depth depend on ϕ, thus giving a sparsity-independent space algorithm for boundary-linked expander decomposition (BLD for short). This result is stated in terms of parameters $b, \epsilon, \phi \in (0,1)$, $\gamma \geq 1$: a (b,ϵ,ϕ,γ)-BLD is a partition $\mathcal{U}$ with at most an ϵ fraction of crossing edges and every $U \in \mathcal{U}$ induces a ϕ/γ-expander $G[U]^{b/\phi}$ (see Section 1.3). Using this terminology, we obtain the following result.

▶ **Theorem 7** (Exponential time decoding BLD). *Let $G = (V,E)$ be a graph given in a dynamic stream, and let $b \in (0,1)$ be a parameter such that $b \leq 1/\log^2 n$. Then, there is an algorithm that maintains a linear sketch of G in $\widetilde{O}(n/b^3)$ space. For any $\epsilon \in [n^{-2}, b\log n]$, the algorithm decodes the sketch to compute, with high probability and in $\widetilde{O}(n/b^3)$ space and $2^{O(n)}$ time, a (b,ϵ,ϕ,γ)-BLD of G for*

$$\phi = \Omega\left(\frac{\epsilon}{\log n}\right) \quad \text{and} \quad \gamma = O(1)\,.$$

From this, one can easily conclude our main result.

▶ **Theorem 8** (See Theorem 1). *Let $G = (V,E)$ be a graph given in a dynamic stream. Then, there is an algorithm that maintains a linear sketch of G in $\widetilde{O}(n)$ space. For any $\phi \in (0,1)$ such that $\phi \leq c/\log^2 n$ for a small enough constant $c > 0$, the algorithm decodes the sketch to compute a $(O(\phi \log n), \phi)$-ED of G with high probability, in $\widetilde{O}(n)$ space and $2^{O(n)}$ time.*

We remark that for $\phi \geq \Omega(1/\log^2 n)$, one can use the algorithm of [16] to still have an $\widetilde{O}(n)$ space construction of a $(O(\phi \log n), \phi)$-ED.

Proof. First note that without loss of generality we can assume $\phi \geq 1/n^2$, otherwise an ED can simply consist of the connected components of G (which can be computed in dynamic streams in $\widetilde{O}(n)$ space [4]). Then, we note that since c is small enough and $\phi \leq c/\log^2 n$, one can always define $\epsilon = C \cdot \phi \cdot \log n$ for an appropriate constant $C > 0$ while ensuring $1/n^2 \leq \epsilon \leq 1/\log n$. We can thus prove the theorem by equivalently showing that there is an algorithm that maintains a linear sketch of G in $\widetilde{O}(n)$ space, and that for all $\epsilon \in [1/n^2, 1/\log n]$ decodes the sketch to compute, with high probability, an $(\epsilon, \Omega(\epsilon/\log n)))$-ED of G in $\widetilde{O}(n)$ space and $2^{O(n)}$ time.

Let then $\epsilon \in [1/n^2, 1/\log n]$. We use the algorithm from Theorem 7 with parameter ϵ and a parameter b of our choice. We need to meet two preconditions: $b \leq 1/\log^2 n$ and $\epsilon \leq b \log n$. Since we assume $\epsilon \leq 1/\log n$, we can set $b = 1/\log^2 n$, and all the prerequisites are fulfilled. Then the algorithm from Theorem 7 runs in $2^{O(n)}$ time and takes $\widetilde{O}(n/b^3) = \widetilde{O}(n)$ bits of space. The output $\mathcal{U}$ is a (b,ϵ,ϕ,γ)-BLD of G with high probability, where $\phi = \Omega(\epsilon/\log n)$ and $\gamma = O(1)$. Since a (b,ϵ,ϕ,γ)-BLD of G is an $(\epsilon,\phi/\gamma)$-ED of G, we have obtained an $(\epsilon, \Omega(\epsilon/\log n))$-ED of G with high probability. ◀

The exponential time in the decoding is due to the subtask of finding a balanced sparse cut. As we show, one can make the decoding time polynomial by resorting to known offline approximation algorithms [29, 30]. However, we only have $\log^{\Omega(1)} n$-approximations for finding a balanced sparse cut, and in particular, we do not expect (under NP-hardness and the Unique Games Conjecture) there to be a polynomial time $O(1)$-approximation [10]. Such super-constant factor error incurs some loss in the quality of decomposition and space requirement, which, nevertheless, remains independent of the sparsity.

▶ **Theorem 9** (Polynomial time decoding BLD). *Let $G = (V, E)$ be a graph given in a dynamic stream, and let $b \in (0, 1)$ be a parameter such that $b \leq 1/\log^5 n$. Then, there is an algorithm that maintains a linear sketch of G in $n/b^3 \cdot \log^{O(\log n/\log \frac{1}{b})} n$ space. For any $\epsilon \in [n^{-2}, b \log n]$, the algorithm decodes the sketch to compute, with high probability and in $n/b^3 \cdot \log^{O(\log n/\log \frac{1}{b})} n$ space and $\mathrm{poly}(n)$ time, a $(b, \epsilon, \phi, \gamma)$-BLD of G for*

$$\phi = \Omega\left(\frac{\epsilon}{\log^4 n}\right) \quad \text{and} \quad \gamma = \log^{O\left(\frac{\log n}{\log 1/b}\right)} n\,.$$

A polynomial time version of Theorem 8 then follows from Theorem 9.

▶ **Theorem 10** (See Theorem 2). *Let $G = (V, E)$ be a graph given in a dynamic stream, and let $b \in (0, 1)$ be a parameter such that $b \leq 1/\log^5 n$. Then, there is an algorithm that maintains a linear sketch of G in $n/b^3 \cdot \log^{O(\log n/\log \frac{1}{b})} n$ space. For any $\phi \in (0, 1)$ such that $\phi \leq b/\log^{C \cdot \log n/\log \frac{1}{b}} n$ for a large enough constant $C > 0$, the algorithm decodes the sketch to compute a $(\phi \cdot \log^{O(\log n/\log \frac{1}{b})} n, \phi)$-ED of G with high probability, in $n/b^3 \cdot \log^{O(\log n/\log \frac{1}{b})} n$ space and $\mathrm{poly}(n)$ time.*

We remark that setting, say, $b = 2^{-\sqrt{\log n}}$ in the above result gives a $n^{1+o(1)}$ space algorithm for computing a $(\phi \cdot n^{o(1)}, \phi)$-ED for any $\phi \leq 2^{-2C \log\log n\sqrt{\log n}}$. For larger values of ϕ, one can use the polynomial time algorithm of [16] to still get a $n^{1+o(1)}$ space construction for a $(\phi \cdot n^{o(1)}, \phi)$-ED.

Proof. As in the proof of Theorem 8, we can assume $\phi \geq 1/n^2$. Also, by virtue of C being a large enough constant and $\phi \leq b/\log^{C \cdot \log n/\log \frac{1}{b}} n$, one can always define ϵ to be $\epsilon = \phi \cdot \log^{C \cdot \log n/\log \frac{1}{b}} n$ while ensuring $n^{-2} \leq \epsilon \leq b \log n$. Then, we equivalently prove that for any $b \in (0, 1)$ with $b \leq 1/\log^5 n$ there is an algorithm that maintains a linear sketch of G in $n/b^3 \cdot \log^{O(\log n/\log \frac{1}{b})} n$ space, and that for any $\epsilon \in (0, 1)$ such that $n^{-2} \leq \epsilon \leq b \log n$ decodes the sketch to compute, with high probability, an $(\epsilon, \epsilon/\log^{O(\log n/\log \frac{1}{b})} n)$-ED of G in $n/b^3 \cdot \log^{O(\log n/\log \frac{1}{b})} n$ space and $\mathrm{poly}(n)$ time.

Let then $b, \epsilon \in (0, 1)$ with $b \leq 1/\log^5 n$ and $n^{-2} \leq \epsilon \leq b \log n$. We use the algorithm from Theorem 9 with the same parameters b and ϵ, since every admissible pair of parameters b and ϵ fulfils the conditions of Theorem 9. The space complexity is also the same, and the running time is $\mathrm{poly}(n)$. Again, observe that a $(b, \epsilon, \phi, \gamma)$-BLD of G is an $(\epsilon, \phi/\gamma)$-ED of G. Hence the claim, since Theorem 9 gives

$$\frac{\phi}{\gamma} = \Omega\left(\frac{\epsilon}{\log^4 n}\right) \cdot \frac{1}{\log^{O\left(\frac{\log n}{\log 1/b}\right)} n} = \frac{\epsilon}{\log^{O\left(\frac{\log n}{\log 1/b}\right)} n}\,. \qquad \blacktriangleleft$$

The proofs of Theorem 7 and Theorem 9 can be found in the full version of this paper.

3 Two-Level Expander Decomposition Incurs a Sparsity Dependence

Given a graph $G = (V, E)$ in a stream and parameters $\epsilon, \phi \in (0, 1)$, we consider the problem of computing a two-level (ϵ, ϕ)-RED of G (see Section 1.3). In other words, we study the problem of computing an (ϵ, ϕ)-ED $\mathcal{U}$ of G and an (ϵ, ϕ)-ED $\mathcal{U}'$ of the graph $G' = (V, E \setminus \mathcal{U})$, where $E \setminus \mathcal{U}$ denotes the set of inter-cluster edges of $\mathcal{U}$, i.e. the edges of E that are not entirely contained in a cluster $U \in \mathcal{U}$. We remark that we wish to do so in a single pass over the stream.

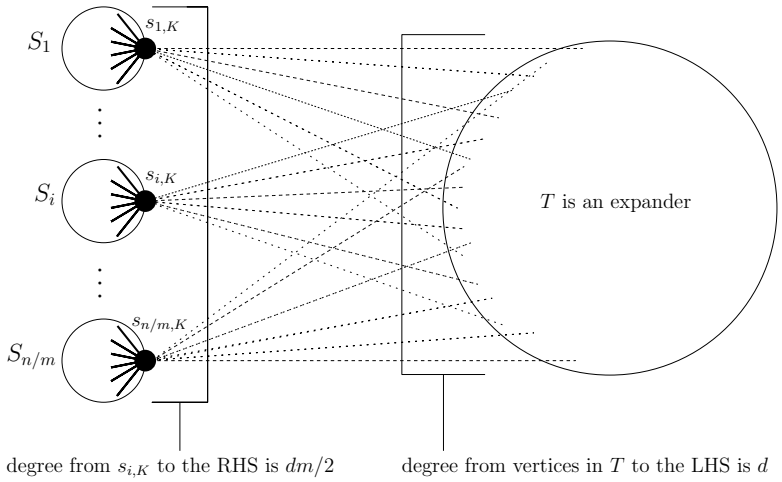


Figure 1 Illustration of the graph we use for proving the lower bound. Thick bullets represent important vertices, thick lines represent important edges, dotted lines represent edges connecting the important vertices to T.

A Natural Algorithmic Approach and Why It Fails

A naive attempt to solve this problem would be that of sketching the graph twice, for example by using our algorithm from Theorem 1. One can use the first sketch to construct the first level ED $\mathcal{U}$. After, the hope is that one can send updates to the second sketch so as to remove the intra-cluster edges and then decode this sketch into an ED of $G' = (V, E \setminus \mathcal{U})$ using again Theorem 1. However, this hope is readily dashed. Indeed, sketching algorithms break down if we send a removal update for an edge that was not there in the first place, and we do not have knowledge of which of the pairs $\binom{U}{2}$ are in E and which are not.

Our Contribution: Space Lower Bound

We show that, in sharp contrast to our algorithm for constructing a one-level expander decomposition, this problem requires $\widetilde{\Omega}(n/\phi)$ space, i.e. a dependence on $1/\phi$ is unavoidable. Formally, we obtain the following result.

▶ **Theorem 11** (See Theorem 3). *Let $\ell \geq 2$ and let $\epsilon, \phi \in (0,1)$ such that $\epsilon = 1 - \Omega(1)$, $\phi \leq \epsilon$, and $\phi \geq C \cdot \max\{\epsilon^2, 1/n\}$ for a large enough constant $C > 0$. Any streaming algorithm that with probability at least $9/10$ computes an ℓ-level (ϵ, ϕ)-RED requires $\Omega(n/\epsilon)$ bits of space.*

The above theorem gives an $\widetilde{\Omega}(n/\phi)$ space lower bound for algorithms that compute a RED with near-optimal parameters, i.e. algorithms that achieve $\epsilon = \widetilde{O}(\phi)$ for any $1/n \ll \phi \ll 1/\log n$.

Setup and Hard Instances

Throughout this section, the symbols $\ll$ and $\gg$ mean smaller or larger by a large constant factor. Let us fix the RED parameters $\epsilon, \phi \in (0,1)$, and let us restrain ourselves to the regime $\phi \gg \epsilon^2$ (and of course $\phi \ll \epsilon$). We prove the lower bound by giving a distribution over hard instances $G = (V, E)$. This distribution is parametrised by integers d and m such that $1 \ll d \ll m \ll n$, $m \gg 1/\phi$, $m \ll 1/\epsilon^2$, and $d \ll \frac{1}{\epsilon}$. With these parameters fixed, our hard distribution $\mathcal{G}$ is defined below in Definition 12. An illustration is given in Figure 1.

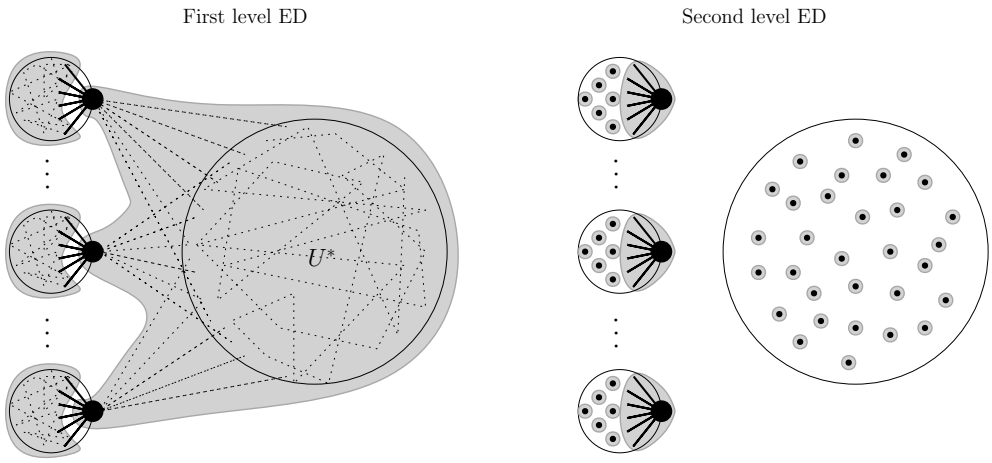


Figure 2 Illustration of the ideal expander decomposition of the graph. Thick bullets represent important vertices, smaller bullets represent ordinary vertices, thick lines represent important edges, dotted lines represent the rest of the edges. Grey areas represent the clusters in the decomposition.

▶ **Definition 12** (Distributions $\mathcal{G}$ and $\mathcal{G}'$ – Informal). *We partition V arbitrarily into two sets S and T with $n/2$ vertices each, and further partition S into n/m sets $S_1, \ldots, S_{n/m}$ with $m/2$ vertices each. The edge set of the graph $G = (V, E) \sim \mathcal{G}$ is defined as follows.*

1. *For each $i \in [n/m]$, the induced subgraph $G[S_i]$ is an Erdős-Rényi random graph with $m/2$ vertices and degree $\approx d$. We denote by $\mathcal{G}'$ the distribution of the subgraph $G[S]$.*
2. *The induced subgraph $G[T]$ is a* fixed *d-regular $\Omega(1)$-expander.*
3. *We fix for convenience an arbitrary labelling $s_{i,1}, \ldots, s_{i,m/2}$ of the vertices in each S_i, and we sample an index K uniformly from $[m/2]$. Then, for every $i \in [n/m]$, we add $dm/2$ edges from $s_{i,K}$ to T so that each $t \in T$ has d incident edges connecting to S.*

Roughly speaking, our hard instances should be composed of n/m regular expanders that are densely connected to T, which is also an expander, through a selection of "special" vertices. We will show that the hardness arises from recovering information about certain important vertices and edges, defined below and also illustrated in Figure 1.

▶ **Definition 13** (Important vertices and edges – Informal). *Let $G = (V, E) \sim \mathcal{G}$. We define the set of important vertices $V^* = \{s_{i,K} : i \in [n/m]\}$ to be the set of vertices of S that are connected to T, and define the set of important edges $E^* = \{\{s_{i,K}, v\} : i \in [n/m], v \in S\}$ to be the set of edges in the induced subgraph $G[S]$ that are incident on V^*.*

The lower bound proof has two steps: we first prove that a two-level RED leaks a non-trivial amount of information about the graph $G \sim \mathcal{G}$; then we prove that, in order to obtain such amount of information, the algorithm must use a lot of space.

3.1 Two-Level Expander Decomposition of the Hard Instance

In this section, we show that any valid two-level RED reveals a lot of informations about the important edges. The following lemma shows that a non-trivial amount of important edges are inter-cluster edges in the first level decomposition. An ideal decomposition is illustrated in Figure 2.

▶ **Lemma 14** (Informal). *Let $G = (V, E) \in \mathrm{supp}(\mathcal{G})$. Then, any (ϵ, ϕ)-ED $\mathcal{U}$ of G satisfies*

$$|E^* \setminus \mathcal{U}| \geq \frac{4}{5} \cdot |E^*| \ .$$

Proof sketch. By definition of the graph, there are $\Theta(dn)$ edges in the graph. Since there is at most an ϵ fraction of crossing edges, there are only $O(\epsilon dn)$ crossing edges. Note that there are $\Theta(dn)$ edges in $G[T]$, so only an $O(\epsilon)$ fraction of the edges in $G[T]$ are crossing edges. Furthermore, $G[T]$ is a regular expander: this implies that there is a large cluster $U^* \in \mathcal{U}$ comprising a $1 - O(\epsilon)$ fraction of T, together with a $1 - O(\epsilon)$ fraction of important vertices. The latter is true since the edges between S and T make up a constant fraction of the total volume and only a small fraction of the edges can be crossing. We refer the reader to the full version of this paper for more details.

The edges in the subgraph $G[S]$ also account for a constant fraction of the total volume. Together with the fact that each S_i induces a regular expander, for many of the S_i's we will have a cluster in $\mathcal{U}$ that contains most of S_i. Now consider a set S_i such that the important vertex of S_i is in U^* and most of the vertices in S_i are all in the same cluster. If most of the vertices in S_i are also inside U^*, then consider the cut from $(U^* \cap S_i) \setminus \{s_{i,K}\}$ to $(U^* \setminus S_i) \cup \{s_{i,K}\}$. The cut size is at most the number of important edges in S_i, which is $O(d)$. On the other hand, the volume of the cut is $\Theta(dm)$ since most of the vertices of S_i are in the cluster. Recalling that $m \gg \frac{1}{\phi}$, one concludes that the cut is sparse. Therefore, we ruled out the possibility of having many vertices of S_i in U^*. See the full version of this paper for a detailed discussion.

In summary, as illustrated in Figure 2, in any valid expander decomposition, most of the vertices in T and most of the important vertices are inside a giant cluster U^*, and for most of the S_i's, there is a cluster other than U^* that contains most of the vertices in S_i. For any such S_i, most of the important edges inside it are then crossing edges. ◀

The second level expander decomposition, i.e. an expander decomposition of the inter-cluster edges from the first level, is also quite structured, as illustrated in the ideal RED of Figure 2.

▶ **Lemma 15** (Informal). *Let $G = (V, E) \in \mathrm{supp}(\mathcal{G})$, and let $\mathcal{U}_1, \mathcal{U}_2$ be any 2-level (ϵ, ϕ)-RED of G. Then, there are at most $n/10$ vertices in S that are non-isolated vertices[3] in $\mathcal{U}_2$. Moreover, at least a $2/3$ fraction of important edges are not in $E \setminus \mathcal{U}_2$, i.e. a $2/3$ fraction of important edges are inside clusters of $\mathcal{U}_2$.*

Proof sketch. The number of crossing edges in the first level decomposition is $O(\epsilon dn)$, which is much less than n since $d \ll \frac{1}{\epsilon}$. This means that most of the vertices are isolated vertices in the second level. Moreover, by Lemma 14, most of the important edges are crossing edges in the first level decomposition. Among these $\Theta(dn/m)$ edges, at most $O(\epsilon^2 dn)$ edges can be crossing edges in the second level decomposition. Recalling that $m \ll \frac{1}{\epsilon^2}$, we see that most of the important edges are not crossing edges in the second level decomposition. ◀

3.2 Lower Bound via Communication Complexity

Our streaming lower bound will be proven in the two-player one-way communication model. In this setting, Alice gets the edges in $G[S]$ and $G[T]$, and Bob gets the edges between S and T. We prove that in order to give a two-level RED, Alice needs to send $\Omega(dn)$ bits of information to Bob.

The high level idea is the following. Note that the identity of the important vertices can be only revealed by edges given to Bob. Thus, given Alice's input, every vertex in S has the same probability to be an important vertex, which means that every edge in S has the same

[3] We call a vertex v non-isolated in a decomposition $\mathcal{U}$ if $\mathcal{U}$ puts v in a cluster with other vertices, i.e. v does not constitute a singleton cluster in $\mathcal{U}$.

probability to be an important edge. Therefore, in order to make sure that Bob recovers most of the important edges (which is morally equivalent to computing a two-level RED, as suggested by Figure 2), Alice needs to send most of the edges in S to Bob, which is $\Omega(dn)$.

To make the above idea concrete, we consider a communication problem where Alice is given a graph, and Bob is asked to output a not too large set of pairs that contains a good fraction of the edges of Alice's input. We first reduce this new problem to the two-level RED problem. Then, we will prove a communication complexity lower bound for this problem.

▶ **Definition 16** (Informal). *In the communication problem* RECOVER*, Alice's input is a graph $G' = (S, E')$ where $|S| = n/2$, and Bob's output is a set of pairs of vertices $F \subseteq \binom{S}{2}$ that must satisfy $|F| \leq nm/10$ and $|F \cap E'| \geq \Omega(|E'|)$, i.e. at least a constant fraction of the edges in G' are in F.*

Now, the idea is to plant Alice's input for RECOVER into our instance from Definition 12. The input distribution for RECOVER is sampled from the distribution $\mathcal{G}'$. In other words, the distribution of Alice's input is the same as the left-hand side part of $G \sim \mathcal{G}$ (i.e. the subgraph $G[S]$). Then, in the reduction, Alice gets the edges of $G[S]$ and $G[T]$ while Bob gets the edges between S and T. By virtue of the discussion in the previous section, we expect a RED of G to allow Bob to recover many important edges in $G[S]$. Hence, Bob could simulate the RED algorithm for all the $m/2$ possible choices of the random index $K \sim [m/2]$ that defines the important edges (see Definition 12): in this way, the k-th RED should reveal information about Alice's edges that are incident on the vertices $\{s_{i,k}\}_i$. Therefore, by varying k over $[m/2]$, Bob should obtain information about all the edges in Alice's graph. More precisely, the reduction is the following.

▶ Reduction (Informal). Let $\mathcal{A}$ be a deterministic streaming algorithm for computing a 2-level (ϵ, ϕ)-RED. Alice, given her input graph $G' = (S, E')$ generated by $\mathcal{G}'$, feeds her edges E' to $\mathcal{A}$, together with the fixed edges of $G[T]$. Then, she sends the memory state of $\mathcal{A}$ to Bob. Upon receiving the message, Bob makes $m/2$ copies of $\mathcal{A}$ and initialises them to the memory state he received from Alice. Call these copies $\mathcal{A}_1, \ldots, \mathcal{A}_{m/2}$. Next, for each $k \in [m/2]$, call $G_k = (V, E_k)$ the graph we obtain from $\mathcal{G}$ when $K = k$ and the left-hand side $G_k[S]$ is exactly Alice's input G' (so that the vertices $\{s_{i,k}\}_i$ are the important vertices in G_k, see Definition 12 and Definition 13). Then, Bob feeds the edges $E_k(S, T)$ to $\mathcal{A}_k$. Let then $\mathcal{U}_1^k, \mathcal{U}_2^k$ be the RED output by $\mathcal{A}_k$. Bob finally constructs his output set F as follows: for each $k \in [m/2]$, add the pair $\{s, s_{i,k}\}$ to F for every $i \in [n/m]$ and every vertex $s \in S_i$ that is not isolated in $\mathcal{U}_2^k$. ⌟

By Lemma 15, the number of non-isolated vertices in the second-level decomposition is at most $n/10$. Hence, the total number of pairs added to F is at most $nm/10$, thus satisfying the first requirement of Definition 16. Moreover, by Lemma 15, at least a $2/3$ fraction of the important edges are not crossing edges in the second level. This means that for each k, at least a $2/3$ fraction of the edges that are incident on $s_{i,k}$ is added to F. In turn, this implies that F contains at least a constant fraction of the edges in E', thus satisfying the second requirement of Definition 16. More precisely, one can prove the following.

▶ **Lemma 17** (Informal). *If there is a deterministic L-bit space streaming algorithm $\mathcal{A}$ that computes a 2-level (ϵ, ϕ)-RED with constant probability over inputs $G \sim \mathcal{G}$, then there is a deterministic protocol $\mathcal{R}$ that solves* RECOVER *with constant probability over inputs $G' \sim \mathcal{G}'$. The communication complexity of $\mathcal{R}$ is at most L.*

The final component of the proof is the communication complexity lower bound for RECOVER.

▶ **Lemma 18** (Informal). *The one-way communication complexity of solving* RECOVER *with constant probability over inputs sampled from* $\mathcal{G}'$ *is* $\Omega(dn)$.

Proof sketch. Roughly speaking, we show that the posterior distribution of the input conditioned on the output is shifted away from its prior distribution.

Recall that when the input $G' = (S, E')$ is sampled from $\mathcal{G}'$, the graph is a disjoint union of n/m random graphs with $m/2$ vertices each and degree $\approx d$. There are roughly

$$\left(\binom{m/2}{d}^{m/2}\right)^{n/m} \approx \left(\frac{m}{2d}\right)^{dn/2}$$

possible inputs in total and the information complexity is $\Omega(dn)$. Conditioning on the the output F of a correct protocol for RECOVER, the number of possible inputs is greatly decreased. In particular, to determine the input E', we need to select a constant fraction, say $2/3$ for example, of the pairs from F, and select the rest of the edges (a $1/3$ fraction, in our example) arbitrarily. Since $|F|$ is at most $nm/10$, $|E'| = \Theta(dn)$, and $\binom{S}{2} \approx nm/2$, the total number of possible inputs is then roughly

$$\binom{nm/10}{2/3 \cdot dn} \cdot \binom{nm/2}{1/3 \cdot dn} \approx \left(\frac{3m}{20d}\right)^{1/3 \cdot dn} \cdot \left(\frac{3m}{2d}\right)^{1/6 \cdot dn} < \left(\frac{m}{3d}\right)^{dn/2}.$$

This means the information complexity of the input is decreased by a constant factor, which means that the protocol needs to communicate $\Omega(dn)$ bits of information. ◀

Finally, one can conclude the main result Theorem 11 combining Lemma 17 and Lemma 18. The formal proofs and definitions are deferred to the full version of this paper.

References

1. Arpit Agarwal, Sanjeev Khanna, Huan Li, and Prathamesh Patil. Sublinear algorithms for hierarchical clustering. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL: `http://papers.nips.cc/paper_files/paper/2022/hash/16466b6c95c5924784486ac5a3feeb65-Abstract-Conference.html`.
2. AmirMahdi Ahmadinejad, John Peebles, Edward Pyne, Aaron Sidford, and Salil P. Vadhan. Singular value approximation and sparsifying random walks on directed graphs. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 846–854. IEEE, 2023. `doi:10.1109/FOCS57990.2023.00054`.
3. Kook Jin Ahn and Sudipto Guha. Graph sparsification in the semi-streaming model. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th Internatilonal Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 328–338. Springer, 2009. `doi:10.1007/978-3-642-02930-1_27`.
4. Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 459–467. SIAM, 2012. `doi:10.1137/1.9781611973099.40`.
5. Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In Michael Benedikt, Markus Krötzsch, and Maurizio Lenzerini, editors, *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 5–14. ACM, 2012. `doi:10.1145/2213556.2213560`.

6 Vedat Levi Alev, Nima Anari, Lap Chi Lau, and Shayan Oveis Gharan. Graph clustering using effective resistance. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pages 41:1–41:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ITCS.2018.41.

7 András A. Benczúr and David R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM J. Comput.*, 44(2):290–319, 2015. doi:10.1137/070705970.

8 Yi-Jun Chang, Seth Pettie, Thatchaphol Saranurak, and Hengjie Zhang. Near-optimal distributed triangle enumeration via expander decompositions. *J. ACM*, 68(3):21:1–21:36, 2021. doi:10.1145/3446330.

9 Yi-Jun Chang and Thatchaphol Saranurak. Improved distributed expander decomposition and nearly optimal triangle enumeration. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 66–73. ACM, 2019. doi:10.1145/3293611.3331618.

10 Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Comput. Complex.*, 15(2):94–114, 2006. doi:10.1007/S00037-006-0210-9.

11 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 612–623. IEEE, 2022. doi:10.1109/FOCS54457.2022.00064.

12 Yu Chen, Michael Kapralov, Mikhail Makarov, and Davide Mazzali. On the streaming complexity of expander decomposition, 2024. doi:10.48550/arXiv.2404.16701.

13 Yu Chen, Sanjeev Khanna, and Huan Li. On weighted graph sparsification by linear sketching. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 474–485. IEEE, 2022. doi:10.1109/FOCS54457.2022.00052.

14 Timothy Chu, Yu Gao, Richard Peng, Sushant Sachdeva, Saurabh Sawlani, and Junxing Wang. Graph sparsification, spectral sketches, and faster resistance computation, via short cycle decompositions. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 361–372. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00042.

15 Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1158–1167. IEEE, 2020. doi:10.1109/FOCS46700.2020.00111.

16 Arnold Filtser, Michael Kapralov, and Mikhail Makarov. Expander decomposition in dynamic streams. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*, volume 251 of *LIPIcs*, pages 50:1–50:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ITCS.2023.50.

17 Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 71–80. ACM, 2011. doi:10.1145/1993636.1993647.

18 Gramoz Goranci, Monika Henzinger, Danupon Nanongkai, Thatchaphol Saranurak, Mikkel Thorup, and Christian Wulff-Nilsen. Fully dynamic exact edge connectivity in sublinear time. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 70–86. SIAM, 2023. doi:10.1137/1.9781611977554.CH3.

19 Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2212–2228. SIAM, 2021. doi:10.1137/1.9781611976465.132.

20 Venkatesan Guruswami, Jun-Ting Hsieh, Pravesh K. Kothari, and Peter Manohar. Efficient algorithms for semirandom planted csps at the refutation threshold. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 307–327. IEEE, 2023. doi:10.1109/FOCS57990.2023.00026.

21 Yiding Hua, Rasmus Kyng, Maximilian Probst Gutenberg, and Zihang Wu. Maintaining expander decompositions via sparse cuts. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 48–69. SIAM, 2023. doi:10.1137/1.9781611977554.CH2.

22 Ravi Kannan, Santosh S. Vempala, and Adrian Vetta. On clusterings - good, bad and spectral. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 367–377. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892125.

23 Michael Kapralov, Robert Krauthgamer, Jakab Tardos, and Yuichi Yoshida. Towards tight bounds for spectral sparsification of hypergraphs. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 598–611. ACM, 2021. doi:10.1145/3406325.3451061.

24 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 561–570. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.66.

25 Michael Kapralov, Aida Mousavifar, Cameron Musco, Christopher Musco, Navid Nouri, Aaron Sidford, and Jakab Tardos. Fast and space efficient spectral sparsification in dynamic streams. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1814–1833. SIAM, 2020. doi:10.1137/1.9781611975994.111.

26 David R. Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 21–30. ACM/SIAM, 1993. URL: http://dl.acm.org/citation.cfm?id=313559.313605.

27 David R. Karger. Random sampling in cut, flow, and network design problems. In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 648–657. ACM, 1994. doi:10.1145/195058.195422.

28 Jason Li. Deterministic mincut in almost-linear time. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 384–395. ACM, 2021. doi:10.1145/3406325.3451114.

29 Jason Li, Danupon Nanongkai, Debmalya Panigrahi, and Thatchaphol Saranurak. Near-linear time approximations for cut problems via fair cuts. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 240–275. SIAM, 2023. doi:10.1137/1.9781611977554.CH10.

30 Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2616–2635. SIAM, 2019. doi:10.1137/1.9781611975482.162.

31 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 81–90. ACM, 2004. doi:10.1145/1007352.1007372.

32 Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM J. Comput.*, 40(4):981–1025, 2011. doi:10.1137/08074489X.

33 Luca Trevisan. Approximation algorithms for unique games. *Theory Comput.*, 4(1):111–128, 2008. doi:10.4086/TOC.2008.V004A005.

Lower Bounds on 0-Extension with Steiner Nodes

Yu Chen
EPFL, Lausanne, Switzerland

Zihan Tan
Rutgers University, Piscataway, NJ, USA

Abstract

In the 0-*Extension problem*, we are given an edge-weighted graph $G = (V, E, c)$, a set $T \subseteq V$ of its vertices called terminals, and a semi-metric D over T, and the goal is to find an assignment f of each non-terminal vertex to a terminal, minimizing the sum, over all edges $(u, v) \in E$, the product of the edge weight $c(u, v)$ and the distance $D(f(u), f(v))$ between the terminals that u, v are mapped to. Current best approximation algorithms on 0-Extension are based on rounding a linear programming relaxation called the *semi-metric LP relaxation*. The integrality gap of this LP, is upper bounded by $O(\log |T|/ \log \log |T|)$ and lower bounded by $\Omega((\log |T|)^{2/3})$, has been shown to be closely related to the quality of cut and flow vertex sparsifiers.

We study a variant of the 0-Extension problem where Steiner vertices are allowed. Specifically, we focus on the integrality gap of the same semi-metric LP relaxation to this new problem. Following from previous work, this new integrality gap turns out to be closely related to the quality achievable by cut/flow vertex sparsifiers with Steiner nodes, a major open problem in graph compression. We show that the new integrality gap stays superconstant $\Omega(\log \log |T|)$ even if we allow a super-linear $O(|T| \log^{1-\varepsilon} |T|)$ number of Steiner nodes.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

Keywords and phrases Graph Algorithms, Zero Extension, Integrality Gap

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.47

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/pdf/2401.09585.pdf`

Funding *Zihan Tan*: Supported by a grant to DIMACS from the Simons Foundation (820931).

Acknowledgements We would like to thank Julia Chuzhoy for many helpful discussions. We want to thank Arnold Filtser for pointing to us some previous works on similar problems.

1 Introduction

In the 0-*Extension* problem (0-Ext), we are given an undirected edge-weighted graph $G = (V, E, c)$, a set $T \subseteq V$ of its vertices called *terminals*, and a metric D on terminals, and the goal is to find a mapping $f : V \to T$ that maps each vertex to a terminal in T, such that each terminal is mapped to itself (i.e., $f(t) = t$ for all $t \in T$), and the sum $\sum_{(u,v)\in E} c(u, v) \cdot D(f(u), f(v))$ is minimized.

The 0-Ext problem was first introduced by Karzanov [22]. It is a generalization of the *multi-way cut* problem (by setting $D(t, t') = 1$ for all pairs $t, t' \in T$) [15, 7, 17, 5, 2, 6, 4], and a special case of the *metric labeling* problem [23, 10, 3, 20, 14]. Călinescu, Karloff and Rabani [8] gave the first approximation algorithm for 0-Ext, achieving a ratio of $O(\log |T|)$, by rounding the solution of a *semi-metric LP relaxation* (LP-Metric), which is presented below.


$$\text{(LP-Metric)} \qquad \begin{aligned} \text{minimize} \quad & \textstyle\sum_{(u,v)\in E} c(u,v)\cdot\delta(u,v) \\ s.t. \quad & (V,\delta) \text{ is a semi-metric space} \\ & \delta(t,t') = D(t,t'), \quad \forall t,t' \in T \end{aligned}$$

Fakcharoenphol, Harrelson, Rao and Talwar [16] later gave a modified rounding algorithm on the same LP, improving the ratio to $O(\log|T|/\log\log|T|)$, which is the current best-known approximation. On the other hand, this LP was shown to have integrality gap $\Omega(\sqrt{\log|T|})$ [8], and this was recently improved to $\Omega((\log|T|)^{2/3})$ by Schwartz and Tur [31]. Another LP relaxation called *earthmover distance relaxation* (LP-EMD) was considered by Chekuri, Khanna, Naor and Zosin [10] and utilized to obtain an $O(\log|T|)$-approximation of the metric labeling problem (and therefore also the 0-Ext problem). It has been shown [21] by Karloff, Khot, Mehta and Rabani that this LP relaxation has an integrality gap $\Omega(\sqrt{\log|T|})$. Manokaran, Naor, Raghavendra and Schwartz [28] showed that the integrality gap of this LP relaxation leads to a hardness of approximation result, assuming the Unique Game Conjecture.

In addition to being an important problem on its own, the 0-Ext problem and its two LP relaxations are also closely related to the construction of *cut/flow vertex sparsifiers*, a central problem in the paradigm of graph compression. Given a graph G and a set $T \subseteq V(G)$ of terminals, a cut sparsifier of G with respect to T is a graph H with $V(H) = T$, such that for every partition (T_1, T_2) of T, the size of the minimum cut separating T_1 from T_2 in G and the size of the minimum cut separating T_1 from T_2 in H, are within some small factor q, which is also called the *quality* of the sparsifier[1]. Moitra [29] first showed that every graph with k terminals admits a cut sparsifier with quality bounded by the *integrality gap* of its LP-Metric (hence $O(\log k/\log\log k)$). Later on, Leighton and Moitra [26], and Makarychev and Makarychev [27] concurrently obtained the same results for flow sparsifiers, and then Charikar, Leighton, Li and Moitra [9] showed that the best flow sparsifiers can be computed by solving an LP similar to LP-EMD. On the lower bound side, it was shown after a line of work [26, 9, 27] that there exist graphs with k terminals whose best flow sparsifier has quality $\tilde{\Omega}(\sqrt{\log k})$.

A major open question on vertex sparsifiers is:

Q1. Can better quality sparsifiers be achieved by allowing a small number of Steiner vertices?

In other words, what if we no longer require that the sparsifier H only contain terminals, but just require that H contain all terminals and its size be bounded by some function f on the number of terminals (for example, $f(k) = 2k, k^2$ or even 2^k)? Chuzhoy [13] constructed $O(1)$-quality cut/flow sparsifiers with size dependent on the number of terminal-incident edges in G. Andoni, Gupta and Krauthgamer [1] showed the construction for $(1+\varepsilon)$-quality flow sparsifiers for quasi-bipartite graphs. For general graphs, they constructed a sketch of size $f(k,\varepsilon)$ that stores all feasible multicommodity flows up to a factor of $(1+\varepsilon)$, raising the hope for a special type of $(1+\varepsilon)$-quality flow sparsifier, called contraction-based flow sparsifiers, of size $f(k,\varepsilon)$ for general graphs, which was recently invalidated by Chen and Tan [12], who showed that contraction-based flow sparsifiers whose size are bounded by any function $f(k)$ must have quality $1+\Omega(1)$. But it is still possible for such flow sparsifiers with constant quality and finite size to exist. Prior to this work, Krauthgamer and Mosenzon [24] showed that there exist 6-terminal graphs G whose quality-1 flow sparsifiers must have an arbitrarily large size.

[1] flow sparsifiers has a slightly more technical definition, which can be found in [19, 26, 13, 1].

Given the concrete connection between the 0-Ext problem and cut/flow sparsifiers, it is natural to ask a similar question for 0-Ext:

Q2. Can better approximation of 0-Ext *be achieved by allowing a small number of Steiner vertices?*

In this paper, we formulate and study the following variant of the 0-Ext problem, which we call the *0-Extension with Steiner Nodes* problem (0EwSN). (We note that a similar variant was mentioned in [1], and we provide a comparison between them in more detail in Appendix A.) We are also given a function $f : \mathbb{Z} \to \mathbb{Z}$ with $f(k) \geq k$ for all $k \in \mathbb{Z}$, this should be the total number of Steiner vertices.

0-Extension with Steiner Nodes

In an instance of 0EwSN(f), the input consists of an edge-weighted graph $G = (V, E, c)$, a subset $T \subseteq V$ of k vertices, that we call *terminals*, and a metric D on terminals in T, which is exactly the same as 0-Ext. A solution to the instance (G, T, D) consists of

- a partition $\mathcal{F}$ of V with $|\mathcal{F}| \leq f(k)$, such that distinct terminals of T belong to different sets in $\mathcal{F}$; we call sets in $\mathcal{F}$ *clusters*, and for each vertex $u \in V$, we denote by $F(u)$ the cluster in $\mathcal{F}$ that contains it;
- a semi-metric δ on the clusters in $\mathcal{F}$, such that for each pair $t, t' \in T$, $\delta(F(t), F(t')) = D(t, t')$.

We define the *cost* of a solution $(\mathcal{F}, \delta)$ as $\mathsf{cost}(\mathcal{F}, \delta) = \sum_{(u,v)\in E} c(u, v) \cdot \delta(F(u), F(v))$, and its *size* as $|\mathcal{F}|$. The goal is to compute a solution $(\mathcal{F}, \delta)$ with size at most $f(k)$ and minimum cost.

The difference between 0EwSN(f) and 0-Ext is that, instead of enforcing every vertex to be mapped to a terminal, in 0EwSN(f) we allow vertices to be mapped to $(f(k) - k)$ non-terminals (or Steiner nodes), which are the clusters in $\mathcal{F}$ that do not contain terminals. We are also allowed to manipulate the distances between these non-terminals, conditioned on not destroying the induced metric D on terminals. Clearly, when $f(k) = k$, the 0EwSN(f) problem degenerates to the 0-Ext problem.

It is easy to see that (LP-Metric) is still an LP relaxation for 0EwSN(f), as each solution $(\mathcal{F}, \delta)$ to 0EwSN naturally corresponds to a semi-metric δ' on V (where we can set $\delta'(u, u') = \delta(F(u), F(u'))$ for all pairs $u, u' \in V$). Denote by $\mathsf{IG}_f(k)$ the worst integrality gap for (LP-Metric) to any 0EwSN(f) instance with at most k terminals. In fact, similar to the connection between the integrality gap of (LP-Metric) and the quality achievable by flow sparsifiers [29, 26], it was recently shown by Chen and Tan [12] that the value of $\mathsf{IG}_f(k)$ is also closely related to the quality achievable by flow sparsifiers with Steiner nodes. Specifically, for any function f, every graph G with k terminals has a quality-$\big((1 + \varepsilon) \cdot \mathsf{IG}_f(k)\big)$ flow sparsifier with size bounded by $(f(k))^{(\log k/\varepsilon)^{k^2}}$. This means that any positive answer to question Q2 (by proving that $\mathsf{IG}_f(k) = o(\log k/ \log\log k)$ for some f) also gives a positive answer to question Q1.

This makes it tempting to study the 0EwSN problem. Specifically, can we prove any better-than-$O(\log k/ \log\log k)$ upper bound for $\mathsf{IG}_f(k)$, for any function f? To the best of our knowledge, no such bound is known for any f, leaving the problem wide open. In fact, no non-trivial lower bound on $\mathsf{IG}_f(k)$ is known for even very small function like $f(k) = O(k)$.

1.1 Our Results

In this paper, we make a first step in investigating the value of $\mathsf{IG}_f(k)$, by giving a superconstant lower bound on $\mathsf{IG}_f(k)$ for near-linear functions f. Our main result is summarized in the following theorem.

▶ **Theorem 1.** *For any $0 < \varepsilon < 1$ and any size function $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ with $f(k) = O(k \log^{1-\varepsilon} k)$, the integrality gap of the LP-relaxation (*LP-Metric*) is $\mathsf{IG}_f(k) = \Omega(\varepsilon \log \log k)$.*

We remark that our lower bound for the integrality gap of $\mathsf{0EwSN}(f)$ does not imply a size lower bound for $O(\log \log k)$-quality flow sparsifiers. However, if the same lower bound can be proved for a slightly generalized version of $\mathsf{0EwSN}(f)$, that was proposed in [1] and analyzed in [12], then it will imply an $\Omega(k \log^{1-\varepsilon} k)$ size lower bound for $O(\log \log k)$-quality flow sparsifiers. We provide a detailed discussion in Appendix A.

1.2 Technical Overview

We now discuss some high-level ideas in the proof of Theorem 1. Given any k, we will construct an unweighted graph G on n vertices (where $n \approx k \log k$) and k terminals, and show that any solution of the 0EwSN problem with size $O(k \log^{1-\varepsilon} k)$ has cost lower bounded by $\Omega(\log \log k)$ times the number of edges in G.

Our hard instance is a constant degree expander (with an arbitrary set of its k vertices as terminals). There are two main reasons to choose such a graph. First, in previous work [8] for proving the $\Omega(\sqrt{\log k})$ integrality gap lower bound for the 0-Extension problem, a graph called "expander with tails" was used. Though the tails in their construction appear useless for our purpose, as we allow a super-linear number of Steiner vertices which easily accomodate a single-edge tail for each terminal, the expander graph turns out to still be the core structure that is hard to compress. Second, it has been shown [3] that 0-Extension problem on minor-free graphs has integrality gap $O(1)$, so our hard example has to contain large cliques as minor, which makes expanders, a favorable choice. For technical reasons, we need some additional properties like Hamiltonicity and high girth. See Section 3.1 for more details.

Next we want to lower bound the cost of an 0EwSN solution of size $O(k \log^{1-\varepsilon} k)$. We first consider a special type of solutions where each cluster is mapped to a vertex on the graph. Recall that a solution consists of a partition $\mathcal{F}$ of $V(G)$ into clusters and a metric δ on clusters in $\mathcal{F}$. Specifically, in this special type of solutions, we require that each cluster $F \in \mathcal{F}$ corresponds to a distinct vertex v_F (called its *center*) in G, and for all pairs F, F' the metric $\delta(F, F')$ coincides with the shortest-path distance between $v_F, v_{F'}$ in G. We show that all special solutions have cost $\Omega(n \cdot \varepsilon \log \log k)$. Intuitively, as the graph is a constant degree expander, every center v_F is within distance $(\varepsilon/100) \cdot \log \log k$ to at most $(\log k)^{\varepsilon/10}$ other centers, but its cluster F contains $(\log k)^{\varepsilon}$ vertices on average and so it has $\Omega((\log k)^{\varepsilon})$ inter-cluster edges. As we measure the distance between clusters using their centers, only a small fraction of the inter-cluster edges will cost less than $(\varepsilon/100) \cdot \log \log k$, making the total new edge length $\Omega(n \cdot \varepsilon \log \log k)$ (as the number of inter-cluster edges in a balanced expander partition is $\Omega(n)$). Careful calculations are needed to turn these informal arguments into a rigorous proof. See Section 3.2 for more details.

Afterwards, we show that general solutions can actually be reduced to the special type of solutions considered in the first step, losing only an $O(1)$ factor in its cost. In fact, it has been recently shown [12] that, to analyze the cost of any 0EwSN instance, it suffices to consider 0EwSN solutions whose metric δ is embeddable into a geodesic structure of the

terminal-induced shortest path distance metric called *tight span*. Our main contribution here, on a conceptual level, is showing that *for a graph with high girth, its tight span structure locally coincides with the graph structure itself*. In this sense, we can compare any solution to some special solution considered in Step 1. On a global level, in such a comparison it turns out that we will lose a factor which is approximately the *diameter-girth ratio*, which we can manage to get to $O(1)$ with an additional short-cycle-removing step in the construction of the expander. We believe this diameter-girth ratio quantifies how "local" a graph structure is, and should be of independent interest to other graph problems on shortest-path distances. To carry out the technical steps, we employ a notion called *continuiazation* of a graph recently studied in [11]. See Section 3.3 for more details.

2 Preliminaries

By default, all logarithms are to the base of 2.

Let $G = (V, E, \ell)$ be an edge-weighted graph, where each edge $e \in E$ has weight (or *length*) ℓ_e. For a vertex $v \in V$, we denote by $\deg_G(v)$ the degree of v in G. For each pair $S, T \subseteq V$ of disjoint subsets, we denote by $E_G(S, T)$ the set of edges in G with one endpoint in S the other endpoint in T. For a pair v, v' of vertices in G, we denote by $\mathsf{dist}_G(v, v')$ (or $\mathsf{dist}_\ell(v, v')$) the shortest-path distance between v and v' in G. We define the *diameter* of G as $\mathsf{diam}(G) = \max_{v,v' \in V} \{\mathsf{dist}_G(v, v')\}$, and we define the *girth* of G, denoted by $\mathsf{girth}(G)$, as the minimum weight of any cycle in G. We may omit the subscript G in the above notations when the graph is clear from the context.

Given a graph G, its *conductance* is defined as

$$\Phi(G) = \min_{S \subseteq V, S \neq \emptyset, S \neq V} \left\{ \frac{|E_G(S, V \setminus S)|}{\min\left\{\sum_{v \in S} \deg_G(v), \sum_{v \notin S} \deg_G(v)\right\}} \right\}.$$

We say that G is a *ϕ-expander* iff $\Phi(G) \geq \phi$.

3 Proof of Theorem 1

In this section we prove the main result Theorem 1, which shows that, when we only have $O(k \log^{1-\varepsilon} k)$ Steiner nodes, the best ratio we can get is $\Omega(\varepsilon \log \log k)$. We begin by describing the hard instance in Section 3.1, which is essentially a high-girth expander with a subset of vertices designated as terminals. Then in Section 3.2 we show that a special type of solutions may not have small cost. Finally, in Section 3.3 we generalize the arguments in Section 3.2 to analyze an arbitrary solution, completing the proof of Theorem 1. Some technical details in Section 3.3 are deferred to Section 3.4.

3.1 The Hard Instance

Let k be a sufficiently large integer. Let $n > k$ be an integer such that $k = \left\lceil \frac{n \log \log n}{\log n} \right\rceil$. Let V be a set of n vertices. Let Σ be the set of all permutations on V. For a permutation $\sigma \in \Sigma$, we define its corresponding edge set $E_\sigma = \{(v, \sigma(v)) \mid v \in V\}$.

We now define the hard instance (G, T, D). Graph G is constructed in two steps. In the first step, we construct an auxiliary graph G'. Its vertex set is V. Its edge set is obtained as follows. We sample three permutations $\sigma_1, \sigma_2, \sigma_3$ uniformly at random from Σ, and then let $E(G') = E_{\sigma_1} \cup E_{\sigma_2} \cup E_{\sigma_3}$. In the second step, we remove all short cycles in G' to obtain G. Specifically, we first compute a breath-first-search tree τ starting from an arbitrary vertex

of G'. We then iteratively modify G' as follows. While G' contains a cycle C of length at most $(\log n)/100$, we find an edge of $C \setminus \tau$ (note that such an edge must exist, as τ is a tree), and remove it from G'. We continue until G' no longer contains cycles of length at most $(\log n)/100$. We denote by G the resulting graph. The terminal set T is an arbitrary subset of V with size k. For each edge $e \in E(G)$, its weight $c(e)$ is defined to be 1, and its *length* ℓ_e is also defined to be 1. The metric D on the set T of terminals is simply defined to be the shortest-path distance (in G) metric on T induced by edge length $\{\ell_e\}_{e \in E(G)}$.

We next show some basic properties of the graphs G' and G. We start with the following observations and propositions.

▶ **Observation 2.** *G' is a 6-regular graph, so $|E(G)| \leq |E(G')| \leq 3n$.*

▶ **Observation 3.** $\mathsf{girth}(G) \geq (\log n)/100$.

▶ **Proposition 4.** *The probability that $|E(G') \setminus E(G)| \geq n^{0.3}$ is at most $O(n^{-0.2})$.*

Proof. Let $v_1, \ldots, v_L$ be a sequence of $L \leq (\log n)/100$ distinct vertices of V. We now show that the probability that the cycle $(v_1, \ldots, v_L, v_1)$ exists in $E(G')$ is at most $\big(6/(n-L)\big)^L$. Indeed, to realize the cycle edge (v_i, v_{i+1}), for some $\ell \in \{1,2,3\}$, $\sigma_\ell(v_i) = v_{i+1}$ or $\sigma_\ell(v_{i+1} = v_i$, where for convenience we say $v_{L+1} = v_1$. There are 6 possible events. In order to form the cycle, we need to form L edges, and each edge has 6 possible events, which means there are at most 6^L ways to form the cycle in total. Consider any possible way, we have $\ell_1 \ldots, \ell_L \in \{1,2,3\}$ and $j_1, \ldots, j_L \in \{0,1\}$ such that for any index $1 \leq i \leq L$, we have $\sigma_{\ell_i}(v_{i+j_i}) = v_{i+1-j_i}$. Let $\mathcal{E}_i$ denote this event, we have $\Pr[\mathcal{E}_i | \mathcal{E}_1, \ldots, \mathcal{E}_{i-1}] \leq 1/(n-i)$. Thus the probability that all events $\mathcal{E}_i$ happen is at most $1/(n-L)^L$. Applying union bound on all the ways to form the cycle, the probability that the cycle exists in $E(G')$ is at most $\big(6/(n-L)\big)^L$.

Therefore, the expected number of cycles in G' with length at most $(\log n)/100$ is at most

$$\sum_{3 \leq L \leq (\log n)/100} \frac{n(n-1)\cdots(n-L+1)}{2L \cdot \left(\frac{n-L}{6}\right)^L} \leq \sum_{3 \leq L \leq (\log n)/100} \frac{6^L}{2L} \cdot \left(1 + \frac{L}{n-L}\right)^L$$
$$\leq \sum_{3 \leq L \leq (\log n)/100} 6^L \leq n^{0.1}.$$

Therefore, from Markov Bound, with probability $n^{-0.2}$, the number of cycles in G' with length at most $(\log n)/100$ is at most $n^{0.3}$. Note that we delete at most one edge per each cycle, so $|E(G') \setminus E(G)|$ is less than the number of cycles in G' with length at most $(\log n)/100$, the proposition follows. ◀

We use the following previous results on the conductance and the Hamiltonicity of G'.

▶ **Lemma 5** ([30])**.** *With probability $1 - o(1)$, $\Phi(G') = \Omega(1)$.*

▶ **Corollary 6.** *With probability $1 - o(1)$, the diameter of G is at most $O(\log n)$.*

Proof. From the construction of G, G contains a BFS tree of G', so the diameter of G is at most twice the diameter of G'. Therefore, it suffices to show that, if $\Phi(G') \geq \Omega(1)$, then the diameter of graph G' is at most $O(\log n)$, which we do next.

Let v be an arbitrary vertex of G'. For each integer t, we define the set $B_t = \{v' \mid \mathsf{dist}(v, v') \leq t\}$, and $\alpha_t = \sum_{v' : \mathsf{dist}_{G'}(v,v') \leq t} \deg(v')$, namely the sum of degrees of all vertices in B_t.

Denote $t^* = \max\{t \mid \alpha_t \leq |E(G')|\}$. Note that, for each $1 \leq t \leq t^*$, as $\Phi(G') \geq \Omega(1)$, $|E(B_t, V \setminus B_t)| \geq \Omega(\alpha_t)$. Therefore,

$$\alpha_{t+1} \geq \alpha_t + \sum_{v' \in B_{t+1} \setminus B_t} \deg(v) \geq \alpha_t + |E(B_t, V \setminus B_t)| \geq \alpha_t \cdot (1 + \Omega(1)).$$

It follows that $t^* \leq O(\log n)$. Therefore, for any pair $v, v' \in V$, the set of vertices that are at distance at most $t^* + 1$ from v must intersect the set of vertices that are at distance at most $t^* + 1$ from v', as otherwise the sum of degrees in all vertices in these two sets is greater than $2|E(G')|$, a contradiction. Consequently, the diameter of G' is at most $2t^* + 2 \leq O(\log n)$. ◀

▶ **Lemma 7** ([18]). *With probability $1 - o(1)$, the subgraph of G' induced by edges of $E_{\sigma_1} \cup E_{\sigma_2}$ is Hamiltonian.*

Now if we consider the semi-metric LP relaxation (LP-Metric) of this instance (G, T, D), then clearly the graph itself gives a solution δ to (LP-Metric). Specifically, $\delta(u, u') = \mathsf{dist}_\ell(u, u')$, where $\mathsf{dist}_\ell(\cdot, \cdot)$ the shortest-path (in G) distance metric on V induced by the lengths $\{\ell_e\}_{e \in E(G)}$. Such a solution has cost $|E(G)| = O(n)$ (as all edges have weight $c(e) = 1$). Therefore, in order to prove Theorem 1, it suffices to show that any solution $(\mathcal{F}, \delta)$ with size $O(k \log^{1-\varepsilon} k)$ has cost at least $\Omega(\varepsilon n \log\log n) = \Omega(\varepsilon n \log\log k)$.

Observe that, the graph G constructed above is essentially a bounded-degree high-girth expander, which is similar to the hard instance used in [26] for proving the $\Omega(\log\log k)$ quality lower bound for flow vertex sparsifier (without Steiner nodes). However, our proof in the following subsections takes a completely different approach from the approach in [26].

3.2 Proof of Theorem 1 for Canonical Solutions

In this subsection, we prove the cost lower bound for a special type of solutions to the 0EwSN(f) instance (G, T, D) which we call canonical. Specifically, a solution is canonical $(\mathcal{F}, \delta)$ if

- each cluster $F \in \mathcal{F}$ corresponds to a distinct vertex of V, we call this vertex the *center* of F, denote as $v(F)$ (note however that $v(F)$ does not necessarily lie in F). For each terminal $t \in T$, the unique cluster $F \in \mathcal{F}$ that contains t, $v(F) = t$; and
- for each pair F, F' of clusters in $\mathcal{F}$, $\delta(F, F') = \mathsf{dist}_G(v(F), v(F'))$.

In this subsection, we show that, with high probability, any canonical solution of size $o(n/\log^\varepsilon n)$ has cost $\Omega(\varepsilon n \log\log n)$.

Consider now a canonical solution $(\mathcal{F}, \delta)$ to the instance. We say that $F \in \mathcal{F}$ is *large* iff $|F| \geq n^{0.1}$, otherwise we say it is *small*. We distinguish between the following cases, depending on the total size of large clusters.

Recall that $\mathsf{cost}(\mathcal{F}, \delta) = \sum_{(u,u') \in E(G)} \delta(F(u), F(u'))$, where $F(u)$ ($F(u')$, resp.) is the unique cluster in $\mathcal{F}$ that contains u (u', resp.). We call $\delta(F(u), F(u'))$ the *contribution* of edge (u, u') to the cost $\mathsf{cost}(\mathcal{F}, \delta)$.

Case 1: The total size of large clusters is at most $0.1n$

As the solution $(\mathcal{F}, \delta)$ is canonical,

$$\delta(F(u), F(u')) = \mathsf{dist}_G(v(F(u)), v(F(u')) \geq \mathsf{dist}_{G'}(v(F(u)), v(F(u')),$$

as G is obtained from G' by only deleting edges. We say that a pair F, F' of clusters are *friends* (denoted as $F \sim F'$), iff $\mathsf{dist}_{G'}(v(F), v(F')) \leq \varepsilon \log\log n/30$. We say that an edge

(u, u') is *unfriendly*, iff the pair of clusters that contain u and u' are not friends. Therefore, in order to show $\mathsf{cost}(\mathcal{F}, \delta) = \Omega(\varepsilon n \log\log n)$, it suffices to show that there are $\Omega(n)$ unfriendly edges in G'. In particular, since graph G is obtained from G' by deleting at most $n^{0.3}$ edges, there are $\Omega(n)$ edges contributing at least $\varepsilon \log\log n$ each to $\mathsf{cost}(\mathcal{F}, \delta)$. Note that, as G is a 6-regular graph, each cluster is a friend to at most $6^{\varepsilon \log\log n/30} < \log^{\varepsilon/10} n$ clusters in $\mathcal{F}$.

The following lemma shows that there with high probability, const fraction of the edges in G' are unfriendly edges whose lengths are $\Omega(\varepsilon \log\log n)$. This lemma completes the proof in this case.

▶ **Lemma 8.** *With probability* $1-o(1)$*, the random graph* G' *satisfies that, for any partition* $\mathcal{F}$ *of* V *into* $|\mathcal{F}| \le O(n/\log^{\varepsilon} n)$ *clusters such that* $\sum_{|F|\ge n^{0.1}} |F| \le 0.1n$*, and for any friendship relation on* $\mathcal{F}$ *in which each cluster* F *is a friend to at most* $\log^{0.1\varepsilon} n$ *other clusters,* G' *contains at least* $n/10$ *unfriendly edges, i.e.,* $\sum_{F \not\sim F'} |E_{G'}(F, F')| \ge n/10$.

Proof. Recall that G' is obtained by sampling three random permutations $\sigma_1, \sigma_2, \sigma_3$ from Σ and taking the union of their corresponding edge sets $E_{\sigma_1}, E_{\sigma_2}, E_{\sigma_3}$. We alternatively view G' as constructed in two steps. In the first step, we obtain a graph $\hat{G}$ by sampling two random permutations σ_1, σ_2 from Σ and letting $\hat{G} = (V, E_{\sigma_1} \cup E_{\sigma_2})$. In the second step, we sample a third permutation σ_3 from Σ and let $G' = (V, E(\hat{G}) \cup E_{\sigma_3})$. From Lemma 7, with high probability, $\hat{G}$ contains a Hamiltonian cycle on V.

For convenience, we denote by $(\mathcal{F}, \sim)$ a pair of clustering $\mathcal{F}$ and the friendship relation on clusters of $\mathcal{F}$. We say that the pair $(\mathcal{F}, \sim)$ is *valid*, iff $|\mathcal{F}| \le O(n/\log^{\varepsilon} n)$, $\sum_{|F|\ge n^{0.1}} |F| \le 0.1n$, and each cluster F is a friend to at most $\log^{\varepsilon/10} n$ other clusters.

▷ Claim 9. For any Hamiltonian cycle C on V, there are at most $n^{n/4}$ valid pairs $(\mathcal{F}, \sim)$ satisfying that $\sum_{F \not\sim F'} |E_C(F, F')| < n/10$.

Proof. Denote by $L = c^* n/\log^{\varepsilon} n$ the number of clusters of $\mathcal{F}$, and let $\mathcal{F} = \{F_1, \ldots, F_L\}$.

First, the number of possible friendship relations on $\mathcal{F}$ such that each cluster of $\mathcal{F}$ is a friend to at most $\log^{0.1\varepsilon} n$ other clusters is at most

$$\binom{L}{\log^{0.1\varepsilon} n}^{L} \le \binom{\frac{c^* n}{\log^{\varepsilon} n}}{\log^{0.1\varepsilon} n}^{\frac{c^* n}{\log^{\varepsilon} n}} \le \left(\frac{c^* n}{\log^{\varepsilon} n}\right)^{\frac{c^* n}{\log^{\varepsilon} n} \cdot \log^{0.1\varepsilon} n} < n^{c^* n \log^{-0.05\varepsilon} n}.$$

Assume now that we have a fixed friendship relation $\sim$ on the clusters in $\mathcal{F}$. We now count the number of clusterings $\mathcal{F}$ with $\sum_{F \not\sim F'} |E_C(F, F')| < n/10$. Denote $C = (v_1, v_2, \ldots, v_n, v_1)$. First, the number of possible unfriendly edge set (which is a subset of $E(C)$ of size at most $0.1n$) is at most

$$\sum_{i=0}^{n/10} \binom{n}{i} \le n \cdot \binom{n}{n/10} < n \cdot \left(\frac{en}{n/10}\right)^{n/10} < n^{n \log^{-0.5} n}.$$

We now count the number of clusterings $\mathcal{F}$ that, together with the fixed friendship relation $\sim$, realizes a specific unfriendly edge set. We will sequentially pick, for each i from 1 to n, a set among $\{F_1, \ldots, F_L\}$ to add the vertex v_i to. The first vertex v_1 has L choices. Consider now some index $1 \le i \le n-1$ and assume that we have picked sets for $v_1, \ldots, v_i$. If (v_i, v_{i+1}) is an unfriendly edge, then vertex v_{i+1} has L choices; if (v_i, v_{i+1}) is not an unfriendly edge, this means that v_{i+1} must go to some cluster that is a friend of the cluster we have picked for v_i (or v_{i+1} can go to the same cluster as v_i), so v_{i+1} has at most $\log^{0.1\varepsilon} n + 1$ choices. As there are no more than $0.1n$ unfriendly edges, the number of possible clusterings $\mathcal{F}$ is at most

$$n \cdot (\log^{0.1\varepsilon} n)^n \cdot \left(\frac{n}{\log^{\varepsilon} n}\right)^{0.1n} < n^{n/5}.$$

Altogether, the number of valid pairs $(\mathcal{F}, \sim)$ satisfying that $\sum_{F \not\sim F'} |E_C(F, F')| \geq n/10$ is at most

$$n^{c^* n \log^{-0.05\varepsilon} n} \cdot n^{n \log^{-0.5} n} \cdot n^{n/5} < n^{n/4}.$$ ◁

▷ Claim 10. For every valid pair $(\mathcal{F}, \sim)$, the probability that the edge set E_{σ_3} of a random permutation σ_3 contains at most $n/10$ unfriendly edges is at most $n^{-n/3}$.

Proof. We say that a cluster $F \in \mathcal{F}$ is *bad* if it does not have a friend cluster of size at least $n^{0.4}$, otherwise we say it is *good*. We first prove the following observation that most vertices lie in a bad cluster.

▶ **Observation 11.** $\sum_{F:bad} |F| \geq 0.8n$.

Proof. As the pair $(\mathcal{F}, \sim)$ is valid, $\sum_{|F| \geq n^{0.1}} |F| \leq 0.1n$, so $\mathcal{F}$ contains at most $0.1 \cdot n^{0.6}$ clusters with size at least $n^{0.4}$. As each cluster is a friend to at most $\log^{0.1\varepsilon} n$ other clusters, $\mathcal{F}$ contains at most $\left(0.1 \cdot n^{0.6} \cdot \log^{0.1\varepsilon} n\right)$ good sets. Therefore, the total size of all good clusters is at most $0.1n + n^{0.1} \cdot 0.1 \cdot n^{0.6} \cdot \log^{0.1\varepsilon} n < 0.2n$. The observation now follows. ◀

We alternatively construct the random permutation σ_3 as follows. We arrange the vertices in V into a sequence $(v_1, \ldots, v_n)$, such that each of the first half $v_1, \ldots, v_{n/2}$ lies in some bad set. Now sequentially for each $1, 2, \ldots, n$, we sample a vertex u_i (without replacement) from V and designate it as $\sigma_3(v_i)$. It is easy to observe that the permutation σ_3 constructed in this way is a random permutation from Σ.

The following observation completes the proof of Claim 10.

▶ **Observation 12.** *The probability that the number of unfriendly edges in* $\{(v_i, \sigma_3(v_i)) \mid 1 \leq i \leq 9n/10\}$ *is less than* $0.1n$ *is at most* $n^{-n/3}$.

Proof. For any v in a bad cluster, the number of vertices in its friend clusters is at most $n^{0.4} \log^{0.1} n$. For each $1 \leq i \leq 9n/10$, when we pick $\sigma_3(v_i)$, we have at least $n/10$ choices from the remaining element in V, and as v_i is in a bad set, at most $n^{0.4} \log^{0.1} n$ of them will not create an unfriendly edge. Therefore, the probability that the edge we sample is not a bad edge is at most $\frac{1}{\sqrt{n}}$. Let X_i be the indicator random variable such that $X_i = 1$ if $(v_i, \sigma_3(v_i))$ is not a bad edge. By Azuma's Inequality (Chernoff Bounds on martingales, see e.g., [25]),

$$\Pr\left[\sum_{i=1}^{2n/3} X_i > 4n/5\right] < \left(\frac{5}{4\sqrt{n}}\right)^{4n/5} < n^{-n/3}.$$

Thus, with probability at least $1 - n^{-n/3}$, the set $\{(v_i, \sigma_3(v_i)) | 1 \leq i \leq 9n/10\}$ contains at least $9n/10 - 4n/5 = n/10$ bad edges. ◀

◁

Combining Claim 9 and Claim 10, we get that, over the randomness in the construction of G', the probability that there exists a pair $(\mathcal{F}, \sim)$ in which each cluster F is a friend to at most $\log^{0.1\varepsilon} n$ other clusters, such that G' contains less than $n/10$ unfriendly edges, is at most $n^{-n/3} \cdot n^{n/4} = n^{-n/12}$. This completes the proof of Lemma 8. ◀

Case 2: The total size of large clusters is greater than $0.1n$

We denote by V' the union of all large clusters in $\mathcal{F}$. We start by proving the following claim.

▷ Claim 13. There exists a collection of $k/4$ edge-disjoint paths in G, such that each path connects a distinct terminal to a distinct vertex of V'.

Proof. We construct a graph $\hat{G}$ as follows. We start from graph G', and add two vertices s, t to it. We then connect s to each terminal in T by an edge, and connect each vertex in V' to t by an edge. All edges in $\hat{G}$ has unit capacity. We claim that there exists a collection $\mathcal{P}$ of $k/3$ edge-disjoint paths in $\hat{G}$, such that each path connects a distinct terminal to a distinct vertex of V'. Note that this implies Claim 13. This is because the number of edges in $E(G') \setminus E(G)$ is at most $n^{0.3} < k/12$, and each such edge is contained in at most one path in $\mathcal{P}$ (since the paths in $\mathcal{P}$ are edge-disjoint), so at least $k/3 - k/12 \geq k/4$ paths in $\mathcal{P}$ are entirely contained in G. We now prove the claim. From the max-flow min-cut theorem, it suffices to show that the minimum s-t cut in $\hat{G}$ contains at least $k/3$ edges.

Consider any s-t cut $(S \cup \{s\}, (V \setminus S) \cup \{t\})$ in $\hat{G}$ and denote by E' the set of edges in this cut. We distinguish between the following cases.

Case 1: $|S| \leq |V|/2$. Recall that G is a 6-regular graph, so $\sum_{v \in S} \deg(v) \leq \sum_{v \notin S} \deg(v)$. Then from Lemma 5, $|E'| \geq \sum_{v \in S} \deg(v)/2 \geq |S|/2$. If $|S| \geq 2k/3$, then $|E'| \geq k/3$. If $|S| < 2k/3$, then at least $k/3$ terminals lie in $V \setminus S$. As there is an edge connecting s to each terminal, $|E'| \geq k/3$.

Case 2: $|S| > |V|/2$. Via similar arguments, we can show that $|E'| \geq |V'|/3 \geq 0.1n/3 > k/3$. ◁

We denote by $\mathcal{P}$ the collection of paths given by Claim 13. We now use these paths to complete the proof. Consider such a path $P = (u_1, \ldots, u_r)$. Denote by F_i the cluster that contains u_i, then the contribution of P to the cost $\mathsf{cost}(\mathcal{F}, \delta)$ is

$$\sum_{(u_i, u_{i+1}) \in E(P)} \delta(F_i, F_{i+1}) = \sum_{1 \leq i \leq r-1} \mathsf{dist}_G(v(F_i), v(F_{i+1})) \geq \mathsf{dist}_G(v(F_1), v(F_r)) \geq \mathsf{dist}_{G'}(v(F_1), v(F_r)).$$

(We have used the property that for every pair $v, v' \in V$, $\mathsf{dist}_G(v, v') \geq \mathsf{dist}_{G'}(v, v')$, as G is obtained from G' by only deleting edges.)

Recall P connects a terminal to a vertex in V'. Recall that each large cluster has size at least $n^{0.1}$, so there are at most $n^{0.9}$ of them. Therefore, if we denote by V'' the subset of vertices that large clusters corresponds to, then $|V''| \leq n^{0.9}$. For each path $P \in \mathcal{P}$, we denote by t_P the terminal endpoint of P (that is, $u_1 = v(F_1) = t_P$), and by v''_P the vertex that the cluster containing u_r corresponds to (that is, $v''_P = v(F_r)$), then $\sum_{(u_i, u_{i+1}) \in E(P)} \delta(F_i, F_{i+1}) \geq \mathsf{dist}_\ell(t_P, v''_P)$. As the paths in $\mathcal{P}$ are edge-disjoint, their contribution to $\mathsf{cost}(\mathcal{F}, \delta)$ can be added up, i.e.,

$$\mathsf{cost}(\mathcal{F}, \delta) \geq \sum_{P \in \mathcal{P}} \mathsf{dist}_\ell(t_P, v''_P). \tag{1}$$

On the one hand, as graph G' is 6-regular, for each $v'' \in V''$, the number of vertices at distance at most $\log n/100$ to v'' is at most $6^{\log n/100} \leq n^{1/30}$. Therefore, there are at most $n^{1/30} \cdot n^{0.9} = n^{14/15}$ terms on the RHS of Equation (1) that at most $\leq \log n/100$. On the other hand, there are at least $k/4 = \Omega(\frac{n \log \log n}{\log n})$ terms on the RHS of Equation (1), so at least $k/4 - n^{14/15} \geq k/5$ terms has value at least $\log n/100$. Consequently, $\mathsf{cost}(\mathcal{F}, \delta) \geq (k/5) \cdot (\log n/100) = \Omega(k \log n) = \Omega(n \log \log n)$.

3.3 Completing the Proof of Theorem 1

We have shown in Section 3.2 all canonical solutions with size $O(k \log^{1-\varepsilon} n)$ have cost $\Omega(\varepsilon n \log \log n)$. In this subsection, we complete the proof of Theorem 1 by showing that, intuitively, an arbitrary solution $(\mathcal{F}, \delta)$ to the instance (G, T, D) can be "embedded" into a canonical solution, without increasing its cost by too much.

We start by introducing the notion of *continuization*.

Continuization of a graph

Let $G = (V, E, \ell)$ be an edge-weighted graph. Its *continuization* is a metric space $(V^{\mathsf{con}}, \ell^{\mathsf{con}})$, that is defined as follows. Each edge $(u, v) \in E$ is viewed as a continuous line segment $\mathsf{con}(u, v)$ of length $\ell_{(u,v)}$ connecting u, v, and the point set V^{con} is the union of the points on all lines $\{\mathsf{con}(u, v)\}_{(u,v)\in E}$. Specifically, for each edge $(u, v) \in E$, the line $\mathsf{con}(u, v)$ is defined as

$$\mathsf{con}(u, v) = \left\{(u, \alpha) \mid 0 \le \alpha \le \ell_{(u,v)}\right\} = \left\{(v, \beta) \mid 0 \le \beta \le \ell_{(u,v)}\right\},$$

where (u, α) refers to the unique point on the line that is at distance α from u, and (v, β) refers to the unique point on the line that is at distance β from v, so $(u, \alpha) = (v, \ell_{(u,v)} - \alpha)$.

The metric ℓ^{con} on V^{con} is naturally induced by the shortest-path distance metric $\mathsf{dist}_\ell(\cdot, \cdot)$ on V as follows. For a pair p, p' of points in V^{con},

- if p, p' lie on the same line (u, v), say $p = (u, \alpha)$ and $p' = (u, \alpha')$, then $\ell^{\mathsf{con}}(p, p') = |\alpha - \alpha'|$;
- if p lies on the line (u, v) with $p = (u, \alpha)$ and p' lies on the line (u', v') with $p' = (u', \alpha')$, then

$$\begin{aligned}\ell^{\mathsf{con}}(p, p') = \min\{&\mathsf{dist}_\ell(u, u') + \alpha + \alpha', \quad \mathsf{dist}_\ell(u, v') + \alpha + (\ell_{(u',v')} - \alpha'),\\ &\mathsf{dist}_\ell(v, u') + (\ell_{(u,v)} - \alpha) + \alpha', \quad \mathsf{dist}_\ell(v, v') + (\ell_{(u,v)} - \alpha) + (\ell_{(u',v')} - \alpha')\}.\end{aligned}$$

Clearly, every vertex $u \in V$ also belongs to V^{con}, and for every pair $u, u' \in V$, $\mathsf{dist}_\ell(u, u') = \ell^{\mathsf{con}}(u, u')$. For a path P in G connecting u to u', it naturally corresponds to a set P^{con} of points in V^{con}, which is the union of all lines corresponding to edges in $E(P)$. The set P^{con} naturally inherits the metric ℓ^{con} restricted on P^{con}. We will also call P^{con} a path in the continuization $(V^{\mathsf{con}}, \ell^{\mathsf{con}})$.

We show that, for each graph G with a set T of terminals, then any other metric w on a set of points containing T, such that w restricted on T is identical to dist_G restricted on T, can be "embedded" into the continuation of G, with expected stretch bounded by some structural measure that only depends on G. Specifically, we prove the following main technical lemma.

▶ **Lemma 14.** *Let (G, T, ℓ) be any instance of* 0EwSN *such that G is not a tree (so $\mathsf{girth}(G) < +\infty$), and let $(\mathcal{F}, \delta)$ be any solution to it. Let $(V^{\mathsf{con}}, \ell^{\mathsf{con}})$ be the continuization of graph G. Then there exists a random mapping $\phi : \mathcal{F} \to V^{\mathsf{con}}$, such that*

- *for each terminal $t \in T$, if F is the (unique) cluster in $\mathcal{F}$ that contains t, then $\phi(F) = t$; and*
- *for every pair $F, F' \in \mathcal{F}$,*

$$\mathbb{E}\left[\ell^{\mathsf{con}}(\phi(F), \phi(F'))\right] \le O\left(\frac{\mathsf{diam}(G)}{\mathsf{girth}(G)}\right) \cdot \delta(F, F').$$

Before we prove Lemma 14 in Section 3.4, we provide the proof of Theorem 1 using it.

Proof of Theorem 1. Consider any solution $(\mathcal{F}, \delta)$ to the instance (G, T, ℓ) constructed in Section 3.1 with size $|\mathcal{F}| \leq o(k \cdot \log^{1-\varepsilon} k)$. From Observation 3 and Corollary 6, $\frac{\mathsf{diam}(G)}{\mathsf{girth}(G)} \leq \frac{O(\log n)}{(\log n)/100} = O(1)$. From Lemma 14, there exists a random mapping $\phi : \mathcal{F} \to V^{\mathsf{con}}$, such that for every pair $F, F' \in \mathcal{F}$, $\mathbb{E}\left[\ell^{\mathsf{con}}(\phi(F), \phi(F'))\right] \leq O(1) \cdot \delta(F, F')$.

Fix such a mapping ϕ, we define a canonical solution $(\mathcal{F}, \hat{\delta})$ based on $(\mathcal{F}, \delta)$ as follows. The collection of clusters is identical to the collection $\mathcal{F}$. For each $F \in \mathcal{F}$, recall that $\phi(F)$ is a point in V^{con}. Assume the point $\phi(F)$ lies on the line (u, v) and is closer to u than to v (i.e., $\ell^{\mathsf{con}}(\phi(F), u) \leq \ell^{\mathsf{con}}(\phi(F), v)$), then we let u be the vertex in V that it corresponds to. For each pair $F, F' \in \mathcal{F}$, with F corresponding to u_F and and F' corresponding to $u_{F'}$, we define $\hat{\delta}(F, F') = \mathsf{dist}_\ell(u_F, u_{F'})$. As graph G in the instance (G, T, δ) constructed in Section 3.1 is an unweighted graph, it is easy to see that

$$\hat{\delta}(F, F') = \mathsf{dist}_\ell(u_F, u_{F'}) \leq \ell^{\mathsf{con}}(\phi(F), \phi(F')) + 2.$$

As the mapping ϕ is random, $\hat{\delta}$ is also random, and so $\mathbb{E}\left[\hat{\delta}(F, F')\right] \leq O(1) \cdot \delta(F, F') + 2$.

From the properties of mapping ϕ in Lemma 14, we are guaranteed that such a solution $(\mathcal{F}, \hat{\delta})$ is a canonical solution. Moreover, from linearity of expectation,

$$\begin{aligned}\mathbb{E}\left[\mathsf{cost}(\mathcal{F}, \hat{\delta})\right] = \mathbb{E}\left[\sum_{(u,v)\in E} \hat{\delta}(F(u), F(v))\right] &= \sum_{(u,v)\in E} O\Big(\delta(F(u), F(v))\Big) + 2 \\ &= O\Big(\mathsf{cost}(\mathcal{F}, \delta)\Big) + O(n).\end{aligned}$$

Therefore, it follows that there exists a canonical solution $(\mathcal{F}, \hat{\delta})$, such that $\mathsf{cost}(\mathcal{F}, \hat{\delta}) \leq O(\mathsf{cost}(\mathcal{F}, \delta) + n)$. As we have shown in Section 3.2 that any canonical solution $(\mathcal{F}, \hat{\delta})$ with $|\mathcal{F}| \leq o(k \log^{1-\varepsilon} k)$ satisfies that $\mathsf{cost}(\mathcal{F}, \hat{\delta}) = \Omega(\varepsilon n \log\log n)$, it follows that $\mathsf{cost}(\mathcal{F}, \delta) = \Omega(\varepsilon n \log\log n)$. This implies that the integrality gap of (LP-Metric) is at least $\Omega(\varepsilon \log\log n)$. ◀

3.4 Proof of Lemma 14

In this subsection, we provide the proof of Lemma 14. We first consider the special case where G is a tree, and then prove Lemma 14 for the general case.

▶ **Lemma 15.** *Let (G, T, ℓ) be an instance of* 0EwSN *where G is a tree. Let $(\mathcal{F}, \delta)$ be an solution to it. Let $(V^{\mathsf{con}}, \ell^{\mathsf{con}})$ be the continuization of G. Then there exists a mapping $\phi : \mathcal{F} \to V^{\mathsf{con}}$, such that*

- *for each terminal $t \in T$, if F is the (unique) cluster in $\mathcal{F}$ that contains t, then $\phi(F) = t$; and*
- *for every pair $F, F' \in \mathcal{F}$, $\ell^{\mathsf{con}}(\phi(F), \phi(F')) \leq \delta(F, F')$.*

Proof. For each terminal $t \in T$, we denote by F_t the cluster in $\mathcal{F}$ that contains it, and set $\phi(F_t) = t$. For each cluster $F \in \mathcal{F}$ that does not contain any terminals, we define

$$\nu(F) = \min\left\{\frac{1}{2} \cdot \big(\delta(F, F_t) + \delta(F, F_{t'}) - \delta(F_t, F_{t'})\big) \mid t, t' \in T\right\}.$$

Denote by t_1, t_2 the pair of terminals (t, t') that minimizes $(\delta(F, F_t) + \delta(F, F_{t'}) - \delta(F_t, F_{t'}))/2$. As G is a tree, there is a unique shortest path connecting t_1 to t_2 in G, and therefore there exists a unique point p in V^{con} (that lies on the t_1-t_2 shortest path in V^{con}) with $\ell^{\mathsf{con}}(p, t_1) = \delta(F, F_{t_1}) - \nu(F)$ and $\ell^{\mathsf{con}}(p, t_2) = \delta(F, F_{t_2}) - \nu(F)$. We set $\phi(F) = p$.

▷ Claim 16. For every cluster $F \in \mathcal{F}$ and every terminal $t \in T$, $\ell^{\mathsf{con}}(\phi(F), t) \leq \delta(F, F_t) - \nu(F)$.

Proof. Note that the point $\phi(F)$ lies on the (unique) shortest path between a pair t_1, t_2 of terminals. For $t \in \{t_1, t_2\}$, clearly the claim holds. Consider any other terminal t. Clearly $\phi(F)$ lies on either the path connecting t to t_1 or the path connecting t to t_2. Assume without lose of generality that $\phi(F)$ is on the path connecting t_1 and t. Since G is a tree, $\ell^{\mathsf{con}}(\phi(F), t_1) + \ell^{\mathsf{con}}(\phi(F), t) = \ell^{\mathsf{con}}(t_1, t) = \mathsf{dist}_\ell(t_1, t)$. On the other hand, by definition of $\nu(F)$ and $\phi(F)$, $\delta(F, F_{t_1}) + \delta(F, F_t) \geq \delta(F_{t_1}, F_t) + 2 \cdot \nu(F)$ holds, and $\ell^{\mathsf{con}}(\phi(F), t_1) = \delta(F, F_{t_1}) - \nu(F)$. Therefore, $\delta(F, F_t) \geq \ell^{\mathsf{con}}(\phi(F), t) + \nu(F)$. ◁

We now show that, for every pair $F, F' \in \mathcal{F}$, $\ell^{\mathsf{con}}(\phi(F), \phi(F')) \leq \delta(F, F')$. Denote by t_1, t_2 the pair of terminals whose shortest path contains $\phi(F)$, and by t_1', t_2' the pair of terminals whose shortest path contains $\phi(F')$. Assume without lose of generality that $\phi(F)$ is on the tree path between $\phi(F')$ and t_1, so $\ell^{\mathsf{con}}(\phi(F'), \phi(F)) + \ell^{\mathsf{con}}(\phi(F), t_1) = \ell^{\mathsf{con}}(\phi(F'), t_1)$. On the other hand, from the definition of $\phi(F)$ and Claim 16, $\ell^{\mathsf{con}}(\phi(F), t_1) = \delta(F, F_{t_1}) - \nu(F)$ and $\ell^{\mathsf{con}}(\phi(F'), t_1) \leq \delta(F', F_{t_1}) - \nu(F')$. Therefore,

$$\ell^{\mathsf{con}}(\phi(F), \phi(F')) \leq \delta(F', F_{t_1}) - \delta(F, F_{t_1}) - \nu(F') + \nu(F) \leq \delta(F, F') + \nu(F) - \nu(F').$$

Similarly, $\ell^{\mathsf{con}}(\phi(F), \phi(F')) \leq \delta(F, F') + \nu(F') - \nu(F)$. Altogether, $\ell^{\mathsf{con}}(\phi(F), \phi(F')) \leq \delta(F, F')$. ◀

In the remainder of this subsection, we complete the proof of Lemma 14. Denote $g = \mathsf{girth}(G)$. Let r be a real number chosen uniformly at random from the interval $[g/60, g/30]$, so $r \leq g/30$ always holds. For each terminal $t \in T$, we denote by F_t the cluster in $\mathcal{F}$ that contains it, and set $\phi(F_t) = t$, so the first condition in Lemma 14 is satisfied.

For each cluster $F \in \mathcal{F}$, we define $A_F = \min\{\delta(F, F_t) \mid t \in T\}$. We first determine the image $\phi(F)$ for all clusters F with $A_F \leq r$, in a similar way as Lemma 15 as follows.

Define

$$\nu(F) = \min\left\{\frac{1}{2} \cdot \left(\delta(F, F_t) + \delta(F, F_{t'}) - \frac{\delta(F_t, F_{t'})}{2}\right) \mid t, t' \in T\right\}.$$

Denote by t_1, t_2 the pair (t, t') that minimizes the above formula. We prove the following claim.

▷ Claim 17. $\delta(F, F_{t_1}) + \delta(F, F_{t_2}) \leq 4 \cdot A_F \leq 4r$.

Proof. Let t be the terminal such that $\delta(F, F_t) = A_F$. By definition of $\nu(F)$,

$$\nu(F) \leq \frac{\delta(F, F_t) + \delta(F, F_t) - \frac{1}{2} \cdot \delta(F_t, F_t)}{2} = A_F.$$

On the other hand, from triangle inequality,

$$\nu(F) = \frac{\delta(F, F_{t_1}) + \delta(F, F_{t_2}) - \frac{1}{2} \cdot \delta(F_{t_1}, F_{t_2})}{2} \geq \frac{\delta(F, F_{t_1}) + \delta(F, F_{t_2})}{4}.$$

Altogether, $\delta(F, F_{t_1}) + \delta(F, F_{t_2}) \leq 4 \cdot A_F$. ◁

From Claim 17, $\delta(F_{t_1}, F_{t_2}) \leq \delta(F, F_{t_1}) + \delta(F, F_{t_2}) \leq 4r < g/3$. Therefore, there is a unique shortest path connecting t_1 to t_2 in G, as otherwise G must contain a cycle of length at most $2g/3$, contradicting the fact that $\mathsf{girth}(G) = g$.

We then set $\phi(F)$ to be the point in V^{con} that lies in the t_1-t_2 shortest path, such that $\ell^{\mathsf{con}}(\phi(F), t_1) = 2 \cdot (\delta(F, F_{t_1}) - \nu(F))$ and $\ell^{\mathsf{con}}(\phi(F), t_2) = 2 \cdot (\delta(F, F_{t_2}) - \nu(F))$. Note that, by definition of t_1, t_2,

$$\ell^{\mathsf{con}}(\phi(F), t_1) + \ell^{\mathsf{con}}(\phi(F), t_2) = 2 \cdot \delta(F, F_{t_1}) + 2 \cdot \delta(F, F_{t_2}) - 4 \cdot \nu(F) = \delta(F_{t_1}, F_{t_2}).$$

We prove the following claim, that is similar to Claim 16.

▷ Claim 18. For every terminal t with $\delta(F, F_t) \leq 6r$, $\ell^{\mathsf{con}}(\phi(F), t) \leq 2 \cdot (\delta(F, F_t) - \nu(F))$.

Proof. From Claim 17,

$$\begin{aligned}\delta(F_t, F_{t_1}) + \delta(F_t, F_{t_2}) + \delta(F_{t_1}, F_{t_2}) &\leq \Big(2 \cdot \delta(F, F_t) + \delta(F, F_{t_1}) + \delta(F, F_{t_2})\Big) + \delta(F_{t_1}, F_{t_2}) \\ &\leq 12r + 4r + 4r + 4r < g.\end{aligned}$$

Therefore, the point $\phi(F)$ must lie on either the t-t_1 shortest path or the t-t_2 shortest path in V^{con}, as otherwise the union of t-t_1 shortest path, t-t_2 shortest path, and t_1-t_2 shortest path contains a cycle of length less than g, a contradiction.

Assume without loss of generality that $\phi(F)$ lies on the t-t_1 shortest path, so

$$\ell^{\mathsf{con}}(\phi(F), t) = \delta(F_t, F_{t_1}) - \ell^{\mathsf{con}}(\phi(F), t_1) = \delta(F_t, F_{t_1}) - 2(\delta(F, F_{t_1}) - \nu(F)).$$

By definition of $\nu(F)$, $\delta(F_t, F_{t_1}) \leq 2 \cdot (\delta(F, F_t) + \delta(F, F_{t_1}) - 2 \cdot \nu(F))$. Therefore,

$$\ell^{\mathsf{con}}(\phi(F), t) \leq 2\Big(\delta(F, t) + \delta(F, F_{t_1}) - 2\nu(F) - \big(\nu(F, F_{t_1}) - \nu(F)\big)\Big) = 2(\delta(F, F_t) - \nu(F)).$$

◁

We now show in the next claim that the second condition in Lemma 14 holds for pairs of clusters that are close in δ.

▷ Claim 19. For every pair $F, F' \in \mathcal{F}$ with $A_F, A_{F'}, \delta(F, F') \leq r$, $\ell^{\mathsf{con}}(\phi(F), \phi(F')) \leq 2 \cdot \delta(F, F')$.

Proof. Since $\delta(F, F') < r$, from triangle inequality and Claim 17,

$$\delta(F', F_{t_1}) \leq \delta(F', F) + \delta(F, F_{t_1}) \leq r + 4r \leq 5r.$$

Similarly, $\delta(F', F_{t_2}) \leq 5r$. Then from Claim 18,

$$\ell^{\mathsf{con}}(\phi(F'), t_1) \leq 2 \cdot (\delta(F', F_{t_1}) - \nu(F')) \leq 2 \cdot \delta(F', F_{t_1}) \leq 10r,$$

and symmetrically, $\ell^{\mathsf{con}}(\phi(F'), t_2) < 10r$. Therefore,

$$\ell^{\mathsf{con}}(\phi(F'), t_1) + \ell^{\mathsf{con}}(\phi(F'), t_2) + \ell^{\mathsf{con}}(t_1, t_2) < 20r + 4r < g.$$

Therefore, the point $\phi(F)$ must lie either on the $\phi(F')$-t_1 shortest path or the $\phi(F')$-t_2 shortest path in V^{con}, as otherwise the union of $\phi(F')$-t_1 shortest path, $\phi(F')$-t_2 shortest path, and t_1-t_2 shortest path in V^{con} contains a cycle of length less than g, a contradiction.

Assume without loss of generality that $\phi(F)$ lies on the $\phi(F')$-t_1 shortest path. Then

$$\begin{aligned}\ell^{\mathsf{con}}(\phi(F), \phi(F')) &= \ell^{\mathsf{con}}(\phi(F'), t_1(F)) - \ell^{\mathsf{con}}(\phi(F), t_1(F)) \\ &\leq 2\Big(\delta(F', t_1(F)) - \nu(F')\Big) - 2\Big(\delta(F, t_1(F)) - \nu(F)\Big) \\ &\leq 2\Big(\delta(F, F') + \nu(F) - \nu(F')\Big).\end{aligned}$$

Similarly, $\ell^{\mathsf{con}}(\phi(F), \phi(F')) \leq 2\big(\delta(F, F') + \nu(F') - \nu(F)\big)$. Altogether, $\ell^{\mathsf{con}}(\phi(F), \phi(F')) \leq 2\delta(F, F')$. ◁

We now complete the construction of the mapping ϕ, by specifying the images of all clusters $F \in \mathcal{F}$ with $A_F > r$. Let t^* be an arbitrarily chosen terminal in T. For all clusters $F \in \mathcal{F}$ with $A_F > r$, we simply set $\phi(F) = t^*$.

It remains to show that the second condition in Lemma 14 holds for all pairs $F, F' \in \mathcal{F}$. Consider a pair F, F'. Assume first that $\delta(F, F') > g/60$. Then

$$\ell^{\mathsf{con}}(\phi(F), \phi(F')) \leq \mathsf{diam}(G) \leq \frac{60 \cdot \mathsf{diam}(G)}{\mathsf{girth}(G)} \cdot \delta(F, F').$$

Assume now that $\delta(F, F') \leq g/60$, and without loss of generality that $A_F \leq A_{F'}$. Note that, from triangle inequality, for every terminal $t \in T$, $\delta(F', F_t) \leq \delta(F, F_t) - \delta(F, F')$. This implies that $A_{F'} - A_F \leq \delta(F, F')$. Therefore, the probability that the random number r takes value from the interval $[A_F, A_{F'}]$ is at most $\frac{\delta(F,F')}{g/60}$. Note that, if $r \leq A_F$, then $\phi(F) = \phi(F') = t^*$ and $\ell^{\mathsf{con}}(\phi(F), \phi(F')) = 0$. And if $r \geq A_{F'}$, then from Claim 19, $\ell^{\mathsf{con}}(\phi(F), \phi(F')) \leq 2\delta(F, F')$. Altogether,

$$\mathbb{E}\left[\ell^{\mathsf{con}}(\phi(F), \phi(F'))\right] \leq \frac{60 \cdot \mathsf{diam}(G)}{\mathsf{girth}(G)} \cdot \delta(F, F') + 2\delta(F, F') \leq O\left(\frac{\mathsf{diam}(G)}{\mathsf{girth}(G)}\right) \cdot \delta(F, F').$$

References

1. Alexandr Andoni, Anupam Gupta, and Robert Krauthgamer. Towards (1+ε)-approximate flow sparsifiers. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 279–293. SIAM, 2014.
2. Haris Angelidakis, Yury Makarychev, and Pasin Manurangsi. An improved integrality gap for the călinescu-karloff-rabani relaxation for multiway cut. In *Integer Programming and Combinatorial Optimization: 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, pages 39–50. Springer, 2017.
3. Aaron Archer, Jittat Fakcharoenphol, Chris Harrelson, Robert Krauthgamer, Kunal Talwar, and Éva Tardos. Approximate classification via earthmover metrics. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1079–1087. Society for Industrial and Applied Mathematics, 2004.
4. Kristóf Bérczi, Karthekeyan Chandrasekaran, Tamás Király, and Vivek Madan. Improving the integrality gap for multiway cut. *Mathematical Programming*, 183(1-2):171–193, 2020.
5. Niv Buchbinder, Joseph Naor, and Roy Schwartz. Simplex partitioning via exponential clocks and the multiway cut problem. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 535–544, 2013.
6. Niv Buchbinder, Roy Schwartz, and Baruch Weizman. Simplex transformations and the multiway cut problem. In *Proceedings of the twenty-eighth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2400–2410. SIAM, 2017.
7. Gruia Călinescu, Howard Karloff, and Yuval Rabani. An improved approximation algorithm for multiway cut. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 48–52, 1998.
8. Gruia Calinescu, Howard Karloff, and Yuval Rabani. Approximation algorithms for the 0-extension problem. *SIAM Journal on Computing*, 34(2):358–372, 2005.
9. Moses Charikar, Tom Leighton, Shi Li, and Ankur Moitra. Vertex sparsifiers and abstract rounding algorithms. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 265–274. IEEE, 2010.
10. Chandra Chekuri, Sanjeev Khanna, Joseph Naor, and Leonid Zosin. A linear programming formulation and approximation algorithms for the metric labeling problem. *SIAM Journal on Discrete Mathematics*, 18(3):608–625, 2004.
11. Yu Chen and Zihan Tan. Towards the characterization of terminal cut functions: a condition for laminar families. *arXiv preprint*, 2023. `arXiv:2310.11367`.
12. Yu Chen and Zihan Tan. On (1+ ε)-approximate flow sparsifiers. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1568–1605. SIAM, 2024.

13 Julia Chuzhoy. On vertex sparsifiers with steiner nodes. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 673–688, 2012.

14 Julia Chuzhoy and Joseph Naor. The hardness of metric labeling. *SIAM Journal on Computing*, 36(5):1376–1386, 2007.

15 Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.

16 Jittat Fakcharoenphol, Chris Harrelson, Satish Rao, and Kunal Talwar. An improved approximation algorithm for the 0-extension problem. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 257–265. Society for Industrial and Applied Mathematics, 2003.

17 Ari Freund and Howard Karloff. A lower bound of $8/(7 + \frac{1}{k-1})$ on the integrality ratio of the călinescu–karloff–rabani relaxation for multiway cut. *Information Processing Letters*, 75(1-2):43–50, 2000.

18 Alan Frieze. Hamilton cycles in the union of random permutations. *Random Structures & Algorithms*, 18(1):83–94, 2001.

19 Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *Journal of Computer and System Sciences*, 57(3):366–375, 1998.

20 Howard Karloff, Subhash Khot, Aranyak Mehta, and Yuval Rabani. On earthmover distance, metric labeling, and 0-extension. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 547–556, 2006.

21 Howard Karloff, Subhash Khot, Aranyak Mehta, and Yuval Rabani. On earthmover distance, metric labeling, and 0-extension. *SIAM Journal on Computing*, 39(2):371–387, 2009.

22 Alexander V Karzanov. Minimum 0-extensions of graph metrics. *European Journal of Combinatorics*, 19(1):71–101, 1998.

23 Jon Kleinberg and Eva Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. *Journal of the ACM (JACM)*, 49(5):616–639, 2002.

24 Robert Krauthgamer and Ron Mosenzon. Exact flow sparsification requires unbounded size. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2354–2367. SIAM, 2023.

25 William Kuszmaul and Qi Qi. The multiplicative version of azuma's inequality, with an application to contention analysis. *arXiv preprint*, 2021. `arXiv:2102.05077`.

26 F Thomson Leighton and Ankur Moitra. Extensions and limits to vertex sparsification. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 47–56. ACM, 2010.

27 Konstantin Makarychev and Yury Makarychev. Metric extension operators, vertex sparsifiers and lipschitz extendability. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 255–264. IEEE, 2010.

28 Rajsekar Manokaran, Joseph Seffi Naor, Prasad Raghavendra, and Roy Schwartz. Sdp gaps and ugc hardness for multiway cut, 0-extension, and metric labeling. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 11–20. ACM, 2008.

29 Ankur Moitra. Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 3–12. IEEE, 2009.

30 Doron Puder. Expansion of random graphs: New proofs, new results. *Inventiones mathematicae*, 201(3):845–908, 2015.

31 Roy Schwartz and Nitzan Tur. The metric relaxation for 0-extension admits an $\omega(\log^{2/3} k)$ gap. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1601–1614, 2021.

A Comparison between 0EwSN and the variant in [1]

The Steiner node variant of 0-Ext in [1]

In [1], the following problem (referred to as $\mathsf{0EwSN}_{\mathsf{AGK}}$) was proposed.

The input consists of

- an edge-capacitated graph $G = (V, E, c)$, with length $\{\ell_e\}_{e\in E}$ on its edges;
- a set $T \subseteq V$ of k terminals; and
- a *demand* $\mathcal{D} : T \times T \to \mathbb{R}^+$ on terminals.

A solution consists of

- a partition $\mathcal{F}$ of V with $|\mathcal{F}|$, such that distinct terminals of T belong to different sets in $\mathcal{F}$; for each vertex $u \in V$, we denote by $F(u)$ the cluster in $\mathcal{F}$ that contains it;
- a semi-metric δ on the clusters in $\mathcal{F}$, such that: $\sum_{t,t'} \mathcal{D}(t,t') \cdot \delta(F(t), F(t')) \geq \sum_{t,t'} \mathcal{D}(t,t') \cdot \mathsf{dist}_\ell(t,t')$, where $\mathsf{dist}_\ell(\cdot,\cdot)$ is the shortest-path distance (in G) metric induced by edge length $\{\ell_e\}_{e\in E(G)}$.

The cost of a solution $(\mathcal{F}, \delta)$ is $\mathsf{cost}(\mathcal{F}, \delta) = \sum_{(u,v)\in E} c(u,v) \cdot \delta(F(u), F(v))$, and its *size* is $|\mathcal{F}|$.

From (LP1) and Proposition 4.2 in [1], it is proved that:

▶ **Proposition 20.** *Given a graph G and a subset T of its terminals, and a function f, if for every length $\{\ell_e\}_{e\in E(G)}$ and every demand D, the instance $(G, T, \ell, \mathcal{D})$ of $\mathsf{0EwSN}_{AGK}$ has a solution $(\mathcal{F}, \delta)$ with size $|\mathcal{F}| \leq f(k)$ and cost*

$$\sum_{(u,v)\in E} c(u,v) \cdot \delta(F(u), F(v)) \leq q \cdot \sum_{(u,v)\in E} c(u,v) \cdot \mathsf{dist}_\ell(u,v),$$

then there is a quality-$(1+\varepsilon)q$ flow sparsifier H for G w.r.t T with $|V(H)| \leq (f(k))^{(\log k/\varepsilon)^{k^2}}$.

The 0EwSN problem

In studying the integrality gap of (LP-Metric), we are essentially considering the following problem.

The input consists of

- an edge-capacitated graph $G = (V, E, c)$, with length $\{\ell_e\}_{e\in E}$ on its edges; and
- a set $T \subseteq V$ of k terminals.

A solution consists of

- a partition $\mathcal{F}$ of V with $|\mathcal{F}|$, such that distinct terminals of T belong to different sets in $\mathcal{F}$; for each vertex $u \in V$, we denote by $F(u)$ the cluster in $\mathcal{F}$ that contains it;
- a semi-metric δ on the clusters in $\mathcal{F}$, such that for all pairs $t, t' \in T$, $\delta(F(t), F(t')) = \mathsf{dist}_\ell(t,t')$, where $\mathsf{dist}_\ell(\cdot,\cdot)$ is the shortest-path distance (in G) metric induced by edge length $\{\ell_e\}_{e\in E(G)}$.

The cost and the size of a solution is defined in the same way as $\mathsf{0EwSN}_{\mathsf{AGK}}$.

The difference between two problems are underlined. Specifically, in $\mathsf{0EwSN}_{\mathsf{AGK}}$ it is only required that some "average terminal distance" does not decrease, while in our problem it is required that all pairwise distances between terminals are preserved. Clearly, our requirement for a solution is stronger, which implies that any valid solution to our instance is also a valid solution to the same $\mathsf{0EwSN}_{\mathsf{AGK}}$ instance (with arbitrary $\mathcal{D}$). Therefore, we have the following corollary.

▶ **Corollary 21.** *Given a graph G and a subset T of its terminals, and a function f, if for every length $\{\ell_e\}_{e\in E(G)}$, the instance (G,T,ℓ) of* 0EwSN *has a solution $(\mathcal{F},\delta)$ with size $|\mathcal{F}| \le f(k)$ and cost*

$$\sum_{(u,v)\in E} c(u,v)\cdot\delta(F(u),F(v)) \le q\cdot\sum_{(u,v)\in E} c(u,v)\cdot \mathsf{dist}_\ell(u,v),$$

then there is a quality-$(1+\varepsilon)q$ flow sparsifier H for G w.r.t T with $|V(H)| \le (f(k))^{(\log k/\varepsilon)^{k^2}}$.

On the other hand, the main result of our paper is a lower bound for the 0EwSN problem. As 0EwSN has stronger requirement (for solutions) than $\mathsf{0EwSN}_{\mathsf{AGK}}$, our lower bound does not immediately imply a lower bound for $\mathsf{0EwSN}_{\mathsf{AGK}}$ or for the flow sparsifier. However, if we can show that, for some function f, there exists a graph G, a terminal set T with size k, and a demand $\mathcal{D}$ on T, such that any solution $(\mathcal{F},\delta)$ with size $|\mathcal{F}| \le f(k)$ has cost at least

$$\sum_{(u,v)\in E} c(u,v)\cdot\delta(F(u),F(v)) \ge q\cdot\sum_{(u,v)\in E} c(u,v)\cdot \mathsf{dist}_\ell(u,v),$$

then this, from the (LP1) and the discussion in [1], implies that any quality-$o(q)$ contraction-based flow sparsifier for G has size at least $f(k)$.

Solving Woeginger's Hiking Problem: Wonderful Partitions in Anonymous Hedonic Games

Andrei Constantinescu[1]
ETH Zürich, Switzerland

Pascal Lenzner
Hasso Plattner Institute, University of Potsdam, Germany

Rebecca Reiffenhäuser
University of Amsterdam, The Netherlands

Daniel Schmand
University of Bremen, Germany

Giovanna Varricchio
University of Calabria, Rende, Italy

Abstract

A decade ago, Gerhard Woeginger posed an open problem that became well-known as "Woeginger's Hiking Problem": Consider a group of n people that want to go hiking; everyone expresses preferences over the size of their hiking group in the form of an interval between 1 and n. Is it possible to efficiently assign the n people to a set of hiking subgroups so that every person approves the size of their assigned subgroup? The problem is also known as efficiently deciding if an instance of an anonymous Hedonic Game with interval approval preferences admits a wonderful partition.

We resolve the open problem in the affirmative by presenting an $O(n^5)$ time algorithm for Woeginger's Hiking Problem. Our solution is based on employing a dynamic programming approach for a specific rectangle stabbing problem from computational geometry. Moreover, we propose natural, more demanding extensions of the problem, e.g., maximizing the number of satisfied participants and variants with single-peaked preferences, and show that they are also efficiently solvable. Last but not least, we employ our solution to efficiently compute a partition that maximizes the egalitarian welfare for anonymous single-peaked Hedonic Games.

2012 ACM Subject Classification Theory of computation → Dynamic programming; Theory of computation → Algorithmic game theory; Theory of computation → Discrete optimization

Keywords and phrases Algorithmic Game Theory, Dynamic Programming, Anonymous Hedonic Games, Single-Peaked Preferences, Social Optimum, Wonderful Partitions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.48

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2311.02067` [17]

Funding *Giovanna Varricchio*: The author is grateful for the support by the PNRR MUR project FAIR – Future AI Research (PE00000013), and the DFG, German Research Foundation, grant (Ho 3831/5-1).

Acknowledgements We would like to thank the Schloss Dagstuhl – Leibniz-Zentrum für Informatik who hosted the event "Computational Social Dynamics" (Seminar 22452) in November 2022 [24], where the work leading to this paper was started. We are grateful to Martin Hoefer, Sigal Oren, and Roger Wattenhofer for organizing this event. We additionally thank Roger Wattenhofer for the useful discussions concerning this work.

[1] Corresponding author.

1 Introduction

Suppose there are n attendees of a workshop, who aim to go for a joint hike during a break. Keeping the whole group together is logistically challenging, so typically the attendees split into smaller subgroups that will do the hike together. It is natural that different attendees might have different preferences for the sizes of the subgroups they will eventually join. In particular, each attendee i reports an interval $[\ell_i, r_i]$ signifying that they will be content if they are in a group of size $s \in [\ell_i, r_i]$. The organizers of the hike now face the following problem: Is there a polynomial time algorithm that determines whether there is a partition of the attendees into subgroups such that they are all content with the sizes of their subgroups?

In 2013, Gerhard Woeginger famously phrased this problem underlying an anonymous hedonic game to explain an open question stemming from his work [29]. It is one of the very nice problems that he used to share at various occurrences, e.g., at the coffee machine, while waiting in a seminar room, or even during a joint walk.[2] As this is one of the problems he explained to many people, it became known as Woeginger's *Hiking Problem*.

In this work we answer Woeginger's question in the affirmative by exploiting a tight connection to a variant of the rectangle stabbing problem from computational geometry that can be solved in polynomial time via elegant dynamic programming.

Using Woeginger's motivation of the problem, many natural extensions arise that we introduce in this paper. If the sought partition does not exist, then what is the maximum number of attendees that can be satisfied with their group sizes, or what is the minimum number of attendees to exclude such that the remaining ones can be partitioned to all become satisfied? Moreover, we also consider a version where the hikers have single-peaked preferences over the group sizes and a partition is sought that minimizes the utilitarian or egalitarian cost, where cost is defined as a function of the assigned and the ideal group size. We show that these more demanding problems can also be solved efficiently. Finally, we discuss the relationship between the hiking problem and the problem of maximizing the egalitarian welfare in (general) anonymous Hedonic Games.

1.1 Related Work

Hedonic Games (HGs), introduced by Dreze and Greenberg in [19], model multi-agent systems where selfish agents have to be partitioned into coalitions and have preferences over the possible outcomes. Such games are called hedonic as agents' preferences only depend on the coalition they belong to but not on how the other agents are grouped. HGs have been widely studied (see [7] for a survey) and numerous prominent subclasses have been identified based on properties of the agents' preferences or other possible constraints: [8, 14, 1, 26, 21, 2]. Simple examples of such classes are anonymous HGs [9], where the preferences of the agents depend only on the size of their coalition and not on the individual participants, or HGs with approval-based (Boolean) preferences [6], where agents have binary values for the coalitions. Woeginger's Hiking Problem resides in the intersection of these classes.

The HGs literature has typically focused on the existence and computation of stable or optimal solutions, see, e.g., [14]. The most desirable, ideal partition is the one where every agent is assigned to one of her best coalitions – called *perfect* (or *wonderful*) *partition* [3].

[2] Gerhard's ability to explain open research questions in an easily accessible way has been extraordinarily motivating. In particular, this work would not have started without Gerhard meeting one of the authors for lunch and explaining the problem exactly in this way. With this work we contribute to the recent line of publications celebrating the life and work of Gerhard Woeginger, see [25].

Unfortunately, such a solution rarely exists and the related decision or computational problem is usually hard [3, 27]. Even in simple cases such as anonymous approval-based HGs, it is NP-complete to determine the existence of a wonderful partition [18, 29]. Such a result holds true even if the number of approved coalition sizes of each agent is at most 2 [18]. The problem of finding a wonderful partition in Hedonic Games is related to the one of maximizing the utilitarian welfare, i.e. the sum of agents' utilities, for Boolean utilities. An overall picture of the complexity of finding wonderful partitions in Boolean Hedonic Games, including anonymous ones, is given in [27]. For general utility functions, the problem of maximizing the utilitarian or egalitarian social welfare has been studied in various settings, e.g., in fractional [5] and additively separable Hedonic Games [4]. To the best of our knowledge, these objectives have not been considered in the context of anonymous Hedonic Games.

In our paper, we show the tractability of computing a wonderful partition for instances with interval approvals as formulated by Woeginger [29]. To this aim, we provide a dynamic program that relies on the approach used in [22] to solve a capacitated rectangle stabbing problem from computational geometry. Here, the goal is to stab a set of rectangles with a minimum subset of a given set of lines, each intersecting (i.e., potentially *stabbing*) some of the rectangles. Each line has a maximum number of rectangles it can stab, and to stab all rectangles, one is allowed to use multiple copies of each line.

We also consider variants with single-peaked cost functions of the agents. Single-peaked preferences date back to Black [12]. Such preferences are a common theme in the Economics and Game Theory literature. In particular, they play a prominent role in different fields such as Hedonic Diversity Games [16, 13], Schelling Games [11, 23], and in various works on voting and social choice [28, 30, 10, 20, 15].

1.2 Model

The hiking problem as formulated by Gerhard Woeginger is a special case of anonymous Hedonic Games with approval-based preferences. In an anonymous Hedonic Game with approval-based preferences we are given a set N of n agents to be partitioned into coalitions. Each agent $i \in N$ reports an approval set $S_i \subseteq [n]$ representing the approved group sizes for agent i. In particular, agent i wants to be in some group of size s such that $s \in S_i$. An approval set S_i is said to be an *interval* if $S_i = \{\ell_i, \ell_i + 1, \dots, r_i\}$, for some $1 \leq \ell_i \leq r_i \leq n$. The agents have to be partitioned into coalitions, i.e., subsets of the agent set. This induces a *partition* π of the set of agents N. We denote by $\pi(i)$ the coalition agent i belongs to in the partition π. We follow the notation in Woeginger's survey paper [29] and call a partition π *wonderful* if each agent approves of the size of its coalition in π, i.e., for each agent i, $|\pi(i)| \in S_i$. This leads to the following natural computational problem called Wonderful-Partition.

Wonderful-Partition
Input: A set N of agents and size approval sets $(S_i)_{i \in N}$.
Problem: Decide whether there exists a wonderful partition of the agents. If yes, compute one.

Woeginger's Hiking Problem is Wonderful-Partition with interval approval sets, which we formally define as follows.

Wonderful-Partition-Intervals (Hiking)
Input: A set N of agents and for each agent $i \in N$ two numbers $\ell_i \leq r_i$ such that $S_i = \{\ell_i, \ldots, r_i\}$.
Problem: Decide whether there exists a wonderful partition of the agents. If yes, compute one.

We also consider natural extensions of the hiking problem, to be introduced later.

1.3 Our Contribution

We solve Woeginger's Hiking Problem in the affirmative by giving an $O(n^5)$-algorithm that computes a wonderful partition for an instance with n agents that each have interval approval sets (see Figure 1 (a)), if such a partition exists. For this we use a dynamic programming approach for a rectangle stabbing problem from computational geometry. Moreover, we extend this approach to achieve an $O(n^5)$-algorithm for computing the minimum set of hikers that have to be excluded from the hike, in order for a wonderful partition to become possible. We also give an $O(n^7)$-algorithm for deciding if a wonderful partition exists if exactly x hikers are excluded. This is then used to derive an $O(n^7 \log n)$-algorithm for finding a partition that maximizes the number of hikers that approve of their assigned group size. These approaches can also be extended to a setting where hikers have weights.

All these positive results hold for the case where the agents have interval approval sets (Figure 1 (a)). The special case where every hiker only approves of one group size, i.e., approval intervals of size 1, can be solved efficiently by checking if for all i the number of agents approving only of size i is divisible by i. In contrast to this, it was already known that the problem is NP-hard even if each attendee has at most two different approved group sizes that need not form an interval [18] (see Figure 1 (b)). We extend the original proof slightly by establishing a connection to a graph orientation problem. For the sake of completeness and to improve readability, it can be found in the full version [17].

We complete the picture by considering agents with single-peaked preferences over group sizes. In this setting an agent has a cost that is a function of its assigned group size and its ideal group size. If all agents incur a cost that is given by a fixed function that depends on their peak (but different peak values are allowed, see Figure 1 (c)) and if this function satisfies a mild technical condition, we compute a partition that minimizes the social cost in time $O(n^2(\alpha + 1))$, if at most α agents can be excluded from the hike. This holds for the utilitarian setting, i.e., minimizing the sum of the agents' costs, and also for the egalitarian setting, where the maximum agent cost is to be minimized. Finally, we prove polynomial time

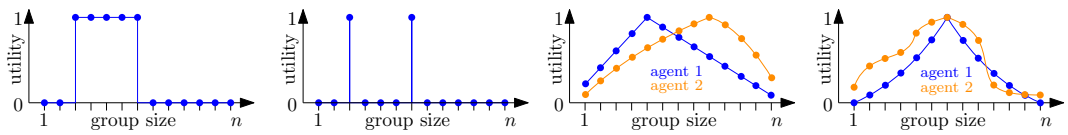


Figure 1 Examples of utility functions considered in this paper. (a) interval approval sets, (b) non-interval approval sets with two approved group sizes, (c) utility functions are single-peaked and all agents with the same peak have the same utility function (under some additional mild technical assumptions), (d) individual single-peaked utility functions for all agents (as shown, the functions may differ even if they have the same peak).

equivalence of the wonderful partition problem and the problem of minimizing the egalitarian social cost in (general) anonymous Hedonic Games. We use this result to show that even for the setting where each agent has its own single-peaked cost function (see Figure 1 (d)), a partition that minimizes the egalitarian social cost can be computed in $O(n^5 \log n)$.

All omitted details can be found in the full version [17].

2 Efficient Algorithm for Woeginger's Hiking Problem, and Extensions

We prove that Wonderful-Partition-Intervals can be solved in polynomial time by casting the problem as a version of capacitated rectangle stabbing, which can be efficiently solved via an elegant dynamic programming approach introduced by Even, Levi, Rawitz, Schieber, Shahar, and Srividenko [22]. In the capacitated rectangle stabbing problem, we are given axis parallel rectangles and a set of lines. The goal is to find a minimum number of lines that intersect and stab all rectangles. Each line has a maximum number of intersecting rectangles it can stab, and to stab all rectangles, one is allowed to use multiple copies of each line. Woeginger's Hiking Problem can be seen as a special case of the capacitated rectangle stabbing problem in one dimension, where the agents' intervals correspond to 1-dimensional rectangles and stabbing rectangles at some integer position i corresponds to creating a partition of size i where the respective agents belonging to the stabbed rectangles are included. Using this relationship, we give a simpler and faster $\mathcal{O}(n^5)$ dynamic program that is specifically tailored to our needs. Note that this reduction from our partition problem to a version of rectangle stabbing draws a powerful new connection between the problem types that is not limited to our specific problem version, but is possible in more generality; e.g., it can be adapted to the variant where there can be at most one group of each size.

In the following, we use the central observations of Even, Levi, Rawitz, Schieber, Shahar, and Srividenko [22] to derive a dynamic programming solution to Woeginger's Hiking problem, i.e., to Wonderful-Partition-Intervals. The definition and analysis of which will then also power our results on the more demanding problem variants discussed in the introduction.

2.1 Dynamic Program for Woeginger's Hiking Problem

Consider an instance of Wonderful-Partition-Intervals, i.e., a set N of n agents, without loss of generality $N = [n]$, and n pairs $(\ell_i, r_i)_{i \in N}$ such that agent $i \in [n]$ approves of sizes $S_i = \{\ell_i, \dots, r_i\}$. We are interested in checking whether there exists a wonderful partition of the agents; i.e., one where every agent approves of their coalition size. For consistent tie-breaking reasons, throughout this section we write $i \prec j$ for two distinct agents $i, j \in [n]$ if either $r_i < r_j$, or $r_i = r_j$ and $i < j$. Note that $\prec$ is a well-defined strict linear order and, without loss of generality, assume that agents are ordered so that $1 \prec \dots \prec n$.

As the main technical ingredient of our approach, given a subset $N' \subseteq N$ of agents, we say that a wonderful partition π of N' is *earliest-due-date* if for any two agents $i \prec j$ it does not hold that $\ell_i \leq |\pi(j)| < |\pi(i)|$. This terminology is borrowed from the scheduling literature, and based on the following well-known observation: let us view the agents' allowed intervals of coalition sizes on an axis labeled with the natural numbers $1, \dots, n$, and consider an arbitrary collection of coalition sizes we might decide on using, in non-decreasing order of coalition size. Then, going in this order, it is always safe to include those agents first whose right interval endpoints are the smallest: in said order of coalitions, they are the ones that stop being servable first, making an earliest-due-date approach impose the least restrictions on later assignments. In other words, whenever a wonderful partition into the given coalition sizes exists, there also exists one that is earliest-due-date. We prove this formally below for completeness. Note that, to make the process well-defined, instead of comparing agents by right endpoints, we compare them by $\prec$.

▶ **Lemma 1.** *If a subset $N' \subseteq N$ of agents admits a wonderful partition, then it admits an earliest-due-date wonderful partition.*

Proof. Consider a wonderful partition π of N'. If it satisfies the required property, then we are done, otherwise, consider $i, j \in N'$ such that $i \prec j$ and $\ell_i \leq |\pi(j)| < |\pi(i)|$. These conditions imply that $|\pi(j)| \in [\ell_i, r_i]$ and $|\pi(i)| \in [\ell_j, r_j]$. Hence, one can construct a new wonderful partition π' from π by exchanging the groups of agents i and j; i.e., $\pi'(i) = (\pi(j) \setminus \{j\}) \cup \{i\}$ and $\pi'(j) = (\pi(i) \setminus \{i\}) \cup \{j\}$. Subsequently, set $\pi \leftarrow \pi'$ and repeat the same procedure until the required condition is satisfied. To complete the proof, we need to show that this is eventually the case. To do so, since $1 \prec \ldots \prec n$, each exchange strictly increases the sequence $(|\pi(n)|, \ldots, |\pi(1)|)$ lexicographically. Because this sequence is bounded from above by $(n, \ldots, n)$, the process eventually ends. ◀

Hence, from now on, we will only seek earliest-due-date wonderful partitions. This crucial restriction, which we have shown to be without loss of generality, will allow us to bootstrap a dynamic programming algorithm that determines a wonderful partition if one exists.

To construct our algorithm, we first show that earliest-due-date partitions admit an attractive recursive decomposition. Consider an earliest-due-date wonderful partition π of the agents (if any exist), and consider the size of the coalition that agent n is a part of; i.e., $|\pi(n)|$. Moreover, consider an arbitrary agent $i \neq n$. Agent i is part of a coalition of size $|\pi(i)|$. Because π is earliest-due-date, by definition it can not be the case that $\ell_i \leq |\pi(n)| < |\pi(i)|$, so either $|\pi(n)| < \ell_i$ or $|\pi(i)| \leq |\pi(n)|$ holds.

▶ **Lemma 2.** *Consider an earliest-due-date wonderful partition π of N. Partition the agents as $N = N_- \cup N_+ \cup \{n\}$, where $N_- = \{i \in N \setminus \{n\} \mid \ell_i \leq |\pi(n)|\}$ and $N_+ = \{i \in N \setminus \{n\} \mid \ell_i > |\pi(n)|\}$, then it holds that:*

1. *For all $i \in N_-$ we have $|\pi(i)| \leq |\pi(n)|$;*
2. *For all $i \in N_+$ we have $|\pi(i)| > |\pi(n)|$.*

Proof. For (2) note that any $i \in N_+$ by definition satisfies $\ell_i > |\pi(n)|$. Since we have the requirement that $|\pi(i)| \in [\ell_i, r_i]$, this means that $|\pi(i)| \geq \ell_i > |\pi(n)|$, as desired.

For (1), consider some $i \in N_-$. By definition, we have that $\ell_i \leq |\pi(n)|$. Assume for a contradiction that $|\pi(i)| > |\pi(n)|$. This implies that $\ell_i \leq |\pi(n)| < |\pi(i)|$. Since $r_i \leq r_n$ and by the sorting criterion we also have $i \prec n$, which contradicts that π is earliest-due-date. ◀

Notably, the same result holds true if we work with a subset of the agents $N' \subsetneq N$ and n is replaced with the agent $a \in N'$ with maximum r_a.

Armed as such, to build intuition, Lemma 2 tells us that no two agents $a_- \in N_-$ and $a_+ \in N_+$ can go into the same group, so a first attempt at a recursive algorithm looking for a wonderful partition of N, to be later optimized by memorization/dynamic programming, would proceed as follows: start with $N' = N$; at each step, identify the agent $a \in N'$ maximizing r_a and exhaust over all possibilities for $|\pi(a)|$; for each one, write $N' = N'_- \cup N'_+ \cup \{a\}$ as defined in Lemma 2 and recurse with N'_- and N'_+. We will specify the details of the process such that, if the recursive calls yield wonderful partitions of N'_- and N'_+, a wonderful partition of N can also be constructed by incorporating agent a into them. Of course, the last step of reasoning is incorrect, since one needs to make sure that a group of size $|\pi(a)|$ with one space available indeed exists, and this information needs to be somehow propagated across recursive calls. Moreover, it is unclear what the number of sets N' reachable by the recursion is. This has to be polynomially bounded so that memorization/dynamic programming leads to a polynomial-time algorithm.

Let us first address the second issue outlined above, namely the size of the state space. This turns out to be relatively simple: define $N(x_1, x_2, i) = \{j \in N \mid j \leq i \text{ and } \ell_j \in [x_1, x_2]\}$. Then, we can adapt the recursive approach: instead of storing N' explicitly, start with $x_1 = 1$ and $x_2 = n$, as well as $i = n$, from which implicitly $N' = N(x_1, x_2, i)$. At each step we will first check whether i, which by the sorting criterion is the largest agent maximizing r_i, is in $N' = N(x_1, x_2, i)$; i.e., whether $\ell_i \in [x_1, x_2]$. If not, then $N(x_1, x_2, i) = N(x_1, x_2, i-1)$ and we simply recurse with the same x_1, x_2 and $i' = i - 1$. Otherwise, i is the largest agent in N' maximizing r_i, so we can exhaust as before over all possible values for $|\pi(i)|$ and perform two recursive calls in each case: one with $(x_1', x_2', i') = (x_1, |\pi(i)|, i-1)$ and one with $(x_1', x_2', i') = (|\pi(i)| + 1, x_2, i - 1)$. We have yet to solve the correctness issue, but we have made progress: the state space now consists of triples (x_1, x_2, i) such that $1 \leq x_1 \leq x_2 \leq n$ and $1 \leq i \leq n$, which there are $O(n^3)$ of.

Let us now turn our attention to ensuring correctness. To do so, we need to investigate more closely how to ensure that the group of size $|\pi(i)|$ of agent i (recall that we only exhaust over its size, not over which agents are in it) exists and is used to its full capacity in the solution constructed by the recursion. There is one crucial guarantee given by Lemma 2 that we have so far not exploited. Namely, the recursive call with $(x_1, |\pi(i)|, i-1)$ should only use groups of sizes $s \in [x_1, |\pi(i)|]$ and the one with $(x_1', x_2', i') = (|\pi(i)| + 1, x_2, i - 1)$ should only use groups of sizes $s \in [|\pi(i)| + 1, x_2]$. In general, state (x_1, x_2, i) should only consider groups of sizes $s \in [x_1, x_2]$. This can be ensured by requiring that the exhausted value for $|\pi(i)|$ additionally satisfies $x_1 \leq |\pi(i)| \leq x_2$. This small, seemingly inconsequential, refinement of the approach will ultimately allow us to propagate information about incomplete groups across states in the recursion. Before describing how this is done in general, to build more intuition, let us consider the first level of the recursion, namely $(x_1, x_2, i) = (1, n, n)$. At this level, the algorithm exhausts over the possible values for $|\pi(n)| \in [\ell_n, r_n]$. For each possibility, in the ensuing recursive call with $(x_1', x_2', i') = (|\pi(n)| + 1, n, n - 1)$ all used groups will be of size at least $|\pi(n)| + 1 > |\pi(n)|$ so nothing special needs to be done in this case since those agents can not be part of agent n's group. However, in the other call, having $(x_1', x_2', i') = (1, |\pi(n)|, n - 1)$ some $|\pi(n)| - 1$ other agents will need to share a group of size $|\pi(n)|$ with agent n, and this is not yet modeled by our approach. To model this effect across recursive calls, we introduce a fourth variable $0 \leq k < x_2$ to the state of our recursive algorithm; i.e., each state is now of the form (x_1, x_2, i, k). This variable intuitively signifies that there exists (from upward in the recursion) an incomplete group, currently of size k, whose final size should be x_2. With this setup, the starting top-level call will now be with $(x_1, x_2, i, k) = (1, n, n, 0)$. For each value of $|\pi(n)|$, the two resultant recursive calls will be with $(x_1', x_2', i', k') = (1, |\pi(n)|, n - 1, 1)$[3] and $(x_1', x_2', i', k') = (|\pi(n)| + 1, n, n - 1, 0)$. Note how the former call has $k' = 1$, signifying that we just "opened a new (incomplete) group of size one, whose final size should be $x_2' = |\pi(n)|$" The fundamental reason why such an approach can work is that in any node of the recursion tree, there is at most a single incomplete group to keep track of. Note that this fact crucially depends on each call (x_1, x_2, i, k) only considering partitions into groups of sizes between x_1 and x_2. We still need to describe the transitions for general calls (x_1, x_2, i, k) given the newly added parameter k. The formal details will following below, but in rough lines, the call for N_- creates a new group; i.e., $k' = 1$; while the call for N_+ keeps the currently open one; i.e., $k' = k$; the exception comes when $|\pi(i)| = x_2$, in which case the call for N_- adds to the same group, possibly closing the group; i.e., $k' = (k + 1) \pmod{x_2}$; and no call for N_+ is generated.

[3] Strictly speaking, this should be $(x_1', x_2', i', k') = (1, |\pi(n)|, n - 1, 1 \bmod |\pi(n)|)$. We omit this detail from the higher-level exposition to improve readability.

▶ **Theorem 3.** *Deciding whether a wonderful partition exists and computing one if so can be achieved in $O(n^5)$ time.*

Proof. We show how to solve the decision version. Constructing a wonderful partition for yes-instances can be subsequently achieved by standard techniques. We proceed by dynamic programming (DP). Define the Boolean DP array $dp[x_1, x_2, i, k]$ for $1 \leq x_1 \leq x_2 \leq n$, $0 \leq i \leq n$ and $0 \leq k < x_2$, with the meaning $dp[x_1, x_2, i, k] = 1$ if and only if there exists a wonderful partition of agents in $N(x_1, x_2, i)$ using only groups of sizes in $[x_1, x_2]$ and assuming that we start with an incomplete group of current size k which has to have final size x_2. Naturally, if $k = 0$ this means starting with no partially filled group. The final answer will be available at the end in $dp[1, n, n, 0]$. To compute the DP table, we use the following recurrence relations and base cases (using a hyphen as stand-in for any group size):

1. $dp[-, -, 0, 0] = 1$;
2. $dp[-, -, 0, k] = 0$ if $k \neq 0$;
3. $dp[x_1, x_2, i, k] = dp[x_1, x_2, i-1, k]$ if $\ell_i \notin [x_1, x_2]$;
4. $dp[x_1, x_2, i, k] = \bigvee_{x=\ell_i}^{\min(x_2, r_i)} F_x$ if $\ell_i \in [x_1, x_2]$, where

$$F_x := \begin{cases} dp[x_1, x_2, i-1, (k+1) \bmod x_2] & x = x_2 \\ dp[x_1, x, i-1, 1 \bmod x] \wedge dp[x+1, x_2, i-1, k] & x < x_2. \end{cases}$$

The first three cases are immediate from the definition of the DP. For the last case, we iterate over $x \in [\ell_i, \max(x_2, r_i)]$ which is agent i's group size. There are two cases: if $x = x_2$, then we put i into the group of current size k and final size x_2 that we assumed to have at our disposal; this increases the size of the group by one, or completes it if $k = x_2 - 1$; otherwise, $x < x_2$, and we recurse into partitioning agents in $N(x_1, x, i-1)$ into groups of sizes between x_1 and x, and $N(x+1, x_2, i-1)$ into groups of sizes between $x+1$ and x_2. For the first call, this generates a new group of size 1 (unless $x = 1$), while for the latter, this keeps the previously open group of current size k and final size x_2 that we assumed to have.

To compute the DP table in an acyclic fashion, it suffices to iterate through i in ascending order. The complexity of the approach is $O(n^5)$ because there are $O(n^4)$ states and computing the value for states of type (4) requires iterating through $O(n)$ values of x. ◀

2.2 Extensions

Using a similar DP approach, we can solve the following natural extension.

Hiking-Min-Delete
Input: A set N of agents and for each agent $i \in N$ two numbers $\ell_i \leq r_i$ such that $S_i = \{\ell_i, \ldots, r_i\}$.
Problem: Compute a set $N' \subseteq N$ of minimum size such that $N \setminus N'$ has a wonderful partition. Output N' and a wonderful partition of $N \setminus N'$.

The approach relies on essentially the same recursive reasoning as before, except that we now also consider the possibility of "ignoring" an agent i and hence recursing with $(x_1', x_2', i', k') = (x_1, x_2, i-1, k)$. Moreover, instead of making the recursion return whether a wonderful partition is possible or not, we make it return the minimum number of agents that need to be removed so that this is possible. In light of this, the "ignore i" recursive call incurs a cost of 1 removed agent. The details are formalized in the following.

▶ **Theorem 4.** *Hiking-Min-Delete is solvable in $O(n^5)$ time.*

Proof. We use a similar approach as in the proof of Theorem 3. As before, it suffices to show how to compute the minimum size of N', as computing such a set N' and a corresponding wonderful partition for $N \setminus N'$ follow by standard techniques. Define the integer-valued DP array $dp[x_1, x_2, i, k]$ for $1 \leq x_1 \leq x_2 \leq n$, $0 \leq i \leq n$ and $0 \leq k < x_2$, with the meaning that $dp[x_1, x_2, i, k]$ contains the minimum number of agents which need to be removed from $N(x_1, x_2, i)$ so that the remaining agents admit a wonderful partition into groups of sizes in $[x_1, x_2]$ assuming that we start with an incomplete group of current size k which has to have final size x_2. Note that, in contrast to the previous DP, the value 0 corresponds to removing no agents, which is the best possible outcome, while previously it corresponded to the need for removing at least one agent. The final answer will be available at the end in $dp[1, n, n, 0]$. To compute the DP table, we use the following recurrence relations and base cases:

1. $dp[-, -, 0, 0] = 0$;
2. $dp[-, -, 0, k] = \infty$ if $k \neq 0$;
3. $dp[x_1, x_2, i, k] = dp[x_1, x_2, i-1, k]$ if $\ell_i \notin [x_1, x_2]$;
4. $dp[x_1, x_2, i, k] = \min\{1 + dp[x_1, x_2, i-1, k], X\}$ if $\ell_i \in [x_1, x_2]$, where

$$X := \min\{F_x \mid \ell_i \leq x \leq \min(x_2, r_i)\}$$

and

$$F_x := \begin{cases} dp[x_1, x_2, i-1, (k+1) \bmod x_2] & x = x_2 \\ dp[x_1, x, i-1, 1 \bmod x] + dp[x+1, x_2, i-1, k] & x < x_2. \end{cases}$$

The reasoning stays largely the same as before, with the only difference being the new $1 + dp[x_1, x_2, i-1, k]$ term, which accounts for discarding agent i and incurring a cost of 1 corresponding to removing an agent. ◀

Another subtly distinct natural variant of Woeginger's Hiking Problem is the following:

Hiking-Max-Satisfied
Input: A set N of agents and for each agent $i \in N$ two numbers $\ell_i \leq r_i$ such that $S_i = \{\ell_i, \ldots, r_i\}$.
Problem: Compute a partition π of N maximizing the number of agents approving of their coalition sizes.

Indeed, Hiking-Max-Satisfied and Hiking-Min-Delete are related, in that if N' is a minimum-size set of agents such that $N \setminus N'$ has a wonderful partition, then it is also possible to satisfy at least $n - |N'|$ agents in a partition of N. This is because agents in N' can be put together in a group in tandem to a wonderful partition of $N \setminus N'$ to get a partition of N with at least $n - |N'|$ satisfied agents. However, it might be possible to satisfy more than $n - |N'|$ agents if unsatisfied agents do not all go into the same group.

Hence, solving this variant of the problem introduces new challenges that require further insight. Let us fix a number k and ask whether it is possible to satisfy at least $|N| - k$ agents. If the answer is affirmative, we also want a partition achieving this. To solve Hiking-Max-Satisfied, we will binary search for the smallest $0 \leq k \leq |N|$ for which the answer is affirmative, call it k^*, and then recover a partition for k^*. It remains to show how to solve the problem for a fixed value of k. To do so, we need to adjust the angle from which we look at the problem. In particular, instead of looking for a partition satisfying at least k agents, we will look for a size-k subset $N' \subseteq N$ (corresponding to k agents which we do not

require to be satisfied) such that $(N \setminus N') \cup D_k$ admits a wonderful partition, where D_k is a set of k dummy agents happy with any group size. Intuitively, k agents are replaced with dummies not minding their group size. Because it is always no worse to remove agents from N in contrast to removing agents from D_k, it is enough to ask to remove exactly k agents from $N \cup D_k$ such that the remaining agents admit a wonderful partition. Checking whether this is possible and finding a corresponding wonderful partition reduces to the following more general problem, which we will show can be solved in polynomial time.

Hiking-x-Delete
Input: A set N of agents, for each agent $i \in N$ two numbers $\ell_i \leq r_i$ such that $S_i = \{\ell_i, \dots, r_i\}$, and also a number $0 \leq x \leq |N|$.
Problem: Compute a set $N' \subseteq N$ of size x such that $N \setminus N'$ has a wonderful partition (or report impossibility). Output N' and a wonderful partition of $N \setminus N'$.

We now show how to solve Hiking-x-Delete in polynomial time. We can once again try to attack the problem recursively with our usual state (x_1, x_2, i, k). Just like we did for Hiking-Min-Delete, we will have recursive calls for ignoring an agent; i.e., adding it to the set N' of removed agents.[4] In order to ensure that exactly x agents are removed, we add another variable r to the state of the recursion: (x_1, x_2, i, k, r), where r is how many agents we want to remove. The top-level call will be invoked with $r = x$. The recursive approach for Hiking-Min-Delete now translates relatively swiftly to the new setting. The main difference is the case where a state of the form $(x_1, x_2, i, k, -)$ with $\ell_i \in [x_1, x_2]$ performs two recursive calls to $(x_1, x, i-1, 1, -)$ and $(x+1, x_2, i-1, k, -)$. In particular, say the state is (x_1, x_2, i, k, r), then what should be the values r' and r'' for the two recursive calls? Intuitively, there is no fixed answer, since it could be that we remove more agents in the first or in the second call. In fact, we have full freedom over how to split the r removals across the two calls as long as $r' + r'' = r$. Hence, we will iterate over all options $0 \leq r' \leq r$ and set $r'' = r - r'$, and in each case call recursively with $(x_1, x, i-1, 1, r')$ and $(x+1, x_2, i-1, k, r'')$.

▶ **Theorem 5.** *Hiking-x-Delete is solvable in $O(n^7)$ time.*

Proof. As before, it suffices to check feasibility, a solution can then be recovered using standard techniques. Define the Boolean DP array $dp[x_1, x_2, i, k, r]$ for $1 \leq x_1 \leq x_2 \leq n$, $0 \leq i \leq n$, $0 \leq k < x_2$, and $0 \leq r \leq x$ with the meaning that $dp[x_1, x_2, i, k, r] = 1$ if and only if there exist r agents that can be removed from $N(x_1, x_2, i)$ such that the remaining agents admit a wonderful partition into groups of sizes in $[x_1, x_2]$ assuming that we start with an incomplete group of current size k which has to have final size x_2. At the end, $dp[1, n, n, 0, x]$ will be 1 if and only if it is possible to remove x agents from N such that the remaining agents admit a wonderful partition. To compute the DP table, we use the following:

1. $dp[-, -, 0, 0, 0] = 1$;
2. $dp[-, -, 0, k, r] = 0$ if $k \neq 0$ or $r \neq 0$;
3. $dp[x_1, x_2, i, k, r] = dp[x_1, x_2, i-1, k, r]$ if $\ell_i \notin [x_1, x_2]$;
4. $dp[x_1, x_2, i, k, r] = dp[x_1, x_2, i-1, k, r-1, d] \vee X$[5] if $\ell_i \in [x_1, x_2]$, where $X := \bigvee_{x=\ell_i}^{\min(x_2, r_i)} F_x$ and

$$F_x := \begin{cases} dp[x_1, x_2, i-1, (k+1) \bmod x_2, r] & x = x_2 \\ \bigvee_{r'=0}^{r} \left(dp[x_1, x, i-1, 1 \bmod x, r'] \wedge dp[x+1, x_2, i-1, k, r-r'] \right) & x < x_2. \end{cases}$$

[4] Note that previously we used N' to denote $N(x_1, x_2, i)$ when discussing the recursive algorithm for the original hiking problem. We do not keep this notation here.
[5] Only X for $r = 0$ as otherwise we would be referring to the invalid value $r = -1$.

We explain the four cases separately: for cases (1) and (2) we have $i = 0$; i.e., the set of yet-to-be-considered agents is empty, so $k = r = 0$ is necessary and sufficient for the table to store a 1, signifying feasibility. As before, case (3) straightforwardly ignores an agent that is not part of the current set of active agents.

The more interesting case is case (4). First, the $dp[x_1, x_2, i-1, k, r-1]$ term corresponds to removing an agent, and it only applies to $r \geq 1$, as in the footnote. This decreases r by 1 since we have just removed an agent. The $X := \bigvee_{x=\ell_i}^{\min(x_2,r_i)} F_x$ term is more involved. The selection of $\ell_i \leq x \leq \min(x_2, r_i)$ corresponds to selecting the size of the group that agent i will be part of. For $x = x_2$, the analysis is similar to our previous DPs. For $x < x_2$, on the other hand, we moreover split the r agent removals into r' removals for the first recursive call and $r'' = r - r'$ for the second. Whether both calls are successful is represented by the expression $dp[x_1, x, i-1, 1 \bmod x, r'] \wedge dp[x+1, x_2, i-1, k, r'']$.

As before, to compute the DP table in an acyclic fashion, it suffices to iterate through i in ascending order. The complexity is $O(n^7)$ because there are $O(n^5)$ states and computing the value for states of type (4) requires iterating through $O(n^2)$ values for (x, r). ◀

Using the above binary search approach we get a solution for Hiking-Max-Satisfied that runs only a $O(\log n)$ factor slower than the runtime for Hiking-x-Delete.

▶ **Theorem 6.** *Hiking-Max-Satisfied is solvable in $O(n^7 \log n)$ time.*

2.3 Further Weighted Extensions

If not all agents can be satisfied, Hiking-Min-Delete and Hiking-Max-Satisfied provide two ways of implementing a compromise. However, both treat unsatisfied/deleted agents equally. In certain settings, it might be more desirable to take into account the different *entitlements* of the agents; i.e., one agent might have been dissatisfied with their group size during the previous edition of the workshop, or another agent might be the senior invited speaker. One modelling option is to assign a weight w_i to each agent $i \in N$ and weigh the dissatisfied/deleted agents accordingly leading to the following variants:

Hiking-Min-Delete-Weighted
Input: A set N of agents and for each agent $i \in N$ two numbers $\ell_i \leq r_i$ such that $S_i = \{\ell_i, \dots, r_i\}$. Moreover, for each agent $i \in N$, a number $w_i \in \mathbb{R}_{\geq 0}$.
Problem: Compute a set $N' \subseteq N$ minimizing $\sum_{i \in N'} w_i$ such that $N \setminus N'$ has a wonderful partition. Output N' and a wonderful partition of $N \setminus N'$.

Hiking-Max-Satisfied-Weighted
Input: A set N of agents and for each agent $i \in N$ two numbers $\ell_i \leq r_i$ such that $S_i = \{\ell_i, \dots, r_i\}$. Moreover, for each agent $i \in N$, a number $w_i \in \mathbb{R}_{\geq 0}$.
Problem: Compute a partition π of the agents such that if N_π is the set of agents approving of their coalition sizes in π, then $\sum_{i \in N_\pi} w_i$ is maximized.

Our dynamic programs, with minor modifications which we sketch next, can also be used to solve the weighted variants. We begin with Hiking-Min-Delete-Weighted.

▶ **Theorem 7.** *Hiking-Min-Delete-Weighted is solvable in $O(n^5)$ time.*

Proof Sketch. In the proof of Theorem 4, we defined the DP as follows: "$dp[x_1, x_2, i, k]$ contains the minimum number of agents which need to be removed from $N(x_1, x_2, i)$ so that the remaining agents admit a wonderful partition into groups of sizes in $[x_1, x_2]$ assuming

that we start with an incomplete group of current size k which has to have final size x_2". To handle weights, this is replaced by "$dp[x_1, x_2, i, k]$ contains the minimum total weight of agents which need to be removed from $N(x_1, x_2, i)$ so that the remaining agents admit a wonderful partition into groups of sizes in $[x_1, x_2]$ assuming that we start with an incomplete group of current size k which has to have final size x_2". The rest of the proof stays the same, with the minor adaptation that the fourth recurrence relation accordingly becomes $dp[x_1, x_2, i, k] = \min\{w_i + dp[x_1, x_2, i-1, k], X\}$, where previously $w_i = 1$ has been used. ◀

To get a similar result for Hiking-Max-Satisfied-Weighted following our previous proof outline, we first define a weighted analogue of Hiking-x-Delete, as follows.

Hiking-x-Delete-Min-Weight
Input: A set N of agents, for each agent $i \in N$ two numbers $\ell_i \leq r_i$ such that $S_i = \{\ell_i, \ldots, r_i\}$, and also a number $0 \leq x \leq |N|$. Moreover, for each agent $i \in N$, a number $w_i \in \mathbb{R}_{\geq 0}$.
Problem: Compute a set $N' \subseteq N$ of size x minimizing $\sum_{i \in N'} w_i$ such that $N \setminus N'$ has a wonderful partition (or report impossibility). Output N' and a wonderful partition of $N \setminus N'$.

▶ **Theorem 8.** *Hiking-x-Delete-Min-Weight is solvable in $O(n^7)$ time.*

Proof Sketch. In the proof of Theorem 5 for the unweighted version, we defined a boolean DP as follows: "$dp[x_1, x_2, i, k, r] = 1$ if and only if there exist r agents that can be removed from $N(x_1, x_2, i)$ such that the remaining agents admit a wonderful partition into groups of sizes in $[x_1, x_2]$ assuming that we start with an incomplete group of current size k which has to have final size x_2". For the current problem, we want the states to signal not only possibility/impossibility but also what is the minimum total weight of those r removed agents. Hence, we replace this definition by "$dp[x_1, x_2, i, k, r]$ is the minimum total weight of r agents that can be removed from $N(x_1, x_2, i)$ such that [...] (or ∞ if impossible)". The rest of the reasoning stays analogous with minor changes: the values $0, 1$ in the base cases become $\infty, 0$, disjunctions ($\vee$) are replaced by "min" and conjunctions ($\wedge$) by $+$. Finally, the fourth recurrence relation becomes $dp[x_1, x_2, i, k, r] = \min\{w_i + dp[x_1, x_2, i-1, k, r-1, d], X\}$. ◀

We make use of Theorem 8 to show the following.

▶ **Theorem 9.** *Hiking-Max-Satisfied-Weighted is solvable in $O(n^8)$ time.*

Proof Sketch. In the proof of Theorem 6 for the unweighted case, we were looking for the largest k such that there exists a size-k subset $N' \subseteq N$ (corresponding to k agents which we do not require to be satisfied) such that $(N \setminus N') \cup D_k$ admits a wonderful partition, where D_k was a set of k dummy agents happy with any group size. We then argued that we could relax to asking for a size-k subset $N' \subseteq (N \cup D_k)$ such that $(N \cup D_k) \setminus N'$ admits a wonderful partition, which can be done using our polynomial-time algorithm for Hiking-x-Delete-Min in Theorem 5. This time, we follow a similar approach, except without binary search: we will try out all values $0 \leq k \leq n$ and ask for a size-k subset $N' \subseteq (N \cup D_k)$ such that $(N \cup D_k) \setminus N'$ admits a wonderful partition. For a fixed k, out of all abiding N', we want one minimizing $\sum_{i \in N'} w_i$. Such an N' can be computed using our $O(n^7)$ algorithm for Hiking-x-Delete-Min-Weight in Theorem 8. We do this for all values $0 \leq k \leq n$ and take the minimum-weight solution, adding an extra $O(n)$ factor, so the overall complexity is $O(n^8)$. ◀

3 Single-Peaked Preferences over Group Sizes

We extend the binary version of the hiking problem by considering single-peaked preferences over group sizes. We assume that each agent i has an ideal group size s_i and the cost of agent i if placed in a group of size s is given by a cost function dependent on s_i and s.

Given these ingredients, minimizing the social cost can be done in two variants: a utilitarian variant and an egalitarian variant. In the utilitarian variant, the goal is to minimize the total cost of the agents, while in the egalitarian version we want the cost of the agent having the highest cost to be as low as possible, i.e., we replace summation with maximum. We are also interested in a variation of the problem where the hike organizers consider it reasonable to exclude at most α of the agents from the hike, the goal becoming to select the agents to remove and then to organize a hike with best social cost among the remaining agents.[6] Note that we assume that each excluded agent has a cost of 0. This is different from assuming that excluded agents have to hike alone, which is covered by the case $\alpha = 0$.

More formally, we assume that each agent $i \in N$ announces an ideal coalition size s_i; i.e., agent i would be most happy when belonging to a coalition of size s_i. Moreover, given some cost function $cost : N^2 \to \mathbb{R}$ agent i incurs a cost of $cost(s_i, s)$ if placed in a coalition of size $s \in [n]$ and a cost equal to 0 if not participating in the hike. We assume c to be monotone, i.e., $cost(s_i, s) \leq cost(s_i, s')$ if $s_i \leq s \leq s'$ or $s' \leq s \leq s_i$. Notice that since agent i incurs a disutility equal to $cost(s_i, s)$, where s_i is the most preferred size of i, the monotonicity condition on $cost(\cdot, \cdot)$ implies that the preferences of agents are single-peaked w.r.t. the natural ordering. We refer to the next section for a formal definition. Given a partition π, we recall that $\pi(i)$ denotes the coalition of agent i; if i is not participating in the hike we write $\pi(i) = \perp$. Furthermore, by slight abuse of notation, for an agent i we write $i \in \pi$ to indicate that agent i takes part in the hike; i.e., $\pi(i) \neq \perp$.

The utilitarian social cost of a partition π is given by $cost(\pi) = \sum_{i \in \pi} cost(s_i, |\pi(i)|)$ while the egalitarian social cost is given by $cost(\pi) = \max_{i \in \pi} cost(s_i, |\pi(i)|)$. The goal is, therefore, to find a partition minimizing the social cost under the constraint $|\{i \mid \pi(i) = \perp\}| \leq \alpha$, or equivalently, $|\{i \mid i \in \pi\}| \geq n - \alpha$, where α is the maximum number of agents that are allowed to not participate in the hike. Without loss of generality, we assume that $s_1 \leq \ldots \leq s_n$.

We begin by proving two structural properties of optimal solutions that will allow us to greatly reduce the space of solutions that have to be considered, hence enabling us later to give efficient dynamic programming algorithms computing optimal partitions for both the utilitarian and the egalitarian settings. For the utilitarian social cost we need a mild assumption on the cost function *cost*.

▶ **Definition 10.** *A function* $cost : N^2 \to \mathbb{R}$ *fulfills the* quadrangle inequality *if and only if*

$$cost(a, c) + cost(b, d) \leq cost(a, d) + cost(b, c) \text{ for all } a \leq b \leq c \leq d.$$

Analogously, it fulfills the reverse quadrangle inequality *if and only if*

$$cost(a, c) + cost(b, d) \leq cost(a, d) + cost(b, c) \text{ for all } a \geq b \geq c \geq d.$$

Note that quadrangle inequality and reverse quadrangle inequality are equivalent if *cost* is symmetric, i.e. $cost(a, b) = cost(b, a)$. Moreover, notice that if $cost(\cdot, \cdot)$ is the Euclidean distance on $\mathbb{R}$ or $cost(a, b) = |b - a|^k$ for any $k \geq 1$, then it satisfies both the aforementioned quadrangle inequalities. The first observation that our approach will hinge upon is that it is enough to consider *size-monotonic* partitions.

[6] The original problem can be seen to be the $\alpha = 0$ case.

▶ **Definition 11** (Size-Monotonicity). *A partition π is* size-monotonic *if for any two agents $i, j \in \pi$, with $i < j$, it holds that $|\pi(i)| \leq |\pi(j)|$.*

Roughly speaking, there are optimal solutions where all participating agents with lower preferred coalition sizes belong to smaller coalitions than agents with higher preferred sizes. We prove this fact in the following.

▶ **Lemma 12.** *The following properties hold.*

(a) *In the utilitarian setting, if $cost(\cdot, \cdot)$ is monotone and fulfills quadrangle inequality and reverse quadrangle inequality, then there exists a size-monotonic optimal solution.*

(b) *In the egalitarian setting, if $cost(\cdot, \cdot)$ is monotone, then there exists a size-monotonic optimal solution.*

Finally, we will show that it is enough to consider size-monotonic partitions which additionally are *compact*. The latter is defined as follows:

▶ **Definition 13** (Compactness). *A coalition C is* compact *if it is of the form $C = \{i, i + 1, \ldots, j\}$ for some $i \leq j$. A solution π is* compact *if all coalitions $C \in \pi$ are compact.*

We now prove that we can modify any optimal size-monotonic partition so that it is size-monotonic and compact.

▶ **Lemma 14.** *There exists an optimal partition which is size-monotonic and compact.*

With the above observations, we now know that it is enough to give an efficient algorithm to compute the best size-monotonic and compact solution. We do so in the following. In fact, our algorithm will only directly leverage compactness.[7] To begin, for any two agents $i \leq j$ define $c(i, j)$ to be the social cost induced by agents $i, i + 1, \ldots, j$ when forming the coalition $\{i, i + 1, \ldots, j\}$. In particular, $c(i, j) = \sum_{k=i}^{j} cost(s_k, j - i + 1)$ in the utilitarian case, and similarly with summation replaced by maximum in the egalitarian case. With this definition in place, note that selecting the best compact solution π with at most α agents not taking part in the hike amounts to selecting compact non-intersecting coalitions $C_1, \ldots, C_\ell$ where $C_k = \{a_k, a_k + 1, \ldots, b_k - 1, b_k\}$, such that $\sum_{k=1}^{\ell}(b_k - a_k + 1) \geq n - \alpha$, and the sum/maximum of $c(a_1, b_1), \ldots, c(a_\ell, b_\ell)$ is minimized. Without loss of generality we can assume that $b_1 < a_2$, $b_2 < a_3$, $\ldots$, $b_{\ell-1} < a_\ell$, i.e., we assume that the coalitions are sorted by index in increasing order. Before giving the actual algorithm, we note that, to get the best efficiency possible, we will need that the values $c(i, j)$, for all pairs (i, j) with $i \leq j$, can be computed in total time $O(n^2)$ as a preprocessing step. We show this now.

▶ **Lemma 15.** *All values $c(i, j)$ for $i \leq j$ can be computed in total time $O(n^2)$.*

Proof. We will compute the values separately for each value of $j - i$. In particular, for each $0 \leq \ell < n$ we will compute all values $c(i, i + \ell)$ for $1 \leq i \leq n - \ell$ in linear time. To do this, for a fixed ℓ, note the contribution of agent k to the c-values that it counts into is precisely $cost(s_k, \ell + 1)$. Therefore, the values $[c(i, i + \ell)]_{1 \leq i \leq n-\ell}$ that we want to compute are aggregate queries over a sliding window of length $\ell + 1$ over the sequence $[cost(s_k, \ell + 1)]_{1 \leq k \leq n}$. Depending on the utilitarian/egalitarian goal, the aggregate can be either summation or maximum, but in either case, all the $n - \ell$ aggregates can be computed in linear time using standard sliding window techniques. ◀

[7] However, size-monotonicity is crucial in showing that considering compact solutions is enough.

We are now ready to present our algorithm. We construct a weighted directed acyclic graph $\mathcal{G}$ corresponding to the problem instance, as follows. We are given a source node $s = (0, 0)$ and a target node $t = (n + 1, *)$. For the remaining vertices, we have one vertex for each pair (i, j) with $1 \leq i \leq n+1$ and $0 \leq j \leq \alpha$. Intuitively, vertex (i, j) has the meaning "agent $1 \leq i \leq n+1$ is the next one to consider[8] and so far we have excluded $0 \leq j \leq \alpha$ agents from the hike. The source s is connected with a directed edge towards $(1, 0)$ and such an edge has weight 0, while t is reachable from the node $(n + 1, j)$, for each $0 \leq j \leq \alpha$, via an edge of weight 0. For the remaining edges, we add the following two types:

- We add a weighted edge $(i, j) \xrightarrow{0} (i+1, j+1)$ for all $1 \leq i \leq n$ and $0 \leq j < \alpha$. Intuitively, these correspond to excluding agent i from the hike.
- We add a weighted edge $(i, j) \xrightarrow{c(i,k)} (k + 1, j)$ for all $1 \leq i \leq k \leq n$ and $0 \leq j \leq \alpha$,. Intuitively, these correspond to adding a new coalition $C = \{i, i + 1, \ldots, k\}$ to the hike, incurring a cost of $c(i, k)$.

The next lemma establishes how paths in $\mathcal{G}$ correspond to compact solutions to our problem and its statement immediately follows by the construction of $\mathcal{G}$.

▶ **Lemma 16.** *There is a bijection from compact solutions to s-t paths in $\mathcal{G}$. Moreover, the social cost of a compact solution is the cost of the associated path, defined as either the sum or the maximum of the costs of its constituent edges.*

As a result, computing an optimal compact solution amounts to finding a minimum cost s-t path in $\mathcal{G}$; this gives us a polynomial time algorithm for computing an optimal solution.

▶ **Theorem 17.** *A hike with minimum social cost can be computed in time $O(n^2(\alpha + 1))$.*

Proof. By Lemma 14 it is enough to compute the best hike among compact solutions. To do so, we construct the graph $\mathcal{G}$ corresponding to the problem instance. Subsequently, we compute an s-t path in $\mathcal{G}$ of minimum cost. This can be done in time linear in the size of the graph, as the graph is acyclic. Correctness is assured by Lemma 16. For the time bound, note that the number of vertices in the graph is $O(n(\alpha + 1))$ and the number of edges is $O(n^2(\alpha + 1))$. Moreover, edge costs can be computed in constant time after $O(n^2)$ total precomputation by Lemma 15. Overall, we get a time complexity of $O(n^2(\alpha + 1))$. ◀

4 Wonderful Partitions versus Minimum Egalitarian Partitions

So far we assumed each agent has an ideal group size, a peak, and there exists a cost function $cost(x, y)$ which expresses the cost that any agent having ideal group size x incurs if placed in a coalition of size y. More broadly, agents may express their disutility with any cost function, that is, for each agent i there is a mapping $cost_i : N \to \mathbb{R}$ simply expressing the cost agent i incurs when assigned to a coalition of a certain size.

In this section, we are interested in finding the minimum egalitarian cost achievable by any partition and we denote by Min-Eg the problem of computing this value. In Section 3, we have already shown that whenever $cost_i(s) = cost(s_i, s)$, Min-Eg can be computed in $O(n^2)$. In what follows, we exploit the connection between Wonderful-Partition and Min-Eg delineating the tractability of the latter with respect to the properties of the cost functions. First, we show a general connection between Wonderful-Partition and Min-Eg.

▶ **Proposition 18.** *Wonderful-Partition and Min-Eg are polynomial time equivalent.*

[8] Indeed, there is a "dummy" agent $n + 1$ signifying that there are no more agents to consider.

Proof. A Wonderful-Partition instance can be transformed into a Min-Eg instance by setting for each agent a cost function $cost_i$ where $cost_i(j) = 1$ if agent i does not approve coalition size j and $cost_i(j) = 0$, otherwise. Hence, the Wonderful-Partition instance is a yes instance if Min-Eg is 0 and it is a no instance, otherwise.

If we have a Min-Eg instance, there are at most n^2 distinct values $cost_i(j)$. Assume there are k distinct such values and let us sort them from the lowest to the highest, namely, $c_1 < \ldots < c_k$. For each value c_h, we can define a Wonderful-Partition instance where a coalition of size $j \in [n]$ is approved by agent i if and only if $cost_i(j) \leq c_h$. We can therefore determine if there exists a partition having egalitarian welfare of at most c_h by solving Wonderful-Partition on the just described instance. Clearly, if there exists a partition having an egalitarian cost of at most c then there exists a partition having an egalitarian cost of value at most c' for each $c \leq c'$. Conversely, if there is no partition having an egalitarian welfare of at most c then there is no partition having an egalitarian cost of at most c'', for each $0 \leq c'' \leq c$. Therefore, we can use binary search among the possible values $c_1, \ldots, c_k$ to find the minimum value c such that a partition having an egalitarian welfare of at most c exists. This solves Min-Eg. ◀

▶ **Corollary 19.** *If Wonderful-Partition can be decided in time $T(n)$, then Min-Eg can be solved in $O((n^2 + T(n)) \cdot \log_2 n)$.*

Given the polynomial time equivalence between these two problems, it follows that solving Min-Eg is in general computationally intractable because Wonderful-Partition is NP-hard as soon as the approval sets are not intervals [18].

Nevertheless, there exists a special class of costs that can be solved using our dynamic programming approach for Wonderful-Partition described in Section 2.1. Such a class is a generalization of what we discussed in Section 3: Namely, each agent has an ideal group size s_i, and the closer the coalition size is to s_i the lower the cost. We align to the Hedonic Games literature calling this property *naturally single-peakedness*.[9]

▶ **Definition 20.** *A cost function $cost : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is said to be* naturally single-peaked *if there exists an ideal group size, a* peak*, $p \in [n]$ such that $h < k \leq p$ or $h > k \geq p$ imply that $cost(k) < cost(h)$ holds.*

We observe that the reduction from Min-Eg to Wonderful-Partition described in Proposition 18 produces an interval instance in the case of naturally single-peaked cost functions. With this, we obtain the following theorem.

▶ **Theorem 21.** *Min-Eg for naturally single-peaked costs can be found in time $O(n^5 \log_2 n)$.*

Proof. We can use a similar idea as in the polynomial time reduction from Min-Eg to Wonderful-Partition, as described in Proposition 18. In Min-Eg we are looking for a partition that minimizes the cost of the agent having the highest disutility. Here, since we consider naturally single-peaked preferences, the disutility of an agent i is decreased the closer the size of their assigned group $|\pi(i)|$ is to their ideal group size s_i (the peak value of agent i). We can now fix, for a value c, some distances Δ, Δ' such that $cost_i(s) \leq c$ for each $s \in [s_i - \Delta, s_i + \Delta']$. Therefore, we define that any agent i approves their assigned group size if it is in the interval $[s_i - \Delta, s_i + \Delta']$; this defines the instance of Wonderful-Partition-Intervals to be solved for determining if there exists a partition of having an egalitarian cost of at most c.

Now, applying Corollary 19 and Theorem 3, the statement follows. ◀

[9] We say *naturally* as general single-peakedness may be defined w.r.t. any fixed ordering of coalition sizes. In our setting we consider cost functions that are single-peaked w.r.t. the natural ordering $1, \ldots, n$.

Notice that the computational complexity guaranteed by Theorem 17 is way more efficient than the one of Theorem 21. However, the former holds true for very specific naturally single-peaked costs, while the latter establishes the tractability of Min-Eg whenever agents have naturally single-peaked costs.

5 Conclusions

We resolved an open problem posed a decade ago by Gerhard Woeginger by giving a polynomial-time algorithm via establishing a connection to a version of rectangle stabbing, and investigated several interesting variants. We give a complete picture on the tractability of the Hiking Problem itself and show that maximizing the number of satisfied participants or deleting the minimal number such that the remaining participants admit a wonderful partition is polynomial time solvable. The tractability of both the original decision-, and the according optimization problems is crucially enabled by the existence of optimal solutions that exhibit simple and intuitive structural properties, fueling the algorithmic solutions based on Dynamic Programming. Last but not least, we employ our solution to efficiently compute a partition that maximizes the egalitarian welfare for anonymous naturally single-peaked Hedonic Games. We note that our approach also works for general interval instances, that is, for a given permutation σ of numbers $1, \dots, n$, intervals are defined over the numbers in order of the permutation, i.e., $\sigma(1), \dots, \sigma(n)$. This extends our results from naturally single-peaked to general single-peaked cost functions. The problem of minimizing utilitarian cost for general single-peaked cost functions remains open.

References

1 José Alcalde and Pablo Revilla. Researching with whom? Stability and manipulation. *Journal of Mathematical Economics*, 40(8):869–887, 2004. doi:10.1016/j.jmateco.2003.12.001.

2 Haris Aziz, Florian Brandl, Felix Brandt, Paul Harrenstein, Martin Olsen, and Dominik Peters. Fractional hedonic games. *ACM Transactions on Economics and Computation*, 7(2):1–29, 2019. doi:10.1145/3327970.

3 Haris Aziz, Felix Brandt, and Paul Harrenstein. Pareto optimality in coalition formation. *Games and Economic Behavior*, 82:562–581, 2013. doi:10.1016/j.geb.2013.08.006.

4 Haris Aziz, Felix Brandt, and Hans Georg Seedig. Computing desirable partitions in additively separable hedonic games. *Artificial Intellgence*, 195:316–334, 2013. doi:10.1016/J.ARTINT.2012.09.006.

5 Haris Aziz, Serge Gaspers, Joachim Gudmundsson, Julián Mestre, and Hanjo Täubig. Welfare maximization in fractional hedonic games. In *International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 461–467. AAAI Press, 2015. URL: http://ijcai.org/Abstract/15/071.

6 Haris Aziz, Paul Harrenstein, Jérôme Lang, and Michael J. Wooldridge. Boolean hedonic games. In *Principles of Knowledge Representation and Reasoning, KR 2016*, pages 166–175. AAAI Press, 2016. URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12869.

7 Haris Aziz and Raul Savani. Hedonic games. In *Handbook of Computational Social Choice*. Handbook of Computational Social Choice. Cambridge University Press, 2016.

8 Coralio Ballester. Np-completeness in hedonic games. *Games and Economic Behavior*, 49(1):1–30, 2004. doi:10.1016/j.geb.2003.10.003.

9 Suryapratim Banerjee, Hideo Konishi, and Tayfun Sönmez. Core in a simple coalition formation game. *Social Choice and Welfare*, 18:135–153, 2001. doi:10.1007/s003550000067.

10 Nadja Betzler, Arkadii Slinko, and Johannes Uhlmann. On the computation of fully proportional representation. *Journal of Artificial Intelligence Research*, 47:475–519, 2013. doi:10.1613/JAIR.3896.

11 Davide Bilò, Vittorio Bilò, Pascal Lenzner, and Louise Molitor. Tolerance is necessary for stability: Single-peaked swap schelling games. In *International Joint Conference on Artificial Intelligence, IJCAI 2022*, pages 81–87, 2022. doi:10.24963/IJCAI.2022/12.

12 Duncan Black. On the rationale of group decision-making. *Journal of political economy*, 56(1):23–34, 1948. doi:10.1086/256633.

13 Niclas Boehmer and Edith Elkind. Individual-based stability in hedonic diversity games. In *AAAI Conference on Artificial Intelligence, AAAI 2020*, pages 1822–1829, 2020. doi:10.1609/AAAI.V34I02.5549.

14 Anna Bogomolnaia and Matthew O. Jackson. The stability of hedonic coalition structures. *Games and Economic Behavior*, 38(2):201–230, 2002. doi:10.1006/GAME.2001.0877.

15 Felix Brandt, Markus Brill, Edith Hemaspaandra, and Lane A. Hemaspaandra. Bypassing combinatorial protections: Polynomial-time algorithms for single-peaked electorates. *Journal of Artificial Intelligence Research*, 53:439–496, 2015. doi:10.1613/JAIR.4647.

16 Robert Bredereck, Edith Elkind, and Ayumi Igarashi. Hedonic diversity games. In *International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2019*, pages 565–573, 2019. URL: http://dl.acm.org/citation.cfm?id=3331741.

17 Andrei Constantinescu, Pascal Lenzner, Rebecca Reiffenhäuser, Daniel Schmand, and Giovanna Varricchio. Solving woeginger's hiking problem: Wonderful partitions in anonymous hedonic games, 2023. arXiv:2311.02067.

18 Andreas Darmann, Edith Elkind, Sascha Kurz, Jérôme Lang, Joachim Schauer, and Gerhard Woeginger. Group activity selection problem with approval preferences. *International Journal of Game Theory*, 47(3):767–796, 2018. doi:10.1007/S00182-017-0596-4.

19 Jacques H Dreze and Joseph Greenberg. Hedonic coalitions: Optimality and stability. *Econometrica: Journal of the Econometric Society*, pages 987–1003, 1980. doi:10.2307/1912943.

20 Edith Elkind, Piotr Faliszewski, and Piotr Skowron. A characterization of the single-peaked single-crossing domain. *Social Choice and Welfare*, 54(1):167–181, 2020. doi:10.1007/S00355-019-01216-3.

21 Edith Elkind and Michael J. Wooldridge. Hedonic coalition nets. In *International Joint Conference on Autonomous Agents and Multiagent Systems AAMAS 2009*, pages 417–424, 2009. URL: https://dl.acm.org/citation.cfm?id=1558070.

22 Guy Even, Retsef Levi, Dror Rawitz, Baruch Schieber, Shimon Shahar, and Maxim Sviridenko. Algorithms for capacitated rectangle stabbing and lot sizing with joint set-up costs. *ACM Transactions on Algorithms*, 4(3):34:1–34:17, 2008. doi:10.1145/1367064.1367074.

23 Tobias Friedrich, Pascal Lenzner, Louise Molitor, and Lars Seifert. Single-peaked jump schelling games. In *International Symposium on Algorithmic Game Theory, SAGT 2023*, pages 111–126, 2023. doi:10.1007/978-3-031-43254-5_7.

24 Martin Hoefer, Sigal Oren, Roger Wattenhofer, and Giovanna Varricchio. Computational Social Dynamics (Dagstuhl Seminar 22452). *Dagstuhl Reports*, 12(11):28–44, 2023. doi:10.4230/DagRep.12.11.28.

25 Jan Karel Lenstra, Franz Rendl, Frits Spieksma, and Marc Uetz. In memoriam Gerhard Woeginger. *Journal of Scheduling*, 25(5):503–505, 2022. doi:10.1007/s10951-022-00748-4.

26 Martin Olsen. Nash stability in additively separable hedonic games and community structures. *Theory of Computing Systems*, 45:917–925, 2009. doi:10.1007/s00224-009-9176-8.

27 Dominik Peters. Complexity of hedonic games with dichotomous preferences. In *AAAI Conference on Artificial Intelligence, AAAI 2016*, pages 579–585. AAAI Press, 2016. doi:10.1609/AAAI.V30I1.10047.

28 Toby Walsh. Uncertainty in preference elicitation and aggregation. In *AAAI Conference on Artificial Intelligence, AAAI 2007*, pages 3–8, 2007. URL: http://www.aaai.org/Library/AAAI/2007/aaai07-001.php.

29 Gerhard Woeginger. Core stability in hedonic coalition formation. In *International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2013*, pages 33–50. Springer, 2013. doi:10.1007/978-3-642-35843-2_4.

30 Lan Yu, Hau Chan, and Edith Elkind. Multiwinner elections under preferences that are single-peaked on a tree. In *International Joint Conference on Artificial Intelligence, IJCAI 2013*, pages 425–431, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6777.

An Optimal Sparsification Lemma for Low-Crossing Matchings and Its Applications to Discrepancy and Approximations

Mónika Csikós
Université Paris Cité, IRIF, CNRS UMR 8243 and DI-ENS, Université PSL, France

Nabil H. Mustafa
Université Sorbonne Paris Nord, Laboratoire LIPN, CNRS 7030, France

Abstract

Matchings with low crossing numbers were originally introduced in the late 1980s in the seminal works of Welzl [35, 36] and Chazelle-Welzl [11]. They have since become fundamental structures in combinatorics, computational geometry, and algorithms.

In this paper, we study matchings with low crossing numbers and their relation to random samples. In particular, our main technical result states that, given a set system $(X, \mathcal{S})$ with dual VC-dimension d and a parameter $\alpha \in (0, 1]$, a random set of $\tilde{\Theta}\left(n^{1+\alpha}\right)$ edges of $\binom{X}{2}$ contains a linear-sized matching with crossing number $O\left(n^{1-\alpha/d}\right)$.

Furthermore, we show that this bound is optimal up to a logarithmic factor.

By incorporating the above sampling step to existing algorithms, we obtain improved running times, by a factor of $\tilde{\Theta}(n)$, for computing matchings with low crossing numbers. This immediately implies new bounds for a number of well-studied problems, such as combinatorial discrepancy, ε-approximations and their applications.

To the best of our knowledge, these are the first near-linear time algorithms for general, non-geometric set systems, for a) matchings with *sub-linear* crossing numbers, and b) discrepancy beating the standard deviation bound. As an immediate consequence we get fast algorithms for computing $o(1/\varepsilon^2)$-sized ε-approximations.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases low-crossing matchings, uniform sampling, discrepancy, approximations

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.49

Category Track A: Algorithms, Complexity and Games

Funding Supported by the French ANR PRC grant ADDS (ANR-19-CE48-0005), ANR AlgoriDAM (ANR-19-CE48-0016), and Fondation Sciences Mathématiques de Paris (FSMP).

1 Introduction

A perfect matching of a set X is a partition of X into $|X|/2$ disjoint pairs[1]. Given a set system $(X, \mathcal{S})$, we say that a set $S \in \mathcal{S}$ *crosses* a pair $\{x, y\} \subseteq X$ iff $|S \cap \{x, y\}| = 1$. Then for a perfect matching M of X, the *crossing number* of M with respect to $\mathcal{S}$ is defined to be the maximum number of edges of M crossed by any $S \in \mathcal{S}$.

Matchings with low crossing numbers were originally introduced by Welzl [35, 36] for the special case where X is a set of points in $\mathbb{R}^d$ and $\mathcal{S}$ is induced on X by half-spaces. His result was then improved and generalized by Chazelle and Welzl [11] to a broader class of set systems using the notion of the *dual shatter function* $\pi^*_{\mathcal{S}}$ of $(X, \mathcal{S})$, which is defined as follows:

[1] If $|X|$ is odd, then we partition X into $\lfloor |X|/2 \rfloor$ disjoint pairs plus a singleton set.

> For any $k \leq |\mathcal{S}|$, $\pi^*_{\mathcal{S}}(k)$ denotes the maximum number of equivalence classes on X defined by a k-element subfamily $\mathcal{R} \subseteq \mathcal{S}$, where $x, y \in X$ are equivalent w.r.t. $\mathcal{R}$ iff x belongs to the same sets of $\mathcal{R}$ as y. The number of such equivalence classes, for a given $\mathcal{R} \subseteq \mathcal{S}$, is essentially the number of non-empty cells, w.r.t. X, in the Venn diagram of the sets of $\mathcal{R}$.

The family of set systems with polynomially bounded dual-shatter function includes set systems with bounded dual VC-dimension and most of the commonly-studied geometric cases – e.g., the primal and dual set systems induced by half-spaces, balls, intersections/unions of bounded complexity geometric objects and more generally, algebraic varieties [26].

The following theorem on the existence of matchings with low crossing number is a celebrated and fundamental result in computational geometry.

▶ **Theorem A** ([11, 20]). *Let $(X, \mathcal{S})$ be a set system with $n = |X|$, and dual shatter function $\pi^*_{\mathcal{S}}(k) = O(k^d)$. Then there exists a perfect matching on X with crossing number $O\left(n^{1-1/d} + \ln |\mathcal{S}|\right)$.*

Theorem A has had numerous applications, including range searching, extremal results for hypergraphs with bounded VC dimension, low-discrepancy colorings in geometric hypergraphs, near-optimal sized epsilon-approximations, to name a few. We refer the reader to these books [10, 25, 29] for more information.

The power of Theorem A, as well as the difficulty in efficiently computing such matchings, come from the same source: a vanishingly small proportion of matchings are low-crossing matchings. Indeed, it is not difficult to see that a random matching will have crossing number $\Omega(n)$. For example, any set containing $n/2$ elements of X will cross, in expectation, a linear number of edges of such a matching.

This contrasts sharply with the case for the related structures of ε-nets and ε-approximations: a large-enough random sample is an ε-net/ε-approximation with high probability and thus a constant fraction of all subsets of X are ε-nets and ε-approximations. The use of randomness fails in our case since these ε-net/ε-approximation bounds rely on the fact that each set in the set system has *large* measure – at least ε-th measure – and the behavior of the random sample can be analyzed independently for each set, which are all known in advance.

Thus the ingenious proof of Theorem A takes a different route: it constructs a matching in $n/2$ iterations, where each iteration considers all remaining edges and picks one minimizing a certain function. This is computationally expensive, and consequently, all old and recent – with one exception – algorithms for building matchings with crossing numbers $o(n)$ in general set systems have $\Omega\left(n^2\right)$ running times, even when $|\mathcal{S}| = O(n)$ [11, 13, 18, 19].

The one exception is the algorithm proposed by Ducoffe *et al.* [16], designed specifically to get below the quadratic (in $|X| + |\mathcal{S}|$) running time barrier: they show that there is a universal constant $c > 2$ such that for any set system $(X, \mathcal{S})$ with VC-dimension D and dual VC-dimension d, one can compute a spanning path with crossing number $\tilde{O}\left(n^{1-1/(c \cdot D \cdot d)}\right)$ in time $\tilde{O}\left(|\mathcal{S}| + n^{2-1/(c \cdot D \cdot d)}\right)$.

2 Our Results

Motivated by the above considerations, we revisit the key question, of independent interest, of the utility of random sampling for constructing matchings with low crossing numbers. Our main technical result is that although a random perfect matching is very far from a low-crossing matching, a slightly larger random sample of edges contains a linear-sized matching that is close to a low-crossing matching. More precisely:

▶ **Lemma 1** (Main lemma; Proof in Section 4). *Let $(X, \mathcal{S})$, $n = |X|$, be a set system with dual shatter function $\pi^*(k) = O(k^d)$, and let $\alpha \in (0, 1]$, $\delta \in (0, 1)$ be two given parameters. Let E be a uniform random sample from $\binom{X}{2}$, where each edge is picked i.i.d. with probability*

$$p = \min\left\{\frac{2\ln n}{n^{1-\alpha}} + \frac{4\ln(2/\delta)}{n^{2-\alpha}},\ 1\right\}.$$

Then with probability at least $1 - \delta$, E contains a matching of size $n/4$ of crossing number

$$O\left(n^{1-\alpha/d} + \ln|\mathcal{S}|\right).$$

Moreover, we show that the above is near-optimal.

▶ **Lemma 2** (Optimality; Proof in Section 5). *For any $d \geq 2$, $c \geq 2$, $\alpha > 0$, and $n_0 \in \mathbb{N}$, there is a set system $(X, \mathcal{S})$ with $|X| = n \geq n_0$, and dual shatter function $\pi^*_{\mathcal{S}}(k) = O(k^d)$, such that the following holds:*

> *let E be a random edge-set obtained by selecting each edge in $\binom{X}{2}$ i.i.d. with probability $p(n) = o\left(n^{\alpha-1}\right)$. Then with probability at least $1/2$, every matching in E of size n/c has crossing number $\omega\left(n^{1-\alpha/d}\right)$ with respect to $\mathcal{S}$, where the constants in the asymptotic notation depend on d and c.*

▶ Remark. The same asymptotic results hold for low-crossing spanning paths and spanning trees.

We find Lemma 1 surprising for the following reason: the classical proof of Theorem A assigns exponentially-increasing weights to the sets of $\mathcal{S}$, which then dictate the choice of the edge picked at each iteration. Thus a different choice of the edge at iteration i could result in a changing of the weight distribution, which then influences the sequence of edges picked for all later iterations. At first glance, a random sample chosen once, and uniformly from $\binom{X}{2}$ cannot simply assure that it will contain many edges from all possible *exponentially* many paths possibly chosen by the algorithm.

In particular, if we fix an initial uniform sample of edges, and build the matching using this sample (by always choosing a light edge from the sample, as in previous algorithms), it introduces a bias (as the set of uncovered points depend on the initial sample of edges) and we cannot assume anymore that among the uncovered points, every edge is picked i.i.d. with a fixed probability. Indeed, with later iterations, the possible paths to be taken care of increase exponentially and the initial random sample has low probability of containing a good perfect matching.

Therefore, we take a different, more subtle, approach in the analysis, similar in spirit to the technique of quasi-uniform sampling of Varadarajan [34] and Chan et al.[9]:

1. instead of a perfect matching, we aim for a *linear-sized* partial matching with crossing number $O\left(n^{1-\alpha/d}\right)$. As we will prove, this requirement is weak-enough for a single sample to work, but strong-enough so that to compute a perfect matching, $O(\log n)$ adaptive uniform samples are sufficient.
2. instead of the classical algorithm, we propose a new randomized version where an edge from our initial, fixed, random sample of edges is picked in each iteration with a carefully chosen probability distribution that *does* depend on the changing weights in each iteration.

While Lemma 1 only guarantees the existence of a low-crossing *partial* matching, it can be used to speed up the fastest existing algorithms, by computing a linear sized matching and recursing on the uncovered points. This leads to the following result that improves the running times of previous-best algorithm of Csikós and Mustafa [13] by nearly a factor of $\Theta(n)$, at the cost of a higher, but still sub-linear crossing number.

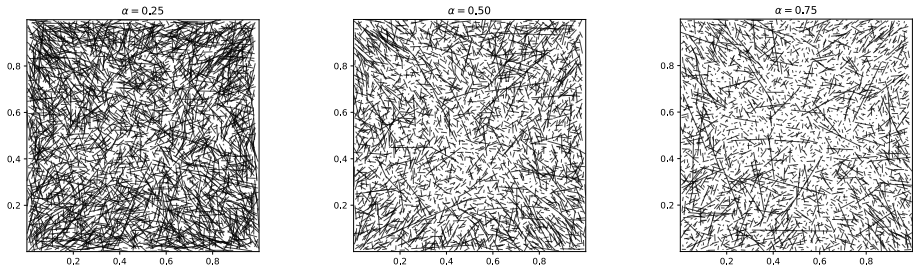
Table 1 $(X, \mathcal{S})$ has dual-shatter function $\pi^*(k) = O(k^d)$, $n = |X|$, $m = |\mathcal{S}|$.

Problem	Our Method		Previous-best	
	Guarantee	Time	Guarantee	Time
matching with low crossing number	$O\left(n^{1-\alpha/d} + \ln m \ln n\right)$	$\tilde{O}\left(n^{(1+\alpha)+\frac{2\alpha}{d}} + m \cdot n^{\frac{2\alpha}{d}}\right)$	$O\left(n^{1-1/d} + \ln m \ln n\right)$	$\tilde{O}\left(n^{2+\frac{2}{d}} + m \cdot n^{\frac{2}{d}}\right)$ [13]
			$O\left(n^{1-1/O(d \cdot 2^d)} + \ln m \ln n\right)$	$\tilde{O}\left(m + n^{2-1/O(d \cdot 2^d)}\right)$ [16]
combinatorial discrepancy	$O\left(\sqrt{n^{1-\alpha/d} \ln m + \ln^2 m \ln n}\right)$	$\tilde{O}\left(n^{(1+\alpha)+\frac{2\alpha}{d}} + m \cdot n^{\frac{2\alpha}{d}}\right)$	$O\left(\sqrt{n^{1-1/d} \ln m}\right)$	$\tilde{O}\left(n^{2+\frac{2}{d}} + m \cdot n^{\frac{2}{d}}\right)$ [13]

▶ **Corollary 3** (Proof in Appendix A.1). *Let $(X, \mathcal{S})$ be a set system with dual shatter function $\pi^*_{\mathcal{S}}(k) = O(k^d)$, $n = |X|$ and $\alpha \in (0, 1]$. Then there is a randomized algorithm which returns a perfect matching of expected crossing number $O\left(n^{1-\alpha/d} + \ln |\mathcal{S}| \ln n\right)$ in expected time $\tilde{O}\left(n^{1+\alpha+\frac{2\alpha}{d}} + |\mathcal{S}| \cdot n^{\frac{2\alpha}{d}}\right)$.*

▶ Remark. The algorithm of Corollary 3 can easily be modified to create a spanning path with the same crossing number guarantee.

Figure 1 The matchings output by the algorithm of Corollary 3 on the set system induced by half-spaces in $\mathbb{R}^2$ are shown below, for $\alpha = 0.25, 0.5$ and 0.75.



Algorithmic Consequences

Besides a new trade-off between quality and speed of computation, Corollary 3 has several algorithmic applications.

1. A better sub-quadratic time (in $|X| + |\mathcal{S}|$) algorithm. Setting a low value for the parameter $\alpha \in (0, 1]$ allows us to get a running time that is close to linear, at the expense of the crossing number. At the other end of the spectrum, we can get below the quadratic running time barrier without sacrificing too much in the crossing number. This improves the result of Ducoffe *et al.* [16] – who computed a spanning path with crossing number $\tilde{O}\left(n^{1-1/(c \cdot D \cdot d)}\right)$ in time $\tilde{O}\left(m + n^{2-1/(c \cdot D \cdot d)}\right)$, where $D = \text{VCdim}(X, \mathcal{S})$ – by letting $\alpha = d/(d+3)$. Besides improving the crossing number, the new algorithm also does not depend on the primal VC dimension.

▶ **Corollary 4.** *Let $(X, \mathcal{S})$ be a set system with dual shatter function $\pi^*_{\mathcal{S}}(k) = O(k^d)$, $n = |X|$. Then there is a randomized algorithm which returns a perfect matching of expected crossing number $O\left(n^{1-1/(d+3)} + \ln |\mathcal{S}| \ln n\right)$ in expected time $\tilde{O}\left(n^{2-\frac{1}{d+3}} + |\mathcal{S}| \cdot n^{\frac{2}{d+3}}\right)$.*

▶ Remark. In the work of Ducoffe *et al.* [16], spanning paths were used as a key algorithmic tool for computing the diameter of graphs. Their main result is a $\tilde{O}\left(k\, n^{2-1/O(d\cdot 2^d)}\right)$ time algorithm to decide whether the diameter of a graph G with distance VC-dimension d is at most k. Using Corollary 4 in their algorithmic framework ([16, Lemma 6]), we get an algorithm that decides whether G has diameter at most k, in time $O\left(k\, n^{2-1/(d+3)}\right)$.

2. Discrepancy. Using Corollary 3 in a standard iterative halving scheme [27] gives us faster approximation algorithms for combinatorial discrepancy, again improving the previous-best algorithms by nearly a factor of $\tilde{\Theta}(n)$, at the cost of a slightly higher discrepancy.

▶ **Corollary 5** (Proof in Appendix A.2). *Let $(X,\mathcal{S})$, $n=|X|$, $m=|\mathcal{S}|$, be a set system and d be a constant such that $\pi^*_{\mathcal{S}}(k)=O\left(k^d\right)$. For any $0<\alpha\le 1$, there is a randomized algorithm which constructs a coloring χ of X with expected discrepancy $O\left(\sqrt{n^{1-\alpha/d}\ln m+\ln^2 m\log n}\right)$, in expected time $\tilde{O}\left(n^{1+\alpha+\frac{2\alpha}{d}}+m\cdot n^{\frac{2\alpha}{d}}\right)$.*

This allows us to get the first near-linear time algorithm, for general non-geometric set systems, that beats the standard deviation discrepancy bound:

▶ **Corollary 6.** *Let $(X,\mathcal{S})$, $n=|X|$, $m=|\mathcal{S}|$, be a set system and d be a constant such that $\pi^*_{\mathcal{S}}(k)=O\left(k^d\right)$. For any $0<\varepsilon\le 1$, there is a randomized algorithm which constructs a coloring χ of X with expected discrepancy $O\left(\sqrt{n^{1-\frac{\varepsilon}{d+2}}\ln m+\ln^2 m\log n}\right)$, in expected time $\tilde{O}\left(n^{1+\varepsilon}+m^{1+\varepsilon}\right)$.*

3. ε-Approximations. The iterative application of Corollary 5 implies the following faster algorithm for computing ε-approximations [10, 29, 32, 24, 14].

▶ **Corollary 7.** *Let $\varepsilon\in(0,1)$, $(X,\mathcal{S})$ be a set system and c,d,D be constants such that $\pi^*_{\mathcal{S}}(k)\le ck^d$ and $\mathrm{VCdim}(X,\mathcal{S})\le D$. Then for any $\alpha\in(0,1)$, there is a randomized algorithm which returns an ε-approximation $A\subset X$ of size*

$$O\left(\left(\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon}\right)^{\frac{d}{d+\alpha}}\right)$$

in expected time

$$\tilde{O}\left(\left(\frac{D}{\varepsilon^2}\right)^{1+\alpha+\frac{2\alpha}{d}}+\left(\frac{D}{\varepsilon^2}\right)^{D+\frac{2\alpha}{d}}\right).$$

3 Previous Results

Matchings with low crossing numbers

The study of perfect matchings (along with spanning paths and spanning trees) with low crossing number was originally introduced for geometric range searching [35, 11]. Since then, they have found applications in various fields, for instance, discrepancy theory [27], learning theory [3], or algorithmic graph theory [16].

The proof of Theorem A is constructive, using the Multiplicative Weights Update (**MWU**) technique. Moreover it works for any (abstract) set system $(X,\mathcal{S})$ with polynomially bounded dual-shatter function. It builds a low-crossing matching iteratively, guided by a weight

Algorithm 1 MatchingMWU$\big((X,\mathcal{S})\big)$.

$\omega_1(S) \leftarrow 1$ for all $S \in \mathcal{S}$
$X_1 \leftarrow X$
for $i = 1, \ldots, n/2$ **do**
  $e_i \leftarrow$ the lightest edge in $\binom{X_i}{2}$ w.r.t. ω_i
  Obtain ω_{i+1} from ω_i by doubling the weights of each set crossing e_i
  $X_{i+1} \leftarrow X_i \setminus \text{endpoints}(e_i)$
return $\{e_1, \ldots, e_{n/2}\}$

function ω on $\mathcal{S}$, with initial weights set to 1. At each iteration, the algorithm adds the "lightest" edge to the matching – that is, the edge that is crossed by sets of minimum total weight. At the end of an iteration, it updates ω by doubling the weight of each set crossing the picked edge. See Algorithm 1. The algorithmic bottleneck is in finding such an edge: for an abstract set system without additional structure, this takes $O(n^2m)$ time for each of the $n/2$ iterations, giving a total running time of $O\left(n^3m\right)$, where $n = |X|$ and $m = |\mathcal{S}|$. Using MWU together with ideas from linear programming duality have led to the current-best running time for computing matchings with crossing number $O\left(n^{1-1/d}\right)$ in time $\tilde{O}\left(n^{2+2/d} + mn^{2/d}\right)$ [13].

A different approach was proposed by Har-Peled [19] (see also [17]). His result implies that if $(X,\mathcal{S})$ has a spanning tree with crossing number $\kappa = \Theta(n^\gamma)$ for some $\gamma \in [1/\log n, 1]$, then a spanning tree of crossing number $O(\kappa/\gamma)$ can be found by solving an LP on $\binom{n}{2}$ variables and $m + n$ constraints, see also [18]. Another result for general set systems having spanning trees with crossing number κ, is based on rounding fractional solutions of minimax integer programs with matroid constraints. This method gives a randomized algorithm that constructs a spanning tree with expected crossing number at most $\kappa + O(\sqrt{\kappa \log m})$ in time $\tilde{O}(mn^4 + n^8)$ [12].

For the geometric case, where X is a set of points in $\mathbb{R}^d$ and $\mathcal{S}$ is induced by half-spaces, Chan [8] gave a $O\left(n \log n\right)$ time algorithm to compute a matching with crossing number $O\left(n^{1-1/d}\right)$ using hierarchical cuttings. More generally, spatial partitioning by polynomials has also been extensively studied in the last decade [1, 2].

Discrepancy

Spencer [31] showed that for any set system (X, S), there exists a coloring of X with discrepancy $O\left(\sqrt{n \ln(m/n)}\right)$, which is tight and improves the general bound for $m = O\left(n\right)$. A series of algorithms for its construction started with the breakthrough work of Bansal [5], who gave the first polynomial-time randomized algorithm (using SDP rounding) to compute a coloring with discrepancy $O\left(\sqrt{n}\ln(m/n)\right)$, which matches the bound of Spencer for $m = O(n)$. Later Lovett and Meka [23] gave a combinatorial randomized algorithm for constructing colorings with discrepancy $O\left(\sqrt{n \ln(m/n)}\right)$ and improved the expected running time to $\tilde{O}\left(n^3 + m^3\right)$; see also [30] for a different proof. The algorithm of Bansal was de-randomized [7] (but still used a non-constructive method to prove the feasability of an underlying SDP), and later, Levy et al.[22] used the multiplicative weights update technique to give a deterministic $O\left(n^4m\right)$-time algorithm to compute a two-coloring with discrepancy $O\left(\sqrt{n \ln(m/n)}\right)$ for an arbitrary set system. See also [6] for a random-walk algorithm for Banaszczyk's discrepancy bound, with running time $O\left(n^{3.3728..} + nm^{2.3728..}\right)$. Most recently, Larsen proposed an

$O(n^2 m \ln(2 + m/n) + n^3)$ time algorithm with hereditary discrepancy guarantees [21]. The discrepancy guarantees of [21] can also be achieved with an $\tilde{O}(\mathrm{nnz}(A) + n^3)$ time algorithm [15], where A is the membership matrix of $(X, \mathcal{S})$. We note that many of these algorithms can be applied in more general settings (e.g. for real-valued matrices), however none of them provide a sub-quadratic time algorithm for the combinatorial discepancy problem in structured set systems.

4 Proof of the Main Lemma

In this section, we prove Lemma 1. Given a weight function $\omega\colon \mathcal{S} \to \mathbb{R}^+$, for $e \in \binom{X}{2}$, set

$$\mathrm{weight}(e, \omega) := \sum_{\substack{S \in \mathcal{S}: \\ e \text{ crosses } S}} \omega(S).$$

Then for any integer $k > 0$, by *"the k lightest edges of $\binom{X}{2}$ w.r.t. ω"*, we refer to the k edges with the smallest values of $\mathrm{weight}(\cdot, \omega)$ (ties broken arbitrarily).

We show the required property of the random sample E via the analysis of the randomized algorithm RelaxedMWU presented in Algorithm 2. In particular, Lemma 1 immediately follows from these two properties.

▶ **Lemma 8.** *Let $(X, \mathcal{S})$ be a set system with dual shatter function $\pi^*_{\mathcal{S}}(k) = O\left(k^d\right)$, $\alpha \in (0, 1]$ and $E \subseteq \binom{X}{2}$. For any halting iteration $T = t$, the set of edges returned by RelaxedMWU$\left((X, \mathcal{S}), \alpha, E\right)$ have crossing number $O\left(t^{1-\alpha/d} + \ln|\mathcal{S}|\right)$.*

▶ **Lemma 9.** *If $E \subseteq \binom{X}{2}$ is an i.i.d. sample where each edge is picked with probability*

$$p = \min\left\{\frac{2\ln n}{n^{1-\alpha}} + \frac{4\ln(2/\delta)}{n^{2-\alpha}},\ 1\right\},$$

then RelaxedMWU$((X, \mathcal{S}), \alpha, E)$ returns at least $n/4$ edges with probability at least $1 - \delta$.

▶ Remark. Lemma 8 does not use the fact that E is chosen randomly, or the randomness used in the algorithm; these are only used by Lemma 9.

Algorithm 2 RelaxedMWU$\left((X, \mathcal{S}), \alpha, E\right)$.

```
1  ω_1(S) ← 1 for all S ∈ S
2  X_1 ← X
3  for iteration i = 1, ..., n/2 do
4      E_i ← the |X_i|^{2−α} lightest edges in (X_i choose 2) w.r.t. ω_i
5      if E ∩ E_i = ∅ then
6          return {e_1, ..., e_{i−1}} and set T = i − 1
7      else
8          Pick an edge e_i from E ∩ E_i uniformly at random
9          Compute ω_{i+1} from ω_i by doubling the weight of each set crossing e_i
10         X_{i+1} ← X_i \ endpoints(e_i)
11 return {e_1, ..., e_{n/2}} and set T = n/2
```

▶ Remark. The precise definition of $\mathcal{E}_i$ (line 4 of Algorithm 2) should consider the $\lfloor |X_i|^{2-\alpha} \rfloor$ lightest edges. For the simplicity of presentation, we replace $\lfloor |X_i|^{2-\alpha} \rfloor$ with $|X_i|^{2-\alpha}$ throughout the proof.

We now prove the two key properties of RelaxedMWU separately.

4.1 Proof of Lemma 8

For any function $f : \mathcal{S} \to \mathbb{R}$, define $f(\mathcal{S}) := \sum_{S \in \mathcal{S}} f(S)$.

We will use the following key lemma; its proof will be presented later.

▶ **Lemma 10.** *Let $(X, \mathcal{S})$ be a set system with dual shatter function $\pi^*_{\mathcal{S}}(k) \le c_1 \cdot k^d$. Then given any $Y \subset X$, a weight function $w : \mathcal{S} \to \mathbb{Z}^+$, and an integer $\ell \in \left[|Y|, \binom{|Y|}{2}\right]$, there are at least ℓ distinct edges in $\binom{Y}{2}$ such that the total weight of the sets of $\mathcal{S}$ crossing each edge is at most*

$$(10c_1)^{1/d} \cdot \frac{w(\mathcal{S}) \cdot \ell^{1/d}}{|Y|^{2/d}}.$$

Let $\{e_1, \ldots, e_t\}$ be the output of RelaxedMWU and for each $i \in [1, t]$, set $\eta_i = \text{weight}(e_i, \omega_i)$.

At the start of iteration i, we have $|X_i| = n - 2i + 2$ and we pick one of the $|\mathcal{E}_i| = |X_i|^{2-\alpha}$ lightest edges of $\binom{X_i}{2}$ w.r.t. ω_i. Applying Lemma 10 with $Y = X_i$, $\omega = \omega_i$ and $\ell = |\mathcal{E}_i|$, gives an upper bound on the weight of each edge in $\mathcal{E}_i$ w.r.t. ω_i. In particular, as $e_i \in \mathcal{E}_i$, we have

$$\eta_i \le (10c_1)^{1/d} \cdot \frac{\omega_i(\mathcal{S}) \cdot |X_i|^{(2-\alpha)/d}}{|X_i|^{2/d}} = (10c_1)^{1/d} \cdot \frac{\omega_i(\mathcal{S})}{|X_i|^{\alpha/d}} = \frac{(10c_1)^{1/d}\omega_i(\mathcal{S})}{(n-2i+2)^{\alpha/d}}. \tag{1}$$

Let κ_t denote the maximum number of edges in $\{e_1, \ldots, e_t\}$ that are crossed by a set in $\mathcal{S}$. By the weight-update rule of the algorithm, we have

$$\omega_{t+1}(\mathcal{S}) \ge \max_{S \in \mathcal{S}} \omega_{t+1}(S) = 2^{\kappa_t},$$

and for any $j \in [1, t]$

$$\omega_{j+1}(\mathcal{S}) = \omega_j(\mathcal{S}) + \eta_j = \omega_j(\mathcal{S}) \left(1 + \frac{\eta_j}{\omega_j(\mathcal{S})}\right).$$

Applying the above equality for $j = t, \ldots, 1$ iteratively and using $\omega_1(\mathcal{S}) = |\mathcal{S}|$, we obtain

$$\omega_{t+1}(\mathcal{S}) = \omega_t(\mathcal{S}) \left(1 + \frac{\eta_t}{\omega_t(\mathcal{S})}\right) = \cdots = |\mathcal{S}| \cdot \prod_{j=1}^{t} \left(1 + \frac{\eta_j}{\omega_j(\mathcal{S})}\right) \le |\mathcal{S}| \cdot \exp\left(\sum_{j=1}^{t} \frac{\eta_j}{\omega_j(\mathcal{S})}\right).$$

Combining the upper and lower bounds for $\omega_{t+1}(\mathcal{S})$, we get

$$2^{\kappa_t} \le \omega_{t+1}(\mathcal{S}) \le |\mathcal{S}| \cdot \exp\left(\sum_{j=1}^{t} \frac{\eta_j}{\omega_j(\mathcal{S})}\right) \quad \Longrightarrow \quad \kappa_t \le \frac{1}{\ln 2}\left(\ln|\mathcal{S}| + \sum_{j=1}^{t} \frac{\eta_j}{\omega_j(\mathcal{S})}\right). \tag{2}$$

Using the upper-bound on η_j from Equation (1), we conclude that for any stopping time $t \in [1, n/2]$, the matching $\{e_1, \ldots, e_t\}$ returned by RelaxedMWU has crossing number at most

$$\begin{aligned}
\frac{1}{\ln 2}\left(\ln|\mathcal{S}| + \sum_{j=1}^{t} \frac{(10c_1)^{1/d}}{(n-2j+2)^{\alpha/d}}\right) &\le \frac{1}{\ln 2}\left(\ln|\mathcal{S}| + \sum_{j=1}^{t} \frac{(10c_1)^{1/d}}{(2t-2j+2)^{\alpha/d}}\right) \quad \left(\textit{since } t \le \frac{n}{2}\right) \\
&\le \frac{1}{\ln 2}\left(\ln|\mathcal{S}| + \sum_{j=1}^{t} \frac{(10c_1)^{1/d}}{(t-j+1)^{\alpha/d}}\right) \\
&= \frac{1}{\ln 2}\left(\ln|\mathcal{S}| + \sum_{j=1}^{t} \frac{(10c_1)^{1/d}}{j^{\alpha/d}}\right) \\
&= \frac{\ln|\mathcal{S}|}{\ln 2} + O\left(t^{1-\alpha/d}\right) = O\left(t^{1-\alpha/d} + \ln|\mathcal{S}|\right).
\end{aligned}$$

This concludes the proof of Lemma 8 assuming Lemma 10. We now return to the proof of Lemma 10, for which we need the following two statements.

▶ **Theorem 11** (Turán's Theorem [33]). *Let $G = (V, E)$ be a graph with no clique of size $r + 1$. Then*

$$|E| \leq \left(1 - \frac{1}{r}\right) \frac{n^2}{2}.$$

▶ **Lemma 12** (Packing Lemma [20, 28, 29]). *Let $(X, \mathcal{S})$ be a set system with shatter function $\pi_{\mathcal{S}}(k) \leq c_1 \cdot k^d$ and let $\delta \in (1, |X|)$ be a parameter. Furthermore, let $\mathcal{P} \subset \mathcal{S}$ be a δ-separated set; that is, $|S_1 \Delta S_2| \geq \delta$ for all $S_1, S_2 \in \mathcal{P}$ (where $S_1 \Delta S_2$ denotes the symmetric difference of S_1 and S_2). Then*

$$|\mathcal{P}| \leq 2c_1 \left(\frac{|X|}{\delta}\right)^d.$$

Proof of Lemma 10. Let $(\mathcal{S}_w, \mathcal{R}_Y)$ denote the set system dual to $(Y, \mathcal{S})$ with multiplicities given by $w(\cdot)$. That is, the base set $\mathcal{S}_w$ consists of sets of $\mathcal{S}$, where each $S \in \mathcal{S}$ has $w(S)$ copies in $\mathcal{S}_w$. And $\mathcal{R}_Y$ consists of $|Y|$ sets, one for each element of Y:

$$\mathcal{R}_Y = \{R_y : y \in Y\}, \quad \text{where} \quad R_y = \{S \in \mathcal{S}_w : y \in S\}.$$

Observe that

- for any $x, y \in Y$, the set $R_x \Delta R_y$ contains precisely the sets in $\mathcal{S}_w$ that cross the edge xy,
- $|\mathcal{S}_w| = w(\mathcal{S})$, and
- the shatter function of $(\mathcal{S}_w, \mathcal{R}_Y)$ is the dual shatter function of $(Y, \mathcal{S})$.

Consider a graph G on Y, where there is an edge between two elements $x, y \in Y$ if and only if xy is crossed by more than δ_ℓ sets in $\mathcal{S}_w$, where we set

$$\delta_\ell = \left(10c_1 \cdot \frac{w(\mathcal{S})^d \ell}{|Y|^2}\right)^{1/d}.$$

Now the **Packing Lemma** implies that any δ_ℓ-separated subset of sets in $\mathcal{R}_Y$ has cardinality at most

$$C_\ell = 2c_1 \left(\frac{w(\mathcal{S})}{\delta_\ell}\right)^d = 2c_1 \frac{w(\mathcal{S})^d}{10c_1 \cdot \frac{w(\mathcal{S})^d \ell}{|Y|^2}} = \frac{|Y|^2}{5\ell}.$$

This implies that G does not contain a clique on $C_\ell + 1$ vertices, and so by **Turán's Theorem**, the number of pairs that are *not* edges in G is at least

$$\binom{|Y|}{2} - \left(1 - \frac{1}{C_\ell}\right) \frac{|Y|^2}{2} = \frac{|Y|^2}{2C_\ell} - \frac{|Y|}{2} = \frac{5\ell}{2} - \frac{|Y|}{2} \geq \ell,$$

where we used that $|Y| \leq \ell$. Thus, we have shown that there are at least ℓ edges which cross sets of total weight at most δ_ℓ. This concludes the proof of Lemma 10 and thus the proof of Lemma 8. ◀

4.2 Proof of Lemma 9

Our goal is to prove that if $E \subseteq \binom{X}{2}$ is a random set of edges, where each edge from $\binom{X}{2}$ is picked i.i.d. with probability

$$p = \min\left\{\frac{2\ln n}{n^{1-\alpha}} + \frac{4\ln(2/\delta)}{n^{2-\alpha}},\ 1\right\},$$

then the random halting time T of $\textsc{RelaxedMWU}\big((X,\mathcal{S}),\alpha,E\big)$ satisfies

$$\mathbb{P}\left[\ T \le n/4\ \right] \le \delta.$$

If $p = 1$, then the statement is trivially true, therefore assume that $p < 1$.

Note that we have two different sources of randomness: we run the algorithm on an initial sample E of edges and at each iteration, if $E \cap \mathcal{E}_i \neq \emptyset$, we sample an edge $e_i \in E \cap \mathcal{E}_i$ uniformly at random. The notation $\mathbb{P}[A]$ denotes the probability of event A under both randomness sources.

We will upper bound the probabilities $\mathbb{P}\left[\ T = i\ \right]$ for each $i = 0, \ldots, n/4$. For the case $i = 0$, since E is an i.i.d. uniform random sample of $\binom{X}{2}$, we have

$$\mathbb{P}\left[\ T = 0\ \right] = \mathbb{P}\left[\ E \cap \mathcal{E}_1 = \emptyset\ \right] = (1-p)^{|\mathcal{E}_1|} = (1-p)^{n^{2-\alpha}}.$$

Now consider the case $i \ge 1$. Observe that the edge-set $\mathcal{E}_i$ depends on the edges chosen in earlier iterations $j < i$. To signify this, for any sequence of $i-1$ edges $(e^1, \ldots, e^{i-1})$, let $\mathcal{E}_i(e^1, \ldots, e^{i-1})$ denote the set of $(n - 2(i-1))^{2-\alpha}$ lightest edges *assuming* that $e^1, \ldots, e^{i-1}$ were added to the matching and the weights of the sets of $\mathcal{S}$ were adjusted multiplicatively accordingly.

We say that a sequence $(e^1, \ldots, e^i)$ is *feasible* if $e^1 \in \mathcal{E}_1$, $e^2 \in \mathcal{E}_2(e^1)$, $\ldots$, $e^i \in \mathcal{E}_i(e^1, \ldots, e^{i-1})$. We denote the set of all feasible sequences of length i by $\mathcal{C}^i$.

For a $\mathbf{c} \in \mathcal{C}^i$, we use the notation $\mathbf{c}^0 = \emptyset$, and $\mathbf{c}^j = (e^1, \ldots, e^j)$ for all $j \in [1, i]$; note that $\mathbf{c}^i = \mathbf{c}$. Let $\mathbf{e}_i = (e_1, \ldots, e_i)$ denote the sequence of *random variables* representing the edges actually chosen at each step by the algorithm up to iteration i, with $\mathbf{e}_0 = \emptyset$ and $\mathbf{e}_j = (e_1, \ldots, e_j)$ for all $j \in [1, i]$.

In the analysis, we will apply the law of total probability over the events that the algorithm picks the edges given by a certain feasible sequence $\mathbf{c} \in \mathcal{C}^i$, that is, $\mathbf{e}_i = \mathbf{c}^i$.

We break the analysis into three steps.

1. Unfolding the probability $\mathbb{P}\left[\ T = i\ \right]$

Given the above notation, we have

$$\begin{aligned}
&\mathbb{P}\left[\ T = i\ \right] = \mathbb{P}\left[\ E \cap \mathcal{E}_1 \neq \emptyset,\ \ldots,\ E \cap \mathcal{E}_i \neq \emptyset,\ E \cap \mathcal{E}_{i+1} = \emptyset\ \right] \\
&= \sum_{\mathbf{c} \in \mathcal{C}^i} \mathbb{P}\left[\ E \cap \mathcal{E}_1(\mathbf{c}^0) \neq \emptyset,\ \ldots,\ E \cap \mathcal{E}_i(\mathbf{c}^{i-1}) \neq \emptyset,\ E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset \ \middle|\ \mathbf{e}_i = \mathbf{c}^i\ \right] \cdot \mathbb{P}\left[\ \mathbf{e}_i = \mathbf{c}^i\ \right] \\
&\le \sum_{\mathbf{c} \in \mathcal{C}^i} \mathbb{P}\left[\ E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset \ \middle|\ \mathbf{e}_i = \mathbf{c}^i\ \right] \cdot \mathbb{P}\left[\ \mathbf{e}_i = \mathbf{c}^i\ \right].
\end{aligned}$$

Note that $\mathcal{E}_{i+1}(\mathbf{c}^i)$ is a fixed set once we are given $\mathbf{c}^i = (e^1, \ldots, e^i)$. Using Bayes' theorem, we can express the conditional probabilities on the R.H.S. of the above inequality as

$$\mathbb{P}\left[\ E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset \ \middle|\ \mathbf{e}_i = \mathbf{c}^i\ \right] = \frac{\mathbb{P}\left[\ \mathbf{e}_i = \mathbf{c}^i \ \middle|\ E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset\ \right] \cdot \mathbb{P}\left[\ E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset\ \right]}{\mathbb{P}\left[\ \mathbf{e}_i = \mathbf{c}^i\ \right]}.$$

Thus, we get

$$\mathbb{P}\left[\, T = i \,\right] \leq \sum_{\mathbf{c}\in\mathcal{C}^i} \mathbb{P}\left[\, \mathbf{e}_i = \mathbf{c}^i \mid E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset \,\right] \cdot \mathbb{P}\left[\, E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset \,\right].$$

Fixing $\mathbf{e}_i = \mathbf{c}^i$ completely determines the set of edges in $\mathcal{E}_{i+1}(\mathbf{c}^i)$, and so

$$= \sum_{\mathbf{c}\in\mathcal{C}^i} \mathbb{P}\left[\, \mathbf{e}_i = \mathbf{c}^i \mid E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset \,\right] \cdot (1-p)^{|\mathcal{E}_{i+1}(\mathbf{c}^i)|}.$$

We now proceed by bounding the probability $\mathbb{P}\left[\, \mathbf{e}_i = \mathbf{c}^i \mid E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset \,\right]$, iteration by iteration:

$$\begin{aligned}
&\mathbb{P}\left[\, \mathbf{e}_i = \mathbf{c}^i \mid E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset \,\right] \\
&= \mathbb{P}\left[\, e_i = e^i \mid E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset,\ \mathbf{e}_{i-1} = \mathbf{c}^{i-1} \,\right] \cdot \mathbb{P}\left[\, \mathbf{e}_{i-1} = \mathbf{c}^{i-1} \mid E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset \,\right] \\
&= \ldots = \prod_{j=1}^{i} \mathbb{P}\left[\, e_j = e^j \mid E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset, \mathbf{e}_{j-1} = \mathbf{c}^{j-1} \,\right],
\end{aligned}$$

recalling that $\mathbf{e}_0 = \mathbf{c}^0 = \emptyset$. We conclude that

$$\mathbb{P}\left[\, T = i \,\right] \leq \sum_{\mathbf{c}\in\mathcal{C}^i} (1-p)^{|\mathcal{E}_{i+1}(\mathbf{c}^i)|} \cdot \prod_{j=1}^{i} \mathbb{P}\left[\, e_j = e^j \mid E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset, \mathbf{e}_{j-1} = \mathbf{c}^{j-1} \,\right]. \tag{3}$$

2. Bounding the probabilities $\mathbb{P}[\, e_j = e^j \mid E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset,\ \mathbf{e}_{j-1} = \mathbf{c}^{j-1} \,],\ j \in [1, i]$

Note that the condition $\mathbf{e}_{j-1} = \mathbf{c}^{j-1}$ fixes the set $\mathcal{E}_j(\mathbf{c}^{j-1})$. As e_j was picked uniformly from $E \cap \mathcal{E}_j(\mathbf{c}^{j-1})$, we further condition on all possible choices of $E \cap \mathcal{E}_j(\mathbf{c}^{j-1})$, with the constraint that $E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset$:

$$\begin{aligned}
&\mathbb{P}\left[\, e_j = e^j \mid E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset,\ \mathbf{e}_{j-1} = \mathbf{c}^{j-1} \,\right] \\
&= \sum_{S' \subseteq \mathcal{E}_j(\mathbf{c}^{j-1})} \left(\mathbb{P}\left[\, e_j = e^j \;\middle|\; E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset,\ \mathbf{e}_{j-1} = \mathbf{c}^{j-1},\ E \cap \mathcal{E}_j(\mathbf{c}^{j-1}) = S' \,\right] \right) \\
&\qquad\qquad \cdot \left(\mathbb{P}\left[\, E \cap \mathcal{E}_j(\mathbf{c}^{j-1}) = S' \;\middle|\; E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset,\ \mathbf{e}_{j-1} = \mathbf{c}^{j-1} \,\right] \right)
\end{aligned}$$

Note that if S' is such that $e^j \notin S'$, then the first probability in the above product is 0. Similarly, if $S' \cap \mathcal{E}_{i+1}(\mathbf{c}^i) \neq \emptyset$, then the second probability is equal to 0. Continuing,

$$\begin{aligned}
&= \sum_{\substack{S' \subseteq \mathcal{E}_j(\mathbf{c}^{j-1}) \setminus \mathcal{E}_{i+1}(\mathbf{c}^i): \\ e^j \in S'}} \left(\mathbb{P}\left[\, e_j = e^j \;\middle|\; E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset,\ \mathbf{e}_{j-1} = \mathbf{c}^{j-1},\ E \cap \mathcal{E}_j(\mathbf{c}^{j-1}) = S' \,\right] \right) \\
&\qquad\qquad \cdot \left(\mathbb{P}\left[\, E \cap \mathcal{E}_j(\mathbf{c}^{j-1}) = S' \;\middle|\; E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset,\ \mathbf{e}_{j-1} = \mathbf{c}^{j-1} \,\right] \right) \\
&= \sum_{\substack{S' \subseteq \mathcal{E}_j(\mathbf{c}^{j-1}) \setminus \mathcal{E}_{i+1}(\mathbf{c}^i): \\ e^j \in S'}} \left(\frac{1}{|S'|} \right) \cdot \left(p^{|S'|} \cdot (1-p)^{|\mathcal{E}_j(\mathbf{c}^{j-1}) \setminus \mathcal{E}_{i+1}(\mathbf{c}^i)| - |S'|} \right).
\end{aligned}$$

Rearranging the sum by the sizes of the S''s containing e^j:

$$= \sum_{\ell=1}^{|\mathcal{E}_j(\mathbf{c}^{j-1})\setminus\mathcal{E}_{i+1}(\mathbf{c}^i)|} \binom{|\mathcal{E}_j(\mathbf{c}^{j-1}) \setminus \mathcal{E}_{i+1}(\mathbf{c}^i)| - 1}{\ell - 1} \cdot \frac{1}{\ell} \cdot p^\ell \cdot (1-p)^{|\mathcal{E}_j(\mathbf{c}^{j-1})\setminus\mathcal{E}_{i+1}(\mathbf{c}^i)|-\ell}$$

$$= \sum_{\ell=1}^{|\mathcal{E}_j(\mathbf{c}^{j-1})\setminus\mathcal{E}_{i+1}(\mathbf{c}^i)|} \frac{\ell}{|\mathcal{E}_j(\mathbf{c}^{j-1}) \setminus \mathcal{E}_{i+1}(\mathbf{c}^i)|} \binom{|\mathcal{E}_j(\mathbf{c}^{j-1}) \setminus \mathcal{E}_{i+1}(\mathbf{c}^i)|}{\ell} \cdot \frac{1}{\ell} \cdot p^\ell \cdot (1-p)^{|\mathcal{E}_j(\mathbf{c}^{j-1})\setminus\mathcal{E}_{i+1}(\mathbf{c}^i)|-\ell}$$

$$= \frac{1}{|\mathcal{E}_j(\mathbf{c}^{j-1}) \setminus \mathcal{E}_{i+1}(\mathbf{c}^i)|} \sum_{\ell=1}^{|\mathcal{E}_j(\mathbf{c}^{j-1})\setminus\mathcal{E}_{i+1}(\mathbf{c}^i)|} \binom{|\mathcal{E}_j(\mathbf{c}^{j-1}) \setminus \mathcal{E}_{i+1}(\mathbf{c}^i)|}{\ell} \cdot p^\ell \cdot (1-p)^{|\mathcal{E}_j(\mathbf{c}^{j-1})\setminus\mathcal{E}_{i+1}(\mathbf{c}^i)|-\ell}.$$

Using the Binomial theorem, and adjusting for the case $\ell = 0$,

$$= \frac{1}{|\mathcal{E}_j(\mathbf{c}^{j-1}) \setminus \mathcal{E}_{i+1}(\mathbf{c}^i)|} \left((p + (1-p))^{|\mathcal{E}_j(\mathbf{c}^{j-1})\setminus\mathcal{E}_{i+1}(\mathbf{c}^i)|} - (1-p)^{|\mathcal{E}_j(\mathbf{c}^{j-1})\setminus\mathcal{E}_{i+1}(\mathbf{c}^i)|} \right)$$

$$= \frac{1}{|\mathcal{E}_j(\mathbf{c}^{j-1}) \setminus \mathcal{E}_{i+1}(\mathbf{c}^i)|} \left(1 - (1-p)^{|\mathcal{E}_j(\mathbf{c}^{j-1})\setminus\mathcal{E}_{i+1}(\mathbf{c}^i)|} \right).$$

We can thus conclude that, setting $a = |\mathcal{E}_j(\mathbf{c}^{j-1}) \setminus \mathcal{E}_{i+1}(\mathbf{c}^i)|$,

$$\mathbb{P}\left[\ e_j = e^j \mid E \cap \mathcal{E}_{i+1}(\mathbf{c}^i) = \emptyset,\ \mathbf{e}_{j-1} = \mathbf{c}^{j-1}\ \right] = \frac{1}{a}\left(1 - (1-p)^a\right) \leq p, \tag{4}$$

where the last bound follows from Bernoulli's inequality $(1+x)^r \geq 1 + rx$ for $x \geq -1$ and $r \geq 1$, which holds in our case since $x = -p \geq -1$ and $r = |\mathcal{E}_j(\mathbf{c}^{j-1}) \setminus \mathcal{E}_{i+1}(\mathbf{c}^i)| \geq 1$.

3. Putting everything together

Let $k_i = \left|\mathcal{E}_i(\mathbf{c}^{i-1})\right| = |X_i|^{2-\alpha} = (n - 2(i-1))^{2-\alpha}$.

Continuing Equation (3) together with the bound from Equation (4), we get

$$\mathbb{P}\left[\ T = i\ \right] \leq (1-p)^{k_{i+1}} \cdot \sum_{\mathbf{c}\in\mathcal{C}^i} \prod_{j=1}^{i} p \ = \ (1-p)^{k_{i+1}} \cdot k_1 \cdot k_2 \cdots k_i \cdot p^i.$$

Summing the above over all iterations,

$$\mathbb{P}\left[\ T \leq i\ \right] \leq (1-p)^{k_1} + \sum_{\ell=1}^{i} (1-p)^{k_{\ell+1}} \cdot k_1 \cdot k_2 \cdots k_\ell \cdot p^\ell$$

Using that $k_1 \geq k_2 \geq \cdots \geq k_{i+1}$,

$$\mathbb{P}\left[\ T \leq i\ \right] \leq (1-p)^{k_{i+1}} + \sum_{\ell=1}^{i} (1-p)^{k_{i+1}} \cdot k_1^\ell \cdot p^\ell = (1-p)^{k_{i+1}} \cdot \sum_{\ell=0}^{i} (k_1 \cdot p)^\ell.$$

Thus, for $i = n/4$, using that $pk_1 = pn^{2-\alpha} \geq 2n \ln n \geq 2$, we obtain

$$\begin{aligned}
\mathbb{P}\left[\ T \leq n/4\ \right] &\leq (1-p)^{k_{n/4+1}} \sum_{\ell=0}^{n/4} (pk_1)^\ell \\
&= (1-p)^{k_{n/4+1}} \frac{(pk_1)^{n/4+1} - 1}{pk_1 - 1} < (1-p)^{k_{n/4+1}} \frac{(pk_1)^{n/4+1}}{pk_1 - 1} \\
&= (1-p)^{k_{n/4+1}} \cdot \frac{(pk_1)}{pk_1 - 1} \cdot (pk_1)^{n/4} \leq (1-p)^{k_{n/4+1}} \cdot 2 \cdot (pk_1)^{n/4} \quad \left(pk_1 > 2\right) \\
&\leq \exp(-pk_{n/4+1}) \cdot 2 \cdot k_1^{n/4}.
\end{aligned}$$

Substituting $k_1 = n^{2-\alpha}$, $k_{n/4+1} = (n/2)^{2-\alpha} \geq n^{2-\alpha}/4$ and $p = \frac{2\ln n}{n^{1-\alpha}} + \frac{4\ln(2/\delta)}{n^{2-\alpha}}$, we conclude

$$\mathbb{P}[\, T \leq n/4 \,] \leq 2\exp\left(-\frac{n\ln n}{2} - \ln\frac{2}{\delta}\right)\cdot\left(n^{2-\alpha}\right)^{n/4} = 2\cdot\frac{1}{n^{n/2}}\cdot\frac{\delta}{2}\cdot n^{n/2-\alpha n/4}$$
$$= n^{-\alpha n/4}\cdot\delta \leq \delta.$$

Therefore, with probability at least $1-\delta$, RelaxedMWU returns a matching of size at least $n/4$, which concludes the proof of Lemma 9. ◀

▶ Remark. In the last equation, to get failure probability at most δ, we crucially need the fact that at the final iteration i, we still have $k_i \geq n^{2-\alpha}/4$, which limits the range of i. This is the technical reason why we can only guarantee that E contains a good partial matching, but the analysis breaks for perfect matchings.

5 Proof of Optimality

In this section we present the proof of Lemma 2. Let X be the set of $n = \left\lfloor n_0^{1/d}\right\rfloor^d$ points defined as $\left[\,1, \left\lfloor n_0^{1/d}\right\rfloor\,\right] \times \cdots \times \left[\,1, \left\lfloor n_0^{1/d}\right\rfloor\,\right] \subset \mathbb{Z}^d$, and let $\mathcal{S}$ consist of the $d\cdot\left\lfloor n_0^{1/d}\right\rfloor$ subsets of X induced by half-spaces of the form

$$H_{i,j} = \left\{x \in \mathbb{R}^d \;:\; x_i \leq j + 1/2\right\}, \quad i = 1,\ldots,d,\ j = 1,\ldots\left\lfloor n_0^{1/d}\right\rfloor.$$

Observe that for any edge $\{x,y\} \in \binom{X}{2}$, the number of sets in $\mathcal{S}$ that crosses $\{x,y\}$ is precisely the ℓ_1-distance[2] of x and y. Using this observation, it is easy to see that for any $k \in \mathbb{N}^+$ and $x \in X$, the number of edges $\{x,y\}$ that are crossed by at most k sets from $\mathcal{S}$ is $O(k^d)$. Thus, there is an absolute constant c_0 (depending on d) such that the total number of edges in $\binom{X}{2}$ crossed by at most k sets from $\mathcal{S}$ is at most $c_0\cdot nk^d$. We refer to these edges as *k-good* and denote their set with $\mathcal{G}_k$.

Let $p(n) = o(n^{\alpha-1})$ and E be a uniform random sample of edges, where each edge of $\binom{X}{2}$ is picked with probability $p(n)$. Setting $k_{p,c}(n) = \left(\frac{1}{4c\cdot c_0 p(n)}\right)^{1/d}$, the expected number of $k_{p,c}(n)$-good edges in E is

$$\mathbb{E}\left[\,|E\cap\mathcal{G}_{k_{p,c}(n)}|\,\right] \leq c_0 n\left(k_{p,c}(n)\right)^d\cdot p(n) = \frac{n}{4c}.$$

Thus, by Markov's inequality, we have $|E\cap\mathcal{G}_{k_{p,c}(n)}| \leq \frac{n}{2c}$ with probability at least $1/2$. Assume that $|E\cap\mathcal{G}_{k_{p,c}(n)}| \leq \frac{n}{2c}$ holds and let $M \subset E$ be any subset of size $\frac{n}{c}$. Then M contains at least $\frac{n}{2c}$ edges which are not $k_{p,c}(n)$-good. Therefore, the number of crossings between the edges of M and the sets of $\mathcal{S}$ is at least

$$\frac{n}{2c}\cdot\left(\frac{1}{4c\cdot c_0\cdot p(n)}\right)^{1/d}.$$

Recall that $|\mathcal{S}| = d\cdot\left\lfloor n_0^{1/d}\right\rfloor \leq dn^{1/d}$ and so by the pigeonhole principle, we get that there is a set in $\mathcal{S}$ that crosses at least

[2] The ℓ_1-distance of x and y is defined as $\ell_1(x,y) = \sum_{i=1}^d |x_i - y_i|$, where x_i is the i-th coordinate of x.

$$\frac{\frac{n}{2c}\cdot\left(\frac{1}{4c\cdot c_0 p(n)}\right)^{1/d}}{|\mathcal{S}|} \geq \frac{\frac{n}{2c}\cdot\left(\frac{1}{4c\cdot c_0 p(n)}\right)^{1/d}}{dn^{1/d}} = \frac{1}{2c\cdot d\cdot(4c\cdot c_0)^{1/d}}\cdot n^{1-1/d}\underbrace{\left(\frac{1}{p(n)}\right)^{1/d}}_{\omega\left(n^{(1-\alpha)/d}\right)}$$

$$= \omega_{d,c}\cdot\left(n^{1-\alpha/d}\right)$$

edges of M. This concludes the proof of Lemma 2. ◀

References

1 P. K. Agarwal and J. Matoušek. On range searching with semialgebraic sets. *Discrete & Computational Geometry*, 11(4):393–418, 1994.

2 P. K. Agarwal, J. Matoušek, and M. Sharir. On range searching with semialgebraic sets. II. *SIAM Journal on Computing*, 42(6):2039–2062, 2013.

3 N. Alon, S. Moran, and A. Yehudayoff. Sign rank versus VC dimension. In *COLT*, 2016.

4 N. Alon and J. H. Spencer. *The probabilistic method*. John Wiley & Sons, 2016.

5 N. Bansal. Constructive algorithms for discrepancy minimization. In *Proceedings of Symposium on Foundations of Computer Science, FOCS*, pages 3–10. IEEE Computer Society, 2010.

6 N. Bansal, D. Dadush, S. Garg, and S. Lovett. The Gram-Schmidt walk: a cure for the Banaszczyk blues. In *Proceedings of the Symposium on Theory of Computing, STOC*, pages 587–597, 2018.

7 N. Bansal and J. H. Spencer. Deterministic discrepancy minimization. *Algorithmica*, 67(4):451–471, 2013.

8 T. M. Chan. Optimal partition trees. *Discrete Comput. Geom.*, 47(4):661–690, 2012.

9 T. M. Chan, E. Grant, J. Könemann, and M. Sharpe. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 1576–1585, 2012.

10 B. Chazelle. *The Discrepancy Method: Randomness and Complexity*. Cambridge University Press, New York, NY, USA, 2000.

11 B. Chazelle and E. Welzl. Quasi-optimal range searching in spaces of finite VC-dimension. *Discrete Comput. Geom.*, pages 467–489, 1989.

12 C. Chekuri, J. Vondrák, and R. Zenklusen. Dependent randomized rounding for matroid polytopes and applications. *arXiv*, abs/0909.4348, 2009.

13 M. Csikós and N. H. Mustafa. Escaping the Curse of Spatial Partitioning: Matchings with Low Crossing Numbers and Their Applications. In *Proceedings of Symposium on Computational Geometry (SoCG 2021)*, volume 189, pages 28:1–28:17, 2021.

14 M. Csikós and N. H. Mustafa. Optimal approximations made easy. *Inf. Process. Lett.*, 176:106250, 2022.

15 Y. Deng, Z. Song, and O. Weinstein. Discrepancy minimization in input-sparsity time. *arXiv*, abs/2210.12468, 2022. doi:10.48550/arXiv.2210.12468.

16 G. Ducoffe, M. Habib, and L. Viennot. Diameter computation on h-minor free graphs and graphs of bounded (distance) VC-dimension. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1905–1922, 2020.

17 S. P. Fekete, M. E. Lübbecke, and H. Meijer. Minimizing the stabbing number of matchings, trees, and triangulations. *Discrete & Computational Geometry*, 40(4):595–621, 2008.

18 P. Giannopoulos, M. Konzack, and W Mulzer. Low-crossing spanning trees: an alternative proof and experiments. In *Proceedings of EuroCG*, 2014.

19 S. Har-Peled. Approximating spanning trees with low crossing number. *arXiv*, abs/0907.1131, 2009. arXiv:0907.1131.

20 D. Haussler. Sphere packing numbers for subsets of the boolean n-cube with bounded Vapnik-Chervonenkis dimension. *Journal of Combinatorial Theory, Series A*, 69(2):217–232, 1995.

21 K. G. Larsen. Fast discrepancy minimization with hereditary guarantees. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 276–289. SIAM, 2023. doi:10.1137/1.9781611977554.CH11.

22 A. Levy, H. Ramadas, and T. Rothvoss. Deterministic discrepancy minimization via the multiplicative weight update method. In *Integer Programming and Combinatorial Optimization (IPCO)*, pages 380–391, 2017.

23 S. Lovett and R. Meka. Constructive discrepancy minimization by walking on the edges. *SIAM J. Comput.*, 44(5):1573–1582, 2015.

24 M. Matheny and J. M. Phillips. Practical low-dimensional halfspace range space sampling. In *Annual European Symposium on Algorithms (ESA)*, volume 112, pages 62:1–62:14, 2018.

25 J. Matoušek. *Geometric Discrepancy: An Illustrated Guide*. Springer Berlin Heidelberg, 1999.

26 J. Matoušek. *Lectures on discrete geometry*, volume 212. Springer Science & Business Media, 2013.

27 J. Matoušek, E. Welzl, and L. Wernisch. Discrepancy and approximations for bounded VC-dimension. *Combinatorica*, 13(4):455–466, 1993.

28 N. H. Mustafa. A simple proof of the shallow packing lemma. *Discret. Comput. Geom.*, 55(3):739–743, 2016.

29 N. H. Mustafa. *Sampling in Combinatorial and Geometric Set Systems*. American Mathematical Society (AMS), 2022.

30 T. Rothvoss. Constructive discrepancy minimization for convex sets. *SIAM J. Comput.*, 46(1):224–234, 2017.

31 J. H. Spencer. Six standard deviations suffice. *Transactions of the American Mathematical Society*, 289:679–706, 1985.

32 M. Talagrand. Sharper bounds for Gaussian and empirical processes. *Annals of Probability*, 22:28–76, 1994.

33 P. Turán. On an external problem in graph theory. *Mat. Fiz. Lapok*, 48:436–452, 1941.

34 K. R. Varadarajan. Weighted geometric set cover via quasi-uniform sampling. In *Proceedings of ACM Symposium on Theory of Computing (STOC 2010)*, pages 641–648, 2010.

35 E. Welzl. Partition trees for triangle counting and other range searching problems. In *Proceedings of Annual Symposium on Computational Geometry (SoCG 1988)*, pages 23–33, 1988.

36 E. Welzl. On spanning trees with low crossing numbers. In *Data Structures and Efficient Algorithms, Final Report on the DFG Special Joint Initiative*, pages 233–249, 1992.

A Appendix

A.1 Proof of Corollary 3

The algorithm achieving the guarantees of Corollary 3 is presented in MatchingPresampled. It is essentially the algorithm presented in [13] run on an initial random sample of edges with a small modification: to incorporate the pre-sampling step in the analysis, we need to recurse slightly more often (after $n/16$ steps instead of $n/4$).

We use the following key lemma for PartialMatching.

▶ **Lemma 13** ([13]). *Let $\tilde{E} \subset E$ denote the set of edges that have non-zero weight when* PartialMatching$\big((X,\mathcal{S}),E,\kappa\big)$ *terminates. Then*

$$\mathbb{E}\left[\max_{S\in\mathcal{S}}\sum_{i=1}^{n/16}\mathrm{I}\left(e_i,S\right)\right]\leq 2\cdot\mathbb{E}\left[\min_{e\in\tilde{E}}\sum_{i=1}^{n/16}\mathrm{I}\left(e,S_i\right)\right]+O\left(\kappa+\ln|E|+\ln|\mathcal{S}|\right). \tag{5}$$

Algorithm 3 MatchingPresampled$\big((X,\mathcal{S}),d,\alpha\big)$.

$M \leftarrow \emptyset$
while $|X| > 16$ **do**
  $n \leftarrow |X|$
  $E \leftarrow$ sample of $O(n^{1+\alpha}\ln n)$ edges from $\binom{X}{2}$
  $\{e_1,\ldots,e_{n/16}\} \leftarrow$ PartialMatching$\big((X,\mathcal{S}),E,(n/16)^{1-\alpha/d}\big)$
  $M \leftarrow M \cup \{e_1,\ldots,e_{n/16}\}$
  $X \leftarrow X \setminus \text{endpoints}(M)$
match the remaining elements of X randomly and add the edges to M
return M

Algorithm 4 PartialMatching$\big((X,\mathcal{S}),E,\kappa\big)$.

$\omega_1(e) \leftarrow 1, \quad \pi_1(S) \leftarrow 1 \qquad \forall e \in E,\ S \in \mathcal{S}$
$\mathbf{p} \leftarrow \min\{106\cdot|X|/\kappa^2\cdot\ln(|E|\cdot|X|/16),\,1\}$
$\mathbf{q} \leftarrow \min\{39\cdot|X|/\kappa^2\cdot\ln(|\mathcal{S}|\cdot|X|/16),\,1\}$
for $i = 1,\ldots,|X|/16$ **do**
  $\omega_i(E) \leftarrow \sum_{e\in E}\omega_i(e)$
  $\pi_i(\mathcal{S}) \leftarrow \sum_{S\in\mathcal{S}}\pi_i(S)$
  choose $e_i \sim \omega_i$; // $\mathbb{P}[e_i = e] = \frac{\omega_i(e)}{\omega_i(E)} \quad \forall e \in E$
  choose $S_i \sim \pi_i$; // $\mathbb{P}[S_i = S] = \frac{\pi_i(S)}{\pi_i(\mathcal{S})} \quad \forall S \in \mathcal{S}$
  $E_i \leftarrow$ sample from E with probability $\mathbf{p}$; // $\mathbb{P}[e \in E_i] = \mathbf{p} \quad \forall e \in E$
  $\mathcal{S}_i \leftarrow$ sample from $\mathcal{S}$ with probability $\mathbf{q}$; // $\mathbb{P}[S \in \mathcal{S}_i] = \mathbf{q} \quad \forall S \in \mathcal{S}$
  ; // $\mathrm{I}(e,S) = 1$ if e crosses S, $\mathrm{I}(e,S) = 0$ otherwise
  for $e \in E_i$ **do**
    $\omega_{i+1}(e) \leftarrow \omega_i(e)\big(1 - \frac{1}{2}\mathrm{I}(e,S_i)\big)$; // halve weight if S_i crosses e
  for $S \in \mathcal{S}_i$ **do**
    $\pi_{i+1}(S) \leftarrow \pi_i(S)\big(1 + \mathrm{I}(e_i,S)\big)$; // double weight if S crosses e_i
  set the weight in ω_{i+1} of e_i and of each edge adjacent to e_i to zero
return $\{e_1,\ldots,e_{|X|/16}\}$

In MatchingPresampled, the subroutine PartialMatching is called with the parameter $\kappa = (n/16)^{1-\alpha/d}$, thus we get the following bound on the expected crossing number of $\{e_1,\ldots,e_{n/16}\}$:

$$\mathbb{E}\left[\max_{S\in\mathcal{S}}\sum_{i=1}^{n/16}\mathrm{I}(e_i,S)\right] \le \frac{1}{2}\cdot\mathbb{E}\left[\min_{e\in\tilde{E}}\sum_{i=1}^{n/16}\mathrm{I}(e,S_i)\right] + O\left(n^{1-\alpha/d}\right). \tag{6}$$

The left-hand side of Equation (5) is precisely the expected crossing number of the edges returned by PartialMatching. It remains to bound the expectation on the right-hand side of Equation (6). By Lemma 1, with probability at least $1-\frac{1}{n}$, the initial sample E contains a matching M_0 of size $\lceil n/4\rceil$ with crossing number

$$c_0\cdot\left(n^{1-\alpha/d} + \ln|\mathcal{S}|\right)$$

for some fixed constant c_0. Assume that it happens, then clearly $M_0 \cap \tilde{E}$ also has crossing number at most $c_0\cdot\left(n^{1-\alpha/d} + \ln|\mathcal{S}|\right)$ with respect to $\mathcal{S}$. Moreover, since we only zeroed the weights of edges adjacent to $2\cdot n/16$ distinct vertices of X, there are at least $n/8$ edges of

M_0 with positive weight when PartialMatching terminates. That is, $\left|M_0 \cap \tilde{E}\right| \geq n/8$ and $n/8 > 0$ since $n > 16$ at each call of PartialMatching . By the pigeonhole principle, there is an edge in $M_0 \cap \tilde{E}$ which is crossed by at most

$$\frac{c_0 \cdot \left(n^{1-\alpha/d} + \ln|\mathcal{S}|\right) \cdot n/16}{n/8} = O\left(n^{1-\alpha/d} + \ln|\mathcal{S}|\right)$$

sets from $S_1, \ldots, S_{n/16}$. Therefore, we have

$$\mathbb{E}\left[\max_{S\in\mathcal{S}} \sum_{i=1}^{n/16} \mathrm{I}(e_i, S)\right] \leq \frac{1}{2} \cdot \mathbb{E}\left[\min_{e\in\tilde{E}} \sum_{i=1}^{n/16} \mathrm{I}(e, S_i)\right] + O\left(n^{1-\alpha/d}\right) = O\left(n^{1-\alpha/d} + \ln|\mathcal{S}|\right),$$

where the last bound holds with with probability at least $1 - \frac{1}{n}$. Since the crossing number of any matching of X is at most $n/2$, the expected crossing number of the matching returned by the subroutine PartialMatching$\left((X,\mathcal{S}), E, (n/16)^{1-\alpha/d}\right)$ is $O\left(n^{1-\alpha/d} + \ln|\mathcal{S}|\right)$.

The bottleneck algorithmic step in PartialMatching is to update the weights of edges and sets belonging to E_i and $\mathcal{S}_i$ at each iteration $i = 1, \ldots, n/16$.

For any i, we have $\mathbb{E}[\,|E_i| + |\mathcal{S}_i|\,] = \tilde{O}\left(n^{1+\alpha}\mathbf{p} + m\mathbf{q}\right)$, thus in expectation, the total running time is $O\left(n\left(n^{1+\alpha} \cdot \min\{n/\kappa^2 \ln n,\, 1\} + m \cdot \min\{n/\kappa^2 \ln m,\, 1\}\right)\right) = \tilde{O}\left(n^{1+\alpha+\frac{2\alpha}{d}} + mn^{\frac{2\alpha}{d}}\right)$.

The algorithm MatchingPresampled makes $\log n$ calls to PartialMatching with exponentially decreasing input sizes. It can easily be deduced that MatchingPresampled returns a matching with expected crossing number $O\left(n^{1-\alpha/d} + \ln|\mathcal{S}| \ln n\right)$ in expected time $\tilde{O}\left(n^{1+\alpha+\frac{2\alpha}{d}} + mn^{\frac{2\alpha}{d}}\right)$. This concludes the proof of Corollary 3. ◀

A.2 Proof of Corollary 5

Now we deduce how Corollary 3 implies Corollary 5. The randomized algorithm that achieves the guarantees of Corollary 5 is presented in Algorithm 5.

Algorithm 5 LowDiscColorPresampled$\left((X,\mathcal{S}), d, \alpha\right)$.

$n \leftarrow |X|$
$\{e_1, \ldots, e_{\lceil n/2\rceil}\} \leftarrow$ MatchingPresampled$\left((X,\mathcal{S}), d, \alpha\right)$
for $i = 1, \ldots, \lceil n/2\rceil$ **do**
  $\{x_i, y_i\} \leftarrow$ endpoints (e_i)
  $\chi(x_i) = \begin{cases} 1 & \text{with probability } 1/2 \\ -1 & \text{with probability } 1/2 \end{cases}$
  $\chi(y_i) = -\chi(x_i)$; // we skip this step if $y_i = x_i$
return χ

▶ **Lemma 14.** *Let $(X,\mathcal{S})$ be a set system, $n = |X|$, $m = |\mathcal{S}| \geq 34$, and let M be a perfect matching of X with crossing number κ with respect to $\mathcal{S}$ and for each edge $\{x, y\} \in M$ define*

$$\chi_M(x) = \begin{cases} 1 & \textit{with probability } 1/2 \\ -1 & \textit{with probability } 1/2 \end{cases}$$

and $\chi_M(y) = -\chi_M(x)$. Then the expected discrepancy of χ_M is at most $\sqrt{3\kappa \ln m}$.

▶ Remark. A "high probability version" of Lemma 14 is well-known [27, Lemma 2.5] and implies the above bound.

Corollary 3 and Lemma 14 immediately imply that the algorithm LowDiscColorPresampled constructs a coloring with expected discrepancy $O\left(\sqrt{n^{1-\alpha/d}\ln m + \ln^2 m \log n}\right)$, in time $\tilde{O}\left(n^{1+\alpha+2\alpha/d} + |\mathcal{S}| \cdot n^{2\alpha/d}\right)$. This concludes the proof of Corollary 5. ◀

Proof of Lemma 14. Let $S \in \mathcal{S}$ be a fixed range. We express the sum $\chi_M(S)$ of colors over elements of S as

$$\chi_M(S) = \sum_{\{x,y\}\in M; x,y\in S} (\chi_M(x) + \chi_M(y)) + \sum_{x\in \mathrm{cr}(S,M)} \chi_M(x) = \sum_{x\in \mathrm{cr}(S,M)} \chi_M(x) \ ,$$

where $\mathrm{cr}(S,M) = \{x \in S : \{x,y\} \in M, y \notin S\}$. Since $\mathrm{cr}(S,M) \le \kappa$ for any $S \in \mathcal{S}$, $\mathrm{disc}(S,\chi_M)$ is a sum of at most κ *independent* random variables. We use the following concentration bound from [4].

▷ Claim 15 (Theorem A.1.1 from [4]). Let $X_1, \ldots, X_k$ be independent $\{-1,1\}$-valued random variables with $\mathbb{P}[X_i = -1] = \mathbb{P}[X_i = 1] = 1/2$. Then for any $\alpha \ge 0$

$$\mathbb{P}\left[\ \left|\sum_{i=1}^{k} X_i\right| > \alpha\ \right] \le 2e^{-\alpha^2/2k}.$$

Applying Claim 15, we get that for any fixed $S \in \mathcal{S}$ and $\alpha > 0$,

$$\mathbb{P}\left[\ |\chi_M(S)| > \alpha\ \right] \le 2e^{-\alpha^2/2\kappa}.$$

By the union bound, we get

$$\mathbb{P}\left[\ \mathrm{disc}_{\mathcal{S}}(\chi_M) > \alpha\ \right] = \mathbb{P}\left[\ \max_{S\in\mathcal{S}} |\chi_M(S)| > \alpha\ \right] \le m \cdot 2e^{-\alpha^2/2\kappa}.$$

Finally, we bound the expected discrepancy by applying Fubini's theorem

$$\mathbb{E}\left[\ \mathrm{disc}_{\mathcal{S}}(\chi_M)\ \right] \stackrel{\mathrm{def}}{=} \int_0^\infty \mathbb{P}\left[\ \mathrm{disc}_{\mathcal{S}}(\chi_M) > \alpha\ \right] d\alpha \le \int_0^\infty \min\left\{2m \cdot e^{-\alpha^2/2\kappa},\ 1\right\} d\alpha$$

$$= \int_0^{\sqrt{2\kappa\ln(2m)}} 1 d\alpha + \int_{\sqrt{2\kappa\ln(2m)}}^\infty 2m \cdot e^{-\alpha^2/2\kappa} d\alpha = \sqrt{2\kappa\ln(2m)} + 2m\sqrt{2\kappa} \int_{\sqrt{\ln(2m)}}^\infty e^{-t^2}\ dt$$

$$= \sqrt{2\kappa\ln(2m)} + 2m\sqrt{2\kappa} \int_{\sqrt{\ln(2m)}}^\infty \frac{t}{t} \cdot e^{-t^2}\ dt \le \sqrt{2\kappa\ln(2m)} + 2m\sqrt{\frac{2\kappa}{\ln(2m)}} \int_{\sqrt{\ln(2m)}}^\infty te^{-t^2}\ dt$$

$$= \sqrt{2\kappa\ln(2m)} + 2m\sqrt{\frac{2\kappa}{\ln(2m)}} \left[\ -\frac{e^{-t^2}}{2}\ \right]_{\sqrt{\ln(2m)}}^\infty = \sqrt{2\kappa\ln(2m)} + \sqrt{\frac{\kappa}{2\ln(2m)}} \le \sqrt{3\kappa\ln m},$$

if $m \ge 34$. This concludes the proof of Lemma 14. ◀

Fully-Scalable MPC Algorithms for Clustering in High Dimension

Artur Czumaj
Department of Computer Science, University of Warwick, Coventry, UK

Guichen Gao
School of Computer Science, Peking University, Beijing, China

Shaofeng H.-C. Jiang
School of Computer Science, Peking University, Beijing, China

Robert Krauthgamer
Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel

Pavel Veselý
Computer Science Institute of Charles University, Prague, Czech Republic

Abstract

We design new parallel algorithms for clustering in high-dimensional Euclidean spaces. These algorithms run in the Massively Parallel Computation (MPC) model, and are *fully scalable*, meaning that the local memory in each machine may be n^σ for arbitrarily small fixed $\sigma > 0$. Importantly, the local memory may be substantially smaller than the number of clusters k, yet all our algorithms are fast, i.e., run in $O(1)$ rounds.

We first devise a fast MPC algorithm for $O(1)$-approximation of uniform Facility Location. This is the first fully-scalable MPC algorithm that achieves $O(1)$-approximation for any clustering problem in general geometric setting; previous algorithms only provide $\mathrm{poly}(\log n)$-approximation or apply to restricted inputs, like low dimension or small number of clusters k; e.g. [Bhaskara and Wijewardena, ICML'18; Cohen-Addad et al., NeurIPS'21; Cohen-Addad et al., ICML'22]. We then build on this Facility Location result and devise a fast MPC algorithm that achieves $O(1)$-bicriteria approximation for k-Median and for k-Means, namely, it computes $(1+\varepsilon)k$ clusters of cost within $O(1/\varepsilon^2)$-factor of the optimum for k clusters.

A primary technical tool that we introduce, and may be of independent interest, is a new MPC primitive for geometric aggregation, namely, computing for every data point a statistic of its approximate neighborhood, for statistics like range counting and nearest-neighbor search. Our implementation of this primitive works in high dimension, and is based on consistent hashing (aka sparse partition), a technique that was recently used for streaming algorithms [Czumaj et al., FOCS'22].

2012 ACM Subject Classification Theory of computation → Massively parallel algorithms; Theory of computation → Facility location and clustering; Theory of computation → Randomness, geometry and discrete structures

Keywords and phrases Massively parallel computing, high dimension, facility location, k-median, k-means

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.50

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2307.07848` [32]

Funding *Artur Czumaj*: Partially supported by the Centre for Discrete Mathematics and its Applications (DIMAP), by EPSRC award EP/V01305X/1, by a Weizmann-UK Making Connections Grant, and by an IBM Award.
Shaofeng H.-C. Jiang: Research partially supported by a national key R&D program of China No. 2021YFA1000900 and a startup fund from Peking University.

Robert Krauthgamer: Partially supported by the Israel Science Foundation grant #1336/23, by a Weizmann-UK Making Connections Grant, by the Israeli Council for Higher Education (CHE) via the Weizmann Data Science Research Center, and by a research grant from the Estate of Harry Schutzman.
Pavel Veselý: Partially supported by GA ČR project 22-22997S and by Center for Foundations of Modern Computer Science (Charles Univ. project UNCE 24/SCI/008).

1 Introduction

Clustering large data sets is a fundamental computational task that has been studied extensively due to its wide applicability in data science, including, e.g., unsupervised learning, classification, data mining. Two highly popular and extremely basic problems are k-Median and k-Means. In the geometric (Euclidean) setting, the k-Median problem asks, given as input an integer k and a set $P \subseteq \mathbb{R}^d$ of n data points, to compute a set $C \subset \mathbb{R}^d$ of k center points, so as to minimize the sum of distances from each point in P to its nearest center. The k-Means problem is similar except that instead of the sum of distances, one minimizes the sum of squares of distances. A closely related problem is (uniform) Facility Location, which can be viewed as a Lagrangian relaxation of k-Median, where the number of centers can vary but it adds a penalty to the objective, namely, each center (called open facility) incurs a given cost $\mathfrak{f} > 0$. Other variants include a similar relaxation of k-Means, or generalizing the squaring each distance to raising it to power $z \geq 1$ (see Section 2 for formal definitions), and there is a vast literature on these computational problems.

Clustering is often performed on massive datasets, and it is therefore important to study clustering in the framework of distributed and parallel computations. We consider fundamental clustering problems in the theoretical model of *Massively Parallel Computation (MPC)*, which captures key aspects of modern large-scale computation systems, such as MapReduce, Hadoop, Dryad, or Spark. The MPC model was introduced over a decade ago by Karloff, Suri, and Vassilvitskii [55], and over time has become the standard theoretical model to study data-intensive parallel algorithms (see, e.g., [11, 41, 47]). At a high level, the MPC model consists of many machines that communicate with each other synchronously in a constrained manner, in order to solve a desired task in a few rounds. In more detail, an MPC system has m *machines*, each with a *local memory* of s words, hence the system's *total memory* is the product $m \cdot s$. Computation takes place in synchronous rounds, where machines perform arbitrary computations on their local memory and then exchange messages with other machines. Every machine is constrained to send and receive at most s words in every round, and every message must be destined to a single machine (not a broadcast). At each round, every machine processes its incoming messages to generate its outgoing messages, usually without any computational restrictions on this processing (e.g., running time). At the end of the computation, the machines collectively output the solution. The efficiency of an algorithm is measured by the *number of rounds* and by the local-memory size s. The total space should be low as well, but this is typically of secondary importance.

The local-memory size s is a key parameter and should be sublinear in the input size N (for if $s \geq N$ then any sequential algorithm can be executed locally by a single machine in one round). We focus on the more challenging regime, called *fully-scalable MPC*, where the local-memory size is an arbitrarily small polynomial, i.e., $s = N^\sigma$ for any fixed $\sigma \in (0, 1)$. This regime is highly desirable because in practice, the local memory of the machines is limited by the hardware available at hand, and the parameter σ can be used to model this limitation. The total memory should obviously be large enough to fit the input, i.e., $m \cdot s \geq N$, and ideally not much larger.

Modern research on MPC algorithms aims to solve fundamental problems in as few rounds as possible, ideally in $O(1)$ rounds, even in the fully-scalable regime, and so far there have been some successes in designing fast (i.e., $O(1)$-rounds) algorithms for fixed $\sigma < 1$; see e.g., [4, 12, 41]. In contrast, many graph problems, including fundamental ones such as connectivity or even distinguishing between 1 and 2 cycles, seem to require super-constant number of rounds (at the same time, proving a super-constant lower bound would yield a breakthrough in circuit complexity [62]). One can also consider smaller local space s, i.e., $\sigma = o(1)$, and in fact many algorithms work as long as $s \geq \text{polylog}(N)$. In this regime, the best one can hope for is usually not $O(1)$ but rather $O(\log_s N)$ rounds, because even broadcasting a single number to all machines requires $\Omega(\log_s N)$ rounds; see [62] for further discussion.

Clustering of massive datasets has received significant attention recently, with numerous algorithms that span multiple computational models, such as streaming [18, 63] or distributed and parallel models [16, 35, 15, 8, 9, 60, 6, 21, 12, 34]. However, this advance is facing a barrier, first identified by Bhaskara and Wijewardena [13]: State-of-the-art methods for k-clustering (i.e., when the number k of clusters is specified in the input) typically fail to yield fully-scalable MPC algorithms, because when implemented in the MPC model, these methods usually require local-memory size $s \geq \Omega(k)$. For example, many streaming algorithms for k-clustering problems are based on a linear sketch and thus readily applicable to the MPC model; however, all known streaming algorithms require $\Omega(k)$ space, and in fact there are lower bounds to this effect [23] even in low dimension. Another example is coresets [43], an extremely effective method to decrease the size of the dataset, even in high dimension; see [29, 17, 26] for the latest in this long line of research. Coresets can often be merged and/or applied repeatedly via the so-called "merge-and-reduce" framework [43], and can thus be applied successfully in many different settings, including the parallel setting. However, this method suffers from the drawback (which so far seems inherent) that each coreset must be stored in its entirety in a single machine, and it is easy to see that a coreset for k-clustering must be of size at least k; see also [26].

These shortcomings have led to a surge of interest in MPC algorithms for clustering, with emphasis on fully-scalable and fast algorithms (i.e., taking $O(1)$ rounds even for large k) that work in high dimension d. The first such result for k-clustering was by Bhaskara and Wijewardena [13]; their $O(1)$-round fully-scalable MPC algorithm achieves polylogarithmic bicriteria approximation for k-Means, i.e., it outputs $k \cdot \text{polylog}(n)$ centers (or clusters) whose cost is within $\text{polylog}(n)$-factor of the optimum for k centers. Recent work by Cohen-Addad et al. [27] achieves for k-Median a true $\text{polylog}(n)$-approximation (i.e., without violating the bound k on number of centers); their algorithm actually computes a hierarchical clustering, that is, a single hierarchical structure of centers that induces an approximate solution for k-Median for every k. These were the best fully-scalable $O(1)$-round MPC algorithms known for k-Median and k-Means prior to our work, and it remained open to achieve better than $O(\log n)$-approximation, even as a bicriteria approximation.

Not surprisingly, previous work on MPC algorithms has focused also on achieving improved approximation for restricted inputs. In particular, $(1+\varepsilon)$-approximation is achieved in [28] for k-Means and k-Median on inputs that are *perturbation-resilient* [14], meaning that perturbing pairwise distances by (bounded) multiplicative factors does not change the optimal clusters; unfortunately, results for such special cases rarely generalize to all inputs. There are also known fully-scalable MPC algorithms for k-center [10, 30], but they are applicable only in low dimension d and thus less relevant to our focus.

Despite these many advances on fully-scalable MPC algorithms, we are not aware of $O(1)$-approximation *for any clustering problem* in high dimension.[1] From a technical perspective, the primary difficulty is to distribute the data points across machines, so that points arriving to the same machine are from the same cluster. This task seems to require knowing the clustering, which is actually the desired output, so we run into a chicken-and-egg problem! Fortunately, powerful algorithmic tools can spot points that are close together, even in high-dimensional geometry. Namely, locality sensitive hashing (LSH) is employed in [13] and tree embedding is used in [27], but as mentioned above, these previous results do not achieve $O(1)$-approximation. We rely instead on the framework of *consistent hashing*, which was first employed for streaming algorithms in [33], although it was originally proposed in [53]; see Section 1.2 for details.

1.1 Our Results

We devise fully-scalable $O(1)$-round MPC algorithms for a range of clustering problems in $\mathbb{R}^d$, most notably Facility Location, k-Median, and k-Means. We first devise an $O(1)$-approximation algorithm for Facility Location and then exploit the connection between the problems to solve k-clustering.

By convention, we express memory bounds in machine words, where each word can store a counter in the range $[\mathrm{poly}(n)]$, i.e., $O(\log n)$ bits, and/or a coordinate of a point from $\mathbb{R}^d$ with restricted precision (comparable to that of a point from P). For example, the input $P \subset \mathbb{R}^d, |P| = n$ fits in nd words. Throughout, the notation $O_\varepsilon(\cdot)$ hides factors that depend on ε. We stress that in the next theorem, the memory bounds are polynomial in the dimension d (and not exponential); this is crucial because the most important case is $d = O(\log n)$, as explained in Remark 1.2f below.

▶ **Theorem 1.1** (Simplified version)**.** *Let $\varepsilon, \sigma \in (0,1)$ be fixed. There is a randomized fully-scalable MPC algorithm that, given a multiset $P \subset \mathbb{R}^d$ of n points distributed across machines with local memory of size $s \geq n^\sigma \cdot \mathrm{poly}(d)$, computes in $O_\sigma(1)$ rounds an $O_\varepsilon(1)$-approximation for uniform Facility Location. The algorithm uses $O(n^{1+\varepsilon}) \cdot \mathrm{poly}(d)$ total space.*

▶ Remark 1.2. This simplified statement omits standard technical details:

a) We consider here fixed σ, but the algorithm works for any local-memory size $s \geq \mathrm{poly}(d \log n)$, and in that case the number of rounds becomes $O(\log_s n)$.

b) Similarly, ε can be part of the input, and the approximation factor is in fact $(1/\varepsilon)^{O(1)}$.

c) The algorithm succeeds with high probability $1 - 1/\mathrm{poly}(n)$.

d) The input P can be a multiset, in contrast to streaming algorithms for such problems [49, 33], where data points must be distinct.

e) The algorithm outputs a feasible solution, i.e., a set of facilities $F \subset \mathbb{R}^d$ and an assignment of the input points to facilities. In fact, in all our algorithms $F \subseteq P$. We focus throughout on computing F, since our methods can easily compute a near-optimal assignment given F.

f) We state here the dependence on d explicitly for sake of completeness, but our result works whenever $s \geq \mathrm{polylog}(dn)$, which is preferable when $d \gg \log n$. This is achieved by reducing to the case $d = O(\log n)$, using a standard dimension reduction, as discussed in Remark 2.4.

[1] Having said that, $(3+\varepsilon)$-approximation is known for (non-geometric) correlation clustering [12].

Our full result (in the full version) is more general in two respects. First, it addresses a generalization of Facility Location where distances are raised to power $z \geq 1$ (see Section 2 for definitions). Second, it assumes access to consistent hashing with parameters Γ and Λ, and here we plugged in a known construction [33] that achieves a tradeoff $\Gamma = O(1/\varepsilon)$ and $\Lambda = n^{\varepsilon}$ for any desired $\varepsilon \in (0,1)$; see Lemma 2.3 and Remark 2.4. The description of the hash function should be included in our MPC algorithm, but it takes only $\mathrm{poly}(d)$ bits, which is easily absorbed in our bounds.

Previously, no fully-scalable MPC algorithm was known for Facility Location, although one can apply the algorithm for k-Median from [27] to obtain an $O(\mathrm{polylog}(n))$-approximation in $O(\log_s n)$ rounds. Another previous result that one could use is a streaming algorithm that achieves $O(1)$-approximation [33]. Although many streaming algorithms, including this one, can be implemented in the MPC model, they approximate the optimal value without reporting a solution; see Section 1.3 for details. Thus, from a technical perspective, our chief contribution is to compute an approximately optimal solution, which is a set of facilities F, in the fully-scalable regime.

k-Clustering. Our second result presents MPC algorithms for k-Median and k-Means that achieve an $(O(\mu^{-2}), 1+\mu)$-bicriteria approximation, for any desired $\mu \in (0,1)$. As usual, (α,β)-*bicriteria approximation* means that for every input, the algorithm outputs at most βk centers whose cost is at most α-factor larger than the optimum cost for k centers. By letting $\mu > 0$ be arbitrarily small but fixed, our algorithm gets arbitrarily close (multiplicatively) to k centers, while still approximating the optimal cost within $O(1)$-factor; in both respects, this is far stronger than the previous bicriteria approximation for k-Means [13]. Our result is incomparable to the previous bound for k-Median, which is a true $O(\mathrm{polylog}(n))$-approximation [27]. Nevertheless, our bicriteria approximation breaks a fundamental technical barrier in their tree-embedding approach, which cannot go below $O(\log n)$ ratio, due to known distortion lower bounds, and moreover does not generalize to k-Means, because it fails to preserve the squared distance.

Our approach is to tackle k-Median and k-Means via Lagrangian relaxation and rely on our algorithm for Facility Location. This approach can inherently handle large k, because our core algorithms for Facility Location can even output n clusters. The Lagrangian technique was initiated by Jain and Vazirani [51], who achieved a true $O(1)$-approximation for k-Median by leveraging special properties of their algorithm for Facility Location (see Section 1.2). However, their primal-dual approach for Facility Location seems challenging to implement in fully-scalable MPC. We thus develop an alternative approach that is inherently more parallel, albeit achieves only bicriteria approximation for k-clustering.

▶ **Theorem 1.3** (Simplified version). *Let $\varepsilon, \sigma \in (0,1)$ be fixed. There is a randomized fully-scalable MPC algorithm that, given $\mu \in (0,1)$, $k \geq 1$, and a multiset $P \subset \mathbb{R}^d$ of n points distributed across machines of memory $s \geq n^{\sigma} \cdot \mathrm{poly}(d)$, computes in $O_\sigma(1)$ rounds an $(O_\varepsilon(\mu^{-2}), 1+\mu)$-bicriteria approximation for k-Median, or alternatively k-Means. The algorithm uses $O(n^{1+\varepsilon}) \cdot d^{O(1)}$ total space.*

This simplified statement omits the same standard details as in Theorem 1.1, and Remark 1.2 applies here as well. In addition, it is known for k-Median and k-Means (but not known for Facility Location) that any (true) finite approximation requires $\Omega(\log_s n)$ rounds [13].

1.2 Technical Overview

We overview the main components in our algorithms, and highlight technical ideas that may find more applications in the future. We focus in this overview on Facility Location and k-Median, noting that our algorithm for k-Means (and other powers $z \geq 1$) is essentially the same. We further assume that $s = n^\sigma \cdot \text{poly}(d)$ for a fixed $\sigma \in (0, 1)$, and we aim to achieve round complexity $O(\log_s n) = O_\sigma(1)$.

Facility Location. Several different algorithmic approaches have been used in the past to achieve $O(1)$-approximation for Facility Location, including LP-rounding [19, 56], primal-dual [51], greedy [50], and local search [5]. Some of these sequential algorithms were adapted to the PRAM model [16], i.e., to run in polylogarithmic parallel time (RNC algorithms). These algorithms can be implemented in the MPC model, but as some logarithmic factors in the time complexity seem inherent, these fall short of $O(1)$ rounds in the fully-scalable regime. Our starting point is the Mettu-Plaxton (MP) algorithm [61], which is a combinatorial algorithm inspired by [51], that has been previously used to achieve $O(1)$-approximation in a few related models, particularly streaming, congested clique, and sublinear-time computation [7, 38, 33]. At a high level, this MP algorithm has two steps, it first computes a "radius" $r_p > 0$ for every $p \in P$, and then uses these r_p values to determine which facilities to open. However, implementing these steps in MPC is technically challenging:

- Computing r_p (approximately) can be reduced to counting the number of points in P within a certain distance from p [7], which is non-trivial to compute in $O(1)$ rounds, because many geometric techniques are ineffective in high dimension, for instance quadtrees/tree embeddings incur large approximation error and grids/nets require large memory. We overcome this issue by devising a new MPC primitive for geometric aggregation (in high dimension), that can handle a wide range of statistics, including the counting mentioned above.
- The MP algorithm determines which facilities to open by scanning the points in P in order of non-decreasing r_p value and deciding greedily whether to open each point as a facility. We design a new algorithm that avoids any sequential decision-making and decides whether to open each facility locally and in parallel. Our new algorithm may thus be useful also in other models.

Let us discuss these two new ideas in more detail.

MPC Primitive for Geometric Aggregation in High Dimension. We propose a new MPC primitive for aggregation tasks in high-dimensional Euclidean spaces (see Theorem 3.1 for details). Given a radius $r > 0$, this primitive outputs, for every data point $p \in P$, a certain statistic of the ball $B_P(p, r) := P \cap \{y \in \mathbb{R}^d : \text{dist}(x, y) \leq r\}$. Our implementation can handle any statistic that is defined by a *composable* function f, which means that $f(\cup_i S_i)$ for disjoint sets $S_i \subset \mathbb{R}^d$ can be evaluated from the values $\{f(S_i)\}_i$. Composable functions include counting the number of points, or finding the smallest label (when data points are labeled). In fact, this primitive can even be used for approximate nearest-neighbor search in parallel for all points (see Section 3.1). This natural aggregation tool plays a central role in all our MPC algorithms, and we expect it to be useful for other MPC algorithms in high dimension.

Technically, exact computation of such statistics may be difficult, and our algorithm estimates the statistic by evaluating it (exactly) on a set $A_P(p, r)$ that approximates the ball $B_P(p, r)$ in the sense that it is sandwiched between $B_P(p, r)$ and $B_P(p, \beta r)$ for some error parameter $\beta \geq 1$. To implement this algorithm in MPC, a natural idea is to "collect" all

data points in the ball $B_P(p,r)$ at a single machine, however this set might be too large to fit in one or even few machines. A standard technique to resolve this issue in low dimension is to impose a grid of fine resolution, say εr, and move each data point to its nearest grid point, which provides a decent approximation (e.g., $\beta = 1 + \sqrt{d}\varepsilon$). However, a ball $B_P(p,r)$ might contain $\varepsilon^{-\Theta(d)}$ grid points, which for high dimension might still not fit in one machine.

Our approach is to use *consistent hashing* (see Definition 2.2), which was first introduced in [53] under the name sparse partition, and was recently employed in the streaming setting for Facility Location in high dimension [33]. Roughly speaking, consistent hashing is a partition of the space $\mathbb{R}^d$, such that each part has diameter bounded by βr, and every ball $B_P(p,r)$ intersects at most $n^{1/\beta}$ parts.[2] Our algorithm moves each data point $p \in P$ to a fixed representative point inside its own part, then computes the desired statistic on each part (namely, on the data points moved to the same representative), and finally aggregates, for each $p \in P$, the $n^{1/\beta}$ statistics of the parts that intersect $B_P(p,r)$. This algorithm can be implemented in $O(\log_s n)$ rounds on MPC, albeit with slightly bigger total space $n^{1+1/\beta}$. An interesting feature of this algorithm is that its core is deterministic and hence leads to new *deterministic* MPC algorithms, including for approximate nearest-neighbor search; see Section 3.1.

Computing A Solution for Facility Location. Our algorithm is based on the MP algorithm, where a key notion is the "radius" $r_p > 0$ defined for each data point $p \in P$. Formally, it takes the value r such that serving all points in the ball $B(p,r)$ by p, i.e., opening a facility at p and assigning points to p, incurs a cost of $|B_P(p,r)| \cdot r$. This value r always exists and is unique. It is known that a constant-factor approximation $\hat{r}_p$ of r_p satisfies that $|B_P(p,\hat{r}_p)| \approx 1/\hat{r}_p$, hence computing $|B_P(p,r)|$ for $O(\operatorname{poly}\log n)$ different values of r suffices to compute an $O(1)$-approximation of r_p [7, Lemma 1]. However, it is not easy to estimate $|B_P(p,r)|$ in MPC (simultaneously for all p and r), and the abovementioned geometric aggregation only estimates $|A_P(p,r)|$ for some $A_P(p,r)$ that is sandwiched between $B(p,r)$ and $B(p,\beta r)$. Nonetheless, we show this weaker estimate suffices for approximating r_p within $O(\beta)$ factor. Thus, our aggregation primitive yields an $O(1)$-approximation for all the r_p values in $O(1)$ rounds.

An $O(1)$-factor estimate of the optimal cost OPT can be computed from an $O(1)$-approximation of the r_p values for all $p \in P$, because $\sum_{p\in P} r_p = \Theta(\mathrm{OPT})$ [7, 38]. Moreover, the r_p values can be used to compute an $O(1)$-approximate *solution* for Facility Location. Specifically, the MP algorithm [61] scans the points in P in order of non-decreasing r_p value, and opens a facility at each point p if so far no facility was opened within distance $2r_p$ from p. The sequential nature of this algorithm makes it inadequate for MPC, and we therefore design a new algorithm that makes decisions in parallel. It has two separate rules to decide whether to open a facility at each point $p \in P$:

(P1) open a facility at p with probability $\Theta(r_p)$, independently of other points; and

(P2) open a facility at p if it has the smallest label among $B_P(p,r_p)$, where each point $q \in P$ is assigned independently a random label $h(q) \in [0,1]$.

Let us give some intuition for these rules. Rule (P1) is a straightforward way to use the r_p values so that the expected opening cost is $O(\sum_p r_p) = O(\mathrm{OPT})$, but it is not sufficient by itself because the connection cost might be too large. Indeed, if a cluster of points is very far from all other points, say, a cluster of t points all with the same $r_p = 1/t$, then with constant

[2] This tradeoff between βr and $n^{1/\beta}$ is just one specific choice of known parameters. Our theorem works with any possible parameters of consistent hashing, see Lemma 2.3 and Remark 2.4.

probability, (P1) does not open any facility inside this cluster, and the closest open facility is prohibitively far. However, (P2) guarantees that at least one facility is opened inside this cluster, at the point that has the smallest label. Rule (P2) is not sufficient by itself as well. Indeed, let $x_1 \in P$ be the minimizer from the viewpoint of p, i.e., have the smallest label in $B_P(p, r_p)$. Then it may happen that there is no facility at x_1 because $B_P(x_1, r_{x_1})$ has another point x_2 with an even smaller label. The same may happen also to x_2, and we may potentially get a long "assignment" sequence $(p = x_0, x_1, x_2, x_3, \ldots, x_t)$, where each x_{i+1} is the minimizer from the viewpoint of x_i and only the last point x_t is an open facility. In this case, the connection cost of p can be as large as $\sum_{i=0}^{t-1} r_{x_i}$ (i.e., matching the bound obtained by the triangle inequality), which might be unbounded relative to r_p. This might happen not only for one point p but actually for many points, and the total connection cost would be prohibitive.

We deal with this assignment issue using rule (P1), and in effect use both rules together. Given such an assignment sequence, we prove that for every $i \geq 0$, either (i) $B_P(x_i, r_{x_i})$ contains an open facility with constant probability, or (ii) $r_{x_{i+1}} \leq r_{x_i}/2$, i.e., the r_p value drops significantly in the next step. Property (i) means that the sequence stops at x_i with constant probability and thus, unless (ii) occurs, the expected length of the assignment sequence is $O(1)$. Property (ii) implies that the connection cost of the next step drops significantly as it is proportional to its r_p value, and hence, we just sum up a subsequence of geometrically decreasing r_p values. Combining the two properties, we obtain that the expected connection cost for every point p is $O(r_p)$. Hence, the expected total connection cost is $O(\sum_p r_p) = O(\mathrm{OPT})$.

The main technical challenge is to show property (i) when (ii) does not occur. Specifically, we need to show that, given a partial reassignment sequence $(p = x_0, \ldots, x_{i-1})$, the probability that the sequence does not terminate at x_i is bounded by a small constant. This event can be broken into two sub-events: (a) rule (P1) does not open any facility at the "new" points of $B_P(x_i, r_{x_i})$, where a point is considered new if it is not in $\cup_{j<i} B_P(x_j, r_{x_j})$; and (b) the smallest label appears at a new point (and thus rule (P2) does not open a facility at x_i). Let $t \in [0, 1]$ be the fraction of points that are new in $B_P(x_i, r_{x_i})$. Then the probability of (a) is roughly $(1 - r_{x_i})^{\Omega(t/r_{x_i})} \approx e^{-\Omega(t)}$, where this calculation crucially uses that (ii) does not occur, which means all new points have a similar r_p value as x_i, and that for every point x, the ball $B_P(x, r_x)$ roughly contains $\Omega(1/r_x)$ points. Since the two events are independent and since (b) happens with probability t by symmetry, we conclude that the probability we need to bound is at most $\exp(-O(t)) \cdot t \leq O(1)$. Here, one can observe that (P1) and (P2) are "complementing" each other to make the said probability small: when t is large, the probability $\exp(-O(t))$ of (a), which comes from rule (P1), is small; otherwise, the probability t of (b), which comes from rule (P2), is small.

The idea of opening facilities using random labels and analyzing the cost by constructing an assignment sequence was previously used in [3]. The context there is of a dynamically changing input, and this technique is used to limit changes in the solution over time, while our goal is to have a fast parallel implementation. Although the high level idea is similar, the setup is quite different, as their algorithm needs a solution to a linear-programming (LP) relaxation for Facility Location, while ours only needs the r_p values; and consequently also the analysis is different, as their analysis uses the LP constraints and bounds the cost relative to LP value, while our analysis crucially uses basic properties of the r_p values.

It remains to bound the opening cost which is the easy part. We show that the number of open facilities is $O(\sum_p r_p)$ in expectation. Indeed, for points selected by rule (P1) this is clear. For rule (P2), since $B_P(p, r_p)$ contains at least $1/r_p$ points, the probability that p has the smallest label in $B_P(p, r_p)$ is at most r_p.

When implementing our algorithm in the MPC model, the only non-trivial part (except for estimating the r_p values) is to check that p has the minimum label in the ball $B(p, r_p)$. Nevertheless, it is sufficient to look for the minimum label in a larger set that approximates the ball, such as a set sandwiched between this ball and a ball whose radius is larger by $O(1)$-factor. Therefore, the MPC primitive for geometric aggregation is sufficient to execute rule (P2).

Overall, these rules compute a set of open facilities. To compute also an assignment of each point $p \in P$ to an open facility, we use our approximate nearest-neighbor search algorithm, to find for each point $p \in P$ its $O(1)$-approximately closest facility (see Section 3.1). Notice that the assignment algorithm searches for the closest facility, but the analysis is still based on the assignment sequence as above, even though the connection cost of this sequence may substantially exceed the distance to the closest facility.

k-Clustering. Our algorithm for k-Median follows the Lagrangian-relaxation framework established by Jain and Vazirani [51] (and used implicitly earlier by Garg [37]). They managed to obtain a true $O(1)$-approximation for k-Median by leveraging a special property guaranteed by their algorithm for Facility Location, namely, its output solution has opening cost cost_O and connection cost cost_C that satisfy $\alpha \cdot \text{cost}_O + \text{cost}_C \leq \alpha \cdot \text{OPT}$ for a certain $\alpha = O(1)$. Unfortunately, this stronger property is obtained via a highly sequential primal-dual approach, and seems difficult to implement efficiently in MPC, particularly because it is too sensitive for the known toolkit for high dimension, like locality sensitive hashing (LSH) and our geometric aggregation via consistent hashing.

We therefore take another approach of relying on a *generic* γ-approximation algorithm for Facility Location, and using it in a black-box manner to obtain bicriteria approximation for k-Median. Our algorithm can output $(1+\mu)k$ centers whose cost is at most $O(1/\mu^2)$-factor larger than the optimum cost for k centers, for any desired $0 < \mu < 1$. This type of tradeoff, where the number of centers is arbitrarily close to k, was relatively less understood, as previous work has focused mostly on a smaller $O(1)$-factor in the cost, but using significantly more than k centers [57, 58, 1, 64, 46]. The result of [59] does give $(1+\mu)k$ centers, and is thus the closest to ours in terms of bicriteria bounds, however it relies explicitly on LP rounding, which seems difficult to implement in MPC. To the best of our knowledge, obtaining $(1+\mu)k$ centers for k-clustering by a black-box reduction to Facility Location was not known before. We believe this new reduction, and the smooth tradeoff it offers, may be of independent interest.

In more detail, the black-box reduction from k-Median to Facility Location goes as follows: Assume momentarily that we know (an approximation of) the optimal clustering cost OPT, and consider the clustering instance as an input for Facility Location with opening cost $\mathfrak{f} := \text{OPT}/k$. The optimal cost of this instance is at most $k \cdot \mathfrak{f} + \text{OPT} = 2\,\text{OPT}$ as the optimal k-Median solution is also a feasible solution for Facility Location with at most k facilities; note that the choice of $\mathfrak{f}$ balances the opening and connection costs in this solution. We then run (any) γ-approximation algorithm for Facility Location, and it will find a solution whose cost is at most $\gamma \cdot 2\,\text{OPT}$. The choice of $\mathfrak{f}$ implies that the number of open facilities in this solution is at most $\gamma \cdot 2\,\text{OPT}/\mathfrak{f} = 2\gamma \cdot k$. Hence, we obtain a $(2\gamma, 2\gamma)$-bicriteria approximation of k-Median. Finally, we remove the assumption of knowing OPT by running this procedure in parallel for a logarithmic number of guesses for OPT, and taking the cheapest solution that uses at most $O(\gamma \cdot k)$ centers. Using our MPC algorithm for Facility Location, this gives an efficient MPC algorithm for clustering with $(O(\gamma), O(\gamma))$-bicriteria approximation guarantees.

In this approach, the number of centers (in the solution) might exceed k by a large constant factor. We now use a different method to reduce it to be arbitrarily close to k. First, we observe that the $(O(\gamma), O(\gamma))$-approximate solution can be used to get a *weak coreset*, namely, a weighted set of at most $O(\gamma \cdot k)$ distinct points, such that any α-approximate set of centers for the coreset is also an $O(\alpha \cdot \gamma)$-approximate solution for the original instance. To obtain the coreset, we just move every point to its approximately nearest facility in the approximate solution, which is a standard step in the clustering literature (see e.g. [42]). Then, weight $w(p)$ of a point p is the number of original points rounded to p. We note that the problem becomes trivial if the coreset fits in one machine, and thus the interesting case is when k is very large.

Given this weak coreset, we find an approximate solution with at most, say, $2k$ centers and cost increased by another factor of $O(\gamma)$, using the following simple but sequential algorithm: Process the coreset points in order of non-increasing weights $w(\cdot)$, and open a center at point p if so far there is no center within distance $\mathrm{OPT}/(k \cdot w(p))$ from p; here we again need a guess for OPT. In a nutshell, the analysis of this algorithm is based on averaging arguments; intuitively, only few points in the weak coreset can have a relatively large connection cost in the optimal solution.

We then convert this sequential algorithm into a parallel one using similar ideas as for Facility Location. That is, for every point in parallel, we add it to the set of centers using two separate rules: (i) independently with probability $1/\gamma$; or (ii) if there is no point with larger weight within distance $\mathrm{OPT}/(k \cdot w(p))$. To implement (ii) in MPC, we need to ensure a consistent tie-breaking, for which a small random perturbation of the weights is sufficient. Finally, to efficiently find the point of maximum (perturbed) weight in the neighborhood of every point, we again employ our MPC primitive for geometric aggregation.

1.3 Related Work

Parallel and Distributed Algorithms. A more general metric setting of Facility Location has been studied earlier in the distributed Congest and CongestedClique models (see, e.g., [38, 44, 45]), and these results immediately transfer into MPC algorithms with $O(n)$ local memory and $O(n^2)$ total space. In particular, in combination with the recent result in [20], these results yield an $O(1)$-round MPC algorithm for metric Facility Location. In this general-metric setting of Facility Location, instances have size $O(n^2)$, which makes the problem significantly different from our geometric setting, e.g., instances in $\mathbb{R}^d$ are trivial if the local memory is $O(n)$. Furthermore, those results for general metrics rely on computing $O(1)$-ruling sets, and by the conditional lower bounds in [40, 31], this seems to require $\omega(1)$ rounds on a fully-scalable MPC. Our algorithms bypass the obstacle of ruling sets by leveraging the geometric structure in $\mathbb{R}^d$ and one can view our high-level contribution as proposing a setting avoiding that obstacle.

Clustering problems (e.g., Facility Location, k-Means, k-Median) have been also studied in the PRAM model of parallel computation [16, 15]. These algorithms can be implemented in the MPC model, but the logarithmic factors in the running time or the approximation ratio seem inherent, and they fall short of achieving $O(1)$ rounds in the fully-scalable regime.

Connections to Streaming. A closely related model is the streaming model, which mainly focuses on sequential processing of large datasets by a single machine with a limited (sublinear) memory. In general, if a streaming algorithm is storing only a linear sketch and uses space $O(s^{1-\varepsilon})$ for a fixed $\varepsilon > 0$, then it can be simulated on MPC in $O(\log_s N)$ rounds (recall

s is the local memory per machine and N is the input size); this was also observed and mentioned in, e.g., [24]. Thus, the various recent results on streaming algorithms in high dimension can be readily applied in MPC, and we briefly discuss the most relevant ones in the following. We note that streaming algorithms typically assume the input is discrete, i.e., $P \subseteq [\Delta]^d$ with $\Delta = \mathrm{poly}(n)$.

For both k-Median [18] and k-Means [63], it is possible to find $(1+\varepsilon)$-approximate solution with k centers using space $\mathrm{poly}(\varepsilon^{-1}kd\log\Delta)$. However, these algorithms are not directly applicable in our setting, since to simulate these algorithms it requires local memory size $s = \Omega(k)$ which is not fully scalable. For Facility Location, [49] gave $O(d\log\Delta)$-approximation (along with several other problems including minimum spanning tree and matching), using space $\mathrm{poly}(d\log\Delta)$. Later on, an $O(d/\log d)$-approximation for Facility Location was obtained using similar space $\mathrm{poly}(d\log\Delta)$, and alternatively $O(1/\varepsilon)$-approximation using space $O(n^\varepsilon)$ [33]. However, these results for Facility Location can only estimate the optimal cost, as storing the solution requires linear space (which is too costly since streaming algorithms aim to use sublinear space). Hence, simulating these results only leads to estimating the optimal cost in MPC, while our result for Facility Location can indeed find an approximate solution.

MPC Algorithms for MST. A similar issue of cost estimation versus finding approximate solutions is present also for the minimum spanning tree (MST) problem. The classical streaming algorithm of [49] was recently improved to an $O(\log n)$-approximation using the same space regime $\mathrm{poly}(d\log\Delta)$ [25] and to an $O(1/\varepsilon^2)$-approximation for the n^ε space regime [24]. However, these results are for estimating the optimal MST cost, and it is still open to find an $O(1)$-approximate solution for high-dimensional MST in $O(\log_s n)$ rounds of MPC. Indeed, the currently best MPC algorithm finds an $O(1)$-approximate MST in $\tilde{O}(\log\log n)$ rounds [52] (using local space polynomial in n), while in $O(\log_s n)$ rounds, one can only find an $O(1)$-approximation in a low dimension [4] (see also [22] for an exact MST algorithm for $d=2$) or a $\mathrm{poly}(\log n)$-approximation in a high dimension [2]. For a related problem of single-linkage clustering in a low dimension, there is a $(1+\varepsilon)$-approximation on MPC in $O(\log n)$ rounds [65].

2 Preliminaries

For integer n, let $[n] \coloneqq \{1,2,\ldots,n\}$. For a function $\varphi : X \to Y$, the image of a subset $S \subseteq X$ is defined $\varphi(S) \coloneqq \{\varphi(x) : x \in S\}$, and the preimage of $y \in Y$ is defined as $\varphi^{-1}(y) \coloneqq \{x \in X : \varphi(x) = y\}$. A Euclidean ball centered at $x \in \mathbb{R}^d$ with radius $r \geq 0$ is defined as $B(x,r) \coloneqq \{y \in \mathbb{R}^d : \mathrm{dist}(x,y) \leq r\}$, where $\mathrm{dist}(x,y) \coloneqq \|x-y\|_2$ refers throughout to Euclidean distance. For a set $P \subset \mathbb{R}^d$, we define $B_P(x,r) \coloneqq B(x,r) \cap P$, which is also a metric ball inside P. Let $\mathrm{diam}(S)$ denote the diameter of $S \subseteq \mathbb{R}^d$. For two sequences S, T, denote their concatenation by $S \circ T$. The aspect ratio of a point set $S \subset \mathbb{R}^d$ is the ratio between the maximum and minimum inter-point distance of S, i.e., $\frac{\max_{x,y\in S}\mathrm{dist}(x,y)}{\min_{x\neq y\in S}\mathrm{dist}(x,y)}$.

▶ **Fact 2.1** (Generalized triangle inequality). *Let (V,ρ) be a metric space, and let $z \geq 1$. Then*

$$\forall x, x', y \in V, \qquad \rho^z(x,y) \leq 2^{z-1}(\rho^z(x,x') + \rho^z(x',y)).$$

Power-z (Uniform) Facility Location. Given a set of data points $P \subset \mathbb{R}^d$, a (uniform) opening cost $\mathfrak{f} > 0$ and some $z \geq 1$, the objective function of Power-z (Uniform) Facility Location for a set of facilities $F \subset \mathbb{R}^d$ is defined as

$$\mathrm{fl}_z(P,F) \coloneqq |F| \cdot \mathfrak{f} + \sum_{p\in P} \mathrm{dist}^z(p,F),$$

where again $\mathrm{dist}(x,y) \coloneqq \|x-y\|_2$ and $\mathrm{dist}(x,S) \coloneqq \min_{y\in S}\mathrm{dist}(x,y)$. From now on, we omit "uniform" from the name of the problem, and simply use Power-z Facility Location. We denote the minimum value of a solution for Power-z Facility Location by $\mathrm{OPT}^{\mathrm{fl}}_z(P) \coloneqq \min_{F\subseteq\mathbb{R}^d} \mathrm{fl}_z(P,F)$; we omit P if it is clear from the context.

(k,z)-Clustering. Given a set of data points $P\subset\mathbb{R}^d$, an integer $k\geq 1$ and some $z\geq 1$, the objective function of (k,z)-Clustering for a center set $C\subset\mathbb{R}^d$ with $|C|\leq k$ is defined as

$$\mathrm{cl}_z(P,C) \coloneqq \sum_{p\in P}\mathrm{dist}^z(p,C).$$

Notice that the special cases $z=1$ and $z=2$ are called k-Median and k-Means, respectively. We denote the minimum value of a solution for (k,z)-Clustering by $\mathrm{OPT}^{\mathrm{cl}}_z(P) \coloneqq \min_{C\subseteq\mathbb{R}^d:|C|\leq k}\mathrm{cl}_z(P,C)$; we again omit P when it is clear from the context.

Consistent Hashing. As mentioned above, our MPC primitive for geometric aggregation relies on *consistent hashing*. We define it below, and then state the best known bounds for its parameters, which are near-optimal [36].

▶ **Definition 2.2** ([33, Definition 1.6]). *A mapping $\varphi:\mathbb{R}^d\to\mathbb{R}^d$ is called a* Γ-gap Λ-consistent hash *with diameter bound $\ell>0$, or simply (Γ,Λ)-hash,*[3] *if it satisfies:*

1. *Diameter: for every image $z\in\varphi(\mathbb{R}^d)$, we have $\mathrm{diam}(\varphi^{-1}(z))\leq\ell$; and*
2. *Consistency: for every $S\subseteq\mathbb{R}^d$ with $\mathrm{diam}(S)\leq\ell/\Gamma$, we have $|\varphi(S)|\leq\Lambda$.*

▶ **Lemma 2.3** ([33, Theorem 5.1]). *For every $\Gamma\in[8,2d]$, there exists a (deterministic) (Γ,Λ)-hash $\varphi:\mathbb{R}^d\to\mathbb{R}^d$ where $\Lambda=\exp(8d/\Gamma)\cdot O(d\log d)$. Furthermore, φ can be described using $O(d^2\log^2 d)$ bits and one can evaluate $\varphi(x)$ for any point $x\in\mathbb{R}^d$ in space $O(d^2\log^2 d)$.*

▶ Remark 2.4. Our main results also hold under the assumption that $s\geq\mathrm{polylog}(dn)$, which is preferable when $d\gg\log n$. It follows by the well-known idea of applying a JL transform π (named after Johnson and Lindenstrauss [54]) with target dimension $O(\log n)$ as a preprocessing of the input $P\subset\mathbb{R}^d$, i.e., running our algorithm on $\pi(P)$ instead of P. Then, with probability $1-1/\mathrm{poly}(n)$, all the guarantees in our results would suffer only an $O(1)$ factor. To implement this preprocessing in MPC using $\mathrm{polylog}(dn)$ words of local memory, we use a bounded-space version of the JL transform [48], that requires only $\mathrm{polylog}(dn)$ words to specify π (in comparison, a naive implementation of π requires $O(d\log n)$ words). One machine can randomly generate this specification of π and broadcast it, and then all machines can apply π locally in parallel. This requires additional $O(\mathrm{polylog}(dn))$ local memory and $O(\log_s n)$ rounds, and these additional costs are easily absorbed in our bounds. Hence, in all our results we can assume without loss of generality that P is replaced by $\pi(P)$. Furthermore, after this preprocessing, we can use $d=O(\log n)$ also in the consistent hashing bounds in Lemma 2.3 to obtain a tradeoff $\Gamma=O(1/\varepsilon)$ and $\Lambda=O(n^\varepsilon)$, for any desired fixed $\varepsilon\in(0,1)$, and also the hash can be described using $\mathrm{polylog}(n)$ bits.

3 MPC Primitive for Geometric Aggregation in High Dimension

In our MPC algorithms, we often face a scenario where we want to compute something for each input point $p\in P$. That computation is a relatively simple problem, like computing the number of points in the ball $B_P(p,r)$ for some global value $r>0$. A more general

[3] Note that this definition is scale invariant with respect to ℓ, i.e., a scaling of $\mathbb{R}^d$ will scale ℓ but not affect the parameters Γ and Λ. Thus, upper and lower bounds can restrict attention to the case $\ell=1$.

version is to allow different radii r (local for each p); another generalization is to compute some function f over the points in $B_P(p,r)$, like finding the point with smallest identifier or smallest distance to p. A naive approach is to collect (copies of) all the points in $B_P(p,r)$ to the same machine, say the one holding p, and then compute f there. This is very challenging and our solution is to approximate these balls by generating sets $A_P(p,r) \approx B_P(p,r)$, and evaluate f on these sets instead of on the balls. The approximation here just means that the set $A_P(p,r)$ is sandwiched between a ball of radius r and one of larger radius, see (1). Informally, we thus compute $f(A_P(p,r)) \approx f(B_P(p,r))$, but of course the approximation here need not be multiplicative.

Our MPC algorithm derives these sets $A_P(p,r)$ from consistent hashing (see Definition 2.2), and thus our description below requires access to such a hash function φ, and moreover the final guarantees depend on the parameters Γ and Λ of the consistent hashing. The theorem below is stated in general, i.e., for any (Γ,Λ)-hash, but we eventually employ known constructions with $\Gamma = O(1/\varepsilon)$ and $\Lambda = n^{\varepsilon}$, for any desired $\varepsilon > 0$ (see Remark 2.4 and Lemma 2.3 for details). Obviously, running this algorithm in MPC requires an implementation of consistent hashing, which might require additional memory; but this memory requirement is typically much smaller than $\sqrt{s}$, and thus the hash function can be easily stored in each machine.

Yet another challenge is that the entire set $A_P(p,r)$ might not fit in a single machine, and we thus impose on f another requirement. We say that a function f is *composable* if for every disjoint $S_1,\dots,S_t \subseteq \mathbb{R}^d$, one can evaluate $f(S_1 \cup \dots \cup S_t)$ from the values of $f(S_1),\dots,f(S_t)$.[4] In our context, f maps finite subsets of $\mathbb{R}^d$ to $\mathbb{R}$. For instance, $f(S) = |S|$ is clearly composable. For a few more interesting examples, suppose every $x \in \mathbb{R}^d$ is associated with a value $h(x) \in \mathbb{R}$. Now if $h(x)$ represents the weight of x, then $f(S) = \sum_{x\in S} h(x)$ is the total weight of S; and if $h(x)$ represents an identifier (perhaps chosen at random), then $f(S) = \min_{x\in S} h(x)$ is the smallest identifier in S.

▶ **Theorem 3.1** (Geometric Aggregation in MPC). *There is a deterministic fully-scalable MPC algorithm with the following guarantees. Suppose that*

- *the input is $r \geq 0$ and a multiset $P \subset \mathbb{R}^d$ of n points distributed across machines with local memory $s \geq \mathrm{poly}(d \log n)$; and*
- *the algorithm has access to a composable function f (mapping finite subsets of $\mathbb{R}^d$ to $\mathbb{R}$) and to a (Γ,Λ)-hash $\varphi : \mathbb{R}^d \to \mathbb{R}^d$.*

Then the algorithm uses $O(\log_s n)$ rounds and $O(\Lambda \cdot \mathrm{poly}(d)) \cdot \tilde{O}(n)$ total space, and computes for each $p \in P$ a value $f(A_P(p,r))$, where $A_P(p,r)$ is an arbitrary set that satisfies

$$B_P(p,r) \subseteq A_P(p,r) \subseteq B_P(p, 3\Gamma \cdot r). \tag{1}$$

(In fact, the set $A_P(p,r)$ is determined by φ.)

Proof. Our algorithm makes use of the following standard subroutines in MPC, and we note that they are deterministic. In the broadcast procedure, to send a message of length at most $\sqrt{s}$ from some machine $\mathcal{M}_0$ to every other ones, one can build an $\sqrt{s}$-ary broadcasting tree whose nodes are the machines (with $\mathcal{M}_0$ as the root), and send/replicate the message level-by-level through the tree (starting from the root). Observe that the height of the tree is $O(\log_s n)$ and hence the entire process runs in $O(\log_s n)$ rounds. The reversed procedure defined on the same broadcast tree, sometimes called converge-cast [39], can be used to

[4] We use general t here because of our intended application, but obviously it follows from the special case $t = 2$.

aggregate messages with size at most $\sqrt{s}$ distributed across machines to some root machine $\mathcal{M}_0$ (for instance, to aggregate the sum of vectors of length $\sqrt{s}$ distributed across machines), in $O(\log_s n)$ rounds. In particular, it can be used to evaluate the composable function f on a (distributed) set S, where each machine $\mathcal{M}$ evaluates $f(S_{\mathcal{M}})$ for its own part $S_{\mathcal{M}} \subseteq S$, and aggregate using converge-cast.

▶ **Lemma 3.2** (Sorting in MPC [41]). *There is a deterministic MPC algorithm that given a set X of N comparable items distributed across machines with local memory s, sorts X such that each $x \in X$ knows its rank and $\forall x < y \in X$, it holds that the machine that holds x has an ID no larger than that of y. The algorithm uses $O(\log_s N)$ rounds and total space of $O(N)$ words.*

We give an outline of our algorithm in Algorithm 1; the implementation details in MPC are discussed below. The algorithm starts with "partitioning" $\mathbb{R}^d$ into buckets with respect to the (Γ, Λ)-hash φ, and approximates each ball $B(p, r)$ by the union of buckets that this ball intersects. This distorts the radius by at most an $O(\Gamma)$-factor. Then, we evaluate the f value on each bucket, and the approximation to $f(B_P(p, r))$ is obtained by "aggregating" the f value for the intersecting buckets of $B(p, r)$, where the composability of f is crucially used.

Implementation Details. Here we discuss how each step of Algorithm 1 is implemented efficiently in MPC. In line 3, since after the sorting, for each $u \in \varphi(P)$ the points in P_u, i.e., the set of points p such that $u = \varphi(p)$, span a (partial) segment of machines with contiguous IDs, one can use a converge-cast in parallel in each segment to aggregate $f(P_u)$. In line 4, although the total space is sufficient to hold all tuples, we may not have enough space to store the $O(\Lambda)$ tuples for a point p in a single machine. Instead, we allocate for every point p a (partial) segment of machines whose total space is $O(\Lambda)$ (which can be figured out via sorting), replicate p's to them (via broadcast), and generate $\varphi(B(p, r))$ in parallel on those machines. Specifically, each machine in the segment is responsible for generating a part of $\varphi(B(p, r))$, and a part can be generated locally without further communication since every machine holds the same deterministic φ. Line 6 and line 7 can be implemented similarly by broadcast and converge-cast, respectively, in parallel on each segment of machines.

Algorithm 1 MPC algorithm for evaluating $f(A_P(p, r))$ for $p \in P$, for given $P \subset \mathbb{R}^d, r > 0$.

1: each machine imposes the same (Γ, Λ)-hash $\varphi : \mathbb{R}^d \to \mathbb{R}^d$ with diameter bound $\ell := 2\Gamma r$
▷ notice that φ is deterministic, hence no communication is required
2: sort P with respect to $\varphi(p)$ for $p \in P$ (using Lemma 3.2)
3: for $u \in \varphi(P)$, evaluate and store $f(P_u)$ where $P_u := \varphi^{-1}(u) \cap P$
4: for each $p \in P$ and $u \in \varphi(B(p, r))$, create and store a tuple (p, u)
▷ as $|\varphi(B(p, r))| = O(\Lambda)$ by Definition 2.2, the total space is enough to hold all tuples
5: sort the tuples with respect to u (using Lemma 3.2)
6: let $\mathcal{T}_u = \{(\cdot, u)\}$, append $f(P_u)$ to all tuples in $\mathcal{T}_u$
▷ $f(P_u)$ is already evaluated and stored, as in line 3
7: sort the tuples with respect to p, and evaluate $f(A_P(p, r))$ for each p, where

$$A_P(p, r) := \bigcup_{u \in \varphi(B(p,r))} P_u$$

Round Complexity and Total Space. The round complexity is dominated by the sorting, broadcast and converge-cast procedures, which all take $O(\log_s n)$ rounds to finish and are invoked $O(1)$ times in total. Therefore, the algorithm runs in $O(\log_s n)$ rounds. The total space is asymptotically dominated by $\text{poly}(d \log n)$ times the total number of tuples, which is $O(\Lambda) \cdot n$ by Definition 2.2.

Correctness. Observe that the algorithm is deterministic, and hence there is no failure probability. It remains to show that $A_P(p,r)$ satisfies that $B_P(p,r) \subseteq A_P(p,r) \subseteq B_P(p,3\Gamma r)$. Recall that $P_u = \varphi^{-1}(u) \cap P$ as defined in line 3, and that $A_P(p,r) = \bigcup_{u \in \varphi(B(p,r))} P_u$ as in line 7. Hence, we have

$$A_P(p,r) = P \cap \varphi^{-1}(\varphi(B(p,r))).$$

Therefore, the first inequality is straightforward as $B_P(p,r) \subseteq \varphi^{-1}(\varphi(B(p,r))) \cap P = A_P(p,r)$ for any mapping φ.

To prove the second inequality, fix some $p \in P$. For a point $q \in A_P(p,r)$, by definition there is a $u_q \in \varphi(B_P(p,r))$ such that $q \in \varphi^{-1}(u_q)$. Then by Definition 2.2, we have $\text{diam}(\varphi^{-1}(u_q)) \leq \ell = 2\Gamma r$ which implies that any point $x \in \varphi^{-1}(u_q)$ satisfies $\text{dist}(x,q) \leq 2\Gamma r$. Now, pick a point $x \in B_P(p,r)$ such that $\varphi(x) = u_q$; such a point exists as $u_q \in \varphi(B_P(p,r))$. Then by definition, $x \in B_P(p,r) \cap \varphi^{-1}(u_q)$ and $\text{dist}(p,x) \leq r$. Hence, by triangle inequality, we have that $\text{dist}(p,q) \leq \text{dist}(p,x) + \text{dist}(x,q) \leq r + 2\Gamma r \leq 3\Gamma r$, which implies that $A_P(p,r) \subseteq B_P(p,3\Gamma r)$. This finishes the proof. ◀

3.1 Application to Nearest Neighbor Search

Given a set $X \subseteq \mathbb{R}^d$ of *terminals* and a set $P \subseteq \mathbb{R}^d$ of data points, the ρ-approximate nearest neighbor search problem asks to find for every $p \in P$ a terminal $x \in X$, such that $\text{dist}(p,x) \leq \rho \cdot \text{dist}(p,X)$. This process is useful in clustering and facility location problems, since one can find an assignment of every data point to its approximately nearest center/facility. We show how to solve this problem using Theorem 3.1, provided the knowledge of the *aspect ratio* Δ of $X \cup P$.

Pick an arbitrary point $x \in X \cup P$, compute in $O(\log_s n)$ rounds the maximum distance $M := \max_{y \in X \cup P} \text{dist}(x,y)$ from x to every other point (via broadcast and converge-cast). Since M is a 2-approximation to $\text{diam}(X \cup P)$, we conclude that for every $x \neq y \in X \cup P$, $M/\Delta \leq \text{dist}(x,y) \leq 2M$. Rescale the instance by dividing M/Δ, then the distances are between 1 and $O(\Delta)$. Then, let $Z := \{2^i : 1 \leq 2^i \leq O(\Delta)\}$. We apply Theorem 3.1 in parallel for $r \in Z$ and f such that $f(Y)$ for $Y \subseteq X$ finds the terminal with the smallest ID in Y (where the ID of a point can be defined arbitrarily as long as it is consistent), and f returns $\perp$ if $Y = \emptyset$. This f is clearly composable. After we obtain the result of Theorem 3.1, i.e., $f(A_X(p,r))$ for $p \in P$ and $r \in Z$, we find in parallel for each $p \in P$ the smallest $r \in Z$ such that $f(A_X(p,r)) \neq \perp$. This way, we explicitly get for each point p an approximately nearest facility in X. This algorithm has approximation factor $\rho = O(\Gamma)$, using total space by an $O(\log \Delta)$-factor larger than that of Theorem 3.1, while the round complexity remains $O(\log_s n)$.

We remark that techniques based on locality sensitive hashing (LSH) can also be applied in MPC to solve the approximate nearest neighbor problem [13, 28]. LSH in fact achieves a slightly better tradeoff, namely, an $O(c)$-approximation using total space $n^{1/c^2} \cdot \tilde{O}(n)$, while our approach requires total space $n^{1/c} \cdot \tilde{O}(n)$, by plugging in Lemma 2.3 and assuming the dimension reduction in Remark 2.4 is performed. Alternatively, if $d \leq O(\log n)$, one can

obtain the same $n^{1/c} \cdot \tilde{O}(n)$ bound without applying the randomized dimension reduction in Remark 2.4, which leads to a deterministic algorithm, whereas approaches based on LSH are inherently randomized.

References

1 Ankit Aggarwal, Amit Deshpande, and Ravi Kannan. Adaptive sampling for k-means clustering. In *Proceedings of the 12th International Workshop on Approximation Algorithms for Combinatorial Optimization, and of the 13th International Workshop on Randomization and Approximation Techniques in Computer Science (APPROX/RANDOM)*, pages 15–28, 2009.

2 AmirMohsen Ahanchi, Alexandr Andoni, MohammadTaghi Hajiaghayi, Marina Knittel, and Peilin Zhong. Massively parallel tree embeddings for high dimensional spaces. In *Proceedings of the 35th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 77–88, 2023. doi:10.1145/3558481.3591096.

3 Hyung-Chan An, Ashkan Norouzi-Fard, and Ola Svensson. Dynamic facility location via exponential clocks. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 708–721, 2015.

4 Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 574–583, 2014. doi:10.1145/2591796.2591805.

5 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

6 Olivier Bachem, Mario Lucic, and Andreas Krause. Scalable k-means clustering via lightweight coresets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 1119–1127, 2018.

7 Mihai Bădoiu, Artur Czumaj, Piotr Indyk, and Christian Sohler. Facility location in sublinear time. In *Proceedings of the 32nd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 866–877, 2005. doi:10.1007/11523468_70.

8 Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable K-Means++. *Proc. VLDB Endow.*, 5(7):622–633, 2012.

9 Maria-Florina Balcan, Steven Ehrlich, and Yingyu Liang. Distributed k-means and k-median clustering on general communication topologies. In *NIPS*, pages 1995–2003, 2013.

10 MohammadHossein Bateni, Hossein Esfandiari, Manuela Fischer, and Vahab S. Mirrokni. Extreme k-center clustering. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, pages 3941–3949, 2021. doi:10.1609/aaai.v35i5.16513.

11 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. *Journal of the ACM*, 64(6):40:1–40:58, 2017. doi:10.1145/3125644.

12 Soheil Behnezhad, Moses Charikar, Weiyun Ma, and Li-Yang Tan. Almost 3-approximate correlation clustering in constant rounds. In *Proceedings of the 63rd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 720–731, 2022. doi:10.1109/FOCS54457.2022.00074.

13 Aditya Bhaskara and Maheshakya Wijewardena. Distributed clustering via LSH based data partitioning. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 569–578. PMLR, 2018. URL: https://proceedings.mlr.press/v80/bhaskara18a.html.

14 Yonatan Bilu and Nathan Linial. Are stable instances easy? *Combinatorics, Probability & Computing*, 21(5):643–660, 2012. doi:10.1017/S0963548312000193.

15 Guy E. Blelloch, Anupam Gupta, and Kanat Tangwongsan. Parallel probabilistic tree embeddings, k-me network design. In *Proceedings of the 24th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 205–213, 2012.

16 Guy E. Blelloch and Kanat Tangwongsan. Parallel approximation algorithms for facility-location problems. In *Proceedings of the 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 315–324, 2010. doi:10.1145/1810479.1810535.

17 Vladimir Braverman, Vincent Cohen-Addad, Shaofeng H.-C. Jiang, Robert Krauthgamer, Chris Schwiegelshohn, Mads Bech Toftrup, and Xuan Wu. The power of uniform sampling for coresets. In *Proceedings of the 63rd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 462–473, 2022. doi:10.1109/FOCS54457.2022.00051.

18 Vladimir Braverman, Gereon Frahling, Harry Lang, Christian Sohler, and Lin F. Yang. Clustering high dimensional dynamic data streams. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 576–585. PMLR, 2017.

19 Jaroslaw Byrka and Karen I. Aardal. An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. *SIAM Journal on Computing*, 39(6):2212–2231, 2010.

20 Mélanie Cambus, Fabian Kuhn, Shreyas Pai, and Jara Uitto. Time and space optimal massively parallel algorithm for the 2-ruling set problem. In *Proceedings of the 37th International Symposium on Distributed Computing (DISC)*, pages 11:1–11:12, 2023. doi:10.4230/LIPICS.DISC.2023.11.

21 Matteo Ceccarello, Andrea Pietracaprina, and Geppino Pucci. Solving k-center clustering (with outliers) in MapReduce and streaming, almost as accurately as sequentially. *Proc. VLDB Endow.*, 12(7):766–778, 2019.

22 Yi-Jun Chang and Da Wei Zheng. Fully scalable massively parallel algorithms for embedded planar graphs. In *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4410–4450, 2024. doi:10.1137/1.9781611977912.155.

23 Jiecao Chen, He Sun, David P. Woodruff, and Qin Zhang. Communication-optimal distributed clustering. In *Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 3720–3728, 2016. URL: https://proceedings.neurips.cc/paper/2016/hash/7503cfacd12053d309b6bed5c89de212-Abstract.html.

24 Xi Chen, Vincent Cohen-Addad, Rajesh Jayaram, Amit Levi, and Erik Waingarten. Streaming Euclidean MST to a Constant Factor. In *Proceedings of the 55th Annual Symposium on Theory of Computing (STOC)*, pages 156–169, 2023. doi:10.1145/3564246.3585168.

25 Xi Chen, Rajesh Jayaram, Amit Levi, and Erik Waingarten. New streaming algorithms for high dimensional EMD and MST. In *Proceedings of the 54th Annual Symposium on Theory of Computing (STOC)*, pages 222–233, 2022. doi:10.1145/3519935.3519979.

26 Vincent Cohen-Addad, Kasper Green Larsen, David Saulpic, and Chris Schwiegelshohn. Towards optimal lower bounds for k-median and k-means coresets. In *Proceedings of the 54th Annual Symposium on Theory of Computing (STOC)*, pages 1038–1051, 2022. doi:10.1145/3519935.3519946.

27 Vincent Cohen-Addad, Silvio Lattanzi, Ashkan Norouzi-Fard, Christian Sohler, and Ola Svensson. Parallel and efficient hierarchical k-median clustering. In *NeurIPS*, pages 20333–20345, 2021. URL: https://proceedings.neurips.cc/paper/2021/hash/aa495e18c7e3a21a4e48923b92048a61-Abstract.html.

28 Vincent Cohen-Addad, Vahab S. Mirrokni, and Peilin Zhong. Massively parallel k-means clustering for perturbation resilient instances. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 4180–4201. PMLR, 2022. URL: https://proceedings.mlr.press/v162/cohen-addad22b.html.

29 Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. A new coreset framework for clustering. In *Proceedings of the 53rd Annual Symposium on Theory of Computing (STOC)*, pages 169–182, 2021.

30 Sam Coy, Artur Czumaj, and Gopinath Mishra. On parallel k-center clustering. In *Proceedings of the 35th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 65–75, 2023. doi:10.1145/3558481.3591075.

31 Artur Czumaj, Peter Davies, and Merav Parter. Component stability in low-space massively parallel computation. In *Proceedings of the 40th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 481–491, 2021. doi:10.1145/3465084.3467903.

32 Artur Czumaj, Guichen Gao, Shaofeng H.-C. Jiang, Robert Krauthgamer, and Pavel Veselý. Fully scalable MPC algorithms for clustering in high dimension. *CoRR*, abs/2307.07848, 2023. doi:10.48550/arXiv.2307.07848.

33 Artur Czumaj, Shaofeng H.-C. Jiang, Robert Krauthgamer, Pavel Veselý, and Mingwei Yang. Streaming facility location in high dimension via geometric hashing. In *Proceedings of the 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 450–461, 2022. The latest version can be found at arXiv:2204.02095 and it has additional results. doi:10.1109/FOCS54457.2022.00050.

34 Mark de Berg, Leyla Biabani, and Morteza Monemizadeh. k-center clustering with outliers in the MPC and streaming model. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium, (IPDPS)*, pages 853–863, 2023. doi:10.1109/IPDPS54959.2023.00090.

35 Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using MapReduce. In *17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 681–689. ACM, 2011. doi:10.1145/2020408.2020515.

36 Arnold Filtser. Scattering and sparse partitions, and their applications. In *47th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 47:1–47:20, 2020. doi:10.4230/LIPIcs.ICALP.2020.47.

37 Naveen Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proceedings of the 37th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 302–309, 1996.

38 Joachim Gehweiler, Christiane Lammersen, and Christian Sohler. A distributed $O(1)$-approximation algorithm for the uniform facility location problem. *Algorithmica*, 68(3):643–670, 2014. doi:10.1007/s00453-012-9690-y.

39 Mohsen Ghaffari. Massively parallel algorithms, 2019. Lecture Notes from ETH Zurich. URL: http://people.csail.mit.edu/ghaffari/MPA19/Notes/MPA.pdf.

40 Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. Conditional hardness results for massively parallel computation from distributed lower bounds. In *Proceedings of the 60th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1650–1663, 2019. doi:10.1109/FOCS.2019.00097.

41 Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the MapReduce framework. In *Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC)*, pages 374–383, 2011. doi:10.1007/978-3-642-25591-5_39.

42 Sudipto Guha, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams. In *Proceedings of the 41st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 359–366, 2000.

43 Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the 36th Annual Symposium on Theory of Computing (STOC)*, pages 291–300, 2004. doi:10.1145/1007352.1007400.

44 James W. Hegeman and Sriram V. Pemmaraju. Sub-logarithmic distributed algorithms for metric facility location. *Distributed Computing*, 28(5):351–374, 2015. doi:10.1007/s00446-015-0243-x.

45 James W. Hegeman, Sriram V. Pemmaraju, and Vivek Sardeshmukh. Near-constant-time distributed algorithms on a congested clique. In *Proceedings of the 28th International Symposium on Distributed Computing (DISC)*, pages 514–530, 2014. doi:10.1007/978-3-662-45174-8_35.

46 Daniel J. Hsu and Matus Telgarsky. Greedy bi-criteria approximations for k-medians and k-means. *CoRR*, abs/1607.06203, 2016. arXiv:1607.06203.

47 Sungjin Im, Ravi Kumar, Silvio Lattanzi, Benjamin Moseley, and Sergei Vassilvitskii. Massively parallel computation: Algorithms and applications. *Foundations and Trends in Optimization*, 5(4):340–417, 2023.

48 Piotr Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proceedings of the 41st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 189–197, 2000. doi:10.1109/SFCS.2000.892082.

49 Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 373–380, 2004. doi:10.1145/1007352.1007413.

50 Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM*, 50(6):795–824, 2003. doi:10.1145/950620.950621.

51 Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001. doi:10.1145/375827.375845.

52 Rajesh Jayaram, Vahab Mirrokni, Shyam Narayanan, and Peilin Zhong. Massively parallel algorithms for high-dimensional Euclidean minimum spanning tree. In *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3960–3996, 2024. doi:10.1137/1.9781611977912.139.

53 Lujun Jia, Guolong Lin, Guevara Noubir, Rajmohan Rajaraman, and Ravi Sundaram. Universal approximations for TSP, Steiner tree, and set cover. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 386–395, 2005. doi:10.1145/1060590.1060649.

54 W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Proceedings of the Conference in Modern Analysis and Probability (New Haven, Connecticut, 1982)*, pages 189–206. American Mathematical Society, 1984.

55 Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for MapReduce. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 938–948, 2010. doi:10.1137/1.9781611973075.76.

56 Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Inf. Comput.*, 222:45–58, 2013.

57 Jyh-Han Lin and Jeffrey Scott Vitter. Approximation algorithms for geometric median problems. *Information Processing Letters*, 44(5):245–249, 1992.

58 Jyh-Han Lin and Jeffrey Scott Vitter. ε-approximations with minimum packing constraint violation. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 771–782, 1992. doi:10.1145/129712.129787.

59 Konstantin Makarychev, Yury Makarychev, Maxim Sviridenko, and Justin Ward. A bi-criteria approximation algorithm for k-means. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 14:1–14:20, 2016. doi:10.4230/LIPICS.APPROX-RANDOM.2016.14.

60 Gustavo Malkomes, Matt J. Kusner, Wenlin Chen, Kilian Q. Weinberger, and Benjamin Moseley. Fast distributed k-center clustering with outliers on massive data. In *NIPS*, pages 1063–1071, 2015.

61 Ramgopal R. Mettu and C. Greg Plaxton. The online median problem. *SIAM Journal on Computing*, 32(3):816–832, 2003. doi:10.1137/S0097539701383443.

62 Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. Shuffles and circuits (On lower bounds for modern parallel computation). *Journal of the ACM*, 65(6):41:1–41:24, 2018. doi:10.1145/3232536.

63 Zhao Song, Lin F. Yang, and Peilin Zhong. Sensitivity sampling over dynamic geometric data streams with applications to k-clustering. *CoRR*, abs/1802.00459, 2018. arXiv:1802.00459.

64 Dennis Wei. A constant-factor bi-criteria approximation guarantee for k-means++. In *NIPS*, volume 29, pages 604–612, 2016. URL: https://proceedings.neurips.cc/paper/2016/hash/357a6fdf7642bf815a88822c447d9dc4-Abstract.html.

65 Grigory Yaroslavtsev and Adithya Vadapalli. Massively parallel algorithms and hardness for single-linkage clustering under ℓ_p distances. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 5596–5605. PMLR, 2018. URL: http://proceedings.mlr.press/v80/yaroslavtsev18a.html.

Computing Tree Decompositions with Small Independence Number

Clément Dallard
Department of Informatics, University of Fribourg, Switzerland

Fedor V. Fomin
University of Bergen, Norway

Petr A. Golovach
University of Bergen, Norway

Tuukka Korhonen
University of Bergen, Norway

Martin Milanič
FAMNIT and IAM, University of Primorska, Koper, Slovenia

Abstract

The independence number of a tree decomposition is the maximum of the independence numbers of the subgraphs induced by its bags. The tree-independence number of a graph is the minimum independence number of a tree decomposition of it. Several NP-hard graph problems, like maximum weight independent set, can be solved in time $n^{\mathcal{O}(k)}$ if the input n-vertex graph is given together with a tree decomposition of independence number k. Yolov in [SODA 2018] gave an algorithm that given an n-vertex graph G and an integer k, in time $n^{\mathcal{O}(k^3)}$ either constructs a tree decomposition of G whose independence number is $\mathcal{O}(k^3)$ or correctly reports that the tree-independence number of G is larger than k.

In this paper, we first give an algorithm for computing the tree-independence number with a better approximation ratio and running time and then prove that our algorithm is, in some sense, the best one can hope for. More precisely, our algorithm runs in time $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(k)}$ and either outputs a tree decomposition of G with independence number at most $8k$, or determines that the tree-independence number of G is larger than k. This implies $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(k)}$-time algorithms for various problems, like maximum weight independent set, parameterized by the tree-independence number k without needing the decomposition as an input. Assuming Gap-ETH, an $n^{\Omega(k)}$ factor in the running time is unavoidable for any approximation algorithm for the tree-independence number.

Our second result is that the exact computation of the tree-independence number is para-NP-hard: We show that for every constant $k \geq 4$ it is NP-hard to decide if a given graph has the tree-independence number at most k.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases tree-independence number, approximation, parameterized algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.51

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2207.09993` [13]

Funding The research leading to these results has received funding from the Research Council of Norway via the project BWCA (grant no. 314528), by the Slovenian Research and Innovation Agency (I0-0035, research program P1-0285 and research projects J1-3001, J1-3002, J1-3003, J1-4008, and J1-4084), and by the research program CogniCom (0013103) at the University of Primorska.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 51; pp. 51:1–51:18

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Tree decompositions are among the most popular tools in graph algorithms. The crucial property of tree decompositions exploited in the majority of dynamic programming algorithms is that each bag of the decomposition can interact with an optimal solution only in a bounded number of vertices. The common measure of a tree decomposition is the width, that is, the maximum size of a bag in the decomposition (minus 1). The corresponding graph parameter is the treewidth of a graph. Many problems that are intractable on general graphs can be solved efficiently when the treewidth of a graph is bounded.

However, it is not always the size of a bag that matters. For example, suppose that every bag of the decomposition is a clique, that is, the graph is chordal. Since every independent set intersects each of the clique-bags in at most one vertex, dynamic programming still computes maximum weight independent sets in such graphs in polynomial time even if the bags could be arbitrarily large. An elegant approach to capturing such properties of tree decompositions is the notion of the tree-independence number of a graph. The *independence number* of a tree decomposition is the maximum of the independence numbers (that is, the maximum size of an independent set) of the subgraphs induced by its bags. The *tree-independence number* of a graph G, denoted by $\mathsf{tree}\text{-}\alpha(G)$, is the minimum independence number of a tree decomposition of G. In particular, the tree-independence number of a chordal graph is at most one, and for any graph G, the value $\mathsf{tree}\text{-}\alpha(G)$ does not exceed the treewidth of G (plus 1) or the independence number $\alpha(G)$ of G.

The family of graph classes with bounded tree-independence number forms a significant generalization of graph classes with bounded treewidth. It also contains dense graphs classes, including graph classes with bounded independence number; classes of intersection graphs of connected subgraphs of graphs with bounded treewidth, studied by Bodlaender, Gustedt, and Telle [6], which in particular include classes of *H-graphs*, that is, intersection graphs of connected subgraphs of a subdivision of a fixed multigraph H, introduced in 1992 by Bíró, Hujter, and Tuza [4] and studied more recently in a number of papers [8, 9, 22]; classes of graphs in which all minimal separators have bounded size, studied by Skodinis in 1999 [38]; and, more generally, classes of graphs in which all minimal separators induce subgraphs with bounded independence number, studied by Dallard, Milanič, and Štorgel [15].

Our results. Yolov [40] gave an algorithm that for a given n-vertex graph G and integer k, in time $n^{\mathcal{O}(k^3)}$ either constructs a tree decomposition of G whose independence number is $\mathcal{O}(k^3)$ or correctly reports that the tree-independence number of G is larger than k. Our first main result is the following improvement over Yolov's algorithm.

▶ **Theorem 1.** *There is an algorithm that, given an n-vertex graph G and an integer k, in time $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(k)}$ either outputs a tree decomposition of G with independence number at most $8k$, or concludes that the tree-independence number of G is larger than k.*

The performance of the algorithm from Theorem 1 (the running time and the need of approximation) are in some sense optimal, for the following reasons. First, by a simple reduction that given an n-vertex graph G produces a $2n$-vertex graph G' with $\mathsf{tree}\text{-}\alpha(G') = \alpha(G)$ hardness results for the independence number (or clique) translate into hardness results for the tree-independence number. In particular, from the result of Lin [30] it follows that a constant-factor approximation of the tree-independence number is W[1]-hard, and from the result of Chalermsook, Cygan, Kortsarz, Laekhanukit, Manurangsi, Nanongkai,

and Trevisan [7] it follows that assuming Gap-ETH[1], there is no $f(k) \cdot n^{o(k)}$-time $g(k)$-approximation algorithm for the tree-independence number for any computable functions f and g. In particular, the time complexity obtained in Theorem 1 is optimal in the sense that an $n^{\Omega(k)}$ factor is unavoidable assuming Gap-ETH, even if the approximation ratio would be drastically weakened.

The above arguments do not exclude the possibility of *exact* computation of the tree-independence number in time $n^{f(k)}$ for some function f. The computational complexity of recognizing graphs with the tree-independence numbers at most k for a fixed integer $k \geq 2$ was asked as an open problem by Dallard, Milanič, and Štorgel [15, Question 8.3][2]. While for values $k = 2$ and $k = 3$ the question remains open, our next result resolves it for any constant $k \geq 4$.[3]

▶ **Theorem 2** (⋆). *For every constant $k \geq 4$, it is NP-complete to decide whether* tree-$\alpha(G) \leq k$ *for a given graph G.*

Let us observe that Theorem 2 implies also that, assuming P ≠ NP, there is no $n^{f(k)}$-time approximation algorithm for the tree-independence number with approximation ratio less than 5/4.

We supplement our main results with a second NP-completeness proof for a problem closely related to computing the tree-independence number and the algorithm of Theorem 1. We consider the problem where we are given a graph G, two non-adjacent vertices u and v, and an integer k, and the task is to decide if u and v can be separated by removing a set of vertices that induces a subgraph with independence number at most k. We show in Theorem 14 that this problem is NP-complete for any fixed integer $k \geq 3$. This hardness result is motivated by the fact that the algorithm of Theorem 1 finds separators with bounded independence number as a subroutine. While for the algorithm of Theorem 1 we design a $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(k)}$-time 2-approximation algorithm for a generalization of this problem, assuming that a tree decomposition of independence number $\mathcal{O}(k)$ is given, this proof shows that this step of the algorithm cannot be turned into an exact algorithm (in our reduction, we can construct along the graph G also a tree decomposition of independence number $\mathcal{O}(k)$ of G).

Previous work and applications of Theorem 1. The notion of the tree-independence number is very natural and it is not surprising that it was introduced independently by several researchers [15, 40]. Yolov in [40] introduced a new width measure called *minor-matching hypertree-width*, tree-μ.[4] He proved that a number of problems including Maximum Independent Set, k-Colouring, and Graph Homomorphism permit polynomial-time solutions for graphs with bounded minor-matching hypertree width. Furthermore, Yolov showed that the minor-matching hypertree-width of a graph is equal to the tree-independence number of the square of its line graph, that is, tree-$\mu(G)$ = tree-$\alpha(L(G)^2)$ holds for all graphs G, where $L(G)^2$ is the graph whose vertices are the edges of G, with two distinct edges adjacent if and only if they have nonempty intersection or there is a third edge

[1] Gap-ETH states that for some constant $\epsilon > 0$, distinguishing between a satisfiable 3-SAT formula and one that is not even $(1-\epsilon)$-satisfiable requires exponential time (see [20, 32]).

[2] There is a linear-time algorithm for deciding if a given graph has the tree-independence number at most 1, because such graphs are exactly the chordal graphs, see, e.g., [25].

[3] The proofs of the statements marked (⋆) are omitted due to space constraints and can be found in the full version of the paper [13].

[4] Minor-matching hypertree-width is defined for hypergraphs, but algorithms for computing decompositions for it are only known for graphs.

intersecting both. Moreover, a tree decomposition of $L(G)^2$ with independence number at most k can be turned into a tree decomposition of G with minor-matching hypertree-width at most k. Then, Yolov gave an $n^{\mathcal{O}(k^3)}$-time $\mathcal{O}(k^2)$-approximation algorithm computing the tree-independence number of an n-vertex graph, implying also the same bounds for computing the minor-matching hypertree-width of a graph. Theorem 1 improves the running time and approximation ratio of Yolov's algorithm. Pipelined with Yolov's reduction, Theorem 1 also implies an 8-approximation of minor-matching hypertree-width of graphs in time $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(k)}$.

Theorem 2 implies also the NP-hardness of deciding whether $\mathsf{tree}\text{-}\mu(G) \leq k$ for every constant $k \geq 4$ because a simple reduction that attaches a pendant vertex to every vertex of a graph G produces a graph G' such that $\mathsf{tree}\text{-}\mu(G') = \mathsf{tree}\text{-}\alpha(G)$ (see [40]).

The tree-independence number of a graph was introduced independently by Yolov [40] and Dallard et al. [15].[5] The original motivation for Dallard et al. [15] stems from structural graph theory. In 2020, Dallard, Milanič, and Štorgel [14, 17] initiated a systematic study of (tw, ω)-bounded graph classes, that is, hereditary graph classes in which the treewidth can only be large due to the presence of a large clique. While (tw, ω)-bounded graph classes are known to possess some good algorithmic properties related to clique and coloring problems (see [9, 10, 14, 15, 17]), the extent to which this property has useful algorithmic implications for problems related to independent sets is an open problem. The connection with the tree-independence number follows from Ramsey's theorem, which implies that graph classes with bounded tree-independence number are (tw, ω)-bounded with a polynomial binding function (see [15]). Dallard, Milanič, and Štorgel [16] conjecture the converse, namely, that every (tw, ω)-bounded graph class has bounded tree-independence number. A related research direction in structural graph theory is the study of induced obstructions to bounded treewidth and tree-independence number; see for example the recent work of Abrishami, Alecu, Chudnovsky, Hajebi, Spirkl, and Vušković [1].

In our opinion, the most interesting application of Theorem 1 lies in the area of graph algorithms for NP-hard optimization problems. Dallard et al. [15] and Yolov in [40] have shown that certain NP-hard optimization problems like Maximum Independent Set, Graph Homomorphism, or Maximum Induced Matching problems can be solved in time $n^{\mathcal{O}(k)}$ if the input graph is given with a tree decomposition of independence number at most k. Lima et al. [29] extended this idea to generic packing problems in which any two of the chosen subgraphs have to be at pairwise distance at least d, for even d. They also obtained an algorithmic metatheorem for the problem of finding a maximum-weight sparse (bounded chromatic number) induced subgraph satisfying an arbitrary but fixed property expressible in counting monadic second-order logic (CMSO_2). In particular, the metatheorem implies polynomial-time solvability of several classical problems like finding the largest induced forest (which is equivalent to Minimum Feedback Vertex Set), finding the largest induced bipartite subgraph (which is equivalent to Minimum Odd Cycle Transversal), finding the maximum number of pairwise disjoint and non-adjacent cycles (Maximum Induced Cycle Packing), and finding the largest induced planar subgraph (which is equivalent to Planarization).

However, the weak spot in all these algorithmic approaches is the requirement that a tree decomposition with bounded independence number is given with the input. Theorem 1 fills this gap by constructing a decomposition of bounded independence number in time that asymptotically matches or improves the time required to solve all these optimization problems.

[5] Yolov called it *α-treewidth* in [40].

Theorem 1 appears to be a handy tool in the subarea of computational geometry concerning optimization problems on geometric graphs. Treewidth plays a fundamental role in the design of exact and approximation algorithms on planar graphs (and more generally, H-minor-free graphs) [3, 19, 26]. The main property of such graphs is that they enjoy the bounded local treewidth property. In other words, any planar graph of a small diameter has a small treewidth. A natural research direction is to extend such methods to intersection graphs of geometric objects [23, 31]. However, even for very "simple" objects like unit disks, the corresponding intersection graphs do not have locally bounded treewidth. On the other hand, in many scenarios, the treewidth-based methods on such graphs could be replaced by tree decompositions of bounded independence number. In particular, de Berg, Bodlaender, Kisfaludi-Bak, Marx, and van der Zanden use tree decompositions whose bags are covered by a small number of cliques, and thus of small independence number, to design subexponential-time algorithms on geometric graph classes [18]. Galby, Munaro, and Yang [24] use Theorem 1 for obtaining polynomial-time approximation schemes for several packing problems on geometric graphs. It is interesting to note that algorithms on geometric graphs often require geometric representation of a graph. Sometimes, like for unit disk graphs, finding such a representation is a challenging computational task [27]. On the contrary, Theorem 1 does not need the geometric properties of objects or their geometric representations and thus could be used for developing so-called robust algorithms [35] on geometric graphs [11].

In parameterized algorithms, Fomin and Golovach [21] and Jacob, Panolan, Raman, and Sahlot [28] used tree decompositions where each bag is obtained from a clique by deleting at most k edges or adding at most k vertices, respectively. These type of decompositions are special types of tree decompositions with bounded independence numbers.

The rest of this paper is organized as follows. In Section 2, we overview the proof of our main algorithmic result Theorem 1. In Section 3 we recall definitions. Section 4 is devoted to the proof of Theorem 1. In Section 5, we survey our computational lower bounds. Due to space constraints, the proofs of the hardness results are omitted and are available in the full version of the paper [13]. We conclude in Section 6 with final remarks and open problems.

2 Overview

In this section we sketch the proof of Theorem 1. For simplicity, we sketch here a version with approximation ratio 11 instead of 8. The difference between the 11-approximation and 8-approximation is that for 11-approximation it is sufficient to use 2-way separators, while for 8-approximation we use 3-way separators.

On a high level, our algorithm follows the classical technique of constructing a tree decomposition by repeatedly finding balanced separators. This technique was introduced by Robertson and Seymour in Graph Minors XIII [37], was used for example in [5, 18, 28], and an exposition of it was given in [12, Section 7.6.2] and in [34].

The challenge in applying this technique is the need to compute separators that are both balanced and small with respect to the independence numbers of the involved vertex sets. Our main technical contribution is an approximation algorithm for finding such separators. In what follows, we will sketch an algorithm that given a graph G, a parameter k, and a set of vertices $W \subseteq V(G)$ with independence number $\alpha(W) = 9k$ either finds a partition (S, C_1, C_2) of $V(G)$ such that each S, C_1, and C_2 are nonempty, S separates C_1 from C_2, $\alpha(S) \leq 2k$, and $\alpha(W \cap C_i) \leq 7k$ for both $i = 1, 2$, or determines that the tree-independence number of G is larger than k. (For $S \subseteq V(G)$ we use $\alpha(S)$ to denote the independence

number of the induced subgraph $G[S]$.) Our algorithm for finding such balanced separators works in two parts. First, we reduce finding balanced separators to finding separators, and then we give a 2-approximation algorithm for finding separators.

At first glance, it is not even clear that small tree-independence number guarantees the existence of such balanced separators. To prove the existence of balanced separators and to reduce the finding of balanced separators to finding separators between specified sets of vertices, instead of directly enforcing that both sides of the separation have a small independence number, we enforce that both sides of the separation have sufficiently large independence number. More precisely, we pick an arbitrary independent set $I \subseteq W$ of size $|I| = 9k$. By making use of the properties of tree decompositions, it is possible to show that there exists a separation (S, C_1, C_2) with $|I \cap C_i| \leq 6k$ for $i \in \{1, 2\}$ and $\alpha(S) \leq k$. Hence $|I \cap C_i| \geq 2k$ for $i \in \{1, 2\}$. By enforcing the condition $|I \cap C_i| \geq 2k$ for both $i \in \{1, 2\}$, we will have that $\alpha(W \cap C_i) \leq 7k$ for both $i \in \{1, 2\}$. (Note that $\alpha(W \cap C_1) + \alpha(W \cap C_2) \leq \alpha(W)$.) Therefore, to find a balanced separator for W or to conclude that the tree-independence number of G is larger than k, it is sufficient to select an arbitrary independent set $I \subseteq W$ with $|I| = 9k$, do $2^{\mathcal{O}(k)}$ guesses for sets $I \cap C_1$ and $I \cap C_2$, and for each of the guesses search for a separator between the sets $I \cap C_1$ and $I \cap C_2$.

In the separator finding algorithm the input includes two sets $V_1 = I \cap C_1$ and $V_2 = I \cap C_2$, and the task is to find a set of vertices S disjoint from both V_1 and V_2 separating V_1 from V_2 with $\alpha(S) \leq 2k$ or to conclude that no such separator S with $\alpha(S) \leq k$ exists. Our algorithm works by first using multiple stages of different branching steps, and then arriving at a special case which is solved by rounding a linear program. We explain some details in what follows.

First, by using iterative compression around the whole algorithm, we can assume that we have a tree decomposition with independence number $\mathcal{O}(k)$ available. We show that any set $S \subseteq V(G)$ with $\alpha(S) \leq k$ can be covered by $\mathcal{O}(k)$ bags of the tree decomposition. This implies that by first guessing the covering bags, we reduce the problem to the case where we search for a separator $S \subseteq R$ for some set $R \subseteq V(G)$ with independence number $\alpha(R) = \mathcal{O}(k^2)$.

Then, we use a branching procedure to reduce the problem to the case where $R \subseteq N[V_1 \cup V_2]$. In the branching, we select a vertex $v \in R \setminus N[V_1 \cup V_2]$, and branch into three subproblems, corresponding to including v into V_1, into V_2, or into a partially constructed solution S. The key observation here is that if we branch on vertices $v \in R \setminus N[V_1 \cup V_2]$, then the branches where v is included in V_1 or in V_2 reduce the value $\alpha(R \setminus N[V_1 \cup V_2])$. By first handling the case with $\alpha(R \setminus N[V_1 \cup V_2]) \geq 2k$ by branching on $2k$ vertices at the same time and then branching on single vertices, this branching results in $2^{\mathcal{O}(\alpha(R))} n^{\mathcal{O}(k)} = 2^{\mathcal{O}(k^2)} n^{\mathcal{O}(k)}$ instances where $R \subseteq N[V_1 \cup V_2]$.

Finally, when we arrive at the subproblem where $R \subseteq N[V_1 \cup V_2]$, we design a 2-approximation algorithm by rounding a linear program. For $v \in R$, let us have variables x_v with $x_v = 1$ indicating that $v \in S$ and $x_v = 0$ indicating that $v \notin S$. Because $R \subseteq N[V_1 \cup V_2]$, the fact that $S \subseteq R$ separates V_1 from V_2 can be encoded by only using inequalities of form $x_v + x_u \geq 1$. To bound the independence number of S, for every independent set I of size $|I| = 2k+1$ we add a constraint that $\sum_{v \in I} x_v \leq k$. Now, a separator S with $\alpha(S) \leq k$ corresponds to an integer solution. We then find a fractional solution and round x_v to 1 if $x_v \geq 1/2$ and to 0 otherwise. Note that this satisfies the $x_v + x_u \geq 1$ constraints, so the rounded solution corresponds to a separator. To bound the independence number of the rounded solution, note that $\sum_{v \in I} x_v \leq 2k$ for independent sets I of size $|I| = 2k + 1$, therefore implying that $\alpha(S) \leq 2k$.

3 Preliminaries

We denote the vertex set and the edge set of a graph $G = (V, E)$ by $V(G)$ and $E(G)$, respectively. The *neighborhood* of a vertex v in G is the set $N_G(v)$ of vertices adjacent to v in G, and the *closed neighborhood* of v is the set $N_G[v] = N_G(v) \cup \{v\}$. These concepts are extended to sets $X \subseteq V(G)$ so that $N_G[X]$ is defined as the union of all closed neighborhoods of vertices in X, and $N_G(X)$ is defined as the set $N_G[X] \setminus X$. The *degree* of v, denoted by $d_G(v)$, is the cardinality of the set $N_G(v)$. When there is no ambiguity, we may omit the subscript G in the notations of the degree, and open and closed neighborhoods, and thus simply write $d(v)$, $N(v)$, and $N[v]$, respectively.

Given a set $X \subseteq V(G)$, we denote by $G[X]$ the subgraph of G induced by X. We also write $G \setminus X = G[V(G) \setminus X]$. Similarly, given a vertex $v \in V(G)$, we denote by $G \setminus v$ the graph obtained from G by deleting v. A graph G is *chordal* if it has no induced cycles of length at least four.

A *clique* (resp. *independent set*) in a graph G is a set of pairwise adjacent (resp. non-adjacent) vertices. The *independence number* of G, denoted by $\alpha(G)$, is the maximum size of an independent set in G. For a set of vertices $X \subseteq V(G)$, the independence number of X is $\alpha(X) = \alpha(G[X])$.

Let $(V_1, V_2, \ldots, V_t)$ be a tuple of disjoint subsets of $V(G)$. A $(V_1, V_2, \ldots, V_t)$-*separator* is a set $S \subseteq V(G)$ such that $S \cap V_i = \emptyset$ for each $i \in [t]$, and in the graph $G \setminus S$ there is no path from V_i to V_j for all pairs $i \neq j$. Such a separator is sometimes called a t-way separator. Note that if V_i is adjacent to V_j for some $i \neq j$, then no $(V_1, V_2, \ldots, V_t)$-separator exists.

A *tree decomposition* of a graph G is a pair $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ where T is a tree and every node t of T is assigned a vertex subset $X_t \subseteq V(G)$ called a bag such that the following conditions are satisfied: (1) every vertex of G is in at least one bag, (2) for every edge $uv \in E(G)$ there exists a node $t \in V(T)$ such that X_t contains both u and v, and (3) for every vertex $u \in V(G)$ the subgraph T_u of T induced by the set $\{t \in V(T) : u \in X_t\}$ is connected (that is, a tree). The *independence number* of a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of a graph G, denoted by $\alpha(\mathcal{T})$, is defined as follows:

$$\alpha(\mathcal{T}) = \max_{t \in V(T)} \alpha(X_t)\,.$$

The *tree-independence number* of G, denoted by $\mathsf{tree}\text{-}\alpha(G)$, is the minimum independence number among all tree decompositions of G.

4 An 8-approximation algorithm for tree-independence number

In this section we prove Theorem 1, that is, we give a $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(k)}$-time algorithm for either computing tree decompositions with independence number at most $8k$ or deciding that the tree-independence number of the graph is more than k. Our algorithm consists of three parts. First, we give a $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(k)}$-time 2-approximation algorithm for finding 3-way separators with independence number at most k, with the assumption that a tree decomposition with independence number $\mathcal{O}(k)$ is given with the input. Then, we apply this separator finding algorithm to find balanced separators, and then apply balanced separators in the fashion of the Robertson-Seymour treewidth approximation algorithm [37] to construct a tree decomposition with independence number at most $8k$. The requirement for having a tree decomposition with independence number $\mathcal{O}(k)$ as an input in the separator algorithm is satisfied by iterative compression (see, e.g., [12]), as we explain at the end of Section 4.3.

The presentation of the algorithm in this section is in reverse order compared to the presentation we gave in Section 2.

4.1 Finding approximate separators

In this subsection, we show the following theorem.

▶ **Theorem 3.** *There is an algorithm, that given a graph G, an integer k, a tree decomposition $\mathcal{T}$ of G with independence number $\alpha(\mathcal{T}) = \mathcal{O}(k)$, and three disjoint sets of vertices $V_1, V_2, V_3 \subseteq V(G)$, in time $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(k)}$ either reports that no (V_1, V_2, V_3)-separator with independence number at most k exists, or returns a (V_1, V_2, V_3)-separator with independence number at most $2k$.*

To prove Theorem 3, we define a more general problem that we call PARTIAL 3-WAY α-SEPARATOR, which is the same as the problem of Theorem 3 except that two sets S_0 and R are given with the input, and we are only looking for separators S with $S_0 \subseteq S \subseteq S_0 \cup R$.

▶ **Definition 4** (PARTIAL 3-WAY α-SEPARATOR)**.** *An instance of PARTIAL 3-WAY α-SEPARATOR is a 5-tuple $(G, (V_1, V_2, V_3), S_0, R, k)$, where G is a graph, V_1, V_2, V_3, S_0, and R are disjoint subsets of $V(G)$, and k is an integer. A solution to PARTIAL 3-WAY α-SEPARATOR is a (V_1, V_2, V_3)-separator S such that $S_0 \subseteq S \subseteq S_0 \cup R$. A 2-approximation algorithm for PARTIAL 3-WAY α-SEPARATOR either returns a solution S with $\alpha(S) \leq 2k$ or determines that there is no solution S with $\alpha(S) \leq k$.*

We give a 2-approximation algorithm for PARTIAL 3-WAY α-SEPARATOR. Then Theorem 3 will follow by setting $S_0 = \emptyset$ and $R = V(G) \setminus (V_1 \cup V_2 \cup V_3)$.

We give our 2-approximation algorithm by giving 2-approximation algorithms for increasingly more general cases of PARTIAL 3-WAY α-SEPARATOR. First, we give a linear programming based 2-approximation algorithm for the special case when $R \subseteq N(V_1 \cup V_2 \cup V_3)$. Then, we use branching and the first algorithm to give a 2-approximation algorithm for the case when $\alpha(R) = \mathcal{O}(k^2)$. Then, we use the input tree decomposition to reduce the general case to the case where $\alpha(R) = \mathcal{O}(k^2)$.

Let us make some observations about trivial instances of PARTIAL 3-WAY α-SEPARATOR. First, we can assume that $\alpha(S_0) \leq k$, as otherwise any solution S has $\alpha(S) > k$ and we can return no immediately. All our algorithms include an $n^{\mathcal{O}(k)}$ factor in the time complexity, so it can be assumed that this condition is always tested. Then, we can also always return no if V_i is adjacent to V_j, where $i \neq j$. This can be checked in polynomial time, so we can assume that this condition is always tested. Note that testing this condition implies that $N(V_1 \cup V_2 \cup V_3) = N(V_1) \cup N(V_2) \cup N(V_3)$. For simplicity, we write $N(V_1 \cup V_2 \cup V_3)$ as it is shorter.

We start with the linear programming based 2-approximation algorithm for the case when $R \subseteq N(V_1 \cup V_2 \cup V_3)$.

▶ **Lemma 5.** *There is an $n^{\mathcal{O}(k)}$-time 2-approximation algorithm for PARTIAL 3-WAY α-SEPARATOR when $R \subseteq N(V_1 \cup V_2 \cup V_3)$.*

Proof. First, note that we may assume that for all $i, j \in \{1, 2, 3\}$, $i \neq j$, there is no path in the graph $G \setminus (R \cup S_0)$ between a vertex of V_i and a vertex of V_j, since otherwise the given instance has no solution at all. Furthermore, under this assumption, we can safely replace each set V_i with the set of vertices reachable from a vertex in V_i in the graph $G \setminus (R \cup S_0)$. We thus arrive at an instance such that $R \subseteq N(V_1 \cup V_2 \cup V_3) \subseteq R \cup S_0$. We then make an integer programming formulation of the problem (with $n^{\mathcal{O}(k)}$ constraints) and show that it gives a 2-approximation by rounding a fractional solution.

For every vertex $v \in R \cup S_0$ we introduce a variable x_v, with the interpretation that $x_v = 1$ if $v \in S$ and $x_v = 0$ otherwise. We force S_0 to be in the solution by adding constraints $x_v = 1$ for all $v \in S_0$. Then, we say that a pair (v_i, v_j) of vertices with $v_i \in N(V_i)$, $v_j \in N(V_j)$,

$i \neq j$, is *connected* if there is a v_i, v_j-path in the graph $G \setminus ((S_0 \cup R) \setminus \{v_i, v_j\})$. For any pair (v_i, v_j) of connected vertices we introduce a constraint $x_{v_i} + x_{v_j} \geq 1$ indicating that v_i or v_j should be selected to the solution. Note that here it can happen that $v_i = v_j$, corresponding to the case when $v_i \in N(V_i) \cap N(V_j)$, resulting in a constraint forcing v_i to be in the solution. Finally, for every independent set $I \subseteq R \cup S_0$ of size $|I| = 2k + 1$, we introduce a constraint $\sum_{v \in I} x_v \leq k$.

We observe that any solution S with $\alpha(S) \leq k$ corresponds to a solution to the integer program. In particular, any (V_1, V_2, V_3)-separator S must satisfy the $x_{v_i} + x_{v_j} \geq 1$ constraints for connected pairs (v_i, v_j), as otherwise there would be a V_i,V_j-path in $G \setminus S$, and a separator with $\alpha(S) \leq k$ satisfies the independent set constraints as otherwise it would contain an independent set larger than k.

Then, we show that any integral solution that satisfies the $x_{v_i} + x_{v_j} \geq 1$ constraints for connected pairs (v_i, v_j) and the constraints $x_v = 1$ for $v \in S_0$ corresponds to a (V_1, V_2, V_3)-separator. Suppose that all such constraints are satisfied by an integral solution corresponding to a set S, but there is a path from V_i to V_j with $i \neq j$ in $G \setminus S$. Take a shortest path connecting $N(V_i) \setminus S$ to $N(V_j) \setminus S$ in $G \setminus S$ for any $i \neq j$, and note that the intermediate vertices of such path do not intersect $N(V_1 \cup V_2 \cup V_3)$, as otherwise we could get a shorter path, and therefore do not intersect R. By $S_0 \subseteq S$, we have that the intermediate vertices do not intersect $R \cup S_0$, so this path is in fact a path in the graph $G \setminus ((S_0 \cup R) \setminus \{v_i, v_j\})$, where $v_i \in N(V_i) \setminus S$ is the first vertex and $v_j \in N(V_j) \setminus S$ is the last vertex. However, in this case (v_i, v_j) is a connected pair and we have a constraint $x_{v_i} + x_{v_j} \geq 1$, which would be violated by such an integral solution, so we get a contradiction.

We solve the linear program in polynomial time (which is $n^{\mathcal{O}(k)}$ as the number of variables is $\mathcal{O}(n)$ and the number of constraints is $n^{\mathcal{O}(k)}$), and round the solution by rounding x_v to 1 if $x_v \geq 1/2$ and otherwise to 0. This rounding will satisfy the $x_{v_i} + x_{v_j} \geq 1$ constraints for connected pairs, so the resulting solution corresponds to a separator. Then, by the constraints $\sum_{v \in I} x_v \leq k$ for independent sets I of size $2k + 1$, the rounded solution will satisfy $\sum_{v \in I} x_v \leq 2k$ for independent sets I of size $2k+1$. Therefore, there are no independent sets of size $2k + 1$ in the resulting solution, so its independence number is at most $2k$. ◀

We will pipeline Lemma 5 with branching to obtain a 2-approximation algorithm for PARTIAL 3-WAY α-SEPARATOR in a setting where $\alpha(R)$ is small. In our final algorithm, $\alpha(R)$ will be bounded by $\mathcal{O}(k^2)$, and this is the part that causes the $2^{\mathcal{O}(k^2)}$ factor in the time complexity.

Let us observe that we can naturally branch on vertices in R in instances of PARTIAL 3-WAY α-SEPARATOR.

▶ **Lemma 6** (Branching). *Let $\mathcal{I} = (G, (V_1, V_2, V_3), S_0, R, k)$ be an instance of* PARTIAL 3-WAY α-SEPARATOR*, and let $v \in R$. If S is a solution of $\mathcal{I}$, then S is also a solution of at least one of*

1. $(G, (V_1 \cup \{v\}, V_2, V_3), S_0, R \setminus \{v\}, k)$,
2. $(G, (V_1, V_2 \cup \{v\}, V_3), S_0, R \setminus \{v\}, k)$,
3. $(G, (V_1, V_2, V_3 \cup \{v\}), S_0, R \setminus \{v\}, k)$, *or*
4. $(G, (V_1, V_2, V_3), S_0 \cup \{v\}, R \setminus \{v\}, k)$.

Moreover, any solution of any of the instances 1-4 is also a solution of $\mathcal{I}$.

Proof. If S is a solution of $\mathcal{I}$, we can partition $V(G) \setminus S$ to parts $V_1' \supseteq V_1$, $V_2' \supseteq V_2$, and $V_3' \supseteq V_3$ by including the vertices in any connected component of $G \setminus S$ containing vertices of V_i into V_i' for all $i \in \{1, 2, 3\}$, and including the vertices in connected components containing

no vertices of V_1, V_2, V_3 into V_1', resulting in a partition (V_1', V_2', V_3') of $V(G) \setminus S$ such that S is a (V_1', V_2', V_3')-separator. Therefore, the first branch corresponds to when $v \in V_1'$, the second when $v \in V_2'$, the third when $v \in V_3'$, and the fourth when $v \in S$.

The direction that any solution of any of the instances 1-4 is also a solution of the original instance is immediate from the fact that we do not remove vertices from any V_i or S_0. ◀

Lemma 6 allows to branch into four subproblems, corresponding to the situation whether a vertex from R goes to V_1, V_2, V_3, or to S_0. Now, our goal is to branch until we derive an instance with $R \subseteq N(V_1 \cup V_2 \cup V_3)$, which can be solved by Lemma 5. In particular, we would like to branch on vertices $v \in R \setminus N(V_1 \cup V_2 \cup V_3)$. The following lemma shows that we can use $\alpha(R \setminus N(V_1 \cup V_2 \cup V_3))$ as a measure of progress in the branching.

▶ **Lemma 7.** *For any vertex $v \in R \setminus N(V_1 \cup V_2 \cup V_3)$ it holds that $\alpha(R \setminus (N[v] \cup N(V_1 \cup V_2 \cup V_3))) < \alpha(R \setminus N(V_1 \cup V_2 \cup V_3))$.*

Proof. If $\alpha(R \setminus (N[v] \cup N(V_1 \cup V_2 \cup V_3))) \geq \alpha(R \setminus N(V_1 \cup V_2 \cup V_3))$, then we could take an independent set $I \subseteq R \setminus (N[v] \cup N(V_1 \cup V_2 \cup V_3))$ such that $|I| = \alpha(R \setminus N(V_1 \cup V_2 \cup V_3)$ and construct an independent set $I \cup \{v\} \subseteq R \setminus N(V_1 \cup V_2 \cup V_3)$ of size $|I \cup \{v\}| > \alpha(R \setminus N(V_1 \cup V_2 \cup V_3))$, which is a contradiction. ◀

Lemma 7 implies that when branching on a vertex $v \in R \setminus N(V_1 \cup V_2 \cup V_3)$, the branches where v is moved to V_1, V_2, or V_3 decrease $\alpha(R \setminus N(V_1 \cup V_2 \cup V_3))$. This will be the main idea of our algorithm for the case when $\alpha(R)$ is bounded, which we give next.

▶ **Lemma 8.** *There is a $2^{\mathcal{O}(\alpha(R))}n^{\mathcal{O}(k)}$-time 2-approximation algorithm for* PARTIAL 3-WAY α-SEPARATOR.

Proof. We give a branching algorithm, whose base case is the case when $R \subseteq N(V_1 \cup V_2 \cup V_3)$, which is solved by Lemma 5. The main idea is to analyze the branching by the parameter $\alpha(R \setminus N(V_1 \cup V_2 \cup V_3))$, in particular with $\alpha(R \setminus N(V_1 \cup V_2 \cup V_3)) = 0$ corresponding to the base case. The branching itself will be "exact" in the sense that all four cases of Lemma 6 are always included, in particular, the 2-approximation is caused only by the application of Lemma 5.

First, while $\alpha(R \setminus N(V_1 \cup V_2 \cup V_3)) \geq 2k$, which can be checked in $n^{\mathcal{O}(k)}$ time, we branch as follows. We select an independent set $I \subseteq R \setminus N(V_1 \cup V_2 \cup V_3)$ of size $|I| = 2k$, and for all of its vertices we branch on whether to move it into V_1, V_2, V_3, or S_0, i.e., according to Lemma 6. Because I is an independent set, at most k of the vertices can go to S_0, so at least k go to V_1, V_2, or V_3. Also for the reason that I is an independent set, Lemma 7 can be successively applied for all of the vertices that go to V_1, V_2, or V_3. Therefore, this decreases $\alpha(R \setminus N(V_1 \cup V_2 \cup V_3))$ by at least k, so the depth of this recursion is at most $\alpha(R)/k$, so the total number of nodes in this branching tree is at most $(4^{2k})^{\alpha(R)/k} = 2^{\mathcal{O}(\alpha(R))}$.

Then, we can assume that $\alpha(R \setminus N(V_1 \cup V_2 \cup V_3)) < 2k$. We continue with a similar branching, this time branching on single vertices. In particular, as long as $R \setminus N(V_1 \cup V_2 \cup V_3)$ is nonempty, we select a vertex in it and branch on whether to move it into V_1, V_2, V_3, or S_0. To analyze the size of this branching tree, note that by Lemma 7, when moving a vertex into V_1, V_2, or V_3, the value of $\alpha(R \setminus N(V_1 \cup V_2 \cup V_3))$ decreases by at least one. Therefore, as initially $\alpha(R \setminus N(V_1 \cup V_2 \cup V_3)) < 2k$, any root-to-leaf path of the branching tree contains less than $2k$ edges that correspond to such branches, and therefore any root-leaf path can be characterized by specifying the $< 2k$ indices corresponding to such edges, and then for these indices whether they correspond to V_1, V_2, or V_3. The length of any root-leaf path is at most n because $|R \setminus N(V_1 \cup V_2 \cup V_3)|$ decreases in every branch, and therefore the number of different root-to-leaf paths is at most $n^{2k}3^{2k} = n^{\mathcal{O}(k)}$, and therefore the total number of nodes in this branching tree is $n^{\mathcal{O}(k)}$.

Therefore, the total size of both of the branching trees put together is $2^{\mathcal{O}(\alpha(R))}n^{\mathcal{O}(k)}$, so our algorithm works by $2^{\mathcal{O}(\alpha(R))}n^{\mathcal{O}(k)}$ applications of Lemma 5, resulting in a $2^{\mathcal{O}(\alpha(R))}n^{\mathcal{O}(k)}$-time algorithm. ◀

Finally, what is left is to reduce the general case to the case where $\alpha(R) = \mathcal{O}(k^2)$. We do not know if this can be done in general, but we manage to do it by using a tree decomposition with independence number $\mathcal{O}(k)$. To this end, we show the following lemma. Given a graph G, a tree decomposition $\mathcal{T}$ of G, and a set $W \subseteq V(G)$, we say that W is *covered* by a set $\mathcal{B}$ of bags of $\mathcal{T}$ if every vertex in W is contained in at least one of the bags in $\mathcal{B}$. The lemma generalizes the well-known fact that any clique W of G, that is, a set with $\alpha(W) = 1$, is covered by a single bag of the tree decomposition.

▶ **Lemma 9.** *Let G be a graph, $\mathcal{T}$ a tree decomposition of G, and W a nonempty set of vertices of G. Then W is covered by a set of at most $2\alpha(W) - 1$ bags of $\mathcal{T}$.*

Proof. We denote $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$. Every edge $ab \in E(T)$ corresponds to a partition $(X_a \cap X_b, C_a, C_b)$ of $V(G)$, where C_a is the set of vertices of $G \setminus (X_a \cap X_b)$ in the bags of $\mathcal{T}$ that are closer to a than b, C_b is the set of vertices of $G \setminus (X_a \cap X_b)$ in the bags of $\mathcal{T}$ that are closer to b than a, and $X_a \cap X_b$ is a (C_a, C_b)-separator.

First, assume that for every edge ab either $C_a \cap W = \emptyset$ or $C_b \cap W = \emptyset$. If both $C_a \cap W = C_b \cap W = \emptyset$, then $W \subseteq X_a \cap X_b$, and we cover W by a single bag X_a (or X_b). Thus we may assume that for every edge exactly one of the sets $C_a \cap W$ and $C_b \cap W$ is nonempty. We orient the edge ab from a towards b if $C_b \cap W \neq \emptyset$, and from b to a if $C_a \cap W \neq \emptyset$. Because T is a tree, there exists a node $t \in V(T)$ such that all edges incident with t are oriented towards t. This implies that X_t covers W because otherwise some edge would be oriented away from t.

Note that if $\alpha(W) = 1$, then W is a clique and indeed for every edge ab either $C_a \cap W = \emptyset$ or $C_b \cap W = \emptyset$. The remaining case is that $\alpha(W) \geq 2$ and that there exists an edge ab such that both $|C_a \cap W| \geq 1$ and $|C_b \cap W| \geq 1$ hold. In this case, we use induction on $\alpha(W)$. Since $X_a \cap X_b$ is a (C_a, C_b)-separator, we have that $\alpha(C_a \cap W) + \alpha(C_b \cap W) \leq \alpha(W)$. Therefore we can take the union of a smallest set of bags covering $C_a \cap W$, a smallest set of bags covering $C_b \cap W$, and the bag X_a. By induction, this set of bags covering W contains at most $2\alpha(W \cap C_a) - 1 + 2\alpha(W \cap C_b) - 1 + 1 \leq 2\alpha(W) - 1$ bags. ◀

With Lemma 9, we can use a tree decomposition $\mathcal{T}$ with independence number $\alpha(\mathcal{T}) = \mathcal{O}(k)$ to 2-approximate the general case of PARTIAL 3-WAY α-SEPARATOR as follows. By Lemma 9, any solution S with $\alpha(S) \leq k$ is covered by at most $2k - 1$ bags of $\mathcal{T}$. Therefore, with $\mathcal{T}$ available (and having $n^{\mathcal{O}(1)}$ bags by standard arguments), we can guess a set of at most $2k - 1$ bags of $\mathcal{T}$ that covers S, and intersect R by the union of these bags, resulting in $\alpha(R) \leq \alpha(\mathcal{T})(2k - 1) = \mathcal{O}(k^2)$. Therefore, we solve the general case by $n^{\mathcal{O}(k)}$ applications of Lemma 8 with $\alpha(R) = \mathcal{O}(k^2)$, resulting in a total time complexity of $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(k)}$. This completes the proof of Theorem 3.

4.2 From separators to balanced separators

In this subsection we show that Theorem 3 can be used to either find certain balanced separators for sets $W \subseteq V(G)$ or to show that the tree-independence number of the graph is large.

To enforce the "balance" condition, we cannot directly enforce that the separator separates a given set W into sets of small independence numbers. (This would result in time complexity exponential in $|W|$ instead of $\alpha(W)$.) Instead, we fix a maximum independent set in W,

and enforce that this independent set is separated in a balanced manner. As long as the independent set is large enough, this will enforce that the separator is balanced also with respect to α. The following lemma, which is an adaptation of a well-known lemma given in Graph Minors II [36] is the starting point of this approach.

▶ **Lemma 10.** *Let G be a graph with the tree-independence number at most k and $I \subseteq V(G)$ an independent set. Then there exists a partition (S, C_1, C_2, C_3) of $V(G)$ (where some C_i can be empty) such that S is a (C_1, C_2, C_3)-separator, $\alpha(S) \leq k$, and $|I \cap (C_i \cup C_j)| \geq |I|/2 - k$ for any pair $i, j \in \{1, 2, 3\}$ with $i \neq j$.*

Proof. Let $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ be a tree decomposition of G with $\alpha(\mathcal{T}) \leq k$. As in Lemma 9, we introduce the following notation. Every edge $ab \in E(T)$ of T corresponds to a partition $(X_a \cap X_b, C_a, C_b)$ of $V(G)$, where C_a is the set of vertices of $G \setminus (X_a \cap X_b)$ in the bags of $\mathcal{T}$ that are closer to a than b, C_b is the set of vertices of $G \setminus (X_a \cap X_b)$ in the bags of $\mathcal{T}$ that are closer to b than a, and $X_a \cap X_b$ is a (C_a, C_b)-separator. We orient the edge ab from a to b if $|C_b \cap I| > |I|/2$, from b to a if $|C_a \cap I| > |I|/2$, and otherwise arbitrarily. Now, because T is a tree, there exists a node $t \in V(T)$ such that all of its incident edges are oriented towards it. Therefore, for all connected components C of $G \setminus X_t$, we see that $|V(C) \cap I| \leq |I|/2$, as otherwise some edge would be oriented out of t.

As long as the number of such connected components C is at least 4, we can take two of them with the smallest values of $|V(C) \cap I|$ and merge them, in the end arriving at a partition (X_t, C_1, C_2, C_3) of $V(G)$ such that X_t is a (C_1, C_2, C_3)-separator, $\alpha(X_t) \leq k$, and $|I \cap C_i| \leq |I|/2$ for all $i \in \{1, 2, 3\}$. Then, because $|I \cap C_i| \leq |I|/2$, we have that $|I \cap (V(G) \setminus C_i)| \geq |I|/2$. Therefore, since $|I \cap X_t| \leq k$, it follows that $|I \cap (C_j \cup C_\ell)| \geq |I|/2 - k$, where (i, j, ℓ) is any permutation of the set $\{1, 2, 3\}$. ◀

Next we use Lemma 10 together with the separator algorithm of Theorem 3 to design an algorithm for finding α-balanced separators of sets W with $\alpha(W) = 6k$.

▶ **Lemma 11.** *There is an algorithm that for a given graph G, an integer k, a tree decomposition of G with independence number $\mathcal{O}(k)$, and a set $W \subseteq V(G)$ with $\alpha(W) = 6k$, in time $2^{\mathcal{O}(k^2)} n^{\mathcal{O}(k)}$ either concludes that the tree-independence number of G is larger than k, or finds a partition (S, C_1, C_2, C_3) of $V(G)$ such that S is a (C_1, C_2, C_3)-separator, $\alpha(S) \leq 2k$, at most one of C_1, C_2, and C_3 is empty, and $\alpha(W \cap C_i) \leq 4k$ for each $i \in \{1, 2, 3\}$.*

Proof. First, we take an arbitrary independent set $I \subseteq W$ of size $|I| = 6k$, which can be found in $n^{\mathcal{O}(k)}$ time. If the tree-independence number of G is at most k, then, by Lemma 10, there exists a partition (S, C_1, C_2, C_3) of $V(G)$ such that S is a (C_1, C_2, C_3)-separator, $\alpha(S) \leq k$, and $|I \cap (C_i \cup C_j)| \geq |I|/2 - k \geq 2k$ for any pair $i, j \in \{1, 2, 3\}$ with $i \neq j$. We guess the intersection of such a partition with I, in particular we guess the partition $(S \cap I, C_1 \cap I, C_2 \cap I, C_3 \cap I)$, immediately enforcing that it satisfies the constraints $|I \cap (C_i \cup C_j)| \geq 2k$ for $i \neq j$.

For each such guess, we use Theorem 3 to either find a $(C_1 \cap I, C_2 \cap I, C_3 \cap I)$-separator with independence number at most $2k$ or to decide that no such separator with independence number at most k exists. The set S of the partition guaranteed by Lemma 10 is indeed a $(C_1 \cap I, C_2 \cap I, C_3 \cap I)$-separator with independence number at most k, so if the algorithm reports for every guess that no such separator exists, we return that G has the independence number larger than k.

Otherwise, for some guess a $(C_1 \cap I, C_2 \cap I, C_3 \cap I)$-separator S' with $\alpha(S') \leq 2k$ is found, and we return the partition (S', C'_1, C'_2, C'_3), where C'_i is the union of the vertex sets of components of $G \setminus S'$ that contain a vertex of $C_i \cap I$, for all $i \in \{1, 2, 3\}$. Because

$|I \cap (C_i \cup C_j)| \geq 2k$ for $i \neq j$, we see that $\alpha(W \cap C_\ell) \leq \alpha(W) - 2k \leq 4k$, where (i, j, ℓ) is an arbitrary permutation of the set $\{1, 2, 3\}$, because otherwise we could use the union of $I \cap (C_i \cup C_j)$ and a maximum independent set in $W \cap C_\ell$ to construct an independent set in W of size larger than $\alpha(W)$. Also, because $|I \cap (C_i \cup C_j)| \geq 2k$ for any pair $i \neq j$, the set C_i cannot be empty for more than one i.

The algorithm works by first finding an independent set of size $6k$ and then using the algorithm of Theorem 3 at most 4^{6k} times, so the total time complexity is $\mathcal{O}(n^{6k}) + 4^{6k} \cdot 2^{\mathcal{O}(k^2)} n^{\mathcal{O}(k)} = 2^{\mathcal{O}(k^2)} n^{\mathcal{O}(k)}$. ◀

4.3 Constructing the decomposition

Everything is prepared for the final step of the proof – the algorithm constructing a tree decomposition with independence number at most $8k$ by using the balanced separator algorithm of Lemma 11. Our algorithm constructs a tree decomposition from root to the leaves by maintaining an "interface" W and breaking it with balanced separators. This is a common strategy used for various algorithms for constructing tree decompositions and branch decompositions. In our case, perhaps the largest hurdle in the proof is the analysis that the size of the recursion tree and the constructed decomposition is polynomial in n.

A rooted tree decomposition is a tree decomposition where one node is designated as the root.

▶ **Lemma 12.** *There is an algorithm that for a given graph G, an integer k, a tree decomposition of G with independence number $\mathcal{O}(k)$, and a set $W \subseteq V(G)$ with $\alpha(W) \leq 6k$, in time $2^{\mathcal{O}(k^2)} n^{\mathcal{O}(k)}$ either determines that the tree-independence number of G is larger than k or returns a rooted tree decomposition $\mathcal{T}$ of G with independence number at most $8k$ such that W is contained in the root bag of $\mathcal{T}$.*

Proof. The algorithm will be based on recursively constructing the decomposition, using W as the interface in the recursion. First, if $\alpha(G) \leq 6k$, we return the trivial tree decomposition with only one bag $V(G)$. Otherwise, we start by inserting arbitrary vertices of G into W until the condition $\alpha(W) = 6k$ holds.

Then, we apply the algorithm of Lemma 11 to find a partition (S, C_1, C_2, C_3) of $V(G)$ such that S is a (C_1, C_2, C_3)-separator, $\alpha(S) \leq 2k$, $\alpha(W \cap C_i) \leq 4k$ for each $i \in \{1, 2, 3\}$, and at most one of C_1, C_2, C_3 is empty, or to determine that the tree-independence number of G is larger than k, in this case returning no immediately. Then, we construct the tree decomposition recursively as follows: for each $i \in \{1, 2, 3\}$, we recursively use the algorithm with the graph $G_i = G[C_i \cup S]$ and the set $W_i = (C_i \cap W) \cup S$. Let $\mathcal{T}_1$, $\mathcal{T}_2$, $\mathcal{T}_3$ be the obtained tree decompositions and let r_1, r_2, r_3 be their root nodes. We create a new root node r with a bag $X_r = S \cup W$, and connect r_1, r_2, and r_3 as children of r.

Lemma 11 guarantees that $\alpha(C_i \cap W) \leq 4k$ and $\alpha(S) \leq 2k$, and therefore $\alpha(W_i) \leq 6k$. Also, $\alpha(S \cup W) \leq 8k$ because $\alpha(W) \leq 6k$ and $\alpha(S) \leq 2k$. Therefore, the independence number of the constructed tree decomposition is at most $8k$. The constructed tree decomposition satisfies all conditions of tree decompositions: Because S is a separator between C_1, C_2, and C_3, when recursing into the graphs $G_i = G[C_i \cup S]$ for $i \in \{1, 2, 3\}$, the union of the graphs G_1, G_2, and G_3 includes all vertices and edges of G. Therefore, by induction, every vertex and edge will be contained in some bag of the constructed tree decomposition (the base case is $\alpha(G) \leq 6k$). By induction, the decomposition satisfies also the connectivity condition: if a vertex occurs in G_i and G_j for $i \neq j$, then it is in S and therefore in the bag X_r and also in the sets W_i and W_j and therefore in the root bags X_{r_i} and X_{r_j} of $\mathcal{T}_i$ and $\mathcal{T}_j$.

It remains to argue that the size of the recursion tree (and equivalently the size of the decomposition constructed) is $n^{\mathcal{O}(1)}$. First, by the guarantee of Lemma 11 that C_i is empty for at most one $i \in \{1,2,3\}$, we have that each G_i has strictly fewer vertices than G, and therefore the constructed tree has height at most n. We say that a constructed node t is a forget node if its bag contains a vertex v so that its parent's bag does not contain v. The number of forget nodes is at most n because a vertex can be forgotten only once in a tree decomposition.

Recall that in the start of each recursive call, on a graph G_i and a subset W_i, we either recognize that $\alpha(G_i) \leq 6k$, creating a leaf node in this case, or add vertices to W_i until $\alpha(W_i) = 6k$. In the latter case, as these added vertices were not initially in W_i, they are not in the bag of the parent node, and therefore the node constructed in such a call will be a forget node if any such vertices are added. Therefore, the new node constructed can be a non-forget non-leaf node only if $\alpha(W_i) = 6k$ already for the initial input W_i. Then, we observe that $\alpha(W_i) = 6k$ can hold for the initial input W_i only if $\alpha(C_i \cap W) = 4k$ did hold for the corresponding component C_i of the parent and the corresponding set W. Therefore, as $\alpha(C_i \cap W) = 4k$ can hold for at most one $i \in \{1,2,3\}$, we have that any node can have at most one non-forget non-leaf child node.

It follows that non-forget non-leaf nodes can be decomposed into maximal paths going between a node and its ancestor, and these paths have a length at most n by the height of the tree. Each such path either starts at the root or its highest node is a child of a forget node. Thus, the number of maximal paths of non-forget non-leaf nodes is at most n, and therefore the number of non-forget non-leaf nodes is at most n^2. The number of leaf nodes is at most three times the number of non-leaf nodes, so the total number of nodes is $\mathcal{O}(n^2)$.

Therefore, the algorithm works by $\mathcal{O}(n^2)$ applications of the algorithm of Lemma 11, and therefore its time complexity is $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(k)}$. ◀

It remains to observe that by using iterative compression, we can satisfy the requirement of Lemma 12 to have a tree decomposition with independence number $\mathcal{O}(k)$ as an input (in particular, here the independence number will be at most $8k+1$), and therefore Lemma 12 implies Theorem 1.

In more detail, we order the vertices of G as $v_1, \ldots, v_n$, and iteratively compute tree decompositions with independence number at most $8k$ for induced subgraphs $G[\{v_1, \ldots, v_i\}]$, for increasing values of i. The iterative computation guarantees that when computing the tree decomposition for $G[\{v_1, \ldots, v_i\}]$, we have the tree decomposition for $G[\{v_1, \ldots, v_{i-1}\}]$ with independence number at most $8k$ available, which can be used to obtain a tree decomposition with independence number at most $8k+1$ of $G[\{v_1, \ldots, v_i\}]$ by adding v_i to each bag to be used as the input tree decomposition. More precisely, we use Lemma 12 to either determine in time $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(k)}$ that the tree-independence number of $G[\{v_1, \ldots, v_i\}]$ is larger than k or obtain a rooted tree decomposition $\mathcal{T}$ of $G[\{v_1, \ldots, v_i\}]$ with independence number at most $8k$. As the tree-independence number does not increase when taking induced subgraphs, if for some induced subgraph we conclude that the tree-independence number is larger than k, we can conclude the same holds also for G. Otherwise, after n steps we will have a rooted tree decomposition $\mathcal{T}$ of G with independence number at most $8k$.

5 Hardness of computing tree-independence number

In this section, we complement our main algorithmic result by complexity lower bounds. First, we use the W[1]-hardness of independent set approximation by Lin [30] and the Gap-ETH result of Chalermsook et al. [7] to prove the following theorem.

▶ **Theorem 13** (⋆). *For any constant $c \geq 1$, there is no algorithm running in $f(k) \cdot n^{\mathcal{O}(1)}$ time for a computable function $f(k)$ that, given an n-vertex graph and a positive integer k, can distinguish between the cases* tree-$\alpha(G) \leq k$ *and* tree-$\alpha(G) > ck$, *unless FPT = W[1]. Moreover, assuming* Gap-ETH, *for any computable function $g(k) \geq k$, there is no algorithm running in $f(k) \cdot n^{o(k)}$ time for a computable function $f(k)$ that, given an n-vertex graph and a positive integer k, can distinguish between the cases* tree-$\alpha(G) \leq k$ *and* tree-$\alpha(G) > g(k)$.

Theorem 13 implies that it is W[1]-hard to decide whether tree-$\alpha(G) \leq k$ for the parameterization by k. However, the problem is, in fact, harder. We prove that it is NP-complete to decide whether tree-$\alpha(G) \leq 4$.

▶ **Theorem 2** (⋆). *For every constant $k \geq 4$, it is NP-complete to decide whether* tree-$\alpha(G) \leq k$ *for a given graph G.*

Finally, we show that, for every fixed integer $k \geq 3$, deciding if two given vertices of a graph can be separated by removing a set of vertices that induces a graph with independence number at most k is NP-complete. To put this result in perspective, note that the case with $k = 1$ is polynomial since we can compute all clique cutsets in polynomial time using Tarjan's algorithm [39]. We leave open the case with $k = 2$.

▶ **Theorem 14** (⋆). *For every integer $k \geq 3$, it is NP-complete to decide, given a graph H and two distinct vertices $u, v \in V(H)$, if there exists a u,v-separator S such that $\alpha(H[S]) \leq k$.*

6 Conclusion

The main result of our paper is an algorithm that, given an n-vertex graph G and an integer k, in time $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(k)}$ either outputs a tree decomposition of G with independence number at most $8k$, or concludes that the tree-independence number of G is larger than k. This yields also the same result for computing the minor-matching hypertree-width of a graph [40]. Our results allow to solve in $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(k)}$ time a plethora of problems when the inputs are restricted to graphs of tree-independence number k or minor-matching hypertree-width k [15, 29, 40]. We now show that this result is tight in several aspects.

First, one could ask what is the most general width-parameter defined by a min-max formula over the bags of a tree decomposition (see, e.g. [2, 33]) that allows to solve problems like Maximum Independent Set in polynomial time when bounded? For parameters where the width of a bag depends only on the induced subgraph of the bag, this turns out to be tree-α. In particular, we recall that Maximum Independent Set is NP-hard on graphs with each edge subdivided twice, but such graphs admit a tree decomposition where one bag is a large independent set, and the induced subgraphs of the other bags are isomorphic to 4-vertex paths. It follows that if the width-measure of a bag is monotone, i.e., it does not increase when taking induced subgraphs, it must be unbounded whenever α is unbounded. In other words, if there would be a width parameter tree-λ defined as the minimum, over all tree decomposition, of the maximum of $\lambda(G[X_t])$ over the bags X_t of the tree decomposition, where λ is a monotone graph invariant, then either Maximum Independent Set is already NP-hard when λ is a constant, or the parameterization by tree-α is more general than the parameterization by tree-λ.

The width parameter tree-μ, the minor-matching hypertree-width, escapes this argument because it does not only depend on the subgraphs induced by the bags, but also on the neighborhoods of the bags. In particular, for tree-μ the width of a bag X_t is defined as the maximum cardinality of an induced matching in G whose every edge intersects X_t. A

similar example shows that this type of parameters where the width of a bag X_t depends on $G[N[X_t]]$ cannot be generalized much more: If we start from Maximum Independent Set on cubic graphs and subdivide each edge four times, we obtain graphs where Maximum Independent Set is NP-hard, but that admit tree decompositions that contain one large bag X_t such that every connected component of $G[N[X_t]]$ is a 3-vertex path, while for all other bags X_t their closed neighborhood $N[X_t]$ has bounded size.

Further, we remind that our main result is computationally tight. In particular, in Theorem 13, we proved that it is unlikely that there is a $g(k)$-approximation algorithm for the tree-independence number with running time $f(k)n^{o(k)}$, for any computable function g. This shows that the $n^{\Omega(k)}$-factor in the running time is unavoidable up to some reasonable complexity assumptions. For exact computation of the tree-independence number, we proved in Theorem 2 that it is NP-complete to decide whether tree-$\alpha(G) \leq k$ for every constant $k \geq 4$. Since tree-$\alpha(G) = 1$ if and only if G is a chordal graph, Theorem 2 leads to the question about the complexity of deciding whether the tree-independence number is at most k for $k = 2$ and 3. In Theorem 14, we demonstrated that for every fixed integer $k \geq 3$, deciding if two given vertices of a graph can be separated by removing a set of vertices that induces a graph with independence number at most k is NP-complete. This result indicates that it may be already NP-complete to decide whether tree-$\alpha(G) \leq 3$. We hesitate to state any conjecture for the case $k = 2$.

The final question is about the place of computing the tree-independence number in the polynomial hierarchy. For a fixed k, deciding whether tree-$\alpha(G) \leq k$ is in NP. However, when k is a part of the input, the problem is naturally placed in the class Σ_2^P on the second level of the polynomial hierarchy. Is the problem Σ_2^P-complete?

References

1 Tara Abrishami, Bogdan Alecu, Maria Chudnovsky, Sepehr Hajebi, Sophie Spirkl, and Kristina Vušković. Tree independence number i. (even hole, diamond, pyramid)-free graphs, 2024. arXiv:2305.16258.

2 Isolde Adler. *Width functions for hypertree decompositions*. PhD thesis, Albert-Ludwigs-Universität Freiburg im Breisgau, 2006.

3 Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. Assoc. Comput. Mach.*, 41(1):153–180, 1994.

4 M. Bíró, M. Hujter, and Zs. Tuza. Precoloring extension. I. Interval graphs. *Discrete Math.*, 100(1-3):267–279, 1992. doi:10.1016/0012-365X(92)90646-W.

5 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016. doi:10.1137/130947374.

6 Hans L. Bodlaender, Jens Gustedt, and Jan Arne Telle. Linear-time register allocation for a fixed number of registers. In Howard J. Karloff, editor, *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 25-27 January 1998, San Francisco, California, USA*, pages 574–583. ACM/SIAM, 1998. URL: http://dl.acm.org/citation.cfm?id=314613.314994.

7 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From gap-exponential time hypothesis to fixed parameter tractable inapproximability: Clique, dominating set, and more. *SIAM J. Comput.*, 49(4):772–810, 2020. doi:10.1137/18M1166869.

8 Steven Chaplick, Fedor V. Fomin, Petr A. Golovach, Dušan Knop, and Peter Zeman. Kernelization of graph Hamiltonicity: proper H-graphs. *SIAM J. Discrete Math.*, 35(2):840–892, 2021. doi:10.1137/19M1299001.

9 Steven Chaplick, Martin Töpfer, Jan Voborník, and Peter Zeman. On H-topological intersection graphs. *Algorithmica*, 83(11):3281–3318, 2021. doi:10.1007/s00453-021-00846-3.

10 Steven Chaplick and Peter Zeman. Combinatorial problems on H-graphs. *Electron. Notes Discret. Math.*, 61:223–229, 2017. doi:10.1016/j.endm.2017.06.042.

11 Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discret. Math.*, 86(1-3):165–177, 1990. doi:10.1016/0012-365X(90)90358-O.

12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

13 Clément Dallard, Fedor V. Fomin, Petr A. Golovach, Tuukka Korhonen, and Martin Milanič. Computing tree decompositions with small independence number. *CoRR*, abs/2207.09993, 2022. arXiv:2207.09993.

14 Clément Dallard, Martin Milanič, and Kenny Štorgel. Treewidth versus clique number in graph classes with a forbidden structure. In Isolde Adler and Haiko Müller, editors, *Graph-Theoretic Concepts in Computer Science – 46th International Workshop, WG 2020, Leeds, UK, June 24-26, 2020, Revised Selected Papers*, volume 12301 of *Lecture Notes in Computer Science*, pages 92–105. Springer, 2020. doi:10.1007/978-3-030-60440-0_8.

15 Clément Dallard, Martin Milanič, and Kenny Štorgel. Treewidth versus clique number. II. tree-independence number. *J. Comb. Theory, Ser. B*, 164:404–442, 2024. doi:10.1016/J.JCTB.2023.10.006.

16 Clément Dallard, Martin Milanič, and Kenny Štorgel. Treewidth versus clique number. III. Tree-independence number of graphs with a forbidden structure. *Journal of Combinatorial Theory, Series B*, 167:338–391, 2024. doi:10.1016/j.jctb.2024.03.005.

17 Clément Dallard, Martin Milanič, and Kenny Štorgel. Treewidth versus clique number. I. Graph classes with a forbidden structure. *SIAM J. Discrete Math.*, 35(4):2618–2646, 2021. doi:10.1137/20M1352119.

18 Mark de Berg, Hans L. Bodlaender, Sándor Kisfaludi-Bak, Dániel Marx, and Tom C. van der Zanden. A framework for exponential-time-hypothesis-tight algorithms and lower bounds in geometric intersection graphs. *SIAM J. Comput.*, 49(6):1291–1331, 2020. doi:10.1137/20M1320870.

19 Erik D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on graphs of bounded genus and H-minor-free graphs. *Journal of the ACM*, 52(6):866–893, 2005.

20 Irit Dinur. Mildly exponential reduction from gap 3SAT to polynomial-gap label-cover. *Electron. Colloquium Comput. Complex.*, TR16-128, 2016. URL: https://eccc.weizmann.ac.il/report/2016/128, arXiv:TR16-128.

21 Fedor V. Fomin and Petr A. Golovach. Subexponential parameterized algorithms and kernelization on almost chordal graphs. *Algorithmica*, 83(7):2170–2214, 2021. doi:10.1007/s00453-021-00822-x.

22 Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond. On the tractability of optimization problems on H-graphs. *Algorithmica*, 82(9):2432–2473, 2020. doi:10.1007/s00453-020-00692-9.

23 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Excluded grid minors and efficient polynomial-time approximation schemes. *J. ACM*, 65(2):10:1–10:44, 2018. doi:10.1145/3154833.

24 Esther Galby, Andrea Munaro, and Shizhou Yang. Polynomial-time approximation schemes for independent packing problems on fractionally tree-independence-number-fragile graphs. In Erin W. Chambers and Joachim Gudmundsson, editors, *39th International Symposium on Computational Geometry, SoCG 2023, June 12-15, 2023, Dallas, Texas, USA*, volume 258 of *LIPIcs*, pages 34:1–34:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.SoCG.2023.34.

25 Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

26 Martin Grohe. Local tree-width, excluded minors, and approximation algorithms. *Combinatorica*, 23(4):613–632, 2003. doi:10.1007/s00493-003-0037-9.

27 Petr Hlinený and Jan Kratochvíl. Representing graphs by disks and balls (a survey of recognition-complexity results). *Discret. Math.*, 229(1-3):101–124, 2001. doi:10.1016/S0012-365X(00)00204-1.

28 Ashwin Jacob, Fahad Panolan, Venkatesh Raman, and Vibha Sahlot. Structural parameterizations with modulator oblivion. *Algorithmica*, 84(8):2335–2357, 2022. doi:10.1007/s00453-022-00971-7.

29 Paloma T. Lima, Martin Milanič, Peter Muršič, Karolina Okrasa, Paweł Rzążewski, and Kenny Štorgel. Tree decompositions meet induced matchings: beyond Max Weight Independent Set. arXiv:2402.15834, 2024. doi:10.48550/arXiv.2402.15834.

30 Bingkai Lin. Constant approximating k-clique is W[1]-hard. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1749–1756. ACM, 2021. doi:10.1145/3406325.3451016.

31 Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, Jie Xue, and Meirav Zehavi. Subexponential parameterized algorithms on disk graphs (extended abstract). In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2005–2031. SIAM, 2022. doi:10.1137/1.9781611977073.80.

32 Pasin Manurangsi and Prasad Raghavendra. A birthday repetition theorem and complexity of approximating dense CSPs. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 78:1–78:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.ICALP.2017.78.

33 Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60(6):Art. 42, 51, 2013. doi:10.1145/2535926.

34 Michal Pilipczuk. Computing tree decompositions. In Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen, editors, *Treewidth, Kernels, and Algorithms – Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *Lecture Notes in Computer Science*, pages 189–213. Springer, 2020. doi:10.1007/978-3-030-42071-0_14.

35 Vijay Raghavan and Jeremy P. Spinrad. Robust algorithms for restricted domains. *J. Algorithms*, 48(1):160–172, 2003. doi:10.1016/S0196-6774(03)00048-8.

36 Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

37 Neil Robertson and Paul D. Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory. Series B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

38 Konstantin Skodinis. Efficient analysis of graphs with small minimal separators. In Peter Widmayer, Gabriele Neyer, and Stephan J. Eidenbenz, editors, *Graph-Theoretic Concepts in Computer Science, 25th International Workshop, WG '99, Ascona, Switzerland, June 17-19, 1999, Proceedings*, volume 1665 of *Lecture Notes in Computer Science*, pages 155–166. Springer, 1999. doi:10.1007/3-540-46784-X_16.

39 Robert E. Tarjan. Decomposition by clique separators. *Discrete Math.*, 55(2):221–232, 1985. doi:10.1016/0012-365X(85)90051-2.

40 Nikola Yolov. Minor-matching hypertree width. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 219–233. SIAM, 2018. doi:10.1137/1.9781611975031.16.

Simultaneously Approximating All ℓ_p-Norms in Correlation Clustering

Sami Davies
Department of EECS and Simons Institute, University of California at Berkeley, CA, USA

Benjamin Moseley
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA

Heather Newman
Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

This paper considers correlation clustering on unweighted complete graphs. We give a combinatorial algorithm that returns a *single* clustering solution that is *simultaneously* $O(1)$-approximate for all ℓ_p-norms of the disagreement vector; in other words, a combinatorial $O(1)$-approximation of the *all-norms* objective for correlation clustering. This is the first proof that minimal sacrifice is needed in order to optimize different norms of the disagreement vector. In addition, our algorithm is the first combinatorial approximation algorithm for the ℓ_2-norm objective, and more generally the first combinatorial algorithm for the ℓ_p-norm objective when $1 < p < \infty$. It is also faster than all previous algorithms that minimize the ℓ_p-norm of the disagreement vector, with run-time $O(n^\omega)$, where $O(n^\omega)$ is the time for matrix multiplication on $n \times n$ matrices. When the maximum positive degree in the graph is at most Δ, this can be improved to a run-time of $O(n\Delta^2 \log n)$.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Approximation algorithms, correlation clustering, all-norms, lp-norms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.52

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2308.01534` [20]

Funding *Sami Davies*: Supported by an NSF Computing Innovation Fellowship while at Northwestern University.
Benjamin Moseley: Supported in part by a Google Research Award, Inform Research Award, Carnegie Bosch Junior Faculty Chair, NSF grants CCF-2121744 and CCF-1845146, and ONR Award N000142212702.
Heather Newman: Supported in part by a Google Research Award, Inform Research Award, Carnegie Bosch Junior Faculty Chair, NSF grants CCF-2121744 and CCF-1845146, and ONR Award N000142212702.

1 Introduction

Correlation clustering is one of the most prominent problems in clustering, as it cleanly models community detection problems [38, 36] and provides a way to decompose complex network structures [39, 32]. The input to the unweighted correlation clustering problem is a complete graph $G = (V, E)$, where $|V| = n$ and each edge $e \in E$ is labeled positive $(+)$ or negative $(-)$. If the edge (u, v) is positive, this indicates that u and v are similar, and analogously if the edge (u, v) is negative, this indicates that u and v are dissimilar. The output of the problem is a partition of the vertex set into parts $C_1, C_2, \ldots$, where each part represents a cluster.

The output should cluster similar vertices together and separate dissimilar vertices. Specifically, for a fixed clustering (i.e., partition of the vertices), a positive edge (u, v) is a *disagreement* with respect to the clustering if u and v are in different clusters and an *agreement* if u and v are in the same cluster. Similarly, a negative edge (u, v) is a disagreement with respect to the clustering if u and v are in the same cluster and an agreement if u and v are in different clusters. The goal is to find a clustering that minimizes some objective that is a function of the disagreements.[1] For example, the most commonly studied objective minimizes the total number of disagreements.

As an easy example to illustrate the problem, consider a social network. Every pair of people has an edge between them, and the edge is positive if the two people have ever met before, and negative otherwise. The goal of correlation clustering translates to partitioning all the people into clusters so that people are in the same cluster as their friends/acquaintances and in different clusters than strangers. The difficulty in constructing a clustering is that the labels may not be consistent, making disagreements unavoidable. Consider in the social network what happens when there is one person with two friends who have never met each other (u, v, w with (u, v) and (u, w) positive but (v, w) negative). The choice of objective matters in determining the best clustering.

For a given clustering $\mathcal{C}$, let $y_{\mathcal{C}}(u)$ denote the number of edges incident to u that are disagreements with respect to $\mathcal{C}$ (we drop $\mathcal{C}$ and write y when it is clear from context). The most commonly considered objectives are $\|y_{\mathcal{C}}\|_p = \sqrt[p]{\sum_{u \in V} y_{\mathcal{C}}(u)^p}$ for $p \in \mathbb{R}_{\geq 1} \cup \{\infty\}$, the ℓ_p-norms of the *disagreement vector* y. Note that the optimal objective values may drastically vary for different norms too. (For instance, in the example in Appendix A of [35], $V = A \sqcup B$[2], where $|A| = |B| = n/2$, and all edges are positive except for a negative matching between A and B. The optimal ℓ_∞-norm objective value is 1 whereas the optimal for ℓ_1 is $\Theta(n)$.) When $p = 1$, this objective minimizes the total number of disagreements. Setting $p = \infty$ minimizes the maximum number of disagreements incident to any node, ensuring a type of worst-case fairness.[3] Balancing these two extremes – average welfare on one hand and fairness on the other – is the ℓ_2-norm, which minimizes the variance of the disagreements at each node.

Correlation clustering was proposed by Bansal, Blum, and Chawla [7] with the objective of minimizing the ℓ_1-norm of the disagreement vector. The problem is NP-hard and several approximation algorithms have been proposed [7, 4, 16, 18]. Puleo and Milenkovic [35] proposed studying ℓ_p-norms of the disagreement vector for $p > 1$, and they give a 48-approximation for any fixed p. Charikar, Gupta, and Schwartz [15] introduced an improved 7-approximation, which Kalhan, Makarychev, and Zhou [29] further improved to a 5-approximation. When $p > 1$, up until recently, the only strategies were LP or SDP rounding, and it has been of interest to develop fast combinatorial algorithms [37]. Davies, Moseley, and Newman [19] introduced a combinatorial $O(1)$-approximation algorithm for $p = \infty$ (see also [25] for a different combinatorial algorithm), and leave open the question of discovering a combinatorial $O(1)$-approximation algorithm for $1 < p < \infty$.

In all prior work, solutions obtained for ℓ_p-norms are tailored to each norm (i.e., p is part of the input to the algorithm), and it was not well-understood what the trade-offs were between solutions that optimize different norms. Solutions naively optimizing one norm can be arbitrarily bad for other norms (see Figure 1). A natural question is whether this loss from using a solution to one objective for another is avoidable. More specifically:

[1] Note that the sizes and number of clusters are unspecified.

[2] $\sqcup$ denotes disjoint union.

[3] In the social network example, minimizing the ℓ_1-norm corresponds to finding a clustering that minimizes the total number of friends who are separated plus the total number of strangers who are in the same cluster. The ℓ_∞-norm corresponds to finding a clustering minimizing the number of friends any person is separated from plus the number of strangers in that person's same cluster.

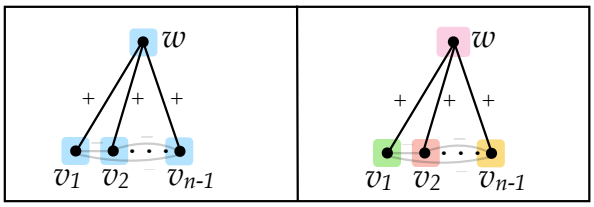


Figure 1 Two clusterings of the star graph, which has one node (w) with positive edges to all nodes, and the rest of the edges negative. **Left:** Clustering assigns all nodes to one (blue) cluster, and is (almost) optimal for the ℓ_∞-norm with cost $\Theta(n)$. **Right:** Clustering assigns all nodes to different clusters and is (almost) optimal for the ℓ_1-norm with cost $\Theta(n)$. The left solution is terrible for the ℓ_1-norm, as the negative clique has $\Theta(n^2)$ edges that are disagreements.

For any graph input to unweighted, complete correlation clustering, does there exist a partition (clustering) that is ***simultaneously*** $O(1)$*-approximate for all* ℓ_p*-norm objectives?*

Phrased another way, does there exist a *universal algorithm* for ℓ_p-norm correlation clustering – one which is guaranteed to produce a solution that well-approximates many objectives at once? When the goal is to simultaneously optimize every ℓ_p-norm, this is known as the *all-norms* objective.[4] Universal algorithms and the all-norms objective are well-studied in combinatorial optimization problems, such as load balancing and set cover (see Section 1.2 for more discussion). In the context of correlation clustering, such an algorithm outputs a partition that has good global performance (i.e. ℓ_1-norm) and also has no individual node with too many adjacent disagreements (i.e. ℓ_∞-norm). Universal algorithms exist for some problems and are provably impossible for others. The question looms, what can be said about universal algorithms for correlation clustering?

As far as we are aware, there are no known results for the all-norms objective in other clustering problems. In fact, for the popular k-median and k-center problems, it is actually *impossible* to $O(1)$-approximate (or even $o(\sqrt{n})$-approximate) these two objectives simultaneously [5].

1.1 Results

This paper is focused on optimizing all ℓ_p-norms ($p \geq 1$) for correlation clustering at the same time. The main result of the paper answers the previous question positively: perhaps surprisingly, there is a single clustering that simultaneously $O(1)$-approximates the optimal for all ℓ_p-norms. Further, it can be found through an *efficient combinatorial algorithm*. This is also the first known combinatorial approximation algorithm for the ℓ_2-norm objective and more generally ℓ_p-norm objective for fixed $2 \leq p < \infty$.

In what follows, let $O(n^\omega)$ denote the run-time of $n \times n$ matrix multiplication.

▶ **Theorem 1.** *Let $G = (V, E)$ be an instance of unweighted, complete correlation clustering on $|V| = n$ nodes. There exists a combinatorial algorithm returning a single clustering that is simultaneously an $O(1)$-approximation[5] for all ℓ_p-norm objectives, for all $p \in \mathbb{R}_{\geq 1} \cup \{\infty\}$, and its run-time is $O(n^\omega)$.*

[4] In some of the literature, for instance that of Golovin et al. [24], it is called the *all-ℓ_p-norms* objective.
[5] Note this is independent of p.

The algorithm gives the *fastest run-time* of any $O(1)$-approximation algorithm for the ℓ_p-norm objective when $p \in \mathbb{R}_{>1}$. Further, the run-time can be improved when the positive degree of the graph is bounded, as shown in the following corollary.

▶ **Corollary 2.** *Let Δ denote the maximum positive degree in an instance $G = (V, E)$ of unweighted, complete correlation clustering on $|V| = n$ nodes. Suppose G is given as an adjacency list representation of its positive edges. There exists a combinatorial algorithm returning a single clustering that is simultaneously an $O(1)$-approximation for all ℓ_p-norm objectives, for all $p \in \mathbb{R}_{\geq 1} \cup \{\infty\}$, and its run-time is $O(n\Delta^2 \log n)$.*

The run-time of the algorithm matches the fastest known algorithm for the ℓ_∞-norm objective [19], in both the general case and when the maximum positive degree is bounded. The best-known algorithm before our work relied on solving a convex relaxation on $|V|^2$ variables and $|V|^3$ constraints. We improve the run-time by avoiding this bottleneck.

In the setting when the positive edges form a regular graph, the interested reader may also find a clean proof (which is much simpler than that of Theorem 1) in Section 3 showing there is a solution that is simultaneously $O(1)$-approximate for the ℓ_1-norm and ℓ_∞-norm objectives.

1.2 Related work

Correlation clustering was introduced by Bansal, Blum, and Chawla [7]. The version they introduced also studies the problem on unweighted, complete graphs, but is concerned with minimizing the ℓ_1-norm of the disagreement vector. For this problem, Ailon, Charikar, and Newman [4] designed the Pivot algorithm, which is a randomized algorithm that in expectation obtains a 3-approximation. While we know algorithms with better approximations for ℓ_1 correlation clustering than Pivot [16, 18], the algorithm remains a baseline in correlation clustering due to its simplicity. (However, Pivot can perform arbitrarily badly – i.e., give $\Omega(n)$ approximation ratios – for other ℓ_p-norms; see again the example in Appendix A of [35].) It is an active area of research to develop algorithms for the ℓ_1-norm that focus on practical scalability [11, 17, 33, 36, 14]. Correlation clustering has also been studied on non-complete, weighted graphs [15, 29], with conditions on the cluster sizes [34], and with asymmetric errors [26]. In fact, in recent work Veldt [37] highlighted the need for deterministic techniques in correlation clustering that do not use linear programming. Much interest in correlation clustering stems from its connections to applications, including community detection, natural language processing, location area planning, and gene expression [38, 36, 39, 32, 9, 21].

Puleo and Milenkovic [35] introduced correlation clustering with the goal of minimizing the ℓ_p-norm of the disagreement vector. They show that even for minimizing the ℓ_∞-norm on complete, unweighted graphs, the problem is NP-hard (Appendix C in [35]). Several groups found $O(1)$-approximation algorithms for minimizing the ℓ_p-norm on complete, unweighted graphs [35, 15, 29], the best of which is currently the 5-approximation of Kalhan, Makarychev, and Zhou [29]. Many other interesting objectives for correlation clustering focus on finding solutions that are (in some sense) fair or locally desirable [3, 8, 1, 22, 27, 2]. All of these previous works that study general ℓ_p-norms or other notions of fairness or locality rely on solving a convex relaxation. This has two downsides: (1) the run-time of the algorithms are bottle-necked by the time it takes to solve the relaxation with at least $\Omega(n^2)$ many variables and $\Omega(n^3)$ constraints; in fact, it is time-consuming to even enumerate the $\Omega(n^2)$ variables and $\Omega(n^3)$ constraints; and (2) the solution is only guaranteed to be good for one particular value of p.

Several problems have been studied with the goal of finding a solution that is a good approximation for several objectives simultaneously. The all-norms objective was introduced by Azar et al. [6], where the goal is to design a ρ-approximation algorithm for all ℓ_p-norm objectives of a problem. They originally introduced the objective for the restricted assignment load balancing problem and showed an all-norms 2-approximation. Further follow-up on the all-norms objective has been done for load balancing [30, 10, 31], and for set cover [24]. The term "universal" algorithm has also been used for Steiner tree [12, 13], TSP [28], and clustering [23], though in these settings the goal is different, namely, to find a solution that is good for any potential *input*; e.g., in Universal Steiner Tree, the goal is to find a spanning tree where for any set of terminals, the sub-tree connecting the root to the terminals is a good approximation of the optimal.

2 Preliminaries

We will introduce notation, and then we will discuss two relevant works – the papers by Kalhan, Makarychev, and Zhou [29] and Davies, Moseley, and Newman [19].

2.1 Notation

Recall our input to the correlation clustering problem is $G = (V, E)$, an unweighted, complete graph on n vertices, and every edge is assigned a label of either positive $(+)$ or negative $(-)$. Let the set of positive edges be denoted E^+ and the set of negative edges E^-. Then, we can define the *positive neighborhood* and *negative neighborhood* of a vertex u as $N_u^+ = \{v \in V \mid (u, v) \in E^+\}$ and $N_u^- = \{v \in V \mid (u, v) \in E^-\}$, respectively. We further assume without loss of generality that every vertex has a positive self-loop to itself.

A *clustering* $\mathcal{C}$ is a partition of V into *clusters* $C_1, \ldots, C_k$ (but recall that k is *not* pre-specified). Let $C(u)$ denote the cluster that vertex u is in, i.e., if $\mathcal{C}$ has k clusters, there exists exactly one $i \in [k]$ such that $C(u) = C_i$. It is also helpful to consider the vertices in a different cluster than u, and so we let $\overline{C(u)} = V \setminus C(u)$ denote this. We say that a positive edge $e = (u, v) \in E^+$ is a *disagreement* with respect to $\mathcal{C}$ if $v \in \overline{C(u)}$. On the other hand, we say that a negative edge $e = (u, v) \in E^-$ is a disagreement with respect to $\mathcal{C}$ if $v \in C(u)$. For a fixed clustering $\mathcal{C}$, we denote the *disagreement vector* of $\mathcal{C}$ as $y_{\mathcal{C}} \in \mathbb{Z}^n_{\geq 0}$, where for $u \in V$, $y_{\mathcal{C}}(u)$ is the number of edges incident to u that are disagreements with respect to $\mathcal{C}$. We omit the subscript throughout the proofs when a clustering is clear.

Throughout, we let OPT be the optimal objective value, and the ℓ_p-norm to which it corresponds will be clear from context. The next fact follows from the definitions seen so far (recalling also the positive self-loops).

▶ **Fact 3.** *For any $u, v \in V$, $n = |N_u^+ \cap N_v^+| + |N_u^- \cap N_v^-| + |N_u^+ \cap N_v^-| + |N_u^- \cap N_v^+|$.*

2.2 Summary of work by Kalhan, Makarychev, and Zhou

The standard linear program relaxation for correlation clustering is given in (P) below.[6] In the *integer* LP, the variable x_{uv} indicates whether vertices u and v will be in the same cluster (0 for yes, 1 for no), and the disagreement vector is y; the optimal solution to the integer LP has value OPT, while the optimal solution to the relaxation gives a lower bound on OPT. Note the triangle inequality is enforced on all triples of vertices, inducing a semi-metric space

[6] Technically this is a convex program as the objective is convex; we say LP as the constraints are linear.

on V. Throughout this paper, as in [19], we refer to the algorithm by Kalhan, Makarychev, and Zhou as the KMZ algorithm. The *KMZ algorithm* has two phases: it solves (P), and then uses the *KMZ rounding algorithm* to obtain an integral assignment of vertices to clusters. At a high-level, the KMZ rounding algorithm is an iterative, ball-growing algorithm that uses the semi-metric to guide its choices. Their algorithm is a 5-approximation, and produces different clusterings for different p, since the optimal solution x^* to (P) depends on p.

$$\begin{aligned} &\min \|y\|_p \\ \text{s.t. } y_u &= \sum_{v \in N_u^+} x_{uv} + \sum_{v \in N_u^-} (1 - x_{uv}) && \forall u \in V \\ x_{uv} &\leq x_{uw} + x_{vw} && \forall u, v, w \in V \\ 0 &\leq x_{uv} \leq 1 && \forall u, v \in V. \end{aligned} \tag{P}$$

▶ **Definition 4.** *Let f be a semi-metric on V, i.e., taking $x = f$ gives a feasible solution to (P). The* fractional cost of f in the ℓ_p-norm objective *is the value of (P) that results from setting $x = f$. When p is clear from context, we will simply call this the fractional cost of f.*

2.3 Summary of work by Davies, Moseley, and Newman

The main take-away from the work of Kalhan, Makarychev, and Zhou [29] is that one only requires a semi-metric on the set of vertices, whose cost is comparable to the cost of an optimal solution, as input to the KMZ rounding algorithm. Thus, the insight of Davies, Moseley, and Newman [19] for the ℓ_∞-norm objective is that one can combinatorially construct such a semi-metric without solving an LP, and at small loss in the quality of the fractional solution. They do this by introducing the *correlation metric*.

▶ **Definition 5** ([19])**.** *For all $u, v \in V$, the correlation metric defines the distance between u and v as*

$$d_{uv} = 1 - \frac{|N_u^+ \cap N_v^+|}{|N_u^+ \cup N_v^+|} = \frac{|N_u^+ \cap N_v^-| + |N_u^- \cap N_v^+|}{|N_u^+ \cap N_v^+| + |N_u^+ \cap N_v^-| + |N_u^- \cap N_v^+|}.$$

Note that the rewrite in the second equality is apparent from Fact 3.

The correlation metric captures useful information succinctly. Intuitively, if u and v have relatively large positive intersection, i.e., $N_u^+ \cap N_v^+$ is large compared to their other relevant joint neighborhoods $(N_u^+ \cap N_v^-) \cup (N_u^- \cap N_v^+)$, then from the perspective of u and v, fewer disagreements are incurred by putting u and v in the same cluster than by putting them in different clusters. This is because if u and v are in the same cluster, then they have disagreements on edges (u, w) and (v, w) for $w \in (N_u^+ \cap N_v^-) \cup (N_u^- \cap N_v^+)$, but if they are in different clusters, then u and v have disagreements on edges (u, w) and (v, w) for $w \in N_u^+ \cap N_v^+$. For more on intuition behind the correlation metric, see Section 2 in [19].

Davies, Moseley, and Newman [19] prove that the correlation metric d can be used as input to the KMZ rounding algorithm by showing that (1) d satisfies the triangle inequality and (2) the fractional cost of d in the ℓ_∞-norm (recall Definition 4) is no more than 8 times the value of the optimal integral solution (OPT). Since the KMZ rounding algorithm loses a factor of at most 5, inputting d to that algorithm returns a 40-approximation algorithm. A benefit of the correlation metric is that it can be computed in time $O(n^\omega)$, and even faster when the subgraph on positive edges is sparse.

2.4 Technical overview

It is not hard to see that the correlation metric cannot be used as input to the KMZ algorithm for ℓ_p-norms other than $p = \infty$, as one cannot bound the fractional cost of the correlation metric against the optimal with only an $O(1)$-factor loss. To see why, consider the star again, as in Figure 1. Here, for all $u, v \in \{v_1, \ldots, v_{n-1}\}$, $d_{uv} = 1 - 1/(n - (n-3)) = 2/3$, but for the ℓ_1-norm, we need the semi-metric to have the value $1 - d_{uv}$ be close to 0, i.e. $O(1/n)$, for such u, v, in order for the fractional cost to be comparable to the value of OPT for $p = 1$.

There are several possible fixes one could try to make to the correlation metric. One idea is that since one can interpret the correlation metric as a coarse approximation of the probability the Pivot algorithm[7] separates u and v, one could try to adapt the correlation metric to more accurately approximate this probability.[8] Another idea, inspired by an observation below, is that one could define a semi-metric for edges in E^+ and another semi-metric for edges in E^-, but then there is the difficulty of showing the triangle inequality holds when positive and negative edges are mixed. Both of these ideas were, for us, unsuccessful.

Instead, the following two observations of how the correlation metric works with respect to the ℓ_1-norm led us to an effective adaptation:

1. One can bound the fractional cost, *restricted to positive edges*, of the correlation metric in the ℓ_1-norm by an $O(1)$-factor times the optimal solution's cost (see Claim 1 in Appendix C of the full version [20]). Negative edges still pose a challenge.
2. If the subgraph of positive edges is regular, then we can actually bound the fractional cost of the correlation metric in the ℓ_1-norm on negative edges as well.[9] See Section 3.

These observations led us to ask whether some adjustments to the correlation metric might yield a semi-metric with bounded fractional cost in the ℓ_1-norm or even the ℓ_p-norm more generally (while still remaining bounded in the ℓ_∞-norm). Moreover, since the KMZ rounding algorithm does not depend on p (whereas in the KMZ algorithm, the solution to the LP *does* depend on p), inputting the same semi-metric to the rounding algorithm produces the same clustering for all ℓ_p-norms!

We are ready to define the *adjusted correlation metric*. Let Δ_u denote the positive degree of u (the degree of u in the subgraph of positive edges).

▶ **Definition 6.** *Define the* adjusted correlation metric $f : E \to [0, 1]$ *as follows:*

1. *For d the correlation metric, i.e., $d_{uv} = 1 - \frac{|N_u^+ \cap N_v^+|}{|N_u^+ \cup N_v^+|}$, initially set $f = d$.*
2. *If $e \in E^-$ and $d_e > 0.7$, set $f_e = 1$ (round up).*
3. *For $u \in V$ such that $|N_u^- \cap \{v : d_{uv} \leq 0.7\}| \geq \frac{10}{3}\Delta_u$, set $f_{uv} = 1$ for all $v \in V \setminus \{u\}$.*

The idea in Step 3 is that if the fractional cost of negative edges incident to u is sufficiently large, we instead trade this for the cost of positive disagreements, as the rounding algorithm will now put u in its own cluster. For the ℓ_∞-norm, this trade-off is innocuous. For ℓ_p-norms in general, a refined charging argument is needed to show that post-processing d in this way sufficiently curbs the (too large) fractional cost of d.

In Section 3, we start with a warm-up exercise and show that if the graph on positive edges is regular, then the (original) correlation metric d has $O(1)$-approximate fractional cost. Note this section is not necessary to understanding the rest of the paper, but we

[7] Pivot operates as follows: Choose a random unclustered $u \in V$. Take u and all its unclustered positive neighbors and let this be the newest cluster. Continue until all vertices in V are clustered.

[8] If in (P), x_{uv} is set exactly to the probability that u and v are separated by Pivot, then x will be a feasible solution with cost at most $3\mathsf{OPT}$. However, this probability seems difficult to express in closed form, even approximately.

[9] In contrast, one cannot bound the fractional cost of the correlation metric on the star (Figure 1).

include it in the main body because we find the proof here is clean and lends insight into the challenges for the irregular case. The main technical result of the paper is Section 4, where we prove Theorem 1 by showing that the adjusted correlation metric can be input to the KMZ rounding algorithm. Namely, we will first show (quite easily) that the adjusted correlation metric satisfies an approximate triangle inequality. Then, it remains to upper bound the fractional cost of the adjusted correlation metric against OPT. We tackle this with a combinatorial charging argument. This argument leverages a somewhat different approach from that used in [19] and is simpler than their proof for only the ℓ_∞-norm. The *constant* approximation factor obtained from inputting the adjusted correlation metric to the KMZ rounding algorithm is bounded above (and below) by universal constants for all p (this is the worst case and one can get better constants for each p).

3 A Special Case: Regular Graphs

In general, the original correlation metric d (Definition 5) does not necessarily have bounded fractional cost for the ℓ_1-norm objective (or more generally for ℓ_p-norm objectives). So, we use the adjusted correlation metric f (Definition 6) as input to the KMZ rounding algorithm. In this section, we show that *if the subgraph of positive edges is regular*, then the correlation metric d can be used as is (i.e., without the adjustments in Steps 2 and 3 of Definition 6) to yield a clustering that is constant approximate for the ℓ_1-norm and ℓ_∞-norm simultaneously:

▶ **Theorem 7.** *Let $G = (V, E)$ be an instance of unweighted, complete correlation clustering, and let E^+ denote the set of positive edges. Suppose that the subgraph induced by E^+ is regular. The fractional cost of d in the ℓ_1-norm objective is within a constant factor of OPT:*

$$\sum_{u\in V}\sum_{v\in N_u^+} d_{uv} + \sum_{u\in V}\sum_{v\in N_u^-}(1-d_{uv}) = O(OPT).$$

Therefore, the clustering produced by inputting d to the KMZ rounding algorithm is a constant-factor approximation simultaneously for the ℓ_1-norm and ℓ_∞-norm objectives.

Proof. Let Δ be the (common) degree of the positive subgraph. To show that the fractional cost of d in the ℓ_1-norm objective is $O(\mathsf{OPT})$ for regular graphs, we will use a dual fitting argument. The LP relaxation we consider is from [4], which uses a dual fitting argument to show constant approximation guarantees for Pivot (although the proof here does not otherwise resemble the proof for Pivot). The primal is given by

$$\min\Big\{\sum_{e\in E} x_e \mid x_{ij}+x_{jk}+x_{ki}\geq 1, \forall ijk\in\mathcal{T}, x\geq 0\Big\} \tag{P'}$$

where $\mathcal{T}$ is the set of bad triangles (i.e. triangles with exactly two positive edges and one negative edge). For $x\in\{0,1\}^{|E|}$, x corresponds to disagreements in a clustering: we set $x_e = 1$ if e is a disagreement and $x_e = 0$ otherwise. The constraints state that every clustering must make a disagreement on every bad triangle. Thus, (P') is a relaxation for the ℓ_1-norm objective. In fact, we will prove the stronger statement that the fractional cost is $O(\mathsf{OPT}_{P'})$, where $\mathsf{OPT}_{P'}$ is the optimal objective value of (P').

The dual is given by

$$\max\Big\{\sum_{T\in\mathcal{T}} y_T \mid \sum_{T\in\mathcal{T}:T\ni e} y_T \leq 1, \forall e\in E, y\geq 0\Big\}. \tag{D'}$$

We show that by setting $y_T = \frac{1}{2\Delta}$ for all $T \in \mathcal{T}$, y satisfies the following properties:

1. y is feasible in (D').
2. The fractional cost of d is at most $6 \cdot \sum_{T \in \mathcal{T}} y_T$.

Letting $\mathsf{OPT}_{D'}$ be the optimal objective value of (D'), we have $6 \cdot \sum_{T \in \mathcal{T}} y_T \leq 6 \cdot \mathsf{OPT}_{D'} = 6 \cdot \mathsf{OPT}_{P'} \leq 6 \cdot \mathsf{OPT}$, which will conclude the proof.

To prove feasibility, we case on whether e is positive or negative.

If $e \in E^-$, then $|\{T \in \mathcal{T} : T \ni e\}| = |N_u^+ \cap N_v^+| \leq \Delta$, where equality is by the definition of a bad triangle. So $\sum_{T \in \mathcal{T}: T \ni e} y_T \leq \frac{\Delta}{2\Delta} \leq 1$.

If $e \in E^+$, then $|\{T : T \ni e\}| = |(N_u^+ \cap N_v^-) \cup (N_u^- \cap N_v^+)| \leq 2\Delta$. We conclude y is feasible, since $\sum_{T \in \mathcal{T}: T \ni e} y_T \leq \frac{2\Delta}{2\Delta} = 1$.

Now we need to show that the fractional cost of d is bounded in terms of the objective value of (D'). First we bound the fractional cost of the negative edges:

$$\sum_{(u,v) \in E^-} (1 - d_{uv}) \leq \sum_{(u,v) \in E^-} |N_u^+ \cap N_v^+| / \Delta = \sum_{e \in E^-} \sum_{T \in \mathcal{T}: T \ni e} 1/\Delta = \sum_{e \in E^-} \sum_{T \in \mathcal{T}: T \ni e} 2y_T,$$

where in the first inequality we have used that $|N_u^+ \cup N_v^+| \geq \Delta$. Next we bound the fractional cost of the positive edges:

$$\sum_{(u,v) \in E^+} d_{uv} \leq \sum_{(u,v) \in E^+} (|N_u^+ \cap N_v^-| + |N_u^- \cap N_v^+|)/\Delta = \sum_{e \in E^+} \sum_{T \in \mathcal{T}: T \ni e} 1/\Delta = \sum_{e \in E^+} \sum_{T \in \mathcal{T}: T \ni e} 2y_T.$$

So the total fractional cost is bounded by $\sum_{e \in E} \sum_{T \in \mathcal{T}: T \ni e} 2y_T = 6 \cdot \sum_{T \in \mathcal{T}} y_T$, since each triangle contains three edges. This is what we sought to show. Since the fractional cost of d is bounded for the ℓ_1-norm objective (and the ℓ_∞-norm objective by [19]), using d as input to KMZ rounding algorithm produces a clustering that is simultaneously $O(1)$-approximate for the ℓ_1- and ℓ_∞-norm objectives. ◀

4 Proof of Theorem 1

The goal of this section is to prove Theorem 1 and the subsequent Corollary 2. We begin by outlining that the adjusted correlation metric satisfies an approximate triangle inequality in Subsection 4.1. Then in Subsection 4.2, we prove the fractional cost of the adjusted correlation metric in any ℓ_p-norm objective is an $O(1)$ factor away from the optimal solution's value. We tie it all together to prove Theorem 1 and Corollary 2 in Subsection 4.3.

We start with an easy but key proposition. Loosely, it states that if two vertices are close to each other according to d, then they have a large shared positive neighborhood.

▶ **Proposition 8.** *Fix vertices $u, v \in V$ and a clustering $\mathcal{C}$ on V such that $d_{uv} \leq 0.7$ and $|N_u^+ \cap C(u)| \,/\, |N_u^+| \geq 0.85$. Then $|N_u^+ \cap N_v^+ \cap C(u)| \geq 0.15 \cdot |N_u^+|$.*

4.1 Triangle inequality

Recall that the correlation metric d satisfies the triangle inequality (see Section 4.2 in [19]). We will show that the adjusted correlation metric f satisfies an approximate triangle inequality, which is sufficient for the KMZ rounding algorithm. Formally, we say that a function g is a *δ-semi-metric* on some set S if it is a semi-metric on S, except instead of satisfying the triangle inequality, g satisfies $g(u, v) \leq \delta \cdot (g(u, w) + g(v, w))$ for all $u, v, w \in S$.

▶ **Lemma 9** (Triangle Inequality). *The adjusted correlation metric f is a $\frac{10}{7}$-semi-metric.*

The proof of Lemma 9 is straightforward given that d satisfies the triangle inequality.

Lemma 3 in [19] proves that one can input a semi-metric that satisfies an approximate triangle inequality (instead of the exact triangle inequality) to the KMZ rounding algorithm (with some loss in the approximation factor). We summarize the main take-away below.

▶ **Lemma 10** ([19]). *If g is a δ-semi-metric on the set V, instead of a true semi-metric (i.e., 1-semi-metric), then the KMZ algorithm loses a factor of $1+\delta+\delta^2+\delta^3+\delta^4$.*[10]

Since we show in Lemma 9 that f is a $\frac{10}{7}$-semi-metric, we lose a factor of 12 in inputting f to the KMZ algorithm (along with the factor loss from the fractional cost).

4.2 Bounding the fractional cost of ℓ_p-norms

This section bounds the fractional cost of the adjusted correlation metric for the ℓ_p-norms. The following lemma considers the case where $p=\infty$. The general case is handled after.

▶ **Lemma 11.** *The fractional cost of the adjusted correlation metric f in the ℓ_∞-norm objective is at most $56 \cdot \mathsf{OPT}$, where OPT is the cost of the optimal integral solution.*

The lemma follows from the fact that the fractional cost of the correlation metric d in the ℓ_∞-norm is known to be bounded by [19], and that it only decreases when d is replaced by f due to Definition 6. See Appendix B in the full version [20] for a proof.

We use two primary lemmas – one for the positive edge fractional cost and one for the negative edge fractional cost – to show that the adjusted correlation metric well approximates the optimal for general ℓ_p-norms.

▶ **Lemma 12.** *The fractional cost of the adjusted correlation metric f in the ℓ_p-norm objective is a constant factor (independent of p) away from the cost of the optimal integral ℓ_p solution.*

Proof. Let y be the disagreement vector for an optimal clustering $\mathcal{C}$ in the ℓ_p-norm, for any $p \in \mathbb{R}_{\geq 1} \cup \{\infty\}$. When $p=\infty$, see Lemma 11. For $p \in \mathbb{R}_{\geq 1}$, by definition $\mathsf{OPT}^p = \sum_{w\in V}(y(w))^p$, and the pth power of the fractional cost of f is given by

$$cost(f)^p = \sum_{u\in V}\left[\sum_{v\in N_u^+} f_{uv} + \sum_{v\in N_u^-}(1-f_{uv})\right]^p.$$

Observe that $$cost(f)^p \leq 2^p \underbrace{\sum_{u\in V}\Big(\sum_{v\in N_u^+} f_{uv}\Big)^p}_{(S^+)^p} + 2^p \underbrace{\sum_{u\in V}\Big(\sum_{v\in N_u^-}(1-f_{uv})\Big)^p}_{(S^-)^p}.$$

We refer to bounding $(S^+)^p$ as bounding the fractional cost of the positive edges, and likewise $(S^-)^p$ for the negative edges. The first sum, $(S^+)^p$, is bounded in Lemma 13 and the second sum, $(S^-)^p$, is bounded in Lemma 16. Using those two bounds, together we have $cost(f) \leq [2^p((S^+)^p + (S^-)^p)]^{1/p} \leq 529$, for $p \in [1,\infty)$. Specifically, the middle term is maximized at $p=1$, giving the bound of 529, and tends to below 214 as $p\to\infty$. (A more tailored analysis gives a constant of 74 for $p=1$; see Appendix C in the full version [20].) ◀

We note that, as our main interest is determining whether a simultaneous constant approximation is even possible (and a combinatorial one, at that), we did not pay particular attention to optimizing constants, but suspect these could be greatly reduced.

[10] When $\delta=1$, this factor equals 5, which is the loss in the KMZ algorithm.

4.2.1 Fractional cost of positive edges in ℓ_p-norms

We first bound the fractional cost of the positive edges.

▶ **Lemma 13.** *For $p \in \mathbb{R}_{\geq 1}$, the fractional cost of the adjusted correlation metric f in the ℓ_p-norm objective for the set of positive edges is a constant factor approximation to the optimal, i.e.,*

$$(S^+)^p = \sum_{u \in V} \Big(\sum_{v \in N_u^+} f_{uv} \Big)^p \leq 2^p \cdot [(8^p/2+1)((20/3)^p + 2 + 2 \cdot 4^p) + 8^p + 1] \cdot \mathit{OPT}^p.$$

One of the challenges in bounding the cost of f is that disagreements in the ℓ_p-norm objective for $p \neq 1$ are asymmetric, in that a disagreeing edge charges $y(u)$ and $y(v)$ (whereas for $p = 1$ we can just sum the number of disagreeing edges). Step 3 rounds up the edges incident to u when the tradeoff is good *from u's perspective*. However, an edge (u, v) may be rounded up to 1 when this tradeoff is good from v's perspective, but not from u's perspective. The high-level idea for why this is fine is that if u and v are close under d, their positive neighborhoods overlap significantly and, in some average sense, u can charge to v. Proving this requires a double counting argument using a bipartite auxiliary graph. If u and v are far under d, on the other hand, we can charge to the cost of the correlation metric, which will be bounded on an appropriate subgraph. The second challenge is showing that the ℓ_p-norm of the disagreement vector, restricted to vertices u that are made singletons in Step 3, is bounded. This again requires a double counting argument.

Proof. Fix an optimal clustering $\mathcal{C}$. We partition vertices based on membership in $C(u)$ or $\overline{C(u)}$ (as defined in Subsection 2.1). Let y denote the disagreement vector of $\mathcal{C}$. We have

$$(S^+)^p = \sum_{u \in V} \Big(\sum_{v \in N_u^+} f_{uv} \Big)^p \leq 2^p \underbrace{\sum_{u \in V} \Big(\sum_{v \in N_u^+ \cap C(u)} f_{uv} \Big)^p}_{S_1^+} + 2^p \underbrace{\sum_{u \in V} \Big(\sum_{v \in N_u^+ \cap \overline{C(u)}} f_{uv} \Big)^p}_{S_2^+}.$$

It is easy to bound S_2^+ by using the trivial upper bound $f_{uv} \leq 1$:

$$\boxed{S_2^+ = \sum_{u \in V} \Big(\sum_{v \in N_u^+ \cap \overline{C(u)}} f_{uv} \Big)^p \leq \sum_{u \in V} \Big(\sum_{v \in N_u^+ \cap \overline{C(u)}} 1 \Big)^p \leq \sum_{u \in V} (y(u))^p = \mathsf{OPT}^p,}$$

where we used that every edge $(u, v) \in E^+$ with $v \notin C(u)$ is a disagreement incident to u.

Next, we bound S_1^+. Let R_1 be the set of u for which Step 3 of Definition 6 applies. For these u, we have $f_{uv} = 1$ for all $v \in V \setminus \{u\}$. Let $R_2 = V \setminus R_1$. For $u \in R_2$ and $v \in N_u^+$, we have that either $v \in R_2$, in which case $f_{uv} = d_{uv}$; or $v \in R_1$, in which case $f_{uv} = 1$. (Note that V is the disjoint union of R_1 and R_2.) So

$$S_1^+ = \underbrace{\sum_{u \in R_1} \Big(\sum_{v \in N_u^+ \cap C(u), v \neq u} 1 \Big)^p}_{S_{11}^+} + \underbrace{\sum_{u \in R_2} \Big(\sum_{v \in N_u^+ \cap C(u)} f_{uv} \Big)^p}_{S_{12}^+},$$

and in particular

$$\begin{aligned}
S_{12}^+ &= \sum_{u\in R_2}\Big(\sum_{\substack{v\in N_u^+\cap C(u)\cap R_1}} 1 + \sum_{v\in N_u^+\cap C(u)\cap R_2} d_{uv}\Big)^p \\
&= \sum_{u\in R_2}\Big(\sum_{\substack{v\in N_u^+\cap C(u)\cap R_1\\ d_{uv}\le 1/4}} 1 + \sum_{\substack{v\in N_u^+\cap C(u)\cap R_1\\ d_{uv}\ge 1/4}} 1 + \sum_{v\in N_u^+\cap C(u)\cap R_2} d_{uv}\Big)^p \\
&\le \sum_{u\in R_2}\Big(\sum_{\substack{v\in N_u^+\cap C(u)\cap R_1\\ d_{uv}\le 1/4}} 1 + \sum_{\substack{v\in N_u^+\cap C(u)\cap R_1\\ d_{uv}\ge 1/4}} 4\cdot d_{uv} + \sum_{v\in N_u^+\cap C(u)\cap R_2} d_{uv}\Big)^p \\
&\le \sum_{u\in R_2}\Big(\sum_{\substack{v\in N_u^+\cap C(u)\cap R_1\\ d_{uv}\le 1/4}} 1 + \sum_{v\in N_u^+\cap C(u)} 4\cdot d_{uv}\Big)^p \\
&\le \underbrace{2^p\sum_{u\in R_2}\Big(\sum_{\substack{v\in N_u^+\cap R_1\\ d_{uv}\le 1/4}} 1\Big)^p}_{S_{13}^+} + \underbrace{8^p\cdot\sum_{u\in R_2}\Big(\sum_{v\in N_u^+\cap C(u)} d_{uv}\Big)^p}_{S_{14}^+}.
\end{aligned}$$

First we bound S_{13}^+. We will strongly use that $d_{uv}\le 1/4$ in the inner sum. Observe:

▶ **Proposition 14.** *Let d be the correlation metric, and $d_{uv}\le 1/4$. Then $|N_u^+|\le \frac{7}{3}\cdot|N_v^+|$.*

Next, we will need to create a bipartite auxiliary graph $H=(R_2,R_1,F)$ with R_2 and R_1 being the sides of the partition, and F being the edge set. We will then use a double counting argument. Place an edge between $u\in R_2$ and $v\in R_1$ if $uv\in E^+$ and $d_{uv}\le 1/4$. Then we have precisely that $S_{13}^+=\sum_{u\in R_2}\deg_H(u)^p$. We will show that

$$\boxed{S_{13}^+ = \sum_{u\in R_2}\deg_H(u)^p \le 4^{p-1}\cdot\sum_{v\in R_1}|N_v^+|^p \le 4^{p-1}\cdot((20/3)^p+2+2\cdot 4^p)\cdot\mathsf{OPT}^p} \tag{1}$$

where the last bound follows from Proposition 15, which we establish separately below. We will bound via double counting the quantity L, defined below. Let $N_H(\cdot)$ denote the neighborhoods in H of the vertices.

$$\begin{aligned}
L &:= \sum_{f=uv\in F}(\deg_H(u)+\deg_H(v))^{p-1} \le \sum_{v\in R_1}\sum_{u\in N_H(v)}(\deg_H(v)+\deg_H(u))^{p-1} \\
&\le \sum_{v\in R_1}\sum_{u\in N_H(v)}\big(|N_v^+|+|N_u^+|\big)^{p-1} \le \sum_{v\in R_1}\sum_{u\in N_H(v)} 4^{p-1}\cdot|N_v^+|^{p-1} \qquad (2)\\
&\le 4^{p-1}\cdot\sum_{v\in R_1}|N_v^+|\cdot|N_v^+|^{p-1} = 4^{p-1}\cdot\sum_{v\in R_1}|N_v^+|^p
\end{aligned}$$

where in (2) we've used Proposition 14. Note that L is upper bounded by the right-hand side in (1). Now it just remains to show that L is lower bounded by the left-hand side in (1).

$$\begin{aligned}
L &= \sum_{f=uv\in F}(\deg_H(u)+\deg_H(v))^{p-1} = \sum_{u\in R_2}\sum_{v\in N_H(u)}(\deg_H(u)+\deg_H(v))^{p-1} \\
&\ge \sum_{u\in R_2}\sum_{v\in N_H(u)}\deg_H(u)^{p-1} = \sum_{u\in R_2}\deg_H(u)\cdot\deg_H(u)^{p-1} = \sum_{u\in R_2}\deg_H(u)^p,
\end{aligned}$$

which is what we sought to show. Now we bound S_{14}^+.

$$\begin{aligned} S_{14}^+ &\le \sum_{u\in V}\left(\sum_{v\in N_u^+\cap C(u)} \frac{|N_u^+\cap N_v^-| + |N_u^-\cap N_v^+|}{|N_u^+\cup N_v^+|}\right)^p \\ &\le \sum_{u\in V}\left(\sum_{v\in N_u^+}\frac{y(u)+y(v)}{|N_u^+\cup N_v^+|}\right)^p \le \sum_{u\in V}|N_u^+|^{p-1}\sum_{v\in N_u^+}\frac{(y(u)+y(v))^p}{|N_u^+\cup N_v^+|^p} \\ &\le 2^p\sum_{u\in V}\sum_{v\in N_u^+}|N_u^+|^{p-1}\cdot\frac{y(u)^p}{|N_u^+\cup N_v^+|^p} + 2^p\sum_{u\in V}\sum_{v\in N_u^+}|N_u^+|^{p-1}\cdot\frac{y(v)^p}{|N_u^+\cup N_v^+|^p}. \end{aligned}$$

In the second line, the first inequality uses the fact that for $w\in(N_u^+\cap N_v^-)\cup(N_u^-\cap N_v^+)$, then at least one of $(u,w),(v,w)$ is a disagreement, since $v\in C(u)$ in the inner summation of the first line. The second inequality in the second line uses Jensen's inequality.

To bound the first double sum above, we use an averaging argument:

$$\sum_{u\in V}\sum_{v\in N_u^+}|N_u^+|^{p-1}\cdot\frac{y(u)^p}{|N_u^+\cup N_v^+|^p} \le \sum_{u\in V}\sum_{v\in N_u^+}\frac{y(u)^p}{|N_u^+|} = \sum_{u\in V}y(u)^p = \mathsf{OPT}^p.$$

To bound the second double sum, we first have to flip it:

$$\begin{aligned} \sum_{u\in V}\sum_{v\in N_u^+}|N_u^+|^{p-1}\cdot\frac{y(v)^p}{|N_u^+\cup N_v^+|^p} &= \sum_{v\in V}\sum_{u\in N_v^+}|N_u^+|^{p-1}\cdot\frac{y(v)^p}{|N_u^+\cup N_v^+|^p} \\ &\le \sum_{v\in V}\sum_{u\in N_v^+}|N_u^+|^{p-1}\frac{y(v)^p}{|N_u^+|^{p-1}\cdot|N_v^+|} = \sum_{v\in V}y(v)^p = \mathsf{OPT}^p. \end{aligned}$$

In total, we have

$$\boxed{S_{14}^+ \le 2\cdot 2^p\cdot\mathsf{OPT}^p = 2^{p+1}\cdot\mathsf{OPT}^p}$$

and $$\boxed{S_{12}^+ \le 2^p\cdot S_{13}^+ + 8^p\cdot S_{14}^+ \le 2^p\cdot 4^{p-1}\cdot((20/3)^p+2+2\cdot 4^p)\cdot\mathsf{OPT}^p + 8^p\cdot\mathsf{OPT}^p.}$$

Next we turn to bounding S_{11}^+. Recall that $R_1 = \{u : |N_u^-\cap\{v : d_{uv}\le 0.7\}| \ge \frac{10}{3}\cdot\Delta_u\}$ and

$$S_{11}^+ \le \sum_{u\in R_1}|N_u^+\cap C(u)|^p \le \sum_{u\in R_1}|N_u^+|^p.$$

So it suffices to bound the right-hand side, which we do in the following proposition.

▶ **Proposition 15.** *Let R_1 be the set of u for which Step 3 of Definition 6 applies. Then*

$$\sum_{u\in R_1}|N_u^+|^p \le ((20/3)^p+2+2\cdot 4^p)\cdot\mathit{OPT}^p.$$

Proof of Proposition 15. For $u\in R_1$, define $R_1(u) = N_u^-\cap\{v : d_{uv}\le 0.7\}$, so in particular $|R_1(u)|\ge\frac{10}{3}\cdot\Delta_u$. Fix a vertex $u\in R_1$. We consider a few cases. The crux is Case 2a(ii).

Case 1. *At least a 0.15 fraction of N_u^+ is in clusters other than $C(u)$.*
Let $u\in V^1$ be the vertices in this case. This means that $0.15\cdot|N_u^+|\le y(u)$, so

$$\boxed{\sum_{u\in V^1}|N_u^+|^p \le \sum_{u\in V^1}\frac{1}{0.15^p}y(u)^p \le (20/3)^p\cdot\mathsf{OPT}^p.}$$

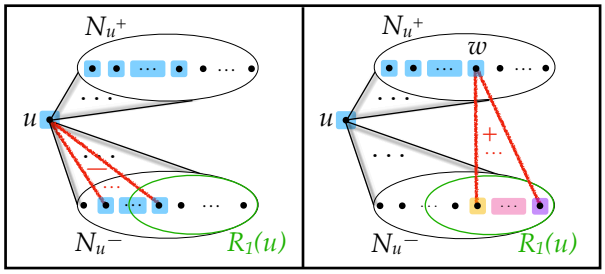


Figure 2 Left: Case 2a(i). For $v \in |N_u^- \cap C(u)|$, (u, v) is a disagreement. **Right:** Case 2a(ii). For $w \in |N_u^+ \cap C(u)|$ and $v \in N_w^+ \cap R_1'(u)$, (w, v) is a disagreement.

Case 2. *At least a 0.85 fraction of N_u^+ is in $C(u)$.*
We further partition the cases based on how much $R_1(u)$ intersects $C(u)$.

Case 2a: *At least half of $R_1(u)$ is in clusters other than $C(u)$.*
We partition into cases (just one more time!) based on the size of $N_u^- \cap C(u)$. See Figure 2.

Case 2a(i): *At least half of $R_1(u)$ is in clusters other than $C(u)$ and $|N_u^- \cap C(u)| \geq \Delta_u$.*
Let $u \in V^{2a(i)}$ be the vertices in this case. Note that $y(u) \geq |N_u^- \cap C(u)|$. Then

$$\sum_{u \in V^{2ai}} |N_u^+|^p = \sum_{u \in V^{2ai}} \Delta_u^p \leq \sum_{u \in V^{2ai}} |N_u^- \cap C(u)|^p \leq \sum_{u \in V^{2ai}} y(u)^p \leq \mathsf{OPT}^p.$$

Case 2a(ii): *At least half of $R_1(u)$ is in clusters other than $C(u)$ and $|N_u^- \cap C(u)| \leq \Delta_u$.*

Let $u \in V^{2a(ii)}$ be the vertices in this case. Denote the vertices in $R_1(u)$ that are in clusters other than $C(u)$ by $R_1'(u)$. By definition of Case 2a(ii), $|R_1'(u)| \geq \frac{5}{3} \cdot \Delta_u$. A key fact we will use is that $|C(u)| \leq 2 \cdot \Delta_u$:

$$|C(u)| = |N_u^- \cap C(u)| + |N_u^+ \cap C(u)| \leq \Delta_u + \Delta_u = 2 \cdot \Delta_u.$$

For $u \in V^{2a(ii)}$ and $w \in N_u^+ \cap C(u)$, define $\varphi(u, w) = |R_1'(u) \cap N_w^+|$.
Each $w \in N_u^+ \cap C(u)$ dispenses $\varphi(u, w)^p / |C(u)|$ charge to u. Also, observe that for $v \in R_1'(u)$, we have that $d_{uv} \leq 0.7$, so we know by Proposition 8 that $|N_u^+ \cap N_v^+ \cap C(u)| \geq 0.15 \cdot |N_u^+|$. This implies that

$$\begin{aligned}
\sum_{w \in N_u^+ \cap C(u)} |R_1'(u) \cap N_w^+| &= \sum_{w \in N_u^+ \cap C(u)} \sum_{v \in R_1'(u) \cap N_w^+} 1 = \sum_{v \in R_1'(u)} \sum_{\substack{w \in \cap N_v^+ \\ \cap C(u) \cap N_u^+}} 1 \\
&= \sum_{v \in R_1'(u)} |C(u) \cap N_u^+ \cap N_v^+| \geq \sum_{v \in R_1'(u)} 0.15 \cdot |N_u^+| \\
&= 0.15 \cdot |N_u^+| \cdot |R_1'(u)| \geq 0.15 \cdot \Delta_u \cdot \frac{5}{3} \Delta_u = 0.25 \cdot \Delta_u^2.
\end{aligned}$$

By Jensen's inequality, the amount of charge each u satisfying Case 2a(ii) receives is at least

$$\begin{aligned}
\frac{1}{|C(u)|} \sum_{w \in N_u^+ \cap C(u)} \varphi(u, w)^p &\geq \frac{1}{|C(u)|} \cdot \frac{1}{|N_u^+ \cap C(u)|^{p-1}} \cdot \Big(\sum_{w \in N_u^+ \cap C(u)} \varphi(u, w) \Big)^p \\
&\geq \frac{1}{2\Delta_u} \cdot \frac{1}{\Delta_u^{p-1}} \cdot \left(0.25 \cdot \Delta_u^2\right)^p \geq \frac{1}{2} \cdot 0.25^p \cdot |N_u^+|^p,
\end{aligned}$$

Next we need to upper bound the amount of charge dispensed in total to *all* u satisfying Case 2a(ii). Note by definition that $\varphi(u,w) \le y(w)$. Each vertex $w \in V$ dispenses at most $y(w)^p/|C(u)| = y(w)^p/|C(w)|$ charge to each $u \in C(w) \cap N_w^+$. So in total each w dispenses at most $|C(w)| \cdot y(w)^p/|C(w)| = y(w)^p$ charge to all u satisfying Case 2a(ii). Now we put together the lower and upper bounds on the total charge dispensed:

$$\sum_{w\in V} y(w)^p \ge \text{charge dispensed} \ge \sum_{u\in V^{2a(ii)}} \frac{1}{|C(u)|} \sum_{w\in N_u^+\cap C(u)} \varphi(u,w)^p$$
$$\ge \sum_{u\in V^{2a(ii)}} \frac{1}{2}\cdot 0.25^p \cdot |N_u^+|^p.$$

In all, $$\boxed{\sum_{u\in V^{2a(ii)}} |N_u^+|^p \le 2\cdot 4^p \cdot \sum_{w\in V} y(w)^p \le 2\cdot 4^p \cdot \mathsf{OPT}^p.}$$

Case 2b: *At least half of $R_1(u)$ is in $C(u)$.*

Let $u \in V^{2b}$ be the vertices in this case. Denote the vertices in $R_1(u)$ that are in $C(u)$ by $R_1''(u)$. By definition of Case 2b, $|R_1''(u)| \ge \frac{5}{3}\cdot\Delta_u$. Since every vertex in $R''(u)$ is in N_u^-, there are at least $|R''(u)|$ disagreements incident to u. So $y(u) \ge |R''(u)| \ge \frac{5}{3}\cdot\Delta_u$, giving

$$\boxed{\sum_{u\in V^{2b}} |N_u^+|^p = \sum_{u\in V^{2b}} \Delta_u^p \le \sum_{u\in V^{2b}} y(u)^p \le \mathsf{OPT}^p.}$$

Adding the terms in the boxed expressions across all cases, the proposition follows. ◀

So we have $$\boxed{S_{11}^+ \le \sum_{u\in R_1} |N_u^+|^p \le ((20/3)^p + 2 + 2\cdot 4^p)\cdot \mathsf{OPT}^p.}$$

Adding together all the cases, we conclude that

$$(S^+)^p \le 2^p\cdot(S_1^+ + S_2^+) \le 2^p\cdot(S_{11}^+ S_{12^+} + S_2^+) \le 2^p\cdot[(8^p/2+1)((20/3)^p+2+2\cdot 4^p)+8^p+1]\cdot\mathsf{OPT}^p.$$

◀

4.2.2 Fractional cost of negative edges in ℓ_p-norms

This section bounds the cost of negative edges. The meanings of $\mathcal{C}$, $C(\cdot)$, and y are as before.

▶ **Lemma 16.** *For $p \in \mathbb{R}_{\ge 1}$, the fractional cost of the adjusted correlation metric f in the ℓ_p-norm objective for the set of negative edges is a constant factor away from optimal:*

$$(S^-)^p = \sum_{u\in V}\Big(\sum_{v\in N_u^-}(1-f_{uv})\Big)^p \le 2^p((200/9)^p + 1 + (10/3)^p + 2\cdot(20/3)^p)\cdot \mathit{OPT}^p.$$

Proof. We have

$$(S^-)^p = \sum_{u\in V}\Big(\sum_{v\in N_u^-}(1-f_{uv})\Big)^p \le 2^p \underbrace{\sum_{u\in V}\Big(\sum_{v\in N_u^-\cap C(u)}(1-f_{uv})\Big)^p}_{S_1^-}$$
$$+ 2^p \underbrace{\sum_{u\in V}\Big(\sum_{v\in N_u^-\cap \overline{C(u)}}(1-f_{uv})\Big)^p}_{S_2^-}.$$

It is easy to bound S_1^- by using the trivial upper bound $1 - f_{uv} \le 1$:

$$\boxed{S_1^- = \sum_{u\in V}\Big(\sum_{v\in N_u^-\cap C(u)}(1-f_{uv})\Big)^p \le \sum_{u\in V}\Big(\sum_{v\in N_u^-\cap C(u)}1\Big)^p \le \sum_{u\in V} y(u)^p = \mathsf{OPT}^p,}$$

where we have used that every edge $(u, v) \in E^-$ with $v \in C(u)$ is a disagreement incident to u. Next, we bound S_2^-. Let R_1 and R_2 be as in the previous subsection: $R_1 = \{u : |N_u^- \cap \{v : d_{uv} \le 0.7\}| \ge \frac{10}{3}\cdot\Delta_u\}$ and $R_2 = V \setminus R_1$. For $u \in R_2$, define

$$V_u = \{v : v \in N_u^- \cap \overline{C(u)}, d_{uv} \le 0.7\}.$$

Note that the definition of V_u is the same as $R_1'(u)$ in the previous subsection, but here V_u is only defined for $u \in R_2$, while $R_1'(u)$ was defined for $u \in R_1$. For $u \in R_1$, we have $1 - f_{uv} = 0$ for every $v \in V \setminus \{u\}$. So the outer sum in S_2^- only need be taken over $u \in R_2$:

$$S_2^- = \sum_{u\in R_2}\Big(\sum_{v\in N_u^-\cap\overline{C(u)}}(1-f_{uv})\Big)^p \le \sum_{u\in R_2}\Big(\sum_{\substack{v:v\in N_u^-\cap\overline{C(u)},\\ d_{uv}\le 0.7}}(1-d_{uv})\Big)^p \le \sum_{u\in R_2}|V_u|^p$$

In the second equality, we have used that if $u \in R_2$ and $v \in N_u^-$, then $f_{uv} = d_{uv}$, unless f_{uv} was rounded up to 1 in Step 2 of Definition 6 (which happens when $d_{uv} > 0.7$), or f_{uv} was rounded up to 1 in Step 3 (in which case $1 - f_{uv} = 0 \le 1 - d_{uv}$).

A key observation is that since $u \in R_2$, it is the case that $|V_u| \le \frac{10}{3}\cdot\Delta_u$.

Fix a vertex $u \in R_2$. We consider a few cases.

Case 1. *At least a 0.15 fraction of N_u^+ is in clusters other than $C(u)$.*
Define V^1 to be the set of $u \in R_2$ satisfying Case 1. Then for $u \in V^1$, $0.15\cdot|N_u^+| \le y(u)$, and

$$|V_u| \le \frac{10}{3}\Delta_u \le \frac{1}{0.15}\cdot\frac{10}{3}y(u) = \frac{200}{9}y(u).$$

so $$\boxed{\sum_{u\in V^1}|V_u|^p \le (200/9)^p\cdot\sum_{u\in V^1}y(u)^p \le (200/9)^p\cdot\mathsf{OPT}^p.}$$

Case 2. *At least a 0.85 fraction of N_u^+ is in $C(u)$.*
Define V^2 to be the set of $u \in R_2$ that satisfy Case 2. Fix $u \in V^2$ and $v \in V_u$. Define $N_{u,v} = N_u^+ \cap N_v^+ \cap C(u)$. Since $d_{uv} \le 0.7$ and by the assumption of this case, using Proposition 8 we have

$$|N_{u,v}| = |N_u^+ \cap N_v^+ \cap C(u)| \ge 0.15\cdot\Delta_u.$$

Observe that since $v \notin C(u)$ for $v \in V_u$, (v, w) is a (positive) disagreement for all $w \in N_{u,v}$.
Case 2a: $|N_u^- \cap C(u)| \ge \Delta_u$.
Define V^{2a} to be the set of $u \in V^2$ that satisfy Case 2a. Since all edges (u, v) with $v \in N_u^- \cap C(u)$ are disagreements, we have $y(u) \ge \Delta_u$. Recalling that $|V_u| \le \frac{10}{3}\cdot\Delta_u$ for $u \in R_2$, we have

$$\boxed{\sum_{u\in V^{2a}}|V_u|^p \le \sum_{u\in V^{2a}}(10/3\cdot\Delta_u)^p \le (10/3)^p\cdot\sum_{u\in V^{2a}}y(u)^p \le (10/3)^p\cdot\mathsf{OPT}^p.}$$

Case 2b: $|C(u)| \leq 2\Delta_u$.

Define V^{2b} to be the $u \in V^2$ satisfying Case 2b. Fix $w \in N_u^+ \cap C(u)$ and $u \in V^{2b}$. Define

$$\varphi(u,w) = |V_u \cap N_w^+|,$$

i.e. $\varphi(u,w)$ is the number of $v \in V_u$ with $w \in N_{u,v}$. Each $w \in N_u^+ \cap C(u)$ dispenses $\frac{\varphi(u,w)^p}{|C(u)|}$ charge to u. Also,

$$\begin{aligned}\sum_{w \in N_u^+ \cap C(u)} \varphi(u,w) &= \sum_{w \in N_u^+ \cap C(u)} |V_u \cap N_w^+| = \sum_{w \in N_u^+ \cap C(u)} \sum_{v \in V_u \cap N_w^+} 1 \\ &= \sum_{v \in V_u} \sum_{w \in N_{u,v}} 1 = \sum_{v \in V_u} |N_{u,v}| \geq |V_u| \cdot 0.15 \cdot \Delta_u.\end{aligned}$$

By Jensen's inequality, the amount of charge each u satisfying Case 2b receives is at least

$$\begin{aligned}\frac{1}{|C(u)|} \sum_{w \in N_u^+ \cap C(u)} \varphi(u,w)^p &\geq \frac{1}{|C(u)|} \cdot \frac{1}{|N_u^+ \cap C(u)|^{p-1}} \Big(\sum_{w \in N_u^+ \cap C(u)} \varphi(u,w) \Big)^p \\ &\geq \frac{1}{2\Delta_u} \cdot \frac{1}{\Delta_u^{p-1}} \left(|V_u| \cdot 0.15 \cdot \Delta_u\right)^p = \frac{1}{2} \cdot 0.15^p \cdot |V_u|^p,\end{aligned}$$

To upper bound the amount of charge dispensed in total to *all* u satisfying Case 2b, first note that $\varphi(u,w) \leq y(w)$. Also, each vertex $w \in V$ only distributes charge to $u \in C(w) \cap N_w^+$, and the amount of charge distributed to each such u is

$$\frac{\varphi(u,w)^p}{|C(u)|} = \frac{\varphi(u,w)^p}{|C(w)|} \leq \frac{y(w)^p}{|C(w)|},$$

so that in total each w dispenses at most $\frac{y(w)^p}{|C(w)|} \cdot |C(w)| \leq y(w)^p$ charge. Putting together the lower and upper bounds on the amount of charge dispensed:

$$\begin{aligned}\sum_{w \in V} y(w)^p \geq \text{total charge dispensed} &\geq \sum_{u \in V^{2b}} \frac{1}{|C(u)|} \sum_{w \in N_u^+ \cap C(u)} \varphi(u,w)^p \\ &\geq \sum_{u \in V^{2b}} \frac{1}{2} \cdot 0.15^p \cdot |V_u|^p.\end{aligned}$$

In all, $\boxed{\sum_{u \in V^{2b}} |V_u|^p \leq 2 \cdot (20/3)^p \cdot \sum_{w \in V} y(w)^p = 2 \cdot (20/3)^p \cdot \mathsf{OPT}^p.}$

Combining the cases, we see that $(S^-)^p \leq 2^p((200/9)^p + 1 + (10/3)^p + 2 \cdot (20/3)^p) \cdot \mathsf{OPT}^p$. ◀

4.3 Proofs of Theorem 1 and Corollary 2

Here we show that Theorem 1 follows directly from the preceding lemmas. We defer the proof of Corollary 2 to the full version [20].

Proof of Theorem 1. First we show that Lemma 12 implies that the clustering resulting from inputting f into the KMZ rounding algorithm is $O(1)$-approximate in any ℓ_p-norm. Since the rounding algorithm does not depend on p, the clustering will be the same for all p. Let $\mathcal{C}^*$ be the clustering produced by running the KMZ rounding algorithm with the adjusted correlation metric f as input. Let $\mathsf{ALG}(u)$ be the number of edges incident to u that are disagreements with respect to $\mathcal{C}^*$. From [29] and Lemmas 9 and 10, we have that

for every $u \in V$, $\mathsf{ALG}(u) \leq 12 \cdot y_u$ where y_u is as in LP P when taking $x = f$. So $||y||_p$ is the fractional cost of f in the ℓ_p-norm. In what follows, $||\mathsf{ALG}||_p$ is the objective value of $\mathcal{C}^*$ in the ℓ_p-norm and $\mathsf{OPT}(p)$ is the optimal objective value in the ℓ_p-norm. Thus using Lemma 12 in the last inequality, we have

$$||\mathsf{ALG}||_p \leq 12 \cdot ||y||_p \leq 12 \cdot 529 \cdot \mathsf{OPT}(p) = 6348 \cdot \mathsf{OPT}(p).$$

The overall run-time is $O(n^\omega)$. From the analysis in [19], computing the correlation metric takes time $O(n^\omega)$, and the KMZ rounding algorithm takes time $O(n^2)$. We just have to show the post-processing of d in Steps 2 and 3 of Definition 6 that were done in order to obtain the adjusted correlation metric f can be done quickly. Indeed, Step 2 takes $O(n^2)$ time as it simply iterates through the edges. Step 3 also takes $O(n^2)$ time, since it visits each vertex and iterates through the neighbors. Thus, the run-time remains $O(n^\omega)$. ◀

5 Conclusion

This paper considered correlation clustering on unweighted, complete graphs, a problem that arises in many settings including community detection and the study of large networks. All previous works that study minimizing the ℓ_p-norm (for $p \in \mathbb{R}_{>1}$) of the disagreement vector rely on solving a large, convex relaxation (which is costly to the algorithm's run-time) and produce a solution that is only $O(1)$-approximate for one specific value of p. We innovate upon this rich line of work by (1) giving the first combinatorial algorithm for the ℓ_p-norms for $p \in \mathbb{R}_{>1}$, (2) designing scalable algorithms for this practical problem, and (3) obtaining solutions that are $O(1)$-approximate for all ℓ_p-norms (for $p \in \mathbb{R}_{\geq 1} \cup \{\infty\}$) simultaneously. We emphasize this last point, as such solutions are good in both global and local senses, and thus may be more desirable than typical optimal or approximate solutions. The existence of these solutions reveals a surprising structural property of correlation clustering.

One question is whether there is a simpler existential (not necessarily algorithmic) proof that there exists an $O(1)$-approximation for the all-norm objective for correlation clustering.

It is also of interest to implement the KMZ algorithm with the adjusted correlation metric as input, and empirically gain an understanding of how good the adjusted correlation metric is for different ℓ_p-norms. We suspect that our analysis is lossy (we did not focus on optimizing constants), and that the approximation obtained would be of much better quality.

References

1. Saba Ahmadi, Sainyam Galhotra, Barna Saha, and Roy Schwartz. Fair correlation clustering. *arXiv preprint*, 2020. `arXiv:2002.03508`.
2. Saba Ahmadi, Samir Khuller, and Barna Saha. Min-max correlation clustering via multicut. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 13–26. Springer, 2019.
3. Sara Ahmadian, Alessandro Epasto, Ravi Kumar, and Mohammad Mahdian. Fair correlation clustering. In *International Conference on Artificial Intelligence and Statistics*, pages 4195–4205. PMLR, 2020.
4. Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):1–27, 2008.
5. Soroush Alamdari and David Shmoys. A bicriteria approximation algorithm for the k-center and k-median problems. In *Approximation and Online Algorithms: 15th International Workshop, WAOA 2017, Vienna, Austria, September 7–8, 2017, Revised Selected Papers 15*, pages 66–75. Springer, 2018.

6 Yossi Azar, Leah Epstein, Yossi Richter, and Gerhard J Woeginger. All-norm approximation algorithms. *Journal of Algorithms*, 52(2):120–133, 2004.

7 Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine learning*, 56(1):89–113, 2004.

8 Mohammadhossein Bateni, Vincent Cohen-Addad, Alessandro Epasto, and Silvio Lattanzi. Scalable and improved algorithms for individually fair clustering. In *Workshop on Trustworthy and Socially Responsible Machine Learning, NeurIPS 2022*, 2022.

9 Amir Ben-Dor and Zohar Yakhini. Clustering gene expression patterns. In *Proceedings of the third annual international conference on computational molecular biology*, pages 33–42, 1999.

10 Aaron Bernstein, Tsvi Kopelowitz, Seth Pettie, Ely Porat, and Clifford Stein. Simultaneously load balancing for every p-norm, with reassignments. In *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

11 Francesco Bonchi, David Garcia-Soriano, and Edo Liberty. Correlation clustering: from theory to practice. In *KDD*, page 1972, 2014.

12 Costas Busch, Chinmoy Dutta, Jaikumar Radhakrishnan, Rajmohan Rajaraman, and Srivathsan Srinivasagopalan. Split and join: Strong partitions and universal steiner trees for graphs. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 81–90. IEEE, 2012.

13 Ostas Busch, Arnold Filtser, Daniel Hathcock, D Ellis Hershkowitz, Rajmohan Rajaraman, et al. One tree to rule them all: Poly-logarithmic universal steiner tree. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 60–76. IEEE, 2023.

14 Sayak Chakrabarty and Konstantin Makarychev. Single-pass pivot algorithm for correlation clustering. keep it simple! *arXiv preprint*, 2023. `arXiv:2305.13560`.

15 Moses Charikar, Neha Gupta, and Roy Schwartz. Local guarantees in graph cuts and clustering. In Friedrich Eisenbrand and Jochen Könemann, editors, *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, volume 10328 of *Lecture Notes in Computer Science*, pages 136–147. Springer, 2017. `doi:10.1007/978-3-319-59250-3_12`.

16 Shuchi Chawla, Konstantin Makarychev, Tselil Schramm, and Grigory Yaroslavtsev. Near optimal lp rounding algorithm for correlationclustering on complete and complete k-partite graphs. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 219–228, 2015.

17 Flavio Chierichetti, Nilesh Dalvi, and Ravi Kumar. Correlation clustering in mapreduce. In *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 641–650, 2014.

18 Vincent Cohen-Addad, Euiwoong Lee, and Alantha Newman. Correlation clustering with sherali-adams. *Symposium on Foundations of Computer Science (FOCS).*, 2022.

19 Sami Davies, Benjamin Moseley, and Heather Newman. Fast combinatorial algorithms for min max correlation clustering. In *International Conference on Machine Learning*. PMLR, 2023.

20 Sami Davies, Benjamin Moseley, and Heather Newman. Simultaneously approximating all ℓ_p-norms in correlation clustering, 2024. `arXiv:2308.01534`.

21 Erik D Demaine and Nicole Immorlica. Correlation clustering with partial information. In *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*, pages 1–13. Springer, 2003.

22 Zachary Friggstad and Ramin Mousavi. Fair correlation clustering with global and local guarantees. In *Workshop on Algorithms and Data Structures*, pages 414–427. Springer, 2021.

23 Arun Ganesh, Bruce M Maggs, and Debmalya Panigrahi. Universal algorithms for clustering problems. *ACM Transactions on Algorithms*, 19(2):1–46, 2023.

24 Daniel Golovin, Anupam Gupta, Amit Kumar, and Kanat Tangwongsan. All-norms and all-l_p-norms approximation algorithms. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2008.

25 Holger Heidrich, Jannik Irmai, and Bjoern Andres. A 4-approximation algorithm for min max correlation clustering. *arXiv preprint*, 2023. arXiv:2310.09196.

26 Jafar Jafarov, Sanchit Kalhan, Konstantin Makarychev, and Yury Makarychev. Local correlation clustering with asymmetric classification errors. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 4677–4686. PMLR, 2021. URL: http://proceedings.mlr.press/v139/jafarov21a.html.

27 Jafar Jafarov, Sanchit Kalhan, Konstantin Makarychev, and Yury Makarychev. Local correlation clustering with asymmetric classification errors. In *International Conference on Machine Learning*, pages 4677–4686. PMLR, 2021.

28 Lujun Jia, Guolong Lin, Guevara Noubir, Rajmohan Rajaraman, and Ravi Sundaram. Universal approximations for tsp, steiner tree, and set cover. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 386–395, 2005.

29 Sanchit Kalhan, Konstantin Makarychev, and Timothy Zhou. Correlation clustering with local objectives. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 9341–9350, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/785ca71d2c85e3f3774baaf438c5c6eb-Abstract.html.

30 Jon Kleinberg, Yuval Rabani, and Éva Tardos. Fairness in routing and load balancing. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 568–578. IEEE, 1999.

31 Zach Langley, Aaron Bernstein, and Sepehr Assadi. Improved bounds for distributed load balancing. In *34th International Symposium on Distributed Computing (DISC 2020). Schloss Dagstuhl – Leibniz-Zentrum für Informatik*, 2020.

32 Andrew McCallum and Ben Wellner. Conditional models of identity uncertainty with application to noun coreference. *Advances in neural information processing systems*, 17, 2004.

33 Xinghao Pan, Dimitris Papailiopoulos, Benjamin Recht, Kannan Ramchandran, and Michael I Jordan. Scaling up correlation clustering through parallelism and concurrency control. In *DISCML Workshop at International Conference on Neural Information Processing Systems*, 2014.

34 Gregory J Puleo and Olgica Milenkovic. Correlation clustering with constrained cluster sizes and extended weights bounds. *SIAM Journal on Optimization*, 25(3):1857–1872, 2015.

35 Gregory J. Puleo and Olgica Milenkovic. Correlation clustering and biclustering with locally bounded errors. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 869–877. JMLR.org, 2016. URL: http://proceedings.mlr.press/v48/puleo16.html.

36 Jessica Shi, Laxman Dhulipala, David Eisenstat, Jakub Lacki, and Vahab S. Mirrokni. Scalable community detection via parallel correlation clustering. *Proc. VLDB Endow.*, 14:2305–2313, 2021.

37 Nate Veldt. Correlation clustering via strong triadic closure labeling: Fast approximation algorithms and practical lower bounds. In *International Conference on Machine Learning*, pages 22060–22083. PMLR, 2022.

38 Nate Veldt, David F Gleich, and Anthony Wirth. A correlation clustering framework for community detection. In *Proceedings of the 2018 World Wide Web Conference*, pages 439–448, 2018.

39 Anthony Wirth. Correlation clustering. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning and Data Mining*, pages 280–284. Springer, 2017. doi:10.1007/978-1-4899-7687-1_176.

Parameterized Algorithms for Coordinated Motion Planning: Minimizing Energy

Argyrios Deligkas
Department of Computer Science, Royal Holloway, University of London, Egham, UK

Eduard Eiben
Department of Computer Science, Royal Holloway, University of London, Egham, UK

Robert Ganian
Algorithms and Complexity Group, TU Wien, Austria

Iyad Kanj
School of Computing, DePaul University, Chicago, IL, USA

M. S. Ramanujan
Department of Computer Science, University of Warwick, Coventry, UK

Abstract

We study the parameterized complexity of a generalization of the coordinated motion planning problem on graphs, where the goal is to route a specified subset of a given set of k robots to their destinations with the aim of minimizing the total energy (i.e., the total length traveled). We develop novel techniques to push beyond previously-established results that were restricted to solid grids.

We design a fixed-parameter additive approximation algorithm for this problem parameterized by k alone. This result, which is of independent interest, allows us to prove the following two results pertaining to well-studied coordinated motion planning problems: (1) A fixed-parameter algorithm, parameterized by k, for routing a single robot to its destination while avoiding the other robots, which is related to the famous Rush-Hour Puzzle; and (2) a fixed-parameter algorithm, parameterized by k plus the treewidth of the input graph, for the standard Coordinated Motion Planning (CMP) problem in which we need to route all the k robots to their destinations. The latter of these results implies, among others, the fixed-parameter tractability of CMP parameterized by k on graphs of bounded outerplanarity, which include bounded-height subgrids.

We complement the above results with a lower bound which rules out the fixed-parameter tractability for CMP when parameterized by the total energy. This contrasts the recently-obtained tractability of the problem on solid grids under the same parameterization. As our final result, we strengthen the aforementioned fixed-parameter tractability to hold not only on solid grids but all graphs of bounded local treewidth – a class including, among others, all graphs of bounded genus.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases coordinated motion planning, multi-agent path finding, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.53

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2404.15950

Funding *Argyrios Deligkas*: Supported by Engineering and Physical Sciences Research Council (EPSRC) grant EP/X039862/1.
Robert Ganian: Project No. Y1329 of the Austrian Science Fund (FWF), Project No. ICT22-029 of the Vienna Science Foundation (WWTF).
Iyad Kanj: DePaul URC Grants 606601 and 350130.
M. S. Ramanujan: Supported by Engineering and Physical Sciences Research Council (EPSRC) grants EP/V007793/1 and EP/V044621/1.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 53; pp. 53:1–53:18

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The task of routing a set of robots (or agents) from their initial positions to specified destinations while avoiding collisions is of great importance in a multitude of different settings. Indeed, there is an extensive body of works dedicated to algorithms solving this task, especially in the computational geometry [1,24–26,30,32–34,37], artificial intelligence [6,35,38] and robotics [3, 23, 36] research communities. Such algorithms typically aim at providing a schedule for the robots which is not only safe (in the sense of avoiding collisions), but which also optimizes a certain measure of the schedule – typically its makespan or energy (i.e., the total distance traveled by all robots). In this article, we focus solely on the task of optimizing the latter of these two measures.

A common formalization of our task of interest is given by the Coordinated Motion Planning problem (also known as Multiagent Pathfinding). In the context of energy minimization, the problem can be stated as follows: Given a graph G, a budget ℓ and a set $\mathcal{M}$ of robots, each equipped with an initial vertex and destination vertex in G, compute a schedule which delivers all the robots to their destinations while ensuring that the combined length traveled of all the robots is at most ℓ. Here, a schedule can be seen as a sequence of commands to the robots, where at every time step each of the robots can move to an adjacent vertex as long as no vertex or edge is used by more than a single robot at that time step. Coordinated Motion Planning, which generalizes the famous NP-hard (n^2-1)-puzzle [11, 29], has been extensively studied – both in the discrete setting considered here as well as in various continuous settings [4, 6, 10, 14, 23, 28, 32–36, 38–40] – and has also been the target of specific computing challenges [17]. In other variants of the problem, there is no requirement to route *all* the robots to their destinations – sometimes the majority (or all but 1) of the robots are simply movable obstacles that do not have destinations of their own. This is captured, e.g., by the closely-related and well-studied Rush Hour puzzle/problem [7, 19] and by Graph Motion Planning with 1 Robot (GCMP1) [27], which both feature a single robot with a designated destination.

Both Coordinated Motion Planning and GCMP1 are known to be NP-hard [11, 29, 39]. While several works have already studied Coordinated Motion Planning in the context of approximation and classical complexity theory, a more fine-grained investigation of the difficulty of finding optimal schedules through the lens of *parameterized complexity* [9, 13] was only carried out recently [15, 18]. The work in [15] established the fixed-parameter tractability[1] of Coordinated Motion Planning parameterized by either the number k of robots or the budget ℓ (as well as the makespan variant when parameterized by k), but only on solid grids; a *solid* grid is a standard rectangular $p \times q$ grid (i.e., with no holes), for some $p, q \in \mathbb{N}$. The more recent work in [18] showed the W[1]-hardness of the makespan variant of Coordinated Motion Planning w.r.t. the number of robots. The paper [18] also showed the NP-hardness of the makespan problem-variant on trees, and presented parameterized complexity results with respect to several combinations of parameters. In this article, we focus our attention on GCMP [27], which generalizes both Coordinated Motion Planning and GCMP1 by allowing an arbitrary partitioning of the robots into those with destinations and those which act merely as movable obstacles[2].

[1] A problem is *fixed-parameter tractable* w.r.t. a parameter k if it can be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$, where n is the input size and f is a computable function.

[2] While previous hardness results for GCMP considers serial motion of robots (e.g., [27]), the hardness applies to parallel motion as well. Here we obtain algorithms for the coordinated motion variant with parallel movement, but note that the results can be directly translated to the serial version.

Contribution. The aim of this article is to push our understanding of the parameterized complexity of finding energy-optimal schedules beyond the class of solid grids. While this aim was already highlighted in the aforementioned paper on solid grids [15, Section 5], the techniques used there are highly specific to that setting and it is entirely unclear how one could generalize them even to the setting of subgrids; a *subgrid* is a subgraph of a solid grid and we define its *height* to be the minimum of its two dimensions.

As our two main contributions, we provide novel fixed-parameter algorithms (1) for GCMP1 parameterized by the number of robots alone (Theorem 1), and (2) for GCMP parameterized by the number of robots plus the treewidth of the graph (Theorem 2). Theorem 2 implies, as a corollary, the fixed-parameter tractability of GCMP parameterized by k plus the minimum dimension of the subgrid.

▶ **Theorem 1.** GCMP1 *is* FPT *parameterized by the number k of robots.*

▶ **Theorem 2.** GCMP *is* FPT *parameterized by the number of robots and the treewidth of the input graph.*

The main technical tool we use to obtain both of these results is a novel fixed-parameter approximation algorithm for Generalized Coordinated Motion Planning parameterized by k alone, where the approximation error is only *additive* in k. We believe this result – summarized in Theorem 3 below – to be of independent interest.

▶ **Theorem 3.** *There is an* FPT *approximation algorithm for* GCMP *parameterized by the number k of robots which guarantees an additive error of $\mathcal{O}(k^5)$.*

The proof of Theorem 1 builds upon Theorem 3. For the proof of Theorem 2, we need to combine the approximation algorithm with novel insights concerning the "decomposability" of schedules along small separators in order to design a treewidth-based dynamic programming algorithm for the problem. A brief summary of the ideas used in this proof is provided at the beginning of Section 3.

We complement our positive results which use k as a parameter with an algorithmic lower bound showing that Coordinated Motion Planning is W[1]-hard (and hence not fixed-parameter tractable under well-established complexity-theoretic assumptions) when parameterized by the energy budget ℓ. This result (Theorem 4 below) contrasts the fixed-parameter tractability of the problem under the same parameterization when restricted to solid grids [15, Theorem 19].

▶ **Theorem 4.** GCMP *is* W[1]*-hard when parameterized by ℓ.*

While Theorem 4 establishes the intractability of the problem when parameterized by the energy on general graphs, one can in fact show that GCMP is fixed-parameter tractable under the same parameterization when the graphs are "well-structured" in the sense of having bounded *local treewidth* [16, 22]. This implies fixed-parameter tractability, e.g., on graphs of bounded genus and generalizes the aforementioned result on grids [15, Theorem 19].

▶ **Theorem 5.** GCMP *is* FPT *parameterized by ℓ on graph classes of bounded local treewidth.*

Further Related Work. As surveyed above, the computational complexity of GCMP has received significant attention, particularly by researchers in the fields of computational geometry, AI/Robotics, and theoretical computer science. The problem has been shown to remain NP-hard under a broad set of restrictions, including on graphs where only a single

vertex is not occupied [21], on grids [3], and bounded-height subgrids [20]. On the other hand, a feasibility check for the existence of a schedule can be carried out in polynomial time [41]. The recent AAMAS blue sky paper [31] also highlighted the need of understanding the hardness of the problem and asked for a deeper investigation of its parameterized complexity.

2 Terminology and Problem Definition

The graphs considered in this paper are undirected simple graphs. We assume familiarity with the standard graph-theoretic concepts and terminology [12]. For a subgraph H of a graph G and two vertices $u, v \in V(H)$, we denote by $\text{dist}_H(u, v)$ the length of a shortest path in H between u and v. We write $[n]$, where $n \in \mathbb{N}$, for $\{1, \ldots, n\}$.

We also assume basic familiarity with parameterized complexity theory, including *fixed-parameter tractability*, *parameterized reductions*, and the class W[1] [9, 13].

Treewidth. Treewidth is a structural parameter that provides a way of expressing the resemblance of a graph to a forest. Formally, the treewidth of a graph is defined via the notion of tree decompositions as follows.

▶ **Definition 6** (Tree decomposition)**.** A *tree decomposition* of a graph G is a pair (T, β) of a tree T and $\beta : V(T) \to 2^{V(G)}$, such that:

- $\bigcup_{t \in V(T)} \beta(t) = V(G)$,
- for any edge $e \in E(G)$, there exists a node $t \in V(T)$ such that both endpoints of e belong to $\beta(t)$, and
- for any vertex $v \in V(G)$, the subgraph of T induced by the set $T_v = \{t \in V(T) : v \in \beta(t)\}$ is a tree.

The *width* of (T, β) is $\max_{v \in V(T)}\{|\beta(v)|\} - 1$. The *treewidth* of G is the minimum width of a tree decomposition of G.

Let (T, β) be a tree decomposition of a graph G. We refer to the vertices of the tree T as *nodes*. We always assume that T is a rooted tree and hence, we have a natural parent-child and ancestor-descendant relationship among nodes in T. A *leaf* node nor a *leaf* of T is a node with degree exactly one in T which is different from the root node. All the nodes of T which are neither the root node or a leaf are called *non-leaf* nodes. The set $\beta(t)$ is called the *bag* at t. For two nodes $u, t \in T$, we say that u is a *descendant* of t, denoted $u \preceq t$, if t lies on the unique path connecting u to the root. Note that every node is its own descendant. If $u \preceq t$ and $u \neq t$, then we write $u \prec t$. For a tree decomposition (T, β) we also have a mapping $\gamma : V(T) \to 2^{V(G)}$ defined as $\gamma(t) = \bigcup_{u \preceq t} \beta(u)$.

We use the following structured tree decomposition in our algorithm.

▶ **Definition 7** (Nice tree decomposition)**.** Let (T, β) be a tree decomposition of a graph G, where r is the root of T. The tree decomposition (T, β) is called a *nice tree decomposition* if the following conditions are satisfied.

1. $\beta(r) = \emptyset$ and $\beta(\ell) = \emptyset$ for every leaf node ℓ of T; and
2. every non-leaf node (including the root node) t of T is of one of the following types:
   - **Introduce node:** The node t has exactly one child t' in T and $\beta(t) = \beta(t') \cup \{v\}$, where $v \notin \beta(t')$.
   - **Forget node:** The node t has exactly one child t' in T and $\beta(t) = \beta(t') \setminus \{v\}$, where $v \in \beta(t')$.
   - **Join node:** The node t has exactly two children t_1 and t_2 in T and $\beta(t) = \beta(t_1) = \beta(t_2)$.

A graph class closed under vertex and edge deletion is said to have *bounded local treewidth* [16,22] if the treewidth of each graph in the class is upper-bounded by a function of its diameter. Examples of classes of bounded local treewidth include, e.g., graphs of bounded genus [5].

Problem Definition. In our problems of interest, we are given an undirected graph G and a set $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ of k robots where $\mathcal{R}$ is partitioned into two sets $\mathcal{M}$ and $\mathcal{F}$. Each $R_i \in \mathcal{M}$ has a starting vertex s_i and a destination vertex t_i in $V(G)$ and each $R_i \in \mathcal{F}$ is associated only with a starting vertex $s_i \in V(G)$. We refer to the elements in the set $\{s_i \mid i \in [k]\} \cup \{t_i \mid R_i \in \mathcal{M}\}$ as *terminals*. The set $\mathcal{M}$ contains robots that have specific destinations they must reach, whereas $\mathcal{F}$ is the set of remaining "free" robots. We assume that all the s_i are pairwise distinct and that all the t_i are pairwise distinct. At each time step, a robot may either move to an adjacent vertex, or stay at its current vertex, and robots may move simultaneously. We use a discrete time frame $[0, t]$, $t \in \mathbb{N}$, to reference the sequence of moves of the robots and in each time step $x \in [0, t]$ every robot remains stationary or moves.

A *route* for robot R_i is a tuple $W_i = (u_0, \ldots, u_t)$ of vertices in G such that (i) $u_0 = s_i$ and $u_t = t_i$ if $R_i \in \mathcal{M}$ and (ii) $\forall j \in [t]$, either $u_{j-1} = u_j$ or $u_{j-1}u_j \in E(G)$. Put simply, W_i corresponds to a "walk" in G, with the exception that consecutive vertices in W_i may be identical (representing waiting time steps), in which R_i begins at its starting vertex at time step 0, and if $R_i \in \mathcal{M}$ then R_i reaches its destination vertex at time step t. Two routes $W_i = (u_0, \ldots, u_t)$ and $W_j = (v_0, \ldots, v_t)$, where $i \neq j \in [k]$, are *non-conflicting* if (i) $\forall r \in \{0, \ldots, t\}$, $u_r \neq v_r$, and (ii) $\nexists r \in \{0, \ldots, t-1\}$ such that $v_{r+1} = u_r$ and $u_{r+1} = v_r$. Otherwise, we say that W_i and W_j *conflict*. Intuitively, two routes conflict if the corresponding robots are at the same vertex at the end of a time step, or go through the same edge (in opposite directions) during the same time step.

A *schedule* S for $\mathcal{R}$ is a set of pairwise non-conflicting routes $W_i, i \in [k]$, during a time interval $[0, t]$. The (*traveled*) *length* of a route (or its associated robot) within S is the number of time steps j such that $u_j \neq u_{j+1}$. The *total traveled length* of a schedule is the sum of the lengths of its routes; this value is often called the *energy* in the literature (e.g., see [17]).

Using the introduced terminology, we formalize the Generalized Coordinated Motion Planning with Energy Minimization (GCMP) problem below.

GCMP

Input: A tuple $(G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$, where G is a graph, $\mathcal{R} = \{R_i \mid i \in [k]\}$ is a set of robots partitioned into sets $\mathcal{M}$ and $\mathcal{F}$, where each robot in $\mathcal{M}$ is given as a pair of vertices (s_i, t_i) and each robot in $\mathcal{F}$ as a single vertex s_i, and $k, \ell \in \mathbb{N}$.

Problem: Is there a schedule for $\mathcal{R}$ of total traveled length at most ℓ?

By observing that the feasibility check of Yu and Rus [41] transfers seamlessly to the case where some robots do not have destinations, we obtain the following.

▶ **Proposition 8** ([41]). *The existence of a schedule for an instance of* GCMP *can be decided in linear time. Moreover, if such a schedule exists, then a schedule with total length traveled of $\mathcal{O}(|V(G)|^3)$ can be computed in $\mathcal{O}(|V(G)|^3)$ time.*

Proposition 8 implies inclusion in NP, and allows us to assume henceforth that every instance of GCMP is feasible (otherwise, in linear time we can reject the instance). We denote by GCMP1 the restriction of GCMP to instances where $|\mathcal{M}| = 1$. We remark that even though GCMP is stated as as a decision problem, all the algorithms provided in this paper are constructive and can output a corresponding schedule (when it exists) as a witness.

3 An Additive FPT Approximation for GCMP

In this section, we give an FPT approximation algorithm for GCMP with an additive error that is a function of the number k of robots.

We start by providing a high-level, low-rigor intuition for the main result of this section. Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of GCMP. Ideally, we would like to route the robots in $\mathcal{M}$ along shortest paths to their destinations, while having the other robots move away only a "little" ($\epsilon(k)$-many steps) to unblock the shortest paths for the robots in $\mathcal{M}$. Unfortunately, this might not be possible as it is easy to observe that a free robot might have to travel a long distance in order to unblock the shortest paths of other robots. (For instance, think about the situation where a free robot is positioned in the middle of a long path of degree-2 vertices that the shortest paths traverse.) However, we will show that such a situation could only happen if the blocking robots are positioned in simple graph structures containing a long path of degree-2 vertices.

We then exploit these simple structures to "guess" in FPT-time, for each robot, a location which it visits in an optimal solution and which is in the vicinity of a safe location, called a "haven"; this haven is centered around a "nice" vertex of degree at least 3, and allows the robot to avoid any passing robot within $\epsilon(k)$ moves. We show how to navigate the robots to these havens optimally in the case of GCMP1, and with an $\epsilon(k)$ overhead in the case of GCMP. Moreover, this navigation is well-structured and can be leveraged to show that no robot in an optimal solution will visit the same vertex many times. A similar navigation takes place at the end, during the routing of the robots from their havens to their destinations.

Once we obtain such a reduced instance in which all starting positions and destinations of the robots are in havens, we can use our intended strategy to navigate each robot in $\mathcal{M}$ along a shortest path, with only an $\epsilon(k)$ overhead, which immediately gives us the approximation result and the property that no robot visits any vertex more than $\epsilon(k)$ times in an optimal solution. This latter property about the optimal solution is crucial, as we exploit it later in Section 5 to design an intricate dynamic programming algorithm over a tree decomposition of the input graph.

In addition, each free robot moves at most $\epsilon(k)$ times in the considered solution for the reduced instance. This, together with the equivalence between $\mathcal{I}$ and the reduced instance in the case of GCMP1, allow us to restrict the movement of the free robots in an optimal solution to only $\epsilon(k)$-many locations, which we use in Section 4 to obtain the FPT algorithm for GCMP1.

We start by defining the notion of a nice vertex and its haven.

▶ **Definition 9** (Nice Vertex). A vertex $w \in V(G)$ is *nice* if there exist three connected subgraphs C_1, C_2, C_3 of G such that: (i) the pairwise intersection of the vertex sets of any pair of these subgraphs is exactly w, and (ii) $|V(C_1)| \geq k+1$, $|V(C_2)| \geq k+1$, and $|V(C_3)| \geq 2$. If w is nice, let $x \in V(C_3)$ be a neighbor of w, and define the *haven* H_w of w to be the subgraph of G induced by the vertices in $\{x\} \cup V(C_1) \cup V(C_2)$ whose distance from w in H_w (i.e., $\text{dist}_{H_v}(w, u)$, for $u \in V(C_1) \cup V(C_2)$) is at most k.

For a set $S \subseteq \mathcal{R}$ of robots and a subgraph H, a *configuration* of S w.r.t. H is an injection $\iota : S \longrightarrow V(H)$. The following lemma shows that we can take the robots in a haven from any configuration to any other configuration in the haven, while incurring a total of $\mathcal{O}(k^3)$ travel (in the haven) length.

▶ **Lemma 10.** *Let w be a nice vertex, let C_1, C_2, C_3 be three subgraphs satisfying conditions (i) and (ii) of Definition 9, and let H_w be a haven for w. Then for any set of robots $S \subseteq \mathcal{R}$ in H_w with current configuration $\iota(S)$, any configuration $\iota'(S)$ with respect to H_w can be obtained from $\iota(S)$ via a sequence of $\mathcal{O}(k^3)$ moves that take place in H_w.*

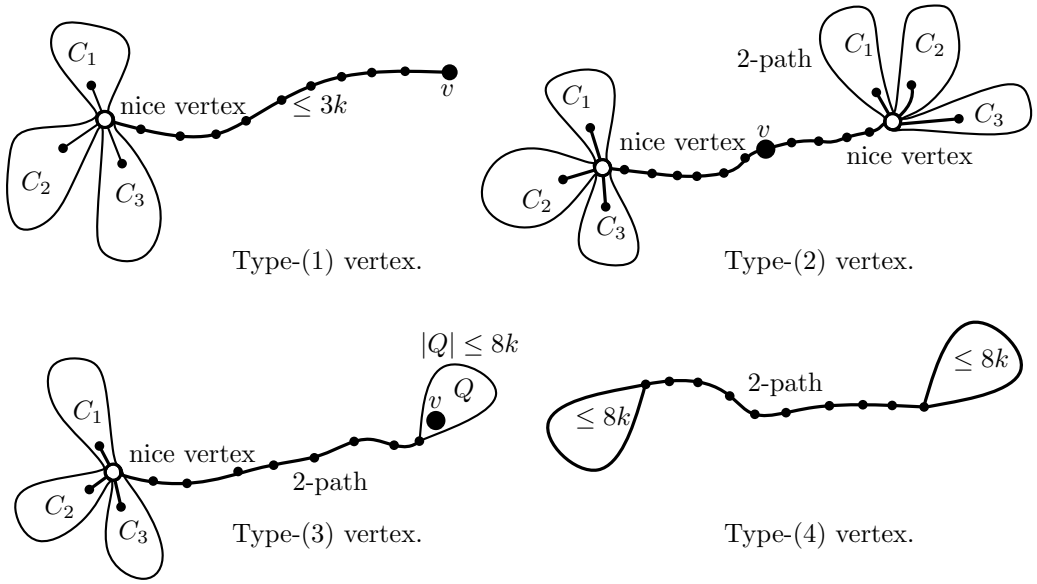


Figure 1 Illustration of the four types for a vertex v that is not nice.

Call a path P in G a *2-path* if P is an induced path of degree-2 vertices in G. The next lemma is used to characterize the possible graph structures around a vertex that is not nice; an illustration for the four cases handled by the lemma is provided in Figure 1.

▶ **Lemma 11.** *For every vertex v that is not nice, one of the following holds:*

(1) *There is a nice vertex at distance at most $3k$ from v; or*

(2) *vertex v is on a 2-path whose both endpoints are nice vertices; or*

(3) *there is a 2-path P such that one of its endpoints is nice and the other is contained in a connected subgraph Q of size at most $8k$ that P cuts from the nice vertex and such that v is in $P \cup Q$; or*

(4) *Either $|V(G)| \leq 8k$, or G consists of a 2-path P, each of whose endpoints is contained in a subgraph of size at most $8k$, such that P cuts the two subgraphs from one another.*

▶ **Definition 12** (Vertex Types)**.** Let v be a vertex in G that is not nice. We say that v is a *type-(i) vertex*, where $i \in \{1, \ldots, 4\}$, if v satisfies Statement (i) in Lemma 11 but does not satisfy any Statement (j), where $j < i$ (if Statement (j) exists). Note that, by Lemma 11, every vertex v that is not nice must be a type-(i) vertex for some $i \in \{1, \ldots, 4\}$.

The following lemma shows that we can restrict our attention to the case where there is no type-(4) vertex in G.

▶ **Lemma 13.** *Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of* GCMP*. If there exists a type-(4) vertex $v \in V(G)$ then $\mathcal{I}$ can be solved in time $\mathcal{O}^*((4k^2 + 16k)^{2k})$ and hence is* FPT*.*

Before we are ready to prove the main theorem for this section, we first need the following lemma, which will allow us to restrict the movement of the free robots.

▶ **Lemma 14.** *Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of* GCMP*. For every $R_i \in \mathcal{F}$ that is within distance λ_i from a nice vertex, there is a set $B(R_i, \lambda_i)$ of vertices of cardinality $k^{\mathcal{O}(\lambda_i \cdot k + k^4)}$ such that, for any optimal solution $opt(\mathcal{I})$, there exists an optimal solution $opt'(\mathcal{I})$ satisfying that, in $opt'(\mathcal{I})$ R_i moves only on vertices in $B(R_i, \lambda_i)$ and all the remaining robots move exactly the same in $opt(\mathcal{I})$ and $opt'(\mathcal{I})$. Moreover, $B(R_i, \lambda_i)$ is computable in $\mathcal{O}(|V(G)| + |E(G)|)$ time.*

▶ **Theorem 15.** *Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of* GCMP. *In* FPT*-time, we can reduce $\mathcal{I}$ to $p = g(k)$-many instances (for some computable function $g(k)$) $\mathcal{I}_1, \ldots, \mathcal{I}_p$ such that there exists $j \in [p]$ satisfying:*

1. *If $\mathcal{M} = \{R_1\}$ then $\mathcal{I}_j = (G_j, \mathcal{R}_j = (\mathcal{M}_j, \emptyset), k_j, \ell_j)$, where $k_j \leq k$ and $\ell_j \leq \ell$, is a yes-instance of* GCMP *if and only if $\mathcal{I}$ is a yes-instance. Moreover, in this case there is an optimal solution $opt(\mathcal{I}_j)$ such that $|opt(\mathcal{I}_j)| = \mathrm{dist}_{G_j}(s_1, t_1) + \mathcal{O}(k^5)$, and for every robot $R_i \in \mathcal{M}_j \setminus \{R_1\}$, the moves of R_i in $opt(\mathcal{I}_j)$ are restricted to a vertex set of cardinality at most $k^{\mathcal{O}(k^4)}$ that is computable in linear time.*
2. *In polynomial time we can compute a schedule for $\mathcal{I}_j$ with total traveled length at most $\sum_{R_i \in \mathcal{M}_j} \mathrm{dist}(s_i, t_i) + \mathcal{O}(k^5)$, and given such a schedule for $\mathcal{I}_j$, in polynomial time we can compute a schedule for $\mathcal{I}$ of cost at most $|opt(\mathcal{I})| + \mathcal{O}(k^5)$.*

Furthermore, every optimal schedule for $\mathcal{I}$ satisfies the property that every vertex in G is visited at most $\mathcal{O}(k^5)$ many times in the schedule.

Proof Sketch. Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of GCMP, and let $opt(\mathcal{I})$ denote an optimal schedule for $\mathcal{I}$. We give a nondeterministic reduction that produces a single instance $\mathcal{I}_j = (G_j, R_j = (\mathcal{M}_j, \emptyset), k_j, \ell_j)$, which can be made deterministic in FPT-time.

To construct $\mathcal{I}_j$, we will be making guesses about the initial and final segments of the robots' routes in $opt(\mathcal{I})$, which may lead to redefining the starting and final positions for some of them, with the purpose of getting them close to havens – as explained at the beginning of this section. The guessing is based on the types of the starting and final positions of the robots (see Lemma 11). We set $\ell_j = \ell$.

Case I. If the starting or the final position of a robot is at distance at most $11k$ from a nice vertex, we do not change it.

Case II. If the starting or the final position of a robot is a type-(2) vertex on a path P that connects two nice vertices, then we look separately at all the robots that start on P and all the robots that end on P (guessing for each free robot whether it ends on P). Let us first consider the robots that start on P. We make a guess, for each of these robots, about whether it ever leaves P, and if it does, from which endpoint of P it leaves for the first time. This splits the robots on P into three sets S_{Left}, S_{Mid}, S_{Right}. Observe that these three sets have to be "consecutive" on P (i.e., their robots respect the order of the sets), and we push the starting points of the robots in S_{Left} and in S_{Right} to distance at most k from the nice vertex they leave from for the first time. Similarly, we split the robots that end on P into the sets T_{Left}, T_{Mid}, T_{Right}. We redefine the destinations of the robots in T_{Left} and T_{Right} to vertices at distance at most k from their respective nice vertices through which they enter P, and we compute the additional travel length incurred by moving them directly from these positions at distance at most k from nice vertices to their destinations and subtract this number from ℓ_j. Recall that, according to our guess, S_{Mid} and T_{Mid} each contains precisely the robots that never leave P, and hence $S_{\mathrm{Mid}} = T_{\mathrm{Mid}}$. We show that for most of the robots in S_{Mid}, any optimal solution takes them directly from their starting positions to their destinations, except for those that start and end at distance at most k from the same nice vertex. For those that do not start and end at distance at most k from the same nice vertex, we compute the travel length incurred by taking them directly to their destinations, subtract it from ℓ_j, and delete these robots from $\mathcal{R}$. Finally, if $S_{\mathrm{Mid}} \neq \emptyset$, then no robot can use P to move between its two endpoints. Moreover, if a robot is on P, then it does not need to go beyond the k-closest vertices on P to a nice vertex. Hence, if $S_{\mathrm{Mid}} \neq \emptyset$, then we delete all the vertices on P that are at distance at least $k + 1$ from a nice vertex.

Case III. If the starting or the final position of a robot is a type-(3) vertex, which consists of a nice vertex n_R joined by a 2-path P to a vertex u in a connected subgraph C of size at most $8k$, then we start by treating this situation the same as that of a type-(2) vertex, with the role of the second nice vertex replaced by u. In particular, we guess the sets S_{Left}, S_{Mid}, S_{Right}, T_{Left}, T_{Mid}, T_{Right}, where S_{Right} (resp. T_{Right}) are the robots that start their routing by going from a vertex on P to u, or from u to a vertex on P. We also change the starting and ending positions for the robots that are not in S_{Mid} to vertices that are at distance at most k from either n_R or u; we compute the routing that reflects this change, and subtract its travel length from ℓ_j.

If $S_{\text{Mid}} \neq \emptyset$, then the vertices on which the robots in S_{Mid} are positioned separate the robots that are in $S_{\text{Right}} \cup T_{\text{Right}}$ plus the robots that start or end in C, from the rest of the graph. Note that at this point, there is no interaction between the robots in $S_{\text{Right}} \cup T_{\text{Right}} \cup S_{\text{Mid}}$ and the other robots in $\mathcal{R}$. Therefore, we can treat C and the part of P containing the robots in $S_{\text{Right}} \cup T_{\text{Right}} \cup S_{\text{Mid}}$ as a separate instance, which has the same structure as that of a type-(4) vertex. We solve this instance optimally using Lemma 13, and keep on $V(P)$ only the k-closest vertices to n_R. On the other hand, if $S_{\text{Mid}} = \emptyset$ then some robots that start (resp. end) in S_{Right} (resp. T_{Right}) or in C might also visit n_R. We show that, in the case of GCMP1, this happens only in a very restricted setting where the single robot in $\mathcal{M}$ starts or ends in C, which can be dealt with easily with the help of Lemma 13. On the other hand, if $|\mathcal{M}| \geq 2$, then this is no longer the case. We now further guess which of the robots that start (resp. end) in S_{Right} (resp. T_{Right}) or in C also visit n_R on the other end of P. Using Proposition 8, we can show that we can always route these robots at the beginning (resp. at the end) to (resp. from) a position at distance at most k from n_R with overhead at most $\mathcal{O}(k^3)$ over an optimal routing. Afterwards, we keep in G only the vertices on P at distance at most k from n_R, and remove the rest of $V(C) \cup V(P)$.

This concludes the reduction to obtain the instance $\mathcal{I}_j$ from $\mathcal{I}$.

Let us now prove Statement 1. Observe that all the free robots can now be either at distance at most $11k$ from a nice vertex and we can use Lemma 14 to compute the set $B(R_i, 11k)$ where they can move; or alternatively, they can be at distance at most k from a connected subgraph of size at most $8k$ associated with a type-(3) vertex and they will stay there, and hence, their movement in this case is restricted to $9k$ vertices.

We now show how to compute a schedule for $\mathcal{I}_j$ with total traveled length at most $\text{dist}(s_1, t_1) + \mathcal{O}(k^5)$. To do so, we first move all the robots that are at distance at most $11k$ from a nice vertex v to a haven H_v of v. We then compute a shortest path P between s_1 and t_1. We note that P can only interact with a connected subgraph C of a type-(3) vertex at the beginning or at the end. In both cases, we can use Lemma 13 to compute an optimal routing of R_1 from C (resp. to) to the first (resp. from the last) haven P intersects. Afterwards, all free robots on P are in havens. We let R_1 follow P. Whenever P interacts with a haven H that contains a free robot, we replace the subpath of P between the first interaction of R_1 with H and the last interaction of R_1 with H with a schedule computed by Lemma 10 that reconfigures H taking R_1 between its two interaction positions. Each interaction with a haven increases the length of P by $\mathcal{O}(k^3)$ and there are at most k havens that contain a free robot. Finally, we move the free robots at the end to their guessed destinations in $\mathcal{I}_j$. We can show that this incurs a travel length of $\mathcal{O}(k^5)$.

For Statement 2, if $|\mathcal{M}| = 1$ then Statement 2 follows from Statement 1. Otherwise, all starting positions and destinations are at distance at most $11k$ from nice vertices. Due to this, we can show that we can start by incurring an $\mathcal{O}(k^2)$ travel length to route each robot to a nearest haven, and finish by incurring $\mathcal{O}(k^4)$ travel length to route all the robots from

havens at distance at most $11k$ from their destinations to their destinations. We can then route the robots one-by-one from a "starting" haven to a "destination" haven along a shortest path in the same manner we routed the robot R_1 in the case of GCMP1. The routing of each robot incurs an overhead of $\mathcal{O}(k^4)$ above its shortest path length, and hence the total overhead for routing all the robots between havens is $\mathcal{O}(k^5)$ and Statement 2 follows. ◀

The approximation algorithm claimed in the introduction (restated below) follows directly from Statement 2 of Theorem 15.

▶ **Theorem 3.** *There is an* FPT *approximation algorithm for* GCMP *parameterized by the number k of robots which guarantees an additive error of $\mathcal{O}(k^5)$.*

4 An FPT Algorithm for GCMP1 Parameterized by k

The aim of this section is to establish Theorem 1 by using Theorem 15.

▶ **Theorem 1.** GCMP1 *is* FPT *parameterized by the number k of robots.*

Proof Sketch. Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of GCMP1. By Statement 1 of Theorem 15, in FPT-time we can compute an equivalent instance $\mathcal{I}_j = (G_j, \mathcal{R}_j = (\mathcal{M}_j, \emptyset), k_j, \ell_j)$ to $\mathcal{I}$ satisfying that $k_j \leq k$ and $|opt(\mathcal{I}_j)| = \text{dist}_{G_j}(s_1, t_1) + \mathcal{O}(k^5)$, where $\mathcal{M} = \{R_1\}$. Moreover, for every robot $R_i \in \mathcal{M}_j \setminus \{R_1\}$, the moves of R_i in $opt(\mathcal{I}_j)$ are restricted to the vertices of a vertex-set $B(R_i)$ of cardinality at most $k^{\mathcal{O}(k^4)}$ that is computable in linear time.

Define a state graph $\mathcal{Q}$ whose vertices are k-tuples with coordinates defined as follows. The first coordinate of a k_j-tuple corresponds to R_1 and can be any of the at most n vertices in $V(G_j)$; the i-th coordinate of a tuple, where $i \in \{2, \ldots, k_j\}$, encodes the possible location of R_i and is confined to the vertices in $B(R_i)$. We purge any tuple in which a vertex in $V(G_j)$ appears in more than one coordinate of the tuple (i.e., is occupied by more than one robot in the same time step). Since $|B(R_i)| = k^{\mathcal{O}(k^4)}$, for $i \in \{2, \ldots, k_j\}$, it follows that the number of vertices in $\mathcal{Q}$ is at most $\mathcal{O}(n \cdot k^{\mathcal{O}(k^5)})$. Two vertices/states S and S' in $\mathcal{Q}$ are adjacent if there is a valid (i.e., causing no collision) single (parallel) move for the robots from their locations in S to their locations in S'; the weight of an edge (S, S') in $\mathcal{Q}$ is the number of robots in S that have moved (i.e., their positions have changed).

Define the starting configuration S_{start} in $\mathcal{Q}$ to be the k_j-tuple whose coordinates correspond to the starting positions of the robots in $\mathcal{I}_j$. Define S_{final} to be the k_j-tuple whose coordinates correspond to the destinations of the robots in $\mathcal{I}_j$. Now compute a shortest (weighted) path from S_{start} to S_{final} in $\mathcal{Q}$ (e.g., using Dijkstra's algorithm) and accept the instance $\mathcal{I}$ if and only if the weight of the computed shortest path is at most ℓ_j. The running time of the algorithm is $\mathcal{O}(|V(\mathcal{Q})|^2) = \mathcal{O}(n^2 \cdot k^{\mathcal{O}(k^5)})$, and it is clear that the above algorithm decides the instance correctly. It follows that GCMP1 is FPT parameterized by k. ◀

5 An FPT Algorithm Parameterized by Treewidth and k

In this section, we present an FPT algorithm for GCMP parameterized by the number of robots and the treewidth of the input graph combined. This result implies that the problem is FPT on certain graph classes such as graphs of bounded outerplanarity, which include subgrids of bounded height. In this sense, the result can be seen as complementary to the recently established NP-hardness of the problem on bounded-height subgrids [20].

▶ **Definition 16** (Semi-routes). A *semi-route* for R_i is a tuple $W_i = (u_0^i, \ldots, u_t^i)$ such that each u_j^i is either a vertex of G or the symbol $\perp$, and such that (i) $u_0^i = s_i$ and moreover, if $R_i \in \mathcal{M}$, then $u_t^i = t_i$, and (ii) $\forall j \in [t]$, either $u_{j-1}^i = u_j^i$ or $u_{j-1}^i u_j^i \in E(G)$ or one out of u_{j-1}^i and u_j^i is the symbol $\perp$. The notion of two conflicting semi-routes is identical to that for routes, except that we only consider time steps where neither semi-routes has $\perp$.

Intuitively, the above definition gives us a notion of partial routes, where the robots can be thought of as having become "invisible" (but still potentially in motion) for some time steps during a route (assume only a small part of the graph is visible to us). These time steps where a robot disappears from view are represented by the symbol $\perp$. Note that a robot can "reappear" at a different vertex than the one it "disappeared" at. A *semi-schedule* for $\mathcal{R}$ is a set of semi-routes during a time interval $[0, t]$ that are pairwise non-conflicting. The *length* of a semi-route and a semi-schedule is naturally defined as the number of time steps in which the robot moves from one vertex to a different vertex, and the *total traveled length* of a semi-schedule is the sum of the lengths of its semi-routes.

A *boundaried graph* [9] is a graph G with a set $X \subseteq V(G)$ of distinguished vertices called *boundary vertices*; the set X is called the *boundary* of G. A boundaried graph (G, X) is called a *p-boundaried* graph if $|X| \leq p$. A *p-boundaried subgraph* of a graph G is a p-boundaried graph (H, Z) such that (i) H is a vertex-induced subgraph of G and (ii) Z separates $V(H) \setminus Z$ from $V(G) \setminus V(H)$.

In what follows, let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of GCMP. Let (H, Z) be a p-boundaried subgraph of G with boundary Z containing all the terminals. Let S be a schedule for this instance with routes $W_i, i \in [k]$.

▶ **Definition 17** (Signatures of Schedules). Call the tuples in the set $(Z \cup \{\uparrow, \downarrow\})^k$, *configuration tuples* for (H, Z). We define the *signature* of the schedule S with respect to (H, Z) as a sequence of tuples $\tau_0, \ldots, \tau_t$, where each τ_i is a configuration tuple defined as follows: the j^{th} coordinate of τ_i signifies whether at the end of time step i

- R_j is on a vertex $v \in Z$, in which case this value is v; or
- whether it is inside H but not on the boundary, in which case this value is $\downarrow$; or
- whether it is disjoint from H, in which case this value is $\uparrow$.

The tuple τ_i is called the *signature of the schedule S with respect to (H, Z) at time step i*. The tuple τ_0 and τ_t are called *starting* and *ending* configuration tuples.

▶ **Definition 18** (Checkpoints of Schedules). We say that the schedule S:

- *externally affects* (H, Z) *at time steps* i and $i+1$ if some robot moves from a vertex outside H to a vertex in Z during time step $i+1$, or if some robot moves from some vertex of Z to a vertex outside H during time step $i+1$. That is, for some robot R_j, we either have that $u_i^j \notin V(H)$ and $u_{i+1}^j \in Z$, or we have $u_i^j \in Z$ and $u_{i+1}^j \notin V(H)$.
- *internally affects* (H, Z) at time steps i and $i+1$ if some robot moves from a vertex of H to a vertex of Z during time step $i+1$, or if some robot moves from some vertex of Z to a vertex in H during time step $i+1$. That is, for some robot R_j, we have $u_i^j \neq u_{i+1}^j$ and moreover, we either have that $u_i^j \in V(H)$ and $u_{i+1}^j \in Z$ or we have $u_i^j \in Z$ and $u_{i+1}^j \in V(H)$.
- *affects* (H, Z) at time steps i and $i+1$ if it internally or externally affects (H, Z) at time steps i and $i+1$.

The pairs of consecutive time steps at which the schedule affects (H, Z) are called *checkpoints of the schedule with respect to* (H, Z). We drop the explicit reference to (H, Z) whenever it is clear from the context. Two checkpoints $(x, x+1)$ and $(y, y+1)$ are said to be *consecutive* if $x < y$ and there is no checkpoint $(z, z+1)$ such that $x < z < y$.

Roughly speaking, the checkpoints are the time steps at which the schedule interacts in some way with the separator Z. The interaction involves a robot either moving to or from a vertex of Z. As a result, the notion of checkpoints enables one to "decompose" a solution along the vertices of Z between consecutive checkpoints. We will argue that whenever Z is sufficiently small, the number of possible checkpoints will also be small.

▶ **Definition 19** (Semi-schedules). A *semi-schedule with respect to* (H, Z) is a semi-schedule where every vertex on every semi-route is contained in $V(H)$ and for every subtuple $(u, \perp, \ldots, \perp, v)$ in a semi-route, where $u, v \in V(H)$, it must be the case that $u, v \in Z$.

The intuition here is that whenever the robot "vanishes" or "reappears", it happens at the boundary Z. This allows us to analyze how a schedule interacts with the subgraph H since every movement between H and the rest of G must happen through the boundary Z.

▶ **Definition 20** (Signatures of Semi-schedules). The *signature* $\tau_0, \ldots, \tau_t$ of a semi-schedule with respect to (H, Z) is defined similarly to that of a schedule after making the assumption that $\perp$ denotes a vertex outside H, that is, we have the symbol $\uparrow$ in the j^{th} coordinate of tuple τ_i if and only if the robot R_j has $u_i^j = \perp$. A semi-schedule *externally affects* (H, Z) at time steps i and $i+1$ if some robot "moves" from $\perp$ to Z during time step $i+1$ or it "moves" from Z to $\perp$ during time step $i+1$. That is, for some robot R_j, we have $u_i^j = \perp$ and $u_{i+1}^j \in Z$ or we have $u_i^j \in Z$ and $u_{i+1}^j = \perp$. The notions of internally affecting (H, Z), affecting (H, Z), checkpoints with respect to (H, Z) are defined exactly as for schedules.

▶ **Definition 21** (Checkpoint Tuples). *Checkpoint tuples* of a schedule or semi-schedule S with respect to (H, Z) are defined as those configuration tuples for (H, Z) that appear in the signature of S at the time steps participating in checkpoints of S with respect to (H, Z). The *checkpoint tuple sequence* of S with respect to (H, Z) is the set of checkpoint tuples ordered according to their respective time steps.

Our dynamic programming algorithm will make use of the following observation about signatures of schedules and semi-schedules to narrow down the search space.

▶ **Observation 22.** *Let $\tau_0, \ldots, \tau_t$ be the signature of a schedule or semi-schedule with respect to (H, Z). Then, the following properties hold:*

1. *$\tau_0 = (s_1, \ldots, s_k)$. Moreover, the coordinates of τ_t that correspond to the robots in $\mathcal{M}$ are fixed, that is, $\tau_t = (t_1, \ldots, t_{|\mathcal{M}|}, v_1, \ldots, v_{k-|\mathcal{M}|})$ for some vertices $v_1, \ldots, v_{k-|\mathcal{M}|}$ distinct from $\{t_1, \ldots, t_{|\mathcal{M}|}\}$.*
2. *The first two and last two tuples in the signature are at checkpoints. That is, $(0, 1)$ and $(t-1, t)$ are checkpoints.*
3. *Let $(x, x+1), (y, y+1)$ be consecutive checkpoints such that $x+1 < y$. Then, the tuples $\tau_{x+2}, \ldots, \tau_{y-1}$ are identical.*
4. *Let $(x, x+1)$ be a checkpoint. There is a coordinate j such that $\tau_x[j] \neq \tau_{x+1}[j]$ and at least one of $\tau_x[j]$ or $\tau_{x+1}[j]$ is a vertex of Z.*
5. *Let $(x, x+1)$ be a checkpoint. There is no coordinate j such that $\tau_x[j] = \uparrow$ and $\tau_{x+1}[j] = \downarrow$ or $\tau_x[j] = \downarrow$ and $\tau_{x+1}[j] = \uparrow$.*
6. *For every time step x, τ_x contains at most one occurrence of any vertex of Z.*
7. *Let $(x, x+1)$ be a checkpoint and suppose that $\tau_x[j] \neq v \in Z$ and $\tau_{x+1}[j] = v$. Then, either there is no j' such that $\tau_x[j'] = v$ or it must be the case that $\tau_{x+1}[j'] \neq v$.*
8. *Parallel movements between vertices of the boundary Z in a single time step must happen along edges that exist and moreover, no edge can be used by two robots.*

Proof. The first two statements follow from the fact that the terminals are contained in Z. The third, fourth and fifth statements follow from the definition of checkpoints and the fact that the boundary is a separator between $V(H) \setminus Z$ and $V(G) \setminus V(H)$. The sixth and seventh statements are due to the fact that no vertex can be occupied by two robots at the same time. The eighth statement is due to the fact that a schedule or semi-schedule must be comprised of pairwise non-conflicting routes or semi-routes. ◀

▶ **Observation 23.** *Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of* GCMP. *Let (H, Z) be a p-boundaried subgraph of G with boundary Z containing all the terminals. If there is a schedule S for $\mathcal{I}$ in which each vertex is entered by any robot at most $\nu(k)$ times, then the following hold.*

1. *The number of checkpoints of S with respect to (H, Z) is bounded by $\mathcal{O}(pk \cdot \nu(k))$.*
2. *The number of possible checkpoint tuple sequences of S with respect to (H, Z) is bounded by $g(k, p) = p^{\mathcal{O}(pk \cdot \nu(k))}$.*

We now proceed by defining the partial solutions which play a central role in our algorithm.

▶ **Definition 24** (Partial Solutions)**.** Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be an instance of GCMP. Let (H, Z) be a p-boundaried subgraph of G with boundary Z containing all the terminals. Given an ordered set of configuration tuples $\gamma_1, \ldots, \gamma_q$ for (H, Z) where q is even, γ_1 is a starting configuration tuple and γ_q is an ending configuration tuple, a *partial solution corresponding to the sequence* $(\gamma_1, \gamma_2), \ldots, (\gamma_{q-1}, \gamma_q)$ is a semi-schedule S with respect to (H, Z) such that the checkpoint tuple sequence of S with respect to (H, Z) is exactly these tuples in this order. That is, the checkpoints of S with respect to (H, Z) are $(x_1, x_2), \ldots, (x_{q-1}, x_q)$ and $\tau_{x_i} = \gamma_i$ for each $i \in [q]$.

▶ **Theorem 2.** GCMP *is* FPT *parameterized by the number of robots and the treewidth of the input graph.*

Proof Sketch. Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be the given instance of GCMP. We start with a tree decomposition (T, β) of G of optimal width and add the terminals to every bag. Call the resulting tree decomposition (T, β') and let its width be w. By invoking Theorem 3 and setting $\nu(k) = \mathcal{O}(k^5)$ in Observation 23, we infer that the length of the checkpoint tuple sequence of S with respect to each boundaried graph $(G^{\downarrow}_{x,T}, \beta'(x))$ where $x \in V(T)$, is at most twice the number of checkpoints in this sequence. Hence, the length of this checkpoint tuple sequence is bounded by $\lambda(k, w) = \mathcal{O}(wk^6)$. Here, $G^{\downarrow}_{x,T}$ denotes the graph induced by the vertices that lie either in the bag $\beta'(x)$ or below it. Moreover, by the same observation, the number of possible checkpoint tuple sequences of S with respect to each boundaried graph $(G^{\downarrow}_{x,T}, \beta'(x))$ is at most $g(k, w+1) = w^{\mathcal{O}(wk^6)}$.

Based on this fact, our goal is to use dynamic programming to compute, for every $x \in V(T)$ and boundaried graph $(G^{\downarrow}_{x,T}, \beta'(x))$, and for every possible checkpoint tuple sequence of length at most $\lambda(k, w)$ with respect to this boundaried graph, the length of the best partial solution corresponding to this checkpoint tuple sequence within the boundaried graph. When x is the root node, $G^{\downarrow}_{x,T}$ is exactly the input graph G. Hence, the solution is given by the minimum entry in the table computed at the root node. Note that we only describe an algorithm to compute the length of an optimal solution. However, it is straightforward to see that an optimal schedule can also be produced in the same running time. ◀

6 Using the Total Energy as the Parameter

In the final part of our paper, we consider the complexity of GCMP in settings where we are dealing with many robots but the energy budget ℓ is small. While intuitively this may seem like an "easier" case due to the fact that ℓ immediately bounds the number of robots

that will end up moving in a hypothetical schedule, we show that the problem (and even its restriction to the often-studied case where all robots have destinations) is unlikely to be fixed-parameter tractable when parameterized by ℓ.

▶ **Theorem 4.** GCMP *is* W[1]*-hard when parameterized by* ℓ.

Proof. We provide a parameterized reduction from the well-known W[1]-complete Multicolored Clique problem [9, 13]: decide whether a given κ-partite graph $(V_1, \ldots, V_\kappa, E)$ contains a clique of size κ. To avoid any confusion, we explicitly remark that in this proof we use κ to denote the parameter of the initial instance of Multicolored Clique and not the number of robots in the resulting instance of GCMP.

Our reduction takes an instance $(V_1, \ldots, V_\kappa, E)$ of Multicolored Clique and constructs an instance $(G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), \kappa', \ell)$ of GCMP as follows. To obtain G, we:

1. subdivide each edge $e \in E$ κ^3 many times;
2. attach a single new pendant vertex v' to each vertex $v \in V_1 \cup \cdots \cup V_\kappa$ in the original graph; and
3. for each pair of integers i, j such that $1 \leq i < j \leq \kappa$, construct a new vertex $s_{i,j}$ and make it adjacent to every vertex in V_i, and construct a new vertex $t_{i,j}$ and make it adjacent to every vertex in V_j.

For $\mathcal{R} = (\mathcal{M}, \mathcal{F})$, we set $\mathcal{F} = \emptyset$ and add two sets of robots into $\mathcal{M}$. First, for each vertex v in $V_1 \cup \cdots \cup V_\kappa$, we construct a *blocking* robot r^v and set v as its starting point as well as its destination. Second, for each of the newly constructed $s_{i,j} \in V(G)$, we construct a *clique* robot $r^{i,j}$, set $s_{i,j}$ as its starting point and $t_{i,j}$ as its destination. Finally, we set $\ell := 2\kappa + \binom{\kappa}{2} \cdot (\kappa^3 + 3)$ and $\kappa' := |\mathcal{R}|$. This completes the description of our reduction.

Towards proving correctness, we first observe that a hypothetical schedule for $(G, \mathcal{R}, \kappa', \ell)$ can never use less than $\ell := 2\kappa + \binom{\kappa}{2} \cdot (\kappa^3 + 3)$ travel length. Indeed, there are $\binom{\kappa}{2}$ many clique robots, and the shortest path between their starting and destination points has length $\kappa^3 + 3$. Moreover, every schedule must route each clique robot $r^{i,j}$ through at least one vertex in V_i and at least one vertex in V_j, and hence a hypothetical solution must spend a length of at least 2 to move at least one of the blocking robots in each V_p, $p \in [\kappa]$, out of the way and then back to its initial position (which is also its destination).

Now, assume that $(G, \mathcal{R}, \kappa', \ell)$ is a yes-instance. By the argument above, this implies that there is a schedule which moves precisely one of the blocking robots in each V_p, $p \in [\kappa]$; let us denote by w_p the starting vertex of the unique blocking robot which is moved from V_p. Since $(G, \mathcal{R}, \kappa', \ell)$ is a yes-instance, E must contain an edge between each w_i and w_j, $1 \leq i < j \leq \kappa$; in particular, these vertices induce a clique in $(V_1, \ldots, V_\kappa, E)$.

For the converse, assume that $(V_1, \ldots, V_\kappa, E)$ contains a κ-clique $w_1, \ldots, w_\kappa$. We construct a solution for $(G, \mathcal{R}, \kappa', \ell)$ as follows. First, for each robot starting at w_i, $i \in [\kappa]$, we spend a length of 1 to move it to the pendant vertex w_i'. Next, we route each robot $r^{i,j}$ from $s_{i,j}$ to w_i, through the $(\kappa^3 + 1)$-length path to w_j (whereas the existence of this path is guaranteed by the existence of the edge $w_i w_j \in E$), and then to its destination $t_{i,j}$. Finally, we route each blocking robot which originally started at w_i back to w_i from w_i'. The travelled length of this schedule is precisely $2\kappa + \binom{\kappa}{2} \cdot (\kappa^3 + 3)$, as desired. ◀

As mentioned in the introduction, we complement Theorem 4 with a fixed-parameter algorithm on graph classes of bounded local treewidth.

Before proceeding to the proof, we recall the syntax of Monadic Second Order Logic (MSO) of graphs. It contains the logical connectives $\vee$, $\wedge$, $\neg$, $\Leftrightarrow$, $\Rightarrow$, variables for vertices, edges, sets of vertices and sets of edges, and the quantifiers $\forall$ and $\exists$, which can be applied to these variables. It also contains the following binary relations: (i) $u \in U$, where u is a vertex

variable and U is a vertex set variable; (ii) $d \in D$, where d is an edge variable and D is an edge set variable; (iii) $\mathbf{inc}(d, u)$, where d is an edge variable, u is a vertex variable, with the interpretation that the edge d is incident to u; (iv) equality of variables representing vertices, edges, vertex sets and edge sets.

The well-known Courcelle's theorem [2, 8] states that checking whether a given graph G models a given MSO formula ϕ can be done in FPT time parameterized by the treewidth of G and the size of ϕ. Morover, this result holds even if some vertices of G are given labels or colors (i.e., we allow a fixed number of additional unary relations over $V(G)$). This is because one can produce an equivalent graph G' such that G has bounded treewidth if and only if G' does, plus an alternate MSO formula ϕ' such that G models ϕ if and only if G' models ϕ'.

▶ **Theorem 5.** GCMP *is* FPT *parameterized by* ℓ *on graph classes of bounded local treewidth.*

Proof. Let $\mathcal{I} = (G, \mathcal{R} = (\mathcal{M}, \mathcal{F}), k, \ell)$ be the given instance of GCMP where G belongs to a graph class that is closed under taking subgraphs and in which the treewidth of a graph of diameter d is bounded by $f(d)$ for some function f. Let $\mathcal{M}'$ denote the set of those robots in $\mathcal{M}$ whose final position is distinct from the starting position. Since each of the robots in $\mathcal{M}'$ must move at least one step, it follows that if $|\mathcal{M}'| > \ell$, then the instance is a no-instance. So, assume that $|\mathcal{M}'| \leq \ell$. Let U denote the set of starting points of the robots in $\mathcal{M}'$.

Consider an optimal schedule S. We say that an edge e appears in a route $W = (u_0, \ldots, u_t)$ in S if the endpoints of e are the vertices u_{j-1} and u_j for some $j \in [t]$. We say that e appears in S if it appears in some route in S. Let E_S denote the set of edges of G that appear in S. We observe that every connected component of the subgraph of G induced by E_S (call it G_S) contains a vertex of U. If this were not the case and there is a connected component of G_S disjoint from U, then we can delete all the routes whose edges appear in this connected component and still have a feasible schedule for the given instance, contradicting the optimality of S. Moreover, since S has total energy at most ℓ, it follows that at most ℓ edges can appear in S. Hence, it follows that S is contained in the subgraph H defined as the union of the subgraphs induced by the ℓ-balls centered at the starting positions of the robots in $\mathcal{M}'$.

Notice that since H is obtained by taking the union of at most ℓ graphs, each of diameter at most 2ℓ, it follows that H has diameter at most $2\ell^2$ and so, the treewidth of H is upper bounded by $f(2\ell^2)$. It remains to argue that GCMP is FPT parameterized by the energy and treewidth of the input graph. We do this by making some guesses, obtaining a bounded number of MSO formulas such that input is a yes-instance if and only if at least one of these formulas is true on the graph H with a label function (with constant number of labels) and then invoking Courcelle's theorem [2, 8]. Let $\mathcal{M}''$ denote the subset of $\mathcal{M} \setminus \mathcal{M}'$ contained in H and let $\mathcal{R}' = \mathcal{R} \cap V(H)$.

Suppose that $\alpha \leq \ell$ robots move in S and let the routes in S be $\{W_i = (v_1^i, \ldots, v_{d_i}^i) \mid i \in [\alpha]\}$. Assume that the first $|\mathcal{M}'|$ routes, $W_1, \ldots, W_{|\mathcal{M}'|}$, correspond to the robots in $\mathcal{M}'$. We guess α and $d_1, \ldots, d_\alpha$ as these are all at most ℓ. Moreover, we guess, for every pair of vertices v_j^i and v_q^p whether they are equal. Let this be expressed by a function $\mu(v_j^i, v_q^p)$ that evaluates to 1 if and only if these vertices are guessed to be equal. Thus, even though we do not know the actual vertices in the solution (except for the starting and endpoint positions of the robots in $\mathcal{M}'$), this guess gives us the "structure" of the entire solution. Notice that the number of guesses is bounded by a function of ℓ. Now, we consider the labeled graph H where the starting points of all robots in $\mathcal{R}' \setminus \mathcal{M}''$ are labeled red and the starting points of the robots in $\mathcal{M}''$ are labeled blue. Fix a guess of α, the numbers $d_1, \ldots, d_\alpha$ and a guess of the function μ and express the following as an MSO formula.

There are ordered vertex sets $\{W_i = (v^i_1, \ldots, v^i_{d_i}) \mid i \in [\alpha]\}$ such that the following hold:

1. Each W_i is a walk in H.
2. For every i, j, p, q, v^i_j and v^p_q are equal if and only if $\mu(v^i_j, v^p_q)$ is 1.
3. The walks W_i are pairwise non-conflicting routes.
4. For each of the first $|\mathcal{M}'|$ walks $W_1, \ldots, W_{|\mathcal{M}'|}$, the initial and final vertices are precisely the starting and ending points, respectively, of some robot in $\mathcal{M}'$.
5. For each labeled vertex appearing on any walk W_i, this vertex must be the first vertex of some walk W_i.
6. For each walk W_i starting at a blue vertex, it must terminate at the same blue vertex.

Expressing the above in MSO is straightforward to do, so we do not go in to lower level details of the MSO formula.

We next argue that the input is a yes-instance if and only if there is a guess of α, the numbers $d_1, \ldots, d_\alpha$ and a guess of the function μ such that the resulting MSO formula is true on H. Consider the forward direction. An optimal schedule S with routes $W_1, \ldots, W_\alpha$ naturally gives us a "correct" guess of α, $d_1, \ldots, d_\alpha$ and the function μ. Moreover notice that Properties 1-4 are clearly satisfied. For Property 5, notice that only the starting points of robots that never move can be disjoint from the set of initial vertices of the routes $\{W_i \mid i \in [\alpha]\}$ in the optimal schedule S. Since these robots do not move, they also cannot appear on any route W_i or they would obstruct a robot that moves. Finally, Property 6 is satisfied since every robot in $\mathcal{M}''$ that moves also ends up returning to its starting point. The converse is trivial. ◀

7 Conclusion

In this paper, we presented parameterized algorithms for fundamental coordinated motion planning problems on graphs. In particular, we proved that GCMP1 is fixed-parameter tractable parameterized by the number of robots, and that GCMP is fixed-parameter tractable parameterized by the number k of robots and the treewidth of the input graph combined. This latter result implies that GCMP is fixed-parameter tractable in several graph classes which may be of interest w.r.t. application domains, including graphs of bounded outerplanarity.

We conclude by highlighting three directions that may be pursued in future works:

1. Is GCMP FPT parameterized by k (alone) on planar graphs, or even on general graphs?
2. What is the complexity of GCMP on trees? While GCMP1 is known to be in P on trees, the complexity of the general problem on trees remains unresolved.
3. The focus of this paper was on minimizing the travel length/energy, which is one of the most-studied optimization objectives. Nevertheless, there are other important objectives, most notably that of minimizing the makespan. It seems that at least in the settings considered here, optimizing the makespan may be a computationally more challenging problem. In particular, the question of whether Coordinated Motion Planning on trees is FPT parameterized by k remains open.

References

1 Aviv Adler, Mark de Berg, Dan Halperin, and Kiril Solovey. Efficient multi-robot motion planning for unlabeled discs in simple polygons. *IEEE Transactions on Automation Science and Engineering*, 12(4):1309–1317, 2015.

2 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.

3 Jacopo Banfi, Nicola Basilico, and Francesco Amigoni. Intractability of time-optimal multirobot path planning on 2D grid graphs with holes. *IEEE Robotics and Automation Letters*, 2(4):1941–1947, 2017.

4 Bahareh Banyassady, Mark de Berg, Karl Bringmann, Kevin Buchin, Henning Fernau, Dan Halperin, Irina Kostitsyna, Yoshio Okamoto, and Stijn Slot. Unlabeled multi-robot motion planning with tighter separation bounds. In *SoCG*, volume 224, pages 12:1–12:16, 2022.

5 Édouard Bonnet, Jan Dreier, Jakub Gajarský, Stephan Kreutzer, Nikolas Mählmann, Pierre Simon, and Szymon Torunczyk. Model checking on interpretations of classes of bounded local cliquewidth. In Christel Baier and Dana Fisman, editors, *LICS*, pages 54:1–54:13, 2022.

6 Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, David Tolpin, Oded Betzalel, and Solomon Eyal Shimony. ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. In *IJCAI*, pages 740–746, 2015.

7 Josh Brunner, Lily Chung, Erik D. Demaine, Dylan H. Hendrickson, Adam Hesterberg, Adam Suhl, and Avi Zeff. 1×1 rush hour with fixed blocks is PSPACE-complete. In *FUN*, volume 157, pages 7:1–7:14, 2021.

8 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

10 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Henk Meijer, and Christian Scheffer. Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. *SIAM Journal on Computing*, 48(6):1727–1762, 2019.

11 Erik D. Demaine and Mikhail Rudoy. A simple proof that the $(n^2 - 1)$-puzzle is hard. *Theoretical Computer Science*, 732:80–84, 2018.

12 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

13 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

14 Adrian Dumitrescu. Motion planning and reconfiguration for systems of multiple objects. In Sascha Kolski, editor, *Mobile Robots*, chapter 24. IntechOpen, Rijeka, 2007.

15 Eduard Eiben, Robert Ganian, and Iyad Kanj. The parameterized complexity of coordinated motion planning. In *SoCG*, volume 258, pages 28:1–28:16, 2023.

16 David Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000.

17 Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. Computing coordinated motion plans for robot swarms: The CG: SHOP challenge 2021. *ACM Journal on Experimental Algorithmics*, 27:3.1:1–3.1:12, 2022.

18 Foivos Fioravantes, Dusan Knop, Jan Matyás Kristan, Nikolaos Melissinos, and Michal Opler. Exact algorithms and lowerbounds for multiagent pathfinding: Power of treelike topology. *CoRR*, abs/2312.09646, 2023.

19 Gary William Flake and Eric B. Baum. Rush hour is PSPACE-complete, or "why you should generously tip parking lot attendants". *Theoretical Computer Science*, 270(1-2):895–911, 2002.

20 Tzvika Geft and Dan Halperin. Refined hardness of distance-optimal multi-agent path finding. In *AAMAS*, pages 481–488, 2022.

21 Oded Goldreich. *Finding the shortest move-sequence in the graph-generalized 15-puzzle is NP-hard*. Springer, 2011.

22 Martin Grohe. Local tree-width, excluded minors, and approximation algorithms. *Combinatorica*, 23(4):613–632, 2003.

23 Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

24 John E. Hopcroft, Wolfgang J. Paul, and Leslie G. Valiant. On time versus space and related problems. In *FOCS*, pages 57–64. IEEE Computer Society, 1975.

25 John E. Hopcroft, Jacob T. Schwartz, and Micha Sharir. On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the "Warehouseman's Problem. *The International Journal of Robotics Research*, 3(4):76–88, 1984.

26 Paul Liu, Jack Spalding-Jamieson, Brandon Zhang, and Da Wei Zheng. Coordinated motion planning through randomized k-Opt (CG challenge). In *SoCG*, volume 189, pages 64:1–64:8, 2021.

27 Christos H. Papadimitriou, Prabhakar Raghavan, Madhu Sudan, and Hisao Tamaki. Motion planning on a graph (extended abstract). In *STOC*, pages 511–520, 1994.

28 Geetha Ramanathan and Vangalur S. Alagar. Algorithmic motion planning in robotics: Coordinated motion of several disks amidst polygonal obstacles. In *ICRA*, volume 2, pages 514–522, 1985.

29 Daniel Ratner and Manfred Warmuth. The $(n^2 - 1)$-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111–137, 1990.

30 John H Reif. Complexity of the mover's problem and generalizations. In *FOCS*, pages 421–427, 1979.

31 Oren Salzman and Roni Stern. Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems. In *AAMAS*, pages 1711–1715, 2020.

32 Jacob T. Schwartz and Micha Sharir. On the piano movers' problem: III. coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers. *The International Journal of Robotics Research*, 2:46–75, 1983.

33 Jacob T. Schwartz and Micha Sharir. On the "piano movers'" problem I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on Pure and Applied Mathematics*, 36(3):345–398, 1983.

34 Jacob T. Schwartz and Micha Sharir. On the "piano movers" problem. II. General techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4(3):298–351, 1983.

35 Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

36 Irving Solis, James Motes, Read Sandström, and Nancy M. Amato. Representation-optimal multi-robot motion planning using conflict-based search. *IEEE Robotics Automation Letters*, 6(3):4608–4615, 2021.

37 Kiril Solovey and Dan Halperin. On the hardness of unlabeled multi-robot motion planning. *The International Journal of Robotics Research*, 35(14):1750–1759, 2016.

38 Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015.

39 Jingjin Yu and Steven M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*, pages 1443–1449, 2013.

40 Jingjin Yu and Steven M. LaValle. Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, 32(5):1163–1177, 2016.

41 Jingjin Yu and Daniela Rus. Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms. In *WAFR*, volume 107 of *Springer Tracts in Advanced Robotics*, pages 729–746, 2014.

Nearly Optimal Independence Oracle Algorithms for Edge Estimation in Hypergraphs

Holger Dell
Goethe University Frankfurt, Germany
IT University of Copenhagen and Basic Algorithms Research Copenhagen (BARC), Denmark

John Lapinskas
University of Bristol, UK

Kitty Meeks
University of Glasgow, UK

Abstract

Consider a query model of computation in which an n-vertex k-hypergraph can be accessed only via its independence oracle or via its colourful independence oracle, and each oracle query may incur a cost depending on the size of the query. Several recent results (Dell and Lapinskas, STOC 2018; Dell, Lapinskas, and Meeks, SODA 2020) give efficient algorithms to approximately count the hypergraph's edges in the colourful setting. These algorithms immediately imply fine-grained reductions from approximate counting to decision, with overhead only $\log^{\Theta(k)} n$ over the running time n^{α} of the original decision algorithm, for many well-studied problems including k-Orthogonal Vectors, k-SUM, subgraph isomorphism problems including k-Clique and colourful-H, graph motifs, and k-variable first-order model checking.

We explore the limits of what is achievable in this setting, obtaining unconditional lower bounds on the oracle cost of algorithms to approximately count the hypergraph's edges in both the colourful and uncoloured settings. In both settings, we also obtain algorithms which essentially match these lower bounds; in the colourful setting, this requires significant changes to the algorithm of Dell, Lapinskas, and Meeks (SODA 2020) and reduces the total overhead to $\log^{\Theta(k-\alpha)} n$. Our lower bound for the uncoloured setting shows that there is no fine-grained reduction from approximate counting to the corresponding uncoloured decision problem (except in the case $\alpha \geq k-1$): without an algorithm for the colourful decision problem, we cannot hope to avoid the much larger overhead of roughly $n^{(k-\alpha)^2/4}$. The uncoloured setting has previously been studied for the special case $k = 2$ (Peled, Ramamoorthy, Rashtchian, Sinha, ITCS 2018; Chen, Levi, and Waingarten, SODA 2020), and our work generalises the existing algorithms and lower bounds for this special case to $k > 2$ and to oracles with cost.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → Oracles and decision trees; Mathematics of computing → Graph algorithms

Keywords and phrases Graph oracles, Fine-grained complexity, Approximate counting, Hypergraphs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.54

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2211.03874` [14]

Funding *Kitty Meeks*: Supported by EPSRC grant EP/V032305/1.

1 Introduction

Many decision problems in computer science, particularly those in NP, can naturally be expressed in terms of determining the existence of a witness. For example, solving SAT requires determining the existence of a satisfying assignment to a CNF formula. All such problems Π naturally give rise to a counting version #Π, in which we ask for the number of

witnesses. It is well-known that #Π is often significantly harder than Π; for example, Toda's theorem implies that it is impossible to solve #P-complete counting problems in polynomial time with access to an NP-oracle unless the polynomial hierarchy collapses. However, the same is not true for *approximately* counting witnesses (to within a factor of two, say). For example, it is known that: if Π is a problem in NP, then there is an FPRAS for #Π using an NP-oracle [30]; if Π is a problem in $W[i]$, then there is an FPTRAS for #Π using a $W[i]$-oracle [25]; and that the Exponential Time Hypothesis is equivalent to the statement that there is no subexponential-time approximation algorithm for #3-SAT [12].

In this paper we are concerned with analogous results in the fine-grained setting, which considers exact running times rather than coarse-grained classifications such as polynomial, FPT, or subexponential; such results turn out to be inextricably bound to graph oracle results of independent interest.

Past work in this area has focused on the family of *uniform witness problems* [13]. Roughly speaking, these are problems which can be expressed as counting edges in a k-hypergraph G in which the edges correspond to witnesses and induced subgraphs correspond to sub-problems. (See Section 3 for a detailed definition.) Many of the most important problems in fine-grained and parameterised complexity can be expressed as uniform witness problems including k-SUM, k-OV, k-Clique, Hamming weight-k solutions to CNFs, Size-k Graph Motif, most subgraph detection problems (including weighted problems such as Zero-Weight k-Clique and Negative-Weight Triangle), and first-order model-checking [13], in addition to certain database queries [16] and patterns in graphs [9]. Here k may be either a constant, as in the case of k-SUM, or a parameter, as in the case of k-Clique. In this setting, invoking a decision algorithm on a sub-problem of the original problem corresponds to invoking an oracle to test, given a set of vertices S, whether the induced subgraph $G[S]$ contains any edges; this oracle is called an *independence oracle* for G and is well-studied in its own right (see Section 4 for an overview).

Surprisingly, there is a partial analogue of the above reductions from approximate counting to decision in this setting. If the vertices of G are coloured, given a set $S \subseteq V(G)$, a *colourful independence oracle* tests whether $G[S]$ contains any edges with one vertex of each colour. This typically corresponds to a natural colourful variant of the original decision problem – for example, for k-Clique, it corresponds to deciding whether a k-coloured graph contains a size-k clique with one vertex of each colour. These oracles are again well-studied in their own right (see Section 4), and for many but not all uniform witness problems they can be efficiently simulated using the independence oracle. Given access to a colourful independence oracle for a graph G, we can count G's edges to within a factor of $1 \pm \varepsilon$ using $\varepsilon^{-2}k^{\mathcal{O}(k)} \log^{\Theta(k)} n$ oracle queries [13]. (See [4] for an improvement to the log factor.) In fact, we can say more – if we can simulate the colourful independence oracle in time n^{α_k} with $\alpha_k \geq 1$, then these queries dominate the running time and we obtain an approximate counting algorithm with running time $n^{\alpha_k} \cdot \varepsilon^{-2}k^{\mathcal{O}(k)} \log^{\Theta(k)} n$ in the usual word-RAM model. Translating back out of the oracle setting, this means that if we simulate the oracle by running an algorithm for the colourful decision problem, then for constant k and ε, we obtain an approximate counting algorithm with only polylogarithmic overhead over that decision algorithm. This result has led to several improved approximate counting algorithms – see [13] for applications to k-OV over finite fields and graph motifs, [16] for applications to database queries, and [9] for applications to patterns in graphs.

We are left with two major open problems of concern to researchers in fine-grained complexity, parameterised complexity and graph oracles, and we expect our paper to be of interest to all three communities. First, can the result of [13] be generalised from colourful

independence oracles to independence oracles? This would imply, for example, a fine-grained reduction from approximate induced sub-hypergraph counting to induced sub-hypergraph detection. In this setting, efficiently simulating the colourful independence oracle using the independence oracle requires solving a long-standing open problem – see Section 3 – so the result of [13] does not straightforwardly apply. Second, in the parameterised setting, the factor of $\log^{\Theta(k)} n$ is not truly polylogarithmic, but equivalent to a factor of $k^{\mathcal{O}(k)}n^{o(1)}$. Can it be improved to $\log^{\mathcal{O}(1)} n$?

In this paper, we answer both questions, and in the process substantially generalise recent graph oracle results for the $k = 2$ case [11]. In both the colourful and uncoloured settings, we pin down the optimal oracle algorithm almost exactly. In both cases this algorithm improves on the current state of the art, and it allows for the desired fine-grained reductions if and only if the cost of calling the oracle on an x-vertex set (corresponding to the run-time of a decision algorithm on an x-element instance) is close to x^k. Moreover, our lower bounds are unconditional – they do not rely on conjectures such as SETH or FPT $\neq$ W[1].

In a little more detail, suppose for the moment that $\varepsilon = 1/2$, and that the cost of calling the oracle on an x-vertex set is x^{α_k} for some $\alpha_k \in [0, k]$. In the uncoloured setting, we define a function $g(k, \alpha_k) \approx (k - \alpha)^2/(4k)$ (see (2.1.1)) and show that an overhead of $2^{\mathcal{O}(k)}n^{g(k,\alpha_k)\pm o(1)}$ is both achievable and required; we have $g(k, \alpha_k) = 0$ when $\alpha_k \geq k - 1$, so in this regime we obtain a fine-grained reduction. In the colourful setting, we show that the $\log^{\Theta(k)} n$ overhead of [13, 4] can be improved to $\log^{\Theta(k-\alpha_k)} n$, but no further; thus polylogarithmic overhead is possible if and only if $k - \alpha_k \in \mathcal{O}(1)$ as $k \to \infty$. For general values of ε, both of our upper bounds have an additional multiplicative overhead of $\mathcal{O}(\varepsilon^{-2})$, which is common in approximate counting algorithms.

In the rest of the paper, we state our results for graph oracles more formally in Section 2, followed by their (immediate) corollaries for uniform witness problems in Section 3. We then give an overview of related work in Section 4, followed by a brief description of our proof techniques in Section 5 and a generalisation to non-polynomial running times in Section 6. A full version of the paper is available as [14].

2 Oracle results

Our results are focused on two graph oracle models on k-hypergraphs: independence oracles and colourful independence oracles. Both oracles are well-studied in their own right from a theoretical perspective, as they are both natural generalisations of group testing from unary relations to k-ary relations, and the apparent separation between them in power is already a source of substantial interest. They also provide a point of comparison for a rich history of sublinear-time algorithms for oracles which provide more local information, such as degree oracles. See the introduction of [11] for a more detailed overview of the full motivation, and Section 4 for a survey of past results.

In both the colourful and uncoloured case, while formally the oracles are bitstrings and a query takes $\mathcal{O}(1)$ time, in order to obtain reductions from approximate counting problems to decision problems in Section 3 we will simulate oracle queries using a decision algorithm. As such, rather than focusing on the *number* of queries as a computational resource, we define a more general *cost function* which will correspond to the running time of the algorithm used to simulate the query; thus the cost of a query will scale with its size. In our application, this allows for more efficient reductions by exploiting cheap queries, while also substantially strengthening our lower bounds. Indeed, simulating an oracle query typically requires between $\text{poly}(k)$ and $\text{poly}(n)$ time, so a lower bound on the total number of queries required would tell us very little; meanwhile, setting the cost of all queries to 1 in our results yields tight bounds for the number of queries required.

We are also concerned with the *running times* of our oracle algorithms, again due to our applications in Section 3. We work in the standard RAM-model of computation with $\Theta(\log n)$ bits per word and access to the usual $\mathcal{O}(1)$-time arithmetic and logical operations on these words; in addition, oracle algorithms can perform oracle queries, which are considered to take $\mathcal{O}(1)$ time.

As shorthand, for all real $x, y > 0$ and $\varepsilon \in (0,1)$, we say that x is an *ε-approximation* to y if $|x - y| < \varepsilon y$. We define an *ε-approximate counting algorithm* to be an oracle algorithm that is given n and k as explicit input, is given access to an oracle representing an n-vertex k-hypergraph G, and outputs an ε-approximation to the number of edges of G, denoted by $e(G)$. We allow ε to be part of the input (for upper bounds) or fixed (for lower bounds).

2.1 Our results for the uncoloured independence oracle

Given a k-hypergraph G with vertex set $[n]$, the *(uncoloured) independence oracle* is the bitstring $\texttt{IND}(G)$ such that for all sets $S \subseteq [n]$, $\texttt{IND}(G)_S = 1$ if $G[S]$ contains no edges and 0 otherwise. Thus a query to $\texttt{IND}(G)_S$ allows us to test whether or not the induced subgraph $G[S]$ contains an edge. We define the *cost* of an oracle call $\texttt{IND}(G)_S$ to be a polynomial function of the form $\mathrm{cost}_k(S) = |S|^{\alpha_k}$, where the map $k \mapsto \alpha_k$ satisfies $\alpha_k \in [0, k]$ but is otherwise arbitrary. (This upper bound is motivated by the fact that we can trivially enumerate all edges of G by using $\mathcal{O}(n^k)$ queries to all size-k subsets of $[n]$, incurring oracle cost at most $n^k \cdot k^{\alpha_k}$.)

It is not too hard to show that the naive $\mathcal{O}(n^k)$-cost exact edge-counting algorithm of querying every possible edge and the naive $\mathcal{O}(n^{\alpha_k})$-cost algorithm to decide whether any edge is present by querying $[n]$ are both essentially optimal. For approximate counting we prove the following, where for all real numbers x we write $\lfloor x \rceil \coloneqq \lfloor x + 1/2 \rfloor$ for the value of x rounded to the nearest integer, rounding up in case of a tie.

▶ **Theorem 1** (Uncoloured independence oracle, polynomial cost function)**.** *Let $\alpha_k \in [0, k]$ for all $k \geq 2$, let $\mathrm{cost}_k(x) = x^{\alpha_k}$, and let*

$$g(k, \beta) \coloneqq \frac{1}{k} \cdot \left\lfloor \frac{k-\beta}{2} \right\rceil \cdot \left(k - \beta - \left\lfloor \frac{k-\beta}{2} \right\rceil \right). \tag{2.1.1}$$

There is a randomised ε-approximate counting algorithm $\texttt{Uncol}(\texttt{IND}(G), \varepsilon, \delta)$ with failure probability at most δ, worst-case running time

$$\mathcal{O}\Big(\log(1/\delta) \big(k^{5k} + \varepsilon^{-2} 2^{5k} \log^5 n \cdot n^{g(k,1)} \cdot n \big) \Big),$$

and worst-case oracle cost

$$\mathcal{O}\Big(\log(1/\delta) \big(k^{7k} + \varepsilon^{-2} 2^{5k} \log^5 n \cdot n^{g(k,\alpha_k)} \cdot n^{\alpha_k} \big) \Big)$$

under cost_k. Moreover, every randomised $(1/2)$-approximate edge-counting $\texttt{IND}$-oracle algorithm with failure probability at most $1/10$ has $\Omega((n^{g(k,\alpha_k)}/k^{3k}) \cdot n^{\alpha_k})$ worst-case expected oracle cost under cost_k.

Observe that the polynomial overhead $n^{g(k,\alpha_k)}$ of approximate counting over decision is roughly equal to $n^{(k-\alpha_k)^2/(4k)}$. If $\alpha_k = 0$, then the worst-case oracle cost of an algorithm is simply the worst-case number of queries that it makes. Thus Theorem 1 generalises known matching upper and lower bounds of $\widetilde{\Theta}(\sqrt{n})$ queries in the graph case [11], both by allowing $k > 2$ and by allowing $\alpha_k > 0$. (See Section 4 for more details.) Moreover, if $\alpha_k \geq k - 1$,

then $g(k, \alpha_k) = 0$; thus in this case, Theorem 1 shows that approximate counting requires the same oracle cost as decision, up to a polylogarithmic factor. Taking $k = 2$ and $\alpha_k = 1$, this implies that whenever we can simulate an edge-detection oracle for a graph in linear time, then we can also obtain a linear-time approximate edge-counting algorithm (up to polylogarithmic factors). Analogous upper bounds on the running time and oracle cost of `Uncol` also hold for any "reasonable" cost function of the form $\text{cost}_k(n) = n^{\alpha_k + o(1)}$; for details, see Section 6 and Theorem 10.

2.2 Our results for the colourful independence oracle

Given a k-hypergraph G with vertex set $[n]$, the *colourful independence oracle* is the bitstring $\texttt{cIND}(G)$ such that for all disjoint sets $S_1, \dots, S_k \subseteq [n]$, $\texttt{cIND}(G)_{S_1,\dots,S_k} = 1$ if G contains no edge $e \in E(G)$ with $|S_i \cap e| = 1$ for all i, and 0 otherwise. We view $S_1, \dots, S_k$ as colour classes in a partial colouring of $[n]$; thus a query to $\texttt{cIND}(G)_{S_1,\dots,S_k}$ allows us to test whether or not G contains an edge with one vertex of each colour. (Note that we do not require $S_1 \cup \dots \cup S_k = [n]$.) Analogously to the uncoloured case, we define the *cost* of an oracle call $\texttt{cIND}(G)_{S_1,\dots,S_k}$ to be a polynomial function of the form $\text{cost}_k(S_1, \dots, S_k) = \text{cost}_k(|S_1| + \dots + |S_k|) = (|S_1| + \dots + |S_k|)^{\alpha_k}$, where the map $k \mapsto \alpha_k$ satisfies $\alpha_k \in [0, k]$ but is otherwise arbitrary.

It is not too hard to show that the naive $\mathcal{O}(n^k)$-cost exact edge-counting algorithm of querying every possible edge and the naive $\mathcal{O}((k^k/k!)n^{\alpha_k})$-cost algorithm to decide whether any edge is present by randomly colouring the vertices are both essentially optimal, and indeed we prove as Proposition 65 of the full version that any such decision algorithm requires cost $\Omega(n^{\alpha_k})$. For approximate counting, we prove the following.

▶ **Theorem 2** (Colourful independence oracle, polynomial cost function). *Let $\alpha_k \in [0, k]$ for all $k \geq 2$, let $\text{cost}_k(x) = x^{\alpha_k}$, and let $T := \log(1/\delta)\varepsilon^{-2}k^{27k}\log^{4(k-\lceil\alpha_k\rceil)+18} n$. There is a randomised ε-approximate edge counting algorithm $\texttt{Count}(\texttt{cIND}(G), \varepsilon, \delta)$ with worst-case running time $\mathcal{O}(T \cdot n^{\max(1,\alpha_k)})$, worst-case oracle cost $\mathcal{O}(T \cdot n^{\alpha_k})$ under cost_k, and failure probability at most δ. Moreover, every randomised $(1/2)$-approximate edge counting $\texttt{cIND}$-oracle algorithm with failure probability at most $1/10$ has worst-case oracle cost*

$$\Omega\left(k^{-9k}\left(\frac{\log n}{\log\log n}\right)^{k-\lfloor\alpha_k\rfloor-3} \cdot n^{\alpha_k}\right)$$

under cost_k.

The upper bound replaces a $\log^{\Theta(k)} n$ term in the query count of the previous best-known algorithm ([5] for $\alpha_k = 0$) by a $\log^{\Theta(k-\alpha_k)} n$ term in the multiplicative overhead over decision, giving polylogarithmic overhead over decision when $k - \alpha_k = \mathcal{O}(1)$. The lower bound shows that this term is necessary and cannot be reduced to $\log^{\mathcal{O}(1)} n$; this is a new result even for $\alpha_k = 0$. (See Section 4 for more details.) Analogous upper bounds on the running time and oracle cost of `Count` also hold for any "reasonable" cost function of the form $\text{cost}_k(n) = n^{\alpha_k + o(1)}$; for details, see Section 6 and Theorem 10.

2.3 Approximate sampling results

There is a known fine-grained reduction from approximate sampling to approximate counting [13]. Strictly speaking this, reduction is proved for $\alpha_k = 1$ with a colourful independence oracle, but the only actual use of the oracle in the reduction is to enumerate all edges in a set X with $\mathcal{O}(|X|^k)$ size-k queries, so it transfers immediately to our setting. The upper bounds of Theorems 1 and 2 therefore also yield approximate sampling algorithms with overhead $2^{\mathcal{O}(k)}\log^{\mathcal{O}(1)} n$ over approximate counting.

2.4 A parameterised complexity motivation for our lower bound results

To understand an important motivation for the lower bounds in our results, consider as an example the longest path problem: Given (G, k), does there exist a simple path of length k? A long sequence of works in parameterised complexity led to a spectacular algorithm [7] for this problem in undirected graphs that runs in time $1.66^k \cdot \text{poly}(n)$. There is a somewhat shorter sequence of works for the corresponding approximate counting version of the problem, which culminated in a $4^k \text{poly}(n)$-time algorithm [8, 20].

Instead of designing ever-more sophisticated algorithms for approximately counting k-paths in order to get closer to the running time of the decision problem, our dream result would instead be a subexponential-time approximate-counting-to-decision reduction that uses the decision problem in a black-box fashion and causes only a factor $2^{o(k)} \text{poly}(n)$ overhead in the running time. This way, any improvements to the decision algorithm would automatically carry over. One way to formalise what the black-box can do is captured by defining the k-hypergraph whose edges are the k-paths of the underlying graphs; using an algorithm for the k-path decision problem, it is trivial to simulate the independence oracle and easy to simulate the colourful independence oracle for this hypergraph.

Theorem 2 implies that any decision algorithm for k-path can be turned into an approximate counting algorithm by paying a $\log^{\mathcal{O}(k)} n$-factor overhead in the running time. While this is still a fixed-parameter tractable running time, it leads to a useless algorithm, since the running time is much worse than $c^k \text{poly}(n)$. The main consequence of Theorem 2 for k-path stems not from this meaningless upper bound, but from the lower bound, which is new even for $\alpha_k = 0$: Our results imply that if the decision algorithm for k-path is formalized using the colourful independence oracle, then the overhead of the approximate-counting-to-decision reduction must be $\log^{\Omega(k)} n$, and so a subexponential-time reduction cannot exist. Conversely, if a useful approximate-counting-to-decision reduction exists, it cannot merely be based on the hypergraph whose edges consist of all k-paths; instead, the reduction would have to have access to and exploit the underlying structure of the original graph. We believe that this is a useful insight for the design of future algorithms for approximate counting.

3 Reductions from approximate counting to decision

Theorems 1 and 2 can easily be applied to obtain reductions from approximate counting to decision for many important problems in fine-grained and parameterised complexity. The following definition is taken from [13]; recall that a counting problem is a function $\#\Pi\colon \{0,1\}^* \to \mathbb{N}$ and its corresponding decision problem is defined via $\Pi = \{x \in \{0,1\}^* \colon \#\Pi(x) > 0\}$.

▶ **Definition 3.** *The decision problem* Π *is a* uniform witness problem *if there is a function that maps instances* $x \in \{0,1\}^*$ *to uniform hypergraphs* G_x *such that the following statements hold:*

- **(i)** $\#\Pi(x)$ *is equal to the number* $e(G_x)$ *of edges in* G_x*;*
- **(ii)** $V(G_x)$ *and the size* $k(G_x)$ *of edges in* $E(G_x)$ *can be computed from* x *in time* $\widetilde{\mathcal{O}}(|x|)$*;*
- **(iii)** *there exists an algorithm which, given* x *and* $S \subseteq V(G_x)$*, in time* $\widetilde{\mathcal{O}}(|x|)$ *prepares an instance* $I_x(S) \in \{0,1\}^*$ *such that* $G_{I_x(S)} = G_x[S]$ *and* $|I_x(S)| \in \mathcal{O}(|x|)$*.*

The set $E(G_x)$ *is the set of* witnesses *of the instance* x*.*

Intuitively, we can think of a uniform witness problem as a problem of counting witnesses in an instance x that can be naturally expressed as edges in a hypergraph G_x, in such a way that induced subgraphs of G_x correspond to sub-instances of x. This allows us to simulate a query to $\texttt{IND}(G_x)_S$ by running a decision algorithm for Π on the instance $I_x(S)$, and if

our decision algorithm runs on an instance y in time $T(|y|)$ then this simulation will require time $\widetilde{\mathcal{O}}(T(|S|))$. Typically there is only one natural map $x \mapsto G_x$, and so we consider it to be a part of the problem statement. Simulating the independence oracle in this way, the statement of Theorem 1 yields the following.

▶ **Theorem 4.** *Suppose $\alpha_k \in [1, k]$ for all $k \geq 2$. Let Π be a uniform witness problem. Suppose that given an instance x of Π, writing $n = |V(G_x)|$ and $k = k(G_x)$, there is an algorithm to solve Π on x with error probability at most $1/3$ in time $\widetilde{\mathcal{O}}(n^{\alpha_k})$. Then there is an ε-approximation algorithm for $\#\Pi(x)$ with error probability at most $1/3$ and running time*

$$k^{\mathcal{O}(k)} + \varepsilon^{-2} n^{\alpha_k} \cdot 2^{\mathcal{O}(k)} n^{g(k,\alpha_k)}.$$

Note that the running time of the algorithm for $\#\Pi$ is the sum of the oracle cost and the running time of the algorithm of Theorem 1; by requiring $\alpha_k \geq 1$, we ensure this is dominated by the oracle cost. (Indeed, for most uniform witness problems it is very easy to prove that every decision algorithm must read a constant proportion of the input, and so we will always have $\alpha_k \geq 1$.) The lower bound of Theorem 1 implies that the $n^{\alpha_k + g(k,\alpha_k)}$ term in the running time cannot be substantially improved with any argument that relativises; thus in simple terms, there is a generic fine-grained reduction from approximate counting to decision if and only if the decision algorithm runs in time $\Omega(n^{k-1})$.

It is instructive to consider an example. Take Π to be the problem Induced-H of deciding whether a given input graph G contains an induced copy of a fixed graph H. In this case, the hypergraph corresponding to an instance G will have vertex set $V(G)$ and edge set $\{X \subseteq V(G) \colon G[X] \simeq H\}$; thus the witnesses are vertex sets which induce copies of H in G. The requirements of Definition 3(i) and (ii) are immediately satisfied, and Definition 3(iii) is satisfied since deleting vertices from the hypergraph corresponds to deleting vertices of G. Thus writing $k = |V(H)|$, Theorem 4 gives us a reduction from approximate #Induced-H to Induced-H with overhead $\varepsilon^{-2} 2^{\mathcal{O}(k)} n^{g(k,\alpha_k)}$ over the cost of the decision algorithm. Moreover, on applying the fine-grained reduction from approximate sampling to counting in [13] we also obtain an approximate uniform sampling algorithm with overhead $\varepsilon^{-2} 2^{\mathcal{O}(k)} n^{g(k,\alpha_k)}$. Many more examples of uniform witness problems to which Theorem 4 applies can be found in the introduction of [13].

We now describe the corresponding result in the colourful oracle setting, which we now set out – again, the following definition is taken from [13].

▶ **Definition 5.** *Suppose Π is a uniform witness problem. Colourful-Π is defined as the problem of, given an instance $x \in \{0,1\}^*$ of Π and a partition of $V(G_x)$ into disjoint sets $S_1, \dots, S_k$, deciding whether $\texttt{cIND}_{G_x}(S_1, \dots, S_k) = 0$ holds.*

Continuing our previous example, in Colourful-Induced-H, we are given a (perhaps improper) vertex colouring of our input graph G, and we wish to decide whether G contains an induced copy of H with exactly one vertex from each colour. Simulating an oracle call to $\texttt{cIND}(G_x)_{S_1,\dots,S_k}$ corresponds to running a decision algorithm for Colourful-Π on the instance $I_x(S_1 \cup \dots \cup S_k)$ with colour classes $S_1, \dots, S_k$, and if this decision algorithm runs on an instance y in time $T(|y|)$ then this simulation will require time $\widetilde{\mathcal{O}}(T(|S_1| + \dots + |S_k|))$. Simulating the colourful independence oracle in this way, the statement of Theorem 2 yields the following.

▶ **Theorem 6.** *Suppose $\alpha_k \in [1, k]$ for all $k \geq 2$. Let Π be a uniform witness problem. Suppose that given an instance x of Π, writing $n = |V(G_x)|$ and $k = k(G_x)$, there is an algorithm to solve Colourful-Π on x with error probability at most $1/3$ in time $\mathcal{O}(n^{\alpha_k})$. Then there is an ε-approximation algorithm for $\#\Pi(x)$ with error probability at most $1/3$ and running time $\varepsilon^{-2} n^{\alpha_k} \cdot k^{\mathcal{O}(k)} (\log n)^{\mathcal{O}(k - \alpha_k)}$.*

As in the uncoloured case, the requirement $\alpha_k \geq 1$ ensures that the dominant term in the running time is the time required to simulate the required oracle queries, and the lower bound of Theorem 2 implies that the $\log^{\mathcal{O}(k-\alpha_k)} n$ term in the running time cannot be substantially improved with any argument that relativises. Again writing $k = |V(H)|$, Theorem 6 gives us a reduction from approximate #Induced-H to Colourful-Induced-H with overhead $\varepsilon^{-2}k^{\mathcal{O}(k)}(\log n)^{\mathcal{O}(k-\alpha_k)}$ over the cost of the decision algorithm. This result improves on the reduction of [13, Theorem 1.7] by a factor of $\log^{\Omega(\alpha_k)} n$, and using the fine-grained reduction from approximate sampling to counting in [13] it can immediately be turned into an approximate uniform sampling algorithm.

Observe that in most cases, there is far less overhead over decision in applying Theorem 6 to reduce #Induced-H to Colourful-Induced-H than there is in applying Theorem 4 to reduce #Induced-H to Induced-H. In some cases, such as the case where H is a k-clique, there are simple fine-grained reductions from Colourful-Induced-H to Induced-H, and in this case Theorem 6 is an improvement. However, it is not known whether the same is true of all choices of H, and indeed even an FPT reduction from Colourful-Induced-H to Induced-H would imply the long-standing dichotomy conjecture for the embedding problem introduced in [18]. More generally, but still within the setting of uniform witness problems, the problem of detecting whether a graph contains a size-k set which either spans a clique or spans an independent set is in FPT by Ramsey's theorem, but its colourful version is W[1]-complete [22].

While the distinction between colourful problems and uncoloured problems is already well-studied in subgraph problems, these results strongly suggest that it is worth studying in many other contexts in fine-grained complexity as well. Indeed, it is easy to simulate $\texttt{IND}(G)$ from $\texttt{cIND}(G)$ with random colouring; thus the lower bound of Theorem 1 and the upper bound of Theorem 2 imply that there is a fine-grained reduction from uncoloured approximate counting to colourful decision, but not to uncoloured decision. By studying the relationship between colourful problems and their uncoloured counterparts, we may therefore hope to shed light on the relationship between approximate counting and decision.

Finally, we observe that the set of running times allowed by Theorems 4 and 6 may not be sufficiently fine-grained to derive meaningful results for some problems. In fine-grained complexity, even a subpolynomial improvement to a polynomial-time algorithm may be of significant interest – the classic example is the Negative-Weight-Triangle algorithm of [31], which runs in $n^3/e^{\Omega(\sqrt{\log n})}$ time on an n-vertex instance, compared to the naive $\mathcal{O}(n^3)$-time algorithm. In order to "lift" such improvements from decision problems to approximate counting, we must generalise the upper bounds of Theorems 1 and 2 to cost functions of the form $\text{cost}_k(x) = x^{\alpha_k \pm o(1)}$ while maintaining low overhead. We do so in Theorem 10 for all "reasonable" cost functions, including any function of the form $\text{cost}_k(x) = n^{\alpha_k}\log^{\beta_k} n$ and any function of the form $\text{cost}_k(x) = n^{\alpha_k}e^{\pm(\log n)^{\gamma_k}}$ where $\gamma_k < 1$. A full list of technical requirements is given in Section 6, but the most important one is regular variation – this is a standard notion from probability theory for "almost polynomial" functions, and requiring it avoids pathological cases where (for example) we may have $\text{cost}_k(x) = \mathcal{O}(x)$ as $x \to \infty$, but $\text{cost}_k(2x_i) = \omega(\text{cost}_k(x_i))$ as $i \to \infty$ along some sequence $(x_i : i \geq 1)$.

4 Discussion of related work

In order to compare algorithms without excessive re-definition of notation, throughout this subsection we consider the problem of ε-approximating the number of edges in an m-edge, n-vertex k-hypergraph.

Colourful and uncoloured independence oracles were introduced in [3] in the graph setting, then first generalised to hypergraphs in [6]. Edge estimation using these oracles was first studied in the graph setting (i.e. for $k = 2$) in [3], which gave an $\varepsilon^{-4}\log^{\mathcal{O}(1)} n$-query algorithm for colourful independence oracles and an $(\varepsilon^{-4} + \varepsilon^{-2}\min\{\sqrt{m}, n^2/m\})\log^{\mathcal{O}(1)} n = (\varepsilon^{-4} + \varepsilon^{-2}n^{2/3})\log^{\mathcal{O}(1)} n$-query algorithm for uncoloured independence oracles. The connection to approximate counting in fine-grained and parameterised complexity was first studied in [12].

For colourful independence oracles in the graph setting, [12] (independently from [3]) gave an algorithm using $\varepsilon^{-2}\log^{\mathcal{O}(1)} n$ `cIND` queries and $\varepsilon^{-2}n\log^{\mathcal{O}(1)} n$ adjacency queries. [1] subsequently gave a *non-adaptive* algorithm using $\varepsilon^{-6}\log^{\mathcal{O}(1)} n$ `cIND` queries.

The case of edge estimation in k-hypergraphs (i.e. for arbitrary $k \geq 2$) was first considered independently by [13, 4]; [13] gave an algorithm using $\varepsilon^{-2}k^{\mathcal{O}(k)}\log^{4k+\mathcal{O}(1)} n$ queries, while [4] gave an $\varepsilon^{-4}k^{\mathcal{O}(k)}\log^{4k+\mathcal{O}(1)}$-query algorithm. [13] also introduced a reduction from approximate sampling to approximate counting in this setting (which also applies in the uncoloured setting) with overhead $k^{\mathcal{O}(k)}\log^{\mathcal{O}(1)} n$. [5] then improved the query count further to $\varepsilon^{-2}k^{\mathcal{O}(k)}\log^{3k+\mathcal{O}(1)} n$.

In this paper, we give an algorithm with total query cost $\varepsilon^{-2}k^{\mathcal{O}(k)}\log^{4(k-\alpha_k)+\mathcal{O}(1)} n$ under $\mathrm{cost}_k(x) = x^{\alpha_k}$, giving polylogarithmic overhead when $\alpha_k \approx k$. We also give a lower bound which shows that a $\log^{\Theta(k-\alpha_k)}$ term is necessary; no lower bounds were previously known even for $\alpha_k = 0$ (i.e. the case where the total query cost equals the number of queries).

For uncoloured independence oracles of graphs, [11] improved the algorithm of [3] to use

$$\varepsilon^{-\Theta(1)}\min\{\sqrt{m}, n/\sqrt{m}\}\operatorname{polylog} n = \varepsilon^{-\Theta(1)}\sqrt{n}\operatorname{polylog} n \tag{4.0.1}$$

queries and gave a matching lower bound. (It is difficult to tell the exact value of the $\Theta(1)$ term from the proof, but it is at least 9 – see the definition of N in the proof of Lemma 3.9 on p. 15.) It is worth noting that the bound in (4.0.1) is stated as a function of both n and m. We believe that our results can be stated in such a way as well, but we defer doing so to the journal version of this paper.

To the best of our knowledge, no results on uncoloured edge estimation for $\alpha_k > 0$ or $k > 2$ have previously appeared in the literature. However, we believe it would be easy to partially generalise the proof of [3] to this setting. Very roughly speaking, their argument works by running a naive sampling algorithm and a more subtle branch-and-bound approximation algorithm in parallel, with the sampling algorithm running quickly on dense graphs and the branch-and-bound algorithm running quickly on sparse graphs. The main obstacle to generalising this approach would be the branch-and-bound approximation algorithm; however, by replacing it with a slower branch-and-bound *enumeration* algorithm for k-hypergraphs such as [23], we believe we would obtain worst-case oracle cost $k^{\mathcal{O}(k)} + \varepsilon^{-2}2^{\mathcal{O}(k)}n^{\alpha_k+(k-\alpha_k)/2}$ under $\mathrm{cost}_k(x) = x^{\alpha_k}$; this technique is well-known in the literature and also appears in e.g. [29]. By comparison (see Figure 1), the algorithm of Theorem 1 achieves a much smaller worst-case oracle cost of $k^{\mathcal{O}(k)} + \varepsilon^{-2}2^{\mathcal{O}(k)}n^{\alpha_k+g(k,\alpha_k)}$, where $g(k,\alpha_k) \approx (k-\alpha_k)^2/(4k) < (k-\alpha_k)/2$ and where $g(k,\alpha_k) = 0$ for $\alpha_k \geq k-1$. This substantially improves on the algorithm implicit in [3], and indeed is optimal up to a factor of $\varepsilon^{\Theta(1)}k^{\Theta(k)}$. Also, our algorithm has a better dependence on ε compared with [11] when $k = 2$; however, we bound the cost only in terms of n and not in terms of m.

Although a full survey is beyond the scope of this paper, there are natural generalisations of (colourful and uncoloured) independence oracles [27], and edge estimation problems are studied for other oracle models including neighbourhood access [28]. Other types of oracle are also regularly applied to fine-grained complexity in other models, notably including cut oracles [24, 2]. Perhaps surprisingly, we were unable to find any previous examples in the

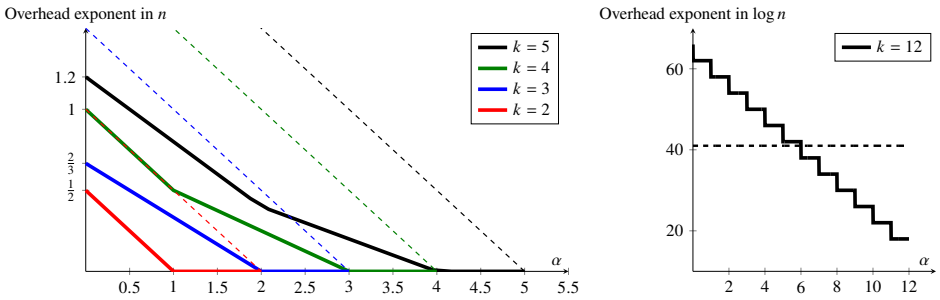


Figure 1 *Left:* If each IND-query of size x has cost x^{α_k}, then up to $k^{\mathcal{O}(k)} \log^{\mathcal{O}(1)} n$ factors, we show in Theorem 1 that $n^{g(k,\alpha_k)+\alpha_k}$ is the smallest possible IND-oracle cost to $(1/2)$-approximate the number of edges. Plotted here in *thick lines* is the overhead $\alpha \mapsto g(k,\alpha)$ in the exponent of n for $k \in \{2,3,4,5\}$, and in *dashed lines* is the larger overhead exponent $\alpha \mapsto (k-\alpha)/2$ obtained from a naive generalisation of the techniques of [3]. *Right:* If each cIND-query of size x has cost x^{α_k}, then up to $k^{\mathcal{O}(k)}(\log n)^{\mathcal{O}(1)}(\log\log n)^{\mathcal{O}(k-\alpha_k)}$ factors, we show in Theorem 2 that $n^{\alpha_k} \log^{4(k-\lceil\alpha_k\rceil)+18} n$ is the smallest possible cIND-oracle cost to $(1/2)$-approximate the number of edges. Plotted in *thick lines* is the overhead $\alpha \mapsto 4(k-\lceil\alpha\rceil)+18$ in the exponent of $\log n$ for $k = 12$, and the *dashed line* depicts the overhead $\alpha \mapsto 3k+5$ obtained by using the bound on the number of queries by [5]; our bound is better if $\lceil\alpha_k\rceil \geq k/4 + \Theta(1)$.

literature of unconditional oracle lower bounds relative to a general query cost function, and so our definitions and methods are novel in that sense. Of course, many existing works prove unconditional lower bounds in terms of query count [28], or consider algorithmic construction of graph oracles with bounded query times [19], or provide oracle algorithms with fast running times in addition to low query counts [26]. In our setting, however, a lower bound in terms of query cost is absolutely necessary. Recall that for us, query cost is the running time of an algorithm for the decision problem we are reducing to, and the algorithmic results of Theorems 4 and 6 all rely on smaller queries running faster; thus to prove they are "best possible" in any meaningful sense, we absolutely require the formal notion of query cost set out in Section 2.

Outside the oracle setting, it was recently proved [21] that any decision algorithm built around the representative family technique of [17] can be turned into an approximate counting algorithm with substantially lower overheads in k than those of [13]. More recently, several important decision problems in the fine-grained setting with $k = 3$ have been discovered to be "equivalent" to their *exact* counting versions [10]. This work is not directly comparable to ours, as they work in a substantially stronger setting and use a correspondingly weaker notion of equivalence. Their equivalence is in the sense of equivalence of conjectures – for example, they prove that if there is an $O(n^{2-\varepsilon})$-time algorithm for 3-SUM, then there exists $0 < \varepsilon' < \varepsilon$ such that there is an $O(n^{2-\varepsilon'})$-time algorithm for #3-SUM. We stress that for exact counting problems as studied in [10], such equivalence results are genuinely deep and surprising. However, for the approximate counting problems we study, analogous results are typically quite easy to prove via the standard combination of sampling and branch-and-bound approaches discussed above, and so we instead study the stronger notion of fine-grained reductions from approximate counting to decision. Where no such reductions exist, we aim to nail down the exact value of ε'. (Recall also that there are uniform witness problems whose decision versions are in FPT but whose exact counting versions are W[1]-hard [22], so we cannot hope to extend our results to exact counting in this setting.)

5 Proof techniques

5.1 Colourful upper bound

We first discuss the proof of the upper bound of Theorem 2, our cIND-oracle algorithm for edge estimation using the colourful independence oracle of an n-vertex k-hypergraph G. In [13], it is implicitly proved that a cIND-oracle algorithm that computes a "coarse approximation" to $e(G)$, which is accurate up to multiplicative error b, can be bootstrapped into a full ε-approximation algorithm with overhead $\varepsilon^{-2}2^{\mathcal{O}(k)}\log^{\mathcal{O}(1)} n \cdot b^2$. (See Theorem 49 of the full version for details.) It therefore suffices to improve the coarse approximation algorithm of [13] from $k^{\mathcal{O}(k)}\log^{\Theta(k)} n$ queries and $k^{\mathcal{O}(k)}\log^{\Theta(k)} n$ multiplicative error to $k^{\mathcal{O}(k)}\log^{\Theta(k-\alpha_k)} n$ query cost and $k^{\mathcal{O}(k)}\log^{\Theta(k-\alpha_k)} n$ multiplicative error. Moreover, by a standard colour-coding argument, it suffices to make this improvement when G is k-partite with vertex classes $V_1,\dots,V_k$ known to the algorithm.

Oversimplifying a little, and assuming n is a power of two, the algorithm of [13] works by guessing a probability vector $(Q_1,\dots,Q_k)\in\{1,1/2,1/4,\dots,1/n\}^k$. It then deletes vertices from $V_1,\dots,V_k$ independently at random to form sets $\mathcal{X}_1,\dots,\mathcal{X}_k$, so that for all $v\in V_j$ we have $\mathbb{P}(v\in\mathcal{X}_j)=Q_j$. After doing so, in expectation, $Q_1Q_2\dots Q_k$ proportion of the edges of G remain in the induced k-partite subgraph $G[\mathcal{X}_1,\dots,\mathcal{X}_k]$. If $e(G)\ll 1/(Q_1\dots Q_k)$, it is easy to show with a union bound that no edges are likely to remain. What is more surprising is that there exist $q_1,\dots,q_k$ with $q_1\dots q_k\approx 1/e(G)$ such that if $\vec{Q}=\vec{q}$, then at least one edge is likely to remain in $G[\mathcal{X}_1,\dots,\mathcal{X}_k]$. Thus the algorithm of [13] iterates over all $\log^{\Theta(k)} n$ possible values of $\vec{Q}$, querying $\texttt{cIND}(G)$ on $\mathcal{X}_1,\dots,\mathcal{X}_k$ for each, and then outputs the least value m such that $e(G[\mathcal{X}_1,\dots,\mathcal{X}_k])>0$ for some $Q_1,\dots,Q_k$ with $1/(Q_1\dots Q_k)=m$.

Our algorithm improves on this idea as follows. First, [13] does not actually prove the existence of the vector $\vec{q}$ described above – it relies on a coupling between the different guesses of $\vec{Q}$. We require not only the existence of $\vec{q}$ but also a structural result which may be of independent interest. For all $I\subseteq[k]$ and all $\zeta\in(0,1]$, we define an (I,ζ)-*core* to be an induced subgraph $H=G[Y_1,\dots,Y_k]$ of G such that:

- **(i)** H contains at least $k^{-\mathcal{O}(k)}$ proportion of the edges of G.
- **(ii)** For all $i\in I$, the set Y_i is very small, containing at most $2/\zeta$ vertices.
- **(iii)** For all $i\notin I$, every vertex of Y_i is contained in at most ζ proportion of edges in H.

As an example, the most extreme core is the *rooted star*: It consists of some vertices $r_i\in Y_i$ for all $i\in I$ and *all* k-partite edges e with $e\supseteq\{r_i:i\in I\}$. We prove in Lemma 56 of the full version that, for all $\zeta\in(0,1]$, there is some $I\subseteq[k]$ such that G contains an (I,ζ)-core H.

Suppose for the moment that we are given the value of I, but not $Y_1,\dots,Y_k$. By property (i), it would then suffice to approximate $e(H)$ using $k^{\mathcal{O}(k)}\log^{\mathcal{O}(k-\alpha_k)} n$ query cost. If $|I|\geq\alpha_k$, then we can adapt the idea of the algorithm of [13], but taking $Q_j=1$ for all $i\notin I$ to use only $\log^{\mathcal{O}(k-\alpha_k)} n$ queries in total; intuitively, this is possible due to property (ii), which implies that this is the "correct" setting. We set this algorithm out as CoarseLargeCore in Section 4.1.3 of the full version.

If instead $|I|\leq\alpha_k$, then we will exploit the fact that query cost decreases polynomially with instance size by breaking H into smaller instances. For all $i\notin I$, we randomly delete vertices from V_i with a carefully-chosen probability p. Property (iii), together with a martingale bound (see Lemma 59 of the full version), guarantees that the number of edges in the resulting hypergraph G' will be concentrated around its expectation of $p^k e(G)$. If we had access to $Y_1,\dots,Y_k$, we could then intersect V_i with Y_i for all $i\in I$ to obtain a substantially smaller instance, whose edges we could count with cheap queries; we could then divide the result by p^k to obtain an estimate for $e(G)$. Unfortunately we do not have

access to $Y_1, \dots, Y_k$, but we can still break G' into smaller sub-hypergraphs by applying colour-coding to the vertex sets V_i with $i \in I$, and as long as $|I| \leq \alpha_k$ this still gives enough of a saving in query cost to prove the result. We set this algorithm out as `CoarseSmallCore` in Section 4.1.2 of the full version.

Now, we are not in fact given the value of I in the (I, ζ)-core of G. But both `CoarseLargeCore` and `CoarseSmallCore` fail gracefully if they are given an incorrect value of I, returning an underestimate of $e(G)$ rather than an overestimate. We can therefore simply iterate over all 2^k possible values of I, applying `CoarseLargeCore` or `CoarseSmallCore` as appropriate, and return the maximum resulting estimate of $e(G)$. This proves the result.

5.2 Colourful lower bound

We now discuss the proof of the lower bound of Theorem 2. Using the minimax principle, to prove the bound for randomised algorithms, it is enough to give a pair of random graphs G_1 and G_2 with $e(G_2) \gg e(G_1)$ and prove that no *deterministic* algorithm A can distinguish between G_1 and G_2 with constant probability and worst-case oracle cost as in the bound. We base these random graphs on the main bottleneck in the algorithm described in the previous section: the need to check all possible values of $\mathcal{Q}$ in a k-partite k-hypergraph with an (I, ζ)-core where $|I| \approx k - \alpha_k$.

We take G_1 to be an Erdős-Rényi k-partite k-hypergraph with edge density $p := t^{-(k-\lfloor \alpha_k \rfloor - 2)/2}$. We take the vertex classes $V_1, \dots, V_k$ of G_1 to have equal size t, so that $t = n/k$. We then define a random complete k-partite graph H as follows. We first define a random vector $\vec{\mathcal{Q}}$ of probabilities, then take binomially random subsets $\mathcal{X}_1, \dots, \mathcal{X}_{k-\lfloor \alpha_k \rfloor - 2}$ of $V_1, \dots, V_{k-\lfloor \alpha_k \rfloor - 2}$, with $\mathbb{P}(v \in \mathcal{X}_j) = \mathcal{Q}_j$ for all $v \in V_j$. For $j \geq k - \lfloor \alpha_k \rfloor - 1$, we take $\mathcal{X}_j \subseteq V_j$ to contain a single uniformly random vertex. We then define H to be the complete k-partite graph with vertex classes $\mathcal{X}_1, \dots, \mathcal{X}_k$, and form $G_2 = G_1 \cup H$ by adding the edges of H to G_1. We choose $\mathcal{Q}$ in such a way that $\mathcal{Q}_1 \cdot \ldots \cdot \mathcal{Q}_{k-\lfloor \alpha_k \rfloor - 2}$ is guaranteed to be a bit larger than $pt^{\lfloor \alpha_k \rfloor + 2}$, so that $\mathbb{E}(e(H)) \gg \mathbb{E}(e(G_1))$. Intuitively, this corresponds to the situation of a randomly planted (I, ζ)-core where $I = \{k - \lfloor \alpha_k \rfloor - 1, \dots, k\}$ – we will show that the algorithm A needs to essentially guess the value of $\mathcal{Q}$ using expensive queries.

To show that a low-cost deterministic algorithm A cannot distinguish G_1 from G_2, suppose for simplicity that A is *non-adaptive*, so that its future oracle queries cannot depend on the oracle's past responses. In this setting, it suffices to bound the probability of a fixed query $S = (S_1, \dots, S_k)$ distinguishing G_1 from G_2.

It is not hard to show that without loss of generality we can assume $S_i \subseteq V_i$ for all $i \in [k]$. If $|S_j| \ll t$ for some $j \geq k - \lfloor \alpha_k \rfloor - 1$, then with high probability S_j will not contain the single "root" vertex of $\mathcal{X}_j$, so $H[S_1, \dots, S_k]$ will contain no edges and S will not distinguish G_1 from G_2. With some effort (a simple union bound does not suffice), this idea allows us to essentially restrict our attention to large, expensive queries S. However, if $|S_1| \dots |S_k|$ is large, then with high probability $G_1[S_1, \dots, S_k]$ will contain an edge, so again S will not distinguish G_1 from $G_2 = G_1 \cup H$. With some more effort, this allows us to essentially restrict our attention to queries where for some possible value $\vec{q}$ of $\vec{\mathcal{Q}}$ we have $|S_j| \approx 1/q_j$ for all $j \leq k - \lfloor \alpha \rfloor - 2$; we choose these possible values to be far enough apart that such a query is only likely to distinguish G_1 from G_2 if $\vec{\mathcal{Q}} = \vec{q}$. There are roughly $((\log n)/\log\log n)^{k-\lfloor \alpha_k \rfloor - 2}$ possible values of $\mathcal{Q}_j$, so the result follows.

Of course, in our setting A may be adaptive, and this breaks the argument above. Since the query A makes depends on the results of past queries, we cannot bound the probability of a fixed query distinguishing G_1 from G_2 in isolation – we must condition on the results of past queries. This is not a small technical point – it is equivalent to allowing A to be

adaptive in the first place. The most damaging implication is that we could have a query $S = (S_1, \dots, S_k)$ with $|S_1| \dots |S_k|$ very large but such that $G_1[S_1, \dots, S_k]$ contains no edges, because most of the potential edges have already been exposed as not existing in past queries. We are able to deal with this while preserving the spirit of the argument above, by arguing based on the number of unexposed edges rather than $|S_1| \dots |S_k|$, but it requires significantly more effort and a great deal of care.

5.3 Uncoloured upper bound

We now discuss the proof of the upper bound of Theorem 1. We adapt a classic framework for approximate counting algorithms that originated in [30], and that was previously applied to edge counting in [12]. We first observe that by using an algorithm from [23], we can enumerate the edges in an n-vertex k-hypergraph G with $2^{\mathcal{O}(k)} \log^{\mathcal{O}(1)} n \cdot e(G)$ queries to an independence oracle. Suppose we form an induced subgraph G_i of G by deleting vertices independently at random, keeping each vertex with probability 2^{-i}; then in expectation, we have $e(G_i) = 2^{-ki} e(G)$. If $e(G_i)$ is small, and $e(G_i) \approx \mathbb{E}(e(G_i))$, then we can efficiently count the edges of G_i using [23] and then multiply by 2^{ki} to obtain an estimate of $e(G)$. We can then simply iterate over all i from 0 to $\log n$ and return an estimate based on the first i such that $e(G_i)$ is small enough for [23] to return a value quickly.

Of course in general we do not have $e(G_i) \approx \mathbb{E}(e(G_i))$! One issue arises if, for some $r \in [k-1]$, every edge of G contains a common size-r set R – a "root". Then with probability $1 - 2^{-ri}$, at least one vertex in R will be deleted and G_i will contain no edges. We address this issue in the simplest way possible: by taking more samples. Roughly speaking, suppose we are given i, and that we already know (based on the failure of previous values of i to return a result) that $e(G) > n^{k-r-1}$ for some $0 \leq r \leq k-1$. This implies that G cannot have any "roots" of size greater than r. Rather than taking a single random subgraph G_i, we take $t_i \approx 2^{ri}$ independent copies of G_i and sum their edge counts using [23]; thus if G does contain a size-r root, we are likely to include it in the vertex set of at least one sample. Writing Σ_i for the sum of their edge counts, we then return $\Sigma_i / (t_i 2^{-ik})$ if the enumeration succeeds.

The exact expression we use for t_i is more complicated than 2^{ri}, due to the possibility of multiple roots – see Section 3.2.2 of the full version for a more detailed discussion – but the idea is the same. The proof that Σ_i is concentrated around its expectation is an (admittedly somewhat involved) application of Chebyshev's inequality, in which the rooted "worst cases" correspond to terms in the variance of Σ_i. We consider it surprising and interesting that such a conceptually simple approach yields an optimal upper bound, and indeed gives us a strong hint as to how we should prove the lower bound of Theorem 1.

5.4 Uncoloured lower bound

We finally discuss the proof of the lower bound of Theorem 1. As in the colourful case, we apply the minimax principle, so we wish to define random k-hypergraphs G_1 and G_2 with $e(G_2) \gg e(G_1)$ which cannot be distinguished by a low-cost deterministic algorithm A.

Our construction of G_1 and G_2 is natural from the above discussion, and the special case with $k = 2$ and $\alpha_k = 0$ is very similar to the construction used in [11]. We take G_1 to be an Erdős-Rényi k-hypergraph with edge probability roughly $k!/n^r$. We choose a random collection of size-r sets in $V(G_1)$ to be "roots", with a large constant number of roots present in expectation, and we define a k-hypergraph H to include every possible edge containing at least one of these roots as a subset. We then define $G_2 := G_1 \cup H$.

Similarly to the colourful lower bound, in the non-adaptive setting, any fixed query S_i with $|S_i|$ large is likely to contain an edge of G_1, and any fixed query with $|S_i|$ small is unlikely to contain a root and therefore unlikely to contain any edges of H; in either case, the query does not distinguish G_1 from G_2. Also as in the colourful case, generalising this argument from the non-adaptive setting to the adaptive setting requires a significant amount of care and effort.

6 More general cost functions for upper bounds

In our previous theorem statements we focused on polynomial cost functions of the form $\text{cost}_k(S) = |S|^{\alpha_k}$ for some $\alpha_k \in [0, k]$, and these functions are easy to work with. However, even subpolynomial improvements to an algorithm's running time can be of interest and by considering more general cost functions we can lift these improvements from decision algorithms to approximate counting algorithms. To take a well-known example, in the negative-weight triangle problem, we are given an n-vertex edge-weighted graph G and asked to determine whether it contains a triangle of negative total weight; this problem is equivalent to a set of other problems under subcubic reductions, including APSP [32]. The naive $\Theta(n^3)$-time algorithm can be improved by a subpolynomial factor of $e^{\Theta(\sqrt{\log n})}$ [31], but as stated in the introduction our result would not lift this improvement from decision to approximate counting – we would need to take $k = 3$ and $\text{cost}(n) = n^3/e^{\Theta(\sqrt{\log n})}$. (For clarity, in this specific case the problem is equivalent to its colourful version and a fine-grained reduction is already known [12, 13].)

While we cannot hope to say anything meaningful in the algorithmic setting with a fully general cost function, our results do extend to all cost functions which might reasonably arise as running times. A natural first attempt to formalise what we mean by "reasonable" would be to consider cost functions of the form $\text{cost}_k(n) = n^{\alpha_k + o(1)}$ as $n \to \infty$. Unfortunately, such cost functions can still have a pathological property which makes a fine-grained reduction almost impossible: the $o(1)$ term might vary wildly between different values of n. For example, if we take $\text{cost}_k(n) = n^{\alpha_k + \sin(\pi n/2)/\sqrt{\log(n)}}$, then we have $\text{cost}_k(n) = n^{\alpha_k + o(1)}$, but $\text{cost}_k(2n)/\text{cost}_k(n)$ could be $\omega(\text{polylog}(n))$, $\Theta(1)$ or $o(1/\text{polylog}(n))$ depending on whether n is congruent to 3, 2 or 1 modulo 4 (respectively). It is even possible to construct a similar example which is monotonically increasing. We will need to be able to approximate $f(Ax)$ by $A^{\alpha_k} f(x)$ and so we require something slightly stronger, borrowing a standard notion from the probability literature for distributions which are "almost power laws".

▶ **Definition 7.** *A function* $f\colon (0,\infty) \to \mathbb{R}$ *is* regularly-varying *if, for all fixed* $A > 0$,

$$\lim_{x\to\infty} |f(Ax)/f(x)| \in (0,\infty).$$

We say f *is* slowly-varying *if this limit is always* 1.

Observe that any cost function likely to arise as a running time is regularly-varying. While Definition 7 may look overly general for our purposes, in fact it gives us exactly what we need – regularly-varying functions are all of the form $f(x) = x^{\alpha + o(1)}$ as $x \to \infty$, are eventually increasing whenever $\alpha > 0$, and satisfy very strict bounds on $f(Ax)/f(x)$ as $x \to \infty$ as detailed in the following lemma. (These properties are all well-known, see e.g. Feller [15].) With these properties in hand, the extra generality of Definition 7 adds very little overhead to our existing arguments.

▶ **Lemma 8.** *Let $f\colon (0,\infty)\to\mathbb{R}$ be a function. Then f is regularly-varying if and only if there exists a unique $\alpha\in\mathbb{R}$, called the* index *of f, and a unique slowly-varying function $\sigma\colon(0,\infty)\to\mathbb{R}$, called the* slowly-varying component *of f, such that $f(x)=x^{\alpha}\sigma(x)$. Moreover, any regularly-varying function f and slowly-varying function σ satisfy the following properties.*

- **(i)** *$\sigma(x)=x^{o(1)}$ as $x\to\infty$.*
- **(ii)** *For all fixed $A>0$, $\lim_{x\to\infty}(\sigma(Ax)/\sigma(x))=1$. Moreover, for all closed intervals $I\subseteq\mathbb{R}$, this limit is uniform over all $A\in I$.*
- **(iii)** *For all $\delta>0$, there exists x_0 such that for all $x\geq x_0$ and all $A_x\geq 1$,*

$$A_x^{-\delta}\leq\sigma(A_x x)/\sigma(x)\leq A_x^{\delta}.$$

- **(iv)** *If f has positive index, then there exists x_0 such that f is strictly increasing on $[x_0,\infty)$.*

Our cost function will be a function of both the parameter k and the number n of vertices in the instance, and we will only require regular variation in n. We are now ready to state the technical requirements on our cost functions.

▶ **Definition 9.** *For each $k\geq 2$, let $\mathrm{cost}_k\colon(0,\infty)\to(0,\infty)$. We say that $\mathrm{cost}=\{\mathrm{cost}_k\colon k\geq 2\}$ is a* regularly-varying parameterised cost function *if $\mathrm{cost}_k(x)\leq x^k$ for all k and x, and there exists a slowly-varying function $\sigma\colon(0,\infty)\to(0,\infty)$ and a map $k\mapsto\alpha_k$ satisfying the following properties:*

- **(i)** *for all k and x, $\mathrm{cost}_k(x)=x^{\alpha_k}\sigma(x)$;*
- **(ii)** *for all k, $\alpha_k\in[0,k]$;*
- **(iii)** *for all k and x, $\mathrm{cost}_k(x)\leq x^k$;*
- **(iv)** *either $\liminf_{k\to\infty}\alpha_k>0$ or there exists x_0 such that for all k, cost_k is non-decreasing on (x_0,∞);*
- **(v)** *there is an algorithm to compute $\lfloor\alpha_k\rfloor$ in time $\mathcal{O}(k^{9k})$.*

We say that k is the parameter *of* cost*, α is the* index *of* cost*, and σ is the* slowly-varying component *of* cost*.*

Point (i) is the main restriction, and the one we have been discussing until now. Points (ii) and (iii) have already been discussed in the introduction. In the colourful case we will need to compute $\lfloor\alpha_k\rfloor$, so point (v) avoids an additive term in the running time; the precise choice of k^{9k} is purely for technical convenience. Finally, the purpose of point (iv) is to guarantee monotonicity (together with point (i) and Lemma 8(iv)). Requiring point (iv) is unlikely to affect applications of our results – typically such applications would satisfy either $\alpha_k=0$ (for tracking total query count) or $\alpha_k\geq 1$ (for tracking total running time, where query cost is the running time of a decision algorithm which needs to read its entire input).

▶ **Theorem 10.** *Theorems 1 and 2 remain valid when the cost function $\mathrm{cost}_k(x)=x^{\alpha_k}$ is replaced by an arbitrary regularly-varying parameterised cost function. Likewise, Theorems 4 and 6 remain valid when the the running time $\tilde{O}(n^{\alpha_k})$ of the decision algorithm is replaced by an arbitrary regularly-varying parameterised cost function.*

References

1 Raghavendra Addanki, Andrew McGregor, and Cameron Musco. Non-adaptive edge counting and sampling via bipartite independent set queries. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 2:1–2:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.2.

2 Simon Apers, Yuvan Efron, PawełGawrychowski, Troy Lee, Sagnif Mukhopadhyay, and Danupon Nanongkai. Cut query algorithms with star contraction. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, 2022, to appear. arXiv:2201.05674.

3 Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. *ACM Trans. Algorithms*, 16(4):52:1–52:27, 2020. doi:10.1145/3404867.

4 Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Hyperedge estimation using polylogarithmic subset queries. *CoRR*, abs/1908.04196, 2019. arXiv:1908.04196.

5 Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Faster counting and sampling algorithms using colorful decision oracle. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPIcs*, pages 10:1–10:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.10.

6 Arijit Bishnu, Arijit Ghosh, Sudeshna Kolay, Gopinath Mishra, and Saket Saurabh. Parameterized query complexity of hitting set using stability of sunflowers. In Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao, editors, *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, volume 123 of *LIPIcs*, pages 25:1–25:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ISAAC.2018.25.

7 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *J. Comput. Syst. Sci.*, 87:119–139, 2017. doi:10.1016/j.jcss.2017.03.003.

8 Cornelius Brand, Holger Dell, and Thore Husfeldt. Extensor-coding. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 151–164. ACM, 2018. doi:10.1145/3188745.3188902.

9 Marco Bressan and Marc Roth. Exact and approximate pattern counting in degenerate graphs: New algorithms, hardness results, and complexity dichotomies. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 276–285, 2022. doi:10.1109/FOCS52979.2021.00036.

10 Timothy Chan, Virginia Vassilevska Williams, and Yinzhan Xu. Fredman's trick meets dominance product: Fine-grained complexity of unweighted APSP, 3SUM counting, and more. *CoRR*, abs/2303.14572, 2023. arXiv:2303.14572.

11 Xi Chen, Amit Levi, and Erik Waingarten. Nearly optimal edge estimation with independent set queries. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2916–2935. SIAM, 2020. doi:10.1137/1.9781611975994.177.

12 Holger Dell and John Lapinskas. Fine-grained reductions from approximate counting to decision. *ACM Trans. Comput. Theory*, 13(2):8:1–8:24, 2021. doi:10.1145/3442352.

13 Holger Dell, John Lapinskas, and Kitty Meeks. Approximately counting and sampling small witnesses using a colorful decision oracle. *SIAM J. Comput.*, 51(4):849–899, 2022. doi:10.1137/19m130604x.

14 Holger Dell, John Lapinskas, and Kitty Meeks. Nearly optimal independence oracle algorithms for edge estimation in hypergraphs. *CoRR*, 2024. arXiv:2211.03874, doi:10.48550/arXiv.2211.03874.

15 William Feller. *An introduction to probability theory and its applications. Vol. II.* Second edition. John Wiley & Sons Inc., New York, 1971.

16 Jacob Focke, Leslie Ann Goldberg, Marc Roth, and Stanislav Zivný. Approximately counting answers to conjunctive queries with disequalities and negations. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '22, pages 315–324, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3517804.3526231.

17 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016. doi:10.1145/2886094.

18 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1), March 2007. doi:10.1145/1206035.1206036.

19 Hung Le. Approximate distance oracles for planar graphs with subpolynomial error dependency. In *Proceedings of the 2023 annual ACM-SIAM symposium on discrete algorithms (SODA)*, pages 1877–1904, 2023.

20 Daniel Lokshtanov, Andreas Björklund, Saket Saurabh, and Meirav Zehavi. Approximate counting of k-paths: Simpler, deterministic, and in polynomial space. *ACM Trans. Algorithms*, 17(3):26:1–26:44, 2021. doi:10.1145/3461477.

21 Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Efficient computation of representative weight functions with applications to parameterized counting (extended version). In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 179–198. SIAM, 2021. doi:10.1137/1.9781611976465.13.

22 Kitty Meeks. The challenges of unbounded treewidth in parameterised subgraph counting problems. *Discrete Applied Mathematics*, 198:170–194, 2016. doi:10.1016/j.dam.2015.06.019.

23 Kitty Meeks. Randomised enumeration of small witnesses using a decision oracle. *Algorithmica*, 81(2):519–540, 2019. doi:10.1007/s00453-018-0404-y.

24 Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: Sequential, cut-query, and streaming algorithms. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 496–509, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3357713.3384334.

25 Moritz Müller. Randomized approximations of parameterized counting problems. In Hans L. Bodlaender and Michael A. Langston, editors, *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, volume 4169 of *Lecture Notes in Computer Science*, pages 50–59. Springer, 2006. doi:10.1007/11847250_5.

26 Pan Peng and Jiapeng Zhang. Towards a query-optimal and time-efficient algorithm for clustering with a faulty oracle. In *Proceedings of Machine Learning Research (COLT)*, volume 134, pages 1–19, 2021.

27 Cyrus Rashtchian, David P. Woodruff, and Hanlin Zhu. Vector-matrix-vector queries for solving linear algebra, statistics, and graph problems. In Jaroslaw Byrka and Raghu Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2020, August 17-19, 2020, Virtual Conference*, volume 176 of *LIPIcs*, pages 26:1–26:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.26.

28 Jakub Tetek and Mikkel Thorup. Edge sampling and graph parameter estimation via vertex neighborhood accesses. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1116–1129. ACM, 2022. doi:10.1145/3519935.3520059.

29 Marc Thurley. An approximation algorithm for #k-SAT. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 78–87. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.78.

30 Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47:85–93, 1986. doi:10.1016/0304-3975(86)90135-0.

31 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.

32 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893.

Exploiting Automorphisms of Temporal Graphs for Fast Exploration and Rendezvous

Konstantinos Dogeas
Department of Computer Science, Durham University, UK

Thomas Erlebach
Department of Computer Science, Durham University, UK

Frank Kammer
THM, University of Applied Sciences Mittelhessen, Gießen, Germany

Johannes Meintrup[1]
HM, University of Applied Sciences Mittelhessen, Gießen, Germany

William K. Moses Jr.
Department of Computer Science, Durham University, UK

Abstract

Temporal graphs are dynamic graphs where the edge set can change in each time step, while the vertex set stays the same. Exploration of temporal graphs whose snapshot in each time step is a connected graph, called connected temporal graphs, has been widely studied. In this paper, we extend the concept of graph automorphisms from static graphs to temporal graphs and show for the first time that symmetries enable faster exploration: We prove that a connected temporal graph with n vertices and orbit number r (i.e., r is the number of automorphism orbits) can be explored in $O(rn^{1+\epsilon})$ time steps, for any fixed $\epsilon > 0$. For $r = O(n^c)$ for constant $c < 1$, this is a significant improvement over the known tight worst-case bound of $\Theta(n^2)$ time steps for arbitrary connected temporal graphs. We also give two lower bounds for temporal exploration, showing that $\Omega(n \log n)$ time steps are required for some inputs with $r = O(1)$ and that $\Omega(rn)$ time steps are required for some inputs for any r with $1 \leq r \leq n$.

Moreover, we show that the techniques we develop for fast exploration can be used to derive the following result for rendezvous: Two agents with different programs and without communication ability are placed by an adversary at arbitrary vertices and given full information about the connected temporal graph, except that they do not have consistent vertex labels. Then the two agents can meet at a common vertex after $O(n^{1+\epsilon})$ time steps, for any constant $\epsilon > 0$. For some connected temporal graphs with the orbit number being a constant, we also present a complementary lower bound of $\Omega(n \log n)$ time steps.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases dynamic graphs, parameterized algorithms, algorithmic graph theory, graph automorphism, orbit number

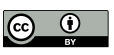
Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.55

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2312.07140` [25]

Funding *Johannes Meintrup*: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 379157101.

[1] Corresponding Author



51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 55; pp. 55:1–55:18

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

For many decades, graph theory has been a tool used to model and study many real world problems and phenomena [15]. A usual assumption for many of these problems is that the graphs have a fixed structure. However, there are quite a number of cases where the structure of a system changes over time. For example, consider the problem of routing in transportation networks (roads, rails) where specific connections can become unavailable (due to a disaster) or they are active during specific times (due to safety). Such scenarios can be modeled using temporal graphs, a sequence of graphs over the same vertex set where the edges possibly change in each time step. The temporal graph setting has received significant interest from the research community in the recent past as seen in recent surveys [19, 48].

In this paper, we study two problems on temporal graphs. The first is the *temporal exploration problem* (TEXP), which has been studied, e.g., by Michail and Spirakis [49, 50] and by Ilcinkas et al. [37]. It requires an agent to explore all vertices of the temporal graph as quickly as possible. The second is the *temporal rendezvous problem* (TRP), which we formulate for the first time in this paper. It requires two heterogeneous agents (in terms of the programs they run) to rendezvous on a temporal graph when they cannot communicate with one another. For both problems, we assume that each agent has complete knowledge of the temporal graph in advance (a common assumption [3, 14, 27, 28, 30, 31, 37, 49, 50, 62]). However, in the case of TRP the agents may have different names for the same vertices, i.e., the local labels of the vertices may be different.

The problem of exploration has been well studied in the static setting since it was introduced in 1951 by Shannon [59]. It has also been intensively studied in the temporal graph setting since 2014 (see all references from the previous paragraph). On an application-oriented note, the problem captures the setting where a person is trying to visit various parts of a city using public transportation. For example, train schedules involve multiple train stations (vertices) with trains running between them at different times (i.e., repeatedly changing edges). Thus, planning a visit to multiple destinations over a given day using railways is an example of solving TEXP.

The rendezvous problem can be broadly categorized into two types: symmetric and asymmetric rendezvous. The version where agents have the same strategy (symmetric rendezvous) was introduced by Alpern [10]. The version where agents can have distinct strategies (asymmetric rendezvous) was introduced by Alpern [7] and is the focus of research in this paper. TRP is a natural extension of the asymmetric rendezvous problem to the dynamic setting. As a real world example, consider a pair of tourists who want to explore a city together and have to agree on a strategy to meet up in case they are separated and their cell phones die. In this scenario, they may use public transportation (dynamically changing network) to meet and agree in advance to use different strategies that guarantee that they meet quickly.

In this paper, we present results that extend the literature in two ways. Firstly, we formalize the TRP problem in the setting where agents have complete knowledge of the temporal graph a priori and develop good upper and lower bounds for it. Secondly, we utilize interesting structural properties (namely automorphisms of a graph and associated orbits of vertices) in order to analyze bounds on the number of time steps required by temporal walks that solve certain problems. In particular, we develop upper and lower bounds for both TEXP and TRP that leverage the aforementioned graph properties. To the best of our knowledge, this is the first work that takes advantage of these graph properties to study problems in temporal graphs.

1.1 Our Contributions

We present results for two problems: first, the TEMPORAL EXPLORATION PROBLEM (TEXP) which, for a given temporal graph $\mathcal{G}$, asks for a temporal walk that visits all vertices of $\mathcal{G}$, and secondly, the TEMPORAL RENDEZVOUS PROBLEM (TRP), which considers two agents that try to meet in the given temporal graph, meaning they must be stationed at the same vertex in the same time step.

One of our primary contributions is formalizing the problem of TRP in Section 2, and in doing so extending the problem of asymmetric rendezvous to the temporal graph setting where agents have complete knowledge of the temporal graph in advance. Another significant contribution is that we show how to leverage the use of a structural graph property, namely the automorphism group of a temporal graph and the associated notion of orbits, to bound the number of time steps required by algorithms we devise for these problems. To the best of our knowledge, this work is the first instance of leveraging such properties to study any problem on temporal graphs apart from a practical implementation of a generator for listing all non-isosomorphic simple temporal graphs [18]. Intuitively, an automorphism of a graph is a mapping from the set of vertices of the graph to the same set of vertices that preserves the neighborhood relation between the vertices. The set of automorphisms of a temporal graph consists of the *intersection* of the sets of automorphisms of the graph at each time step. The set of automorphisms of the temporal graph, along with function composition as group operation, forms the automorphism group of the temporal graph. An orbit of the automorphism group of a temporal graph is a maximal set of vertices such that each vertex can be mapped to any other vertex in the set via an automorphism of the group. Intuitively, the vertices in an orbit look indistinguishable to an agent that has full information about the temporal graph but without meaningful vertex labels. As the agents are able to compute a consistent numbering of the orbits (Lemma 13), they can agree to meet in some specific orbit, but not at a specific vertex. Therefore, temporal graphs where all orbits are large (or that even have a single orbit containing all vertices) appear to be the most challenging graphs for solving TRP. Our result providing fast exploration schedules in temporal graphs with few orbits is therefore a crucial ingredient for enabling our solution to TRP to handle all possible temporal graphs (including those with a single orbit).

We give precise definitions and present further preliminaries in Section 2. Then, we introduce some useful utilities related to automorphisms in Section 3. These will be used in later sections, where we present the following results.

Upper bounds. In Section 4, we develop a deterministic algorithm to solve TEXP in $O(rn^{1+\epsilon})$ time steps for any fixed $\epsilon > 0$ (see Corollary 12), where n is the number of vertices in the temporal graph and r is the number of orbits of the automorphism group of the temporal graph. Note that r can range in value from 1 to n. Thus, for $r = O(n^c)$ for constant $c < 1$, this is a significant improvement over the known tight worst-case bound of $\Theta(n^2)$ time steps for arbitrary connected temporal graphs [27]. In Section 5, we leverage this algorithm for TEXP to develop a deterministic solution for TRP using $O(n^{1+\epsilon})$ time steps for any fixed $\epsilon > 0$ (see Theorem 14). Our focus is on bounding the time steps of the temporal walks required to solve TEXP and TRP, and not on optimizing the running time of the respective algorithms to compute such walks.

Lower bounds. We complement our algorithms with lower bounds for both TEXP and TRP in Section 6. In particular, we design an instance of TRP such that any solution for it requires $\Omega(n \log n)$ time steps (see Theorem 16). We then show how this translates to a

lower bound of $\Omega(n \log n)$ time steps for some instances of TEXP where the temporal graph at hand has an orbit number $r = O(1)$ (see Corollary 17). By revisiting the lower bound of [27] for TEXP, which focused on arbitrary temporal graphs, and studying it through the lens of automorphisms and orbits, we can obtain a more fine-grained lower bound of $\Omega(rn)$ time steps (see Lemma 15) for some temporal graphs with orbit number $1 \leq r \leq n$. Notice that the multiplicative gap between our upper and lower bounds for both problems is only a factor of $O(n^\epsilon)$.

Relevance of the orbit number. It is well known in graph theory that almost all (static) graphs are *rigid* [12], meaning that the automorphism group contains only the trivial identity function. This is in contrast to observations in practice. Many real-world graphs have non-trivial automorphisms. A recent analysis of real-world graphs in the popular database `https://networkrepository.com` showed that over 70% of the analyzed graphs had non-trivial automorphisms [12]. One may reasonably expect that real-world temporal graphs have similar properties. Symmetries are also abundant in graphs arising in chemistry [11], which have been studied in temporal settings as well [57]. We believe that these observations provide a strong motivation for studying temporal graph problems for temporal graphs with fewer than n orbits, e.g., with algorithms parameterized via the number of orbits. Furthermore, we emphasize that our results for TRP hold for all temporal graphs, independent of the orbit number parameter, even though they utilize techniques we develop for TEXP parameterized by the orbit number. Roughly speaking, for orbit number r, the penalty factor r in the number of time steps to solve TEXP is saved when solving TRP by focusing only on a smallest orbit of size at most n/r.

1.2 Technical Overview and Challenges

Firstly, we give an intuitive overview of our upper bound results. The concepts used in this section are more precisely defined in the next sections.

For TEXP, we first consider the problem of visiting all the vertices of one orbit S. One key insight is that, if we have a temporal walk W that visits k vertices of S, we can use the automorphisms of the temporal graph to transform W into other walks that visit different sets of k vertices of S (Lemma 3). Therefore, even if the k vertices visited by W have already been explored earlier, we can transform W into a temporal walk W' that visits a "good" number of previously unexplored vertices of S. The number of previously unexplored vertices of S that W' is guaranteed to visit increases with the number of possible start vertices in S that we allow for W', but the larger that set X of possible start vertices is, the longer it may take the agent to reach the best start vertex in that set. A challenge is to analyze this tradeoff. By carefully relating the size of X to the guaranteed number of unexplored vertices that a walk W' starting at a vertex in X can visit (Corollary 4), we manage to balance the number of time steps needed to move to the start vertex of W' and the number of previously unexplored vertices that W' visits. To show that vertices of X can be reached quickly, we study the structure of edges that connect vertices in different orbits (Lemma 5) and use it to analyze reachability between orbits (Lemmas 6 and 7).

We then employ a recursive construction: We recursively construct a temporal walk W_1 that visits k vertices of S, concatenate W_1 with a temporal walk that moves quickly from the endpoint of W_1 to a good start vertex in S for what follows, and then use a second recursively constructed walk (transformed via an automorphism to a "good" temporal walk W_2 starting in a vertex of S) to visit "nearly" k further vertices of S. A careful analysis then shows that in this way we can visit a constant fraction of the vertices of S in $O(r|S|^{1+\epsilon})$ steps

(Lemma 9 and Corollary 10). We then show that the concatenation of $O(\log |S|)$ such walks (each one again transformed via an automorphism to maximize the number of newly explored vertices) suffices to visit all vertices of S in $O(r|S|^{1+\epsilon} + n \log |S|)$ time steps (Theorem 11), which can be bounded by $O(rn^{1+\epsilon'})$ time steps). By visiting the r orbits one after another, we can finally show that the whole temporal graph can be explored in $O(rn^{1+\epsilon'})$ time steps (Corollary 12).

For TRP, a simple and fast solution one first thinks of is to have the agents simply meet at a vertex with a specific label. However, this is not feasible in the model we consider. In particular, we assume that the agents cannot communicate and, while both agents have complete information about the temporal graph, they do not have access to consistent vertex labels. As such, the agents are unable to agree upon the same vertex based on the vertex label. We rely, instead, on structural graph properties and let the agents meet at a vertex in a smallest orbit: One agent moves to any vertex in that orbit, and the other searches all vertices in that orbit. The first challenge is for the agents to independently identify the same orbit in which to meet. We show that this can be done by letting each agent enumerate all temporal graphs, together with colorings of their orbits, until it encounters for the first time a temporal graph that is isomorphic to the input graph in which TRP is to be solved. In this way both agents obtain the same colored temporal graph and can independently select, among all orbits of smallest size, the one with smallest color (Lemma 13). Then one agent moves to a vertex in that orbit S, while the other searches all vertices of S. The challenge here is to deal with orbits that are large, because previously known techniques would require $\Theta(n^2)$ time steps to explore an orbit of size $\Omega(n)$. Fortunately, this is where our above-mentioned results for exploration of (orbits of) temporal graphs come to the rescue: As $|S| \leq n/r$, we have $O(r|S|^{1+\epsilon} + n \log |S|) = O(n^{1+\epsilon'})$, and hence S can be explored (and TRP solved) in $O(n^{1+\epsilon'})$ time steps (Theorem 14).

Now we turn to our lower bounds. For TEXP, we first observe that the existing lower bound construction that shows that $\Omega(n^2)$ time steps are necessary for exploration on some temporal graphs uses temporal graphs with $\frac{n}{2}+1$ orbits. By slightly varying the construction, we can show for any r in the range from 1 to n that there are temporal graphs with r orbits that require $\Omega(rn)$ time steps for exploration (Lemma 15). Our main lower bound result is the lower bound of $\Omega(n \log n)$ for TRP in temporal graphs with a single orbit (Theorem 16). The temporal graph is a cycle on $n = 2^m - 1$ vertices in every step, and the edges change every $\lfloor n/16 \rfloor$ time steps. Each period of $\lfloor n/16 \rfloor$ time steps in which the edges do not change is called a *phase*. We number the vertices of the cycle from 0 to $n-1$ and consider the binary representation of the vertex labels. In phase 1, each vertex j is adjacent to $j+1$ and $j-1$, while in any phase $i > 1$, it is adjacent to $j + 2^{2i}$ and $j - 2^{2i}$ (with all computations done modulo n). As all vertices are indistinguishable to the agents, we can force each agent to make the same movements no matter where we place it initially. By fixing the five highest-order bits of the agents' start positions in a certain way, we can ensure that the agents do not meet in the first phase, no matter how the remaining bits of their start positions are chosen. Depending on the positions where the agents end up at the end of each phase (relative to their start position in that phase), we fix a certain number of bits of the binary representations of the start positions of the agents: After the first phase, we fix the lowest four bits, and after each further phase, we fix the next-higher two bits. We can show that this is sufficient to ensure that the agents start the next phase at vertices that are sufficiently far from each other in the cycle of that phase. This process can be repeated $\Omega(\log n)$ times, showing that it takes $\Omega(n \log n)$ time steps for the agents to meet. This lower bound for TRP implies also that exploration of the constructed temporal graph requires $\Omega(n \log n)$ steps (Corollary 17).

1.3 Related Work

There is a divide in the research on temporal graphs with respect to the amount of knowledge known in advance by the agents on the graph. When complete knowledge of the temporal graph is known in advance to the agents, as is assumed in this paper, the setting is sometimes referred to as the *post-mortem* setting (see, e.g., Santoro [58]). Since we consider scenarios where agents plan temporal walks using advance knowledge of future time steps, we refer to the post-mortem setting as the *clairvoyant* setting in the remainder of this section.

The clairvoyant setting is in contrast to the *live* setting where agents only have partial knowledge of the temporal graph, and the solutions to problems in these different settings are of a different nature. The reader can refer to the survey by Di Luna [22] for more information on problems and solutions in the live setting. We restrict the rest of this related work section to work in the clairvoyant setting.

Exploration. We trace the progress of solving instances of TEXP, first by identifying which assumptions are required for the problem to even be solvable, and then by identifying properties that were leveraged to give faster and faster solutions.

Michail and Spirakis [49, 50] showed that it is **NP**-complete to decide whether a given temporal graph with a given *lifetime*, i.e., the number of time steps it exists, is explorable when no assumptions are made on the input graph. This holds even if the graph is connected in every time step, termed *connected* (and sometimes called *always-connected* in the literature). Even when a restriction is placed on the *underlying graph*, i.e., the union of the graphs at each time step, such that the underlying graph has pathwidth at most 2, Bodlaender and van der Zanden [14] showed that TEXP is **NP**-complete.

However, the exploration is always possible when the lifetime of the graph is sufficiently large. In particular, Michail and Spirakis [50] showed that a connected graph with n vertices may be explored in $O(n^2)$ time steps. For the rest of the related work on TEXP, we focus on the case of connected temporal graphs with a sufficiently large lifetime. Erlebach et al. [27] showed that exploration on arbitrary temporal graphs takes $\Omega(n^2)$ time steps. By restricting their study of temporal graphs to those where the underlying graph belongs to a special graph class, however, they showed that TEXP can be solved in $o(n^2)$ time steps in several such cases. In particular, when the underlying graph is planar, has bounded treewidth k, or is a $2 \times n$ grid, they showed that TEXP can be solved in $O(n^{1.8} \log n)$ time steps, $O(n^{1.5} k^{1.5} \log n)$ time steps, and $O(n \log^3 n)$ time steps, respectively. They also showed a lower bound of $\Omega(n \log n)$ when the underlying graph is a planar graph of degree at most 4.

The study of TEXP when the temporal graph is restricted continued in several papers. Taghian Alamouti [62] showed that TEXP can be solved in $O(k^2(k!)(2e)^k n)$ time steps when the underlying graph is a cycle with k chords. Adamson et al. [3] improved this to $O(kn)$ time steps. They also improved the upper bounds on TEXP for underlying graphs that have bounded treewidth k or are planar to $O(kn^{1.5} \log n)$ and $O(n^{1.75} \log n)$, respectively. In addition, they strengthened the lower bound for underlying planar graphs by showing that even if the degree is at most 3, the lower bound is $\Omega(n \log n)$. Erlebach et al. [28] further improved work on bounded degree underlying graphs by showing that TEXP can be solved in $O(n^{1.75})$ time steps for such temporal graphs. Ilcinkas et al. [37] showed that when the underlying graph is a cactus, the exploration time is $2^{\Theta(\sqrt{\log n})} n$ time steps.

Other variants of TEXP have been studied where the problem is slightly different, or the edges of the temporal graph vary in some particular way (e.g., periodically, T-interval connected, k-edge deficient), or multiple agents explore the graph [1, 2, 5, 17, 30, 31, 38].

Rendezvous. Symmetric rendezvous [10] and asymmetric rendezvous [7] have received much interest over the years, resulting in numerous surveys [8, 9, 32, 40, 41, 54, 55] being written on the problems covering different settings. In this work, we are the first to extend asymmetric rendezvous to temporal graphs in the clairvoyant setting. Note that there has been previous work [16, 21, 23, 51, 53, 60, 61] that has studied multi-agent rendezvous (also called gathering) in a dynamic graph setting, but that was in the live setting. For an overview of rendezvous in static graphs we refer the reader to [9, 41]. Rendezvous has been studied in deterministic settings [13, 56] and asynchronous settings [24, 43], and has been shown to be solvable in logarithmic space [20].

Other Related Work. Other problems have also been studied in the temporal graph setting, e.g., matchings [46], separators [34, 63], vertex covers [6, 36], containments of epidemics [26], Eulerian tours [17, 44], graph coloring [45, 47], network flow [4], treewidth [33] and cops and robbers [29, 52]. See the survey by Michail [48] for more information.

2 Preliminaries

Basic terminology. We use standard graph terminology. All static graphs are assumed to be simple and undirected. We write $[x] = \{1, \dots, x\}$ for any integer x.

A *temporal graph* $\mathcal{G} = (G_1, \dots, G_\ell)$ is a sequence of static graphs, all with the same vertex set V. For a temporal graph $\mathcal{G}$, the graph $G = (V, \bigcup_{t\in[\ell]} E(G_t))$ is the *underlying graph of* $\mathcal{G}$. We call $\mathcal{G}$ connected if each G_t for $t \in [\ell]$ is connected. We use n to refer to the number of vertices of the (temporal) graph under consideration. A *temporal walk* W is a walk in a temporal graph $\mathcal{G}$ that is time respecting, i.e., traverses edges in strictly increasing time steps. For a temporal walk W starting at time step t we write $W = (u_1, u_2, u_3, \dots)$ to mean the temporal walk starts at vertex u_1 in time step t, is at vertex u_2 at the beginning of time step $t + 1$, and so forth. Note that subsequent vertices in the temporal walk can be the same vertex, i.e., we can wait for an arbitrary number of time steps at a vertex. We say that a temporal walk *spans* x time steps if it starts at some time step t and ends at some time step t' with $t' - t = x$. We assume that the lifetime ℓ is large enough such that a desired temporal walk can be constructed. For all our results it is enough to assume $\ell \geq n^2$, as n^2 time steps suffice for TEXP [27] and therefore also for TRP. For any function $f : X \to X$ for a universe X we write $f^i(x)$ when we mean applying the function i-times iteratively for any $x \in X$ and integer i, e.g., $f^2(x) = f(f(x))$. We use $\circ$ to denote function composition, i.e., for two functions $f : X \to X$ and $g : X \to X$, we denote with $f \circ g$ the function that maps $x \in X$ to $f(g(x))$.

Isomorphisms and automorphisms. Two static graphs G and H are *isomorphic* exactly if a bijection $\theta : V(G) \to V(H)$ (called an *isomorphism*) exists with the following property: two vertices u, v are adjacent in G exactly if $\theta(u)$ is adjacent to $\theta(v)$ in H. We write $G \cong H$ if G is isomorphic to H. An *automorphism* is an isomorphism from a graph G to itself. The set of all automorphisms of a graph G forms a group $\mathrm{Aut}(G)$, with $\circ$ as group operation. Refer to [35, 39, 42] for further reading on the topic of isomorphisms and automorphisms of graphs.

For a temporal graph $\mathcal{G}$ with lifetime ℓ we denote with $\mathrm{Aut}(\mathcal{G})$ the set of all functions σ such that σ is an automorphism of each graph G_t at every time step $t \in [\ell]$ (and hence also an automorphism of the underlying graph G). $\mathrm{Aut}(\mathcal{G})$ with $\circ$ as group operation is the automorphism group of the temporal graph $\mathcal{G}$. We remark that the automorphism group of the temporal graph $\mathcal{G}$ is the intersection of the automorphism groups of the graphs G_t for

$t \in [\ell]$ and that the automorphism group of $\mathcal{G}$ can be viewed as the automorphism group of the underlying graph G of $\mathcal{G}$ with edge labels such that each edge is labeled with the set of time steps in which it occurs.

The *orbit* of a vertex u in a temporal graph $\mathcal{G}$ with respect to $\mathrm{Aut}(\mathcal{G})$ is the set $V' \subseteq V$ of all vertices that u can be mapped to by automorphisms in $\mathrm{Aut}(\mathcal{G})$. Note that, if V' is the orbit of u, then the orbit of every vertex in V' is also V': For any two vertices v, v' in V', the automorphisms σ and σ' that map u to v and v', respectively, can be composed to the automorphism $\sigma' \circ \sigma^{-1}$ that maps v to v'. Furthermore, there cannot be an automorphism ρ that maps v to a vertex outside V' because $\rho \circ \sigma$ would then be an automorphism that maps u to a vertex outside V'; a contradiction to the definition of the orbit V' of u. We denote by $\mathcal{G}/\mathrm{Aut}(\mathcal{G})$ the set of all orbits of the vertices of $\mathcal{G}$. Note that this set forms a partition of V. We call $|\mathcal{G}/\mathrm{Aut}(\mathcal{G})|$ the *orbit number* of $\mathcal{G}$. We call an edge $\{u, v\} \in E(G_t)$ for $t \in [\ell]$ an *orbit boundary edge* if u and v are in different orbits, and *inner orbit edge* otherwise. Two orbits connected by an orbit boundary edge in time step t are called *adjacent (in time step t)*.

We use automorphisms to transform temporal walks, as outlined in the following. For a temporal graph $\mathcal{G}$ with lifetime ℓ and any automorphism $\sigma \in \mathrm{Aut}(\mathcal{G})$ and any temporal walk $W = (u_t, u_{t+1}, \ldots, u_{t'})$ that starts at time t and ends at time t' with $t, t' \in [\ell]$, we say we *apply σ to W* when we construct the temporal walk $W' = (\sigma(u_t), \sigma(u_{t+1}), \ldots, \sigma(u_{t'}))$.

Temporal Exploration Problem. The TEMPORAL EXPLORATION PROBLEM (TEXP) is defined for a given temporal graph $\mathcal{G}$ with vertex set V and lifetime ℓ and asks for the existence of a temporal walk W such that W starts at a given vertex $u \in V$ and visits all vertices of V. In connected temporal graphs with sufficiently large lifetime such a walk always exists, and this is the setting we consider throughout this work. As such, the question of existence is no longer of interest, and instead we focus on the time span of temporal walks that start at u at time step 1 and visit all (or a certain set of) vertices.

The following is an adaptation of a result of Erlebach et al. [27], slightly modified to fit our notation and use case. Intuitively speaking, the lemma states that with every extra time step at least one additional vertex becomes reachable. Thus, starting from any vertex at any time step, we can reach any particular other vertex in $n - 1$ steps. Consequently, there is always a temporal walk that visits all vertices of $\mathcal{G}$ within n^2 time steps.

▶ **Lemma 1** (Reachability, [27]). *Let $\mathcal{G}$ be a connected temporal graph with vertex set V and lifetime ℓ. Denote by $R_{t,t'}(u)$ the set of vertices reachable by some temporal walk starting at vertex $u \in V$ at time step $t \in [\ell]$ and ending at time step $t' \in [\ell]$ with $t' \geq t$. If $R_{t,t'}(u) \neq V$ and $t' < \ell$, then $R_{t,t'}(u) \subsetneq R_{t,t'+1}(u)$.*

Temporal Rendezvous Problem. We consider a problem we call TEMPORAL RENDEZVOUS PROBLEM (TRP) defined as follows. Let $\mathcal{G}$ be a temporal graph with vertex set V and lifetime ℓ. Two agents a_1, a_2 are placed at arbitrary vertices $u_1, u_2 \in V$ (respectively) in G_1 of $\mathcal{G}$. They compute respective temporal walks W_1 and W_2 such that the agents meet at the same vertex in the same time step at least once during these walks. The agents have the full information of $\mathcal{G}$ available, but they cannot communicate with each other. Furthermore, they do not know the location of the other agent, and the vertex labels that one agent sees can be arbitrarily different from the vertex labels that the other agent sees. The lack of consistent labels prohibits trivial solutions such as the two agents agreeing to meet at a vertex with a specific label, e.g., the lowest labeled vertex. We call such agents *label-oblivious*. A solution to TRP is a pair of possibly different programs p_1 and p_2 for agents a_1 and a_2, respectively, that the agents use to compute and execute their temporal walks. The respective start

positions $u_1, u_2 \in V$ of the agents (and the vertex labels that each agent sees) are chosen by an all-knowing adversary once a solution (p_1, p_2) is provided. As we assume that $\ell \geq n^2$, there is always a solution that ensures that the agents meet: Agent a_1 simply waits at its start vertex, while agent a_2 explores the whole graph and visits every vertex, which is always possible within n^2 time steps as mentioned above. Therefore, we are interested in solutions that enable the agents to meet as early as possible and aim to obtain worst-case bounds significantly better than n^2 on the number of steps that are required for TRP.

3 Automorphism Utilities

We now introduce some helpful utilities regarding automorphisms that we use in the following sections. Intuitively, we build up to a framework that allows us to transform a temporal walk W into a temporal walk W' that visits more vertices that are desirable (with respect to the exploration goal) than W does, by applying a well-chosen automorphism to W. The following is needed to give specific guarantees that the transformed temporal walks must uphold. Throughout this section we use and extend techniques of the field of algebraic graph theory [35, 39, 42], adapted to our specific use cases.

For this, we begin with some definitions. Let $\mathcal{G}$ be a temporal graph with vertex set V and let $S \in \mathcal{G}/\mathrm{Aut}(\mathcal{G})$ be any orbit. For any $X \subseteq S$ and any $u \in S$ denote with $\mathrm{Aut}(\mathcal{G})[u, X]$ the set of all automorphisms $\sigma \in \mathrm{Aut}(\mathcal{G})$ that map u to any vertex $v \in X$. We use the shorthand $\mathrm{Aut}(\mathcal{G})[u, x]$ for $\mathrm{Aut}(\mathcal{G})[u, \{x\}]$. We can then show the following.

▶ **Lemma 2.** *Let $S \in \mathcal{G}/Aut(\mathcal{G})$ be any orbit. Then $|Aut(\mathcal{G})[u, x_1]| = |Aut(\mathcal{G})[u, x_2]|$ for any $u, x_1, x_2 \in S$.*

Let $S \in \mathcal{G}/\mathrm{Aut}(\mathcal{G})$. We now consider a special 2-dimensional *automorphism matrix* $\mathcal{M}_{u,Y,X}$ for any $u \in S$, $Y \subseteq S$, and $X \subseteq S$. It has columns $C_1, C_2, \ldots, C_{|Y|+1}$, and a row for each $\sigma \in \mathrm{Aut}(\mathcal{G})[u, X]$. We refer to the row for some σ simply as row σ. The entry in row σ of column C_1 is $\sigma(u)$. Each vertex $y \in Y$ is assigned a unique column among $C_2, \ldots C_{|Y|+1}$ in an arbitrary way. The entry in row σ of the column to which y is assigned is $\sigma(y)$. We now give some intuition about the application of an automorphism matrix. Assume that we are constructing a temporal walk W that has already visited a set $T \subsetneq S$ and we want to extend it to visit the vertices of $S \setminus T$ (with S an arbitrary orbit). To facilitate this extension we first construct a temporal walk W' that visits at least a certain number of vertices of S, but that is not guaranteed to visit vertices of $S \setminus T$. This walk W' cannot be used to extend W in a meaningful way. Instead, using the automorphism matrix we show that there always exists an automorphism $\sigma \in \mathrm{Aut}(\mathcal{G})$ that we can apply to W' to obtain a temporal walk W_σ that visits a guaranteed fraction of vertices of $S \setminus T$. We can then use W_σ as our desired extension. Figure 2 visualizes this concept. In later sections we will show how repeated application of such extensions leads to temporal walks that visit all vertices of an orbit S. Note that the value p in the following lemma represents the fraction of vertices that we still need to visit compared to all vertices in the orbit under consideration. While T is a set of already visited vertices in the application sketched above, the following lemma and Corollary 4 are formulated more generally for arbitrary sets $T \subseteq S$.

▶ **Lemma 3.** *Let $\mathcal{G}$ be a connected temporal graph with lifetime ℓ and let $S \in \mathcal{G}/Aut(\mathcal{G})$ be any orbit. Let $T \subseteq S$ with $p = (|S| - |T|)/|S|$ and $W = (u_1, u_2, \ldots, u_x)$ a temporal walk starting at time step t and ending at t' with $t, t' \in [\ell]$ and $u_1 \in S$ such that W visits k vertices of S. Then there exists a temporal walk W' starting at a vertex $u' \in S$ in time step t and ending at time step t' that visits at least pk vertices of $S \setminus T$.*

When restricting the possible start vertices where the temporal walk W' of Lemma 3 is allowed to start, we obtain the following corollary. In detail, we are given a set $X \subset S$ such that the walk W' is only allowed to begin at a vertex $u' \in X$. In Lemma 3, W' was allowed to start at any vertex of S. Our use case for the following corollary is that X is a set of vertices that can be reached faster, making them better candidates for start vertices when extending walks in the way sketched above Lemma 3.

▶ **Corollary 4.** *Let $\mathcal{G}$ be a connected temporal graph with lifetime ℓ and let $S \in \mathcal{G}/Aut(\mathcal{G})$ be any orbit. Let $T \subsetneq S$ and W a temporal walk starting at time step t and ending at t' with $t, t' \in [\ell]$ such that the first vertex of W is in S and such that W visits k different vertices of S. For any $X \subseteq S$ with $|X| > |T|$ there exists a temporal walk W' starting at a vertex $u' \in X$ at time step t and ending at time step t' that visits at least $(c-1)/c \cdot k$ vertices of $S \setminus T$, with $c = |X|/|T|$.*

4 Upper Bounds for TEXP

A common approach to build a temporal walk for TEXP is to use Lemma 1, i.e., to construct a (large) set X of reachable vertices so that an unseen vertex v of the current walk is in the set X and the walk can then be extended by v. We are interested in exploring the vertices of one orbit quickly, as this will be useful for TRP in Section 5 where the agents try to meet in one orbit, and for TEXP because we can explore a temporal graph orbit by orbit. Therefore, we want to find walks visiting many vertices of one orbit. Our approach is similar to the common approach mentioned above, and so we want to construct a (large) set X of reachable vertices, but now with the property that X is a subset of the orbit under consideration. To construct X, we show in Lemma 6 a kind of "reachability between orbits."

To describe this in more detail, we need the concept of so-called lanes. Intuitively, lanes are defined for a set of vertices that are all contained in some single orbit, and give us knowledge about the vertices that are quickly reachable while only using orbit boundary edges in each time step. Using this concept of lanes we derive a first result for exploring a single large orbit with a temporal walk that spans $O((n^{5/3}+rn)\log n)$ time steps (Theorem 8). In the proof of that lemma we build the final temporal walk iteratively, by concatenating multiple smaller temporal walks. To make sure each new such small temporal walk visits a desired number of vertices not yet visited, we use Lemma 3, which – informally – lets us transform temporal walks that visit too many previously visited vertices into temporal walks that visit many previously unvisited vertices.

We follow this up with a more refined technique that considers the size of the orbit S one wants to explore as a parameter, but also uses the concept of lanes and walk transformations sketched above. It gives us an upper bound of $O(|S|^{1+\epsilon} + n\log|S|)$, for any constant $\epsilon > 0$. This result is formulated in Theorem 11. Finally, we use a repeated application of Theorem 11 to achieve an upper bound for TEXP of $O(rn^{1+\epsilon})$. We start with an auxiliary lemma that focuses on the orbit boundary edges between two orbits.

▶ **Lemma 5.** *Let G_t be the graph at time step t in a connected temporal graph $\mathcal{G}$ and $S, S' \in \mathcal{G}/Aut(\mathcal{G})$, and let G' be the subgraph of G_t that contains only orbit boundary edges. Then all vertices in S have the same degree in the bipartite graph $G'[S \cup S']$.*

To describe reachability between orbits, we have to introduce some extra notation. Let $\mathcal{G}$ be a temporal graph with lifetime ℓ and vertex set V and $S \in \mathcal{G}/\mathrm{Aut}(\mathcal{G})$ be an orbit. We call a *lane* $L_{t,t'}(X)$ with $X \subseteq S$ and $t, t' \in [\ell]$ the set of all vertices reachable from any $u \in X$ in G by any temporal walk W that only uses orbit boundary edges and starts in time step t and ends in time step at most t'. We write $L_{t,t'}(u)$ instead of $L_{t,t'}(\{u\})$. See Fig. 1 for some intuition.

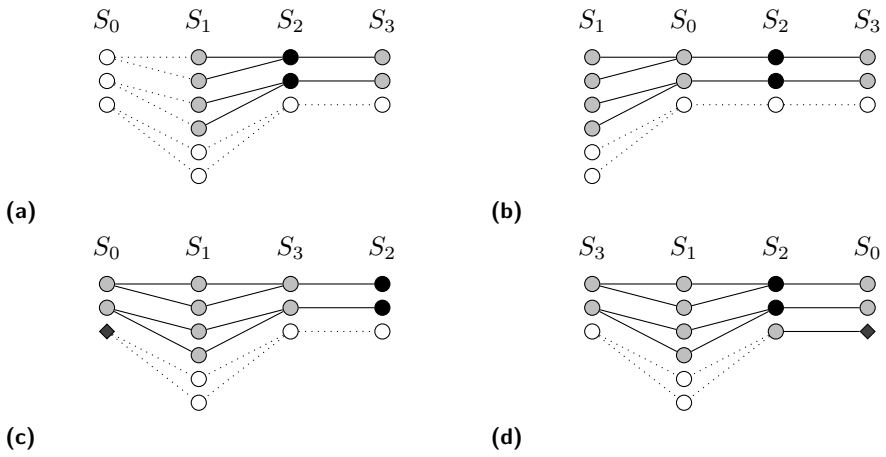


Figure 1 Visualization of the properties of a lane $L_{t,t+r}(X)$. Each vertex in the set X is colored black, and the set of reachable vertices in each time step is colored gray. Each figure represents one additional time step. In the second time step (Figure b) all vertices of the lane are reachable. In the third time step (Figure c) one vertex outside the lane must be reachable, for example the diamond shaped vertex. This is due to the fact that the temporal graph at hand is connected, and thus at least one additional vertex is reachable with every next time step. Here, one additional time step then suffices to reach a vertex of $S_2 \setminus X$ (Figure d).

The next lemma gives us a lower bound on the number of vertices of an orbit S' that can be reached from a subset X of the vertices of an orbit S within r time steps. Intuitively speaking, we show a lower bound on the number of vertices *reachable from orbit S in another orbit S'*. A simple consequence of the following lemma is that, from any start vertex in the temporal graph, at least one vertex in every orbit is reachable within r time steps.

▶ **Lemma 6** (Reachability between Orbits). *Let $\mathcal{G}$ be a connected temporal graph with lifetime ℓ and $S \in \mathcal{G}/Aut(\mathcal{G})$. For any $X \subseteq S$ and $S' \in \mathcal{G}/Aut(\mathcal{G})$ it holds that $|L_{t,t'}(X) \cap S'| \geq \lceil |X| \cdot |S'|/|S| \rceil$ for any $t \in [\ell]$ and $t' = t + r$, where $r = |\mathcal{G}/Aut(\mathcal{G})|$ is the orbit number.*

By using Lemma 1 and Lemma 6, we now bound the number of time steps needed to reach a set of h vertices within an orbit S.

▶ **Lemma 7.** *Let $\mathcal{G}$ be a connected temporal graph with lifetime ℓ and vertex set V. Let $S \in \mathcal{G}/Aut(\mathcal{G})$ and let $r = |\mathcal{G}/Aut(\mathcal{G})|$ be the orbit number. For any $h \leq |S|$, start vertex $u \in S$ and start time t, there exists a set $X \subset S$ with $|X| = h$ such that we can reach any vertex in X in at most $O(\min\{h \cdot n/|S|, hr\} + r)$ time steps. That is, for every vertex u' of X, we have a temporal walk starting at u at time step t and ending at u' at time step t' with $t' - t = O(\min\{h \cdot n/|S|, hr\} + r)$.*

Next we present Theorem 8, which states an upper bound for visiting all vertices of a given orbit S. The rough idea used in the proof is that we iteratively build the final temporal walk W by concatenating smaller temporal walks. In each step of the iteration, a small temporal walk W' is first constructed via Lemma 7 to visit a subset of the vertices of S, which are not necessarily unvisited, but such that the size of the subset is at least a certain threshold value. Using Lemma 3 we find an automorphism σ that we apply to W' to obtain a temporal walk W_σ that visits many unvisited vertices of S. We then extend W via this transformed walk W_σ (see Fig. 2 for a sketch of the proof idea). In this way we can explore all vertices of a large orbit S faster than by repeated application of Lemma 1. The key to obtain a good bound on the number of time steps required is to find a good value for the number of vertices of S visited by each small temporal walk.

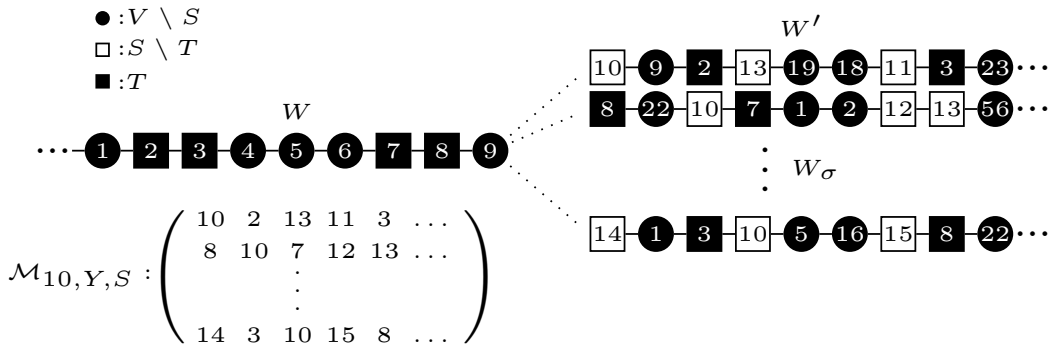


Figure 2 The construction scheme of Theorem 8. W is the temporal walk constructed so far. We aim to extend the walk with a walk W_σ that visits many vertices of $S \setminus T$, where T is the set of vertices of orbit S that we have already visited. To find W_σ, we construct W' and the automorphism matrix (bottom left) for the vertex with label 10 (the start vertex of W'), the set Y of vertices of S visited by W', and the entire orbit S as the set of possible start vertices of W_σ. One of the rows in the matrix then gives us an automorphism σ that, when applied to W', yields the desired walk W_σ.

▶ **Theorem 8.** *Let $\mathcal{G}$ be a temporal graph with lifetime ℓ and vertex set V. Take $S \in \mathcal{G}/Aut(\mathcal{G})$ and $r = |\mathcal{G}/Aut(\mathcal{G})|$ the orbit number. For any $t \in [\ell]$ there exists a temporal walk W starting at time step t that visits all vertices of S and ends at time step t' with $t' - t = O((n^{5/3} + rn) \log n)$.*

The following lemma is concerned with visiting a fraction $1/c$ of the vertices of a given orbit S with a temporal walk. One significant contribution to the number of time steps required by the temporal walk constructed in Theorem 8 is the use of Lemma 3. Roughly speaking, Lemma 3 provides a temporal walk that visits a large number of unvisited vertices, but with the caveat that every vertex of S can potentially be the start vertex of this transformed walk (instead of restricting the potential start vertices for the walk to a smaller subset, which might be reachable more quickly). The consequence of this is that for each such transformation we require, we must plan a "buffer" of n time steps to ensure that all vertices of S are reachable by the time step in which the transformed walk starts (Lemma 1). Corollary 4 provides a "trade-off" for this: a decrease in the set of possible start vertices of the transformed walk W_σ decreases the number of previously unvisited vertices W_σ visits, but also decreases the number of time steps required to reach the first vertex of W_σ. Using this property we construct a recursive algorithm that visits a fraction of the vertices of S quickly instead of applying the iterative construction of Theorem 8. In our recursive construction, the walks we concatenate shrink with each recursive call. If we were to use Lemma 3 during this, we would have an additional n time steps with each recursive call. Instead, Corollary 4 lets us reduce the number of possible start vertices dramatically. The time span required by this walk is then not dependent on n, but dependent on $|S|$ and r (the orbit number), and thus is especially useful for exploring smaller orbits. This can then be used iteratively to construct a temporal walk that visits all vertices of S, which we in turn use to visit all vertices V by visiting all orbits one after the other.

▶ **Lemma 9.** *Let $\mathcal{G}$ be a connected temporal graph with lifetime ℓ and vertex set V. Let $S \in \mathcal{G}/Aut(\mathcal{G})$ and let $r = |\mathcal{G}/Aut(\mathcal{G})|$ be the orbit number. For any $t \in [\ell]$ and any $u \in S$ there exists a temporal walk W that starts at vertex u in time step t and visits a fraction $1/c$ (for any $1 < c < |S|$) of the vertices of S such that W spans $O(rc(|S|/c)^{\phi(c)} \log |S|)$ time steps, with $\phi(c) = 1/(\log f(c))$ and $f(c) = (1 + (c-1)/c)$.*

▶ **Corollary 10.** *Let $\mathcal{G}$ be a temporal graph with lifetime ℓ and vertex set V. Let $S \in \mathcal{G}/Aut(\mathcal{G})$ and $r = |\mathcal{G}/Aut(\mathcal{G})|$ be the orbit number. For any $t \in [\ell]$, any $u \in S$, and any fixed $\epsilon > 0$, there exists a temporal walk W that starts at vertex u in time step t and visits some constant fraction $\alpha < 1$ of the vertices of S such that W spans $O(r|S|^{1+\epsilon})$ time steps.*

We now present an improved version of Theorem 8 for exploring a whole orbit.

▶ **Theorem 11.** *Let $\mathcal{G}$ be a temporal graph with lifetime ℓ and vertex set V. Let $S \in \mathcal{G}/Aut(\mathcal{G})$ and $r = |\mathcal{G}/Aut(\mathcal{G})|$ be the orbit number. For any $t \in [\ell]$, any $u \in V$, and any fixed $\epsilon > 0$, there exists a temporal walk W that starts at vertex u in time step t and visits all vertices of S such that W spans $O(r|S|^{1+\epsilon} + n \log |S|)$ time steps.*

By using the theorem above repeatedly for each orbit, we get a temporal walk for the whole temporal graph.

▶ **Corollary 12.** *Let $\mathcal{G}$ be a temporal graph with lifetime ℓ and vertex set V. For any fixed $\epsilon > 0$, there exists a temporal walk W that spans $O(rn^{1+\epsilon})$ time steps and visits all vertices of V, where $r = |\mathcal{G}/Aut(\mathcal{G})|$ is the orbit number.*

5 Upper Bound for TRP

Using Theorem 11 we show that TRP can be solved by constructing a walk that spans (asymptotically) the same number of steps as a walk for exploring an arbitrary single orbit. The idea is that the two agents identify an orbit in which they meet, and then the first agent moves to this orbit, and after n time steps the second agent starts exploring this orbit. For this the agents must be able to independently identify the same orbit, for which we introduce some additional notation. We extend the definition of isomorphism to temporal graphs as follows. Let $\mathcal{G}, \mathcal{H}$ be two temporal graphs with lifetime ℓ and vertex sets $V_\mathcal{G}$ and $V_\mathcal{H}$, respectively. We call a bijection $\theta : V_\mathcal{G} \to V_\mathcal{H}$ a *temporal isomorphism* if θ is an isomorphism from G_t to H_t for each $t \in [\ell]$ (and thus also an isomorphism from G to H, which denote the underlying graphs of $\mathcal{G}$ and $\mathcal{H}$, respectively). If clear from the context, we say *isomorphism* instead of *temporal isomorphism*.

We define an *integer coloring* as a coloring of the vertices in the vertex set V of a temporal graph $\mathcal{G}$ (with the colors being integer values). The assigned colors induce a partial order $\prec_\mathcal{P}$ on the vertex set V such that, for all vertices $u, v \in V$ that are assigned colors c_u and c_v, respectively, with $c_u \neq c_v$ it holds that $u \prec_\mathcal{P} v$ if $c_u < c_v$. The idea is now that the two agents compute the same integer coloring of the given temporal graph $\mathcal{G}$ with the property that two vertices $u, v \in V$ are assigned the same color c if and only if $u, v \in S$, with S some orbit of $\mathcal{G}/\mathrm{Aut}(\mathcal{G})$. The agents then meet at the smallest orbit, breaking ties via the coloring.

Note that, since the agents do not have access to consistent labels of the vertices in V, they are unable to distinguish between two vertices $u, v \in S$ with S being an orbit. Intuitively, the two agents a_1 and a_2 view $\mathcal{G}$ as different temporal graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively, such that $\mathcal{G}_1 \cong \mathcal{G} \cong \mathcal{G}_2$. A natural idea is for the agents to pick a smallest orbit for their meeting, but the challenge is how to ensure that the agents pick the same orbit if there are multiple equal-size orbits that all have the smallest size. Therefore, in the proof of the following lemma we let the agents iterate over all possible temporal graphs until they find a graph $\mathcal{H}$ with $\mathcal{G}_1 \cong \mathcal{H} \cong \mathcal{G}_2$. Then both agents compute an integer coloring for $\mathcal{H}$ as outlined in the previous paragraph. This coloring is translated to a coloring of $\mathcal{G}_1$ by agent a_1 and to a coloring of $\mathcal{G}_2$ by agent a_2 via isomorphism functions, which are independently computed by the agents.

▶ **Lemma 13.** *Let $\mathcal{G}$ be a temporal graph with vertex set V and lifetime ℓ and let a_1, a_2 be two label-oblivious agents. There exists a pair of programs (p_1, p_2) assigned to a_1 and a_2, respectively, such that each agent computes the same integer coloring of V and such that two vertices $u, v \in V$ have the same color exactly if u, v are in the same orbit of $\mathcal{G}/Aut(\mathcal{G})$.*

We can now easily construct an algorithm for TRP. The agents simply meet in a smallest orbit, breaking ties via the integer coloring (Lemma 13). The first agent moves to said orbit, then the second agent searches the orbit for the first agent. Note that the smallest orbit has size at most n/r, where r is the orbit number. Thus, the bound on the number of time steps provided by Theorem 11 becomes $O(r(n/r)^{1+\epsilon} + n\log(n/r)) = O(n^{1+\epsilon})$, and we obtain the following upper bound for TRP.

▶ **Theorem 14.** *Let $\mathcal{G}$ be a temporal graph with lifetime ℓ and a_1, a_2 two label-oblivious agents. For any fixed $\epsilon > 0$, there exists a pair of programs (p_1, p_2) assigned to a_1, a_2, respectively, such that the two agents are guaranteed to meet after $O(n^{1+\epsilon})$ time steps.*

6 Lower Bounds for TEXP and TRP

We start this section with a simple lower bound for TEXP, which is a fairly straightforward adaptation of the known lower bound of $\Omega(n^2)$ time steps [27]. Following that, we give a lower bound of $\Omega(n \log n)$ time steps for TRP. For this we describe the construction of a temporal graph that is connected and has only a single orbit. We then show how an adversary can choose the starting positions of the two agents that want to meet in order to delay their meeting. Intuitively, the graph we create is a cycle that changes repeatedly after some number of steps. By our construction, the adversary can make sure that after every change of the graph, the two agents are placed far away from each other. In the end, we also show that the resulting lower bound for TRP yields a corresponding lower bound for TEXP.

▶ **Lemma 15.** *For any $1 \leq r \leq n$, there exist n-vertex instances of TEXP with orbit number r that require $\Omega(rn)$ time steps to be explored.*

▶ **Theorem 16.** *For any two agents a_1 and a_2 with arbitrary deterministic programs, there exist instances of TRP where the agents require $\Omega(n \log n)$ time steps to meet.*

▶ **Corollary 17.** *There exist connected temporal graphs $\mathcal{G}$ with vertex set V, lifetime ℓ and a single orbit such that all temporal walks W require $\Omega(n \log n)$ time steps to visit all vertices of V.*

The lower bounds of Theorem 16 and Corollary 17 can be adapted to temporal graphs with r orbits for any constant r as follows: Use the construction from the proof of Theorem 16, but instead of letting the graph G_t in each time step be a single cycle C_i, let G_t contain $2r-1$ copies of C_i, and for each vertex u of C_i connect all copies of u by a path P_u (starting with the vertex u in the first copy of C_i and ending with the vertex u in the $(2r-1)$-th copy of C_i). The resulting temporal graph has r orbits: The vertices of the "middle" copy of C_i form one orbit, and the vertices in the two copies of C_i that have distance k from the middle copy, for $1 \leq k \leq r-1$, also form an orbit. Let $n' = n/r$ denote the number of vertices in one copy of C_i. By the arguments in the proof of Theorem 16, it takes $\Omega(n' \log n')$ time steps for the two agents to reach a location in the same path P_u, and thus TRP requires $\Omega(n' \log n')$ time steps. As $n' = n/r$ and r is a constant, this gives a lower bound of $\Omega(n \log n)$ for TRP. The lower bound of $\Omega(n \log n)$ time steps for exploration of temporal graphs with orbit number r for any constant r then follows as in the proof of Corollary 17.

7 Conclusions & Future Work

In this work, we looked at temporal graphs where agents know the complete information of the temporal graph ahead of time. In this clairvoyant setting, we studied the temporal exploration problem (TEXP) and showed how to bound the exploration time of a temporal graph using the structural graph property of the number of orbits of the automorphism group of the temporal graph. Additionally, we formalized the problem of asymmetric rendezvous in this setting as the temporal rendezvous problem (TRP) and showed how to adapt our ideas for TEXP to solve TRP quickly. For both TEXP and TRP we provided lower bounds such that the gap between upper and lower bounds is $O(n^{\epsilon})$ for any fixed $\epsilon > 0$. There are several ways in which our work can be extended. One line of research for both problems is to reduce the gap between the lower and upper bounds by improving either of them. A second line of work is to study the symmetric variant of rendezvous in the given setting and see if something can be said about it. Another interesting situation to explore is when multiple agents are used to explore the temporal graph (and also if multiple agents need to perform temporal rendezvous) and how much faster solutions in these scenarios might be. Lastly, a possible avenue of research is to study the structural properties provided by automorphism groups and how they can be used to tackle other problems that concern temporal graphs.

References

1 Eric Aaron, Danny Krizanc, and Elliot Meyerson. Dmvp: foremost waypoint coverage of time-varying graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 29–41. Springer, 2014. doi:10.1007/978-3-319-12340-0_3.

2 Eric Aaron, Danny Krizanc, and Elliot Meyerson. Multi-robot foremost coverage of time-varying graphs. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 22–38. Springer, 2014. doi:10.1007/978-3-662-46018-4_2.

3 Duncan Adamson, Vladimir V. Gusev, Dmitriy Malyshev, and Viktor Zamaraev. Faster exploration of some temporal graphs. In *1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SAND.2022.5.

4 Eleni C. Akrida, Jurek Czyzowicz, Leszek Gąsieniec, Łukasz Kuszner, and Paul G. Spirakis. Temporal flows in temporal networks. *Journal of Computer and System Sciences*, 103:46–60, 2019. doi:10.1016/j.jcss.2019.02.003.

5 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Christoforos Raptopoulos. The temporal explorer who returns to the base. *Journal of Computer and System Sciences*, 120:179–193, 2021. doi:10.1016/j.jcss.2021.04.001.

6 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Temporal vertex cover with a sliding time window. *Journal of Computer and System Sciences*, 107:108–123, 2020. doi:10.1016/j.jcss.2019.08.002.

7 Steve Alpern. The rendezvous search problem. *SIAM Journal on Control and Optimization*, 33(3):673–683, 1995. doi:10.1137/S0363012993249195.

8 Steve Alpern. Rendezvous search: A personal perspective. *Operations Research*, 50(5):772–795, 2002. doi:10.1287/opre.50.5.772.363.

9 Steve Alpern and Shmuel Gal. *The theory of search games and rendezvous*, volume 55 of *International Series in Operations Research & Management Science*. Springer Science & Business Media, 2006. doi:10.1007/b100809.

10 Steven Alpern. Hide and seek games. In *Seminar, Institut für höhere Studien, Wien*, volume 26, 1976.

11 K. Balasubramanian. Symmetry groups of chemical graphs. *International Journal of Quantum Chemistry*, 21(2):411–418, 1982. doi:10.1002/qua.560210206.

12 Fabian Ball and Andreas Geyer-Schulz. How symmetric are real-world graphs? A large-scale study. *Symmetry*, 10(1), 2018. doi:10.3390/sym10010029.

13 Subhash Bhagat and Andrzej Pelc. Deterministic rendezvous in infinite trees. *CoRR*, abs/2203.05160, 2022. doi:10.48550/arXiv.2203.05160.

14 Hans L. Bodlaender and Tom C. van der Zanden. On exploring always-connected temporal graphs of small pathwidth. *Information Processing Letters*, 142:68–71, 2019. doi:10.1016/j.ipl.2018.10.016.

15 John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory with applications*, volume 290. Macmillan London, 1976.

16 Marjorie Bournat, Swan Dubois, and Franck Petit. Gracefully degrading gathering in dynamic rings. In *Stabilization, Safety, and Security of Distributed Systems: 20th International Symposium, SSS 2018, Tokyo, Japan, November 4–7, 2018, Proceedings 20*, pages 349–364. Springer, 2018. doi:10.1007/978-3-030-03232-6_23.

17 Benjamin Merlin Bumpus and Kitty Meeks. Edge exploration of temporal graphs. *Algorithmica*, 85(3):688–716, 2023. doi:10.1007/s00453-022-01018-7.

18 Arnaud Casteigts. Efficient generation of simple temporal graphs up to isomorphism. Github repository, 2020. URL: https://github.com/acasteigts/STGen.

19 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *Int. J. Parallel Emergent Distributed Syst.*, 27(5):387–408, 2012. doi:10.1080/17445760.2012.668546.

20 Jurek Czyzowicz, Adrian Kosowski, and Andrzej Pelc. How to meet when you forget: Log-space rendezvous in arbitrary graphs. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '10, pages 450–459, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1835698.1835801.

21 Shantanu Das, Giuseppe Di Luna, Linda Pagli, and Giuseppe Prencipe. Compacting and grouping mobile agents on dynamic rings. In *International Conference on Theory and Applications of Models of Computation*, pages 114–133. Springer, 2019. doi:10.1007/978-3-030-14812-6_8.

22 Giuseppe Antonio Di Luna. Mobile agents on dynamic graphs. *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, pages 549–584, 2019. doi:10.1007/978-3-030-11072-7_20.

23 Giuseppe Antonio Di Luna, Paola Flocchini, Linda Pagli, Giuseppe Prencipe, Nicola Santoro, and Giovanni Viglietta. Gathering in dynamic rings. *Theoretical Computer Science*, 811:79–98, 2020. doi:10.1016/j.tcs.2018.10.018.

24 Yoann Dieudonné, Andrzej Pelc, and Vincent Villain. How to meet asynchronously at polynomial cost. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, pages 92–99, 2013. doi:10.1137/130931990.

25 Konstantinos Dogeas, Thomas Erlebach, Frank Kammer, Johannes Meintrup, and William K. Moses Jr au2. Exploiting automorphisms of temporal graphs for fast exploration and rendezvous, 2023. arXiv:2312.07140.

26 Jessica Enright, Kitty Meeks, George B. Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *Journal of Computer and System Sciences*, 119:60–77, 2021. doi:10.1016/j.jcss.2021.01.007.

27 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. *J. Comput. Syst. Sci.*, 119:1–18, 2021. doi:10.1016/j.jcss.2021.01.005.

28 Thomas Erlebach, Frank Kammer, Kelin Luo, Andrej Sajenko, and Jakob T. Spooner. Two moves per time step make a difference. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.141.

29 Thomas Erlebach and Jakob T. Spooner. A game of cops and robbers on graphs with periodic edge-connectivity. In *46th International Conference on Current Trends in Theory and Practice of Informatics (SOFSEM 2020)*, volume 12011 of *Lecture Notes in Computer Science*, pages 64–75. Springer, 2020. doi:10.1007/978-3-030-38919-2_6.

30 Thomas Erlebach and Jakob T. Spooner. Exploration of k-edge-deficient temporal graphs. *Acta Informatica*, 59(4):387–407, 2022. doi:10.1007/s00236-022-00421-5.

31 Thomas Erlebach and Jakob T. Spooner. Parameterized temporal exploration problems. In *1st Symposium on Algorithmic Foundations of Dynamic Networks, SAND 2022, March 28-30, 2022, Virtual Conference*, volume 221 of *LIPIcs*, pages 15:1–15:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SAND.2022.15.

32 Paola Flocchini. *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*. Springer, 2019. doi:10.1007/978-3-030-11072-7.

33 Till Fluschnik, Hendrik Molter, Rolf Niedermeier, Malte Renken, and Philipp Zschoche. As time goes by: Reflections on treewidth for temporal graphs. In *Treewidth, Kernels, and Algorithms: Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, pages 49–77. Springer International Publishing, 2020. doi:10.1007/978-3-030-42071-0_6.

34 Till Fluschnik, Hendrik Molter, Rolf Niedermeier, Malte Renken, and Philipp Zschoche. Temporal graph classes: A view through temporal separators. *Theoretical Computer Science*, 806:197–218, 2020. doi:10.1016/j.tcs.2019.03.031.

35 Chris Godsil and Gordon F. Royle. *Algebraic Graph Theory*. Number Book 207 in Graduate Texts in Mathematics. Springer, 2001. doi:10.1007/978-1-4613-0163-9.

36 Thekla Hamm, Nina Klobas, George B. Mertzios, and Paul G. Spirakis. The complexity of temporal vertex cover in small-degree graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9):10193–10201, June 2022. doi:10.1609/aaai.v36i9.21259.

37 David Ilcinkas, Ralf Klasing, and Ahmed Mouhamadou Wade. Exploration of constantly connected dynamic graphs based on cactuses. In *International Colloquium on Structural Information and Communication Complexity*, pages 250–262. Springer, 2014. doi:10.1007/978-3-319-09620-9_20.

38 David Ilcinkas and Ahmed Mouhamadou Wade. Exploration of the t-interval-connected dynamic graphs: the case of the ring. In *International Colloquium on Structural Information and Communication Complexity*, pages 13–23. Springer, 2013. doi:10.1007/978-3-319-03578-9_2.

39 Ulrich Knauer and Kolja Knauer. *Algebraic graph theory: morphisms, monoids and matrices*, volume 41. Walter de Gruyter GmbH & Co KG, 2019. doi:10.1515/9783110617368.

40 Evangelos Kranakis, Danny Krizanc, and Euripides Marcou. *The mobile agent rendezvous problem in the ring*. Springer Nature, 2022. doi:10.1007/978-3-031-01999-9.

41 Evangelos Kranakis, Danny Krizanc, and Sergio Rajsbaum. Mobile agent rendezvous: A survey. In *Structural Information and Communication Complexity*, pages 1–9. Springer Berlin Heidelberg, 2006. doi:10.1007/11780823_1.

42 Josef Lauri and Raffaele Scapellato. *Topics in Graph Automorphisms and Reconstruction*. Cambridge University Press, 2016. doi:10.1017/CBO9781316669846.

43 Gianluca De Marco, Luisa Gargano, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, and Ugo Vaccaro. Asynchronous deterministic rendezvous in graphs. *Theoretical Computer Science*, 355(3):315–326, 2006. doi:10.1016/j.tcs.2005.12.016.

44 Andrea Marino and Ana Silva. Königsberg sightseeing: Eulerian walks in temporal graphs. In *Combinatorial Algorithms*, pages 485–500. Springer International Publishing, 2021. doi:10.1007/978-3-030-79987-8_34.

45 Andrea Marino and Ana Silva. Coloring temporal graphs. *Journal of Computer and System Sciences*, 123:171–185, 2022. doi:10.1016/j.jcss.2021.08.004.

46 George B. Mertzios, Hendrik Molter, Rolf Niedermeier, Viktor Zamaraev, and Philipp Zschoche. Computing Maximum Matchings in Temporal Graphs. In *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*, volume 154 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2020.27.

47 George B. Mertzios, Hendrik Molter, and Viktor Zamaraev. Sliding window temporal graph coloring. *Journal of Computer and System Sciences*, 120:97–115, 2021. doi:10.1016/j.jcss.2021.03.005.

48 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Math.*, 12(4):239–280, 2016. doi:10.1080/15427951.2016.1177801.

49 Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Computer Science (MFCS)*, 2014.

50 Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Theoretical Computer Science*, 634:1–23, 2016. doi:10.1016/j.tcs.2016.04.006.

51 Othon Michail, Paul G. Spirakis, and Michail Theofilatos. Beyond rings: Gathering in 1-interval connected graphs. *Parallel Processing Letters*, 31(04):2150020, 2021. doi:10.1142/S0129626421500201.

52 Nils Morawietz, Carolin Rehs, and Mathias Weller. A Timecop's Work Is Harder Than You Think. In *45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*, volume 170 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 71:1–71:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2020.71.

53 Fukuhito Ooshita and Ajoy K. Datta. Brief announcement: feasibility of weak gathering in connected-over-time dynamic rings. In *Stabilization, Safety, and Security of Distributed Systems: 20th International Symposium, SSS 2018, Tokyo, Japan, November 4–7, 2018, Proceedings 20*, pages 393–397. Springer, 2018. doi:10.1007/978-3-030-03232-6_27.

54 Andrzej Pelc. Deterministic rendezvous in networks: A comprehensive survey. *Networks*, 59(3):331–347, 2012. doi:10.1002/net.21453.

55 Andrzej Pelc. Deterministic rendezvous algorithms. In *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, pages 423–454. Springer, 2019. doi:10.1007/978-3-030-11072-7_17.

56 Andrzej Pelc. Deterministic rendezvous algorithms. *CoRR*, abs/2303.10391, 2023. doi:10.48550/arXiv.2303.10391.

57 Philipp Plamper, Oliver J. Lechtenfeld, Peter Herzsprung, and Anika Groß. A temporal graph model to predict chemical transformations in complex dissolved organic matter. *Environmental Science & Technology*, 2023. doi:10.1021/acs.est.3c00351.

58 Nicola Santoro. Time to change: On distributed computing in dynamic networks (keynote). In *19th International Conference on Principles of Distributed Systems, OPODIS 2015, December 14-17, 2015, Rennes, France*, volume 46 of *LIPIcs*, pages 3:1–3:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.OPODIS.2015.3.

59 C. Shannon. Presentation of a maze solving machine. In *Trans. 8th Conf. Cybernetics: Circular, Causal and Feedback Mechanisms in Biological and Social Systems (New York, 1951)*, pages 169–181, 1951.

60 Masahiro Shibata, Naoki Kitamura, Ryota Eguchi, Yuichi Sudo, Junya Nakamura, and Yonghwan Kim. Partial gathering of mobile agents in dynamic tori. In *2nd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2023)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.SAND.2023.2.

61 Masahiro Shibata, Yuichi Sudo, Junya Nakamura, and Yonghwan Kim. Partial gathering of mobile agents in dynamic rings. In *Stabilization, Safety, and Security of Distributed Systems: 23rd International Symposium, SSS 2021, Virtual Event, November 17–20, 2021, Proceedings 23*, pages 440–455. Springer, 2021. doi:10.1007/978-3-030-91081-5_29.

62 Shadi Taghian Alamouti. *Exploring temporal cycles and grids*. PhD thesis, Concordia University, 2020.

63 Philipp Zschoche, Till Fluschnik, Hendrik Molter, and Rolf Niedermeier. The complexity of finding small separators in temporal graphs. *Journal of Computer and System Sciences*, 107:72–92, 2020. doi:10.1016/j.jcss.2019.07.006.

Lower Bounds for Matroid Optimization Problems with a Linear Constraint

Ilan Doron-Arad
Computer Science Department, Technion, Haifa, Israel

Ariel Kulik
Computer Science Department, Technion, Haifa, Israel

Hadas Shachnai
Computer Science Department, Technion, Haifa, Israel

Abstract

We study a family of *matroid optimization problems with a linear constraint* (MOL). In these problems, we seek a subset of elements which optimizes (i.e., maximizes or minimizes) a linear objective function subject to (i) a matroid independent set, or a matroid basis constraint, (ii) additional linear constraint. A notable member in this family is BUDGETED MATROID INDEPENDENT SET (BM), which can be viewed as classic 0/1-KNAPSACK with a matroid constraint. While special cases of BM, such as KNAPSACK WITH CARDINALITY CONSTRAINT and MULTIPLE-CHOICE KNAPSACK, admit a *fully polynomial-time approximation scheme* (Fully PTAS), the best known result for BM on a general matroid is an *Efficient* PTAS. Prior to this work, the existence of a Fully PTAS for BM, and more generally, for any problem in the family of MOL problems, has been open.

In this paper, we answer this question negatively by showing that none of the (non-trivial) problems in this family admits a Fully PTAS. This resolves the complexity status of several well studied problems. Our main result is obtained by showing first that EXACT WEIGHT MATROID BASIS (EMB) does not admit a pseudo-polynomial time algorithm. This distinguishes EMB from the special cases of k-SUBSET SUM and EMB on a linear matroid, which are solvable in pseudo-polynomial time. We then obtain unconditional hardness results for the family of MOL problems in the oracle model (even if randomization is allowed), and show that the same results hold when the matroids are encoded as part of the input, assuming P $\neq$ NP. For the hardness proof of EMB, we introduce the Π-*matroid* family. This intricate subclass of matroids, which exploits the interaction between a weight function and the matroid constraint, may find use in tackling other matroid optimization problems.

2012 ACM Subject Classification Theory of computation

Keywords and phrases matroid optimization, budgeted problems, knapsack, approximation schemes

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.56

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2307.07773 [20]

Funding *Ariel Kulik*: This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 852780-ERC (SUBMODULAR).

1 Introduction

Matroids are simple combinatorial structures, providing a unified abstraction for independence systems such as linear independence in a vector space, or cycle-free subsets of edges in a given graph. A *matroid* is a set system $(E, \mathcal{I})$, where E is a finite set and $\mathcal{I} \subseteq 2^E$ are the *independent sets (IS)* such that (i) $\emptyset \in \mathcal{I}$, (ii) for all $A \in \mathcal{I}$ and $B \subseteq A$ it holds that $B \in \mathcal{I}$, and (iii) for all $A, B \in \mathcal{I}$ where $|A| > |B|$, there is $e \in A \setminus B$ such that $B \cup \{e\} \in \mathcal{I}$.[1]

[1] Properties (ii) and (iii) are known, respectively, as *hereditary property*, and *exchange property*.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 56; pp. 56:1–56:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

While serving as a generic abstraction for numerous applications, matroids possess useful combinatorial properties that allow the development of efficient algorithms. These algorithms include such canonical results as the classic greedy approach for finding a maximum weight independent set of a matroid (see, e.g., [13]), Edmond's algorithm for matroid partitioning [21], and Lawler's algorithm for matroid intersection [31]. In all of the above, polynomial running time is enabled due to the structure of the problem − a single objective function with a matroid constraint. However, in many natural applications, there is an added *linear* constraint.

Consider, for example, the problem of finding a maximum independent set in a matroid subject to a budget constraint. Formally, we are given a set of elements E, a membership oracle for a collection of independent sets $\mathcal{I} \subseteq 2^E$ of a matroid $(E, \mathcal{I})$, a budget $L > 0$, a weight function $w : E \to \mathbb{R}_{\geq 0}$, and a value function $v : E \to \mathbb{R}_{\geq 0}$. A *solution* for the problem is an independent set $S \in \mathcal{I}$ of total weight at most L, i.e., $w(S) \leq L$.[2] The *value* of a solution S is given by $v(S)$, and the objective is to find a solution of maximum value. This problem, known as BUDGETED MATROID INDEPENDENT SET (BM), is a generalization of the classic 0/1-KNAPSACK, which is NP-hard and therefore unlikely to admit an exact polynomial-time algorithm. Thus, obtaining efficient approximations has been a main focus in the study of BM.

For an instance I of an optimization problem $\mathcal{G}$, let $\mathrm{OPT}_{\mathcal{G}}(I)$ be the value of an optimal solution for I. For some $\rho \geq 1$, a *ρ-approximate solution* S for I is a solution of value $v \geq \frac{\mathrm{OPT}_{\mathcal{G}}(I)}{\rho}$ ($v \leq \rho \cdot \mathrm{OPT}_{\mathcal{G}}(I)$) if $\mathcal{G}$ is a maximization (minimization) problem. We say that $\mathcal{A}$ is a *randomized ρ-approximation algorithm* for $\mathcal{G}$ if given an instance I of $\mathcal{G}$ $\mathcal{A}$ returns with probability at least $\frac{1}{2}$ a ρ-approximate solution for I − if a solution exists. If no solution exists − $\mathcal{A}$ returns that I does not have a solution.

Let $|I|$ be the encoding size of an instance I of a problem $\mathcal{G}$. A (randomized) *polynomial-time approximation scheme* (PTAS) for $\mathcal{G}$ is a family of algorithms $(A_\varepsilon)_{\varepsilon>0}$ such that, for any $\varepsilon > 0$, A_ε is a (randomized) polynomial-time $(1+\varepsilon)$-approximation algorithm for $\mathcal{G}$. A (randomized) *Efficient PTAS* (EPTAS) is a (randomized) PTAS $(A_\varepsilon)_{\varepsilon>0}$ with running time of the form $f\left(\frac{1}{\varepsilon}\right) \cdot |I|^{O(1)}$, where f is an arbitrary computable function. The strong dependence of run-times on the error parameter, $\varepsilon > 0$, often renders the above schemes highly impractical. This led to the study of the following more desirable class of schemes. A (randomized) approximation scheme $(A_\varepsilon)_{\varepsilon>0}$ is a (randomized) *Fully PTAS* (FPTAS) if the running time of A_ε is of the form $\left(\frac{|I|}{\varepsilon}\right)^{O(1)}$.[3]

In the past decades, BM was shown to admit a PTAS [2, 12, 25], and more recently an Efficient PTAS [18, 17]. As the special case of 0/1-KNAPSACK admits a Fully PTAS, it is natural to explore the existence of a Fully PTAS for BM. There are known Fully PTASs for BM on restricted families of matroids. This includes KNAPSACK WITH A CARDINALITY CONSTRAINT [7], MULTIPLE-CHOICE KNAPSACK [32], and BM WITH LAMINAR MATROID CONSTRAINT [19]. However, the question whether BM admits a Fully PTAS on general matroids remained open.

In this paper, we resolve this question negatively for BM and other fundamental matroid optimization problems with a linear constraint.

---

[2] For every set X, a function $f : X \to \mathbb{R}_{\geq 0}$ and $Y \subseteq X$ we define $f(Y) = \sum_{e \in Y} f(e)$.

[3] To better distinguish between EPTAS and FPTAS, we use throughout the paper Efficient PTAS and Fully PTAS.

1.1 Our Results

For a matroid $\mathcal{M} = (E, \mathcal{I})$ we define $\text{IS}(\mathcal{M}) = \mathcal{I}$ and $\text{bases}(\mathcal{M}) = \{S \in \mathcal{I} \mid |S| = \mathsf{rank}(\mathcal{M})\}$, where $\mathsf{rank}(\mathcal{M}) = \max_{T \in \mathcal{I}} |T|$ is the *rank* of $\mathcal{M}$, i.e., the maximum cardinality of an independent set. We study a family of *matroid optimization problems with a linear constraint* (MOL). Problems in this family are characterized by three parameters:

(i) The optimization objective opt $-$ either the operator "max" or "min".

(ii) A *matroid feasibility constraint* $\mathcal{F}$ $-$ either the independent sets of a matroid, or the set of *bases* of a matroid. The feasibility constraint is $\mathcal{F} \in \{\text{IS}, \text{bases}\}$.

(iii) A relation $\triangleleft$ $-$ realized by one of the relations "$\geq$" or "$\leq$".

Let $\mathcal{P} = \{\max, \min\} \times \{\text{bases}, \text{IS}\} \times \{\leq, \geq\}$ be the set of *parameters* for MOL problems. Based on the set of parameters $\mathcal{P}$, we define for every triplet a problem in the MOL family. For $P \in \mathcal{P}$ where $P = (\text{opt}, \mathcal{F}, \triangleleft)$, define the P-MATROID OPTIMIZATION WITH A LINEAR CONSTRAINT (P-MOL) problem as follows. An instance is a tuple $I = (E, \mathcal{I}, v, w, L)$ such that $\mathcal{M} = (E, \mathcal{I})$ is a matroid, $v : E \to \mathbb{R}_{\geq 0}$ is the objective function, $w : E \to \mathbb{R}_{\geq 0}$ is a weight function, and $L \in \mathbb{R}_{\geq 0}$ is a *bound* for the linear constraint. A *solution* of I is $S \subseteq E$ which satisfies the matroid feasibility constraint $S \in \mathcal{F}(\mathcal{M})$ and the linear constraint $w(S) \triangleleft L$. The goal is to optimize (i.e., maximize or minimize) the value $v(S)$. Thus, we can formulate a P-MOL optimization problem as

$$\text{opt } v(S) \text{ s.t. } S \in \mathcal{F}(\mathcal{M}),\ w(S) \triangleleft L. \tag{1}$$

Observe that $(\max, \text{IS}, \leq)$-MOL is the BM problem. That is, given a BM instance (equivalently, $(\max, \text{IS}, \leq)$-MOL instance) $I = (E, \mathcal{I}, v, w, L)$, the goal is to find an independent set $S \in \mathcal{I}$ of maximum total value $v(S)$ such that $w(S) \leq L$. Other notable examples for MOL problems are CONSTRAINED MINIMUM BASIS OF A MATROID (CMB) [26], which can be cast as $(\min, \text{bases}, \leq)$-MOL, and KNAPSACK COVER WITH MATROID CONSTRAINT (KCM) [8] formalized by $(\min, \text{IS}, \geq)$-MOL.[4]

We note that (1) does not refer to the *representation* of the instance I. We consider two possible representations. For any $P \in \mathcal{P}$, in an instance $(E, \mathcal{I}, v, w, L)$ of *oracle* P-MOL, the arguments E, v, w, L are given as the input, and the independent sets $\mathcal{I}$ are accessed via a *membership oracle*, which determines whether a given set $S \subseteq E$ belongs to $\mathcal{I}$ in a single query. Thus, the independent sets are not considered in the encoding size of the instance. The term *running time* for problems involving oracles refers to the sum of the number of queries to the oracle and the number of basic operations. Previous works on MOL problems often consider membership oracles [12, 2, 8, 18, 17]. As hardness with oracles does not necessarily imply hardness in non-oracle models (see, e.g., [6, 9]), in Section 4 we show lower bounds for variants of MOL problems in which the independent sets are encoded as part of the input.

Clearly, the problem $(\min, \text{IS}, \leq)$-MOL is trivial since the empty set achieves the optimal objective value. However, for any other $P \in \mathcal{P}$, solving the P-MOL problem is challenging. The *non-trivial* MOL problems are all the MOL problems excluding $(\min, \text{IS}, \leq)$-MOL. That is, P-MOL is non-trivial if $P \in \mathcal{Q}$ where $\mathcal{Q} = \mathcal{P} \setminus \{(\min, \text{IS}, \leq)\}$. Observe that non-trivial MOL problems are NP-hard (e.g., 0/1-KNAPSACK is a special case of $(\max, \text{IS}, \leq)$-MOL); however, all previously studied MOL problems admit approximation schemes.

[4] CMB, KCM, and other MOL problems may not have a solution; however, we can decide in polynomial time if a solution exists, and our definition of approximation algorithms captures instances with no solution.

For certain special cases of MOL problems, e.g., BM with simple matroid constraints, the existence of a Fully PTAS is known for decades [38, 7]. However, for MOL problems with arbitrary matroid constraints, the best known results are Efficient PTAS. While matroids form an important generalization of well known basic constraints, the complexity of the corresponding MOL problems remained open. Specifically, prior to this work, the existence of a MOL problem which does not admit a Fully PTAS was open.

Our main result is that none of the (non-trivial) oracle matroid optimization problem with a linear constraint admits a Fully PTAS, even if randomization is allowed. This unconditioned hardness result is established by deriving a lower bound on the minimum number of queries to the membership oracle.

▶ **Theorem 1.** *For every $P \in \mathcal{Q}$ there is no randomized* Fully PTAS *for oracle P-*MOL.

Table 1 Implications of our results for previously studied MOL problems. All of our bounds hold for randomized algorithms.

Problem	Previous Results	This Paper
Budgeted Matroid Independent Set	Efficient PTAS [18]	**No Fully PTAS**
Budgeted Matroid Intersection	Efficient PTAS [17]	**No Fully PTAS**
Constrained Minimum Basis of Matroid	Efficient PTAS [26]	**No Fully PTAS**
Knapsack Cover with a Matroid	PTAS [8]	**No Fully PTAS**

Theorem 1 conclusively distinguishes MOL problems with arbitrary matroids, such as BM, from special cases with simpler matroid constraints. Furthermore, it shows that existing Efficient PTAS [26, 18, 17] for MOL problems on general matroids are the best possible. Notable implications of our results are given in Table 1, and consequences of our lower bounds for a set of previously studied problems [5, 26, 2, 12, 25, 8, 18, 17, 16] are given in Section 1.3. By resolving the complexity status of MOL problems on general matroids, our results promote future research to design (or rule out) Fully PTAS for MOL problems on restricted matroid classes (see Section 5).

To prove Theorem 1, we turn our attention to the following problem.

▶ **Definition 2.** *An instance of* Exact Matroid Basis (EMB) *is $I = (E, \mathcal{I}, c, T)$, where $(E, \mathcal{I})$ is a matroid, $c : E \to \mathbb{N}$ is a weight function, and $T \in \mathbb{N}$ is a target value. A* solution *is a basis S of $(E, \mathcal{I})$ such that $c(S) = T$. The goal is to decide if there is a solution.*

Similar to MOL problems, EMB does not specify the input. In an instance $(E, \mathcal{I}, c, T)$ of oracle-EMB, E, c, T are explicitly given, and the independent sets $\mathcal{I}$ are accessed via a membership oracle. An instance I of a decision problem $\mathcal{D}$ is a "yes"-instance if the correct answer for I is "yes"; otherwise, I is a "no"-instance. We say that $\mathcal{A}$ is a *randomized algorithm* for a decision problem $\mathcal{D}$ if, given a "yes"-instance I of $\mathcal{D}$, $\mathcal{A}$ returns "yes" with probability at least $\frac{1}{2}$; for a "no"-instance, $\mathcal{A}$ returns "no" with probability 1. The next result rules out a pseudo-polynomial time algorithm for oracle-EMB, thus distinguishing the problem from the special cases of k-subset sum and EMB on linear matroids, which admit a pseudo-polynomial time algorithm [5].

▶ **Theorem 3.** *For any* oracle-EMB *instance $I = (E, \mathcal{I}, c, T)$, there is no randomized algorithm for* oracle-EMB *that runs in time $(n \cdot (T+2) \cdot m)^{O(1)}$, where $n = |E| + 1$ and $m = c(E) + 1$.*

1.2 Technical Contribution

We derive our results by introducing Π-*matroids*. This new family of *paving* matroids carefully exploits a simple weight function to define a matroid that successfully *hides* a specific property Π within its independent sets (see Section 2).[5] Using Π-matroids, we define oracle-EMB instances whose solutions must satisfy the property Π. This shows the unconditional hardness of oracle-EMB, as Π can be discovered only via an exponential number of queries to the membership oracle. Our hardness results for MOL problems (as stated in Theorem 1) are derived via reduction from oracle-EMB.

Despite the abundance of lower bounds for matroid problems [33, 28, 29, 39], as well as for knapsack problems [10, 30, 14, 3], we are not aware of lower bounds that leverage the interaction between the matroid constraint and the additional linear constraint required for deriving our new lower bound for EMB, and consequently for MOL problems. Indeed, if the matroid constraint is removed from (1) (equivalently, $\mathcal{F}(\mathcal{M}) = 2^E$), MOL problems become variants of classic 0/1-KNAPSACK, which admits a Fully PTAS. Alternatively, if the linear constraint imposed by w, L is removed, then we have the polynomially solvable maximum/minimum weight matroid independent set problem. This distinguishes our construction from existing lower bounds for matroid problems, as even previous constructions of *paving* matroids (e.g., [28]) cannot be easily adapted to tackle both the knapsack constraint along with the matroid constraint. Π-matroids may be useful for deriving lower bounds for other problems (see Section 5).

Our unconditional lower bounds apply in the *oracle* model, where the independent sets of the given matroid can be accessed only via a membership oracle. One may question the validity of the bounds for variants of the problems where the matroid is encoded as part of the input. Indeed, in some scenarios, the use of oracles makes problems harder [6, 9]. Thus, we complement our results by showing that the same lower bounds hold under the standard complexity assumption P $\neq$ NP, even if the matroid is encoded as part of the instance and membership can be decided in polynomial time. We accomplish this by designing the family of SAT-*matroids* – a counterpart of the Π-matroid family whose members can be efficiently encoded. This construction can be used to obtain hardness results for other matroid problems in non-oracle models, based on existing analogous lower bounds in the oracle model (e.g., [28]). We elaborate on that in Section 4.

1.3 Implications of Our Results and Prior Work

Below we describe in further detail the implications of our results, and discuss previous work on MOL problems. In the following problems, general matroids are assumed to be accessed via membership oracles.

Exact Matroid Basis (EMB). This is a generalization of the k-SUBSET SUM problem (where $(E, \mathcal{I})$ is a uniform matroid).[6] Thus, EMB is unlikely to be solvable in polynomial time. Instead, we seek a pseudo-polynomial time algorithm whose running time has polynomial dependence on the encoding size of the instance and the target value T. Indeed, the special case of EMB in which the matroid is *representable* (or, linear) admits such a pseudo-polynomial time algorithm [5]. Since the 1990s, it has been an open question whether the result of Camerini et al. [5] can be extended to general matroids. Theorem 3 resolves this question, ruling out the existence of a pseudo-polynomial time algorithm for EMB.

[5] We note that paving matroids have been used in earlier work, e.g., to show intractability of the matroid matching problem in the oracle model [33, 28].
[6] In a uniform matroid, $\mathcal{I} = \{S \subseteq E \mid |S| \leq k\}$.

Budgeted Matroid Independent Set (BM). This problem is cast as $(\max, \mathrm{IS}, \leq)$-MOL. BM is a natural generalization of the classic 0/1-KNAPSACK problem, for which a Fully PTAS has been known since the 1970s [32]. As mentioned above, a Fully PTAS is known also for other special cases of BM. A PTAS for BM was first given in [2] as a special case of BUDGETED MATROID INTERSECTION (BMI). In this generalization of BM, we are given *two* matroids $(E, \mathcal{I}_1)$ and $(E, \mathcal{I}_2)$, and a solution has to be an independent set of *both* matroids. A PTAS for BM also follows from the results of [25, 12] which present PTASs for multi-budgeted variants of BM. An Efficient PTAS for BM was recently given in [18] and for BMI in [17]. The existence of a Fully PTAS for BM was posed as a central open question in [2, 18, 17]. We answer this question negatively, as formalized in Theorem 1, giving a tight lower bound for BM and BMI.

Constrained Minimum Basis of a Matroid (CMB). This problem can be cast as the matroid optimization problem $(\min, \mathrm{bases}, \leq)$-MOL. The CONSTRAINED MINIMUM SPANNING TREE (CST) problem is the special case of CMB in which the matroid $(E, \mathcal{I})$ is graphical [36, 1, 26, 27], namely, there is a graph $G = (V, E)$ such that the independent sets $\mathcal{I}$ are cycle-free subsets of edges in G. A PTAS for CST was given by Ravi and Goemans [36]. This result was improved to an Efficient PTAS by Hassin and Levin [26]. A *bicriteria* FPTAS, which violates the budget constraint by a factor of $(1+\varepsilon)$, was presented in [27]. The authors of [26] mention that their result actually gives an Efficient PTAS for CMB. The existence of a Fully PTAS for CMB remained an open question. Theorem 1 shows that the Efficient PTAS for CMB cannot be improved.

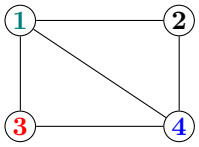
Knapsack Cover with a Matroid (KCM). As a final implication, Theorem 1 rules out the existence of a Fully PTAS for a *coverage* variant of 0/1-KNAPSACK, formulated as $(\min, \mathrm{IS}, \geq)$-MOL. In [8], Chakaravarthy et al. presented a PTAS for KCM using integrality properties of a linear programming formulation of KCM. Moreover, for the special case of KCM with a *partition matroid*, they give a Fully PTAS based on dynamic programming. The existence of a Fully PTAS for KCM on a general matroid was posed in [8] as an open question. Theorem 1 answers this question negatively. Our initial study indicates that an Efficient PTAS for KCM can potentially be obtained by adapting the approach of Hassin and Levin [26] to the setting of KCM. This suggests that our lower bound cannot be strengthened.

1.4 Organization

In Section 2 we introduce the Π-matroid family and give the proof of Theorem 3. In Section 3 we prove Theorem 1, and in Section 4 we show that similar lower bounds hold in the standard computational model. We conclude in Section 5 with a summary and directions for future work. Due to space constraints, some of the proofs are given in the full version of the paper [20].

2 The Hardness of ORACLE EXACT MATROID BASIS

In this section, we prove Theorem 3. We use in the proof the family of Π-*matroids*. For any $m \in \mathbb{N}$, let $[m] = \{1, \ldots, m\}$. A member in the Π-matroid family is given by four arguments: $n, k, \alpha \in \mathbb{N}_{>0}$, and $\Pi \subseteq 2^{[n]}$. The first argument, $n \in \mathbb{N}_{>0}$, is the number of elements, and the ground set is $[n]$. The second argument, $k \in [n]$, is the rank of the matroid. The third argument, $\alpha \in \mathbb{N}_{>0}$, is a target value, that is usually equal to $\mathsf{sum}(S)$ for some $S \subseteq [n]$, where $\mathsf{sum}(S) = \sum_{i \in S} i$. The last argument is a family of subsets $\Pi \subseteq 2^{[n]}$.



$\mathcal{J}_{4,2} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}\}$, $\mathcal{K}_{4,2,5} = \{\{2,4\}, \{2,1\}, \{4,3\}, \{3,1\}\}$, $\mathcal{L}_{4,2,5}(\Pi) = \{\{2,3\}\}$

Figure 1 The independent sets of the Π-matroid $M_{n,k,\alpha}(\Pi)$, with parameters $n = 4$, $k = 2$, and $\alpha = 5$. The secret family Π contains all independent sets in the graph G, where $\{2, 3\}$ is the only independent set in G with k elements.

The set Π is called the *secret family* because finding $S \in \Pi$ is possible only via repeated queries to the membership oracle of the matroid. Since Π can have an arbitrary structure, this may require exhaustive enumeration.

▶ **Definition 4.** *Let $n, k, \alpha \in \mathbb{N}_{>0}$. For some $\Pi \subseteq 2^{[n]}$, define the* **Π-matroid** *on n, k, and α as $M_{n,k,\alpha}(\Pi) = ([n], \mathcal{I}_{n,k,\alpha}(\Pi))$, where*

$$\mathcal{I}_{n,k,\alpha}(\Pi) = \mathcal{J}_{n,k} \cup \mathcal{K}_{n,k,\alpha} \cup \mathcal{L}_{n,k,\alpha}(\Pi)$$

and $\mathcal{J}_{n,k}, \mathcal{K}_{n,k,\alpha}, \mathcal{L}_{n,k,\alpha}(\Pi)$ are defined as follows.

$$\begin{aligned} \mathcal{J}_{n,k} &= \{S \subseteq [n] \mid |S| < k\} \\ \mathcal{K}_{n,k,\alpha} &= \{S \subseteq [n] \mid |S| = k, \mathsf{sum}(S) \neq \alpha\} \\ \mathcal{L}_{n,k,\alpha}(\Pi) &= \{S \subseteq [n] \mid |S| = k, \mathsf{sum}(S) = \alpha, S \in \Pi\}. \end{aligned} \tag{2}$$

In words, $\mathcal{J}_{n,k}$ contains all subsets of strictly less than k elements; $\mathcal{K}_{n,k,\alpha}$ contains all subsets of cardinality k whose total sum is not α. Finally, $\mathcal{L}_{n,k,\alpha}(\Pi)$ contains all subsets of cardinality k and total sum α which also belong to Π. See Figure 1 for an example of a member of the Π-matroid family. Using a simple argument, we show that the set system in Definition 4 is indeed a matroid. For the sets $\mathcal{I}_{n,k,\alpha}(\Pi)$, $\mathcal{J}_{n,k}$, $\mathcal{K}_{n,k,\alpha}$ and $\mathcal{L}_{n,k,\alpha}(\Pi)$ defined in Definition 4, we often omit the subscripts n, k, α and n, k when the values of n, k, α are known by context. For simplicity, for any set A and an element a, let $A + a$, $A - a$ be $A \cup \{a\}$ and $A \setminus \{a\}$, respectively.

▶ **Lemma 5.** *For every $n, k, \alpha \in \mathbb{N}_{>0}$, and $\Pi \subseteq 2^{[n]}$ it holds that $M_{n,k,\alpha}(\Pi)$ is a matroid.*[7]

Proof. We first note that $\emptyset \in \mathcal{J}$ since $0 < k$; therefore, $\emptyset \in \mathcal{I}(\Pi)$. For the hereditary property, let $A \in \mathcal{I}(\Pi)$. For all $B \subset A$ it holds that $|B| < k$; thus, $B \in \mathcal{J}$ and it follows that $B \in \mathcal{I}(\Pi)$. For the exchange property, let $A, B \in \mathcal{I}(\Pi)$ such that $|A| > |B|$. We consider the following cases.

1. $|B| < k - 1$. Then, for all $e \in A \setminus B$ it holds that $|B + e| < k - 1 + 1 = k$. Hence, $B + e \in \mathcal{J}$ and it follows that $B + e \in \mathcal{I}(\Pi)$. Note that there is such $e \in A \setminus B$ because $|A| > |B|$.
2. $|B| = k - 1$ and $|A| = k$. We consider two subcases.
   a. $B \subseteq A$. Then, as $|A| > |B|$ there is $e \in A \setminus B$. Hence, $B + e = A$ (because $|B| = k - 1$ and $|A| = k$). As $A \in \mathcal{I}(\Pi)$, it follows that $B + e \in \mathcal{I}(\Pi)$.

[7] We remark that the lemma can be proved using Theorem 5.3.5 in [23]. We give the proof for completeness.

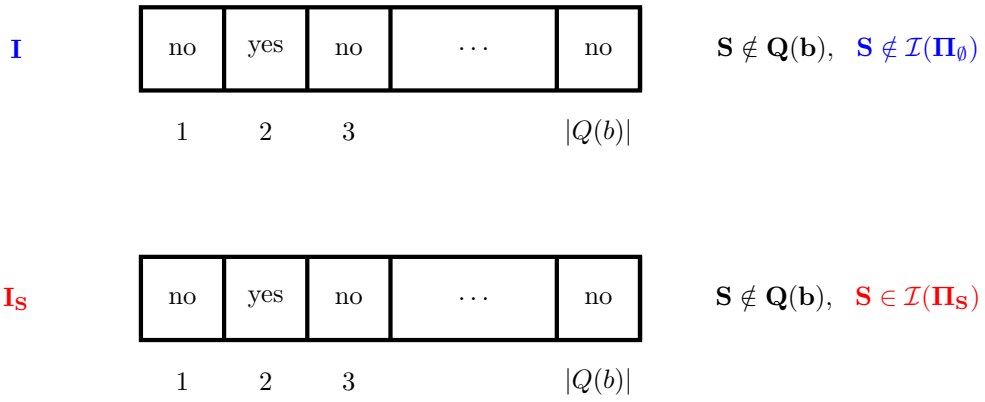


Figure 2 An illustration of the proof of Theorem 3. The figure presents the sequences of queries to the membership oracles by the algorithm on the instances I and I_S for a string of bits b, such that $S \notin Q(b)$. The label "yes" ("no") indicates that the queried set is (not) independent in the matroid. The only query that distinguishes between I and I_S is on the set S, which is not queried; thus, the algorithm returns the same output for I and I_S.

b. $B \not\subseteq A$. Then, as $|B| = k-1$ and $|A| = k$ it follows that $|B \cap A| < |B| = k-1$. Thus, $|A \setminus B| = |A| - |A \cap B| > k - (k-1) = 1$. Hence, there are $e, f \in A \setminus B$ such that $e \neq f$. It follows that there is $g \in \{e, f\}$ such that $\mathsf{sum}(B+g) = \mathsf{sum}(B) + g \neq \alpha$. We conclude from (2) that $B + g \in \mathcal{K}$, implying that $B + g \in \mathcal{I}(\Pi)$. ◀

Observe that $\mathcal{K}_{n,k,\alpha} \cup \mathcal{L}_{n,k,\alpha}(\Pi)$ is the set of bases of the matroid. Moreover, for any arguments n, k, α and Π, the cardinality of every dependent set $S \in 2^{[n]} \setminus \mathcal{I}_{n,k,\alpha}(\Pi)$ is at least the rank of the Π-matroid $M_{n,k,\alpha}(\Pi) = ([n], \mathcal{I}_{n,k,\alpha}(\Pi))$. Such matroids are known as *paving matroids* (see, e.g., [34, 35]). Using Π-matroids, we define the following collection of oracle-EMB instances. In these instances, the matroid is a Π-matroid, where Π is some unknown fixed family of subsets of the ground set.

▶ **Definition 6.** *For every $n, k, \alpha \in \mathbb{N}_{>0}$, and $\Pi \subseteq 2^{[n]}$ define the* induced *oracle*-EMB *instance of n, k, α, Π, denoted $I_{n,k,\alpha}(\Pi)$, as follows. Let $\mathsf{id}_n : [n] \to [n]$, where $\mathsf{id}_n(i) = i$ $\forall i \in [n]$. Then, $I_{n,k,\alpha}(\Pi) = ([n], \mathcal{I}_{n,k,\alpha}(\Pi), \mathsf{id}_n, \alpha)$.*

Observe that the above is indeed an oracle-EMB instance if the independent sets of the given matroid are accessible via a membership oracle. The following is an easy consequence of Definition 6.

▶ **Observation 7.** *For every $n, k, \alpha \in \mathbb{N}_{>0}$ and $\Pi \subseteq 2^{[n]}$, it holds that $I_{n,k,\alpha}(\Pi)$ is an* oracle-EMB *"yes"-instance if and only if there is $S \in \Pi$ such that $|S| = k$ and $\mathsf{sum}(S) = \mathsf{id}_n(S) = \alpha$.*

By Observation 7, an algorithm that finds an independent set of $\mathcal{M}_{n,k,\alpha}(\Pi)$ satisfying $|S| = k$ and $\mathsf{sum}(S) = \alpha$, in fact outputs a subset $S \in \Pi$. As the input for an induced oracle-EMB instance $I_{n,k,\alpha}(\Pi)$ does not contain an explicit encoding of Π, finding $S \in \Pi$ requires a sequence of queries to the membership oracle of $M_{n,k,\alpha}(\Pi)$. Roughly speaking, to decide $I_{n,k,\alpha}(\Pi)$, an algorithm for oracle-EMB must iterate over all (exponentially many) subsets $S \subseteq [n]$ such that $|S| = k$ and $\mathsf{sum}(S) = \alpha$. This is the intuition behind the proof of the next result.

▶ **Theorem 3.** *For any* oracle-EMB *instance $I = (E, \mathcal{I}, c, T)$, there is no randomized algorithm for* ORACLE-EMB *that runs in time $(n \cdot (T+2) \cdot m)^{O(1)}$, where $n = |E| + 1$ and $m = c(E) + 1$.*

Proof. Assume towards contradiction that there exist a constant $d \in \mathbb{N}$ and a randomized algorithm $\mathcal{A}$ that decides every oracle-EMB instance $(E, \mathcal{I}, c, T)$ in time $((T+2) \cdot n \cdot m)^d$, where $n = |E| + 1$ and $m = c(E) + 1$. For every $n, k, \alpha \in \mathbb{N}_{>0}$, consider the set of all subsets of $[n]$ with cardinality k and sum α, i.e.,

$$\mathcal{F}_{n,k,\alpha} = \left\{ S \subseteq [n] \;\middle|\; |S| = k, \mathsf{sum}(S) = \alpha \right\}.$$

To reach a contradiction, we construct an induced oracle-EMB instance on which $\mathcal{A}$ does not compute the proper output with sufficiently high probability. The parameters of the instance are extracted from the following combinatorial claim.

▷ Claim 8. There are $\tilde{n} \in \mathbb{N}_{>0}$, $\tilde{k} \in [\tilde{n}]$, and $\tilde{\alpha} \in \left[\tilde{n}^2\right]$ such that $|\mathcal{F}_{\tilde{n},\tilde{k},\tilde{\alpha}}| > 2 \cdot \left(12 \cdot \tilde{n}^5\right)^d$.

Proof. Since d is a constant, there is $\tilde{n} \in \mathbb{N}_{>0}$ such that

$$\left(12 \cdot \tilde{n}^5\right)^d < \frac{2^{\tilde{n}} - 1}{2 \cdot \tilde{n}^3}. \tag{3}$$

Fix $\tilde{n} \in \mathbb{N}_{>0}$ satisfying (3). Recall that $\sum_{k \in \{0,1,\ldots,\tilde{n}\}} \binom{\tilde{n}}{k} = 2^{\tilde{n}}$ from a basic property of the Pascal triangle; therefore, $\sum_{k \in \{1,\ldots,\tilde{n}\}} \binom{\tilde{n}}{k} = 2^{\tilde{n}} - 1$. Thus, there is $\tilde{k} \in \{1, \ldots, \tilde{n}\}$, such that

$$\binom{\tilde{n}}{\tilde{k}} \geq \frac{2^{\tilde{n}} - 1}{\tilde{n}}. \tag{4}$$

Fix $\tilde{k} \in [\tilde{n}]$ satisfying (4). Observe that for each $S \in 2^{[\tilde{n}]}$ satisfying $|S| = \tilde{k} > 0$, it holds that $1 \leq \mathsf{sum}(S) \leq |S| \cdot \max_{i \in [\tilde{n}]} i = \tilde{n}^2$. Thus, there are $\tilde{n}^2$ possibilities for $\alpha \in \left[\tilde{n}^2\right]$ satisfying $\alpha = \mathsf{sum}(S)$, for some $S \in 2^{[\tilde{n}]}$ such that $|S| = \tilde{k}$. Moreover, there are $\binom{\tilde{n}}{\tilde{k}}$ subsets of $[\tilde{n}]$ of cardinality $\tilde{k}$. By the pigeonhole principle, there is $\tilde{\alpha} \in \left[\tilde{n}^2\right]$ such that $|\mathcal{F}_{\tilde{n},\tilde{k},\tilde{\alpha}}| \geq \frac{\binom{\tilde{n}}{\tilde{k}}}{\tilde{n}^2}$. Thus,

$$|\mathcal{F}_{\tilde{n},\tilde{k},\tilde{\alpha}}| \geq \frac{\binom{\tilde{n}}{\tilde{k}}}{\tilde{n}^2} \geq \frac{2^{\tilde{n}} - 1}{\tilde{n} \cdot \tilde{n}^2} > 2 \cdot \left(12 \cdot \tilde{n}^5\right)^d.$$

The second inequality follows from (4), and the third inequality holds by (3). ◁

Let $\tilde{n} \in \mathbb{N}_{>0}$, $\tilde{k} \in [\tilde{n}]$, and $\tilde{\alpha} \in \left[\tilde{n}^2\right]$ satisfying the conditions of Claim 8. Define t to be the maximum running time of $\mathcal{A}$ on an induced EMB instance $I_{\tilde{n},\tilde{k},\tilde{\alpha}}(\Pi)$ over all $\Pi \in 2^{[\tilde{n}]}$.

▷ Claim 9. $t \leq \left(12 \cdot \tilde{n}^5\right)^d$.

Proof. Let $T = \tilde{\alpha}$, $n = \tilde{n} + 1$, and $m = \mathsf{id}_{\tilde{n}}([\tilde{n}]) + 1$. By the running time guarantee of $\mathcal{A}$, it follows that $t \leq (n \cdot (T+2) \cdot m)^d$. It remains to bound $n \cdot (T+2) \cdot m$. Since $\mathsf{id}_{\tilde{n}}([\tilde{n}]) = \mathsf{sum}([\tilde{n}])$ and $\tilde{\alpha} \leq \tilde{n}^2$,

$$n \cdot (T+2) \cdot m \leq (\tilde{n}+1) \cdot \left(\tilde{n}^2 + 2\right) \left(\mathsf{sum}([\tilde{n}]) + 1\right) \leq 2\tilde{n} \cdot 3\tilde{n}^2 \cdot \left(\frac{\tilde{n} \cdot (\tilde{n}+1)}{2} + 1\right) \leq 12 \cdot \tilde{n}^5.$$

The second inequality follows from the sum of the terms of an arithmetic sequence. By the above and the running time guarantee of $\mathcal{A}$, it follows that $t \leq (n \cdot (T+2) \cdot m)^d \leq \left(12 \cdot \tilde{n}^5\right)^d$. ◁

Given an induced oracle-EMB instance $I_{\tilde{n},\tilde{k},\tilde{\alpha}}(\Pi)$, for some $\Pi \subseteq [\tilde{n}]$, the randomized algorithm $\mathcal{A}$ generates a random string of bits $\bar{b} \in \{0,1\}^t$ and performs a sequence of queries to the membership oracle of $M_{\tilde{n},\tilde{k},\tilde{\alpha}}$, based on $\bar{b}, \tilde{n}, \tilde{k}, \tilde{\alpha}$, and the results of the previous queries. Then, the algorithm decides the given instance based on the queries.

Let $\Pi_\emptyset = \emptyset$, and consider the induced oracle-EMB instance $I = I_{\tilde{n},\tilde{k},\tilde{\alpha}}(\Pi_0)$. Given a string of bits $b \in \{0,1\}^t$, let $Q(b) \subseteq 2^{[\tilde{n}]}$ be the set of all subsets $S \subseteq [\tilde{n}]$ queried by $\mathcal{A}$ on the instance I and the bit-string b (on the membership oracle of $M_{\tilde{n},\tilde{k},\tilde{\alpha}}(\Pi_\emptyset)$). For clarity, we use $\bar{b}$ for a random string, and b for a realization of $\bar{b}$ to a specific string. Note that $Q(b)$ is a set, since the algorithm is deterministic for every $b \in \{0,1\}^t$; conversely, $Q(\bar{b})$ is a random set for a random string $\bar{b} \in \{0,1\}^t$. As the running time of $\mathcal{A}$ on I is bounded by t, it holds that $|Q(b)| \leq t$ for every $b \in \{0,1\}^t$. Let

$$R(I) = \left\{ S \in \mathcal{F}_{\tilde{n},\tilde{k},\tilde{\alpha}} \;\middle|\; \Pr\left(S \in Q(\bar{b})\right) \geq \frac{1}{2} \right\}$$

be all sets in $\mathcal{F}_{\tilde{n},\tilde{k},\tilde{\alpha}}$ that are queried by $\mathcal{A}$ with probability at least $\frac{1}{2}$.

▷ Claim 10. $|R(I)| < |\mathcal{F}_{\tilde{n},\tilde{k},\tilde{\alpha}}|$.

Proof. By the definition of $R(I)$, it holds that

$$\begin{aligned} |R(I)| &= \left| \left\{ S \in \mathcal{F}_{\tilde{n},\tilde{k},\tilde{\alpha}} \;\middle|\; \Pr\left(S \in Q(\bar{b})\right) \geq \frac{1}{2} \right\} \right| \\ &\leq 2 \cdot \sum_{S \in \mathcal{F}_{\tilde{n},\tilde{k},\tilde{\alpha}}} \Pr\left(S \in Q(\bar{b})\right) \\ &= 2 \cdot \sum_{S \in \mathcal{F}_{\tilde{n},\tilde{k},\tilde{\alpha}}} \sum_{b \in \{0,1\}^t} \Pr\left(S \in Q(b)\right) \cdot \Pr(\bar{b} = b). \end{aligned}$$

Thus, by changing the order of summation, we have

$$|R(I)| \leq 2 \cdot \sum_{b \in \{0,1\}^t} \Pr(\bar{b} = b) \cdot \sum_{S \in \mathcal{F}_{\tilde{n},\tilde{k},\tilde{\alpha}}} \Pr\left(S \in Q(b)\right) \leq 2 \cdot \sum_{b \in \{0,1\}^t} \Pr(\bar{b} = b) \cdot |Q(b)|.$$

Since $|Q(b)| \leq t$ for all $b \in \{0,1\}^t$, by the above we have

$$|R(I)| \leq 2t \cdot \sum_{b \in \{0,1\}^t} \Pr(\bar{b} = b) = 2 \cdot t \leq 2 \cdot \left(12 \cdot \tilde{n}^5\right)^d < |\mathcal{F}_{\tilde{n},\tilde{k},\tilde{\alpha}}|.$$

The second inequality follows from Claim 9. The last inequality holds by Claim 8. ◁

By Claim 10, there exists $S \in F_{\tilde{n},\tilde{k},\tilde{\alpha}} \setminus R(I)$. Consider the induced oracle-EMB instance $I_S = I_{\tilde{n},\tilde{k},\tilde{\alpha}}(\Pi_S)$ where $\Pi_S = \{S\}$, and let $B = \{b \in \{0,1\}^t \mid S \notin Q(b)\}$ be all strings for which S is not queried by $\mathcal{A}$ on the instance I. Observe that for all $b \in B$ it holds that the answers to all queries for $T \in Q(b)$ are the same for both oracles (of $M_{\tilde{n},\tilde{k},\tilde{\alpha}}(\Pi_\emptyset)$ and $M_{\tilde{n},\tilde{k},\tilde{\alpha}}(\Pi_S)$). Moreover, the decision on which set to query next depends only on $b, \tilde{n}, \tilde{k}, \tilde{\alpha}$, and the answers to previous queries.

Hence, for all $b \in B$, the executions of $\mathcal{A}$ on the instances I and I_S are identical. Since $\Pi_\emptyset = \emptyset$, by Observation 7, I is a "no"-instance for oracle-EMB; thus, $\mathcal{A}$ returns that I_S is a "no"-instance for every $b \in B$. However, since $\mathsf{id}_n(S) = \mathsf{sum}(S) = \alpha$, $|S| = k$, and $S \in \Pi_S$, it follows that I_S is a "yes"-instance by Observation 7. Therefore, $\mathcal{A}$ does not decide I_S correctly for all $b \in B$. We give an illustration in Figure 2. Since $S \notin R(I)$, it holds that $\Pr\left(\bar{b} \in B\right) = \Pr\left(S \notin Q(\bar{b})\right) > \frac{1}{2}$; thus, with probability greater than $\frac{1}{2}$, $\mathcal{A}$ does not decide correctly the instance I_S. A contradiction to the correctness of $\mathcal{A}$ as a randomized algorithm for oracle-EMB. The statement of the theorem follows. ◀

3 Hardness of Matroid Optimization with a Linear Constraint

In this section we use Theorem 3 to prove Theorem 1. We apply the following reductions from EMB to MOL problems. Recall that $\mathcal{Q} = \mathcal{P} \setminus \{(\min, \text{IS}, \leq)\}$ is the set of parameters for non-trivial MOL problems. Given a P-MOL problem for some $P \in \mathcal{Q}$, and an EMB instance I, the reduction returns an instance $R_P(I)$ of the P-MOL problem. Note that the reduction is purely mathematical, and does not specify the encoding of the instance. This will be useful for obtaining our hardness results in non-oracle computational models (see Section 4).

Given $(\text{opt}, \mathcal{F}, \triangleleft), (\text{opt}', \mathcal{F}', \triangleleft') \in \mathcal{Q}$, we use the notation $(\text{opt is opt}')$, $(\mathcal{F}$ is $\mathcal{F}')$, $(\triangleleft$ is $\triangleleft')$ to denote the boolean expressions of equality between parameters of a MOL problem. For example, $(\text{opt is opt}')$ is *true* if and only if either opt, opt' are both max, or opt, opt' are both min.

▶ **Definition 11.** *Given an* EMB *instance* $I = (E, \mathcal{I}, c, T)$ *and* $P \in \mathcal{Q}$ *where* $P = (\text{opt}, \mathcal{F}, \triangleleft)$, *define the* reduced P-MOL instance *of* I, *denoted by* $R_P(I) = (E, \mathcal{I}, v_I, w_{I,P}, L_{I,P})$, *as follows.*

1. *Define the auxiliary variable*

$$d(P) = \begin{cases} 0 & \text{if } \Big((\text{opt is max}) \text{ and } (\triangleleft \text{ is } \leq)\Big) \text{ or } \Big((\text{opt is min}) \text{ and } (\triangleleft \text{ is } \geq)\Big) \\ 1 & \text{otherwise.} \end{cases}$$

   For example, if $P = (\max, \text{IS}, \leq)$ *then* $d(P) = 0$, *and if* $P' = (\max, \text{IS}, \geq)$ *then* $d(P') = 1$.
2. *Let* $H_I = 2 \cdot \max\{1, c(E)\}$.
3. *For all* $e \in E$ *let* $v_I(e) = H_I + c(e)$.
4. *For all* $e \in E$ *let* $w_{I,P}(e) = H_I + c(e) \cdot (-1)^{d(P)}$.
5. *Let* $k_I = \max_{S \in \mathcal{I}} |S|$ *be the rank of* $(E, \mathcal{I})$.
6. *Define* $L_{I,P} = k_I \cdot H_I + T \cdot (-1)^{d(P)}$.

Now, for every EMB instance $I = (E, \mathcal{I}, c, T)$, define the *error parameter* of I as

$$\varepsilon_I = \frac{1}{8 \cdot (|E| + 1) \cdot (T + 1) \cdot (c(E) + 1)}. \tag{5}$$

Indeed, since the value selected for of the error parameter is sufficiently small, we can use a $(1 + \varepsilon_I)$-approximation for $R_P(I)$ to decide an EMB instance I.

▶ **Theorem 12.** *Given an instance* $I = (E, \mathcal{I}, c, T)$ *of* EMB, *and* $P \in \mathcal{Q}$ *with* $P = (\text{opt}, \mathcal{F}, \triangleleft)$, *the following holds.*

1. *If there is a solution* S *for* $R_P(I)$ *such that* $v_I(S) = k_I \cdot H_I + T$, *then* I *is a "yes"-instance.*
2. *If* I *is a "yes"-instance then: (i)* $R_P(I)$ *has a solution, and (ii) every* $(1+\varepsilon_I)$*-approximate solution* S *for* $R_P(I)$ *satisfies* $v_I(S) = k_I \cdot H_I + T$.

Using Theorem 12 and an assumed randomized Fully PTAS for the P-MOL problem, we can decide EMB in time which contradicts Theorem 3. This gives the proof of Theorem 1. We first prove Theorem 1, and later give the proof of Theorem 12.

▶ **Theorem 1.** *For every* $P \in \mathcal{Q}$ *there is no randomized* Fully PTAS *for oracle* P-MOL.

Proof. Assume towards a contradiction that there is a randomized Fully PTAS $\mathcal{A}$ for oracle P-MOL. We use $\mathcal{A}$ to decide oracle-EMB. Let $I = (E, \mathcal{I}, c, T)$ be an oracle-EMB instance, and consider the following randomized algorithm $\mathcal{B}$ that decides I.

1. Construct the oracle P-MOL instance $R_P(I)$ with the membership oracle of $(E, \mathcal{I})$.
2. Execute $\mathcal{A}$ with the input $R_P(I)$ and ε_I.
3. If $\mathcal{A}$ returns that $R_P(I)$ does not have a solution − Return "no" on I.
4. Otherwise, let $S \leftarrow \mathcal{A}(R_P(I), \varepsilon_I)$ be the solution returned by $\mathcal{A}$.
5. Return "yes" on I if and only if $v_I(S) = H_I \cdot k_I + T$.

Let $n = |E| + 1$ and $m = c(E) + 1$. Note that $R_P(I)$ can be naively constructed from I in time $(n \cdot (T+2) \cdot m)^{O(1)}$ using Definition 11. As $\mathcal{A}$ is a randomized Fully PTAS for oracle P-MOL, and by the selection of the error parameter (5), the running time of $\mathcal{B}$ on I is $(n \cdot (T+2) \cdot m)^{O(1)}$. We now show correctness.

- If $\mathcal{B}$ returns "yes" on I, then Step 4 of the algorithm computes a solution S for $R_p(I)$ satisfying $v_I(S) = H_I \cdot k_I + T$. Thus, by Theorem 12, I is a "yes" instance.
- If I is a "yes" instance then $R_P(I)$ has a solution by Theorem 12. As $\mathcal{A}$ is a randomized Fully PTAS, with probability at least $\frac{1}{2}$ $\mathcal{A}$ returns a $(1+\varepsilon_I)$-approximate solution S for $R_P(I)$ in Step 4. By Theorem 12, $v_I(S) = H_I \cdot k_I + T$ (with probability at least $\frac{1}{2}$). Thus, $\mathcal{B}$ returns "yes" on I with probability at least $\frac{1}{2}$.

Hence, $\mathcal{B}$ is a randomized algorithm which decides the oracle-EMB instance I in time $(n \cdot (T+2) \cdot m)^{O(1)}$. This is a contradiction to Theorem 3. ◀

In the remainder of this section we prove Theorem 12. We start with some basic properties of the reduction outlined in Definition 11.

▶ **Lemma 13.** *Given an instance $I = (E, \mathcal{I}, c, T)$ of* EMB*, let $P \in \mathcal{Q}$ and consider a solution S for $R_P(I)$ satisfying $v_I(S) = k_I \cdot H_I + T$. Then, S is a solution for I.*

Proof. Let $\mathcal{M} = (E, \mathcal{I})$. As S is a solution for $R_P(I)$, it holds that $S \in \mathrm{IS}(\mathcal{M})$; thus, $|S| \leq k_I$. Assume towards contradiction that $|S| < k_I$. Then,

$$v_I(S) = |S| \cdot H_I + c(S) \leq (k_I - 1) \cdot H_I + c(S) \leq (k_I - 1) \cdot H_I + c(E) < k_I \cdot H_I \leq k_I \cdot H_I + T.$$

We reach a contradiction since $v_I(S) = k_I \cdot H_I + T$; thus, $|S| = k_I$, and

$$k_I \cdot H_I + T = v_I(S) = |S| \cdot H_I + c(S) = k_I \cdot H_I + c(S). \tag{6}$$

As $|S| = k_I$, we have that S is a basis of $\mathcal{M}$, and by (6), $c(S) = T$. Hence, S is a solution for I. ◀

The next result is the converse of the statement in Lemma 13.

▶ **Lemma 14.** *Let S be a solution for a given* EMB *instance $I = (E, \mathcal{I}, c, T)$, and let $P \in \mathcal{Q}$ where $P = (\mathrm{opt}, \mathcal{F}, \triangleleft)$. Then, S is a solution for $R_P(I)$ of value $v_I(S) = k_I \cdot H_I + T$.*

Proof. Let $\mathcal{M} = (E, \mathcal{I})$. Since S is a solution for I we have $S \in \mathrm{bases}(\mathcal{M})$; thus, $S \in \mathcal{F}(\mathcal{M})$. Then,

$$w_{I,P}(S) = |S| \cdot H_I + c(S) \cdot (-1)^{d(P)} = k_I \cdot H_I + T \cdot (-1)^{d(P)} = L_{I,P}.$$

The second equality holds since S is a solution for I; thus, $|S| = k_I$ (as S is a basis of $\mathcal{M}$), and $c(S) = T$. We conclude that S is a solution for $R_P(I)$. Finally, note that S satisfies

$$v_I(S) = |S| \cdot H_I + c(S) = k_I \cdot H_I + T$$ ◀

The next claim gives an upper bound on the optimal value for maximization MOL problems. We then derive an analogous lower bound for minimization (non-trivial) MOL problems.

▶ **Lemma 15.** *Let $I = (E, \mathcal{I}, c, T)$ be an* EMB *instance and $P \in \mathcal{Q}$, where $P = (\text{opt}, \mathcal{F}, \triangleleft)$ and* (opt is max)*. Then, for every solution S of $R_P(I)$ it holds that $v_I(S) \leq k_I \cdot H_I + T$.*

Proof. Let S be an optimal solution for $R_P(I)$. Thus, $|S| \leq k_I$ as $S \in \mathcal{I}$. If $|S| < k_I$ then

$$v_I(S) \leq (k_I - 1) \cdot H_I + c(S) \leq (k_I - 1) \cdot H_I + c(E) < k_I \cdot H_I \leq k_I \cdot H_I + T.$$

Otherwise, $|S| = k_I$. Consider the two cases for $\triangleleft$.

1. ($\triangleleft$ is $\leq$). Then, $d(P) = 0$ (see Definition 11); thus, since S is a solution for $R_P(I)$:

$$v_I(S) = w_{I,P}(S) \leq L_{I,P} = k_I \cdot H_I + T.$$

2. ($\triangleleft$ is $\geq$). Then, $d(P) = 1$. As S is a solution for $R_P(I)$,

$$k_I \cdot H_I - c(S) = |S| \cdot H_I - c(S) = w_{I,P}(S) \geq L_{I,P} = k_I \cdot H_I - T. \tag{7}$$

By (7), it follows that $c(S) \leq T$; thus, $v_I(S) = k_I \cdot H_I + c(S) \leq k_I \cdot H_I + T$.

In all the above cases, we have that $v_I(S) \leq k_I \cdot H_I + T$, implying the statement of the lemma. ◀

Now, for minimization problems we have the next result.

▶ **Lemma 16.** *Let $I = (E, \mathcal{I}, c, T)$ be an* EMB *instance, and $P \in \mathcal{Q}$, where $P = (\text{opt}, \mathcal{F}, \triangleleft)$ and* (opt is min)*. Then, for every solution S of $R_P(I)$, it holds that $v_I(S) \geq k_I \cdot H_I + T$.*

Using Lemmas 13–16, we can now prove Theorem 12.

▶ **Theorem 12.** *Given an instance $I = (E, \mathcal{I}, c, T)$ of* EMB*, and $P \in \mathcal{Q}$ with $P = (\text{opt}, \mathcal{F}, \triangleleft)$, the following holds.*

1. *If there is a solution S for $R_P(I)$ such that $v_I(S) = k_I \cdot H_I + T$, then I is a "yes"-instance.*
2. *If I is a "yes"-instance then: (i) $R_P(I)$ has a solution, and (ii) every $(1+\varepsilon_I)$-approximate solution S for $R_P(I)$ satisfies $v_I(S) = k_I \cdot H_I + T$.*

Proof. We note that Property 1 follows directly from Lemma 13. For Property 2, assume that I is a "yes"-instance, then by Lemma 14, there is a solution D for $R_P(I)$ such that $v_I(D) = H_I \cdot k_I + T$. It remains to show Property 2. (ii). Let S be a $(1 + \varepsilon_I)$-approximate solution for $R_P(I)$. We distinguish between two cases.

1. (opt is max). Then, by Lemma 15,

$$0 \leq H_I \cdot k_I + T - v_I(S).$$

Moreover, since S is a $(1 + \varepsilon_I)$-approximate solution for $R_P(I)$, and D is a solution for $R_P(I)$,

$$H_I \cdot k_I + T - v_I(S) \leq H_I \cdot k_I + T - \frac{v_I(D)}{(1 + \varepsilon_I)} = \frac{\varepsilon_I \cdot (H_I \cdot k_I + T)}{(1 + \varepsilon_I)} \leq \varepsilon_I \cdot (H_I \cdot k_I + T).$$

By the above, it follows that

$$|v_I(S) - (H_I \cdot k_I + T)| \leq \varepsilon_I \cdot (H_I \cdot k_I + T).$$

2. (opt is min). This case is analogous to the above. By Lemma 16,

$$0 \leq v_I(S) - (H_I \cdot k_I + T)\,.$$

Since S is a $(1+\varepsilon_I)$-approximate solution for $R_P(I)$, and D is a solution for $R_P(I)$,

$$v_I(S) - (H_I \cdot k_I + T) \leq (1+\varepsilon_I) \cdot v_I(D) - (H_I \cdot k_I + T) = \varepsilon_I \cdot (H_I \cdot k_I + T)\,.$$

By the above,

$$|v_I(S) - (H_I \cdot k_I + T)| \leq \varepsilon_I \cdot (H_I \cdot k_I + T)\,.$$

Thus in both cases it holds that,

$$|v_I(S) - (H_I \cdot k_I + T)| \leq \varepsilon_I \cdot (H_I \cdot k_I + T)\,. \tag{8}$$

Let $n = |E| + 1$ and $m = c(E) + 1$. Then, by the selection of ε_I in (5),

$$\varepsilon_I \cdot (H_I \cdot k_I + T) = \frac{H_I \cdot k_I + T}{8 \cdot n \cdot (T+1) \cdot m} \leq \frac{2 \cdot m \cdot n + T}{8 \cdot n \cdot (T+1) \cdot m} < \frac{4 \cdot m \cdot n \cdot (T+1)}{8 \cdot n \cdot (T+1) \cdot m} = \frac{1}{2}. \tag{9}$$

The first inequality holds since $k_I \leq |E|$ and $H_I \leq 2 \cdot m$. Therefore, by (8) and (9),

$$|v_I(S) - (H_I \cdot k_I + T)| \leq \varepsilon_I \cdot (H_I \cdot k_I + T) < \frac{1}{2}. \tag{10}$$

Since $v_I(S) \in \mathbb{N}$ by Definition 11, it follows from (10) that $v_I(S) = H_I \cdot k_I + T$. This gives the statement of the theorem. ◀

4 Lower Bounds in the Standard Computational Model

Our hardness result in Section 2 shows that oracle Exact Matroid Basis (EMB) is hard, leading to the unconditional lower bounds for all non-trivial oracle MOL problems in Section 3. Nonetheless, these hardness results consider matroids with general membership oracles, and do not give a lower bound for matroids that can be efficiently encoded. This is particularly important, as in some settings oracle models differ from non-oracle models w.r.t complexity [6, 9]. Moreover, some matroids show up in problems that can be encoded efficiently. This includes partition matroids, graphic matroids, linear matroids, etc (see, e.g., [37] for a survey on various families of matroids). Next, we formally define an efficient encoding of matroids.

▶ **Definition 17.** *A function $f : \{0,1\}^* \to 2^{\mathbb{N}} \times 2^{2^{\mathbb{N}}}$ is called* matroid decoder *if for every $I \in \{0,1\}^*$ it holds that $f(I) = \left(E_{f(I)}, \mathcal{I}_{f(I)}\right)$ is a matroid, and the following holds.*

1. *There is an algorithm that given $I \in \{0,1\}^*$ returns $E_{f(I)}$ in time $|I|^{O(1)}$.*
2. *There is an algorithm that given $I \in \{0,1\}^*$ and $S \subseteq E_{f(I)}$ decides if $S \in \mathcal{I}_{f(I)}$ in time $|I|^{O(1)}$.*

There is a simple matroid decoder that can decode every matroid $(E, \mathcal{I})$ (such that $E \subseteq \mathbb{N}$), in which the encoding I explicitly lists $\mathcal{I}$. However, using such a matroid decoder, the encoding size of a matroid might be very large, up to $|I| = \Omega\left(2^{|E|}\right)$, while we often seek algorithms with running times polynomial in $|E|$. One way to overcome this difficulty is via the oracle model considered in previous sections. However, our results in this model may suggest that the hardness of EMB and MOL problems is due to the intrinsic hardness of the oracle model. Yet, there are families of matroids with very efficient encoding. For

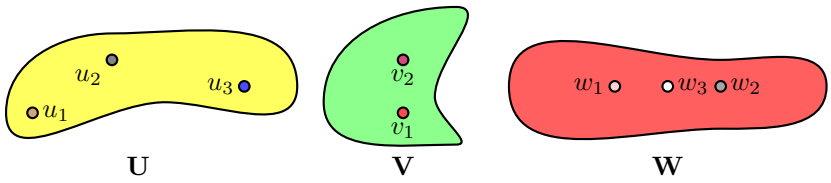


Figure 3 An example of a *partition matroid* $(E, \mathcal{I})$, which can be efficiently encoded. The ground set is $E = \{u_1, u_2, u_3, v_1, v_2, w_1, w_2, w_3\}$, partitioned into three sets: U, V, W. The independent sets are all subsets of E containing at most one element from U, V, and W; that is, $\mathcal{I} = \{S \subseteq E \mid \forall X \in \{U, V, W\} : |S \cap X| \leq 1\}$. A simple efficient encoding of $(E, \mathcal{I})$ is $I = (E, U, V, W)$. Membership can be decided efficiently given I, by checking the feasibility of a given set S w.r.t. U, V and W.

example, a *uniform matroid* $(E, \mathcal{I})$, where $\mathcal{I} = \{S \subseteq E \mid |S| \leq k\}$, can be efficiently encoded using $I = (E, k) \in \{0,1\}^*$. Clearly, the time to decide membership of a given subset $S \subseteq E$ depends only on $|S|$ and k. Another example is given in Figure 3.

We start with a definition of an encoded variant of EMB. We technically define a different problem for every decoder f. The definition of the problem is analogous to the versions of EMB considered earlier in the paper, besides that the matroid is given via an arbitrary bit-string $I \in \{0,1\}^*$, that a matroid decoder f decodes into a matroid $f(I)$.

f-DECODED EXACT MATROID BASIS **(f-decoded EMB)**
Decoder $f : \{0,1\}^* \to 2^{\mathbb{N}} \times 2^{2^{\mathbb{N}}}$ is a matroid decoder.
Instance (I, c, T), where $I \in \{0,1\}^*$, $c : E_{f(I)} \to \mathbb{N}$, $T \in \mathbb{N}$.
Solution A basis S of the matroid $f(I)$ such that $c(S) = T$.
Objective Decide if there is a solution.

▶ **Definition 18.** *The* f-DECODED EXACT MATROID BASIS *(f-decoded EMB) problem is defined as follows.*

- ***Decoder:*** *$f : \{0,1\}^* \to 2^{\mathbb{N}} \times 2^{2^{\mathbb{N}}}$ is a matroid decoder.*
- ***Instance:*** *(I, c, T), where $I \in \{0,1\}^*$, $c : E_{f(I)} \to \mathbb{N}$, $T \in \mathbb{N}$.*
- ***Solution:*** *A basis S of the matroid $f(I)$ such that $c(S) = T$.*
- ***Objective:*** *Decide if there is a solution.*

As a simple example, consider the f_{u}-decoded EMB problem, for a specific matroid decoder f_{u} that decodes uniform matroids. The matroid decoder f_{u} interprets every $I \in \{0,1\}^*$ as $I = (E, k)$ where E is a set (of numbers) and $k \in \mathbb{N}$, and returns the uniform matroid $f_{\mathsf{u}}(I) = (E, \mathcal{I})$ such that $\mathcal{I} = \{S \subseteq E \mid |S| \leq k\}$; clearly, f_{u} is a matroid decoder. Thus, an instance of f_{u}-decoded EMB is a tuple $U = ((E, k), c, T)$ and a solution of U is $S \subseteq E$ such that $|S| = k$ and $c(S) = T$; the goal, as before, is to decide if there is a solution. This problem is commonly known as the k-SUBSET SUM.

Recall that Theorem 3 asserts that oracle-EMB does not admit a pseudo-polynomial time algorithm. However, this does not rule out that hypothetically, for every matroid decoder f there is a pseudo-polynomial time algorithm for f-decoded EMB. The next result excludes this option.

▶ **Theorem 19.** *Assuming* P $\neq$ NP*, there is a matroid decoder f such that there is no algorithm for f-*decoded EMB *that for any f-*decoded EMB *instance $U = (I, c, T)$, where $n = |E_{f(I)}| + 1$ and $m = c(E_{f(I)}) + 1$, runs in time $(n \cdot (T+2) \cdot m)^{O(1)}$.*

The proof of Theorem 19 is given towards the end of this section. Analogously to our hardness result for oracle MOL problems, we use Theorem 19 to give a hardness result for an encoded version of MOL problems. For every matroid decoder f and every $P \in \mathcal{P}$, we define a variant of the P-MATROID OPTIMIZATION WITH A LINEAR CONSTRAINT (P-MOL) problem in which the matroid is given via an arbitrary bit-string which a matroid decoder f decodes into a matroid. Formally, let $P \in \mathcal{P}$, where $P = (\text{opt}, \mathcal{F}, \triangleleft)$, be the parameters of the P-MOL problem. For a matroid decoder f, we define the f-decoded P-MOL problem as follows.

▶ **Definition 20.** *The f-*DECODED *P-*MATROID OPTIMIZATION WITH A LINEAR CONSTRAINT *(f-decoded P-MOL) problem is defined as follows.*

- ***Decoder:*** $f : \{0,1\}^* \to 2^{\mathbb{N}} \times 2^{2^{\mathbb{N}}}$ *is a matroid decoder.*
- ***Instance:*** (I, v, w, L)*, where* $I \in \{0,1\}^*$*,* $v : E_{f(I)} \to \mathbb{R}_{\geq 0}$*,* $w : E_{f(I)} \to \mathbb{R}_{\geq 0}$*,* $L \in \mathbb{R}_{\geq 0}$*.*
- ***Solution:*** *A basis* S *of the matroid* $f(I)$ *such that* $c(S) = T$*.*
- ***Objective:*** opt $v(S)$ *s.t.* $S \in \mathcal{F}\left(f(I)\right),\ w(S) \triangleleft L$*.*

For example, consider the encoded version of the P-MOL for $P = (\max, \text{IS}, \leq)$ with the matroid decoder f_{u} that decodes uniform matroids. An instance of the f_{u}-decoded P-MOL problem is a tuple $U = (I, v, w, L)$ where $I = (E, k)$ is a bit-string used for extracting the uniform matroid $f_{\mathsf{u}}(I) = (E, \mathcal{I})$ such that $\mathcal{I} = \{S \subseteq E \mid |S| \leq k\}$, v is the value function, w is the weight function, and L is the bound. A *solution* of U is $S \subseteq E$ such that $|S| \leq k$ and $w(S) \leq L$; the goal is to find a solution S of maximum value $v(S)$. This problem is widely known as KNAPSACK WITH CARDINALITY CONSTRAINT.

Recall that $\mathcal{Q} = \mathcal{P} \setminus \{(\min, \text{IS}, \leq)\}$ is the set of parameters for non-trivial MOL problems. Using the hardness of f-decoded, for some matroid decoder f (details on f are given towards the end of the section), we show the hardness of the f-decoded variant of all non-trivial MOL problems.

▶ **Theorem 21.** *Assuming* P $\neq$ NP*, for any $P \in \mathcal{Q}$ there is a matroid decoder f such that there is no* Fully PTAS *for f-*decoded *P-*MOL*.*

In the remainder of this section, we prove Theorem 19 and Theorem 21. The matroid decoder used in our proofs decodes a subclass of the Π-matroid family (see Section 2), in which the secret family Π consists of the solutions for a BOOLEAN SATISFIABILITY PROBLEM (SAT) instance.

In a SAT instance $A = (V, \bar{V}, \mathcal{C})$ with $n \in \mathbb{N}$ variables (in a slightly simplified notation), we are given a set $V = \{v_1, \ldots, v_n\}$ of variables, their negations $\bar{V} = \{\bar{v}_1, \ldots, \bar{v}_n\}$, and a set $\mathcal{C} \subseteq 2^{V \cup \bar{V}}$ of clauses. The goal is to decide if there is a set $S \subseteq [n]$ satisfying that for all $C \in \mathcal{C}$ there is $i \in [n]$ such that one of the following holds.

- $v_i \in C$ and $i \in S$.
- $\bar{v}_i \in C$ and $i \notin S$.

Such a set S is called a *solution* of A; let $\mathcal{S}(A)$ be the set of solutions of a SAT instance A. In addition, let $n(A) = n$ be the number of variables in the instance A. The family of SAT-matroids is the subfamily of Π-matroids where $\Pi = \mathcal{S}(A)$ for some SAT instance A (for the notation and definition of Π-matroids, see Definition 4). Specifically,

▶ **Definition 22.** *Let A be a* SAT *instance, $k \in [n(A)]$, and $\alpha \in \left[n(A)^2\right]$. Define the* **SAT-matroid** *on A, k, α as $M_{n(A),k,\alpha}(\mathcal{S}(A)) = \left([n(A)], \mathcal{I}_{n(A),k,\alpha}(\mathcal{S}(A))\right)$.*

We show below that SAT-matroids can be encoded efficiently. For every $I \in \{0,1\}^*$, we interpret I as $I = (A, k, \alpha)$, where A is a SAT instance, $k \in [n(A)]$, and $\alpha \in \left[n(A)^2\right]$; w.l.o.g., we may assume that every $I = (A, k, \alpha) \in \{0,1\}^*$ can be interpreted in that manner; moreover, we can assume that $n(A) \leq |A|$, where $|A|$ is the encoding size of A. The following definition defines a matroid decoder that decodes SAT-matroids from a bit-string.

▶ **Definition 23.** *Define the* SAT-decoder *as the function* $f_{\text{SAT}} : \{0,1\}^* \to 2^{\mathbb{N}} \times 2^{2^{\mathbb{N}}}$ *such that for all* $I = (A, k, \alpha) \in \{0,1\}^*$ *it holds that* $f_{\text{SAT}}(I) = M_{n(A),k,\alpha}(\mathcal{S}(A))$.

In the next result, we show that the SAT-decoder f_{SAT} is indeed a matroid decoder.

▶ **Lemma 24.** f_{SAT} *is a matroid decoder such that* $|I| = |A|^{O(1)}$ *for every* $I = (A, k, \alpha) \in \{0,1\}^*$.

Recall the sets $\mathcal{J}_{n,k}$, $\mathcal{K}_{n,k,\alpha}$, $\mathcal{L}_{n,k,\alpha}(\Pi)$ and $\mathcal{I}_{n,k,\alpha}(\Pi)$ were defined in Definition 4. We will use an algorithm for f-decoded EMB to obtain an algorithm for SAT. To this end, consider the following family of *structured* f_{SAT}-decoded EMB instances.

▶ **Definition 25.** *An* f_{SAT}-DECODED EMB *instance* (I, c, T)*, where* $I = (A, k, \alpha)$ *is called* structured *if* $\alpha \in \left[n(A)^2\right]$*,* $T = \alpha$*, and* $c : [n(A)] \to \mathbb{N}$ *such that for all* $i \in [n(A)]$ *it holds that* $c(i) = i$

The next observation immediately follows from the definition of structured f_{SAT}-decoded EMB instances and SAT-matroids.

▶ **Observation 26.** *for any structured* f_{SAT}-DECODED EMB *instance* $U = (I, c, T)$*, where* $I = (A, k, \alpha)$*, and* $S \subseteq [n(A)]$*, it holds that:* S *is a solution for* U *if and only if* $S \in \mathcal{S}(A)$*,* $|S| = k$*, and* $\mathsf{sum}(S) = c(S) = \alpha$.

We show that given a polynomial algorithm that decides structured f_{SAT}-decoded EMB instances, we can decide SAT. This result easily imply Theorem 19 as shown afterwards.

▶ **Lemma 27.** *Assuming* P $\neq$ NP*, there no algorithm that decides every* f_{SAT}*-decoded* EMB *structured instance* (I, c, T) *in time* $|I|^{O(1)}$.

From the above result, the hardness of the more general f_{SAT}-decoded EMB easily follows. As an immediate corollary, Theorem 28 gives the proof of Theorem 19.

▶ **Theorem 28.** *Assuming* P $\neq$ NP*, there is no algorithm for* f_{SAT}*-decoded* EMB *that runs in time* $(n \cdot (T+2) \cdot m)^{O(1)}$*, for any* f_{SAT}*-decoded* EMB *instance* $U = (I, c, T)$ *where* $n = \left|E_{f_{\text{SAT}}(I)}\right| + 1$ *and* $m = c\left(E_{f_{\text{SAT}}(I)}\right) + 1$.

Finally, we show that the variants of non-trivial matroid optimization with a linear constraint (MOL) problems, in which the decoding is performed by the SAT-decoder f_{SAT}, do not admit Fully PTAS under the standard assumption P $\neq$ NP. The proof is similar to the proof of Theorem 1 in Section 3. Lemma 29 directly gives the proof of Theorem 21.

▶ **Lemma 29.** *Assuming* P $\neq$ NP*, for any* $P \in \mathcal{Q}$ *there is no* Fully PTAS *for* f_{SAT}-decoded P-MOL.

5 Discussion

In this paper, we derive lower bounds for a family of matroid optimization problems with a linear constraint. We show that none of the (non-trivial) members of this family admits a Fully PTAS. In particular, this rules out a Fully PTAS for well studied problems such as BUDGETED MATROID INDEPENDENT SET, CONSTRAINED MINIMUM BASIS OF A MATROID, and KNAPSACK COVER WITH A MATROID. As BM and CMB admit an Efficient PTAS, our lower bounds resolve the complexity status of these problems, which has been open also for the generalization of BUDGETED MATROID INTERSECTION [12, 2, 17]. Our preliminary study shows that using the techniques of [26], we may be able to derive Efficient PTAS for *all* MOL problems. This would imply that Theorem 1 gives a tight lower bound for the entire MOL family. We leave the details for future work.

A key result of this paper is that EXACT MATROID BASIS (EMB) does not admit a pseudo-polynomial time algorithm, unlike the known special cases of k-SUBSET SUM and EMB on a linear matroid. Our proofs can be used to obtain lower bounds for other problems. For example, the hardness result for EMB can be adapted to yield lower bounds for related *parameterized* problems [22, 16]. Moreover, the proof of Theorem 1 can be modified to show that an Efficient PTAS for a non-trivial MOL problem with running time $f\left(\frac{1}{\varepsilon}\right)\cdot\text{poly}(n)$ must satisfy $f\left(\frac{1}{\varepsilon}\right)=\Omega\left(2^{\varepsilon^{-\frac{1}{4}}}\right)$. We leave these generalizations of our results to a later version of this paper.

Our results build on the Π-matroid family introduced in this paper. Such matroids exploit the interaction between a weight function and the underlying matroid constraint of the given problem. Aside from the implications of our results for previously studied problems, the new subclass of Π-matroids may enable to derive lower bounds for other problems. For example, consider the generalization of BM where the objective function is submodular and monotone. This is known as monotone submodular maximization with a knapsack and a matroid constraint [11]. Indeed, if the knapsack constraint is removed, there is a tight $\left(1-\frac{1}{e}\right)$-approximation for the problem [4]. The same bound holds if we relax the matroid constraint [40]. However, the best known approximation for the problem with a knapsack and a matroid constraint is $\left(1-\frac{1}{e}-\varepsilon\right)$ [11]. This setting resembles the status of MOL problems prior to our work, where removing either the linear or the matroid constraint induces a substantially easier problem. The potential use of Π-matroid variants to rule out a $\left(1-\frac{1}{e}\right)$-approximation for the above problem remains an interesting open question.

In the context of solving *configuration* LPs for packing problems with a matroid constraint (e.g., [24, 15]), our lower bound implies that an FPTAS for an LP in this class cannot be obtained using the standard ellipsoid method.

We show unconditional hardness results in the oracle model (even if randomization is allowed), and give analogous lower bounds where the matroids are encoded as part of the input, assuming P $\neq$ NP. Our construction in Section 4 can be used to derive hardness results for other matroid problems in non-oracle models. Specifically, we can obtain in the standard computational model hardness results analogous to those in the oracle model of [28]. This includes a proof that it is NP-hard to decide if a given matroid is uniform, analogous to the unconditional hardness result in the oracle model of [28]. We leave these results for future work.

Our lower bounds for MOL problems on general matroids call for a more comprehensive study of these problems on restricted classes of matroids. We note the existence of Fully PTASs for MOL problems on some restricted matroid classes, e.g., BM on a laminar matroid or KCM on a partition matroid. The question whether (non-trivial) MOL problems admit

Fully PTAS on broader matroid classes, such as graphical matroids or linear matroids, remains open. In particular, it would be interesting to obtain a Fully PTAS for CONSTRAINED MINIMUM SPANNING TREE [26] and BM on a linear matroid − or show that one does not exist.

References

1 Kim Allan Andersen, Kurt Jörnsten, and Mikael Lind. On bicriterion minimal spanning trees: An approximation. *Computers & Operations Research*, 23(12):1171–1182, 1996.

2 André Berger, Vincenzo Bonifaci, Fabrizio Grandoni, and Guido Schäfer. Budgeted matching and budgeted matroid intersection via the gasoline puzzle. *Mathematical Programming*, 128(1):355–372, 2011.

3 Karl Bringmann and Vasileios Nakos. A fine-grained perspective on approximating subset sum and partition. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1797–1815. SIAM, 2021.

4 Gruia Calinescu, Chandra Chekuri, Martin Pal, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.

5 Paolo M. Camerini, Giulia Galbiati, and Francesco Maffioli. Random pseudo-polynomial algorithms for exact matroid problems. *Journal of Algorithms*, 13(2):258–273, 1992.

6 Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM (JACM)*, 51(4):557–594, 2004.

7 Alberto Caprara, Hans Kellerer, Ulrich Pferschy, and David Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123(2):333–345, 2000.

8 Venkatesan T Chakaravarthy, Anamitra Roy Choudhury, Sivaramakrishnan R Natarajan, and Sambuddha Roy. Knapsack cover subject to a matroid constraint. In *Proc. FSTTCS*, 2013.

9 Richard Chang, Benny Chor, Oded Goldreich, Juris Hartmanis, Johan Håstad, Desh Ranjan, and Pankaj Rohatgi. The random oracle hypothesis is false. *Journal of Computer and System Sciences*, 49(1):24–39, 1994.

10 Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005.

11 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 575–584. IEEE, 2010.

12 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Multi-budgeted matchings and matroid intersection via dependent rounding. In *Proc. SODA*, pages 1080–1097, 2011.

13 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

14 Marek Cygan, Marcin Mucha, Karol Wegrzycki, and MichaI WIodarczyk. On problems equivalent to (min,+)-convolution. *ACM Transactions on Algorithms (TALG)*, 15(1):1–25, 2019.

15 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. An AFPTAS for bin packing with partition matroid via a new method for LP rounding. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, 2023.

16 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. Budgeted matroid maximization: A parameterized viewpoint. *IPEC*, 2023.

17 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. An EPTAS for budgeted matching and budgeted matroid intersection via representative sets. In *Proc. ICALP*, pages 49:1–49:16, 2023.

18 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. An EPTAS for budgeted matroid independent set. In *Proc. SOSA*, pages 69–83, 2023.

19 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. An FPTAS for budgeted laminar matroid independent set. *Operations Research Letters*, 51:632–637, 2023.

20 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. Lower bounds for matroid optimization problems with a linear constraint. *arXiv preprint*, 2023. `arXiv:2307.07773`.

21 Jack Edmonds. Minimum partition of a matroid into independent subsets. *J. Res. Nat. Bur. Standards Sect. B*, 69:67–72, 1965.

22 Fedor V Fomin, Petr A Golovach, Tuukka Korhonen, Kirill Simonov, and Giannos Stamoulis. Fixed-parameter tractability of maximum colored path and beyond. In *Proc. SODA*, pages 3700–3712, 2023.

23 András Frank. *Connections in combinatorial optimization*, volume 38. Oxford University Press Oxford, 2011.

24 Kilian Grage, Klaus Jansen, and Kim-Manuel Klein. An EPTAS for machine scheduling with bag-constraints. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, pages 135–144, 2019.

25 Fabrizio Grandoni and Rico Zenklusen. Approximation schemes for multi-budgeted independence systems. In *Proc. ESA*, pages 536–548, 2010.

26 Refael Hassin and Asaf Levin. An efficient polynomial time approximation scheme for the constrained minimum spanning tree problem using matroid intersection. *SIAM Journal on Computing*, 33(2):261–268, 2004.

27 Sung-Pil Hong, Sung-Jin Chung, and Bum Hwan Park. A fully polynomial bicriteria approximation scheme for the constrained spanning tree problem. *Operations Research Letters*, 32(3):233–239, 2004.

28 Per M Jensen and Bernhard Korte. Complexity of matroid property algorithms. *SIAM Journal on Computing*, 11(1):184–190, 1982.

29 Richard M Karp, Eli Upfal, and Avi Wigderson. The complexity of parallel computation on matroids. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 541–550. IEEE, 1985.

30 Ariel Kulik and Hadas Shachnai. There is no EPTAS for two-dimensional knapsack. *Information Processing Letters*, 110(16):707–710, 2010.

31 Eugene L Lawler. Matroid intersection algorithms. *Mathematical programming*, 9(1):31–56, 1975.

32 Eugene L. Lawler. Fast approximation algorithms for knapsack problems. *Math. Oper. Res.*, 4(4):339–356, 1979.

33 László Lovász. The matroid matching problem. *Algebraic methods in graph theory*, 2:495–517, 1978.

34 James G Oxley. *Matroid theory*, volume 3. Oxford University Press, USA, 2006.

35 Rudi Pendavingh and Jorn Van der Pol. On the number of matroids compared to the number of sparse paving matroids. *arXiv preprint*, 2014. `arXiv:1411.0935`.

36 Ram Ravi and Michel X Goemans. The constrained minimum spanning tree problem. In *Algorithm Theory – SWAT'96: 5th Scandinavian Workshop on Algorithm Theory Reykjavík, Iceland, July 3–5, 1996 Proceedings 5*, pages 66–75. Springer, 1996.

37 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

38 Prabhakant Sinha and Andris A Zoltners. The multiple-choice knapsack problem. *Operations Research*, 27(3):503–515, 1979.

39 José A Soto. A simple PTAS for weighted matroid matching on strongly base orderable matroids. *Discrete Applied Mathematics*, 164:406–412, 2014.

40 Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004.

Non-Linear Paging

Ilan Doron-Arad
Computer Science Department, Technion, Haifa, Israel

Joseph (Seffi) Naor
Computer Science Department, Technion, Haifa, Israel

Abstract

We formulate and study *non-linear paging* - a broad model of online paging where the size of subsets of pages is determined by a monotone non-linear set function of the pages. This model captures the well-studied classic weighted paging and generalized paging problems, and also *submodular* and *supermodular* paging, studied here for the first time, that have a range of applications from virtual memory to machine learning.

Unlike classic paging, the cache threshold parameter k does not yield good competitive ratios for non-linear paging. Instead, we introduce a novel parameter ℓ that generalizes the notion of cache size to the non-linear setting. We obtain a tight deterministic ℓ-competitive algorithm for general non-linear paging and a $o\left(\log^2(\ell)\right)$-competitive lower bound for randomized algorithms. Our algorithm is based on a new generic LP for the problem that captures both submodular and supermodular paging, in contrast to LPs used for submodular cover settings. We finally focus on the supermodular paging problem, which is a variant of online set cover and online submodular cover, where sets are repeatedly requested to be removed from the cover. We obtain polylogarithmic lower and upper bounds and an offline approximation algorithm.

2012 ACM Subject Classification Theory of computation

Keywords and phrases paging, competitive analysis, non-linear paging, submodular and supermodular functions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.57

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2404.13334` [14]

Funding *Joseph (Seffi) Naor*: Supported in part by Israel Science Foundation grant 2233/19 and United States – Israel Binational Science Foundation (BSF) grant 2033185.

1 Introduction

In the well studied *paging* problem, we are given a collection of n pages and a cache that can contain up to k pages simultaneously, where $k < n$. At each time step, one of the pages is requested. If the requested page is already in the cache, the request is immediately served. Otherwise, there is a *cache miss* and the requested page is fetched to the cache; to ensure that the cache contains at most k pages, some other page is potentially evicted. In the most fundamental model, the goal is to minimize the number of cache misses (or equivalently, number of evictions).

In more general models, pages may have different sizes and costs (see, e.g., [1, 5]) and then the sum of the sizes of the pages in cache cannot exceed its capacity. However, a linear function over page sizes that defines cache feasibility fails to capture scenarios with more involved relations between subsets of pages that can reside together in cache. Consider a system in which pages share parts of their memory and then only the missing memory parts of a requested page can contribute to the increase in cache size. This setting can be modeled using a *submodular* function that defines cache feasibility. Another example is a setting in


which items stored in cache have dependencies yielding additional overhead in their mutual storage demand. The *rule caching* problem is one such setting and it has been well studied in networking [12, 46, 37, 21, 25, 39, 24, 17, 9, 35, 26, 16, 45, 34, 33, 47, 13].

Our focus in this paper will be on settings where such non-linear behavior exists. We will further motivate our model in detail in Section 1.2.

1.1 Our Model

Before presenting our model, we give some required preliminary definitions. Let $\mathcal{P}$ be a set and let $f : 2^{\mathcal{P}} \to \mathbb{N}$ be a set function of $\mathcal{P}$. The function f is called *monotone* if for every $S \subseteq \mathcal{P}$ and $S' \subseteq S$ it holds that $f(S') \leq S$. In addition, f is called *submodular* if for every $S, S' \subseteq \mathcal{P}$ it holds that $f(S) + f(S') \geq f(S \cup S') + f(S \cap S')$. Conversely, f is called *supermodular* if for every $S, S' \subseteq \mathcal{P}$ it holds that $f(S) + f(S') \leq f(S \cup S') + f(S \cap S')$.

In this work, we introduce a very general model of paging with an arbitrary function defining cache feasibility. In the *non-linear paging* problem, we are given a collection $\mathcal{P}$ of n pages where each page $p \in \mathcal{P}$ has a fixed eviction cost $c(p)$. We are also given a monotone *feasibility* function $f : 2^{\mathcal{P}} \to \mathbb{N}$ that assigns a value to every subset of pages, indicating their size. Finally, we are given a cache threshold k. As in standard paging, in each time step t there is a request p_t for one of the pages. If p_t is already in cache, the request is immediately served. Otherwise, p_t is fetched to the cache and possibly some subset of pages is evicted to ensure that the set of pages in the cache, denoted by S_t, is feasible, i.e., $f(S_t) \leq k$. The goal is to minimize the total cost incurred from page evictions. The classic paging problem is obtained by setting $f(S) = |S|$ for all $S \subseteq \mathcal{P}$. Other interesting applications of our model are described below.

- *Generalized Paging* [5, 1]. Here the feasibility function is linear; that is, for every $S \subseteq \mathcal{P}$ it holds that $f(S) = \sum_{p \in S} f(p)$, where $f(p)$ is the size of page $p \in \mathcal{P}$.
- *Submodular Paging.* The feasibility function is *submodular*. A natural application of this variant is to settings where pages share memory items (see Section 1.2).
- *Supermodular Paging.* The feasibility function is *supermodular*, implying a submodular cover function for pages remaining out of the cache. This setting effectively captures online submodular covering problems [2, 23, 20]. Supermodular paging will be the main focus of our paper.

Supermodular paging is a variant of *online set cover* [2] and *online submodular cover* [20]. In online set cover, we are given a ground set X and a family $\mathcal{S}$ of subsets of X. Requests for elements of X arrive online; if a requested element is not already covered by a previously chosen set, a set $S \in \mathcal{S}$ containing it is chosen, paying a cost $c(S)$. The goal is to minimize the cost of the selected sets. Online submodular cover generalizes online set cover - the goal is to cover a general (monotone) submodular function with an increasing cover demand over time. In supermodular paging (submodular cover), the cover demand does not change over time, as the same page (a set $S \in \mathcal{S}$ in online set cover) may be requested (removed from the cover) multiple times. We show in the full version of the paper (see [14]) that supermodular paging is even more challenging than (online) submodular cover.

1.2 Motivation

Non-linear paging generalizes several fundamental caching problems, capturing many real world applications. Besides known applications of the classic paging models with linear feasibility functions [38, 15, 4, 5, 1], there are many scenarios in which the interaction between

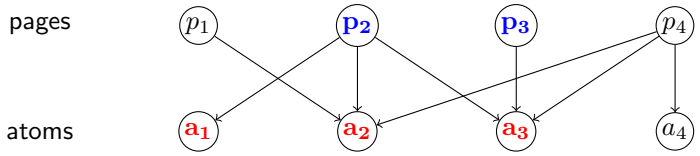


Figure 1 An illustration of a shared memory systems with four pages and four shared memory units ("atoms"). The cache is of size $k = 3$ and contains pages p_2, p_3 (in blue) whose shared memory is of size 3 - the three red atoms. Observe that either fetching p_1 or evicting it will not change the number of atoms stored in cache, however, fetching p_4 requires evicting p_2.

pages is non-linear, requiring the more general non-linear paging model. As already indicated, supermodular paging roughly generalizes online set cover and therefore has both theoretical and practical importance [20, 2]. We describe below several interesting applications of non-linear paging.

A major motivation for the study of non-linear paging is caching in shared memory systems (e.g., [28, 7, 43, 40]). Each process in a multi-process memory system is associated with a *virtual memory* [11], providing the illusion that it has a much larger memory. In some shared memory systems, caching policies are defined over entire processes, that is, the entire virtual address space of a process can be taken to the cache. Since the virtual memory of two processes typically overlaps in physical memory, the increase in (physical) memory in cache is larger if the cache is empty, as the entire memory of a process is loaded to the cache. In contrast, if the cache is nearly full, and a process is loaded to the cache, the increase in total size is smaller, as most of the virtual memory addresses are already in cache. In a similar vein, when caching hot data at the network edge, to avoid serving requests from a remote cloud, space efficiency of similar files is achieved through *deduplication* (e.g. [27]), which is very similar to the way overlaps are handled in virtual memory.

Thus, caching in shared memory systems is a special case of submodular paging. More formally, consider a cache that can store up to k *atoms* from a larger set $A = \{a_1, \ldots, a_n\}$ spanning the physical memory. Each page (process) p contains a subset of atoms $a(p) \subseteq A$ corresponding to the physical memory to which process p is mapped. The feasibility function f ensures that a collection S of pages contains jointly at most k atoms. Thus, $f(S) = \left|\bigcup_{p \in S} a(p)\right|$, and it is a submodular function. See Figure 1 for an illustration.

Paging with a supermodular feasibility function arises in common settings where the storage demand grows rapidly as a function of the number of "pages" stored in cache. In these scenarios, there is a large set of n *entities* (corresponding to vertices), and among subsets of entities certain interactions exist (represented by hyperedges). This data structure, known as a hypergraph, is ubiquitous in applications such as recommendation systems [19, 41] (where vertices represent individuals and hyperedges represent communities), image retrieval [29](vertices represent images and hyperedges represent correlations), and bioinformatics [32] (vertices represent substances and hyperedges stand for biochemical interactions). Other applications arise in machine learning [42, 18, 49] and databases [6].

In various practical settings, the hypergraph is very large (e.g., [36, 31, 22]). Therefore, a natural approach is to store frequently accessed vertices in a cache. However, caching is effective only if all interactions (i.e., hyperedges) among subsets of vertices in cache are also stored therein. As a set of x vertices may have up to 2^x induced hyperedges, the storage demand for hyperedges tends to be significantly larger compared to the number of vertices. Caching hypergraphs in the non-linear paging framework can be formally defined as follows:

consider a set $\mathcal{P}$ of vertices and define a feasibility function f over $\mathcal{P}$, such that for a subset of vertices $S \subseteq \mathcal{P}$, the storage demand $f(S)$ is the number of induced hyperedges in S plus the cardinality of S. Function f is a supermodular function.

For example, a real-world problem related to supermodular paging is caching in device-to-device (D2D) communication networks, with social ties among users and common interests that are used as key factors in determining the caching policy and are modeled via a hypergraph [3]. Here, the number of hyperedges (describing roughly interferences among users of the network, content, transmission rate, etc.) grows in a supermodular manner with respect to the number of users placed in cache. In addition, our reduction from online set cover (similarly, online submodular cover) to supermodular paging implies that applications of online set cover are also applications of supermodular paging.

1.3 Our Results and Techniques

We now present our results and elaborate on the techniques used. We start with general non-linear paging and then proceed to the special case of supermodular paging.

1.3.1 General Non-Linear Paging

In classic paging models, competitive ratios are typically given as a function of k, the cache size. However, non-linear paging is more difficult. Even for non-linear paging instances with $k = 0$ the competitive ratio can be very high. For example, consider a (classic) paging instance I' with cache threshold k'; define a non-linear paging instance I with a (non-linear) feasibility function f for which $f(S) = 0$ if S is feasible for I' and $f(S) = 1$ otherwise. In addition, we set $k = 0$ as the cache threshold of I. Clearly, a solution for I implies a solution for I'. Hence, by the hardness of paging [38, 15] the best competitive ratio of non-linear paging is arbitrarily large as a function of k (we give the remaining details in [14]). Thus, instead of k, we look for a parameter that better captures the competitiveness of general non-linear paging. This parameter turns out to be the maximum cardinality of a *minimally infeasible* set, i.e., an infeasible set where every proper subset of it is feasible[1]. Formally,

▶ **Definition 1.** *A set $S \subseteq \mathcal{P}$ is called* feasible *if $f(S) \leq k$ and is* infeasible *otherwise. Additionally, S is* minimally infeasible *if S is infeasible and every $S' \subset S$ is feasible, and let $\mathcal{M} = \{S \subseteq \mathcal{P} \mid f(S) > k \text{ and } f(S') \leq k \ \forall S' \subset S\}$ be all minimally infeasible sets. Finally, let the* width *of f be*

$$\ell(f) = \max_{S \in \mathcal{M}} (|S| - 1) .$$

We simply let $\ell = \ell(f)$ when it is clear from context. Clearly, for paging (or weighted paging) the width ℓ equals k. Hence, the width accurately captures the optimal performance of paging algorithms: there is a tight ℓ-competitive deterministic algorithm [38] and a tight $\Theta(\log(\ell))$-competitive randomized algorithm [15]. However, the width behaves quite differently in other scenarios. For example, in the setting of submodular paging described in Section 1.2, the instance can have a fixed width $\ell = O(1)$, but the number of pages in cache can be unbounded (e.g., all pages use the same atom). Interestingly, we show that the width gives a tight competitive ratio also for general non-linear paging via a new LP for the problem (see Sections 2 and 2.1).

[1] Technically, we subtract one so that the definition coincides with the parameter k in paging.

▶ **Theorem 2.** *There is a deterministic ℓ-competitive algorithm for* non-linear paging. *Moreover, every deterministic algorithm for* non-linear paging *is at least ℓ-competitive.*

The algorithm achieving Theorem 2 is based on a new LP relaxation, designed for the general setting of non-linear paging. We now explain why previous techniques for classic paging are not sufficient for obtaining a competitive online algorithm for this setting.

Previous work on generalized paging [5, 1] and on other online covering problems [20, 10] use *knapsack cover* constraints. This powerful technique, originated by Wolsey [44], is very useful for relaxing submodular cover constraints using linear inequalities. Indeed, paging problems are often studied from the viewpoint of a *covering* problem (e.g., [4]), where the complement of the cache (i.e., pages outside the cache) needs to be covered. In classic paging, at any point of time at least $n - k$ pages are not in the cache.

Consider the *covering* function $g : 2^{\mathcal{P}} \to \mathbb{N}$ of a feasibility function f defined to be the (non-linear) size requirement *outside* of the cache: $g(S) = f(\mathcal{P}) - f(\mathcal{P} \setminus S)$ for every $S \subseteq \mathcal{P}$. Observe that for classic paging it holds that $g(S) = n - (n - |S|) = |S|$ and the feasibility constraint translates to $g(S) \geq n - k$ at all times. If f is submodular (i.e., submodular paging) then g is supermodular, and vice versa. Knapsack cover constraints yield a relaxation of the covering problem when the cover function g is submodular [20], as is the case in classic paging problems.

However, for submodular paging, the covering function g is supermodular and knapsack cover constraints do not even provide a relaxation of the problem. For example, let $g(S) = 1$ for $S = \mathcal{P}$ and $g(S) = 0$ otherwise. Then, the knapsack constraints are not satisfied by the unique solution that covers a demand of $k = 1$ (the entire set $\mathcal{P}$). Specifically, $\mathcal{P}$ does not satisfy the knapsack constraint for $S = \emptyset$, i.e., $\sum_{p \in \mathcal{P}} x_p \cdot g_{\emptyset}(\{p\}) = 0$, but $g_{\emptyset}(\mathcal{P}) = 1$.

To circumvent the limits of knapsack cover constraints for submodular paging, we formulate a new set of covering constraints that are valid for any feasibility functions f and g. Specifically, the constraints require removing at least one page from every infeasible set. Then, using the online primal-dual approach applied to this set of constraints, we obtain a tight ℓ-competitive deterministic algorithm for non-linear paging. Specifically, upon arrival of a page that induces an infeasible set of pages in cache, our algorithm identifies a minimally infeasible set of pages and continuously increases their corresponding dual variable in the LP, evicting *tight* pages.

Interestingly, as a special case, Theorem 2 gives a simple k-competitive deterministic algorithm for generalized paging; to the best of our knowledge, there are only $(k + 1)$-competitive deterministic algorithms [8, 48] for generalized paging. Thus, our bound is tight for this problem.

▶ **Corollary 3.** *There is a deterministic k-competitive algorithm for* generalized paging.

We emphasize that the lower bound of Theorem 2 can be obtained for *any* function f with a minimally infeasible set of cardinality ℓ (regardless of whether f is linear, submodular, supermodular, or any other function). This shows the robustness of the parameter ℓ as an indicator for the competitiveness of non-linear paging. Thus, it is natural to ask whether the parameter ℓ for non-linear paging is analogous to the parameter k for classic paging when allowing randomization. We answer this question in the negative by showing that in contrast to paging, that admits an $O(\log(\ell))$-competitive randomized algorithm [15], non-linear paging is substantially harder w.r.t. the parameter ℓ.

▶ **Theorem 4.** *Unless* NP $\subseteq$ BPP*, there is no polynomial-time randomized $o(\log^2(\ell))$-competitive algorithm for* non-linear paging.

A very intriguing open question is whether there exists a randomized polylog(ℓ)-competitive algorithm for general non-linear paging. Unfortunately, we show that the LP used for obtaining Theorem 2 has an integrality gap of ℓ, and thus would need to be strengthened to achieve this end (see Section 2.3). Hence, obtaining a polylog(ℓ) competitive factor would require a new set of techniques. As we have already discussed earlier, existing techniques for obtaining randomized online algorithms for paging problems and related variants focus on solving covering linear programs (LP) online, and they break down in the presence of general non-linear paging constraints.

To overcome the integrality gap of our LP, we formulate a stronger LP for non-linear paging (see Section 2.3). Obtaining even a fractional polylog(ℓ)-competitive algorithm for this LP seems a hard task. Thus, we consider another parameter which allows us to get a better competitive ratio. The parameter is the maximum number of pages that fit together in the cache. Formally, define

$$\mu = \max_{S \subseteq \mathcal{P} \text{ s.t. } f(S) \leq k} |S| \tag{1}$$

as the maximum cardinality of a feasible set in cache. Observe that $\mu = k$ for e.g., generalized paging, hence it is a natural parameter to also consider in our setting. We also remark that in many practical settings, such as classic paging, it holds that $n \gg \mu$ and obtaining a competitive ratio that depends on μ (rather than n) is much more desirable.

Clearly, $\mu \geq \ell$. The following example illustrates a scenario demonstrating a scenario where $\mu \gg \ell$. Consider a non-linear paging instance on a set P of pages which is partitioned into disjoint sets X, Y, where $|X| = n$, $|Y| = k+1$, and $n \gg k$. Define a feasibility function f such that for all subsets S of P, $f(S) = |Y \cap S|$. Define the cache threshold as k. Therefore, the only minimally infeasible set is Y; thus, $\ell = k$. On the other hand, the maximum cardinality of a set that fits into cache is the cardinality of all pages in X and any k pages from Y; thus, $\mu = n + k$. Since $n \gg k$ (i.e., n can be chosen to be arbitrarily large with respect to k) it follows that $\mu \gg \ell$.

We give the following fractional algorithm for solving the strengthened LP online.

▶ **Theorem 5.** *There is an $O(\log \mu)$-competitive algorithm for obtaining a fractional solution for the strengthened* LP*.*

It as an interesting open question if a randomized polylog(μ)-competitive algorithm for non-linear paging can be designed by rounding the fractional solution obtained in Theorem 5.

1.3.2 Supermodular Paging

We now discuss our results and techniques for supermodular paging. Our main result is a polylogarithmic randomized competitive algorithm for supermodular paging, i.e., submodular cover paging. As we show later on, our upper bound turns out to be quite close to the lower bound we prove.

▶ **Theorem 6.** *There is an $O\left(\log^2 \mu \cdot \log\left(\frac{c_{\max}}{c_{\min}} \cdot f(\mathcal{P})\right)\right)$-competitive randomized algorithm for* supermodular paging (submodular cover paging)*, where $c_{\max}, c_{\min}$ are the maximum and minimum costs of pages, respectively.*

In particular, for unweighted supermodular paging (where $c_p = 1 \; \forall p \in \mathcal{P}$), the above theorem implies an $O\left(\log^2(\mu) \cdot \log\left(f(\mathcal{P})\right)\right)$-competitive algorithm.

Our algorithm relies on a different LP relaxation than the one described in Section 1.3.1. As we aim to solve supermodular paging, which implies a submodular cover function g, we design an LP relaxation inspired by the submodular cover relaxation of Wolsey [44] (see

also [20]). We solve this LP in the case that the constraints arrive online to obtain a fractional $O(\log(\mu))$-competitive (deterministic) algorithm, while maintaining the property that the entries are either integral or multiples of $\frac{1}{k}$.

To obtain an integral solution, one is tempted to maintain online a set of feasible *cache states* respecting (even approximately) the marginal probabilities induced by the fractional solution, similarly to previous works on weighted paging and generalized paging [4, 5, 1]. However, techniques for online cache state maintenance of [4, 5, 1] seem to break down in the presence of submodular cover constraints. Instead, we augment the fractional solution by an additional polylogarithmic *boosting factor* and perform randomized rounding, with possible corrections to ensure feasibility. The probability of a page to be evicted at some point in time is shown to be proportional to the page's fractional increase normalized by the probability that the page is in cache.

We also give lower bounds for online supermodular paging. Surprisingly, even though online set cover seems starkly different from supermodular paging, in particular since the cover constraints change over time for online set cover, we can show that supermodular paging is in a sense harder to solve online.

▶ **Theorem 7.** *For any $\rho \geq 1$, if there is a ρ-competitive algorithm for* supermodular paging*, then there is a ρ-competitive algorithm for* online set cover *having the same running time up to polynomial factors.*

By Theorem 7 and the results of [2, 23], we give a lower bound on the competitiveness of supermodular paging, indicating the necessity of the factor $\log \mu \cdot \log(\mathcal{P})$.

▶ **Corollary 8.** *Unless* NP $\subseteq$ BPP*, there is no polynomial $o(\log \mu \cdot \log(f(\mathcal{P})))$-competitive algorithm for* supermodular paging*. Moreover, there is no deterministic $o\left(\frac{\log \mu \cdot \log(f(\mathcal{P}))}{\log\log \mu + \log\log(f(\mathcal{P}))}\right)$-competitive algorithm for* supermodular paging *of any running time.*

We remark that some of our results from Section 1.3.1 can be stated using the parameter μ as well. However, as we showed earlier, there are non-linear paging instances in which the parameter $\mu \gg \ell$ and it grows artificially apart from the true hardness of the instance – in terms of competitive analysis. In contrast, every minimally infeasible set of cardinality $\ell + 1$ can be used to obtain the lower bounds for classic paging [38, 15] (i.e., ℓ- deterministic and $O(\log \ell)$-randomized lower bounds). Thus, in an informal sense, an increase in ℓ always incurs an increase in the difficulty of the problem, unlike an increase in μ. For this reason, we believe that searching for competitive algorithms and lower bounds in terms of ℓ, rather than μ, may be of greater interest in the non-linear paging setting.

Finally, we also aim at obtaining an offline approximation algorithms for supermodular paging (i.e., where all requests are known in advance). A first attempt would be to reduce the problem to offline submodular cover, as follows. Define a covering function G on the domain containing all pairs (p, j), for every page p and the j-th time it is requested. Define the value of G on a subset of pairs S as $G(S) = \sum_{t \in T} g(S_t)$, where g is the covering function of the instance (see Section 1.3.1), and S_t is the set of all pages p, where (p, j) belongs to S and the time interval between requests j and $(j + 1)$ for p intersects t. Clearly, the total cover demand, even with a unit demand per-time slot, is a function of T; thus, applying an offline set cover algorithm in a black box manner gives only an $\Omega(\log(T))$-approximation [30]. As T may be very large, a different technique is in place.

Combined with the strong *round-or-separate* algorithm of [20], our techniques yield an approximation algorithm for supermodular paging independently of T (see Section 3). We defer the details to the full version of the paper.

▶ **Theorem 9.** *There is an offline* $O\left(\log\left(\frac{c_{\max}}{c_{\min}} \cdot f(\mathcal{P})\right)\right)$*-approximation algorithm for* supermodular paging.

Due to space constraints, some of the proofs (including our lower bounds) are given only in the full version of the paper [14].

2 Deterministic Algorithm for Non-Linear Paging

In this section, we present a deterministic algorithm for non-linear paging and show that it gives a tight competitive ratio for the problem, thus providing proofs for Theorem 2 and Corollary 3. Our algorithmic results are based on a new LP relaxation for the problem which we introduce here.

2.1 LP Relaxation for Non-Linear Paging

Consider a non-linear paging instance with a set of pages $\mathcal{P}$, a feasibility function f, cache threshold k, a set of time points T, and a page request $p_t \in \mathcal{P}$ for every time $t \in T$. To simplify notation, for a set S and an element p we use $S + p = S \cup \{p\}$ and $S - p = S \setminus \{p\}$. The LP is as follows.

The variables of the LP are $x_p(j)$ for every page $p \in \mathcal{P}$ and the j-th time that p is requested. Intuitively, the value of the variable $x_p(j)$ indicates to what extent page p is evicted between its j-th and $(j+1)$-th requests. The constraints state that for every infeasible set S containing the requested page p_t, for time point t, we must "break" this set - i.e., evicting at least one page from $S - p_t$. If this constraint is satisfied for every such infeasible set by an integral solution, then it induces a feasible set of pages in the cache at all times. See Figure 2 for a visualization of these constraints.

We use n_p to denote the number of requests for page p during the request sequence. Also, we use $r(p,t)$ to denote the number of requests for page p until time t. We assume that $x_p(0)$ is a variable always set to 0, for all $p \in \mathcal{P}$. To simplify notation, for every $t \in T$, let $\mathcal{S}(t) = \{S \subseteq \mathcal{P} \mid p_t \in S \text{ and } f(S) > k\}$ denote the infeasible sets containing p_t. For any $w \in \mathbb{N}$, let $[w] = \{1, 2, \ldots, w\}$. Our LP relaxation is as follows.

$$\begin{aligned} \min \quad & \sum_{p \in \mathcal{P}} \sum_{j \in [n_p]} x_p(j) \cdot c(p) \\ & \text{s.t.} \\ & \sum_{p \in S - p_t} x_p(r(p,t)) \geq 1, && \forall t \in T \ \forall S \in \mathcal{S}(t) \\ & x_p(j) \geq 0 && \forall p \in \mathcal{P} \ \forall j \in [n_p] \end{aligned} \tag{2}$$

We now define the dual LP. For page $p \in \mathcal{P}$ and $j \in [n_p]$ let $I(p,j) = \{t \in T \mid j = r(p,t)\}$ be the *interval* of all time points between the j-th request to p till the last time point before the $(j+1)$-st request for page p. The *dual* of (2) is the following.

$$\begin{aligned} \max \quad & \sum_{t \in T} \sum_{S \in \mathcal{S}(t)} y_t(S) \\ & \text{s.t.} \\ & \sum_{t \in I(p,j)} \sum_{S \in \mathcal{S}(t) \mid p \in S - p_t} y_t(S) \leq c(p), && \forall p \in \mathcal{P} \ \forall j \in [n_p] \end{aligned} \tag{3}$$

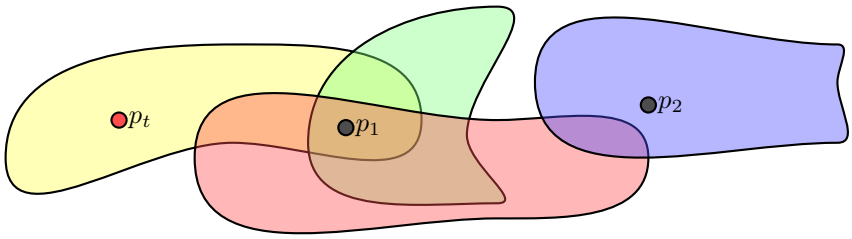


Figure 2 An illustration of the LP constraints. Each colored shape represents a minimally infeasible set of pages upon the arrival of a page p_t. Removing pages p_1 and p_2 ensures cache feasibility.

We use Primal-LP and Dual-LP to denote the values of the optimal solutions for the primal and dual programs, respectively, and by OPT the offline (integral) optimum. The next result follows from weak duality and since our LP is a relaxation of non-linear paging.

▶ **Lemma 10.** Dual-LP $\leq$ Primal-LP $\leq$ OPT.

Proof. The first inequality follows from weak duality, since (3) is the dual program of (2). For the second inequality, we show that (2) is a relaxation of non-linear paging. Consider an integral feasible solution M to our instance; define a solution x for (2) such that $x_p(j) = 1$ if and only if page p is removed from the cache during the interval $I(p, j)$ in M, for every $p \in \mathcal{P}$ and $j \in [n_p]$. Then, for every time $t \in T$ and $S \in \mathcal{S}(t)$, S cannot be fully contained in the cache at time t, since M is a feasible solution, and there is at least one $p \in S - p_t$ which is not in the cache at this time. Therefore, the primal constraint corresponding to t and S is satisfied. We conclude that x is feasible for (2); consequently, (2) is indeed a relaxation of non-linear paging, implying the second inequality. ◀

2.2 A Deterministic Algorithm for Non-Linear Paging

In this section, we give a primal-dual algorithm, based on the LP relaxation of non-linear paging presented in Section 2.1. For brevity, we denote the left handside of the dual constraint corresponding to page p and $j \in [n_p]$ as

$$Y_p(j) = \sum_{t \in I(p,j)} \sum_{S \in \mathcal{S}(t) \mid p \in S - p_t} y_t(S). \tag{4}$$

We call a page p *tight* at time t if $Y_p(j) = c(p)$.

The algorithm initializes an infeasible primal solution and a feasible dual solution as vectors of zeros $\bar{0}$. In every time step t, if the set of pages currently in cache, denoted by Cache_t, is infeasible, then our algorithm finds a subset of pages Q in the cache such that: first, Q is infeasible; second, Q contains the requested page p_t, i.e., $Q \in \mathcal{S}(t)$; third, Q has minimum cardinality amongst all such sets. we increase the variable $y_t(Q)$ continuously. Once one of the pages becomes tight, we remove it from cache. If the cache is feasible, the algorithm proceeds to time $t + 1$; otherwise, we repeat this process with a new set Q' from the current cache. The pseudocode is given in Algorithm 1.

We start by showing the feasibility of the algorithm.

Algorithm 1 Deterministic.

```
Initialize (infeasible) primal and (feasible) dual solutions x ← 0̄ and y ← 0̄.
for time t ∈ T do
    Let Cache_t = {p ∈ P | r(p,t) ≥ 1 and x_p(r(p,t)) = 0} be the pages currently in cache.
    while f(Cache_t) > k do
        Find Q ⊆ Cache_t such that Q ∈ S(t) of minimum cardinality.
        while Y_p(r(p,t)) < c(p) ∀p ∈ Q − p_t do
            Increase y_t(Q) continously.
        end
        for p ∈ Q − p_t such that Y_p(r(p,t)) = c(p) do
            Remove p from cache: x_p(r(p,t)) ← 1,  Cache_t ← Cache_t − p.
        end
    end
end
Return the (primal) solution x and the (dual) solution y.
```

▶ **Lemma 11.** *Algorithm 1 returns primal and dual feasible solutions of* (2).

Proof. By Algorithm 1, the algorithm keeps evicting pages until reaching a feasible set of pages in the cache. By the monotonicity of the feasibility function f, we eventually reach a feasible set in cache: every page p is considered to be feasible alone in the cache, i.e., $f(\{p\}) \leq k$; thus, in the worst case, the content of the cache at the end of time step t is p_t. The above shows that the primal solution x is feasible, being a relaxation of the integer problem. In addition, the dual y is feasible since for all $p \in \mathcal{P}$ and $j \in [n_p]$ it holds that $Y_p(j) \leq c(p)$ by Algorithm 1. ◀

In the following we bound the competitive ratio of the algorithm. Let $c(x)$ be the cost of our (integral) solution x and let $v(y)$ be the value of the dual solution y obtained by Algorithm 1. Using the selection of minimal sets for Q in Algorithm 1 we have the following result. Recall the width parameter ℓ defined in Definition 1.

▶ **Lemma 12.** $c(x) \leq \ell \cdot v(y)$

Proof. By LP (2),

$$\begin{aligned} c(x) &= \sum_{p\in\mathcal{P}} \sum_{j\in[n_p]} x_p(j)\cdot c(p) \leq \sum_{p\in\mathcal{P}} \sum_{j\in[n_p]} x_p(j)\cdot Y_p(j) \\ &= \sum_{p\in\mathcal{P}} \sum_{j\in[n_p]} x_p(j)\cdot \left(\sum_{t\in I(p,j)} \sum_{S\in\mathcal{S}(t)\,\middle|\, p\in S-p_t} y_t(S) \right). \end{aligned} \tag{5}$$

The inequality holds since we only evict tight pages. By changing summation order in (5),

$$c(x) \leq \sum_{t\in T} \sum_{S\in\mathcal{S}(t)} y_t(S)\cdot \sum_{p\in S-p_t} x_p(r(p,t)) \leq \sum_{t\in T} \sum_{S\in\mathcal{S}(t)} y_t(S)\cdot \ell = \ell\cdot v(y).$$

The second inequality holds since in Algorithm 1 we always choose a minimally infeasible set of cardinality at most $\ell+1$; hence, $y_t(S) \neq 0$ only if $|S| \leq \ell+1$. ◀

We are now ready to give the proof of our main theorem.

Proof of Theorem 2. By Lemma 11, the returned solution x is a feasible solution. Combined with Lemma 10 and Lemma 12, $c(x) \leq \ell \cdot v(y) \leq \ell \cdot \text{OPT}$. Along with the tight lower bound from the special case of paging [38], the statement of the theorem follows.

We remark that for any ℓ and function f such that $\ell(f) = \ell$, there is a minimally infeasible set S of cardinality ℓ; as a result, the lower bound for deterministic algorithms of paging [38] can be obtained from the set S. Namely, given a deterministic algorithm $\mathcal{A}$ for unweighted non-linear paging, construct a sequence of requests for pages in S that always requests the page missing in cache by algorithm $\mathcal{A}$; clearly, $\mathcal{A}$ pays a cost of one at each time step, since S is minimally infeasible. Conversely, the offline optimum would evict the page that is requested latest in the future, missing only once in ℓ requests, giving the lower bound of ℓ. ◀

2.3 Integrality Gap

In this section, we show the limitations of the LP. Specifically, we show that the integrality gap of the LP defined in (2) is at least ℓ; together with our algorithmic upper bound the integrality gap is exactly ℓ. We then discuss a stronger LP formulation based on (2).

▶ **Lemma 13.** *The integrality gap of* LP (2) *is* ℓ.

Proof. For some $n \geq 1$, consider an instance with n pages $\mathcal{P} = \{p_1, \ldots, p_n\}$ with uniform costs $c(p) = 1 \; \forall p \in \mathcal{P}$ and a uniform feasibility function f as in classic paging, i.e., $f(S) = |S|$ for all $S \subseteq \mathcal{P}$, and some cache capacity k. Thus, all minimally infeasible sets are of cardinality $k+1$ and it follows that $\ell(f) = k$. Define the sequence of requests $p_1, \ldots, p_n$. That is, each page is requested exactly once. Observe that each request results in a page fault. Hence, any integral solution, in particular the optimal integral solution, evicts at least $n - k$ pages. On the other hand, define a fractional solution such that $x_p(1) = \frac{1}{k}$ for all $p \in \mathcal{P}$, i.e., each page is evicted after its first request with fraction $\frac{1}{k}$ (note that each page is requested exactly once). Consider some infeasible set $S \subseteq \mathcal{P}$ such that $|S| > k$. It holds that

$$\sum_{p \in S} x_p(1) = |S| \cdot \frac{1}{k} \geq \frac{k+1}{k} \geq 1.$$

Thus, x satisfies the constraints of (2). The cost paid by the fractional solution x is $\frac{n}{k}$. Therefore, as n and k can be chosen arbitrarily, the integrality gap of the LP is at least

$$\lim_{n \to \infty} \frac{n-k}{\frac{n}{k}} = \lim_{n \to \infty} \left(k - \frac{k^2}{n} \right) = k = \ell(f).$$

Therefore, in general, the integrality gap cannot be smaller than ℓ. ◀

2.3.1 Discussion: a Stronger LP

The integrality gap example shows that LP (2) is not sufficient for obtaining a randomized polylog(ℓ)-competitive algorithm for general non-linear paging. Instead, we describe a stronger version of our LP (2), in which we require removing from each infeasible set S a set of pages S', so that the complement of S' in S, $S \setminus S'$, will be feasible in the cache (i.e., $f(S \setminus S') \leq k$). The strengthened LP and our fractional algorithm for solving the LP online are presented in [14], giving the proof of Theorem 5.

3 A Randomized Algorithm for Supermodular Paging

We provide here an $O(\log \mu)$-competitive algorithm for a fractional version of supermodular paging. Then, we design randomized online and offline algorithms for supermodular paging.

3.1 LP for Supermodular Paging

The starting point of our fractional algorithm is earlier work on submodular cover LP [44, 20, 10]. Consider a supermodular paging instance with a set of pages $\mathcal{P}$, a feasibility function f inducing the submodular cover function g (recall that $g(S) = f(\mathcal{P}) - f(\mathcal{P} \setminus S)$ for every $S \subseteq \mathcal{P}$), cache threshold k, a set of time points T, and a page request $p_t \in \mathcal{P}$ for every time $t \in T$. We use the following LP relaxation of supermodular paging. As in the LP introduced in Section 2.1, the variables of the LP are $x_p(j)$ for every page $p \in \mathcal{P}$ and the j-th time that p is requested. We will use the same notation as in the LP of (2). Recall that for some $S \subseteq P$ and $p \in \mathcal{P}$ we use $g_S(p) = g_S(\{p\}) = g(S+p) - g(S)$. Let $N = f(\mathcal{P}) - k$ be our *cover demand*; at any point of time, the (non-linear) total size outside of the cache must be at least N.

Our LP relaxation goes as follows; the constraints of the LP require that for every time t and subset of pages S (assumed to already be outside of the cache) we must evict from the cache (fractionally) a total size of at least $N - g(S) = f(\mathcal{P} \setminus S) - k$, since we cannot have more than a total size of k in cache. This constraint is very natural in the linear case (i.e., classic paging), but more involved in the submodular cover setting.

$$\begin{aligned} \min \sum_{p \in \mathcal{P}} \sum_{j \in [n_p]} & x_p(j) \cdot c(p) \\ \text{s.t.} & \\ \sum_{p \in \mathcal{P} - p_t} & x_p(r(p,t)) \cdot g_S(p) \geq N - g(S), && \forall t \in T \ \forall S \subseteq \mathcal{P} \\ & x_p(j) \geq 0 && \forall p \in \mathcal{P} \ \forall j \in [n_p] \end{aligned} \tag{6}$$

In the following we define the *dual* LP of (6).

$$\begin{aligned} \max \sum_{t \in T} \sum_{S \subseteq \mathcal{P}} & y_t(S) \cdot (N - g(S)) \\ \text{s.t.} & \\ \sum_{t \in I(p,j)} \sum_{S \subseteq \mathcal{P} \,\middle|\, p \in S - p_t} & y_t(S) \leq c(p), && \forall p \in \mathcal{P} \ \forall j \in [n_p] \\ & y_t(S) \geq 0 && \forall t \in T \ \forall S \subseteq \mathcal{P}. \end{aligned} \tag{7}$$

We use Primal-LP and Dual-LP to denote the values of the optimal solutions for the primal and dual programs, respectively, and by OPT the offline (integral) optimum.

Before we describe our fractional algorithm, we show that it is sufficient to satisfy only *minimal constraints* rather than all constraints of the LP. Formally, a primal constraint of (6) corresponding to $t \in T$ and $S \subseteq \mathcal{P}$ is called *minimal* for some solution x' if for all $p \in \mathcal{P}$ where $x'_p(r(p,t)) = 1$ it holds that $p \in S$.

▶ **Lemma 14.** *If x' satisfies all minimal constraints of (6), then x' is feasible for (6).*

Our fractional algorithm initializes the (infeasible) primal solution x and the (feasible) dual solution y both as vectors of zeros $\bar{0}$. When the requested page p_t at time t arrives, we do the following til x satisfies all LP constraints (6) up to time t.

At time t, the algorithm considers a *minimally violating set* of pages $Q \subseteq \mathcal{P}$. This set is a minimal set for our solution x violating the primal constraint in (6) corresponding to t and Q. We increase variable $y_t(\mathcal{P} \setminus Q)$ continuously and at the same time increase variables $x_p(r(p,t))$ for all pages in $(\mathcal{P} - p_t) \setminus Q$ except p_t. The increasing rate is a function of $Y_p(r(p,t))$, where

$$Y_p(j) = \sum_{t \in I(p,j)} \sum_{S \subseteq \mathcal{P} \mid p \in S - p_t} y_t(S) \tag{8}$$

is the left hand side of the corresponding dual constraint of p and $j = r(p,t)$ in (6) (analogously to (4)). This growth function has an exponential dependence on $Y_p(r(p,t))$ scaled by the cost of the page $c(p)$ and the number of pages possible in cache - the parameter μ. The growth of the variable $x_p(j)$ stops once it reaches $\frac{1}{2}$.

Algorithm 2 Fractional.

Initialize (infeasible) primal solutions $x, z \leftarrow \bar{0}$
Initialize (feasible) dual solution $y \leftarrow \bar{0}$.
for $t \in T$ **do**
  while x *is not feasible for* t **do**
    Find a minimal set $Q \subseteq \mathcal{P}$ for x such that $\sum_{p \in \mathcal{P} - p_t} x_p(r(p,t)) \cdot g_Q(p) < N - g(Q)$.
    while $\sum_{p \in \mathcal{P} - p_t} x_p(r(p,t)) \cdot g_Q(p) < N - g(Q)$ and Q is minimal for x **do**
      Increase $y_t(\mathcal{P} \setminus Q)$ continously.
      forall $p \in (\mathcal{P} - p_t) \setminus Q$ **do**
        increase $x_p(r(p,t))$ according to

$$x_p(r(p,t)) \leftarrow \frac{1}{\mu} \cdot \left(\exp\left(\frac{\ln(\mu+1)}{c(p)} \cdot Y_p(r(p,t)) \right) - 1 \right).$$

        if $x_p(r(p,t)) - \frac{z_p(r(p,t))}{2} \geq \frac{1}{4 \cdot N \cdot \mu}$ **then**
          $z_p(r(p,t)) \leftarrow 2 \cdot x_p(r(p,t))$.
        end
        if $x_p(r(p,t)) \geq \frac{1}{2}$ **then**
          $z_p(r(p,t)), x_p(r(p,t)) \leftarrow 1$.
        end
      end
    end
  end
end
Return the (primal) solution x and the (dual) solution y.

Once the primal constraint corresponding to t, Q is satisfied, or Q is no longer minimal for x (a page $p \in \mathcal{P} \setminus Q$ reaches 1), there are two cases. If x is feasible, the algorithm proceeds to the next time step. Otherwise, the algorithm repeats the above process with a new minimal violating set Q'. An additional property that will be useful in the analysis is that all non-zero entries of the obtained solution will be larger than $\Omega\left(\frac{1}{N \cdot \mu}\right)$ and there will be no fractional

entries larger than $\frac{1}{2}$. Thus, we update through the algorithm another primal solution z; we update an entry $z_p(j)$ to be the value of $2 \cdot x_p(j)$ whenever the difference $x_p(j) - z_p(j)$ becomes larger than $\Omega\left(\frac{1}{N \cdot \mu}\right)$. Moreover, we increase $z_p(j)$ immediately to 1 once $x_p(j)$ reaches $\frac{1}{2}$. The pseudocode of the algorithm is given in Algorithm 2.

3.2 Analysis of Algorithm 2

We now analyze the competitive ratio of the fractional algorithm. We start by claiming the feasibility of the solutions obtained throughout the execution of the algorithm.

▶ **Lemma 15.** *The primal solution x defined by Algorithm 2 is feasible for the* LP (6).

▶ **Lemma 16.** *The solution z returned by Algorithm 2 is feasible for* (6).

▶ **Lemma 17.** *Algorithm 2 returns a feasible dual solution to the* LP (6).

It remains to prove that the algorithm is $O(\log(\mu))$-competitive. To do so, we bound the increase in the primal x by a $O(\log \mu)$ factor of the dual increase, at any time.

▶ **Lemma 18.** *The cost of x is bounded by $O(\log(\mu))$ times the value of the dual y.*

Proof. Consider an infinitesimal increase in the value of the dual solution y. Specifically, assume that the algorithm chooses a minimal set Q in Algorithm 2 for time step t and that the dual variable $y_t(\mathcal{P} \setminus Q)$ increases infinitesimally by $dy_t(\mathcal{P} \setminus Q)$. Let dx and dy denote the infinitesimal change in the objective value of x and y, respectively. We bound the increase dx in x as a function of the increase dy.

$$\begin{aligned} dx &= \sum_{p \in \mathcal{P} - p_t} dx_p(r(p,t)) \cdot c(p) \\ &= \sum_{p \in (\mathcal{P} - p_t) \setminus Q} \frac{dx_p(r(p,t)) \cdot c(p) \cdot dy_t(\mathcal{P} \setminus Q)}{dy_t(\mathcal{P} \setminus Q)} \\ &= \sum_{p \in (\mathcal{P} - p_t) \setminus Q} \ln(\mu + 1) \cdot \left(x_p(r(p,t)) + \frac{1}{\mu} \right) \cdot dy_t(\mathcal{P} \setminus Q). \end{aligned} \tag{9}$$

The first equality holds since the increase in $y_t(\mathcal{P} \setminus Q)$ induces an increase only on the primal variables corresponding to pages in $(\mathcal{P} - p_t) \setminus Q$ by (6). The last equality follows from the growth rate of a variable $x_p(r(p,t))$, for some $p \in (\mathcal{P} - p_t) \setminus Q$, as a result of the growth in $y_t(\mathcal{P} \setminus Q)$. We separately analyze two of the expressions in (9). First, since y changes as a result of the increase in the variable $y_t(\mathcal{P} \setminus Q)$, by Algorithm 2 it implies that

$$\sum_{p \in (\mathcal{P} - p_t) \setminus Q} x_p(r(p,t)) \cdot g_Q(p) \leq \sum_{p \in \mathcal{P} - p_t} x_p(r(p,t)) \cdot g_Q(p) < N - g(Q). \tag{10}$$

For the second expression, let $S' \in (\mathcal{P} - p_t) \setminus Q$ such that (i) $g_Q(S') \geq N - g(Q)$ and (ii) S' is of minimum cardinality of all such sets. Clearly, there is such S' as $S'' = (\mathcal{P} - p_t) \setminus Q$ satisfies the first condition (p_t is feasible alone in cache). Since S' satisfies the cover constraints it holds that $f(\mathcal{P} \setminus (S' \cup Q) \leq k$; thus, by the definition of μ it holds that $|\mathcal{P} \setminus (S' \cup Q| \leq \mu$.

Therefore,

$$\begin{aligned}\sum_{p\in(\mathcal{P}-p_t)\setminus Q}\frac{1}{\mu} &= \frac{|(\mathcal{P}-p_t)\setminus Q|-1}{\mu}\\ &= \frac{|S'|+|\mathcal{P}\setminus(S'\cup Q)|-1}{\mu}\\ &\leq \frac{N-g(Q)+\mu-1}{\mu}\\ &\leq N-g(Q)+1\leq 2\cdot(N-g(Q))\,.\end{aligned} \tag{11}$$

The first inequality holds since $|S'|\leq N-g(Q)$, as S' is the minimum cardinality set that covers the demand of $N-g(Q)$; thus, the marginal contribution of any page in S' to the cover is at least 1 implying the inequality. The first inequality also uses $|\mathcal{P}\setminus(S'\cup Q|\leq\mu$ as explained above. For the last inequality, note that $N-g(Q)\geq 1$ since we assume that $y_t(\mathcal{P}\setminus Q)$ increases at this time, implying that the corresponding constraint of t and S is not trivially satisfied. Therefore, by (9), (10), and (11),

$$\begin{aligned}dx &\leq \ln(\mu+1)\cdot dy_t(\mathcal{P}\setminus Q)\cdot\left(\sum_{p\in(\mathcal{P}-p_t)\setminus Q}x_p(r(p,t))+\sum_{p\in(\mathcal{P}-p_t)\setminus Q}\frac{1}{\mu}\right)\\ &= \ln(\mu+1)\cdot dy_t(\mathcal{P}\setminus Q)\cdot 3\cdot(N-g(Q))\\ &= O(\log(\mu))\cdot dy.\end{aligned} \tag{12}$$

Thus, by (12), every increase in y incurs an increase of at most a factor $O(\log(\mu))$ in x. Finally, note that if $x_p(j)\geq\frac{1}{2}$ then we immediately increase $x_p(j)$ to 1; this increase the total cost of x by a factor of 2 w.r.t. the value of y. ◀

To conclude, by Algorithm 2 and Algorithm 2 we can trivially bound the cost of z by a constant factor of the cost of x.

▶ **Observation 19.** *The cost of z is bounded by 4 times the cost of x.*

Finally, using the above we summarize the properties of z.

▶ **Lemma 20.** *Algorithm 2 returns a feasible primal solution z to (6) such that the following holds.*

1. *For all $p\in\mathcal{P}$ and $j\in[n_p]$ it holds that either $z_p(j)\in\{0,1\}$ or that $z_p(j)\in\left[\frac{1}{4\cdot N\cdot\mu},\frac{1}{2}\right]$.*
2. *The cost of z is bounded by $O(\log(\mu))$ times the cost of* OPT.

3.3 Randomized Rounding

In this section, we construct a randomized algorithm for supermodular paging based on an online rounding scheme of the solution z to the LP (6) obtained by Algorithm 2. Let

$$C=\max_{p,q\in\mathcal{P}}\frac{c(p)}{c(q)}$$

be the maximum ratio of costs taken over all pairs pages; we assume without the loss of generality that all costs are strictly positive. As a scaling factor for our algorithm, let $\alpha=\log\left(4\cdot C\cdot N^2\cdot\mu^2\right)$ and let z' be the solution obtained by augmenting z by a factor of α. That is, for all $p\in\mathcal{P}$ and $j\in[n_p]$ define $z'_p(j)=\min(1,\alpha\cdot z_p)$.

Our algorithm computes z' in an online fashion. At every time t, after updating z', the algorithm evicts each page $p \in \mathcal{P}$ from the cache with probability chosen carefully so that the total probability that p is missing from cache after this moment is exactly $z'_p(r(p,t))$. If the cache is not feasible after this randomized rounding procedure, we evict pages that increase the total cover until we reach feasibility. The pseudocode of the algorithm is given in Algorithm 3.

Algorithm 3 Randomized Rounding.

for $t \in T$ **do**
  If p_t is missing from cache: fetch p_t.
  Initialize $\Delta_{p_t} \leftarrow 0$.
  Update the solution z according to Algorithm 2.
  Define $z'_p(r(p,t)) \leftarrow \min(1, \alpha \cdot z_p)$ for all $p \in \mathcal{P}$.
  for $p \in \mathcal{P}$ **do**
    Evict p from cache with probability $\frac{z'_p(r(p,t)) - \Delta_p}{1 - \Delta_p}$.
    Update $\Delta_p = z'_p(r(p,t))$.
  end
  Let $\mathcal{F}$ be the pages outside of the cache (part of the cover).
  while $g(\mathcal{F}) < N$ **do**
    Evict a page $p \in \mathcal{P} \setminus \mathcal{F}$ such that $g_{\mathcal{F}}(p) > 0$.
  end
end
Return the integral solution.

Observe that we evict a page p at time t with probability $\frac{z'_p(r(p,t)) - \Delta_p}{\Delta_p}$, conditioned on the event that p is still in the cache. Thus, the probability that p belongs to the cache at the beginning of time t is Δ_p, and is $z'_p(r(p,t))$ after Algorithm 3. Thus, we have the following observation.

▶ **Observation 21.** *For all $p \in \mathcal{P}$ and $t \in T$ the probability that p is missing from the cache at time t is at least $z'_p(r(p,t))$.*

To analyze the performance of the algorithm, we use the following lemma. The proof follows from Lemma 2.5 in [20] combined with Observation 21.

▶ **Lemma 22.** *Let $\mathcal{F}$ be the set of pages outside cache at Algorithm 3 at time t. Then,*

$$\mathbb{E}[g(\mathcal{F})] \geq N - e^{-\alpha} \cdot N \geq N - \frac{1}{2 \cdot \mu^2 \cdot C \cdot N}.$$

Using Lemma 22, we bound the expected cost of the algorithm.

▶ **Lemma 23.** *Algorithm 3 returns a feasible integral solution for* supermodular paging *with expected cost $O(\log(\mu) \cdot \alpha) \cdot \mathrm{OPT} = O\left(\log^2(\mu) \cdot \log(C \cdot N)\right) \cdot \mathrm{OPT}$.*

The proof of Theorem 6 follows from Lemma 23. Moreover, the proof of Theorem 9 follows using the "round-or-separate" approach of [20].

References

1 Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. An o (log k)-competitive algorithm for generalized caching. *ACM Transactions on Algorithms (TALG)*, 15(1):1–18, 2018.

2 Noga Alon, Baruch Awerbuch, and Yossi Azar. The online set cover problem. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 100–105, 2003.

3 Bo Bai, Li Wang, Zhu Han, Wei Chen, and Tommy Svensson. Caching based socially-aware d2d communications in wireless content delivery networks: A hypergraph framework. *IEEE Wireless Communications*, 23(4):74–81, 2016.

4 Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. *Journal of the ACM (JACM)*, 59(4):1–24, 2012.

5 Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Randomized competitive algorithms for generalized caching. *SIAM Journal on Computing*, 41(2):391–414, 2012.

6 Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM (JACM)*, 30(3):479–513, 1983.

7 John K Bennett, John B Carter, and Willy Zwaenepoel. Adaptive software cache management for distributed shared memory architectures. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 125–134, 1990.

8 Pei Cao and Sandy Irani. {Cost-Aware}{WWW} proxy caching algorithms. In *USENIX Symposium on Internet Technologies and Systems (USITS 97)*, 1997.

9 Tao Cheng, Kuochen Wang, Li-Chun Wang, and Chain-Wu Lee. An in-switch rule caching and replacement algorithm in software defined networks. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.

10 Christian Coester, Roie Levin, Joseph Naor, and Ohad Talmon. Competitive algorithms for block-aware caching. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 161–172, 2022.

11 Peter J Denning. Virtual memory. *ACM Computing Surveys (CSUR)*, 2(3):153–189, 1970.

12 Mianxiong Dong, He Li, Kaoru Ota, and Jiang Xiao. Rule caching in sdn-enabled mobile access networks. *IEEE Network*, 29(4):40–45, 2015.

13 Ilan Doron-Arad, Guy Kortsarz, Joseph Naor, Baruch Schieber, and Hadas Shachnai. Approximations and hardness of packing partially ordered items. *In proc. WG*, 2024.

14 Ilan Doron-Arad and Joseph Naor. Non-linear paging, 2024. `arXiv:2404.13334`.

15 Amos Fiat, Richard M Karp, Michael Luby, Lyle A McGeoch, Daniel D Sleator, and Neal E Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

16 Samoda Gamage and Ajith Pasqual. High performance parallel packet classification architecture with popular rule caching. In *2012 18th IEEE International Conference on Networks (ICON)*, pages 52–57. IEEE, 2012.

17 Peixuan Gao, Yang Xu, and H Jonathan Chao. Ovs-cab: Efficient rule-caching for open vswitch hardware offloading. *Computer Networks*, 188:107844, 2021.

18 Yue Gao, Meng Wang, Zheng-Jun Zha, Jialie Shen, Xuelong Li, and Xindong Wu. Visual-textual joint relevance learning for tag-based social image search. *IEEE Transactions on Image Processing*, 22(1):363–376, 2012.

19 Gourab Ghoshal, Vinko Zlatić, Guido Caldarelli, and Mark EJ Newman. Random hypergraphs and their applications. *Physical Review E*, 79(6):066118, 2009.

20 Anupam Gupta and Roie Levin. The online submodular cover problem. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1525–1537. SIAM, 2020.

21 Huawei Huang, Song Guo, Peng Li, Weifa Liang, and Albert Y Zomaya. Cost minimization for rule caching in software defined networking. *IEEE Transactions on Parallel and Distributed Systems*, 27(4):1007–1016, 2015.

22 Jin Huang, Rui Zhang, and Jeffrey Xu Yu. Scalable hypergraph learning and processing. In *2015 IEEE International Conference on Data Mining*, pages 775–780. IEEE, 2015.

23 Simon Korman. On the use of randomization in the online set cover problem. *Weizmann Institute of Science*, 2, 2004.

24 He Li, Song Guo, Chentao Wu, and Jie Li. Fdrc: Flow-driven rule caching optimization in software defined networking. In *2015 IEEE International Conference on Communications (ICC)*, pages 5777–5782. IEEE, 2015.

25 Rui Li, Yu Pang, Jin Zhao, and Xin Wang. A tale of two (flow) tables: Demystifying rule caching in openflow switches. In *Proceedings of the 48th International Conference on Parallel Processing*, pages 1–10, 2019.

26 Rui Li, Bohan Zhao, Ruixin Chen, and Jin Zhao. Taming the wildcards: Towards dependency-free rule caching with freecache. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2020.

27 Shijing Li and Tian Lan. Hotdedup: Managing hot data storage at network edge through optimal distributed deduplication. In *39th IEEE Conference on Computer Communications, INFOCOM 2020, Toronto, ON, Canada, July 6-9, 2020*, pages 247–256. IEEE, 2020.

28 David J Lilja. Cache coherence in large-scale shared-memory multiprocessors: Issues and comparisons. *ACM Computing Surveys (CSUR)*, 25(3):303–338, 1993.

29 Qingshan Liu, Yuchi Huang, and Dimitris N Metaxas. Hypergraph with sampling for image retrieval. *Pattern Recognition*, 44(10-11):2255–2262, 2011.

30 Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM (JACM)*, 41(5):960–981, 1994.

31 Keisuke Murakami and Takeaki Uno. Efficient algorithms for dualizing large-scale hypergraphs. In *2013 Proceedings of the Fifteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 1–13. SIAM, 2013.

32 Rob Patro and Carl Kingsford. Predicting protein interactions via parsimonious network history inference. *Bioinformatics*, 29(13):i237–i246, 2013.

33 Seyed Hamed Rastegar, Aliazam Abbasfar, and Vahid Shah-Mansouri. Rule caching in sdn-enabled base stations supporting massive iot devices with bursty traffic. *IEEE Internet of Things Journal*, 7(9):8917–8931, 2020.

34 Ori Rottenstreich, Ariel Kulik, Ananya Joshi, Jennifer Rexford, Gábor Rétvári, and Daniel S Menasché. Cooperative rule caching for sdn switches. In *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*, pages 1–7. IEEE, 2020.

35 Ori Rottenstreich and János Tapolcai. Optimal rule caching and lossy compression for longest prefix matching. *IEEE/ACM Transactions on Networking*, 25(2):864–878, 2016.

36 Nicolò Ruggeri, Martina Contisciani, Federico Battiston, and Caterina De Bacco. Community detection in large hypergraphs. *Science Advances*, 9(28):eadg9159, 2023.

37 Jang-Ping Sheu and Yen-Cheng Chuo. Wildcard rules caching and cache replacement algorithms in software-defined networking. *IEEE Transactions on Network and Service Management*, 13(1):19–29, 2016.

38 Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

39 Michael Stonebraker, Anant Jhingran, Jeffrey Goh, and Spyros Potamianos. On rules, procedure, caching and views in data base systems. *ACM SIGMOD Record*, 19(2):281–290, 1990.

40 G Edward Suh, Larry Rudolph, and Srinivas Devadas. Dynamic partitioning of shared cache memory. *The Journal of Supercomputing*, 28(1):7–26, 2004.

41 Shulong Tan, Jiajun Bu, Chun Chen, Bin Xu, Can Wang, and Xiaofei He. Using rich social media information for music recommendation via hypergraph model. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 7(1):1–22, 2011.

42 Ze Tian, TaeHyun Hwang, and Rui Kuang. A hypergraph-based learning algorithm for classifying gene expression and arraycgh data with prior knowledge. *Bioinformatics*, 25(21):2831–2838, 2009.

43 Andrew W Wilson Jr. Hierarchical cache/bus architecture for shared memory multiprocessors. In *Proceedings of the 14th annual international symposium on Computer architecture*, pages 244–252, 1987.

44 Laurence A Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.

45 Bo Yan, Yang Xu, and H Jonathan Chao. Adaptive wildcard rule cache management for software-defined networks. *IEEE/ACM Transactions on Networking*, 26(2):962–975, 2018.

46 Bo Yan, Yang Xu, Hongya Xing, Kang Xi, and H Jonathan Chao. Cab: A reactive wildcard rule caching system for software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 163–168, 2014.

47 Jialun Yang, Tao Li, Jinli Yan, Junnan Li, Chenglong Li, and Baosheng Wang. Pipecache: High hit rate rule-caching scheme based on multi-stage cache tables. *Electronics*, 9(6):999, 2020.

48 Neal E Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002.

49 Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. *Advances in neural information processing systems*, 19, 2006.

New Tradeoffs for Decremental Approximate All-Pairs Shortest Paths

Michal Dory
University of Haifa, Israel

Sebastian Forster
Department of Computer Science, University of Salzburg, Austria

Yasamin Nazari
Vrije Universiteit Amsterdam, The Netherlands

Tijn de Vos
Department of Computer Science, University of Salzburg, Austria

Abstract

We provide new tradeoffs between approximation and running time for the decremental all-pairs shortest paths (APSP) problem. For undirected graphs with m edges and n nodes undergoing edge deletions, we provide four new approximate decremental APSP algorithms, two for weighted and two for unweighted graphs. Our first result is $(2+\epsilon)$-APSP with total update time $\tilde{O}(m^{1/2}n^{3/2})$ (when $m = n^{1+c}$ for any constant $0 < c < 1$). Prior to our work the fastest algorithm for weighted graphs with approximation at most 3 had total $\tilde{O}(mn)$ update time for $(1+\epsilon)$-APSP [Bernstein, SICOMP 2016]. Our second result is $(2+\epsilon, W_{u,v})$-APSP with total update time $\tilde{O}(nm^{3/4})$, where the second term is an additive stretch with respect to $W_{u,v}$, the maximum weight on the shortest path from u to v.

Our third result is $(2+\epsilon)$-APSP for unweighted graphs in $\tilde{O}(m^{7/4})$ update time, which for sparse graphs ($m = o(n^{8/7})$) is the first subquadratic $(2+\epsilon)$-approximation. Our last result for unweighted graphs is $(1+\epsilon, 2(k-1))$-APSP, for $k \geq 2$, with $\tilde{O}(n^{2-1/k}m^{1/k})$ total update time (when $m = n^{1+c}$ for any constant $c > 0$). For comparison, in the special case of $(1+\epsilon, 2)$-approximation, this improves over the state-of-the-art algorithm by [Henzinger, Krinninger, Nanongkai, SICOMP 2016] with total update time of $\tilde{O}(n^{2.5})$. All of our results are randomized, work against an oblivious adversary, and have constant query time.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Theory of computation → Shortest paths

Keywords and phrases Decremental Shortest Path, All-Pairs Shortest Paths

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.58

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2211.01152` [34]

Funding *Michal Dory*: This work was partially conducted while the author was a postdoc at ETH Zurich. This work was supported in part by funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 853109), and the Swiss National Foundation (project grant 200021-184735).
Sebastian Forster, Yasamin Nazari, Tijn de Vos: This work is supported by the Austrian Science Fund (FWF): P 32863-N. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 947702).

1 Introduction

The dynamic algorithms paradigm is becoming increasingly popular for studying algorithmic questions in the presence of gradually changing inputs. A natural goal in this area is to design algorithms that process each change to the input as fast as possible to adapt the algorithm's output (or a data structure for querying the output) to the current state of the input. The time spent after an update to perform these computations is called the *update time* of the algorithm. In many cases, bounds on the update time are obtained in an *amortized* sense as an average over a long enough sequence of updates. Among dynamic graph problems, the question of maintaining exact or approximate shortest paths has received considerable attention in the past two decades. The main focus usually lies on maintaining a distance oracle that answers queries for the distance between a pair of nodes. For this problem, we call an algorithm *fully dynamic* if it supports both insertions and deletions of edges, and *partially dynamic* if it supports only one type of updates; in particular we call it *decremental* if it only supports edge deletions (which is the focus of this paper), and *incremental* if it only supports edge insertions.

The running times of partially dynamic algorithms are usually characterized by their bounds on the *total update time*, which is the accumulated time for processing all updates in a sequence of at most m deletions (where m is the maximum number of edges ever contained the graph). A typical design choice, which we also impose in this paper, is small (say polylogarithmic) query time. In particular, our algorithms will have constant query time. While fully dynamic algorithms are more general, the restriction to only one type of updates in partially dynamic algorithms often admits much faster update times. In particular, some partially dynamic algorithms have a total update time that almost matches the running time of the fastest static algorithm, i.e., computing *all* updates does not take significantly more time than processing the graph once.

Decremental shortest paths

For the decremental single-source shortest paths problem, conditional lower bounds [58, 48] suggest that *exact* decremental algorithms have an $\Omega(mn)$ bottleneck in their total update time (up to subpolynomial factors). On the other hand, this problem admits a $(1+\epsilon)$-approximation (also called *stretch*) with total update time $m^{1+o(1)}$ in weighted, undirected graphs [22, 47, 20, 19, 21], which exceeds the running time of the state-of-the art static algorithm by only a subpolynomial factor. Hence for the single-source shortest paths problem on undirected graphs we can fully characterize for which multiplicative stretches the total update time of the fastest decremental algorithm matches (up to subpolynomial factors) the running time of the fastest static algorithm.

Obtaining a similar characterization for the decremental all-pairs shortest paths (APSP) problem is an intriguing open question. Conditional lower bounds [32, 48] suggest an $\Omega(mn)$ bottleneck in the total update time of decremental APSP algorithms with (a) any finite stretch on *directed* graphs and (b) with any stretch guarantee with a multiplicative term of $\alpha \geq 1$ and an additive term of $\beta \geq 0$ with $2\alpha + \beta < 4$ on *undirected* graphs. This motivates the study of decremental (α, β)-approximate APSP algorithms such that $2\alpha + \beta \geq 4$, i.e., with multiplicative stretch $\alpha \geq 2$ or additive stretch $\beta \geq 2$.

Apart from two notable exceptions [46, 5], all known decremental APSP algorithms fall into one of two categories. They either (a) maintain exact distances or have a relatively small multiplicative stretch of $(1+\epsilon)$ or (b) they have a stretch of at least 3. The space in between is largely unexplored in the decremental setting. This stands in sharp contrast to the static

setting where for undirected graphs approximations guarantees different from multiplicative $(1+\epsilon)$ or "3 and above" have been the focus of a large body of works [7, 32, 28, 14, 16, 51, 11, 55, 61, 52, 8, 9]. The aforementioned exceptions in dynamic algorithms [46, 5], concern unweighted undirected graphs and maintain $(1+\epsilon, 2)$-approximations that simultaneously have a multiplicative error of $1+\epsilon$ and an additive error of 2, which implies a purely multiplicative $(2+\epsilon)$-approximation. The algorithms run in $\tilde{O}(n^{5/2})$[1] and $n^{5/2+o(1)}$ total update time respectively. In the next section, we will detail how our algorithms improve over this total update time and generalize to weighted graphs.

Concerning the fully dynamic setting, Bernstein [17] provided an algorithm for $(2+\epsilon)$-APSP which takes $m^{1+o(1)}$ time per update. Although we are not aware of any lower bounds, it seems to be hard to beat this: no improvements have been made since. The lack of progress in the fully-dynamic setting motivates the study in a partially dynamic setting, where we obtain improvements for the decremental case.

1.1 Our Results

In this paper, we provide novel decremental APSP algorithms with approximation guarantees that previously were mostly unexplored in the decremental setting. Our algorithms are randomized and we assume an oblivious adversary. For each pair of nodes, we can not only provide the distance estimate, but we can also report a shortest path π of this length in $\tilde{O}(|\pi|)$ time using standard techniques, see e.g. [43].

$(2+\epsilon)$-APSP for weighted graphs

Our first contribution is an algorithm for maintaining a $(2+\epsilon)$-approximation in weighted, undirected graphs.

▶ **Theorem 1.** *Given a weighted graph G and a constant $0 < \epsilon < 1$, there is a decremental data structure that maintains a $(2+\epsilon)$-approximation of APSP. The algorithm has constant query time and the total update time is w.h.p.*

- $\tilde{O}(m^{1/2}n^{3/2}\log^2(nW))$ *if $m = n^{1+\Omega(1)}$ and $m \leq n^{2-\rho}$ for an arbitrary small constant ρ,*
- $\tilde{O}(m^{1/2}n^{3/2+\rho}\log^2(nW))$ *if $m \geq n^{2-\rho}$ for any constant ρ,*
- $\tilde{O}(m^{1/2}n^{3/2+o(1)}\log^2(nW))$ *otherwise,*

where W is the ratio between the maximum and minimum weight.

The fastest known algorithm with an approximation ratio at least as good as ours is the $(1+\epsilon)$-approximate decremental APSP algorithm by Bernstein [18] with total update time $\tilde{O}(mn)$. With our $(2+\epsilon)$-approximation, we improve upon this total update time when $m = n^{1+\eta}$, for any $\eta > 0$. Furthermore, the fastest known algorithm with a larger approximation ratio than our algorithm is the $(3+\epsilon)$-approximate decremental distance oracle by Łącki and Nazari [53] with total update time $\tilde{O}(m\sqrt{n})$. Our result also has to be compared to the *fully dynamic* algorithm of Bernstein [17] for maintaining a $(2+\epsilon)$-approximation that takes amortized time $m^{1+o(1)}$ per update. Note that in *unweighted* graphs, $(2+\epsilon)$-approximate decremental APSP algorithm can be maintained with total update time $\tilde{O}(n^{2.5})$ (which is implied by the results of Abraham and Chechik [5] and Henzinger, Krinninger, and Nanongkai [46]). Our approach improves upon this bound for most densities, and matches it for $m = n^2$.

[1] In the introduction, we make two simplifying assumptions: (a) ϵ is a constant and (b) the ratio W between the maximum and minimum weight is polynomial in n. Unless otherwise noted, the cited algorithms have constant or polylogarithmic query time. Throughout we use $\tilde{O}(\cdot)$ notation to omit factors that are polylogarithmic in n.

$(2+\epsilon, 1)$-APSP for unweighted and $(2+\epsilon, W_{u,v})$-APSP for weighted graphs

Our second contribution is a faster algorithm with an additional additive error term of 1 for unweighted graphs, which in turn can be used to obtain our unweighted $(2+\epsilon)$-APSP result. The corresponding generalization to weighted graphs can be formulated as follows.

▶ **Theorem 2.** *Given a weighted graph G and a constant $0 < \epsilon < 1$, there is a decremental data structure that maintains a $(2+\epsilon, W_{u,v})$-approximation for APSP, where $W_{u,v}$ is the maximum weight on a shortest path from u to v. The algorithm has constant query time and the total update time is w.h.p. $\tilde{O}(nm^{3/4}\log^2(nW))$ if $m = n^{1+\Omega(1)}$, and $\tilde{O}(n^{1+o(1)}m^{3/4}\log^2(nW))$ otherwise, where W is the ratio between the maximum and minimum weight.*

$(2+\epsilon)$-APSP for unweighted graphs

We obtain this result by a general reduction from mixed approximations to purely multiplicative approximations, which might be of independent interest beyond the dynamic setting. See Theorem 5 for the statement. We can then combine this with Theorem 2 to obtain a fast algorithm for non-dense unweighted graphs.

▶ **Theorem 3.** *Given an undirected unweighted graph G, there is a decremental data structure that maintains a $(2+\epsilon)$-approximation for APSP with constant query time. W.h.p. the total update time is bounded by $\tilde{O}(m^{7/4})$ if $m = n^{1+\Omega(1)}$, and $\tilde{O}(m^{7/4+o(1)})$ otherwise.*

Note that for $m = o(n^{8/7})$ this gives the first subquadratic decremental $(2+\epsilon)$-approximate APSP algorithm. For $m \leq n^{6/5}$, this approach is faster than our weighted result, Theorem 1, which was already beating the unweighted state-of-the-art [5, 46].

$(1+\epsilon, 2(k-1))$-APSP for unweighted graphs

Our fourth contribution is an algorithm for unweighted, undirected graphs that maintains, for any $k \geq 2$, a $(1+\epsilon, 2(k-1))$-approximation, i.e., a distance estimate that has a multiplicative error of $1+\epsilon$ and an additive error of $2(k-1)$.

▶ **Theorem 4.** *Given an undirected unweighted graph G, a constant $0 < \epsilon < 1$ and an integer $2 \leq k \leq \log n$, there is a decremental data structure that maintains $(1+\epsilon, 2(k-1))$-approximation for APSP with constant query time. The expected total update time is bounded by $\min\{O(n^{2-1/k+o(1)}m^{1/k}), \tilde{O}((n^{2-1/k}m^{1/k})O(1/\epsilon)^{k/\rho})\}$ where $m = n^{1+\rho}$.*

Note that for small values of k and if $m = n^{1+\rho}$ for a constant $\rho > 0$, we get total update time of $\tilde{O}(n^{2-1/k}m^{1/k})$, and otherwise we have an extra $n^{o(1)}$ factor. In addition, in the special case of $k = \log n$, we get a near-quadratic update time of $O(n^{2+o(1)})$. The state-of-the-art for a purely multiplicative $(1+\epsilon)$-approximation is the algorithm of Roditty and Zwick with total update time $\tilde{O}(mn)$.[2] It was shown independently by Abraham and Chechik [5] and by Henzinger, Krinninger, and Nanongkai [46] how to improve upon this total update time bound at the cost of an additional small additive error term: a $(1+\epsilon, 2)$-approximation can be maintained with total update time $\tilde{O}(n^{2.5})$. This has been generalized by Henzinger, Krinninger, and Nanongkai [45] to an additive error term of $2(1+\frac{2}{\epsilon})^{k-2}$ and total update time $\tilde{O}(n^{2+1/k}O(\frac{1}{\epsilon})^{k-1})$. We improve upon this tradeoff in two ways: (1) our additive term is independent of $1/\epsilon$ and *linear* in k and (2) our algorithm profits from graphs being sparse.

[2] Note that the algorithms of Roditty and Zwick [59] for unweighted, undirected graphs precedes the more general algorithm of Bernstein [18] for weighted, directed graphs.

Static follow-up work

Interestingly, our techniques inspired a new algorithm for a static 2-approximate distance oracle [33], giving the first such distance oracle algorithm with subquadratic construction time for sparse graphs. Moreover, for $m = n$ they match conditional lower bounds [56, 3].

1.2 Related Work

Static algorithms

The baseline for the static APSP problem are the exact textbook algorithms with running times of $O(n^3)$ and $\tilde{O}(mn)$, respectively. There are several works obtaining improvements upon these running times by either shaving subpolynomial factors or by employing fast matrix multiplication, which sometimes comes at the cost of a $(1+\epsilon)$-approximation instead of an exact result. See [67] and [66], and the references therein, for details on these approaches. For the regime of stretch 3 and more in undirected graphs, a multitude of algorithms has been developed with the distance oracle of Thorup and Zwick [64] arguably being the most well-known constructions. In the following, we focus on summarizing the state of affairs for approximate APSP with stretch between $1+\epsilon$ and 3.

In *weighted*, undirected graphs, Cohen and Zwick [28] obtained a 2-approximation with running time $\tilde{O}(m^{1/2}n^{3/2})$. This running time has been improved to $\tilde{O}(m\sqrt{n}+n^2)$ by Baswana and Kavitha [14] and, employing fast matrix multiplication, to $\tilde{O}(n^{2.25})$ by Kavitha [51]. For $(2+\epsilon)$-APSP, Dory et al. [33] provide two algorithms, for dense graphs: $O(n^{2.214})$ and sparse graphs: $\tilde{O}(mn^{2/3})$. In addition, efficient approximation algorithms for stretches of $\frac{7}{3}$ [28] and $\frac{5}{2}$ [51] have been obtained. These results have recently been generalized by Akav and Roditty [9] who presented an algorithm with stretch $2+\frac{k-2}{k}$ and running time $\tilde{O}(m^{2/k}n^{2-3/k}+n^2)$ for any $k \geq 2$.

In *unweighted*, undirected graphs, a $(1,2)$-approximation algorithm with running time $\tilde{O}(n^{2.5})$ has been presented by Aingworth, Chekuri, Indyk and Motwani [7]. This has been improved by Dor, Halperin, and Zwick [32] that showed a $(1,2)$-approximation with running time $\tilde{O}(\min\{n^{3/2}m^{1/2}, n^{7/3}\})$. They also show a generalized version of the algorithm that gives stretch $(1,k)$ and running time $\tilde{O}(\min\{n^{2-2/(k+2)}m^{2/(k+2)}, n^{2+2/(3k-2)}\})$ for every even $k > 2$. Recently faster $(1,2)$-approximation algorithms based on fast matrix multiplication techniques were developed [31, 36], the fastest of them runs in $O(n^{2.260})$ time [36]. This was extended to a $(1+\epsilon,2)$-approximation in $O(n^{2.152})$ [33]. In addition, recently Roditty [57] extended the approach of [32] to obtain a combinatorial $(2,0)$-approximation for APSP in $\tilde{O}(n^{2.25})$ time in unweighted undirected graphs. Using fast matrix multiplication, this running time was improved to $O(n^{2.032})$ [33, 60].

Berman and Kasiviswanathan [16] showed how to compute a $(2,1)$-approximation in time $\tilde{O}(n^2)$. Subsequent works [11, 14, 55, 61, 52] have improved the polylogarithmic factors in the running time and the space requirements for such nearly 2-approximations. Recently, slightly subquadratic algorithms have been given: an algorithm with stretch $(2(1+\epsilon),5)$ by Akav and Roditty [8], and an algorithm with stretch $(2,3)$ by Chechik and Zang [25].

Decremental algorithms

The fastest algorithms for maintaining exact APSP under edge deletions have total update time $\tilde{O}(n^3)$ [30, 13, 39]. There are several algorithms that are more efficient at the cost of returning only an approximate solution. In particular, a $(1+\epsilon)$-approximation can be maintained in total time $\tilde{O}(mn)$ [59, 18, 49]. If additionally, an additive error of 2 is tolerable,

then a $(1+\epsilon, 2)$-approximation can be maintained in total time $\tilde{O}(n^{2.5})$ in unweighted, undirected graphs [46, 5]. Note that such a $(1+\epsilon, 2)$-approximation directly implies a $(2+\epsilon)$-approximation because the only paths of length 1 are edges between neighboring nodes. All of these decremental approximation algorithms are randomized and assume an oblivious adversary. Deterministic algorithms with stretch $1+\epsilon$ exist for unweighted, undirected graphs with running time $\tilde{O}(mn)$ [46], for weighted, undirected graphs with running time $\tilde{O}(mn^{1+o(1)})$ [21] and for weighted, directed graphs with running time $\tilde{O}(n^3)$ [50].

Decremental approximate APSP algorithms of larger stretch, namely at least 3, have first been studied by Baswana, Hariharan, and Sen [12]. After a series of improvements [59, 22, 6, 47], the state-of-the-art algorithms of [24, 53] maintain $(2k-1)(1+\epsilon)$-approximate all-pairs shortest paths for any integer $k \geq 2$ and $0 < \epsilon \leq 1$ in total update time $\tilde{O}((m+n^{1+o(1)})n^{1/k})$ with query time $O(\log\log(nW))$ and $O(k)$ respectively. All of these "larger stretch" algorithms are randomized and assume an oblivious adversary.

Recently, deterministic algorithms have been developed by Chuzhoy and Saranurak [27] and by Chuzhoy [26]. One tradeoff in the algorithm of Chuzhoy [26] for example provides total update time $\tilde{O}(m^{1+\mu})$ for any constant μ and polylogarithmic stretch. As observed by Mądry [54], decremental approximate APSP algorithms that are deterministic – or more generally work against an adaptive adversary – can lead to fast static approximation algorithms for the maximum multicommodity flow problem via the Garg-Könemann-Fleischer framework [44, 41]. The above upper bounds for decremental APSP have recently been contrasted by conditional lower bounds [4] stating that constant stretch cannot be achieved with subpolynomial update and query time under certain hardness assumptions on 3SUM or (static) APSP.

Fully dynamic algorithms

The reference point in fully dynamic APSP with subpolynomial query time is the *exact* algorithm of Demetrescu and Italiano [29] with update time $\tilde{O}(n^2)$ (with log-factor improvements by Thorup [62]). For undirected graphs, several fully dynamic distance oracles have been developed. In particular, Bernstein [17] developed a distance oracle of stretch $2+\epsilon$ (for any given constant $0 < \epsilon \leq 1$) and update time $O(m^{1+o(1)})$. In the regime of stretch at least 3, tradeoffs between stretch and update time have been developed by Abraham, Chechik, and Talwar [6], and by Forster, Goranci, and Henzinger [42]. Finally, most fully dynamic algorithm with update time sensitive to the edge density can be combined with a fully dynamic spanner algorithm leading to faster update time at the cost of a multiplicative increase in the stretch, see [15] for the seminal work on fully dynamic spanners.

2 High-Level Overview

In this section we provide a high-level overview of our algorithms, for the complete algorithms and their proof we refer to the full version. First we describe our $(2+\epsilon)$-APSP algorithms for weighted graphs, by giving a simpler static version first. Then we describe our $(2+\epsilon, W_{u,v})$-APSP algorithm, where we introduce a notion of *bunch overlap threshold* to overcome the challenge of dynamically maintaining a well-known static adaptive sampling technique [63] (see Section 2.2). We give a reduction from mixed approximations to purely multiplicative approximations, which together with the previous result gives our $(2+\epsilon)$-APSP algorithm for unweighted graphs. Finally, we describe our $(1+\epsilon, 2(k-1))$-APSP algorithm for any $k \geq 2$ in unweighted graphs.

2.1 $(2+\epsilon)$-APSP for Weighted Graphs

We first present a warm-up static algorithm, that we later turn into a dynamic algorithm.

2.1.1 Static 2-APSP

We start by reviewing the concepts of bunch and cluster as defined in the seminal distance oracle construction of [64].

Let $0 < p < 1$ be a parameter and A a set of nodes sampled with probability p. For each node u, the pivot $p(u)$ is the closest node in A to u. The *bunch* $B(v)$ of a node v is the set $B(v) := \{u \in V : d(u,v) < d(v,p(v))\}$ and the *clusters* are $C(u) := \{v \in V : d(u,v) < d(v,p(v))\} = \{v \in V : u \in B(v)\}$. Hence bunches and clusters are reverse of each other: $u \in B(v)$ if any only if $v \in C(u)$.

These structures have been widely used in various settings, including the decremental model (e.g. [46, 58, 53, 24]). We can use these existing decremental algorithms to maintain bunches and clusters, but will need to develop new decremental tools to get our desired tradeoffs that we will describe in Section 2.1.2.

Using the well-known distance oracles of [64], we know that by storing the clusters and bunches and the corresponding distances inside them, we can query 3-approximate distances for any pair. At a high-level, for querying distance between a pair u, v we either have $u \in B(v)$ or $v \in B(u)$ and so we explicitly have stored their distance, or we can get a 3-approximate estimate by computing $\min\{d(u,p(u)) + d(p(u),v), d(v,p(v)) + d(p(v),u)\}$. In the following we explain that in some special cases we can use these estimates to a obtain 2-approximation for a pair u, v, and in other cases by storing more information depending on how $B(u)$ and $B(v)$ overlap we can also obtain a 2-approximation to $d(u,v)$.

Let us assume that for all nodes v we have computed $B(v)$ and have access to all the distances from v to each node $u \in B(v)$. Also assume that we have computed the distances from the set A to all nodes. We now describe how using this information we can get a 2-approximate estimate of *any* pair of nodes u, v in one case. In another case we will discuss what other estimates we need to precompute. Let π be the shortest path between u and v. We consider two cases depending on how the bunches $B(u)$ and $B(v)$ interact with π. A similar case analysis was used in [23, 35] in distributed approximate shortest paths algorithms.

1. There exists $w \in \pi$ such that $w \notin B(u) \cup B(v)$ (left case in Figure 1): Since w is on the shortest path, we either have $d(w,u) \leq d(u,v)/2$ or $d(v,w) \leq d(u,v)/2$. Suppose $d(w,u) \leq d(u,v)/2$. We observe that by definition $d(u,p(u)) \leq d(u,w)$, hence we obtain $d(u,p(u))+d(p(u),v) \leq d(u,p(u))+d(u,p(u))+d(u,v) \leq 2d(u,v)$. The case that $d(v,w) \leq d(u,v)/2$ is analogous, and hence by computing $\min\{d(u,p(u)) + d(p(u),v), d(v,p(v)) + d(p(v),u)\}$ we get a 2-approximation.
2. There exists no $w \in \pi$ such that $w \notin B(u) \cup B(v)$. In other words, $B(u) \cap B(v) \cap \pi = \emptyset$ and there exists at least one edge $\{u', v'\}$ on π where $u' \in B(u)$ and $v' \in B(v)$ (right case in Figure 1[3]): in this case we can find the minimum over estimates obtained through all such pairs u', v', i.e. by computing $d(u,u') + w(u',v') + d(v',v)$.

If we had access to all the above distances for the bunches and the pivots, we could then use them to query 2-approximate distances between any pair $u, v \in V$. As we will see, in our algorithms we only have *approximate bunches and pivots* which lead to $(2+\epsilon)$-approximate queries.

[3] Although the picture implies $B(u)$ and $B(v)$ are disjoint, this also includes the case where they overlap, or even where $u \in B(v)$ or $v \in B(u)$.

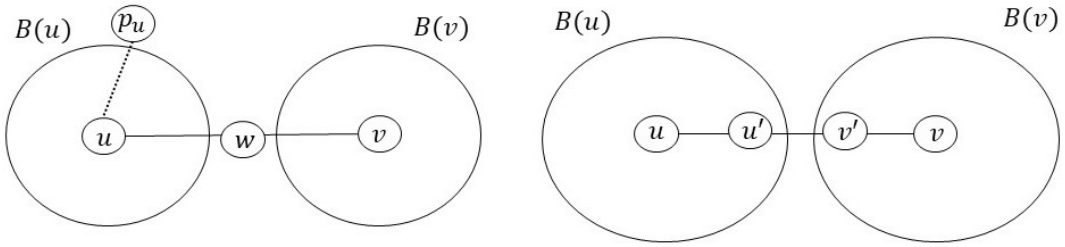


Figure 1 Possible scenarios for the overlap between the bunches $B(u)$, $B(v)$ and the shortest path π from u to v.

Running time

We can compute bunches and clusters in $\tilde{O}(\frac{m}{p})$ time [64]. We need to compute shortest paths from A for the estimates in Case 1. This takes $O(|A|m) = \tilde{O}(pnm)$ time using Dijkstra's algorithm. Finally, we consider Case 2. A straight-forward approach is to consider each pair of vertices u, v, and all $u' \in B(u)$ and $v' \in B(v)$. This takes $O(\frac{n^2}{p^2})$ time.

We use a sophisticated intermediate step that computes the distance between u' and v if there is an edge $\{u', v'\} \in E$ such that $v' \in B(v)$. For each node $v \in V$, we consider all $v' \in B(v)$, for which we consider all neighbors $u' \in N(v')$ and compute the estimate $w(u', v') + d(v', v)$ (and replace current $u' - v$ minimum if it is smaller). Since $|B(v)| = \tilde{O}(\frac{1}{p})$, we can compute these estimates for $d(u', v)$ in $\tilde{O}(\frac{n^2}{p})$ time.

Next, for each pair $u, v \in V$, we take the minimum over the distances $d(u, u') + d(u', v)$ for all u' in the bunch $B(u)$ using the precomputed distances from the previous step in time $\tilde{O}(n^2/p)$, by iterating over all pairs $u, v \in V$, and all $u' \in B(u)$. So in total the adjacent case takes time $\tilde{O}(\frac{n^2}{p})$. Hence the total running time of the algorithm will be $\tilde{O}(pnm + m/p + n^2/p) = \tilde{O}(pnm + n^2/p)$. By setting $p = \sqrt{\frac{n}{m}}$ we obtain the total update time $\tilde{O}(m^{1/2}n^{3/2})$ as stated in Theorem 1.

There are several subtleties that make it more difficult to maintain these estimates in decremental settings since the bunches and which bunches are adjacent keep on changing over the updates. We next discuss how these can be handled. We obtain a total update time that matches the stated static running time (up to subpolynomial factors).

2.1.2 Dynamic Challenges

Maintaining bunches and pivots

First, we need to dynamically maintain bunches efficiently as nodes may join and leave a bunch throughout the updates using an adaptation of prior work. One option would be maintaining the clusters and bunches using the [59] framework, however using this directly is slow for our purposes. Hence we maintain *approximate* clusters and bunches using hopsets of [53]. This algorithm also maintains approximate pivots, i.e. pivots that are within $(1 + \epsilon)$-approximate distance of the true closest sampled node. These estimates let us handle the first case (up to $(1 + \epsilon)$ approximation). One subtlety is that the type of approximate bunches used in [53] is slightly different with the type of approximate bunches we need for other parts of our algorithm. Specifically we need to bound the number of times nodes in a bunch can change. Roughly speaking, we can show this since the graph is decremental and the estimates obtained from the algorithm of [53] are monotone. Hence we perform *lazy bunch updates*, where we only let a bunch grow if the distance to the pivot grows by at least a factor $1 + \epsilon$. This means a bunch grows at most $O(\log_{1+\epsilon}(nW))$ times. In addition, these approximate bunches need to be taken into account into our stretch analysis.

We note that $(1+\epsilon)$-approximate decremental SSSP takes $\tilde{O}(m^{1+o(1)})$ total update time. However, we use the multi-source shortest path result from [53], where the $n^{o(1)}$-term vanishes when applied with polynomially many sources, giving us a total update time of $\tilde{O}(|A|m)$.

Maintaining estimates for Case 2

When we are in Case 2, we need more tools to keep track of estimates going through these nodes since both the bunches and the distances involved are changing. In particular, we explain how by using heaps in different parts of our algorithm we get efficient update and query times. Moreover, note that we have an additional complication: there are two data structures in this step, where one impacts the other. We need to ensure that an update in the original graph does not lead to many changes in the first data structure, which all need to be processed for the second data structure.

To be precise, we need intermediate data structures $Q_{u',v}$ that store approximate distances for u' and v of the form $w(u',v') + \delta(v',v)$ for each $v' \in B(v)$ such that $\{u',v'\} \in E$, as explained for the static algorithm[4]. As opposed to the static algorithm, we do not only keep the minimum, but instead maintain a min-heap, from which the minimum is easily extracted.

However, this approach has a problem: $\delta(v',v)$ might change many times, and for each such change we need to update at most $|N(v')| \leq n$ heaps. To overcome this problem, we use another notion of *lazy distance update*: we only update a bunch if the estimate of a node changes by at least a factor $1+\epsilon$. Combining this with the fact that, due to the lazy *bunch* update, nodes only join a cluster at most $O(\log_{1+\epsilon}(nW))$ times, and thus there is only an $O(\log_{1+\epsilon}(nW)\log(nW))$ overhead in maintaining these min-heaps $Q_{u',v}$ dynamically due to bunch updates.

Similarly, $w(u',v')$ might change many times, which has an impact for every $v \in C(v')$. Since this can be many nodes, we need to use an additional trick so an adversary cannot increase our update time to mn. Again the solution is a lazy update scheme: instead of $w(u',v')$ we use $\tilde{w}(u',v') = (1+\epsilon)^{\lceil \log_{1+\epsilon} w(u',v') \rceil}$, which can only change $\log_{1+\epsilon} W$ times. This comes at the cost of a $(1+\epsilon)$ approximation factor.

We note that our bunches are approximate bunches in three different ways. We have one notion of approximation due to the fact that we are using hopsets (that implicitly maintain bunches on scaled graphs). We have a second notion of approximation due to our lazy bunch update, which only lets nodes join a bunch a bounded number of times. We have a third notion of approximation due to the lazy distance update, which only propagates distance changes $O(\log(nW))$ times. We need to carefully consider how these different notions of approximate bunch interact with each other and with the stretch of the algorithm.

Next, we want to use the min-heaps $Q_{u',v}$ to maintain the distance estimates for $u,v \in V$ (see again the right case in Figure 1). We construct min-heaps $Q'_{u,v}$, with for each pair u,v entries $\delta(u,u') + \delta(u',v)$ for $u' \in B(u)$. Here $\delta(u',v)$ is the minimum from the intermediate data structure $Q_{u',v}$.

An entry $\delta(u,u') + \delta(u',v)$ in $Q'_{u,v}$ has to be updated when either $\delta(u,u')$ or $\delta(u',v)$ change. We have to make sure we do not have to update these entries too often. For the first part of each entry, we bound updates to the distance estimates $\delta(u,u')$ by the lazy distance updates. One challenge here is that when the distance estimates in a bunch change we need to update all the impacted heaps i.e. if $\delta(u,u')$ inside a bunch $B(u)$ changes, we must update all the heaps $Q'_{u,v}$ such that there exists $\{u',v'\} \in E$ with $v' \in B(v)$. For this purpose one

[4] In this section, $\delta(\cdot,\cdot)$ denotes $(1+\epsilon)$-approximate distances in our dynamic data structures.

approach is the following: for each $u \in V, u' \in B(u)$ we keep an additional data structure $\text{Set}_{u,u'}$ that stores for which $v \in V$ we have $v' \in B(v)$ with $\{u', v'\} \in E$. This means that given a change in $\delta(u, u')$, we can update each of the heaps in logarithmic time.

For the second part of the entry, which is the distance estimate $\delta(u', v)$ from the first data structure, this is more complicated. Another subtlety for our decremental data structures is the fact that $\delta(u', v)$ may also decrease due to a new node being added to $B(v)$. However, our lazy bunch update ensures that $\delta(u', v)$ can only decrease $O(\log_{1+\epsilon}(nW))$ times. Further, we use lazy distance increases *in between* such decreases. Hence in total we propagate changes to $\delta(u', v)$ at most $O(\log_{1+\epsilon}(nW)\log(nW))$ times, and thus our decremental algorithm has $\tilde{O}(\log_{1+\epsilon}(nW)\log(nW))$ overhead compared to the static algorithm.

2.2 $(2+\epsilon, W_{u,v})$-APSP for Weighted Graphs

For a moment, let us go back to the first static algorithm. It turns out, that if instead of computing an estimate for two "adjacent" bunches, we keep an estimate for bunches that overlap in at least one node, we obtain a $(2, W_{u,v})$-approximation, where $W_{u,v}$ is the maximum weight on the shortest path from u to v.

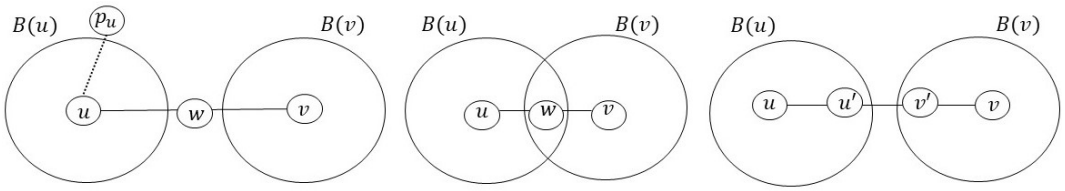


Figure 2 Possible scenarios for the overlap between the bunches $B(u)$ and $B(v)$ and the shortest path π from u to v.

The stretch analysis comes down to the following cases:

- There exists no $w \in \pi \cap B(u) \cap B(v)$ (the left and right case in Figure 2).
- There exists $w \in \pi$ such that $w \in B(u) \cap B(v)$ (the middle case in Figure 2).

The crucial observation here is that the distance estimate via the pivot (Case 1 in the previous section) already gives a $(2, W_{u,v})$-approximation for the adjacent case (the right case in Figure 2). So if we maintain an estimate for $d(u, v)$ if there exists $w \in \pi$ such that $w \in B(u) \cap B(v)$ (the "overlap case"), then we obtain a $(2, W_{u,v})$-approximation in total.

Here we need to create a min-heap for the overlap case, later we show we can maintain in $\tilde{O}(nC_{\max}/p)$ time. First we focus on the challenge of bounding $C_{\max}$.

Efficiency challenge

While the bunches are small, there is no bound on the size of the clusters. To overcome this issue in the static case, Thorup and Zwick developed an alternative way to build the bunches and clusters, that guarantees that the clusters are also small [63]. At a high-level, their approach is to adaptively change (grow) A by sampling big clusters into smaller clusters using a sampling rate proportional to the cluster size and adding these nodes to the set of pivots A. In total, in $\tilde{O}(m/p)$ time we obtain bunches and clusters, *both* of size $\tilde{O}(1/p)$. This idea was developed in the context of obtaining compact routing schemes, and later found numerous applications in static algorithms for approximating shortest paths and distances (see e.g., [14, 8, 10, 9]).

Using this trick, the static algorithm has running time $\tilde{O}(\frac{m}{p} + pnm + \frac{n}{p^2}) = \tilde{O}(nm^{3/4} + m^{4/3})$, for $p = m^{-1/3}$ or $\tilde{O}(mn^{1/2} + n^2)$ for $p = n^{-1/2}$.

However, the adaptive sampling procedure does not seem to be well-suited to a dynamic setting. The reason is that the sampling procedure is adaptive in such a way that when the bunches/clusters change, the set of sampled nodes can also change. Although (a dynamic adaptation of) the algorithm can guarantee that the set of sampled nodes is small at any particular moment, there is no guarantee on the *monotonicity* of the set A of sources that we need to maintain distances from. This would add a significant amount of computation necessary to propagate these changes to other parts of the algorithm. A possible solution is to enforce monotonicity by never letting nodes leave A. This would require us to bound the *total* number of nodes that would ever be sampled, which seems impossible with the current approach. To overcome this, we next suggest a new approach that gives us the monotonicity needed for our dynamic algorithm. For simplicity, we first present this in a static setting.

Introducing bunch overlap thresholds

Our proposed approach is to divide the nodes into two types depending on the number of bunches they appear in. In particular, we consider a threshold parameter τ and we define a node to be a *heavy node* if it is in more than τ bunches (equivalently its cluster contains more than τ nodes), and otherwise we call it a *light node*. In other words, for the set of light nodes we have $C_{\max} \leq \tau$. This threshold introduces a second type of pivot for each node u, which we denote by $q(u)$ that is defined to be the closest heavy node to u. In particular, instead of computing all distance estimates going through nodes $w \in B(u) \cap B(v)$, we only compute them for the case that w is light, and otherwise it is enough to compute the minimum estimates going through the heavy pivots, i.e. $\min\{d(u, q(u)) + d(q(u), v), d(v, q(v)) + d(q(v), u)\}$. We emphasize that these heavy pivots are *not a subset* of our original pivots $p(v), v \in V$, and they have a different behavior in bounding the running time than the *bunch pivots*. As we will see later, introducing these thresholds is crucial in efficiently obtaining the estimates required for Case 2.2 as we can handle the light and heavy case separately.

Stretch of the algorithm with bunch overlap thresholds

As discussed above, the stretch analysis was divided into two cases. In the new variant of the algorithm we add a third case, where we look at the distance estimates going through heavy pivots, i.e. $\min\{d(u, q(u)) + d(q(u), v), d(v, q(v)) + d(q(v), u)\}$. The main idea is that now we only need to consider the difficult case, Case 2.2, if the relevant node is light, which allows us to implement the algorithm efficiently.

If the relevant node $w \in B(u) \cap B(v)$ in Case 2 is heavy, we obtain a 2-approximation through the heavy pivot: we get that the $\min\{d(u, w), d(v, w)\} \leq d(u, v)/2$, since $d(u, v) = d(u, w) + d(w, v)$. W.l.o.g. say $d(u, w) \leq d(u, v)/2$. Then $d(u, q(u)) \leq d(u, w) \leq d(u, v)/2$, hence $d(u, q(u)) + d(q(u), v) \leq d(u, q(u)) + d(q(u), u) + d(u, v) \leq 2d(u, v)$.

Maintaining estimates through heavy nodes

For the stretch analysis above, we need to maintain a shortest path tree from each heavy node. We ensure a bunch grows at most $O(\log_{1+\epsilon}(nW))$ times. Hence by a total load argument we get that the *total* number of nodes in all bunches over all updates is at most $O(\frac{n}{p} \log_{1+\epsilon}(nW))$. This implies there cannot be too many heavy nodes in total, hence we can enforce monotonicity by keeping in V_{heavy} all nodes that were once heavy. As a consequence, we can maintain multi-source approximate distances from all heavy nodes efficiently: we can compute shortest paths from this set to all other nodes in $O(|V_{\text{heavy}}|m) = O(\frac{nm}{p\tau})$ time.

Running time of the algorithm with bunch overlap thresholds

Summarizing, we compute bunches and clusters in $O(m/p)$ time, compute distances from A to V in $O(|A|m) = \tilde{O}(pnm)$ time, and we can compute shortest paths from ththe heavy nodes in $O(|V_{\text{heavy}}|m) = O(\frac{nm}{p\tau})$ time.

Finally, we consider Case 2.2. For each node $u \in V$, we consider every light node $w \in B(u)$, and then all nodes $v \in C(W)$. Since w is light, we have $|C(w)| \leq \tau$, so this takes time $\tilde{O}(n \cdot \frac{1}{p} \cdot \tau)$.

Hence, in the total update time, we obtain $\tilde{O}((pnm + \frac{nm}{p\tau} + n\tau/p)\log^2(nW))$. Setting $p = m^{-1/4}$ and $\tau = m^{1/2}$ gives a total update time of $\tilde{O}(nm^{3/4}\log(nW))$.

Again for a decremental algorithm, we need to combine this new idea with the subtleties we had in the previous section: the bunches, their overlaps, and the heavy nodes keep on changing over the updates. We obtain a total update time that matches the stated static running time (up to subpolynomial factors) for the algorithm using bunch overlap thresholds.

2.2.1 $(2+\epsilon)$-APSP for Unweighted Graphs

We observe that in unweighted graphs mixed approximation can always be turned into a multiplicative approximation – at the cost of a blow-up in the number of vertices. More precisely, we prove the following reduction.

▶ **Theorem 5.** *Let $\mathcal{A}$ be an algorithm that provides a $(a+\epsilon, k)$-approximation for APSP in $\tau_{\mathcal{A}}(n', m')$ time, for any unweighted n'-node m'-edge graph G', and any constants $a, k \in \mathbb{N}_{\geq 1}$ and $\epsilon \in [0, 1)$. Given an unweighted graph $G = (V, E)$ on n nodes and m vertices, we can compute $(a + (k+2)\epsilon, 0)$-APSP in $\tau_{\mathcal{A}}(n + km, (k+1)m)$ time.*

If $\mathcal{A}$ is a dynamic algorithm, then each update takes $(k+1) \cdot$ [update time of $\mathcal{A}$] time. The query time remains the same (up to a constant factor).

The result is obtained by splitting every edge into k edges, by introducing new nodes on the edge. Now a path corresponding to a $+k$ approximation cannot take a non-trivial detour.

We combine this result with Theorem 2 to obtain a $(2+\epsilon)$-approximation for unweighted graphs in $\tilde{O}(m^{7/4})$ total update time.

2.3 $(1+\epsilon, 2(k-1))$-APSP for Unweighted Graphs

In this section, we describe our near-additive APSP algorithm. Our work is inspired by a classic result of Dor, Halperin, and Zwick [32], that presented a static algorithm that computes purely additive $+2(k-1)$ approximation for APSP in $\tilde{O}(n^{2-1/k}m^{1/k})$ time in unweighted graphs. Our goal is to obtain similar results dynamically. More concretely, we obtain decremental $(1+\epsilon, 2(k-1))$-approximate APSP in $O(n^{2-1/k+o(1)}m^{1/k})$ total update time in unweighted graphs.

2.3.1 $(1+\epsilon, 2)$-APSP

To explain the high-level idea of the algorithm, we first focus on the special case that $k = 2$, and that we only want to approximate the distances between pairs of nodes at distance at most d from each other, for a parameter d. In this case, we obtain a $+2$-additive approximation in $\tilde{O}(n^{3/2}m^{1/2}d)$ time. This case already allows to present many of the high-level ideas of the algorithm. Later we explain how to extend the results to the more general case. We start by describing the static algorithm from [32], and then explain the dynamic version. The static version of the data structure is as follows:

- Let $s_1 = (\frac{m}{n})^{1/2}$. Let V_1 be the set of *dense nodes*: $V_1 := \{v \in V : \deg(v) \geq s_1\}$. Also let E_2 be the set of *sparse edges*, i.e. edges with at least one endpoint with degree less than s_1.
- **Node set D_1.** Construct a hitting set D_1 of nodes in V_1. This means that every node $v \in V_1$ has a neighbor in D_1. The size of D_1 is $O(n \log n/s_1)$.
- **Edge set E^*.** Let E^* be a set of size $O(n)$ such that for each $v \in V_1$, there exists $u \in D_1 \cap N(v)$ such that $\{u, v\} \in E^*$.
- **Computing distances from D_1.** Store distances $D_1 \times V$ by running a BFS from each $u \in D_1$ on the input graph G.
- **Computing distances from $V \setminus D_1$.** For each $u \in V \setminus D_1$ store a shortest path tree, denoted by T_u rooted at u by running Dijkstra on $(V, E_2 \cup E^* \cup E_u^{D_1})$, where $E_u^{D_1}$ is the set of weighted edges corresponding to distances in $(\{u\} \times D_1)$ computed in the previous step.

Dynamic data structure

The static algorithm computes distances in two steps. First, it computes distances from D_1 by computing BFS trees in the graph G. This step can be maintained dynamically by using *Even-Shiloach trees (ES-trees)* [40], a data structure that maintains distances in a decremental graph. Maintaining the distances up to distance $O(d)$ takes $O(|D_1|md)$ time. The more challenging part is computing distances from $V \setminus D_1$. Here the static algorithm computes distances in the graphs $H_u = (V, E_2 \cup E^* \cup E_u^{D_1})$. Note that the graphs H_u change dynamically in several ways. First, in the static algorithm, the set E_2 is the set of light edges. To keep the correctness of the dynamic algorithm, every time that a node v no longer has a neighbor in D_1, we should add all its adjacent edges to E_2. Second, when an edge in E^* is deleted from G, we should replace it by an alternative edge if such an edge exists. Finally, the weights of the edges in $E_u^{D_1}$ can change over time, when the estimates change. This means that other than deletions of edges, we can also add new edges to the graphs H_u or change the weights of their edges. Because of these edge insertions we can no longer use the standard ES-tree data structure to maintain the distances, because that only works in a decremental setting.

To overcome this, we use *monotone ES-trees*, a generalization of ES-trees proposed by [46, 45]. In this data structure, to keep the algorithm efficient, when new edges are inserted, the distance estimates do not change. In particular, when edges are inserted, some distances may decrease, however, in such cases the data structure keeps an old larger estimate of the distance. The main challenge in using this data structure is to show that the stretch analysis still holds. In particular, in our case, the stretch analysis of the static algorithm was based on the fact the distances from the nodes $V \setminus D_1$ are computed in the graphs H_u, where in the dynamic setting, we do not have this guarantee anymore.

Stretch analysis

The high-level idea of the stretch analysis uses the special structure of the graphs H_u. To prove that the distance estimate between u and v is at most $d(u, v) + 2$, we distinguish between two cases. First, we maintain the property that as long as v has a neighbor $x \in D_1$, then an edge of the form $\{v, x\}$ is in E^* for $x \in D_1 \cap N(v)$, and since we maintain correctly the distances from D_1, we can prove that we get an additive stretch of at most 2 in this case.

In the second case, all the edges adjacent to v are in $E_2 \subseteq H_u$, and here we can use an inductive argument on the length of the path to prove that we get an additive stretch of at most 2. Note that the estimate that we get can be larger compared to the distance between u and v in the graph H_u, but we can still show that the additive approximation is at most 2 as needed.

Update time

Our update time depends on the size of the graphs H_u. While in the static setting it is easy to bound the number of edges in E_2 and E^*, in the decremental setting, as new edges are added to these sets, we need a more careful analysis. For example, in the static setting $|E^*| = O(n)$, as any node only adds one adjacent edge to the set E^*, where in the decremental setting, we may need to add many different edges to this set because the previous ones got deleted. However, we can still prove that even if a node needs to add to E^* edges to all its adjacent neighbors in D_1 during the algorithm, the size of H_u is small enough as needed. In the full version we show that the total update time is $\tilde{O}(m^{1/2}n^{3/2}d)$ in expectation, this matches the static complexity up to the factor d.

Handling large distances

The algorithm described above is efficient if d is small, to obtain an efficient algorithm for the general case, we combine our approach with a decremental near-additive APSP algorithm. More concretely, we use an algorithm from [45] that allows to compute $(1+\epsilon, \beta)$-approximation for APSP in $O(n^{2+o(1)})$ total update time, for $\beta = n^{o(1)}$. Now we set $d = \Theta(\beta/\epsilon)$, and distinguish between two cases. For pairs of nodes at distance at most d from each other, we get a $+2$-additive approximation as discussed above, in $O(m^{1/2}n^{3/2+o(1)})$ time. For pairs of nodes at distance larger than d, the near-additive approximation is already a $(1 + O(\epsilon))$-approximation by the choice of the parameter d. For such pairs, the additive β term becomes negligible, as their distance is $\Omega(\beta/\epsilon)$. Overall, we get a $(1+\epsilon, 2)$-approximation for APSP in $O(m^{1/2}n^{3/2+o(1)})$ time. In fact, if the graph is dense enough ($m = n^{1+\Omega(1)}$), the $n^{o(1)}$ term is replaced by a poly-logarithmic term.

2.3.2 General k

We can extend the algorithm to obtain a $(1 + \epsilon, 2(k - 1))$-approximate APSP in $O(n^{2-1/k+o(1)}m^{1/k})$ total update time, by adapting the general algorithm of [32] to the dynamic setting. At a high-level, instead of having one hitting set D_1, we have a series of hitting sets $D_1, D_2, ..., D_k$, such that the set D_i hits nodes of degree s_i. We compute distances from D_i in appropriate graphs H_u^i, that are sparser when the set D_i is larger. Balancing the parameters of the algorithm leads to the desired total update time.

2.3.3 Limitations of previous approaches

We next explain why approaches used in previous decremental algorithms cannot be generalized to obtain our near-additive results. First, the unweighted $(1 + \epsilon, 2)$-approximate APSP algorithm that takes $\tilde{O}(n^{5/2}/\epsilon)$ time [46], is inspired by static algorithms for $+2$-additive emulators, sparse graphs that preserve the distances up to a $+2$-additive stretch. The main idea is to maintain a sparse emulator of size $\tilde{O}(n^{3/2})$, and exploit its sparsity to obtain a fast algorithm. This construction however is specific for the $+2$ case, and cannot be generalized to a general k. Note that the update time of the algorithm crucially depends on the size of the

emulator, and to obtain a better algorithm for a general k, we need to be able to construct a sparser emulator for a general k. However, a beautiful lower bound result by Abboud and Bodwin [1] shows that for any constant k, a purely additive $+k$ emulator should have $\Omega(n^{4/3-\epsilon})$ edges. This implies that dynamic algorithms that are based on purely additive emulators seem to require $\Omega(n^{7/3-\epsilon})$ time, and we cannot get running time arbitrarily close to $O(n^2)$. An alternative approach is to build a *near-additive* emulator. This is done in [45], where the authors show a $(1+\epsilon, 2(1+2/\epsilon)^{k-2})$-approximation algorithm for decremental APSP with expected total update time of $\tilde{O}(n^{2+1/k}(37/\epsilon)^{k-1})$. Here the running time indeed gets closer to $O(n^2)$, but this comes at a price of a much worse additive term of $2(1+2/\epsilon)^{k-2}$. While in the static setting there are also other bounds obtained for near-additive emulators, such as $O(1+\epsilon, O(k/\epsilon)^{k-1})$ emulators of size $O(kn^{1+\frac{1}{2^{k+1}-1}})$ [38, 65, 2, 37], in all of the constructions the additive term depends on ϵ, and the dependence on ϵ is known to be nearly tight for small k by lower bounds from [2]. Hence this approach cannot lead to small constant additive terms that do not depend on ϵ.

References

1. Amir Abboud and Greg Bodwin. The 4/3 additive spanner exponent is tight. *Journal of the ACM (JACM)*, 64(4):1–20, 2017.
2. Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. *SIAM Journal on Computing*, 47(6):2203–2236, 2018.
3. Amir Abboud, Karl Bringmann, and Nick Fischer. Stronger 3-sum lower bounds for approximate distance oracles via additive combinatorics. In *Proc. of the 55th Annual ACM Symposium on Theory of Computing (STOC 2023)*, 2023. doi:10.48550/arXiv.2211.07058.
4. Amir Abboud, Karl Bringmann, Seri Khoury, and Or Zamir. Hardness of approximation in P via short cycle removal: cycle detection, distance oracles, and beyond. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2022)*, pages 1487–1500, 2022. doi:10.1145/3519935.3520066.
5. Ittai Abraham and Shiri Chechik. Dynamic decremental approximate distance oracles with $(1+\epsilon, 2)$ stretch. *CoRR*, abs/1307.1516, 2013. arXiv:1307.1516.
6. Ittai Abraham, Shiri Chechik, and Kunal Talwar. Fully dynamic all-pairs shortest paths: Breaking the o(n) barrier. In *Proceedings of the 17th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2014) and the 18th International Workshop on Randomization and Computation (APPROX/RANDOM 2014)*, pages 1–16, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.1.
7. Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999. Announced at SODA 1996. doi:10.1137/S0097539796303421.
8. Maor Akav and Liam Roditty. An almost 2-approximation for all-pairs of shortest paths in subquadratic time. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, (SODA 2020)*, pages 1–11, 2020. doi:10.1137/1.9781611975994.1.
9. Maor Akav and Liam Roditty. A unified approach for all pairs approximate shortest paths in weighted undirected graphs. In *Proceedings of the 29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204, pages 4:1–4:18, 2021. doi:10.4230/LIPIcs.ESA.2021.4.
10. Arturs Backurs, Liam Roditty, Gilad Segal, Virginia Vassilevska Williams, and Nicole Wein. Toward tight approximation bounds for graph diameter and eccentricities. *SIAM Journal on computing (SICOMP)*, 50(4):1155–1199, 2021.
11. Surender Baswana, Vishrut Goyal, and Sandeep Sen. All-pairs nearly 2-approximate shortest paths in $o(n^2 \operatorname{polylog} n)$ time. *Theoretical Computer Science*, 410(1):84–93, 2009. Announced at STACS 2005. doi:10.1016/j.tcs.2008.10.018.

12 Surender Baswana, Ramesh Hariharan, and Sandeep Sen. Maintaining all-pairs approximate shortest paths under deletion of edges. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2003)*, pages 394–403, 2003.

13 Surender Baswana, Ramesh Hariharan, and Sandeep Sen. Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. *Journal of Algorithms*, 62(2):74–92, 2007. Announced at STOC 2002. doi:10.1016/j.jalgor.2004.08.004.

14 Surender Baswana and Telikepalli Kavitha. Faster algorithms for all-pairs approximate shortest paths in undirected graphs. *SIAM Journal on Computing*, 39(7):2865–2896, 2010. Announced at FOCS 2006. doi:10.1137/080737174.

15 Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Transactions on Algorithms*, 8(4):35:1–35:51, 2012. doi:10.1145/2344422.2344425.

16 Piotr Berman and Shiva Prasad Kasiviswanathan. Faster approximation of distances in graphs. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Norbert Zeh, editors, *Proceedings of the 10th International Workshop on Algorithms and Data Structures (WADS 2007*, pages 541–552, 2007. doi:10.1007/978-3-540-73951-7_47.

17 Aaron Bernstein. Fully dynamic $(2 + \epsilon)$ approximate all-pairs shortest paths with fast query and close to linear update time. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009*, pages 693–702. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.16.

18 Aaron Bernstein. Maintaining shortest paths under deletions in weighted directed graphs. *SIAM Journal on Computing*, 45(2):548–574, 2016. Announced at STOC 2013. doi:10.1137/130938670.

19 Aaron Bernstein. Deterministic partially dynamic single source shortest paths in weighted graphs. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming, (ICALP 2017)*, pages 44:1–44:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.44.

20 Aaron Bernstein and Shiri Chechik. Deterministic decremental single source shortest paths: beyond the $O(mn)$ bound. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2016)*, pages 389–397, 2016. doi:10.1145/2897518.2897521.

21 Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental SSSP and approximate min-cost flow in almost-linear time. In *Proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS 2021)*, pages 1000–1008, 2021. doi:10.1109/FOCS52979.2021.00100.

22 Aaron Bernstein and Liam Roditty. Improved dynamic algorithms for maintaining approximate shortest paths under deletions. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011*, pages 1355–1365. SIAM, 2011. doi:10.1137/1.9781611973082.104.

23 Keren Censor-Hillel, Michal Dory, Janne H Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. *Distributed Computing*, 34(6):463–487, 2021.

24 Shiri Chechik. Near-optimal approximate decremental all pairs shortest paths. In Mikkel Thorup, editor, *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2018)*, pages 170–181, 2018. doi:10.1109/FOCS.2018.00025.

25 Shiri Chechik and Tianyi Zhang. Nearly 2-approximate distance oracles in subquadratic time. In *Proc. of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 551–580. SIAM, 2022. doi:10.1137/1.9781611977073.26.

26 Julia Chuzhoy. Decremental all-pairs shortest paths in deterministic near-linear time. In Samir Khuller and Virginia Vassilevska Williams, editors, *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2021)*, pages 626–639. ACM, 2021. doi:10.1145/3406325.3451025.

27 Julia Chuzhoy and Thatchaphol Saranurak. Deterministic algorithms for decremental shortest paths via layered core decomposition. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)*, pages 2478–2496, 2021. doi:10.1137/1.9781611976465.147.

28 Edith Cohen and Uri Zwick. All-pairs small-stretch paths. *Journal of Algorithms*, 38(2):335–353, 2001. Announced at SODA 1997. doi:10.1006/jagm.2000.1117.

29 Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM*, 51(6):968–992, 2004. Announced at STOC 2003. doi:10.1145/1039488.1039492.

30 Camil Demetrescu and Giuseppe F. Italiano. Fully dynamic all pairs shortest paths with real edge weights. *Journal of Computer and System Sciences*, 72(5):813–837, 2006. Announced at FOCS 2001. doi:10.1016/j.jcss.2005.05.005.

31 Mingyang Deng, Yael Kirkpatrick, Victor Rong, Virginia Vassilevska Williams, and Ziqian Zhong. New additive approximations for shortest paths and cycles. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 50:1–50:10. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.50.

32 Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000. Announced at FOCS 1996. doi:10.1137/S0097539797327908.

33 Michal Dory, Sebastian Forster, Yael Kirkpatrick, Yasamin Nazari, Virginia Vassilevska Williams, and Tijn de Vos. Fast 2-approximate all-pairs shortest paths. In *Proc. of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4728–4757. SIAM, 2024. arXiv:2307.09258.

34 Michal Dory, Sebastian Forster, Yasamin Nazari, and Tijn de Vos. New tradeoffs for decremental approximate all-pairs shortest paths. *CoRR*, abs/2211.01152, 2022. doi:10.48550/arXiv.2211.01152.

35 Michal Dory and Merav Parter. Exponentially faster shortest paths in the congested clique. *ACM Journal of the ACM (JACM)*, 69(4):1–42, 2022.

36 Anita Dürr. Improved bounds for rectangular monotone min-plus product and applications. *Information Processing Letters*, page 106358, 2023.

37 Michael Elkin and Ofer Neiman. Near-additive spanners and near-exact hopsets, a unified view. *Bulletin of EATCS*, 1(130), 2020.

38 Michael Elkin and David Peleg. $(1+\epsilon,\beta)$-spanner constructions for general graphs. *SIAM Journal on Computing*, 33(3):608–631, 2004.

39 Jacob Evald, Viktor Fredslund-Hansen, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. Decremental APSP in unweighted digraphs versus an adaptive adversary. In *Proc. of the 48th International Colloquium on Automata, Languages, and Programming, (ICALP 2021)*, pages 64:1–64:20, 2021. doi:10.4230/LIPIcs.ICALP.2021.64.

40 Shimon Even and Yossi Shiloach. An on-line edge-deletion problem. *J. ACM*, 28(1):1–4, 1981. doi:10.1145/322234.322235.

41 Lisa Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal of Discrete Mathematics*, 13(4):505–520, 2000. Announced at FOCS 1999. doi:10.1137/S0895480199355754.

42 Sebastian Forster, Gramoz Goranci, and Monika Henzinger. Dynamic maintenance of low-stretch probabilistic tree embeddings with applications. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, (SODA 2021)*, pages 1226–1245, 2021. doi:10.1137/1.9781611976465.75.

43 Sebastian Forster, Gramoz Goranci, Yasamin Nazari, and Antonis Skarlatos. Bootstrapping dynamic distance oracles. In *Proceedings of the 31th Annual European Symposium on Algorithms (ESA 2023)*, 2023.

44 Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007. Announced at FOCS 1998. doi:10.1137/S0097539704446232.

45 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A subquadratic-time algorithm for decremental single-source shortest paths. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 1053–1072. SIAM, 2014. doi:10.1137/1.9781611973402.79.

46 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Dynamic approximate all-pairs shortest paths: Breaking the $o(mn)$ barrier and derandomization. *SIAM Journal on Computing*, 45(3):947–1006, 2016. Announced at FOCS 2013.

47 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. *Journal of the ACM*, 65(6):36:1–36:40, 2018. Announced at FOCS 2014. doi:10.1145/3218657.

48 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing (STOC 2015)*, pages 21–30, 2015. doi:10.1145/2746539.2746609.

49 Adam Karczmarz and Jakub Łącki. Reliable hubs for partially-dynamic all-pairs shortest paths in directed graphs. In *Proceedings of the 27th Annual European Symposium on Algorithms (ESA 2019)*, pages 65:1–65:15, 2019. doi:10.4230/LIPIcs.ESA.2019.65.

50 Adam Karczmarz and Jakub Łącki. Simple label-correcting algorithms for partially dynamic approximate shortest paths in directed graphs. In *Proceedings of the 3rd Symposium on Simplicity in Algorithms (SOSA 2020)*, pages 106–120, 2020. doi:10.1137/1.9781611976014.15.

51 Telikepalli Kavitha. Faster algorithms for all-pairs small stretch distances in weighted graphs. *Algorithmica*, 63(1-2):224–245, 2012. Announced at FSTTCS 2007. doi:10.1007/s00453-011-9529-y.

52 Mathias Bæk Tejs Knudsen. Additive spanners and distance oracles in quadratic time. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming, (ICALP 2017)*, pages 64:1–64:12, 2017. doi:10.4230/LIPIcs.ICALP.2017.64.

53 Jakub Łącki and Yasamin Nazari. Near-Optimal Decremental Hopsets with Applications. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

54 Aleksander Mądry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC 2010)*, pages 121–130, 2010. doi:10.1145/1806689.1806708.

55 Mihai Patrascu and Liam Roditty. Distance oracles beyond the thorup-zwick bound. *SIAM Journal on Computing*, 43(1):300–311, 2014. Announced at FOCS 2010. doi:10.1137/11084128X.

56 Mihai Patrascu, Liam Roditty, and Mikkel Thorup. A new infinity of distance oracles for sparse graphs. In *Proc. of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012)*, pages 738–747. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.44.

57 Liam Roditty. New algorithms for all pairs approximate shortest paths. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 309–320. ACM, 2023. doi:10.1145/3564246.3585197.

58 Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011. Announced at ESA 2004. doi:10.1007/s00453-010-9401-5.

59 Liam Roditty and Uri Zwick. Dynamic approximate all-pairs shortest paths in undirected graphs. *SIAM J. Comput.*, 41(3):670–683, 2012. Announced at FOCS 2004. doi:10.1137/090776573.

60 Barna Saha and Christopher Ye. Faster approximate all pairs shortest paths. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4758–4827. SIAM, 2024. arXiv:2309.13225.

61 Christian Sommer. All-pairs approximate shortest paths and distance oracle preprocessing. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming, (ICALP 2016)*, volume 55, pages 55:1–55:13, 2016. doi:10.4230/LIPIcs.ICALP.2016.55.

62 Mikkel Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT 2004)*, pages 384–396, 2004. doi:10.1007/978-3-540-27810-8_33.

63 Mikkel Thorup and Uri Zwick. Compact routing schemes. In Arnold L. Rosenberg, editor, *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA 2001*, pages 1–10. ACM, 2001. doi:10.1145/378580.378581.

64 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005.

65 Mikkel Thorup and Uri Zwick. Spanners and emulators with sublinear distance errors. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 802–809, 2006.

66 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM Journal on Computing*, 47(5):1965–1985, 2018. Announced at STOC 2014. doi:10.1137/15M1024524.

67 Uri Zwick. Exact and approximate distances in graphs – A survey. In *Proceedings of the 9th Annual European Symposium on Algorithms (ESA 2001)*, volume 2161, pages 33–48, 2001. doi:10.1007/3-540-44676-1_3.

Decremental Matching in General Weighted Graphs

Aditi Dudeja ✉
University of Salzburg, Austria

Abstract

In this paper, we consider the problem of maintaining a $(1-\varepsilon)$-approximate *maximum weight matching* in a dynamic graph G, while the adversary makes changes to the edges of the graph. In the fully dynamic setting, where both edge insertions and deletions are allowed, Gupta and Peng [20] gave an algorithm for this problem with an update time of $\tilde{O}_\varepsilon(\sqrt{m})$. We study a natural relaxation of this problem, namely the decremental model, where the adversary is only allowed to delete edges. For the unweighted version of this problem in general (possibly, non-bipartite) graphs, [3] gave a decremental algorithm with update time $O_\varepsilon(\text{poly}(\log n))$. However, beating $\tilde{O}_\varepsilon(\sqrt{m})$ update time remained an open problem for the *weighted* version in *general graphs*. In this paper, we bridge the gap between unweighted and weighted general graphs for the decremental setting. We give a $O_\varepsilon(\text{poly}(\log n))$ update time algorithm that maintains a $(1-\varepsilon)$ approximate maximum weight matching under adversarial deletions. Like the decremental algorithm of [3], our algorithm is randomized, but works against an adaptive adversary. It also matches the time bound for the unweighted version upto dependencies on ε and a $\log R$ factor, where R is the ratio between the maximum and minimum edge weight in G.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Weighted Matching, Dynamic Algorithms, Adaptive Adversary

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.59

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2312.08996`

Funding *Aditi Dudeja*: This work is supported by Austrian Science Fund (FWF): P 32863-N. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 947702).

Acknowledgements Thanks to Aaron Bernstein, Sebastian Forster, Yasamin Nazari, and anonymous ICALP reviewers for valuable feedback.

1 Introduction

In a dynamic graph G with vertex set V, an adversary makes updates to the edge set E of the graph, and the algorithm is required to adjust to these changes and maintain a key property of the graph. An algorithm is called *decremental* if it can only cope with edge deletions, *incremental* if it can only cope with edge insertions, and *fully dynamic* if it can cope with both edge insertions and deletions. Typically, the algorithm has to optimize the update time, which is the time taken to adapt to a single edge update. For incremental (respectively, decremental) algorithms, one seeks to optimize the amortized update time, which is the average time taken over m edge insertions (respectively, deletions).

In this paper, we consider the problem of maintaining a $(1-\varepsilon)$-approximate matching in a dynamic graph. For the general fully dynamic setting, where we have an unweighted graph undergoing updates, the best known algorithms for this problem due to Bhattacharya, Kiss, Saranurak [11] and Assadi, Behnezhad, Khanna, Li [2] have update times of $O(m^{0.5-\Omega_\varepsilon(1)})$ and $O(n/\log^* n)$, respectively. Improving these bounds to $O_\varepsilon(\text{poly}(\log n))$ is a fundamental


Table 1 A Comparison of Weighted and Unweighted Dynamic Matching in General Graphs.

Setting	Approximation	Time (Unweighted)	Time (Weighted)
Fully Dynamic	$(1-\varepsilon)$	$O(m^{0.5-\Omega_\varepsilon(1)})$ [11]	$\tilde{O}_\varepsilon(\sqrt{m})$ [20]
Fully Dynamic	$(2-\sqrt{2})$	$\tilde{O}_\varepsilon(1)$ [5]	$\tilde{O}_\varepsilon(\sqrt{m})$ [20]
Fully Dynamic	0.609	$\tilde{O}(\min\{\Delta^{1/3}, m^{1/6}\})$ [7]	$\tilde{O}_\varepsilon(\sqrt{m})$ [20]
Incremental	$(1-\varepsilon)$	$O_\varepsilon(1)$ [18]	$\tilde{O}_\varepsilon(\sqrt{m})$ [20]
Decremental	$(1-\varepsilon)$	$\tilde{O}_\varepsilon(1)$ [3]	$\tilde{O}_\varepsilon(1)$ (new)

open problem, and conditional lower bounds of $\Omega(\sqrt{m})$ (see [21] and [23]) for the exact case, seem to suggest that this problem is quite challenging. This has motivated the study of more relaxed versions of this problem. For example, one line of research has focussed on obtaining considerably faster update times, but at the cost of worse approximation ratios [6, 12]. Another research direction has been to study this problem in partially dynamic settings. In the *incremental setting*, there has been steady progress over the years. Several results [19, 18, 11, 14] culminated in a $O_\varepsilon(1)$ update time algorithm for maintaining $(1-\varepsilon)$-approximate matching in incremental graphs. On the *decremental* side, [9] and [22] gave $O_\varepsilon(\text{poly}(\log n))$ update time algorithms for maintaining $(1-\varepsilon)$-approximate matching in bipartite graphs. Subsequently, [3] extended these results to the case of general graphs by giving a $O_\varepsilon(\text{poly}(\log n))$ update time algorithm for the same approximation ratio.

For the case of weighted graphs, the picture is less clear. A general reduction of Stubbs and Williams [25] shows that any α-approximation unweighted matching algorithm can be converted to a $(0.5-\varepsilon)\cdot\alpha$-approximation weighted matching algorithm with nearly the same update time. Subsequently, [8] showed that one can avoid this factor 2 loss in the approximation ratio for the specific case of **bipartite** graphs. However, for the case of general graphs (possibly, non-bipartite), there is a significant gap between the best known weighted and unweighted dynamic algorithms, as illustrated in Table 1. The main contribution of our paper is to close this gap between weighted and unweighted matching for general graphs in the decremental setting:

▶ **Result 1** (Formalized in Theorem 4). *Given a decremental weighted graph G, there is a randomized algorithm that with high probability maintains a $(1-\varepsilon)$-approximation to the maximum weight matching in G in total time $\tilde{O}_\varepsilon(m)$.*

Concurrent Work. Independent of this work, Chen, Sidford, and Tu also [15] gave a $O_\varepsilon(\text{poly}(\log n))$ update time algorithm for maintaining $(1-\varepsilon)$-maximum weight matching in decremental graphs. Their techniques are very different from ours, and extend the entropy regularization approach of [22], who originally gave this framework for bipartite unweighted graphs. Both approaches are of independent interest, since they use vastly different technical frameworks: [15] use insights from concave optimization, whereas our approach is combinatorial. Our approach also has a better dependence on $\log n$[1], and additionally uses simple algorithmic paradigms such as independent sampling or the static approximate matching algorithms of [16]. In contrast, their approach has better dependence on ε (in particular, their update time has a dependence of $(1/\varepsilon)^{O(1/\varepsilon)}$), but uses more involved algorithmic subroutines such as computing k-partial Gomory-Hu trees.

[1] while they don't state it explicitly, we determined that their algorithm has an update time of $O_\varepsilon(\log^{11} n)$.

2 High-Level Overview

Our high-level approach is to follow the congestion balancing framework of [9], who maintain a fractional matching that is robust to deletions. In order to do this, they hope to compute a fractional matching which "spreads out" its flow. Therefore, they impose a capacity function on the edges. These capacities are initially small, and are increased gradually. In particular, they repeatedly invoke a subroutine M-OR-E*() that does one of the following in $O_\varepsilon(m \cdot \log n)$ time (here $\mu(G)$ refers to the size of the maximum cardinality matching):

1. Either output a fractional matching $\vec{x}$, with $x(e) \leqslant \kappa(e)$ for all $e \in E$, and $\sum_{e \in E} x(e) \geqslant (1-\varepsilon) \cdot \mu(G)$, or,
2. Output a set of bottleneck edges E^*, along which we can increase capacities:
   a. We have, $\kappa(E^*) = O(\mu(G) \cdot \log n)$.
   b. Every large matching M has at least $\varepsilon \cdot \mu(G)$ edges in E^*.

Property 2 ensures that we increase capacities carefully: Property 2a ensures the capacity increase is small, and Property 2b ensures that we only increase capacity along important edges. The subroutine M-OR-E*() can be used as a black-box to get a decremental matching algorithm: if M-OR-E*() outputs a fractional matching, then we can round it to an integral matching (see [26, 10, 13]). Otherwise, the capacity obeying maximum fractional matching is too small, and the algorithm outputs a set of bottleneck edges E^*. Thus, we increase capacity along it. Property 2 ensures the matching obtained is "robust" to deletions since it ensures the capacities are small on average.

The authors of [9] also use an outer subroutine which repeatedly invokes M-OR-E*() to get a decremental matching algorithm. Similar to the case of [3], this subroutine carries over with some impediments for the case of **general weighted graphs** as well. But, M-OR-E*() is far more challenging to implement, and this will be the focus of this extended abstract.

2.1 M-or-E*() for General Weighted Graphs

We first explicate what the impediments for implementing M-OR-E*() in general graphs is. For bipartite graphs, M-OR-E*() is much easier to implement, since approximate fractional matchings obeying capacity constraints can be computed using approximate max flows. Similarly, the set of bottleneck edges E^* correspond to the minimum cut. However, in *general graphs*, we don't have such a natural correspondence to max-flow/min-cut due to the integrality gap. More specifically, we mean that not every fractional matching has a large integral matching in its support. On the other hand, while fractional matchings which satisfy odd set constraints do have large integral matchings in their support, this characterization doesn't seem computationally useful at first. [3], in the context of unweighted graphs defined a fractional matching that is both computationally easy to compute, and also avoids the integrality gap. In particular, their candidate fractional matching either puts flow 1 on an edge or a flow of at most ε. By a folklore lemma, one can show that such fractional matchings satisfy small odd set constraints, and therefore have an integrality gap of $1-\varepsilon$. This is sufficient for us, since we focus on only approximation algorithms.

In this work on weighted decremental graphs, we also focus on finding the fractional matchings mentioned above, since even in weighted graphs they are known to have a small integrality gap. Towards this, we show the following structural lemmas. We note that [3] also prove unweighted versions of these structural lemmas, but as we will state later, their proofs are tailored towards unweighted graphs. Consequently, to prove these lemmas for weighted graphs, we come up with significantly different proof strategies. In what follows, we will use $\mathsf{mwm}(G)$ to denote the maximum weight matching of G.

1. First, given a weighted graph G with capacities κ on the edges of the graph, we want to check if the maximum weight fractional matching obeying odd set constraints (denoted $\mathsf{mwm}(G,\kappa)$) is at least $(1-\varepsilon)\cdot\mathsf{mwm}(G)$. As mentioned before, we can't use approximate min-cost flow to find such a fractional matching. First, such an algorithm is computationally expensive. Secondly, the fractional matching returned by it may not obey odd set constraints. Therefore, akin to [3] we prove a sampling theorem: sample every edge e of G independently, with a probability $p(e) \propto \kappa(e)$ to construct an uncapacitated graph G_s. Then, we show, with high probability, $\mathsf{mwm}(G_s) \geqslant \mathsf{mwm}(G,\kappa) - \varepsilon\cdot n$. Thus, we can now use G_s as a substitute for (G,κ): more concretely, we can compute $(1-\varepsilon)\cdot\mathsf{mwm}(G_s)$ using efficient algorithms such as the one by [16], and thus, get an estimate on $\mathsf{mwm}(G,\kappa)$. The authors in [3] proved the unweighted version of this lemma. However, their proof strategy was to use the Tutte-Berge theorem. The Tutte-Berge theorem is the non-bipartite counterpart of the Hall's theorem. In Hall's theorem, deficiency is defined as $\max_{T\subseteq L}\{|T| - |N(T)|\}$, where L is the left vertex set and is equal to $n-\mu(G)$. In the Tutte-Berge theorem, deficiency is analogously defined. The authors in [3] showed that in G_s, with high probability, the deficiency is at most $n-(\mu(G,\kappa)+\varepsilon\cdot n)$, and consequently, $\mu(G_s) \geqslant \mu(G,\kappa) - \varepsilon\cdot n$. However, since Tutte-Berge theorem is specific to unweighted graphs, our proof deviates from this.
2. Suppose the sampling procedure tells us that $\mathsf{mwm}(G,\kappa) \geqslant (1-\varepsilon)\cdot\mathsf{mwm}(G)$. In this case, we would like to compute a fractional matching $\vec{x}$ such that $\sum_{e\in E} w(e)\cdot x(e) \geqslant \mathsf{mwm}(G,\kappa)$. Akin to [3], we consider any M such that $w(M) \geqslant (1-\varepsilon)\cdot\mathsf{mwm}(G_s)$, and we split up M into two parts $M_H \subseteq M$, which are edges with high capacity, and $M_L \subseteq M$ which are edges with low capacity, and $V_H = V(M_H)$ and $V_L = V(M_L)$. Let E_L be the low capacity edges of G. Then, we can show that with high probability, $|M_H| + \mathsf{mwm}(G[V_L]\cap E_L,\kappa) \geqslant \mathsf{mwm}(G_s) - \varepsilon\cdot n$. Since congestion balancing allows us to round capacities of M_H to 1, we only need to compute $(1-\varepsilon)$-approximation to $\mathsf{mwm}(G[V_L]\cap E_L,\kappa)$, and we would have the required fractional matching. There are a few impediments to showing this theorem, and to computing $\mathsf{mwm}(G[V_L]\cap E_L,\kappa)$.
   a. First, the unweighted analog of this theorem by [3], showed the opposite of the structural theorem in 1: it shows that deficiency of G_s can not reduce by too much either. To do this, they consider the bipartite double cover of G_s and (G,κ) and use Hall's theorem to derive this conclusion. As mentioned before, for weighted graphs, we don't have this tool at our disposal, and therefore, have to make other arguments. We look at the dual linear programs of G_s and (G,κ), and use facts about these to draw our conclusions. We believe our arguments can be seen as a generalization of the approach of [3]. Looking at the proof in this light also simplifies the proof considerably.
   b. The second challenge is the computational aspect. We would like to compute a $(1-\varepsilon)$-approximate maximum weight fractional matching in $G[V_L]\cap E_L$ that obeys κ. The authors in [3] observe that since $G[V_L]\cap E_L$ only consists of low capacity edges, we can transform this into a bipartite graph, and then use an approximate flow algorithm. However, for us that would mean computing an approximate min-cost flow, which is computationally expensive. Another way to do this is to use LP-solvers (see [1]). However, these incur additional $\log n$ factors, which we believe in the context of dynamic graph algorithms can be huge. Therefore, we give a simple scaling based algorithm that repeatedly uses maximal flow as a subroutine to obtain a $(1-\varepsilon)$ approximation to $\mathsf{mwm}(G[V_L]\cap E_L,\kappa)$, while avoiding these $\log n$ factors.

There are other peripheral challenges as well, such as computing the set E^*. [9] showed that these edges E^* corresponded to the min-cut, and [3] showed that one can use the duals of G_s to determine these edges. We extend the proof of [3] to weighted graphs. Secondly, the

congestion balancing framework, as shown in [3] and [9] only applies to unweighted graphs, we are able to show that it extends to weighted graphs as well. Finally, the known rounding schemes of [26, 10, 13] as stated only apply to unweighted graphs. We extend [26] to handle the weighted fractional matching we construct.

▶ Remark 2. The structural theorems stated in Item 1 and Item 2a incur an additive error of $\varepsilon \cdot n$. In order to account for this, we use the vertex sparsification technique of [4], which allows us to construct $O(R \cdot \log n)$ multigraphs H_i such that $|V(H_i)| \leqslant O(^{R \cdot \mathsf{mwm}(G)}/_{\varepsilon})$ and each matching of G is preserved in one of the H_i. Here, R is the ratio of the max-weight edge and min-weight edge in G. As is evident, this reduction will cause our algorithms to have a large dependence on R. However, we can reduce this dependence by using a reduction of [20] (see Lemma 3), which has been used similarly in other prior works (see [8, 15]). We refer the reader to Appendix C of [8] for a proof.

▶ **Lemma 3** ([20]). *Let G be a weighted graph with arbitrary weights with R being the ratio between maximum and minimum edge weights. Let $\varepsilon \in (0, 0.5)$ and let $\gamma_\varepsilon = (^1/_\varepsilon)^{O(1/\varepsilon)}$. If there is an algorithm $\mathcal{A}$ that maintains an α-approximate maximum weight matching in a graph whose weights are $\{1, 2, 3, \cdots, W\}$ with update time $T(n, m, \alpha, W)$, then there is an algorithm $\mathcal{A}'$ that maintains a $(1-\varepsilon) \cdot \alpha$-approximate maximum weight matching in G in update time $O(T(n, m, \alpha, \gamma_\varepsilon) \cdot \log R)$.*

In the rest of the paper we will assume that $\mathsf{mwm}(G) = \Omega(\log n)$. This is because the graph is undergoing deletions, and soon as $\mathsf{mwm}(G)$ drops below this value, we can switch to a different decremental algorithm which we describe in the full version of our paper. Additionally, from Remark 2 we can also assume that $\mathsf{mwm}(G) = \Omega(^{\varepsilon \cdot n}/_W)$. Finally, from Lemma 3 we can assume that our graphs are integer-weighted with weights in $\{1, 2, \cdots, W\}$. All our structural theorems will be proven using these assumptions in mind. We will justify these assumption in the full version of this paper.

3 Preliminaries

Throughout the paper, we will use G to refer to the current version of the graph, and let V and E be the vertex set and edge set of G, respectively. The graphs we deal with are weighted, and we use R to denote the ratio between the max-weight and min-weight edge. Additionally use $\mathsf{mwm}(G)$ to denote the weight of the maximum weight matching of G. During the course of the algorithm, we will maintain a fractional matching, which corresponds to a non-negative vector $\vec{x} \in [0,1]^m$ satisfying the following constraints : $\sum_{v \ni e} x(e) \leqslant 1$. For a set $S \subseteq E$, we let $x(S) = \sum_{e \in S} x(e)$. Given a capacity function κ on the edges of the graph, we say that $\vec{x}$ obeys κ if $x(e) \leqslant \kappa(e)$ for all $e \in E$. For a vector $\vec{x}$, we use $\mathrm{supp}(\vec{x})$ to denote the set of edges that are in the support of $\vec{x}$. For a fractional matching $\vec{x}$, we say that it satisfies odd set constraints if for all odd-sized sets $B \subseteq V$, we have, $\sum_{e \in G[B]} x(e) \leqslant \frac{|B|-1}{2}$. We use $\mathsf{mwm}(G, \kappa)$ to denote the size of the maximum weight fractional matching obeying the odd set constraints and the capacity function κ. Additionally, we will use $\gamma_\varepsilon = (^1/_\varepsilon)^{O(1/\varepsilon)}$, $\alpha_\varepsilon = 2^{^{W^2}/_{\varepsilon^3}} \cdot \log n$ and $\rho_\varepsilon = 2^{^{W^2}/_{\varepsilon^2}} \cdot \log n$. We also use V_{odd} to denote the set of all odd-sized subsets $B \subseteq V$. Our main result is a decremental algorithm for maintaining $(1-\varepsilon)$-approximate maximum weight matching in general graphs. In particular, we formally state our main theorem.

▶ **Theorem 4.** *Let G be a weighted graph with weight function w on the edges of the graph and let $\varepsilon \in (0, 1/2)$. There is a decremental algorithm with total update time $O_\varepsilon(m \cdot \log^7 n \cdot \log R)$ (amortized $\tilde{O}_\varepsilon(1)$) that maintains an integral matching M with $w(M) \geqslant (1-\varepsilon) \cdot \mathsf{mwm}(G)$, with high probability, where G refers to the current version of the graph. The algorithm is randomized but works against an adaptive adversary. The dependence on ε is $2^{(^1/_\varepsilon)^{O(1/\varepsilon)}}$.*

Roadmap for Extended Abstract. As mentioned in the overview, our main contribution is to give an algorithm M-OR-E*() for *weighted general graphs* (we call this subroutine WEIGHTEDM-OR-E*()). Thus, rest of this paper is dedicated to setting this up. We postpone the proof of Theorem 4 to the full version of this paper.

4 Properties of WeightedM-or-E*()

As mentioned in the overview, we will focus our attention on implementing M-OR-E*() in general **weighted** graphs. We will call this subroutine WEIGHTEDM-OR-E*() We additionally mentioned that we will be working with multigraphs, so this will motivate some definitions. After giving these definitions, we state the properties of WEIGHTEDM-OR-E*() that we need. Finally, we show that we are indeed able to implement WEIGHTEDM-OR-E*(). Recall, that is sufficient to design WEIGHTEDM-OR-E*() for the case of graph that have integer weights in $\{1, 2, \cdots, W\}$ by Section 2.1.

▶ **Definition 5.** *Let G be a multigraph. For a pair of vertices $u, v \in V$, we define $D_i(u, v)$ to be the edges e between u and v that have weight i. Additionally, we also have $D(u, v) = \cup_{i=1}^{W} D_i(u, v)$. Since we assume integral weights, these sets are well-defined. If e is an edge between $u, v \in V$, then $D_i(e) \coloneqq D_i(u, v)$ and $D(e) \coloneqq D(u, v)$.*

▶ **Definition 6.** *Let G be a weighted multigraph, with n vertices and m edges. Let κ be the capacity function on the edges of the graph, and let $\vec{x}$ be a fractional matching. We define $\vec{x}^C$ to be a vector, with support size $\min\left\{W \cdot \binom{n}{2}, m\right\}$, where for vertices $u, v \in V$, $x_i^C(u, v) = \sum_{e \in D_i(u,v)} x(e)$. That is, $\vec{x}^C$ is obtained by "collapsing" all edges of the same weight between a pair of vertices together. We now describe the opposite operation: the "distribution" operation. Similarly, suppose $\vec{y}$ is a vector with $|supp(\vec{y})| \leqslant \binom{n}{2} \cdot W$, where, $y_i(u, v)$ is the entry corresponding to the edge of weight i between u and v. Then, we define $\vec{y}^D$ to be an m length vector such that for every $e \in E$ between u, v with $w(e) = i$, $y^D(e) \coloneqq \frac{y_i(u,v) \cdot \kappa(e)}{\kappa(D_i(u,v))}$. Intuitively, $\vec{y}^D$ is a multigraph obtained by distributing the $y_i(u, v)$ among all edges of the same weight.*

We will next, state two folklore observations (proved in the full version of the paper), which motivates the main lemma which we want to prove.

▶ **Observation 7.** *Let $\vec{f}$ be a fractional matching on a multigraph, that puts flow 1 or, at most ε between every pair of vertices $u, v \in V$. Then, $\vec{f}$ satisfies odd set constraints of sets of size at most $1/\varepsilon$.*

▶ **Observation 8.** *Suppose $\vec{x}$ is a fractional matching that satisfies odd set constraints for all odd sets of size smaller than $3/\varepsilon + 1$, then the fractional matching $\vec{z} = \frac{\vec{x}}{(1+\varepsilon)}$ satisfies odd set constraints for all odd sets, and therefore, $\mathsf{mwm}(supp(\vec{x})) \geqslant (1 - \varepsilon) \cdot \sum_{e \in E} w(e) \cdot x(e)$.*

We now state the main lemma, which we focus on proving in the extended abstract.

▶ **Lemma 9.** *Let G be a multigraph with edge weights in $\{1, 2, \cdots, W\}$. Then, there is an algorithm WEIGHTEDM-OR-E*() that takes as input $G, \kappa, \varepsilon \in (0, 0.5)$, and a parameter $\mu \geqslant \mathsf{mwm}(G) \cdot (1 - \varepsilon)$, and in time $O(m \cdot W \cdot \log n/\varepsilon)$ returns one of the following.*

1. *A fractional matching $\vec{x}$ such that $\sum_{e \in E} w(e) \cdot x(e) \geqslant (1 - 2\varepsilon) \cdot \mathsf{mwm}(G)$, with the following properties.*
   - **a.** *Let $e \in D_i(u, v)$ such that $e \in supp(\vec{x})$, with $\kappa(D_i(u, v)) \geqslant 1/\alpha_\varepsilon^2$, then $x(D_i(u, v)) = 1$, and we have, $x(e) = \frac{\kappa(e)}{\kappa(D_i(u,v))}$. Moreover, since $\vec{x}$ is a fractional matching, we have $x(D_j(u, v)) = 0$ for all $j \neq i$.*

   b. *Consider any* $e \in D_i(u,v)$ *such that* $e \in supp(\vec{x})$*, with* $\kappa(D_i(u,v)) \leqslant 1/\alpha_\varepsilon^2$*, then* $x(D_i(u,v)) \leqslant \kappa(D_i(u,v)) \cdot \alpha_\varepsilon$*, and* $x(e) \leqslant \kappa(e) \cdot \alpha_\varepsilon$*. Moreover, for any* $e \in D_j(u,v)$ *with* $\kappa(D_j(u,v)) > 1/\alpha_\varepsilon^2$*, we have* $x(e) = 0$*.*
2. *A set of edges* E^* *such that* $\sum_{e \in E^*} w(e) \cdot \kappa(e) = O(\mathsf{mwm}(G) \cdot \log n)$*, and for any integral matching* M *with* $w(M) \geqslant (1-2\varepsilon) \cdot \mathsf{mwm}(G)$*, we have* $w(M \cap E^*) \geqslant \varepsilon \cdot \mathsf{mwm}(G)$*. Additionally, for all* $e \in E^*$*, we have* $\kappa(e) < 1$*.*

We give some intuition about why $\vec{x}$ avoids the integrality gap. Consider the situation in which the algorithm WeightedM-or-E*() returns a fractional matching $\vec{x}$. For any pair of vertices u, v consider $\sum_{i=1}^{W} x_i^C(u,v)$, then this is either 1 or at most ε by Lemma 9(1). Thus, it satisfies odd set constraints for all odd sets of size at most $1/\varepsilon$ by Observation 7. By Observation 8 , we can then argue that $\vec{x}$ contains an integral matching of weight at least $(1+\varepsilon)^{-1} \cdot \sum_{u \neq v} \sum_{i=1}^{W} x_i^C(u,v) \cdot i$ in its support.

We will use WeightedM-or-E*() as a subroutine in our decremental matching algorithm, and we will get a matching $\vec{x}$ with the following properties. We mention these properties since they will help us visualize the fractional matching better. These properties are evident once we state the algorithm WeightedM-or-E*().

▶ **Property 10.** *The set* E^* *returned WeightedM-or-E*() has the following properties.*
1. *Each time WeightedM-or-E*() returns* E^**, we will increase capacity along* E^* *by multiplying existing capacities by the same factor.*
2. *Consider* $u, v \in V$*, and let* e, e' *be two edges between* u *and* v *such that* $w(e) = w(e')$*, then* $\kappa(e) = \kappa(e')$ *at all times during the run of the algorithm.*

The next property follows directly as a consequence of Lemma 9(1).

▶ **Property 11.** *Let* $\vec{x}$ *be a matching output by Lemma 9, then* $x(e) \leqslant \kappa(e) \cdot \alpha_\varepsilon^2$ *for all* $e \in E$*. This is evident from Lemma 9(1).*

▶ **Definition 12.** *Let* G *be a multigraph, and let* $\vec{x}$ *be a fractional matching of* G*. Then, we split* $supp(\vec{x})$ *into two parts:* $\vec{x}^i$ *and* $\vec{x}^f$*, where,* $\vec{x} = \vec{x}^i + \vec{x}^f$*, and* $supp(\vec{x}^i) = \{e \mid x(D_j(e)) > 1/\alpha_\varepsilon^2 \text{ and } w(e) = j\}$ *and* $supp(\vec{x}^f) = \{e \mid x(D_j(e)) \leqslant 1/\alpha_\varepsilon^2 \text{ and } w(e) = j\}$*. When* $\vec{x}$ *is the output of WeightedM-or-E*(), then these correspond to the integral and fractional parts of* $\vec{x}$*.*

▶ **Property 13.** *Let* G *be any multigraph, and let* $\vec{x}$ *be a fractional matching of* G *that is returned by WeightedM-or-E*(). Then, for any pair of vertices,* u, v*, we have the following properties, which are a consequence of Lemma 9(1a) and Lemma 9(1b):*
1. *Either* $D(u,v) \cap supp(\vec{x}^i) \neq \emptyset$ *or* $D(u,v) \cap supp(\vec{x}^f) \neq \emptyset$*, but not both.*
2. *For any* u, v, j *such that* $D_j(u,v) \cap supp(\vec{x}) \neq \emptyset$*, we have* $D_j(u,v) \subseteq supp(\vec{x})$*.*

Thus, the supports of $\vec{x}^i$ *and* $\vec{x}^f$ *are vertex disjoint.*

▶ **Property 14.** *Let* $\vec{x}$ *be a fractional matching returned by WeightedM-or-E*(), consider* $\vec{z} = \vec{x}^i$*. Then,* $\vec{z}^C$ *is an integral matching. This is implied by Lemma 9(1).*

5 Building Blocks for WeightedM-or-E*()

In this section, we focus on stating the building blocks we need for Lemma 9. The algorithm WeightedM-or-E*() will proceed in phases. In Phase 1, we would like to determine if a given graph G, with weight function w and capacity function κ on the edges, has the

property that $\mathsf{mwm}(G,\kappa) \geqslant (1-\varepsilon)\cdot\mathsf{mwm}(G)$. We give the structural theorem related to this in Section 5.1. If it is indeed the case that $\mathsf{mwm}(G,\kappa) \geqslant (1-\varepsilon)\cdot\mathsf{mwm}(G)$, then the algorithm proceeds to Phase 2, and in this phase we want to find such a fractional matching. We describe the main structural theorem of this phase in Section 5.2. In Phase 2, there is an additional computational question, that of finding a $(1-\varepsilon)$-approximate maximum weight fractional matching efficiently. We give an algorithm for this in Section 5.4. Finally, if in Phase 1 we know that $\mathsf{mwm}(G,\kappa) < (1-\varepsilon)\cdot\mathsf{mwm}(G)$, then we increase capacities along E^*. In Section 5.3, we give a characterization of these edges. We start with stating some probabilistic tools that we will need, for example the Chernoff Bound:

▶ **Lemma 15** ([17]). *Let $X_1, X_2, \cdots, X_k$ be k negatively associated random variables with $0 \leqslant |X_i| \leqslant M$, and let $X = \sum_{i=1}^{k} X_i$. Suppose $\mu = \mathbb{E}[X]$, and $\mu_{min} \leqslant \mu \leqslant \mu_{max}$. Then, we have, for $\delta > 0$,*

$$\Pr\left(X \geqslant (1+\delta)\cdot\mu_{max}\right) \leqslant \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\frac{\mu_{max}}{M}}$$

$$\Pr\left(X < (1-\delta)\cdot\mu_{min}\right) \leqslant \exp\left(-\frac{\delta^2\cdot\mu_{min}}{(2+\delta)\cdot M}\right)$$

Additionally, we will also need Bernstein's inequality.

▶ **Lemma 16.** *Let X be the sum of negatively associated random variables $X_1, \cdots, X_k$ with $X_i \in [0, M]$ for each $i \in [k]$. Then, for $\sigma^2 = \sum_{i=1}^{k} \mathrm{Var}[X_i]$ and all $a > 0$,*

$$\Pr\left(X > \mathbb{E}[X] + a\right) \leqslant \exp\left(\frac{-a^2}{2\left(\sigma^2 + {}^{a\cdot M}/_{3}\right)}\right)$$

5.1 Phase 1 of WeightedM-or-E*()

Recall that we used $\mathsf{mwm}(G,\kappa)$ to denote the weight of the maximum weight fractional matching of G obeying κ as well as the odd set constraints. As in the congestion balancing step, we want to estimate $\mathsf{mwm}(G,\kappa)$. In the bipartite case, because of the correspondence between flows and fractional matchings, one could do this relatively easily. But in non-bipartite graphs, we don't have this correspondence.

In this section, we show the following structural lemma: if in a weighted graph G, we sample every edge with probability $p(e) = \min\{1, \kappa(e)\cdot\rho_\varepsilon\}$ to create a graph G_s, then $\mathsf{mwm}(G_s) \geqslant \mathsf{mwm}(G,\kappa) - \varepsilon\cdot\mathsf{mwm}(G)$. Thus, now we can estimate $\mathsf{mwm}(G_s)$ (and consequently, $\mathsf{mwm}(G,\kappa)$) by using existing results (such [16]). We remark that [3] proved the unweighted version of this lemma, but their proof strategy relied on showing that with high probability, G_s does not contain a Tutte set causing a high deficiency (see Lemma 28 in [3]).

▶ **Lemma 17.** *Let G be an integer weighted multigraph with weights in $\{1, 2, \cdots, W\}$, and with $\mathsf{mwm}(G) \geqslant \max\{{}^{\varepsilon\cdot n}/_{16\cdot W}, {}^{\log n}/_{\varepsilon^4}\}$, where $\varepsilon \in (0, {}^1/_2)$. Let κ be a capacity function on the edges of the graph, and let G_s be obtained by sampling every edge e independently with probability $p(e) = \min\{1, \kappa(e)\cdot\rho_\varepsilon\}$. Then, with high probability, $\mathsf{mwm}(G_s) \geqslant \mathsf{mwm}(G,\kappa) - \varepsilon\cdot\mathsf{mwm}(G)$.*

Proof. Let $\vec{x}$ denote the fractional matching that realizes $\mathsf{mwm}(G,\kappa)$. In order to prove the statement, we will construct a vector $\vec{z}$ in the support of G_s. This vector $\vec{z}$ will have the following properties: it will satisfy the fractional matching constraints and small blossom constraints with high probability. It will also have the property that $\sum_{e\in E)} w(e)\cdot z(e) \geqslant (1-\varepsilon)\cdot\sum_{e\in E} w(e)\cdot x(e) \geqslant \mathsf{mwm}(G,\kappa) - \varepsilon\cdot\mathsf{mwm}(G)$ with high probability as well.

Since with high probability G_s, contains a fractional matching $\vec{z}$ satisfying the above properties, there is an integral matching M in support of G_s with $w(M) \geqslant (1-\varepsilon)\cdot\sum_{e\in E} w(e)\cdot z(e) \geqslant \mathsf{mwm}(G,\kappa) - \varepsilon\cdot\mathsf{mwm}(G)$ (see Observation 8).

Let X_e denote the indicator random variable of the event $e \in G_s$. We define $\vec{z}$ as follows:

$$z(e) = X_e \cdot \frac{x(e)}{\min\{1, \kappa(e)\cdot\rho_\varepsilon\}}\cdot(1-\varepsilon)$$

Note that $\{z(e)\}_{e\in E}$ are independent random variables, and, $\mathbb{E}\left[\sum_{e\in E} z(e)\cdot w(e)\right] = (1-\varepsilon)\cdot\sum_{e\in E} w(e)\cdot x(e)$. Further, consider any e with $\kappa(e) \geqslant \frac{1}{\rho_\varepsilon}$, then $w(e)\cdot z(e) = w(e)\cdot x(e)$ always. Thus, we focus on e with $\kappa(e) < 1/\rho_\varepsilon$. We first have the following observation.

▶ **Observation 18.** *Note that,* $\sum_{e\in E:\kappa(e)<1/\rho_\varepsilon} w(e)\cdot x(e) \geqslant \varepsilon\cdot\mathsf{mwm}(G)$. *Otherwise,* $\sum_{e\in E} z(e)\cdot w(e) \geqslant (1-\varepsilon)\cdot\sum_{e\in E} x(e)\cdot w(e) - \varepsilon\cdot\mathsf{mwm}(G)$ *with probability* 1.

Using Chernoff bound with $M = \frac{W}{\rho_\varepsilon}$ and $\mu_{\min} = \varepsilon\cdot\mathsf{mwm}(G)$ (see Lemma 15), we have,

$$\Pr\left(\sum_{e\in E:\kappa(e)<1/\rho_\varepsilon} z(e)\cdot w(e) \leqslant \sum_{e\in E:\kappa(e)<1/\rho_\varepsilon} w(e)\cdot x(e) - 2\cdot\varepsilon^2\cdot\mathsf{mwm}(G)\right) = \exp\left(\frac{-\varepsilon^4\cdot\mathsf{mwm}(G)\cdot\rho_\varepsilon}{2\cdot W}\right)$$

By assumption, $\mathsf{mwm}(G) \geqslant 100\cdot\log n/\varepsilon^4$, we have that the above claim holds with probability at least $1 - O(1/n^{1/\varepsilon})$. We will now show that it obeys fractional matching constraints with high probability as well. Consider an edge e with $\kappa(e) > 1/\rho_\varepsilon$, we know that $e \in G_s$. Consequently, we have $\mathrm{Var}[z(e)] = 0$ for such an edge, since $z(e) = x(e)$ always. For any edge with $\kappa(e) < 1/\rho_e$, we have, $\mathrm{Var}[z(e)] \leqslant (1-2\varepsilon)\cdot\frac{x(e)}{\rho_\varepsilon}$. Since $z(e)$ are independent random variables, we have $\mathrm{Var}\left[\sum_{v\ni e} z(e)\right] = \sum_{v\ni e}\mathrm{Var}[z(e)] \leqslant (1-2\varepsilon)\cdot\rho_\varepsilon^{-1}$. Moreover, we know that $\mathbb{E}\left[\sum_{e\ni v} z(e)\right] \leqslant (1-\varepsilon)$. Thus, we want to compute the probability of the event that $\sum_{e\ni v} z(e) \geqslant 1$. Note that, in order to do this, it is sufficient to consider the edges e for which $\kappa(e) < \rho_\varepsilon^{-1}$, since for edges other than this, $z(e) = x(e)$. Thus, by Lemma 16, with $a = \varepsilon$, $M = \rho_\varepsilon^{-1}$, $\Pr\left(\sum_{e\ni v} z(e) > 1\right) = O\left(\frac{1}{n^{1/\varepsilon}}\right)$. ◀

Taking a union bound over all vertices in the graph, we have our claim. Next, we want to compute the probability that $\vec{z}$ also satisfies small odd set constraints. To see this, consider any odd set B such that $|B| \leqslant 1/\varepsilon$. We know that by definition of $\vec{x}$, we have, $\mathbb{E}\left[\sum_{e\in G[B]} z(e)\right] \leqslant (1-\varepsilon)\cdot\frac{|B|-1}{2}$. We can bound variance as well, $\mathrm{Var}\left[\sum_{e\in G[B]} z(e)\right] \leqslant (1-\varepsilon)\cdot\frac{|B|-1}{2}\cdot\frac{1}{\rho_\varepsilon}$. Consider the following subclaim.

▶ **Observation 19.** *Let* B *be any odd set with* $3 \leqslant |B| \leqslant 1/\varepsilon$. *Then,* $\frac{|B|-1}{2} \geqslant (1-\varepsilon)\cdot\frac{|B|-1}{2} + \varepsilon$.

Let $\mathcal{E}_B$ be the event that $\sum_{e\in G[B]} z(e) \geqslant \frac{|B|-1}{2}$, then from the above observation, and Lemma 16, $\Pr(\mathcal{E}_B) = O\left(\frac{1}{n^{1/\varepsilon^2}}\right)$. The second equality follows from the fact that we are considering small blossoms, that is, $|B| \leqslant 1/\varepsilon$. Taking a union bound over all small blossoms, we have our claim.

5.2 Phase 2 of WeightedM-or-E*()

The algorithm WeightedM-or-E*() proceeds to Phase 2 only if in Phase 1, if $\mathsf{mwm}(G,\kappa) \geqslant (1-\varepsilon)\cdot\mathsf{mwm}(G)$ (Lemma 17). In Phase 2, we now want to construct a good fractional matching: it should be close to $\mathsf{mwm}(G,\kappa)$ and also satisfy odd set constraints. The fractional

matching will be inspired by Observation 7 and is constructed as follows: Suppose M is a matching in G_s with weight at least $(1-\varepsilon)\cdot \mathsf{mwm}(G_s)$. Then, we will split up M into high-capacity part, M_H, and low capacity part M_L. We can introduce some slack in the congestion balancing framework, and round up the capacities of M_H. We will then compute a $(1-\varepsilon)$-approximate maximum weight fractional matching $\vec{f}$ on the low-capacity edges of $G[V(M_L)]$. Since these capacities are small, we can accomplish by computing a fractional matching in the bipartite double cover of $G[V(M_L)]$. For now, in this section, we will show that: $w(M_H) + \sum_{e\in E} f(e)\cdot w(e) \geqslant \mathsf{mwm}(G_s) - \varepsilon\cdot \mathsf{mwm}(G)$.

▶ **Definition 20.** *Let G be a multigraph, we define $E_L = \{e \in E \mid e \in D_i(u,v), \kappa(D_i(u,v)) \leqslant 1/\alpha_\varepsilon^2\}$. Intuitively, these correspond to the low capacity edges. Similarly, we define $\kappa^+(e) = \kappa(e)\cdot\alpha_\varepsilon$.*

[3] proved an unweighted version of this lemma, but their proof strategy was based on considering the bipartite double cover of G_s an (G,κ), and analysing the size of the Hall set in G_s. We take a different approach based on the dual programs of $\mathsf{mwm}(G_s)$ and $\mathsf{mwm}(G,\kappa)$. We believe our proof is arguably simpler.

▶ **Lemma 21.** *Let G be a weighted multigraph with maximum edge weight W. Suppose $\mathsf{mwm}(G) \geqslant \max\{\varepsilon\cdot n/16\cdot W, \log n/\varepsilon^4\}$ Then, with high probability, for all $X \subseteq V$, we have*

$$\mathsf{mwm}(G_s[X]) \leqslant \mathsf{mwm}(G[X]\cap E_L, \kappa^+) + \varepsilon\cdot\mathsf{mwm}(G)$$

Proof. Consider a fixed X, and from now on, we use $H \coloneqq G[X]\cap E_L$ and $H_s \coloneqq G_s[X]\cap E_L$. To prove this, we will make use of a primal-dual argument. We state the linear program for $\mathsf{mwm}(H,\kappa^+)$, and its dual.

$$\begin{aligned}
&\text{maximize} && \sum_{e\in E(H)} w(e)\cdot x(e) && \\
&\text{subject to} && && \\
&\sum_{e\ni v, e\in E(H)} x(e) && \leqslant 1 && \forall v \in X \\
&\sum_{e\in G[B]\cap E(H)} x(e) && \leqslant \tfrac{|B|-1}{2} && \forall B \in X_{\text{odd}} \\
&x(e) && \leqslant \kappa^+(e) && \forall e \in E(H)
\end{aligned}$$

The corresponding dual program is,

$$\begin{aligned}
&\text{minimize} && f(y,z,r) = \sum_{u\in V} y(u) + \sum_{B\subseteq X_{\text{odd}}} r(B)\cdot\left(\frac{|B|-1}{2}\right) + \sum_{e\in E(H)} z(e)\cdot\kappa^+(e) && \\
&\text{subject to} && && \\
& && y(u)+y(v)+z(e)+\sum_{B\in X_{\text{odd}}:(u,v)\in G[B]} r(B) \geqslant w(e) && \\
& && && \forall e\in E(H) \text{ between } u,v, \forall u,v\in X
\end{aligned}$$

By strong duality, we know that $\mathsf{mwm}(H,\kappa^+) = f(y,z,r)$ for optimal $\vec{y},\vec{z},\vec{r}$. Similarly, for the uncapacitated graph H_s we have the same primal and dual programs, except we don't have the third constraint in the primal program, and in the dual program we omit the z variables.

$$\begin{aligned}
&\text{maximize} && \sum_{e\in E(H_s)} w(e)\cdot x'(e) && \\
&\text{subject to} && && \\
&\sum_{e\ni v, e\in H_s} x'(e) && \leqslant 1 && \forall v\in X \\
&\sum_{e\in G[B]\cap E(H_s)} x'(e) && \leqslant \tfrac{|B|-1}{2} && \forall B\in X_{\text{odd}}
\end{aligned}$$

The corresponding dual program is as follows:

$$\begin{aligned}
&\text{minimize} && \sum_{u\in V} y'(u) + \sum_{B\subseteq X_{\text{odd}}} r'(B)\cdot\left(\frac{|B|-1}{2}\right) && \\
&\text{subject to} && && \\
&y'(u)+y'(v)+ && \sum_{B\in X_{\text{odd}}:(u,v)\in G[B]} r'(B) \geqslant w(e) && \forall e\in E(H_s) \text{ between } u,v, \forall u,v\in X
\end{aligned}$$

Let $g(y',r')$ denote the optimal for dual program corresponding to H_s. Note that this is a random variable, since H_s is a random graph. We will show that with high probability,

$$g(y',r') \leqslant f(y,r,z) + \varepsilon\cdot \mathsf{mwm}(G)$$

By duality, we have, $\mathsf{mwm}(H_s)\leqslant g(y',r')$, and $f(y,z,r)=\mathsf{mwm}(H,\kappa^+)$. This will show our claim for a fixed H, and then we take a union bound over all H.

We will use $\left\{\{y(u)\}_{u\in X}, \{r(B)\}_{B\in X_{\text{odd}}}\right\}$ to get a solution for the dual program for H_s. We will refer to this set of dual variables as an attempted cover. Let $E'=\{e \mid z(e)>0\}$. Observe that the edges left uncovered by attempted cover of H_s are precisely a subset of E'. We now modify the cover as follows.

$$\Delta y(u) = \sum_{e\ni v, e\in H_s} z(e), \text{ and, } y'(u) = y(u)+\Delta y(u)$$

Note that $\left\{\{y'(u)\}_{u\in X}, \{r(B)\}_{B\in X_{\text{odd}}}\right\}$ is a valid cover of H_s. To see this, consider edge $e\in H_s$ and let u,v be the endpoints of H_s. Suppose $e\notin E'$, then, $w(e)\leqslant y'(u)+y'(v)+\sum_{e\in G[B]} r(B)$. Similarly, for an edge $e\in E'$, we have, $w(e)\leqslant y'(u)+y'(v)+\sum_{e\in G[B]} r(B)$. Moreover, $g(y',r')=f(y,r,z)+\sum_{u\in X}\Delta y(u)$. Thus, it is sufficient to bound $\sum_{u\in X}\Delta y(u) = 2\cdot\sum_{e\in E'} z(e)\cdot X_e$, where X_e is the indicator variable that takes value 1 if $e\in H_s$, and 0 otherwise. Note that $\mathbb{E}\left[\sum_{e\in E'} z(e)\cdot X_e\right] \leqslant \mathsf{mwm}(H,\kappa^+)\cdot 2^{-W^2/\varepsilon^3}$. Using Chernoff and choosing $\delta = \frac{\varepsilon\cdot\mathsf{mwm}(G)\cdot 2^{W^2/\varepsilon^3}}{\mathsf{mwm}(H,\kappa^+)}$ and noting that $M=W$, we have,

$$\Pr\left(\sum_{e\in E'} z(e)\cdot X_e \geqslant 2\cdot\varepsilon\cdot\mathsf{mwm}(G)\right) \leqslant \exp\left(-\varepsilon\cdot\mathsf{mwm}(G)\cdot\frac{W}{\varepsilon^2}\right)$$

By assumption, $n\leqslant \frac{\mathsf{mwm}(G)\cdot W}{\varepsilon}$, thus taking a union bound over all subsets, we have our theorem. ◀

5.3 Phase 3: Finding Set E*

The algorithm WeightedM-or-E*() proceeds to Phase 3, if $\mathsf{mwm}(G, \kappa)$ is not large enough (see Lemma 17). At this point we have to increase capacity along some edges in the graph. Recall Lemma 9(2), the properties required of such edges. Phase 3 finds such edges E^*, and makes sure it has that this set has the property that $\sum_{e \in E^*} w(e) \cdot \kappa(e) = O(\mathsf{mwm}(G) \log n)$, and moreover, every good matching in G has significant weight going through E^*.

Similar to the case of [3], we will use the dual variables of G_s to describe E^*. We will first start with describing the dual program corresponding to the general matching LP:

$$\begin{aligned} \text{minimize} \quad & \sum_{u \in V} y(u) + \sum_{B \subseteq V_{\text{odd}}} r(B) \cdot \left(\frac{|B| - 1}{2} \right) \\ \text{subject to} \quad & y(u) + y(v) + \sum_{B : (u,v) \in G[B]} r(B) \geqslant w(u,v) \quad \forall (u,v) \in E \end{aligned}$$

Additionally, we define $yr(e)$ to be the dual constraint corresponding to the edge e, and$f(y, r)$ to be the value of the objective function. We now describe some properties of Static-Weighted-Match(H, ε). This is the algorithm whose properties we use to describe E^*. We state these properties without proof for now and postpone the proof to the full version.

▶ **Lemma 22** ([16]). *There is an $O(m/\varepsilon \cdot \log 1/\varepsilon)$ time algorithm Static-Weighted-Match() that takes as input, a weighted graph G, and a parameter $\varepsilon > 0$, and outputs an integral matching M, and dual vectors $\vec{y}$ and $\vec{r}$ with the following properties.*

1. *It returns an integral matching M such that $w(M) \geqslant (1 - \varepsilon) \cdot \mathsf{mwm}(G)$*
2. *A set Ω of laminar odd-sized sets such that $\{B \in V_{odd} \mid r(B) > 0\} \subseteq \Omega$.*
3. *For all odd-sized sets B such that $|B| \geqslant 1/\varepsilon + 1$, $r(B) = 0$.*
4. *For all $v \in V$, $y(v)$ is an integral multiple of ε, and for all $B \in V_{odd}$, $r(B)$ is an integral multiple of ε.*
5. *For each edge $e \in E$, we have $yr(e) \geqslant (1 - \varepsilon) \cdot w(e)$, that is e is* approximately covered *by $\vec{y}$ and $\vec{r}$.*
6. *The value of the dual objective, $f(y, r)$ is at most $(1 + \varepsilon) \cdot \mathsf{mwm}(G)$.*

We now state our first claim, which in Section 6 will help us show that when $\mathsf{mwm}(G, \kappa) < (1 - 2\varepsilon) \cdot \mathsf{mwm}(G)$, then every large matching in G has a lot of weight in E^*.

▷ Claim 23. Suppose $H \subseteq G$, and $\vec{y}, \vec{r}$ are the dual vectors returned by Static-Weighted-Match(H, ε). Let $E_H = \{e \in E \mid yr(e) \geqslant (1 - \varepsilon) \cdot w(e)\}$. Let M be any matching of G, then $w(M \cap E \setminus E_H) \geqslant w(M) - (1 + \varepsilon)^2 \cdot \mathsf{mwm}(H)$.

Proof. Observe that if we scale up the dual variables $\vec{y}$ and $\vec{r}$ by $(1 + \varepsilon)$, then $\vec{y}$ and $\vec{r}$ is a feasible solution for the dual matching problem for the graph E_H. Thus, by weak duality and Lemma 22(6), we have $\mathsf{mwm}(E_H) \leqslant (1 + \varepsilon) \cdot f(y, r) \leqslant (1 + \varepsilon)^2 \cdot \mathsf{mwm}(H)$. Thus, we have: $w(M) \leqslant (1 + \varepsilon)^2 \cdot \mathsf{mwm}(H) + w(M \cap E \setminus E_H)$. ◁

We now describe the set E^*, and show that $w(\kappa(E^*)) = O(\mathsf{mwm}(G) \cdot \log n)$ with high probability.

▶ **Lemma 24.** *Let G be a multigraph such that $\mathsf{mwm}(G) \geqslant \varepsilon n / W$, and let κ be the capacity function on the edges of the graph. Suppose G_s is the graph obtained by sampling edge e with probability $p(e) = \min\{1, \rho_\varepsilon \cdot \kappa(e)\}$. Let $\vec{y}, \vec{r}$ be the duals returned by Static-Weighted-Match(G_s, ε). Let $E^* = \{e \mid yr(e) < (1 - \varepsilon) \cdot w(e)\}$, then with high probability, $w(\kappa(E^*)) = O(\mathsf{mwm}(G) \cdot \log n)$.*

Proof. We consider the set $\mathcal{D}$ of duals $\vec{y}, \vec{r}$ that satisfy the following properties.

1. For all $v \in V$, $y(v)$ is a multiple of ε, and for all $B \in V_{\text{odd}}$, $r(B)$ is a multiple of ε.
2. Let $\Omega = \{B \mid r(B) > 0\}$, then Ω is laminar.
3. If $r(B) > 0$ for some B, then $|B| \leqslant 1/\varepsilon$.

Observe that $\mathcal{D}$ contains all possible duals that could be returned by Static-Weighted-Match(). Now, we bound $|\mathcal{D}|$.

$$|\mathcal{D}| \leqslant \sum_{i=0}^{2n} \binom{n^{1/\varepsilon}}{i} \cdot \left(\frac{W}{\varepsilon}\right)^{n} \cdot \left(\frac{W}{\varepsilon}\right)^{2n} \leqslant 2^{10/\varepsilon^3 \cdot n \cdot \log W} \leqslant 2^{10/\varepsilon^3 \cdot \frac{\mathsf{mwm}(G)}{W} \cdot \log W}$$

This follows from the following argument. Since Ω is laminar (due to 2), there are at most $2n$ sets contained in Ω, and from 3, we can conclude that these laminar sets are chosen from among $n^{1/\varepsilon}$ sets. Similarly, 1 suggests that each $y(v)$ and chosen $r(B)$ can be assigned W/ε values, thus for a given choice of Ω, there are at most $\left(\frac{W}{\varepsilon}\right)^{n} \cdot \left(\frac{W}{\varepsilon}\right)^{2n}$ choices for $\vec{y}$ and $\vec{r}$. We can derive the last set of equations by the premise of our lemma: that $\mathsf{mwm}(G) \geqslant \varepsilon n/W$. Additionally, observe that $\mathcal{D}$ includes the set of duals that can be returned by G, and therefore $|\mathcal{D}|$ is an upper bound on the set of possible duals returned by Static-Weighted-Match().

Now, consider any $\vec{y}, \vec{r} \in \mathcal{D}$ and let $E_{y,r} = \{e \mid yr(e) < (1-\varepsilon) \cdot w(e)\}$. Observe that for all $e \in E_{y,r}$, we have $\kappa(e) < 1/\rho_\varepsilon$, since otherwise $\kappa(e) = 1$, and e would be included in G_s with probability 1 and be approximately covered by $\vec{y}, \vec{r}$ (by Lemma 22(5)). Thus, for all $e \in E_{y,r}$, we have $\kappa(e) < 1/\rho_\varepsilon$. Now, $\vec{y}, \vec{r}$ are such that $w(\kappa(E_{y,r})) > \mathsf{mwm}(G) \cdot \log n$, then none of the edges in $E_{y,r}$ were sampled. Note that in this case, $\kappa(E_{y,r}) \geqslant \mathsf{mwm}(G) \cdot \log n \cdot W^{-1}$. Let $\mathcal{E}_{y,r}$ denote the event that $\vec{y}, \vec{r}$ is returned by Static-Weighted-Match(G_s, ε) and $\mathcal{E}_2$ be the event that $E_{y,r}$ is not sampled into G_s. Then, observe that in this case,

$$\Pr(\mathcal{E}_{y,r}) \leqslant \Pr(\mathcal{E}_2) \leqslant \prod_{e \in E_{y,r}} (1 - p(e)) \leqslant \exp\left(-\sum_{e \in E_{y,r}} \kappa(e) \cdot \rho_\varepsilon\right) \leqslant \exp\left(-\frac{\mathsf{mwm}(G)}{W} \cdot \rho_\varepsilon\right)$$

Now, we want to upper bound the probability that Static-Weighted-Match(G_s, ε) outputs $\vec{y}, \vec{r}$ such that $w(\kappa(E_{y,r})) \geqslant \mathsf{mwm}(G) \cdot \log n$. To do this, we take a union bound over all $|\mathcal{D}|$ many possible duals. From the previous discussion we know that $|\mathcal{D}| \leqslant \exp\left(\frac{10}{\varepsilon^2} \cdot \frac{\mathsf{mwm}(G)}{W} \cdot \log W\right)$. This concludes the proof. ◀

5.4 Weighted Fractional Matching in Bipartite Graphs

The final ingredient we need for WeightedM-or-E*() is the following lemma, which computes a fractional matching on low capacity edges in bipartite graphs. We will show in Section 6 how to use this as a subroutine to do the same in general graphs.

▶ **Lemma 25.** *Consider a weighted bipartite multigraph G with the following property: for any $u, v \in V$ with $e \neq e'$ between u, v, then $w(e) \neq w(e')$. Suppose κ is the capacity function on the edges of the graph, and suppose the edge weights are in $\{1, 2, \cdots, W\}$. There is an algorithm that in $O(m \cdot W \cdot \log n \cdot 1/\varepsilon)$ time finds a fractional matching $\vec{x}$ obeying the capacity function κ such that $\sum_{e \in E} w(e) \cdot x(e) \geqslant (1-\varepsilon) \cdot \mathsf{mwm}(G, \kappa)$.*

Note that there are algorithms in the literature, which can compute weighted fractional matchings in bipartite graphs: such the one by [1]. However, like many LP solvers, one incurs additional $\log n$ factors in the time bound. In contrast, our algorithm incurs a W factor, but since we use Lemma 3, in the final algorithm, we will only incur a $\log W$ factor for the entire decremental algorithm. In contrast, using LP solvers, we will incur additional

$\log n$ factors for implementing WeightedM-or-E*() alone. In order to come up with the algorithm for Lemma 25, we first recall the maximum weight capacitated fractional matching linear program for **bipartite graphs**, and its corresponding dual.

$$\begin{aligned} &\text{maximize} \sum_{e\in E} w(e)\cdot x(e) \\ &\text{subject to} \\ &\quad x(e) \leqslant \kappa(e) && \forall e \in E \\ &\quad \sum_{e\ni v} x(e) \leqslant 1 && \forall v \in V \end{aligned}$$

$$\begin{aligned} &\text{minimize} \sum_{v\in L} y(v) + \sum_{u\in R} y(u) + \sum_{e\in E} z(e)\cdot\kappa(e) \\ &\text{subject to} \\ &\quad y(u)+y(v)+z(e) \geqslant w(e) && \forall e \in E \end{aligned}$$

Here, $yz(e) = y(u) + y(v) + z(e)$, where u, v are the two endpoints of e.

Our algorithm will be a scaling algorithm, which essentially reduces the problem of finding $(1-\varepsilon)$-approximate **maximum weight fractional matching** to the problem of finding a maximal fractional matching, which can be easily accomplished using known subroutines (see [24, 9, 3]). Before stating our algorithm, we give some definitions.

▶ **Definition 26** (Residual Graph)**.** *Given a bipartite graph $G = (L, R, E)$, with a capacity function κ on the edges of the graph. Let $\vec{x}$ be a fractional matching obeying κ. We define G_x to be the residual graph with respect to $\vec{x}$. In particular, this is a directed graph, where corresponding to each undirected edge $e \in G$, there are two directed edges, e_f (forward edges, directed from L to R), and e_b (backward edge, directed from R to L). Moreover, e_f has a residual capacity of $\kappa(e) - x(e)$, and e_b has a capacity of $x(e)$. Thus, if an edge e is saturated, that is $x(e) = \kappa(e)$, then e_f is not in G_x. Similarly, if $x(e) = 0$, then e_b is not in G_x.*

▶ **Definition 27** (Free Vertices)**.** *Given a graph G, and a fractional matching $\vec{x}$, we say that a vertex v is free with respect to $\vec{x}$ if $\sum_{e\ni v} x(e) < 1$.*

▶ **Definition 28.** *An augmenting path in G_x, will refer to a path starting and ending in a free vertex, and all the intermediate vertices on this path will be saturated.*

Before stating our algorithm, we will state the invariants that the algorithm will maintain. Subsequently, we will show that maintaining these invariants throughout implies that the algorithm will output a matching satisfying the approximation guarantees of Lemma 25 and in the desired runtime.

▶ Remark 29. We round down ε so that $1/\varepsilon$ is an integer.

▶ **Property 30.** *Our algorithm will at all times maintain, a fractional matching $\vec{x}$, and the dual variables $y(u)$ for all $u \in V$, and $z(e)$, for all $e \in E$ with the following properties.*

1. ***Granularity:*** *At all times, $y(u)$ for all $u \in V$, and $z(e)$ for all $e \in E$, are integral multiples of ε.*
2. ***Domination:*** *For all directed edges e_f and e_b in G_x, we have $yz(e) \geqslant w(e) - \varepsilon$.*
3. ***Tightness:*** *For all backward edges $e_b \in G_x$, we additionally have the property that $yz(e) \leqslant w(e) + \varepsilon$.*
4. ***Free Duals:*** *The $y(u)$ for free vertices $u \in L$ are equal, and are at most $y(v)$ for $v \in L \setminus Z$. The free vertex duals in R are always 0. The algorithm terminates when the free vertex duals in L are all 0.*
5. ***Complementary Slackness:*** *If $z(e) > 0$ for some $e \in E$, then $x(e) = \kappa(e)$.*

▶ **Definition 31** (Eligible Edges)**.** *Let $\vec{x}$ be the current fractional matching maintained by the graph, and let G_x denote the residual graph with respect to $\vec{x}$. A forward edge e_f is eligible if $yz(e) = w(e) - \varepsilon$, and backward edge e_b is said to be eligible if $yz(e) = w(e) + \varepsilon$. We use G_x^t to denote the eligible subgraph of G_x.*

▶ **Lemma 32.** *The fractional matching $\vec{x}$ returned by the algorithm mentioned in Property 30 has $\sum_{e\in E} w(e)\cdot x(e) \geqslant (1-\varepsilon)\cdot \mathit{mwm}(G)$.*

Proof. In order to see this, consider the dual variables returned by the algorithm. Note that if we increase each of the duals by a factor of $(1+\varepsilon)$, then the dual solutions $\vec{y}, \vec{z}$ maintained by the algorithm are feasible dual solutions to the dual program (this is by Property 30(2)). Thus, by weak duality, we have, $\sum_{u\in V} y(u) + \sum_{e\in E} z(e)\cdot\kappa(e) \geqslant (1-\varepsilon)\cdot \mathsf{mwm}(G)$. Due to complementary slackness, (Property 30(5)), we have, $\sum_{u\in V} y(u) + \sum_{e\in E} x(e)\cdot z(e) \geqslant (1-\varepsilon)\cdot \mathsf{mwm}(G)$. Due to the fact that free vertex duals, at the end of the algorithm are zero (Property 30(4)), we have, $\sum_{u\in V}\sum_{e\ni u} x(e)\cdot y(u) + \sum_{e\in E} z(e)\cdot x(e) \geqslant (1-\varepsilon)\cdot \mathsf{mwm}(G)$. Simplifying the above expression, and using Property 30(3), that is, tightness, we have, $\sum_{e\in E}(1+4\varepsilon)\cdot w(e)\cdot x(e) \geqslant (1-\varepsilon)\cdot \mathsf{mwm}(G)$.This shows the claim. ◀

The precise algorithm is stated in Algorithm 2. We now show that Algorithm 2 proves Property 30. Later, we will show runtime guarantees of Algorithm 2. Towards this, we start with the following observation.

▶ **Observation 33.** *Let $e\in E(G)$, then e_f and e_b cannot simultaneously appear in G_x^t.*

▶ **Definition 34.** *Let H and G be directed, capacitated graphs with capacity functions c_h and c_g respectively. Then, we say $E(H)\subseteq_c E(G)$ if $e\in H$ implies $e\in G$, and $c_h(e)\leqslant c_g(e)$.*

▷ Claim 35. After Line 13, there are no augmenting paths in the graph G_x^t.

Proof. Let $\vec{y}$ denote the fractional matching after the augmentation step. Then, in order to show the claim, it is sufficient to show $E(G_y^t)\subseteq_c E(G_x^t)$. Consider any forward edge $e_f\in E(G_x^t)$, observe that $e_b\notin E(G_x^t)$ due to Observation 33. Since we don't change the duals, $e_b\notin E(G_y^t)$ as well. Finally, e_f can either be augmented along or not, in either case, we have, $\kappa(e)-x(e)\geqslant\kappa(e)-y(e)$. Such an argument applies for any edge e_b as well. From this, we can conclude that any augmenting path that is present in G_y^t is in G_x^t as well, and this contradicts the fact that we found a maximal set of augmenting paths $\mathcal{P}$. The subsequent claims follow from induction, and we prove them in the full version. ◁

▷ Claim 36 (Granularity). Throughout the algorithm, Property 30(1) holds.

▷ Claim 37 (Domination). Throughout the algorithm, Property 30(2) holds.

▷ Claim 38 (Tightness). Throughout the algorithm, Property 30(3) property holds.

▷ Claim 39 (Free Duals). Throughout the algorithm, Property 30(4) holds.

▷ Claim 40 (Complementary Slackness). Throughout the algorithm, Property 30(5) holds.

5.5 Runtime

Before proving the runtime of Algorithm 2, we start with the following structural lemma.

▶ **Observation 41.** *In G_x^t, after dual adjustment step, there are only forward edges between $L\cap Z$ and $R\setminus Z$, and backward edges between $R\cap Z$ and $L\setminus Z$ i.e. all directed edges between Z and $V\setminus Z$ go from Z to $V\setminus Z$.*

Proof. Suppose there is a backward edge e_b between $L\cap Z$ and $R\setminus Z$ after the dual adjustment step. This implies, that after the dual adjustment step, $yz(e_b)=w(e)+\varepsilon$. Thus, prior to the dual adjustment step, we had that $yz(e_b)=w(e)+2\varepsilon$. However, this would contradict the

tightness property (see Property 30(3)). Similarly, suppose there is a forward edge e_f between $R \cap Z$ and $L \setminus Z$ after the dual adjustment step. This would imply that $yz(e_f) = w(e) - \varepsilon$ after the dual adjustment step. Thus, before the dual adjustment step, $yz(e_f) = w(e) - 2\varepsilon$, which would contradict the domination property (see Property 30(2)). ◀

▶ **Lemma 42.** *The graph G_x^t that is fed into* AUGMENTATION *procedure is acyclic.*

Proof. We will show this by induction. Observe that at the beginning of the algorithm, there are no back edges, so the graph is acyclic. We now show that if no cycles are present in the graph, then dual adjustment and augmentation procedures cannot create cycles.

1. First consider 8, observe that if $e_b \in G_x^t$ is subjected to this, then $x(e) = \kappa(e)$ due to complementary slackness. Thus e_f does not exist, and cannot be added to G_x^t due to this modification. Moreover, Line 8 does not affect any other e. Thus, if the graph G_x^t was acyclic before, then it remains acyclic after Line 8.
2. Let $\vec{y}$ be the fractional matching after augmentation step. As we saw in Claim 35, $E(G_y^t) \subseteq_c E(G_x^t)$. Thus, if G_x^t is acyclic, then G_y^t is acyclic as well.
3. Now consider Line 17. By induction hypothesis, there are no cycles in G_x^t before Line 17 was executed. Suppose a cycle C exists in G_x^t after this step, then C contains either e_f or e_b, where e is an edge subjected to Line 17. However, for e_b, $\kappa(e) = x(e)$, and therefore, e_f cannot exist in G_x^t. Moreover, $e_b \notin G_x^t$ since before the dual modification $yz(e) = w(e) - \varepsilon$, and after dual adjustment, $yz(e) = w(e)$.
4. By induction hypothesis, there are no cycles in G_x^t before Line 18 and Line 19. These steps preserve eligibility and ineligibility status of any edge between $L \cap Z$ and $R \cap Z$ and those between $L \setminus Z$ and $R \setminus Z$. Thus, if a new cycle is created after Line 18 and Line 19, then it must contain a directed edge from Z to $V \setminus Z$, and from $V \setminus Z$ to Z. However, Observation 41 suggests that after this step, the latter are not in G_x^t. ◀

▶ **Observation 43.** *Property 30(4) holds. By Line 18, at the end of each iteration of the while loop, the free duals in $L \cap Z$ drops by ε. Therefore, total number of iterations is $O(\frac{W}{\varepsilon})$.*

▶ **Lemma 44** ([24])**.** *For G_x^t acyclic, maximal augmenting paths can be found in time $O(m \log n)$.*

▶ **Lemma 45.** *The total runtime of the algorithm is $O(m/\varepsilon \cdot W \cdot \log n)$.*

Proof. In a particular iteration of a while loop, we see how each of the steps contribute to the runtime. Consider step Line 8, this takes time $O(m)$ and similarly, the subsequent step of updating G_x^t takes time $O(m)$ as well. Since G_x^t is acyclic (by Lemma 42), we can use Lemma 44 to find the maximal set of augmenting paths in time $O(m \cdot \log n)$. Finally, the dual adjustment steps can also be accomplished in time $O(m)$. From Observation 43 implies the total runtime is $O(m/\varepsilon \cdot W \cdot \log n)$. ◀

6 Algorithm WeightedM-or-E*()

In this section, we now put everything together and describe the algorithm WEIGHTEDM-OR-E*(). Then, we describe how to use WEIGHTED-FRAC-MATCH() to get fractional matchings obeying capacity, and odd set constraints in general graphs.

▶ **Definition 46.** *Given a matching M, define $E_L^M(G, \kappa) = E_L(G, \kappa) \cap M$, and V_L^M to be the set of vertices that are the endpoints of $E_L^M(G, \kappa)$.*

▶ **Definition 47.** *Given a multigraph G with weight function w and capacity function κ on the edges of G, we define the bipartite double cover of G, denoted $\text{BC}(G)$, with capacity function κ_{BC} and weight function w_{BC} as follows:*

1. *For every vertex $v \in V(G)$, make two copies v and v' in $V(\text{BC}(G))$.*
2. *If e is an edge between u, v, then we add an edge e' between u and v' and an edge e'' between u' and v. Moreover, $w_{BC}(e') = w_{BC}(e'') = w(e)$ and $\kappa_{BC}(e') = \kappa_{BC}(e'') = \kappa(e)$.*

▶ **Observation 48.** *For weighted graph G, with capacity κ, $\mathsf{mwm}(\text{BC}(G), \kappa_{BC}) \geqslant 2 \cdot \mathsf{mwm}(G, \kappa)$.*

▶ **Lemma 49.** *Given a weighted multigraph G (possibly non-bipartite) with capacity function κ, $\varepsilon \in (0, 1)$, with edge weights from $\{1, 2, \cdots, W\}$, and with the property that for all $i \in [W]$, and for all vertices $u, v \in V$ we have, $\kappa(D_i((u, v))) \leqslant 1/\alpha_\varepsilon$, then there is an algorithm Weighted-Frac-Match-General() that takes as input $G, \kappa, w, \varepsilon$ and outputs a fractional matching $\vec{x}$ such that $\sum_{e \in E} w(e) \cdot x(e) \geqslant (1 - \varepsilon) \cdot \mathsf{mwm}(G, \kappa)$, and further $\vec{x}$ obeys the capacities and the odd set constraints.*

Proof. In the previous section, we saw the algorithm Weighted-Frac-Match() that solved this problem for bipartite graphs. For the case of general graphs, we proceed as follows. We collapse edges of $\text{BC}(G)$ as in Definition 6, and run Weighted-Frac-Match$(\text{BC}(G), \kappa_{\text{BC}}, w_{\text{BC}}, \varepsilon)$, and obtain a fractional matching $\vec{y}$. We let $\vec{z} = \vec{y}^C$. To translate this into a valid fractional matching $\vec{x}$ in G, do the following: for each $e \in G$, consider $e', e'' \in \text{BC}(G)$, and let $x(e) = \frac{z(e') + z(e'')}{2}$. Note that $\vec{z}$ is a fractional matching, since $\vec{x}$ is and moreover since $\vec{x}$ satisfies capacity constraints since $\vec{z}$ does. Applying Observation 48, we know that $\sum_{e \in E} w(e) \cdot x(e) \geqslant (1 - \varepsilon) \cdot \mathsf{mwm}(G, \kappa)$. Moreover, $\frac{\vec{x}}{1+\varepsilon}$ satisfies all odd set constraints since it satisfies the premise of Observation 8. ◀

We show how Algorithm 1 satisfies the conditions of Lemma 9.

Proof. (Lemma 9) We first show the runtime of the algorithm. First note that G_s can be computed in $O(m)$ time since this only involves sampling edges independently. Moreover, by Lemma 22, we know that the runtime of Static-Weighted-Match(G, ε) is $O(m/\varepsilon \cdot \log 1/\varepsilon)$. Additionally, from Lemma 24, we can conclude that we can obtain set E^* using the output of Static-Weighted-Match(G_s, ε) and the time taken to do this is $O(m)$. Finally, from Lemma 49, we can conclude that we can run Line 7 in time $O(m/\varepsilon \cdot \log 1/\varepsilon \cdot W \cdot \log n)$.

Now, we turn to show Lemma 9(1). In this case, first note that V_L^M and $V(M_I)$ are disjoint. This follows from Algorithm 1 and Definition 46. Thus, since $\vec{y}, \vec{x}$ are fractional matchings, we can conclude that $\vec{z}$ is also a fractional matching. Note that we land in Lemma 9(1) if $w(M) \geqslant (1-\varepsilon) \cdot \mathsf{mwm}(G)$. From Lemma 21 we can conclude that for $H = E_L(G, \kappa) \cap G[V_L^M]$ and $H_s = E_L(G, \kappa) \cap G_s[V_L^M]$, we have that $\mathsf{mwm}(H_s) \leqslant \mathsf{mwm}(H, \kappa^+) + \varepsilon \cdot \mathsf{mwm}(G)$. Thus, we have, $\sum_{e \in E} y(e) \cdot w(e) + \sum_{e \in E} x(e) \cdot w(e) \geqslant (1 - \varepsilon) \cdot \mathsf{mwm}(G) - 2\varepsilon \cdot \mathsf{mwm}(G)$.

We now proceed to proof Lemma 9(1a) and (1b). Consider any edge $e \in \text{supp}(\vec{z})$ with $w(e) = i$ and $\kappa(D_i(e)) \leqslant 1/\alpha_\varepsilon^2$. Thus, we know that $e \in \text{supp}(\vec{x})$, and so by Lemma 49, we have that $z(e) = x(e) \leqslant \kappa^+(e) \leqslant \kappa(e) \cdot \alpha_\varepsilon$ and $z(D_i(e)) = x(D_i(e)) \leqslant \kappa^+(D_i(e)) \leqslant \kappa(D_i(e)) \cdot \alpha_\varepsilon$. Similarly, for any $e \in \text{supp}(\vec{z})$ with $w(e) = i$, and $\kappa(D_i(e)) > 1/\alpha_\varepsilon^2$. We know that $e \in \mathsf{supp}(\vec{y})$. By definition of $\vec{y}$, we have $z(e) = y(e) = \kappa(e)/\kappa(D_i(e))$, and $z(D_i(e)) = 1$. Finally, we prove Lemma 9(2). First consider Lemma 24, this lemma implies that $\sum_{e \in E^*} \kappa(e) \cdot w(e) = O(\mathsf{mwm}(G) \cdot \log n)$. Finally, by Claim 23, we have, $\sum_{e \in M \cap E^*} w(e) \geqslant w(M) - (1+\varepsilon)^2 \cdot \mathsf{mwm}(G_s)$, thus, for any M with $w(M) \geqslant (1-\varepsilon) \cdot \mathsf{mwm}(G)$, we have $\sum_{e \in M \cap E^*} w(e) \geqslant \varepsilon \cdot \mathsf{mwm}(G)$. This is implied by the fact that the algorithm returns E^* when $\mathsf{mwm}(G_s) \leqslant (1 - 5\varepsilon) \cdot \mathsf{mwm}(G)$. Finally, for all edges in E^*, $\kappa(e) < 1$, otherwise they'd be sampled into G_s. ◀

References

1 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II*, pages 526–538, 2011.

2 Sepehr Assadi, Soheil Behnezhad, Sanjeev Khanna, and Huan Li. On regularity lemma and barriers in streaming and dynamic matching. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 131–144. ACM, 2023. doi:10.1145/3564246.3585110.

3 Sepehr Assadi, Aaron Bernstein, and Aditi Dudeja. Decremental matching in general graphs. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 11:1–11:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.11.

4 Sepehr Assadi, Sanjeev Khanna, and Yang Li. The stochastic matching problem with (very) few queries. *ACM Transactions on Economics and Computation (TEAC)*, 7(3):1–19, 2019.

5 Amir Azarmehr, Soheil Behnezhad, and Mohammad Roghani. Fully dynamic matching: (2-$\sqrt{2}$)-approximation in polylog update time. *CoRR*, abs/2307.08772, 2023. doi:10.48550/arXiv.2307.08772.

6 Soheil Behnezhad. Dynamic algorithms for maximum matching size. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 129–162. SIAM, 2023. doi:10.1137/1.9781611977554.CH6.

7 Soheil Behnezhad and Sanjeev Khanna. New trade-offs for fully dynamic matching via hierarchical EDCS. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 3529–3566. SIAM, 2022. doi:10.1137/1.9781611977073.140.

8 Aaron Bernstein, Aditi Dudeja, and Zachary Langley. A framework for dynamic matching in weighted graphs. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 668–681. ACM, 2021. doi:10.1145/3406325.3451113.

9 Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1123–1134. IEEE, 2020.

10 Sayan Bhattacharya and Peter Kiss. Deterministic rounding of dynamic fractional matchings. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.27.

11 Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic $(1+\epsilon)$-approximate matching size in truly sublinear update time. *CoRR*, abs/2302.05030, 2023. doi:10.48550/arXiv.2302.05030.

12 Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. Dynamic matching with better-than-2 approximation in polylogarithmic update time. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 100–128. SIAM, 2023. doi:10.1137/1.9781611977554.CH5.

13 Sayan Bhattacharya, Peter Kiss, Aaron Sidford, and David Wajc. Near-optimal dynamic rounding of fractional matchings in bipartite graphs. *CoRR*, abs/2306.11828, 2023. doi:10.48550/arXiv.2306.11828.

14 Joakim Blikstad and Peter Kiss. Incremental (1-ϵ)-approximate dynamic matching in o(poly(1/ϵ)) update time. In *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 22:1–22:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.22.

15 Jiale Chen, Aaron Sidford, and Ta-Wei Tu. Entropy regularization and faster decremental matching in general graphs. *arXiv preprint arXiv:2312.09077*, 2023.

16 Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM (JACM)*, 61(1):1–23, 2014.

17 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009. URL: `http://www.cambridge.org/gb/knowledge/isbn/item2327542/`.

18 Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. $(1+\epsilon)$-approximate incremental matching in constant deterministic amortized time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1886–1898. SIAM, 2019. `doi:10.1137/1.9781611975482.114`.

19 Manoj Gupta. Maintaining approximate maximum matching in an incremental bipartite graph in polylogarithmic update time. In *FSTTCS*, 2014.

20 Manoj Gupta and Richard Peng. Fully dynamic (1+ e)-approximate matchings. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 548–557, 2013.

21 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 21–30, 2015.

22 Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. Regularized box-simplex games and dynamic decremental bipartite matching. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 77:1–77:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPICS.ICALP.2022.77`.

23 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1272–1287. SIAM, 2016.

24 Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. `doi:10.1016/0022-0000(83)90006-5`.

25 Daniel Stubbs and Virginia Vassilevska Williams. Metatheorems for dynamic weighted matching. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, volume 67 of *LIPIcs*, pages 58:1–58:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. `doi:10.4230/LIPICS.ITCS.2017.58`.

26 David Wajc. Rounding dynamic matchings against an adaptive adversary. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 194–207, 2020.

A Subroutines

Algorithm 1 WeightedM-or-E*$(G, \kappa, \varepsilon, \mu, w)$.

Include each edge $e \in E(G)$ independently with probability $p(e) = \kappa(e) \cdot \rho_\varepsilon$ in G_s.
Let $M, \vec{y}, \vec{r}$ be the output of Static-Weighted-Match(G_s, ε). ▷ Phase 1.
if $w(M) \leqslant (1-6\varepsilon) \cdot \mathsf{mwm}(G)$ **then**
    Return $E^* = \{e \mid yr(e) < (1-\varepsilon) \cdot w(e)\}$. ▷ Phase 3
else ▷ Phase 2
    $M_I \leftarrow M \setminus E_L(G, \kappa)$, $\vec{y} \leftarrow M_I^D$ ▷ Convert M_I into a matching on multigraph
    $\vec{x} \leftarrow$ Weighted-Frac-Match-General$(G[V_L^M] \cap E_L(G, \kappa), \kappa^+, \varepsilon, w)$
end if
Return $\vec{z} \leftarrow \vec{y} + \vec{x}$.

Algorithm 2 WEIGHTED-FRAC-MATCHING(G, κ, ε).

Ensure: A fractional matching $\vec{x}$ with $\sum_{e \in E} w(e) \cdot x(e) \geqslant (1-\varepsilon) \cdot \mathsf{mwm}(G, \kappa)$

procedure INITIALIZATION:
  $x(e) \leftarrow 0$ for all $e \in E$
  $y(u) \leftarrow W - \varepsilon$ for all $u \in L$, $y(u) \leftarrow 0$ for all $u \in R$ ▷ Initializing vertex duals.
  $z(e) \leftarrow 0$ for all $e \in E$ ▷ Initializing edge duals.
end procedure
while y-values of free left vertices are greater than 0 **do**
  for $e_b \in G_x^t$ with $z(e) > 0$ **do**
    $z(e) \leftarrow z(e) - \min\{z(e), yz(e) - w(e) + \varepsilon\}$.
  end for
  Update G_x^t
  procedure AUGMENTATION:
    Find the maximal set $\mathcal{P}$ of augmenting paths in G_x^t.
    Augment along $\mathcal{P}$. Update $\vec{x}$ and G_x^t.
  end procedure
  procedure DUAL ADJUSTMENT:
    Let Z be the set of vertices reachable from free left vertices in G_x^t.
    For all ineligible e_b from $R \setminus Z$ to $L \cap Z$, with $yz(e) = w(e) - \varepsilon$, $z(e) \leftarrow z(e) + \varepsilon$.
    For all $u \in L \cap Z$, $y(u) \leftarrow y(u) - \varepsilon$.
    For all $u \in R \cap Z$, $y(u) \leftarrow y(u) + \varepsilon$.
  end procedure
end while

Testing C_k-Freeness in Bounded-Arboricity Graphs

Talya Eden
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Reut Levi
Efi Arazi School of Computer Science, Reichman University, Herzliya, Israel

Dana Ron
School of Electrical Engineering, Tel Aviv University, Israel

Abstract

We study the problem of testing C_k-freeness (k-cycle-freeness) for fixed constant $k > 3$ in graphs with bounded arboricity (but unbounded degrees). In particular, we are interested in one-sided error algorithms, so that they must detect a copy of C_k with high constant probability when the graph is ϵ-far from C_k-free.

We next state our results for constant arboricity and constant ϵ with a focus on the dependence on the number of graph vertices, n. The query complexity of all our algorithms grows polynomially with $1/\epsilon$.

1. As opposed to the case of $k = 3$, where the complexity of testing C_3-freeness grows with the arboricity of the graph but not with the size of the graph (*Levi, ICALP 2021*)[1] this is no longer the case already for $k = 4$. We show that $\Omega(n^{1/4})$ queries are necessary for testing C_4-freeness, and that $\widetilde{O}(n^{1/4})$ are sufficient. The same bounds hold for C_5.
2. For every fixed $k \geq 6$, any one-sided error algorithm for testing C_k-freeness must perform $\Omega(n^{1/3})$ queries.
3. For $k = 6$ we give a testing algorithm whose query complexity is $\widetilde{O}(n^{1/2})$.
4. For any fixed k, the query complexity of testing C_k-freeness is upper bounded by $O(n^{1-1/\lfloor k/2 \rfloor})$.

The last upper bound builds on another result in which we show that for any fixed subgraph F, the query complexity of testing F-freeness is upper bounded by $O(n^{1-1/\ell(F)})$, where $\ell(F)$ is a parameter of F that is always upper bounded by the number of vertices in F (and in particular is $k/2$ in C_k for even k).

We extend some of our results to bounded (non-constant) arboricity, where in particular, we obtain sublinear upper bounds for all k.

Our $\Omega(n^{1/4})$ lower bound for testing C_4-freeness in constant arboricity graphs provides a negative answer to an open problem posed by (Goldreich, 2021).

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Property Testing, Cycle-Freeness, Bounded Arboricity

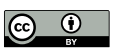
Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.60

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2404.18126`

Funding *Reut Levi*: ISF grant number 1867/20
Dana Ron: ISF grant number 1146/18

[1] As presented in (*Levi, ICALP 2021*), the complexity of the algorithm depends on $\log \log n$, but this dependence can be replaced with at most a polylogarithmic dependence on the arboricity.



51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 60; pp. 60:1–60:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Detecting small subgraphs with specific structures (referred to as *finding network motifs*) is a basic algorithmic task, with a variety of applications in biology, sociology and network science (see e.g. [21, 8, 31, 11, 7, 28, 5, 18, 6, 32, 23]). Of special interest is the natural case of subgraphs that are cycles of a fixed size k, which we denote by C_k. When the algorithm receives the entire graph as input, then by the well known result of Alon, Yuster and Zwick [4], this task can be solved in time $\widetilde{O}(n^\omega)$ where n is the number of graph vertices and ω is the exponent of matrix multiplication.[2] But what if we seek a sublinear-time (randomized) algorithm that does not read the entire graph? Namely, the algorithm is given query access to the graph[3] and should find a C_k when the graph is not C_k-free. This is clearly not possible if the graph contains only a single copy of C_k. However, is it possible to detect such a copy in sublinear-time when the graph is relatively far from being C_k-free? By "relatively far" we mean that it is necessary to remove a non-negligible fraction, denoted ϵ, of its edges in order to obtain an C_k-free graph. A closely related formulation of the question is whether we can design a one-sided error algorithm for testing C_k-freeness.[4]

If the maximum degree in the graph is upper bounded by a parameter $d_{\max}$, then the C_k-freeness testing problem can easily be solved by performing a number of queries that grows polynomially with $d_{\max}$ and exponentially with $\Theta(k)$ [20]. In particular, when $d_{\max} = O(1)$, then there is no dependence on the size of the graph G. We are however interested in considering graphs with varying degrees, so that, in particular, the maximum degree may be much larger than the average degree, and possibly as large as $\Theta(n)$.

For the special and interesting case where $k = 3$, i.e., the cycle is a triangle, Alon, Kaufman, Krivelevich and Ron [3] gave several upper and lower bounds on the query complexity of testing triangle-freeness as a function of the average degree d of the graph (in addition to the dependence on n and ϵ). While the upper and lower bounds are not tight in general, they are tight for $d = O(1)$, where the complexity is $\Theta(\sqrt{n})$ (for constant ϵ). The lower bound in this case is essentially based on "hiding" a small clique.

Since the aforementioned lower bound relies on the existence of a small dense subgraph, a natural question, studied by Levi [26], is whether it is possible to obtain improved (and possibly tight) results when the arboricity of the graph, denoted $arb(G)$, is bounded.[5] Focusing on the result under the assumption that $m \geq n$ (i.e., $d = \Omega(1)$) Levi showed that $\widetilde{O}(arb(G))$ queries are sufficient for testing triangle-freeness (the dependence on $1/\epsilon$ is polynomial), and that $\Omega(arb(G))$ queries are necessary.[6] In particular, when $arb(G)$ is a constant, the complexity is polynomial in $1/\epsilon$ and does not depend on the size of the graph.

In this work we seek to understand the complexity of testing C_k-freeness, in particular with one-sided error, for fixed $k > 3$. Our main focus is on constant arboricity graphs and some of our results extend to bounded arboricity graphs, as well as to F-freeness for any

---

[2] The dependence on k is exponential, but k is considered a constant.

[3] The types of queries typically considered are neighbor queries ("what is the ith neighbor of a vertex v?"), degree queries ("what is the degree of a vertex v?"), and pair queries ("is there an edge between a pair of vertices v and u?").

[4] The problems are equivalent if the algorithm is not given access to degree queries, otherwise the algorithm might find evidence to the existence of a C_k without actually detecting one. We note that all our algorithms do detect copies of C_k when they reject.

[5] The arboricity of a graph G is the minimum number of forests required to cover its edges, and is equal (up to a factor of 2) to the maximum average degree of any subgraph of G.

[6] To be precise, this lower bound holds for $m \geq (arb(G))^3$ – if $m < (\arg(G))^3$ then the lower bound is $\Omega(m^{1/3})$. See also Footnote 1 regarding the upper bound.

general subgraph F (of constant size). We note that the problem of testing cycle-freeness without requiring the cycle to be of specific length, is different from our problem. We further discuss this in Section 1.3. In the next subsection we state our findings.

1.1 Our results

Since our main focus is on graphs with constant arboricity, we first state our results in this setting, and later discuss our extensions to graphs with non-constant arboricity. Throughout this paper we assume that $m = \Omega(n)$ since even obtaining a single edge in the graph requires $\Omega(n/m)$ queries. Our algorithms use degree and neighbor queries and our lower bounds also allow pair queries (see Footnote 3). For simplicity, we state our results for constant ϵ. All our algorithms have a polynomial dependence on $1/\epsilon$.

Our first finding is that, as opposed to the case of $k = 3$, where the complexity of testing C_3-freeness grows with the arboricity of the graph but not with the size of the graph,[7] this is no longer the case for $k = 4$ (and larger k). In particular:

▶ **Theorem 1.** *The query complexity of one-sided error testing of C_4-freeness in constant-arboricity graphs over n vertices is $\widetilde{\Theta}(n^{1/4})$. The same bound holds for testing C_5-freeness.*

Theorem 1 (together with the upper bound in [20]) answers negatively the following open problem raised by Goldreich.

Open problem (number 3.2 in [19]): From bounded degree to bounded arboricity.
Suppose that property Π is testable within complexity $Q(n, \epsilon)$ in the bounded-degree graph model. Provide an upper bound on the complexity of testing Π in the general graph model under the promise that the tested graph has constant arboricity. For example, can the latter complexity be linear in $Q(n, \epsilon)$ while permitting extra $poly(\log n)$ or $1/\epsilon$ factors?

The $\Omega(n^{1/4})$ lower bound for testing C_4-freeness, answers this question negatively. Indeed, testing C_4-freeness in d-bounded degree graphs can be done with $poly(d, \epsilon)$ queries [20], while our lower bound suggest that even in constant arboricity graphs, a polynomial dependence on n is necessary.

When $k \geq 6$, we show that it is no longer possible to obtain a complexity of $\widetilde{O}(n^{1/4})$ as is the case for $k = 4, 5$.

▶ **Theorem 2.** *Let $k \geq 6$. Any one-sided error tester for the property of C_k-freeness in graphs of constant arboricity over n vertices must perform $\Omega(n^{1/3})$ queries.*

While for C_6 we were not able to match the lower bound of $\Omega(n^{1/3})$, we were able to obtain a sublinear-time algorithm, as stated next.

▶ **Theorem 3.** *There exists a one-sided error algorithm for testing C_6-freeness in graphs of constant arboricity over n vertices whose query complexity is $\widetilde{O}(n^{1/2})$.*

For general (fixed) k we prove the following upper bound.

▶ **Theorem 4.** *There exists a one-sided error algorithm for testing C_k-freeness in graphs of constant arboricity over n vertices whose query complexity is $O(n^{1-1/\lfloor k/2 \rfloor})$.*

We also prove a more general result for testing F-freeness for any constant size subgraph F. Below, $\ell(F)$ is as defined in Definition 10, and is always upper bounded by the number of vertices in F.

[7] We note that this is true also for other k-cliques for $k > 3$.

▶ **Theorem 5.** *There exists a one-sided error algorithm for testing F-freeness in graphs of constant arboricity over n vertices whose query complexity is $O(n^{1-1/\ell(F)})$.*

1.1.1 Extensions for general arboricity

We state our results for general arboricity graphs assuming that the algorithm is given an upper bound α on the arboricity of the graph (in the lower bounds the algorithm may be assumed to know the arboricity). Alternatively, if the algorithm receives as an input the number of edges, m, (as in previous results for C_k-freeness [3, 26]) instead of an upper bound on the arboricity, then we can estimate a notion known [26] as the "effective arboricity" of the graph, and depend on it instead of α. This is potentially beneficial since the effective arboricity can be much smaller than the actual arboricity of the graph, and it does not affect the asymptotic running times of our algorithms in terms of the dependence on the size of the graph and α. For further details see Section 2.

For C_4-freeness we give both an upper bound and a lower bound for general arboricity graphs. In particular, we show that a linear dependence on α is sufficient and a $\sqrt{\alpha}$-dependence is necessary (both for one-sided error algorithms) as stated next.

▶ **Theorem 6.** *There exists a one-sided error algorithm for testing C_4-freeness in graphs of arboricity at most α over n vertices whose query complexity is $\tilde{O}\left(\min\{n^{1/4}\alpha, \alpha + n^{3/4}\}\right)$.*[8]

▶ **Theorem 7.** *Testing C_4-freeness with one-sided error in graphs over n vertices with arboricity $c_1 \log n < \alpha < n^{1/2}/c_1'$ for sufficiently large constants c_1 and c_1' requires $\Omega(n^{1/4}\alpha^{1/2})$ queries.*[9]

For general constant size subgraphs F (and in particular C_k) our upper bound also has at most a linear dependence on α (recall that $\ell(F)$ is defined in Definition 10).

▶ **Theorem 8.** *There exists a one-sided error tester for F-freeness whose query complexity is $O\left(k^{2+1/\ell(F)} \cdot m^{1-1/\ell(F)} \cdot \alpha^{1/\ell(F)}\right)$.*

▶ **Corollary 9.** *There exists a one-sided error tester for C_k-freeness whose query complexity for even k is $O\left(k^{2+(2/k))} \cdot m^{1-2/k} \cdot \alpha^{2/k}\right)$, and for odd k is $O\left(k^{2+2/(k+1)} \cdot m^{1-2/(k+1)} \cdot \alpha^{2/(k+1)}\right)$.*

We comment that our lower bound of $\Omega(n^{1/3})$ for one-sided error algorithms, $k \geq 6$ and constant arboricity (stated in Theorem 2) also applies to graphs with non-constant arboricity (by adding a C_k-free subgraph with higher arboricity).[10]

We also note that it is possible to extend our algorithms for C_5 and C_6 freeness so as to get a polynomial (but not linear) dependence on α. However, these extensions do not introduce new techniques (and are most probably not optimal), so we do not present them here.

---

[8] More precisely, for values $\alpha < \log n$, the complexity is $O(n^{1/4}\alpha^{1/2}\log^{1/2} n/\epsilon^3)$, for values $\log n < \alpha < \sqrt{n}$, the complexity is $O(n^{1/4}\alpha/\epsilon^3)$, and for values $\alpha > n^{1/2}$, it is $O((\alpha + n^{3/4})/\epsilon^3)$.

[9] Note that the two-sided error lower bound of $\Omega(n^{1/4})$ for constant arboricity graphs (as stated in Theorem 1) also holds for graphs with higher arboricity α, and in particular, $\alpha = O(\log n)$. This is the case since we can simply add a small subgraph with arboricity α and no C_4s to the lower bound construction.

[10] For an odd k, it suffices to add a dense bipartite graph, and for even k, by the Erdős girth conjecture [16], one can add a subgraph with arboricity $n^{2/k}$.

1.2 A high-level discussion of our algorithms and lower bounds

Before discussing each of our results in more detail, we highlight some common themes. The starting point of all our algorithms is that if a graph is ϵ-far from being C_k-free (for a constant k), then it contains $\Omega(\epsilon m)$ edge-disjoint cycles.[11] We next use the bounded arboricity of the graph. Specifically, if a graph has arboricity at most α, then the number of edges between pairs of vertices that both have degree greater than $\theta_0 = \Theta(\alpha/\epsilon)$, is at most $O(\epsilon m)$.

Hence, there is a set of edge-disjoint C_ks, which we denote by $\mathcal{C}$, such that $|\mathcal{C}| = \Omega(\epsilon m)$, and no C_k in $\mathcal{C}$ contains any edge between two vertices with degree greater than θ_0. In other words, for every k-cycle ρ in $\mathcal{C}$, and for every vertex v with degree greater than θ_0 in ρ, the two neighbors of v in ρ have degree at most θ_0. In particular, when α is a constant, the two neighbors have degree $O(1/\epsilon)$.

At this point our algorithms diverge, but there are two common aspects when $k = 4, 5, 6$, which we would like to highlight. The first is that for the sake of "catching" one of the C_ks in $\mathcal{C}$, it will be useful to consider a subset, $\mathcal{C}'$, in which every vertex v that participates in one of the edge-disjoint C_ks in $\mathcal{C}'$ actually participate in $\Omega(\epsilon \cdot d(v))$ C_ks in $\mathcal{C}'$. The existence of such a subset follows by applying (as a mental experiment) a simple iterative process that removes C_ks with vertices that do not obey this constraint.

To illustrate why it is useful to have such a set $\mathcal{C}'$, consider the case of $k = 4$, and assume that a relatively large fraction of the C_4s in $\mathcal{C}'$ contain, in addition to the two vertices of degree at most $\theta_0 = O(\alpha/\epsilon)$, at least one other vertex that has degree at most $\theta_1 = O(n^{1/2}/\epsilon)$. In this case we can obtain such a vertex v with high probability (as discussed below), and then sample roughly $\sqrt{d(v)/\epsilon} = O(\sqrt{\theta_1/\epsilon}) = O(n^{1/4}/\epsilon)$ of its neighbors, so that the following holds. By (a slight variant of) the birthday paradox, with high constant probability we hit two of its neighbors, u and u', that reside on the same C_4 in $\mathcal{C}'$ (and hence have degree at most θ_0). By querying all the neighbors of u and u', we obtain this C_4.

However, what if for most of the C_4's in $\mathcal{C}'$ there are two vertices with degree significantly larger than $\sqrt{n}$ (that are "one opposite the other" on the C_4s)? Roughly speaking, in this case we exploit the fact that the number of such high-degree vertices is bounded, and we show how to detect a C_4 by performing random walks of length 2. A related issue arises in the case of $k = 6$, when there are three very high degree vertices on most C_6s in $\mathcal{C}'$. In this case we show how to essentially reduce the problem to testing triangle-freeness in a certain auxiliary graph. More precisely, the auxiliary graph is a multi-graph to which we have access only to certain types of queries, so that we cannot apply the algorithm of [3]. However, we can still show how to obtain a triangle in this graph, and hence a C_6 in the original graph. Interestingly, our general lower bound of $\Omega(n^{1/3})$ for C_k-freeness, $k \geq 6$ builds on the lower bound for testing triangle-freeness of [3].

In the following subsections we assume for the sake of the exposition that ϵ is a constant.

1.2.1 The results for C_4-freeness (and C_5-freeness)

We discuss our results for C_4-freeness in graphs with general arboricity. The results for C_5-freeness in constant arboricity graphs are obtained using very similar techniques.

[11] To verify this, let G be a graph that is ϵ-far from being C_k-free for a fixed constant k. Consider any maximal set S of edge-disjoint k-cycles. Since by removing all $k \cdot |S|$ edges on these cycles, the graph can be made cycle-free, $|S| \geq \epsilon m/k = \Omega(\epsilon m)$.

The algorithm

Our algorithm for testing C_4-freeness, Test-C_4-freeness, which has query complexity $\widetilde{O}(n^{1/4}\alpha)$, is governed by two thresholds: $\theta_0 = \Theta(\alpha)$, and $\theta_1 = \Theta(n^{1/2})$. For the sake of the current high-level presentation, we assume that[12] $\alpha \leq n^{1/2}$, so that $\theta_0 \leq \theta_1$.

The algorithm first samples $O(1)$ edges approximately uniformly by invoking a procedure Select-an-Edge,[13] and then randomly selects one of their endpoints. For each vertex v selected, it queries its degree, $d(v)$. If $d(v) \leq \theta_1$, then the algorithm selects $O(\sqrt{d(v)})$ random neighbors of v, and for each selected neighbor u such that $d(u) \leq \theta_0$, it queries all the neighbors of u. If $d(v) > \theta_1$, then the algorithm performs $\widetilde{O}(n^{1/4}\alpha^{1/2})$ random walks of length two from v. If a C_4 is observed in any one of these steps, then the algorithm rejects, otherwise it accepts.

The analysis of the algorithm

By the above description, the algorithm will only reject a graph if it detects a C_4, implying that it never errs on C_4-free graphs. Hence, consider a graph G that is far from being C_4-free. As discussed at the start of Section 1.2, the setting of $\theta_0 = \Theta(\alpha)$ (together with the fact that G is $\Omega(1)$-far from being C_4-free) implies the following. There exists a set, denoted $\mathcal{C}$, of $\Omega(m)$ *edge-disjoint* C_4s in G, such that no C_4 in $\mathcal{C}$ contains an edge between two vertices that both have degree greater than θ_0. Thus, for each C_4 in $\mathcal{C}$, there are at most two vertices with degree greater than θ_0, and they do not neighbor each other.

Considering the second aforementioned degree threshold θ_1 (and recalling that $\theta_1 \geq \theta_0$), we partition $\mathcal{C}$ into two subsets. The first, $\mathcal{C}_1$, consists of those C_4s in $\mathcal{C}$ that contain at most one vertex with degree greater than θ_1, and the second, $\mathcal{C}_2$, of the remaining C_4s in $\mathcal{C}$, which contain exactly two vertices with degree greater than θ_1. Since $\mathcal{C} = \mathcal{C}_1 \uplus \mathcal{C}_2$, at least one of these subsets is of size $\Omega(m)$.

***C_4*s with at most one high-degree vertex.** Suppose first that $|\mathcal{C}_1| = \Omega(m)$. Observe that since each 4-cycle $\rho \in \mathcal{C}_1$ contains at least two vertices with degree at most θ_0 and at most one vertex with degree greater than θ_1, it must contain at least one vertex, with degree at most θ_1 whose neighbors on the C_4 both have degree at most θ_0. For an illustration, see the LHS of Figure 1. Furthermore, we show that there exists a subset of $\mathcal{C}_1$, which we denote by $\mathcal{C}'_1$, such that $|\mathcal{C}'_1| = \Omega(m)$, and every vertex v that participates in one of the C_4s in $\mathcal{C}'_1$, actually participates in $\Omega(d(v))$ edges-disjoint C_4s in $\mathcal{C}_1$. It follows that in this case, when the algorithm selects a random edge (almost uniformly), with high constant probability it will obtain an edge with (at least) one endpoint v having the above properties. Conditioned on the selection of such a vertex v, the algorithm selects $\Theta(\sqrt{d(v)})$ random neighbors of v. By the birthday paradox, with high constant probability, among these neighbors there will be a pair of vertices that reside, together with v, on a common C_4 in $\mathcal{C}'_1$. Once their (at most θ_0) neighbors are queried, this C_4 is revealed.

***C_4*s with two high-degree vertex.** We now turn to the case in which $|\mathcal{C}_2| = \Omega(m)$. Here too we can show that there exists a subset of $\mathcal{C}_2$, denoted $\mathcal{C}'_2$, such that $|\mathcal{C}'_2| = \Omega(m)$, and every vertex v that participates in one of the C_4s in $\mathcal{C}'_2$ actually participates in $\Omega(d(v))$ edges-disjoint C_4s in $\mathcal{C}_2$.

[12] Indeed, graphs with arboricity greater than $n^{1/2}$ necessarily contain at least one C_4, but since we are interested in a one-sided error algorithm, and α is only known to be an upper bound on α, the algorithm cannot reject if it is provided with $\alpha > n^{1/2}$.

[13] This is a fairly standard and simple procedure, where we use the fact that graph has bounded arboricity, so that most of its edges have at least one endpoint with degree θ_0.

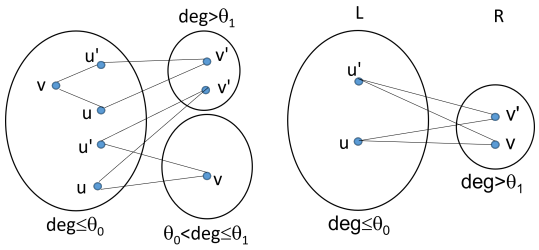


Figure 1 An illustration for some of the cases considered in the analysis of the algorithm for C_4-freeness. On the left side are two examples in which there is a single vertex v' with degree greater than θ_1, so that there is a vertex v with degree at most θ_1 with two neighbors whose degree is at most θ_0. On the right is an illustration when there are two such vertices with degree greater than θ_1.

Recall that by the definitions of $\mathcal{C}$ and $\mathcal{C}_2$ and since $\mathcal{C}'_2 \subseteq \mathcal{C}_2 \subseteq \mathcal{C}$, the following holds. For each 4-cycle ρ in $\mathcal{C}'_2$, since it is in $\mathcal{C}_2$, there are two vertices whose degree is greater than θ_1. Therefore, by the definition of $\mathcal{C}$, they are both adjacent on ρ to two vertices whose degree is at most θ_0. Hence, if we consider the subgraph induced by the vertices and edges of the C_4s in $\mathcal{C}'_2$, it is a bipartite graph, where on one side, denoted L, all vertices have degree at most θ_0, and on the other side, denoted R, all vertices have degree greater than θ_1. Furthermore, by the definition of $\mathcal{C}'_2$, for each vertex in R, a constant fraction of its neighbors (in the original graph G) belong to L, and for each vertex in L, a constant fraction of its neighbors belong to R. For an illustration, see the RHS of Figure 1.

Hence, if we select an edge almost uniformly and pick one of its endpoints with equal probability, with high constant probability we obtain a vertex $v \in R$. Conditioned on this event, since $d(v) > \theta_1$, the algorithm will perform $\widetilde{O}(n^{1/4}\alpha^{1/2})$ random walks of length two from v, and with high constant probability, a constant fraction of these walks will be of the form (v, u, v') where $u \in L$ and $v' \in R$. If for some v' we get two walks, (v, u, v') and (v, u', v') for $u \neq u'$, then a C_4 is detected.

Observe that since all vertices in R have degree greater than $\theta_1 = \Theta(n^{1/2})$, we have that $|R| \leq 2m/\theta_1 = O(n^{1/2}\alpha)$. This can be used to show that the expected number of pairs of walks that induce a C_4 is greater than 1. In order to show that we actually get such a pair with high constant probability, we perform a more careful analysis to bound the variance.

A (two-sided error) lower bound for testing C_4-freeness in constant arboricity graphs

To obtain this lower bound of $\Omega(n^{1/4})$, we define two distributions over graphs. In the support of the first distribution, $\mathcal{D}_0$, all graphs are C_4-free, and in the support of the second distribution, $\mathcal{D}_1$, all graphs are $\Omega(1)$-far from being C_4-free. Furthermore, $\mathcal{D}_0$ is uniform over all graphs isomorphic to a specific graph G_0, and $\mathcal{D}_1$ is uniform over all graphs isomorphic to a specific graph G_1.

We next describe a slightly simplified version of the two graphs (which cannot be used to prove the lower bound, but gives the essence of the proof). Both graphs are bipartite graphs, where one side, Y, contains $\Theta(\sqrt{n})$ vertices, and the other side, X, contains $\Theta(n)$ vertices, In G_0, each vertex in X has a unique pair of neighbors in Y (so there are no C_4s). On the other hand, in G_1, each vertex x in X has a "twin", x', where x and x' have the same pair of neighbors in Y (thus creating $\Omega(n)$ edge-disjoint C_4. See Figure 2. Observe that the arboricity of both graphs is 2 as for any subset of vertices S, the number of edges within S is at most $|S \cap X| \cdot 2$ so the average degree in the subgraph induced by S is at most 2.

In order to prove that no (possibly adaptive) algorithm can distinguish between a graph selected according to $\mathcal{D}_0$ and a graph selected according to $\mathcal{D}_1$, we define two processes, $\mathcal{P}_0$ and $\mathcal{P}_1$, which answer the queries of a testing algorithm while selecting a graph from $\mathcal{D}_0$ (respectively, $\mathcal{D}_1$) "on the fly". The lower bound of $\Omega(n^{1/4})$ follows from the fact that when performing fewer than $n^{1/4}/c$ queries (where c is a sufficiently large constant), for both distributions, with high constant probability, each new neighbor query is answered by a uniformly selected vertex id.

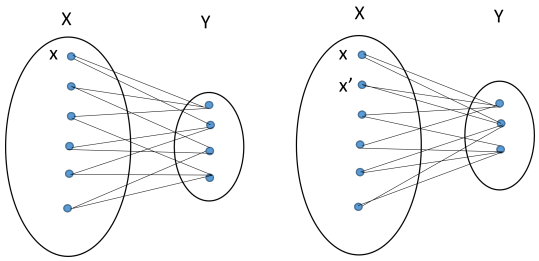


Figure 2 An illustration for the lower bound construction. The graph on the left is C_4-free while the graph on the right contains $\Omega(m)$ edge-disjoint C_4s and is hence $\Omega(1)$-far from being C_4-free.

A one-sided error lower bound for testing C_4-freeness in graphs with arboricity α

We next discuss the lower bound of $\Omega(n^{1/4}\alpha^{1/2})$ for graphs with arboricity $\alpha = \Omega(\log n)$ and one-sided error algorithms.

Here we define a single distribution $\mathcal{D}$ which is uniform over a family of graphs with arboricity α such that almost all graphs in this family are $\Omega(1)$-far from C_4-free.

Roughly speaking, the graphs in the support of $\mathcal{D}$ are random bipartite graphs, where one side, Y, is of size $\Theta(\sqrt{n}\alpha)$ and the other side, X, is of size $\Theta(n)$. Every vertex in X has α neighbors in Y, and every vertex in Y has $\Theta(\sqrt{n})$ neighbors in X. We need to show that if we select such a graph randomly, then on one hand it will be $\Omega(1)$-far from C_4-free, and on the other hand, in order to detect a C_4, any algorithm must perform $\Omega(n^{1/4}\alpha^{1/2})$ queries.

We next discuss the high-level idea as to why the resulting graphs are (with high constant probability) far from being C_4-free. Consider a fixed edge (x, y) in the bipartite graph, where $x \in X, y \in Y$. The number of C_4s this edge participates in is determined by the number of edges between the sets of neighbors of x and y, respectively $\Gamma(x)$ and $\Gamma(y)$. Recall that x has $\Theta(\alpha)$ neighbors and y has $\Theta(\sqrt{n})$ neighbors. Since overall there are $|X| \cdot |Y| = \Theta(n^{3/2}\alpha)$ potential pairs in the bipartite graph, and $\Theta(n\alpha)$ edges, each pair in $X \times Y$ is an edge with probability $\Theta(1/\sqrt{n})$. Hence, the expected number of edges between $\Gamma(x)$ and $\Gamma(y)$ is $|\Gamma(x)| \cdot |\Gamma(y)| \cdot (1/\sqrt{n}) = \Theta(\alpha)$. By analyzing the variance between pairs of edges, we furthermore show that with high constant probability, most edges do not participate in too many C_4s. Combining the two insights, it follows that with high constant probability, the graph is indeed far form being C_4-free.

In order to prove that any algorithm that performs at most $n^{1/4}\alpha^{1/2}/c$ queries (for a sufficiently large constant c), will not detect a C_4 with high constant probability, we actually prove that it will not detect *any* cycle. Roughly speaking, we show that by the randomness of the construction, since $|Y| = \Theta(\sqrt{n}\alpha)$, and the algorithm performs $O(\sqrt{|Y|})$ queries, each new neighbor query is answered by a uniformly distributed vertex that has not yet been observed. Therefore, the algorithm essentially views a forest.

A central challenge that we need to overcome is that we do not want to allow parallel edges, where the above construction might lead to their existence. One possibility is to first define the distribution over graphs with parallel edges and then to remove them. The benefit is that due to the higher degree of independence in the construction, it is somewhat easier to formally prove that the graphs obtained (with parallel edges) are with high probability $\Omega(1)$-far from C_4-free, and this remains the case when we remove parallel edges.

However, this creates a difficulty when we turn to argue that no (one-sided error) algorithm can detect a C_4 unless it makes $\Omega(n^{1/4}\alpha^{1/2})$ queries. The difficulty is due to the fact that in the formal proof we need to deal with dependencies that arise due to varying degrees (which occur because parallel edges are removed). While intuitively, varying degree should not actually "help" the algorithm, this intuition is difficult to formalize. Hence, we have chosen to define the distribution, from the start, over graphs that do not have parallel edges. This choice creates some technical challenges of its own (in particular in the argument that the graphs obtained are $\Omega(1)$-far from C_4-free), but we are able to overcome them. For more details see the full version.

1.2.2 The algorithm for C_6-freeness

Recall that for C_6 we have a (one-sided error) testing algorithm whose query complexity is $\widetilde{O}(n^{1/2})$. In addition to assuming (for the sake of the exposition) that ϵ is a constant, we also ignore polylogarithmic factors in n. Similarly to the algorithm for testing C_4-freeness, the algorithm for testing C_6-freeness in constant arboricity graphs is governed by two thresholds. The first, θ_0, is of the order of the arboricity, so that it is a constant (recall that we assume that ϵ is a constant). The second, θ_2, is of the order of $\sqrt{n}$.

The algorithm repeats the following process several times. It selects a vertex v uniformly at random, and if $d(v) \leq \theta_0$, it performs a *restricted* BFS starting from v to depth 4. Specifically:

1. Whenever a vertex u is reached such that $d(u) \leq \theta_0$, all its neighbors are queried.
2. Whenever a vertex u is reached such that $d(u) > \theta_0$ and u is reached from a vertex u' such that $d(u') \leq \theta_0$, there are two sub-cases. If $d(u) \leq \theta_1$, then all of u's neighbors are queried. Otherwise, θ_1 neighbors of u are selected uniformly at random.
3. Whenever a vertex u is reached from a vertex u' such that both $d(u) > \theta_0$ and $d(u') > \theta_0$, the BFS does not continue from u.

The algorithm rejects if and only if it observes a C_6.

Consider a graph that is far from being C_6-free, so that it contains a set of $\Omega(m) = \Omega(n)$ edge-disjoint C_6s. Furthermore, it contains such a set, denoted $\mathcal{C}$ for which every C_6 in $\mathcal{C}$ contains at most three vertices with degree greater than θ_0, and furthermore, these vertices are *not* adjacent on the C_6. We partition $\mathcal{C}$ into three subsets: $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$, depending on the number of vertices with degree greater than θ_0 that it contains.

If either $|\mathcal{C}_1| = \Omega(m)$, or $|\mathcal{C}_2| = \Omega(m)$, then it is not hard to show that the algorithm will detect a C_6 with high constant probability. The more interesting part of the proof is handling the case in which only $|\mathcal{C}_3| = \Omega(m)$.

In this case we define an auxiliary multi-graph, denoted G', over the set of vertices that participate in C_6s belonging to $\mathcal{C}_3$, and have degree greater than θ_0 (in G). We denote this set of vertices by M, and the set of vertices with degree at most θ_0 that participate in these C_6s, by L.

Assume for simplicity that each vertex in L has degree exactly 2 (i.e., it participates in a single C_6). For each pair of vertices in M, we put in G' a set of parallel edges, whose size equals the number of length-2 paths between them in G that pass through vertices in L. Hence, for each C_6 in $\mathcal{C}$, we have a triangle in G', where these triangles are edge-disjoint, and we denote their set by $\mathcal{T}$. See Figure 3.

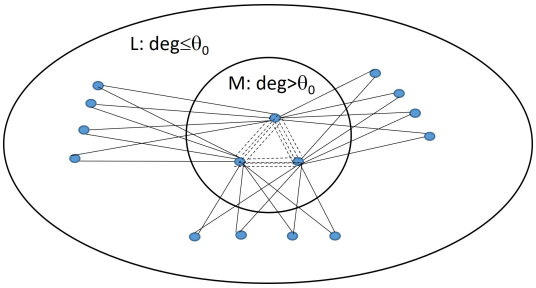


Figure 3 An illustration of the auxiliary (multi-)graph G' in the C_6-freeness testing algorithm. The dashed lines represent edges in G', each one corresponding to a length-2 path in G that passes through a vertex with degree at most θ_0.

Observe that selecting a vertex uniformly at random from L and querying its two neighbors in M corresponds to selecting an edge uniformly at random in G'. If we add an additional simplifying assumption by which (in G), vertices belonging to M only neighbor vertices belonging to L, then our algorithm on G essentially translates to picking a random edge in G'. Then depending on the degree of the endpoints, either querying all their neighbors in G' or θ_1 random neighbors.

Let H denote the subset of vertices in M whose degree in G is greater than θ_1. If relatively many triangles in $\mathcal{T}$ contain at most one vertex in H, then we are done, since these triangles contain an edge for which both endpoints have degree at most θ_1. Hence, it remains to address the case in which almost all triangles in $\mathcal{T}$ have two or three vertices in H.

Roughly speaking, in this case we show that the existence of many edge-disjoint, but not vertex-disjoint, triangles in G' that contain such high-degree vertices implies the existence of "many more" triangles that may be caught by our algorithm. As an illustrative extreme (but easy) special case, assume that in G' there are only three vertices. Then the existence of some number t of edge-disjoint triangles between them, actually implies the existence of t^3 (non edge-disjoint) triangles.

1.2.3 The general lower bound for C_k-freeness, $k \geq 6$

We establish our general lower bound of $\Omega(n^{1/3})$ for one-sided error testing of C_k-freeness when $k \geq 6$ by building on a lower bound for testing triangle-freeness that appears in [3, Lemma 2]. This lower bound for testing triangle-freeness is based on the difficulty of detecting a triangle in graphs selected uniformly from a family $\mathcal{G}_{n'}$ of graphs in which almost all graphs are $\Omega(1)$-far from being triangle-free. All graphs in the family are d-regular tri-partite graphs over n' vertices and the lower bound on the number of queries necessary to detect a triangle (with constant probability), is $\Omega(\min\{d, n'/d\})$. By setting $d = \sqrt{n'}$, the lower bound is $\Omega(\sqrt{n'})$.

We show that, for any constant $k \geq 6$, if we had a one-sided error testing algorithm $\mathcal{A}$ for C_k-freeness of graphs with n vertices and constant arboricity using at most $n^{1/3}/c$ queries (for a constant c), then we would be able to detect triangles in graphs selected uniformly from $\mathcal{G}_{n'}$ using at most $\sqrt{n'}/c'$ queries (for a constant c').

To this end we define an algorithm $\mathcal{A}$ that, given query access to a graph $G' \in \mathcal{G}_{n'}$, implicitly defines a graph G for which the following holds. First, the number of vertices in G is $n = \Theta((n')^{3/2})$, and the number of edges is $m = \Theta(m')$, where m' is the number of edges in G' (so that $m' = \Theta((n')^{3/2})$). Second, G has arboricity 2. Third, the distance of G to C_k-freeness is of the same order as the distance of G' to triangle-freeness. Fourth, there is a one-to-one correspondence between triangles in G' and C_ks in G. The basic idea is to replace edges in the tri-partite graph G' with paths of length $k/3$. See Figure 4

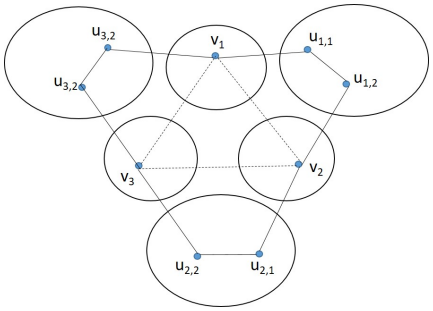


Figure 4 An illustration for the lower bound construction for C_k-freeness in constant arboricity graphs when $k = 9$. The three circles in the middle and the dashed lines represent a graph $G' \in \mathcal{G}_{n'}$. The outer circles represent the additional vertices in G. Since $k = 9$ in this example, each edge in G' is replaced by a path of length 3 in G.

Assuming there existed a testing algorithm $\mathcal{A}$ as stated above, the algorithm $\mathcal{A}'$ would use it to try and find a C_k in G (and hence a triangle in G'). In order to be able to run $\mathcal{A}$ on G, the algorithm $\mathcal{A}'$ must be able to answer queries of $\mathcal{A}$ to G by performing queries to G'. We show how this can be done with a constant multiplicative overhead. Hence, the lower bound of $\Omega(\sqrt{n'})$ for testing triangle-freeness (when the degree is $\Theta(\sqrt{n'})$) translated into a lower bound of $\Omega(n^{1/3})$ for testing C_k-freeness.

1.2.4 The general upper bound for C_k-freeness

Recall that our starting point is that if G is $\Omega(1)$-far from being C_k-free, then it contains a set $\mathcal{C}$ of $\Omega(m)$ edge-disjoint C_k's that do not contain any edge between vertices that both have degree greater than $\theta_0 = \Theta(\alpha)$. We refer to vertices with degree at most θ_0 as *light* vertices, and to those with degree greater than θ_0 as *heavy*. Hence, each C_k in $\mathcal{C}$ has at least $\lceil k/2 \rceil$ light vertices, and each heavy vertex on it neighbors two light vertices.

We present two different algorithms, where each of them is suitable for a different setting. The basic idea of both algorithms is to take a large enough sample of vertices and edges so that the subgraph determined by the sampled light vertices and their incident edges, as well as the sampled edges, contains a copy of C_k. The query complexity of each algorithm is stated following its high-level description.

The first algorithm

Our first algorithm simply samples vertices uniformly, independently at random, and then performs queries that reveal the neighbors of all light vertices in the sample. To analyze what is the sufficient sample size for this algorithm, consider the following generalization of the birthday paradox for k-way collisions. Assume we sample elements under the uniform distribution over $[n]$. Then we obtain a k-way collision after taking $\Theta(n^{1-1/k})$ samples.

Similarly, suppose we sample vertices uniformly from a graph that is composed only of n/k vertex-disjoint copies of C_k. Then, after sampling $\Theta(n^{1-1/k})$ vertices, we will hit all the vertices of at least one of the copies (with high constant probability). Conditioned on this event, if we reveal the neighborhood of all the vertices in the sample, then we obtain a C_k.

The next observation is that, in fact, we only need to hit a *vertex cover* of a copy of a C_k (as opposed to all its vertices). In particular we would like to hit such a cover that contains only light vertices, which we refer to as a *light vertex cover*. For constant α, this yields an improved dependence on k in the exponent, i.e., $O(n^{1-1/\lfloor k/2 \rfloor})$ sampled vertices suffice.

When taking into account the dependence on α (so that it is not necessarily true that $m = O(n)$) and incorporating this in the analysis, we prove that the query complexity is upper bounded by $O(m \cdot (\alpha/m)^{2/k})$ for even k and $O(m \cdot (\alpha/m)^{2/(k+1)})$ for odd k (up to a polynomial dependence on k). Since $\alpha \leq \sqrt{m}$ it follows that the above bounds are at most $O(m^{1-1/k})$ and $O(m^{1-1/(k+1)})$, respectively.

The second algorithm

Our second algorithm is designed for the case in which k is odd and $m = \Omega(\alpha^{(k+3)/2})$. In particular it is preferable when α is constant. We observe that when k is odd, for each C_k in $\mathcal{C}$, there is an edge in which both endpoints are light vertices. Therefore, if we sample edges (almost) uniformly from the graph (using a variant of the procedure Select-an-Edge), then we are likely to hit one of these edges. This additional step reduces the number of vertices we need to hit in each copy by 2, which results in improved complexity for some range of the parameters. In particular, the query complexity of this algorithm is $O(m \cdot (\alpha^2/m)^{2/(k-1)})$. Specifically, when α is a constant, the query complexity of this algorithm (which works for odd k) is $O(m^{1-2/(k-1)})$ (instead of $O(m^{1-2/(k+1)})$).

General subgraph F

Our first algorithm also works for any constant-size subgraph, F, where the upper bound on the sample size is of the form $m^{1-1/\ell(F)}$ where $\ell(F)$ depends on the structure of F, as defined next.

▶ **Definition 10.** *For a graph $F = (V_F, E_F)$ let $\mathcal{VC}(F)$ denote the set of all vertex covers of F. For a vertex cover Z of F we denote by $\mathcal{VC}'(Z)$ the set of vertex covers of F that are subsets of Z. We define $\ell(F) = \max_{Z \in \mathcal{VC}(F)} \left\{ min_{B \in \mathcal{VC}'(Z)} (|B|) \right\}$.*

Observe that by Definition 10, we have that $\ell(F)$ is lower bounded by the size of a minimum vertex cover of F and is upper bounded by $k = |V_F|$.

The high-level idea is that if we want to find a copy of F, it suffices to hit a light vertex cover of this copy and then query all neighbors of the sampled light vertices.

1.3 Related work

In this subsection we shortly discuss several related works, in addition to the two aforementioned works regarding testing C_3-freeness [3, 26].

Testing subgraph-freeness for fixed, constant size subgraphs in the dense-graphs model can be done using a number of queries that depends only on $1/\epsilon$ (where the dependence is a tower of height poly$(1/\epsilon)$), as shown by Alon, Fischer, Krivelevich and Szegedy [2]. Alon [1] proved that a super polynomial dependence on $1/\epsilon$ is necessary, unless the subgraph F is bipartite. Goldreich and Ron addressed the problem in the bounded-degree model [20], and gave a simple algorithm that depends polynomially on $1/\epsilon$ and the maximum degree in the graph, and exponentially on the diameter of F.

A special case of graphs that have bounded arboricity is the family of graphs that exclude a fixed minor (a.k.a. minor-free graphs). Newman and Sohler [29] showed that for this family of graphs, in the bounded-degree model, all properties can be tested with no dependence on the size of the graph G. Moreover, it was recently shown [25, 27] that any property which is monotone and additive[14] (and in particular F-freeness where F is a connected graph) can be tested using a number of queries that is only polynomial in $1/\epsilon$ and d, where d is the degree bound (and $O(d^{\rho(\epsilon)})$ in general $(\epsilon, \rho(\epsilon))$-hyperfinite graphs[15]). For minor-free graphs with unbounded degrees, Czumaj and Sohler [10] showed that a property is testable with one sided error and a number of queries that does not depend on the size of the graph if and only if it can be reduced to testing for a finite family of finite forbidden subgraphs.[16] The correctness of their algorithm relies on the fact that the arboricity of minor-free graphs remains constant even after contractions of edges (which is not the case for general constant-arboricity graphs).

In general graphs, it was shown that k-path freeness [22] and more generally T-freeness where T is a tree of order k [17], can be tested with time and query complexity that depend only on k, assuming the edges of the graph can be accessed uniformly at random. Testing cycle-freeness (where a no instance is a graph that is far from being a forest) was studied in the bounded-degree model in [20], where a two-sided error algorithm was given whose query complexity is polynomial in $1/\epsilon$ and the degree bound. Czumaj et. al [9] showed that the complexity of this problem for one-sided error algorithms in the bounded-degree model is $\widetilde{\Theta}(\sqrt{n})$ (for constant ϵ – their algorithm has a polynomial dependence on $1/\epsilon$), and the algorithm can be adapted to the general-graphs model.

Other sublinear-time graph algorithms for counting and sampling (rather than detecting) subgraphs that give improved results when the graph G has bounded arboricity include [14, 12, 15, 13].

1.4 Organization

We start in Section 2 with some preliminaries. In Section 3 we give the upper bound for testing C_4-freeness. All missing details and proofs appear in the full version of the paper.

2 Preliminaries

Unless stated explicitly otherwise, the graphs we consider are simple, so that in particular they do not contain any parallel edges. We denote the number of vertices in the graph by n and the number of edges by m. Every vertex v in the graph has a unique id, denoted $id(v)$, and its degree is denoted by $d(v)$.

We work in what is known as the *general graph model* [30, 24]. In particular, under this model, the *distance* of a graph G to C_k-freeness, denoted $dist(G, C_k\text{-free})$, is the minimum fraction of edges that should be removed from G in order to obtain a C_k-free graph. As for the allowed queries, a neighbor query to the ith neighbor of a vertex v is denoted by $nbr(v, i)$, and to its degree by $deg(v)$. A pair query between two vertices v_1 and v_2 is denoted by $pair(v_1, v_2)$. Given query access to a graph G and a parameter ϵ, a one-sided error testing

[14] A property is monotone if it closed under removal of edges and vertices. A property is additive if it is closed under the disjoint union of graphs.

[15] Let ρ be a function from $\mathbb{R}_+$ to $\mathbb{R}_+$. A graph $G = (V, E)$ is $(\epsilon, \rho(\epsilon))$-hyperfinite if for every $\epsilon > 0$ it is possible to remove $\epsilon|V|$ edges of the graph such that the remaining graph has connected components of size at most $\rho(\epsilon)$.

[16] They consider a model in which they can perform only random neighbor queries.

algorithm for C_k-freeness should accept G if it is C_k-free, and should reject G with probability at least $2/3$ if $dist(G, C_k\text{-free}) > \epsilon$. If the algorithm may also reject C_k-free graphs with probability at most $1/3$, then it has two-sided error.

As noted in the introduction, we assume our algorithms for graphs whose arboricity is not promised to be constant, are given an upper bound α on the arboricity $arb(G)$ of the tested graph G, and their complexity depends on this upper bound. Alternatively, if the algorithm is provided with the number of edges, m, then it may run a procedure from [26] to obtain a value α^* that with high constant probability satisfies the following: (1) $\alpha^* \leq 2arb(G)$; (2) The number of edges between vertices whose degree is at least $\alpha^*/(c\epsilon)$ for a constant c is at most $(1-\epsilon/c')m$ (for another, sufficiently large, constant c'). Up to polylogarithmic factors in n, the query complexity and running time of the procedure are $O(arb(G)/\epsilon^3)$ with high probability (assuming the average degree is $\Omega(1)$).

Throughout this work we assume, whenever needed, that ϵ is upper bounded by some sufficiently small constant (or else it can be set to that constant).

We also make use of the following claim – whose proof is given in the full version.

▷ Claim 11. For an integer s let $\{\chi_{i,j}\}_{(i,j)\in\Phi(s)}$ be Bernoulli random variables where $\Pr[\chi_{i,j}=1]=\mu$ for every $(i,j)\in\Phi(s)$. Suppose that the following conditions hold for some $c_1>0$ and $c_2>4$.

1. For every $(i_1,j_1)\in\Phi(s)$ and $(i_2,j_2)\in\Phi(s)$ such that the four indices are distinct, χ_{i_1,j_1} and χ_{i_2,j_2} are independent.
2. For every $(i_1,j_1)\in\Phi(s)$ and $(i_2,j_2)\in\Phi(s)$ such that exactly two of the four indices are the same, $\Pr[\chi_{i_1,j_1}=\chi_{i_2,j_2}=1]\leq c_1\cdot\mu^{3/2}$.
3. $s\geq c_2/\sqrt{\mu}$.

Then $\Pr\left[\sum_{(i,j)\in\Phi(s)}\chi_{i,j}=0\right]\leq\frac{1+c_1}{c_2}$.

3 An upper bound of $\widetilde{O}(n^{1/4}\alpha)$ for testing C_4-freeness

In this section we prove the more general (arboricity-dependent) form of the upper bound for testing C_4-freeness which is stated as Theorem 6 in the introduction.

Recall that the assumption on α is that it is an upper bound on the arboricity $arb(G)$. While it is known that for graphs with $arb(G)>\sqrt{n}$ there exists a C_4, we cannot simply reject if we get $\alpha>n^{1/2}$ since it might be that $arb(G)<\sqrt{n}$ (and we want one-sided error). However, in the case that $\alpha>n^{1/2}$, the $n^{1/4}\alpha$ term is replaced by $n^{3/4}$ (and the additive α term is due to the edge sampling).

The algorithm referred to in Theorem 6 is described next.

Algorithm 1 Test-C_4-freeness(n,ϵ,α).

1. Let $\theta_0=4\alpha/\epsilon$, $\theta_1=c_1\cdot\sqrt{n}/\epsilon$ (where c_1 will be determined subsequently) and $\theta_{min}=\min\{\theta_0,\theta_1\}$ (it is useful to read the algorithm while having in mind that $\theta_0\leq\theta_1$ (i.e., $\alpha=O(\sqrt{n})$) so that $\theta_{min}=\theta_0$).
2. Repeat the following $t=\Theta(1/\epsilon)$ times:
   a. Select an edge e by calling the procedure Select-an-Edge(α,ϵ), which appears below. If it does not return an edge, then continue to the next iteration.
   b. Select an endpoint v of e by flipping a fair coin.
   c. If $d(v)\leq\theta_1$, then select $s_1=\Theta(\sqrt{d(v)/\epsilon})$ $(=O(n^{1/4}/\epsilon))$ random neighbors of v, and for each neighbor u such that $d(u)\leq\theta_{min}$ query all the neighbors of u.
   d. Otherwise $(d(v)>\theta_1)$, perform $s_2=\Theta(\sqrt{(n\alpha/\theta_1)\log n/\epsilon^2})$ $(=\tilde{O}(n^{1/4}\alpha^{1/2}/\epsilon^2))$ random walks of length 2 starting from v.
   e. If a C_4 is detected, then return it, "Reject" and terminate.
3. Return "Accept".

We note that the algorithm can be unified/simplified so that it only performs random walks of length-2, where the number of walks is $\Theta(n^{1/4}\alpha/\epsilon^2)$, but then the analysis becomes slightly more complicated.

Algorithm 2 Select-an-edge(ϵ, α).

1. Repeat the following $\Theta(\alpha/\epsilon)$ times:
   a. Select a vertex u uniformly at random.
   b. If $d(u) \leq \theta_0$ for $\theta_0 = 4\alpha/\epsilon$, then with probability $d(u)/\theta_0$ select an edge incident to u uniformly at random and return it.
2. If no edge was selected, then return "Fail".

We start by stating a claim concerning the procedure Select-an-Edge where its proof is deferred to the full version.We then state and prove two additional claims that will be used in the proof of Theorem 6.

▷ Claim 12. With probability at least 2/3 the procedure Select-an-Edge returns an edge. Conditioned on it returning an edge, each edge incident to a vertex with degree at most θ_0 is returned with probability at least $1/(2m')$ and at most $1/m'$, where m' is the number of edges incident to vertices with degree at most θ_0.

▷ Claim 13. Let v be a vertex and let $C(v, \theta_{min})$ be a set of edge-disjoint C_4's containing v such that the neighbors of v on these C_4s all have degree at most θ_{min}, where θ_{min} is as defined in the algorithm.[17] Suppose that $|C(v, \theta_{min})| \geq 1$ and let $\epsilon' = |C(v, \theta_{min})|/d(v)$. If we select $s = 16\sqrt{d(v)/\epsilon'}$ random neighbors of v, and for each selected neighbor u such that $d(u) \leq \theta_{min}$ we query all the neighbors of u, then the probability that we obtain a C_4 is at least 9/10.

Proof. Let $E'(v)$ denote the set of edges incident to v that participate in the set $C(v, \theta_{min})$. By the premise of the claim, $|E'(v)|/d(v) = 2|C(v, \theta_{min})|/d(v) = 2\epsilon'$. Let s' be the number of neighbors of v that are incident to edges in $E'(v)$ among the s selected random neighbors of v. It holds that $\mathbb{E}[s'] = 2\epsilon' \cdot s$, and by the multiplicative Chernoff bound, $s' \geq \epsilon' \cdot s$ with probability at least $1 - e^{-\epsilon' \cdot s/4}$. We first show that this probability is at least 19/20, and then condition on this event. By the setting of $s = 16\sqrt{d(v)/\epsilon'}$, it holds that $\epsilon' \cdot s = 16\sqrt{\epsilon' \cdot d(v)}$, and by the setting of $\epsilon' = |C(v, \theta)|/d(v)$, we get $\epsilon' \cdot s \geq 16\sqrt{|C(v, \theta_{min})|} \geq 16$. Therefore, with probability at least 19/20, $s' > \epsilon' \cdot s = 16\sqrt{\epsilon' \cdot d(v)}$. We condition on this event and consider only those s' selected neighbors of v that are endpoints of $E'(v)$.

For each 4-cycle $\rho \in C(v, \theta_{min})$, let $u_1(\rho)$ and $u_2(\rho)$ be the two neighbors of v on this C_4 (so that they are endpoints of edges in $E'(v)$). Since the C_4s in $C(v, \theta_{min})$ are edge-disjoint, these vertices are distinct. Observe that the s' selected neighbors of v are uniformly distributed in $\bigcup_{\rho \in C(v, \theta_{min})} \{u_1(\rho), u_2(\rho)\}$, and that $s' \geq 16 \cdot \sqrt{|C(v, \theta_{min})|}$. Hence, by the "birthday paradox", with high constant probability, the sample of neighbors of v contains two vertices, $u_1(\rho)$, and $u_2(\rho)$ for some $\rho \in C(v, \theta_{min})$. Conditioned on this event, once the (at most θ_{min}) neighbors of $u_1(\rho)$ and $u_2(\rho)$ are queried, the four-cycle ρ is observed. ◁

▷ Claim 14. Let G be a graph over n vertices and m edges, and let $\theta_1, \epsilon', \epsilon''$ be parameters. Suppose that G contains a bipartite subgraph $G' = (L, R, E(G'))$ such that every vertex in R has degree at least θ_1 in G. Let v be a vertex in R such that v has at least $\epsilon' \cdot d(v)$

[17] Actually, we do not rely on the setting of θ_{min}, so this claim holds for any threshold value.

neighbors in L where each of these neighbors, u, has at least $\epsilon'' \cdot \max\{d(u), \frac{m}{n}\}$ neighbors in R. If $\theta_1 \geq 2\sqrt{n/(\epsilon' \cdot \epsilon'')}$ and we take $s_2 \geq \frac{32}{\epsilon' \cdot \epsilon''} \cdot \sqrt{2 \log n \cdot |R|}$ random walks of length 2 from v for a sufficiently large constant c', then with probability at least 9/10 we shall detect a C_4 in G.

Proof. For a pair of vertices v and $v' \neq v$ in R, let $\ell_2(v, v')$ be the number of length-2 paths between v and v', and let $\ell_2(v, R) = \sum_{v' \in R} \ell_2(v, v')$. Consider taking two random length-2 walks from v, and let $\mathcal{E}_1$ be the event that both of them end at vertices in R. Let $\mathcal{E}_2$ be the event that these two paths are distinct and end at the same vertex. Then for each single vertex $v' \in R$, conditioned on $\mathcal{E}_1$, the probability that the two walks end at v' is exactly $\frac{\ell_2(v,v')}{\ell_2(v,R)} \cdot \frac{\ell_2(v,v')-1}{\ell_2(v,R)}$. Therefore,

$$\Pr[\mathcal{E}_2 \,|\, \mathcal{E}_1] = \sum_{v' \in R} \frac{\ell_2(v, v')}{\ell_2(v, R)} \cdot \frac{\ell_2(v, v') - 1}{\ell_2(v, R)} = \frac{1}{(\ell_2(v, R))^2} \cdot \sum_{v' \in R} (\ell_2(v, v'))^2 - \frac{1}{\ell_2(v, R)} \,. \quad (1)$$

We would like to lower bound the above probability. For the first term on the right-hand-side, by applying the Cauchy-Schwartz inequality we get that

$$\frac{1}{(\ell_2(v, R))^2} \cdot \sum_{v' \in R} (\ell_2(v, v'))^2 \geq \frac{1}{(\ell_2(v, R))^2} \cdot |R| \cdot \left(\frac{\ell_2(v, R))}{|R|} \right)^2 = \frac{1}{|R|} \,. \quad (2)$$

By combining Equations (1) and (2) we get that $\Pr[\mathcal{E}_2 \,|\, \mathcal{E}_1] \geq \frac{1}{|R|} - \frac{1}{\ell_2(v,R)}$. Since each vertex in R has degree at least θ_1, we have that $|R| \leq \frac{2m}{\theta_1}$. By the premise of the claim regarding v and its neighbors, v has $\epsilon' d(v) \geq \epsilon' \cdot \theta_1$ neighbors in L, and each of them has at least $\epsilon'' \cdot (m/n)$ neighbors in R. Therefore,

$$\ell_2(v, R) \geq \epsilon' \cdot \theta_1 \cdot \epsilon'' \cdot \frac{m}{n} \geq \frac{\epsilon' \cdot \epsilon'' \cdot \theta_1^2 \cdot |R|}{2n} \geq 2|R|, \quad (3)$$

where the last inequality is by the premise $\theta_1 \geq 2\sqrt{n/(\epsilon' \cdot \epsilon'')}$. Therefore, $\Pr[\mathcal{E}_2 \,|\, \mathcal{E}_1] \geq \frac{1}{2|R|}$. So far we have shown that when taking two distinct random walks from v, and conditioned on them both ending at R (the event $\mathcal{E}_1$), the two paths collide on the end vertex (and hence result in a C_4) with probability at least $1/2|R|$. We shall now prove, that when taking s length-2 random walks from v, sufficiently many of them indeed end at R, and that with high probability, at least two of them collide, resulting in a C_4.

Consider first the event $\mathcal{E}_1$. By the premise of the claim, v has at least $\epsilon' \cdot d(v)$ neighbors in L, and each u of them has at least $\epsilon'' \max\{d(u), m/n\} \geq \epsilon'' d(u)$ neighbors in R. Therefore, the probability that a single random walk from v ends at R is at least $\epsilon' \cdot \epsilon''$. Hence, if we take $s \geq \frac{32}{\epsilon' \cdot \epsilon''} \sqrt{2 \log n \cdot |R|}$ length-2 random walks from v, and let s' denote the number of walks that end at a vertex in R, we have that $\mathbb{E}[s'] = 32 \cdot \sqrt{2 \log n \cdot |R|}$, and that with probability at least 9/10, we have $s' \geq 16 \cdot \sqrt{2 \log n \cdot |R|}$. We henceforth condition on this event.

Let $\chi'_{i,j}$ denote the event that the ith and jth random walks among the ones that end at R collide on the ending vertex (and thus result in a C_4). By the above discussion, we have that for a specific pair $i \neq j$, $\Pr[\chi'_{i,j} = 1] \geq 1/2|R|$. We now lower bound the probability that at least one pair of random walks from the s' that end in R detects a C_4, i.e. lower bound $\sum_{i,j \in [s']} \chi_{i,j}$, using Claim 11. For that end we also need to upper bound the variance of the sum.

Partition the vertices in R according to $\ell_2(v, v')$, where $R_x(v) = \{v' : 2^{x-1} < \ell_2(v, v') \leq 2^x\}$ for $x = 0, \ldots \log L \leq \log n$. Since $\sum_{v' \in R} \frac{\ell_2(v,v')}{\ell_2(v,R)} \cdot \frac{\ell_2(v,v')-1}{\ell_2(v,R)} > \frac{1}{2|R|}$, there exists at least one setting of x for which $\sum_{v' \in R_x} \frac{\ell_2(v,v')}{\ell_2(v,R)} \cdot \frac{\ell_2(v,v')-1}{\ell_2(v,R)} \geq \frac{1}{2|R| \log n}$. We denote this setting by x^* and observe that $x^* > 0$ (since for every $v' \in R_0$, $\ell_2(v, v') - 1 = 0$).

For every $i, j \in [s']$, $i < j$, we define a Bernoulli random variable $\chi_{i,j}$ that is 1 if and only if the ith and the jth random walks from v (among the s' considered) end at the same $v' \in R_{x^*}$ and pass through a different vertex in L. We next show that we can apply Claim 11 (with s in that claim set to s') to get an upper bound on the probability that $\sum_{i,j \in [s'], i<j} \chi_{i,j} = 0$ (which is an upper bound on the probability that we do not detect a C_4).

By the definition of the random variables, for every $i_1 \neq i_2, j_1 \neq j_2$, it holds that $\chi_{i_1,j_1}, \chi_{i_2,j_2}$ are independent, so that the first condition in Claim 11 is satisfied. Next, for any pair $i, j \in [s']$, $i < j$ we have that

$$\mu = \Pr[\chi_{i,j} = 1] = \sum_{v' \in R_{x^*}} \frac{\ell_2(v,v')}{\ell_2(v,R)} \cdot \frac{\ell_2(v,v') - 1}{\ell_2(v,R)} \geq \frac{1}{2|R| \log n} \,. \tag{4}$$

Therefore, we have that $s' \geq 16 \cdot \sqrt{2|R| \log n} = 16/\sqrt{\mu}$, and so the third condition in Claim 11 is satisfied (for $c_2 = 16$, where s' serves as the parameter s in the claim).

It remains to verify that the second condition holds. For any four indices $i_1, j_1, i_2, j_2 \in [s']$, $i_1 < j_1$, $i_2 < j_2$ such that exactly two of the four indices are the same,

$$\Pr[\chi_{i_1,j_1} = \chi_{i_2,j_2} = 1] = \sum_{v' \in R_{x^*}} \frac{\ell_2(v,v')}{\ell_2(v,R)} \cdot \left(\frac{\ell_2(v,v') - 1}{\ell_2(v,R)} \right)^2 \leq \mu \cdot \frac{2^{x^*} - 1}{\ell_2(v,R)} \,. \tag{5}$$

Since by Equation (4), $\mu = \sum_{v' \in R_{x^*}} \frac{\ell_2(v,v')}{\ell_2(v,R)} \cdot \frac{\ell_2(v,v')-1}{\ell_2(v,R)} \geq \frac{2^{2(x^*-1)}}{2(\ell_2(v,R))^2}$ (as $\ell_2(v,v') \geq 2^{x^*-1}$ for every $v' \in R_{x^*}$ and $\ell_2(v,v') - 1 \geq \ell_2(v,v')/2$), we get that $\Pr[\chi_{i_1,j_1} = \chi_{i_2,j_2} = 1] < \sqrt{2} \cdot \mu^{3/2}$, and so the second condition in Claim 11 holds as well (for $c_1 = \sqrt{2}$). Thus, the current claim follows. ◁

We are now ready to prove Theorem 6.

Proof of Theorem 6. Since the algorithm only rejects a graph G if it detects a C_4, it will always accept graphs that are C_4-free. Hence, we focus on the case that G is ϵ-far from being C_4-free.

Recall that $\theta_0 = 4\alpha/\epsilon$ and let $E_{>\theta_0}$ be the subset of edges in G where both endpoints have degree greater than θ_0. Since the arboricity of G is at most α, there are at most $2m/\theta_0$ vertices with degree greater than θ_0, so that $|E_{>\theta_0}| \leq (2m/\theta_0) \cdot \alpha = \epsilon m/2$ edges.

Since G is ϵ-far from C_4-free, if we remove all edges in $E_{>\theta_0}$, then we get a graph that is at least $(\epsilon/2)$-far from C_4-free. It follows that there exists a set of edge-disjoint C_4s, denoted $\mathcal{C}$, such that no C_4 in $\mathcal{C}$ contains an edge in $E_{>\theta_0}$, and $|\mathcal{C}| \geq \epsilon m/8$.

We next partition $\mathcal{C}$ into two disjoint subsets: $\mathcal{C}_1$ contains those C_4s that have at most one vertex with degree at least θ_1 in them, and $\mathcal{C}_2$ contains those that have at least two such vertices (where in the case $\theta_0 \leq \theta_1$ there will be exactly two). Since $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$, either $|\mathcal{C}_1| \geq \epsilon m/16$ or $|\mathcal{C}_2| \geq \epsilon m/16$ (possibly both).

The case $|\mathcal{C}_1| \geq \epsilon m/16$. Consider first the case that $|\mathcal{C}_1| \geq \epsilon m/16$. In order to analyze this case, we apply a process of "coloring" vertices and edges. Initially, all vertices and edges that participate in C_4s that belong to $\mathcal{C}_1$ are colored *green*, and all other vertices and edges are colored *red*. We next apply the following iterative process. As long as there is a green vertex v whose number of incident green edges is less than $\epsilon d(v)/64$, color v and its green incident edges by red. Observe that the total number of edges colored red by this process is at most $\epsilon m/32$. Furthermore, at the end of this process, every green vertex v has at least

$\epsilon d(v)/64$ incident green edges (and if a vertex is red, then all its incident edges are red). Let $\mathcal{C}_1'$ be the subset of $\mathcal{C}_1$ that consists of those C_4s in $\mathcal{C}_1$ whose edges all remain green after the process (and hence they are green), so that $|\mathcal{C}_1'| \geq \epsilon m/32$.

By the definition of $\mathcal{C}_1$, and hence also $\mathcal{C}_1'$, in each C_4 in $\mathcal{C}_1'$ there is at most one vertex with degree greater than θ_1, and no edges such that both endpoints have degree greater than θ_0. Assume without loss of generality that for each four-cycle $\rho \in \mathcal{C}_1'$, where $\rho = (v_0(\rho), v_1(\rho), v_2(\rho), v_3(\rho))$, $v_2(\rho)$ is the highest degree vertex (where $d(v_2(\rho))$ could be any value between 1 to n). Let $V_0(\mathcal{C}_1') = \bigcup_{\rho \in \mathcal{C}_1'} \{v_0(\rho)\}$ denote this set of vertices (i.e., the ones that are across from the highest degree vertex in a (green) four-cycle in $\mathcal{C}_1'$).

Observation. *For every $\rho \in \mathcal{C}_1'$,*

1. *$d(v_0(\rho)) \leq \theta_1$, and*
2. *$v_1(\rho)$ and $v_3(\rho)$ are of degree at most $\theta_{min} = \min\{\theta_0, \theta_1\}$.*

To verify this observation, note that by the definition of $\mathcal{C}_1'$, for every $\rho \in \mathcal{C}_1'$, there is at most one vertex with degree greater than θ_1, and since $v_2(\rho)$ is the highest degree vertex in ρ, it follows that all three other vertices in ρ are of degree at most θ_1.

We now show that $d(v_1(\rho)) \leq \theta_0$, and the proof for $v_3(\rho)$ is identical. If $d(v_2(\rho)) > \theta_0$, then it must be the case that $d(v_1(\rho)) < \theta_0$, as otherwise both have degree greater than θ_0 and so they cannot be connected, which is a contradiction to them both being incident on the four-cycle ρ. If $d(v_2(\rho)) \leq \theta_0$, then since $v_2(\rho)$ is the highest degree vertex in ρ, $d(v_1(\rho)) \leq d(v_2(\rho)) \leq \theta_0$.

Therefore, for every $v \in V_0(\mathcal{C}_1')$, it has at least $\epsilon d(v)/64$ neighbors u such that (v, u) is green and $d(u) \leq \theta_{min}$. Hence, overall in the graph, the set of vertices $V_0(\mathcal{C}_1')$ has at least $\epsilon m/32$ green edges that are incident to it and their second endpoint is of degree at most $\theta_{min} \leq \theta_0$. It follows that conditioned on an edge being returned by procedure Select-an-Edge, by Claim 12, it returns an edge incident to a vertex $v \in V_0(\mathcal{C}_1')$ with probability at least $(\epsilon m/32)/2m' > \epsilon/128$ (since $m' > \frac{1}{2}m$). So the probability that in some iteration of Test-C_4-freeness a vertex $v_0 \in V_0(\mathcal{C}_1')$ is selected, is at least $1 - (1 - \frac{\epsilon}{128})^t > 9/10$ (recall that $t = \Theta(1/\epsilon)$ so that it suffices to set $t = 500/\epsilon$).

Conditioning on this event, we apply Claim 13. Specifically:

- $\theta_0 = 4\alpha/\epsilon$ (as defined in Step 1 in Algorithm Test-C_4-freeness);
- $C(v_0, \theta_{min})$ is the set of C_4s in $\mathcal{C}_1'$ that are incident to v_0;
- $\epsilon' = |C(v_0, \theta_{min})|/d(v) \geq \epsilon/128$ (since v_0 has at least $\epsilon d(v)/64$ incident green edges, and they can be partitioned into pairs such that each pair belongs to exactly one C_4 in $C(v_0, \theta_{min})$);
- $d(v_0) \leq \theta_1$ (by the above observation);

In order to apply the claim, we must ensure that $s > 16\sqrt{d(v_0)/\epsilon'}$. By the above, it is sufficient to set s_1 in Step 2c, to be $s_1 = 512\sqrt{d(v_0)/\epsilon}$.

Hence, by Claim 13, if Step 2c is applied to v_0, then a C_4 is observed with probability at least 9/10.

The analysis for the case that $|\mathcal{C}_2| \geq \epsilon m/16$ is similar, and due to space constraints, it is deferred to the full version.

We next turn to analyze the query complexity. By the settings of θ_0, θ_1, t, s_1 and s_2 in the algorithm, the query complexity of the algorithm is upper bounded as follows.

$$O\left(\frac{1}{\epsilon} \cdot \left(\frac{\alpha}{\epsilon} + \max\{s_1, s_2\}\right)\right) = O\left(\frac{1}{\epsilon}\left(\frac{\alpha}{\epsilon} + \max\left\{\sqrt{\frac{\theta_1}{\epsilon}} \cdot \theta_{min}, \frac{1}{\epsilon^2} \cdot \sqrt{\frac{n\alpha}{\theta_1} \cdot \log n}\right\}\right)\right) \quad (6)$$

For the case that $\alpha \leq (c_1/4)\sqrt{n}$, we have that $\theta_{min} = \theta_0 = \Theta(\alpha/\epsilon)$ and that $\theta_1 = \Theta(\sqrt{n}/\epsilon)$, and so we get a complexity of

$$O\left(\epsilon^{-3} \cdot n^{1/4}\alpha^{1/2} \cdot \max\{\alpha^{1/2}, \log^{1/2} n\}\right) = O(\epsilon^{-3} \cdot n^{1/4}\alpha) \ , \tag{7}$$

where the last inequality is for $\alpha > \log n$, and otherwise the complexity is $O(\epsilon^{-3} \cdot n^{1/4}\alpha^{1/2}\log^{1/2} n)$.

For the case that $\alpha > (c_1/4)\sqrt{n}$, we have that $\theta_{min} = \theta_1 = \Theta(\sqrt{n}/\epsilon)$. Therefore, the complexity is

$$O(\epsilon^{-3} \cdot (\alpha + n^{3/4})) \ . \tag{8}$$

Thus, the proof is complete. ◀

References

1 Noga Alon. Testing subgraphs in large graphs. *Random Struct. Algorithms*, 21(3-4):359–370, 2002. doi:10.1002/rsa.10056.

2 Noga Alon, Eldar Fischer, Michael Krivelevich, and Mario Szegedy. Efficient testing of large graphs. *Combinatorica*, 20(4):451–476, 2000. doi:10.1007/s004930070001.

3 Noga Alon, Tali Kaufman, Michael Krivelevich, and Dana Ron. Testing triangle-freeness in general graphs. *SIAM Journal on Discrete Mathematics*, 22(2):786–819, 2008.

4 Noga Alon, Raphael Yuster, and Uri Zwick. Color coding. *Journal of the ACM*, 42(4):844–856, 1995.

5 Uri Alon. Network motifs: theory and experimental approaches. *Nature Reviews Genetics*, 8(6):450–461, 2007.

6 Jonathan W. Berry, Bruce Hendrickson, Randall A. LaViolette, and Cynthia A. Phillips. Tolerating the Community Detection Resolution Limit with Edge Weighting. *Physical Review E*, 83(5):056119, May 2011.

7 Ronald S. Burt. Structural holes and good ideas. *American journal of sociology*, 110(2):349–399, 2004.

8 James S. Coleman. Social capital in the creation of human capital. *American Journal of Sociology*, 94:S95–S120, 1988. URL: http://www.jstor.org/stable/2780243.

9 Artur Czumaj, Oded Goldreich, Dana Ron, C. Seshadhri, Asaf Shapira, and Christian Sohler. Finding cycles and trees in sublinear time. *Random Structures and Algorithms*, 45(2):139–184, 2014.

10 Artur Czumaj and Christian Sohler. A characterization of graph properties testable for general planar graphs with one-sided error (it's all about forbidden subgraphs). In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 1525–1548, 2019. doi:10.1109/FOCS.2019.00089.

11 Jean-Pierre Eckmann and Elisha Moses. Curvature of co-links uncovers hidden thematic layers in the World Wide Web. *Proceedings of the National Academy of Sciences*, 99(9):5825–5829, 2002. doi:10.1073/pnas.032093399.

12 Talya Eden, Dana Ron, and Will Rosenbaum. The arboricity captures the complexity of sampling edges. In *46th International Colloquium on Automata, Languages, and Programming, ICALP*, pages 52:1–52:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.52.

13 Talya Eden, Dana Ron, and Will Rosenbaum. Almost optimal bounds for sublinear-time sampling of k-cliques in bounded arboricity graphs. In *49th International Colloquium on Automata, Languages, and Programming, ICALP*, pages 56:1–56:19, 2022. doi:10.4230/LIPIcs.ICALP.2022.56.

14 Talya Eden, Dana Ron, and C. Seshadhri. Sublinear time estimation of degree distribution moments: The arboricity connection. *SIAM J. Discret. Math.*, 33(4):2267–2285, 2019. doi:10.1137/17M1159014.

15 Talya Eden, Dana Ron, and C. Seshadhri. Faster sublinear approximation of the number of k-cliques in low-arboricity graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1467–1478, 2020. doi:10.1137/1.9781611975994.89.

16 Paul Erdős. Extremal problems in graph theory. In *In Proc. Symp. Theory of Graphs and its Applications*, pages 29–36, 1963.

17 Guy Even, Orr Fischer, Pierre Fraigniaud, Tzlil Gonen, Reut Levi, Moti Medina, Pedro Montealegre, Dennis Olivetti, Rotem Oshman, Ivan Rapaport, and Ioan Todinca. Three notes on distributed property testing. In *31st International Symposium on Distributed Computing, DISC*, pages 15:1–15:30, 2017. doi:10.4230/LIPIcs.DISC.2017.15.

18 Brooke Foucault Welles, Anne Van Devender, and Noshir Contractor. Is a friend a friend?: Investigating the structure of friendship networks in virtual worlds. In *CHI Extended Abstracts on Human Factors in Computing Systems*, pages 4027–4032, 2010.

19 Oded Goldreich. Open problems in property testing of graphs. In *Electron. Colloquium Comput. Complex.*, volume 28, page 88, 2021.

20 Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

21 Paul W. Holland and Samuel Leinhardt. A method for detecting structure in sociometric data. *American Journal of Sociology*, 76:492–513, 1970.

22 Kazuo Iwama and Yuichi Yoshida. Parameterized testability. *ACM Trans. Comput. Theory*, 9(4):16:1–16:16, 2018. doi:10.1145/3155294.

23 Matthew O. Jackson, Tomas Rodriguez-Barraquer, and Xu Tan. Social capital and social quilts: Network patterns of favor exchange. *American Economic Review*, 102(5):1857–1897, 2012.

24 Tali Kaufman, Michael Krivelevich, and Dana Ron. Tight bounds for testing bipartiteness in general graphs. *SIAM Journal on Computing*, 33(6):1441–1483, 2004. doi:10.1137/S0097539703436424.

25 Akash Kumar, C. Seshadhri, and Andrew Stolman. Random walks and forbidden minors III: poly($d\varepsilon^{-1}$)–time partition oracles for minor-free graph classes. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 257–268, 2021. doi:10.1109/FOCS52979.2021.00034.

26 Reut Levi. Testing triangle freeness in the general model in graphs with arboricity $o(\sqrt{n})$. In *48th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 198, pages 93:1–93:13, 2021.

27 Reut Levi and Nadav Shoshan. Testing Hamiltonicity (and other problems) in minor-free graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 61:1–61:23, 2021. doi:10.4230/LIPIcs.APPROX/RANDOM.2021.61.

28 Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

29 Ilan Newman and Christian Sohler. Every property of hyperfinite graphs is testable. *SIAM Journal on Computing*, 42(3):1095–1112, 2013. doi:10.1137/120890946.

30 Michal Parnas and Dana Ron. Testing the diameter of graphs. *Random Struct. Algorithms*, 20(2):165–183, 2002. doi:10.1002/rsa.10013.

31 Alejandro Portes. Social capital: Its origins and applications in modern sociology. In Eric L. Lesser, editor, *Knowledge and Social Capital*, pages 43–67. Butterworth-Heinemann, 2000. doi:10.1016/B978-0-7506-7222-1.50006-4.

32 C. Seshadhri, Tamara G. Kolda, and Ali Pinar. Community structure and scale-free collections of Erdös-Rényi graphs. *Physical Review E*, 85(5):056109, May 2012. doi:10.1103/PhysRevE.85.056109.

Parameterized Algorithms for Steiner Forest in Bounded Width Graphs

Andreas Emil Feldmann
Department of Computer Science, University of Sheffield, UK

Michael Lampis
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Abstract

In this paper we reassess the parameterized complexity and approximability of the well-studied Steiner Forest problem in several graph classes of bounded width. The problem takes an edge-weighted graph and pairs of vertices as input, and the aim is to find a minimum cost subgraph in which each given vertex pair lies in the same connected component. It is known that this problem is APX-hard in general, and NP-hard on graphs of treewidth 3, treedepth 4, and feedback vertex set size 2. However, Bateni, Hajiaghayi and Marx [JACM, 2011] gave an approximation scheme with a runtime of $n^{O(k^2/\varepsilon)}$ on graphs of treewidth k. Our main result is a much faster *efficient parameterized approximation scheme (EPAS)* with a runtime of $2^{O(\frac{k^2}{\varepsilon}\log\frac{k}{\varepsilon})} \cdot n^{O(1)}$. If k instead is the vertex cover number of the input graph, we show how to compute the optimum solution in $2^{O(k \log k)} \cdot n^{O(1)}$ time, and we also prove that this runtime dependence on k is asymptotically best possible, under ETH. Furthermore, if k is the size of a feedback edge set, then we obtain a faster $2^{O(k)} \cdot n^{O(1)}$ time algorithm, which again cannot be improved under ETH.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Approximation algorithms analysis

Keywords and phrases Steiner Forest, Approximation Algorithms, FPT algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.61

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2402.09835

Funding *Michael Lampis*: Supported by ANR project ANR-21-CE48-0022 (S-EX-AP-PE-AL).

1 Introduction

The Steiner Forest problem is one of the most well-studied problems in network design [16, 23, 24, 28]. In this problem the input consists of a graph $G = (V, E)$ with positive edge weights, a set of *terminals* $R \subseteq V$, and a set of *demands* $D \subseteq \binom{R}{2}$. The objective is to select a subgraph $F \subseteq G$, minimizing the total cost of selected edges, while ensuring that for every demand pair $\{s, t\} \in D$, s and t are in the same connected component of F. Since edge weights are positive, it is easy to see that the optimal solution is always a forest. The Steiner Forest problem finds many applications (see surveys [11, 28, 32, 33]), for example in telecommunication networks (cf. [33]).

Our goal in this paper is to reassess the complexity of this fundamental problem from the point of view of parameterized complexity and approximation algorithms.[1] In order to recall the context, it is helpful to compare Steiner Forest to the even more well-studied Steiner

[1] We assume the reader is familiar with the basics of parameterized complexity and approximation algorithms, such as the classes FPT and APX and the definition of treewidth, as given in standard textbooks [14, 19, 34]. We give full definitions of all parameters in Section 2.

Tree problem, which is the special case of Steiner Forest where all terminals are required to be connected, i.e., $D = \binom{R}{2}$, and an optimal solution is a tree. Steiner Tree was already included in Karp's seminal list [25] of NP-hard problems from the 1970s. From the approximation point of view, Steiner Tree (and therefore Steiner Forest) is known to be APX-hard [13], but both problems admit constant factor approximations in polynomial time for general input graphs, where the best approximation factors known are $\ln(4)+\varepsilon < 1.39$ [10] and 2 [1, 31], respectively. Despite this similarity, when considering graph width parameters the problems exhibit wildly divergent behaviors from the parameterized complexity point of view: whereas Steiner Tree is FPT parameterized by standard structural parameters such as treewidth and can in fact even be solved in single exponential $2^{O(k)}n^{O(1)}$ time [6] when k is the treewidth, Steiner Forest is NP-hard on graphs of treewidth 3, as shown independently by Gassner [21] and Bateni, Hajiaghayi, and Marx [4].

Steiner Forest is therefore a problem that presents a dramatic jump in complexity in this context, compared to Steiner Tree, as the hardness result on graphs of treewidth 3 rules out even an XP algorithm for parameter treewidth. One of the main positive contributions of Bateni, Hajiaghayi, and Marx [4] was an algorithm attempting to bridge this gap using approximation. In particular, they showed that Steiner Forest admits an approximation scheme for graphs of treewidth k, which computes a $(1+\varepsilon)$-approximation in $n^{O(k^2/\varepsilon)}$ time for any $\varepsilon > 0$. Hence, if we allow slightly sub-optimal solutions, we can at least place the problem in XP parameterized by treewidth. In their paper, Bateni, Hajiaghayi, and Marx [4] remark that because the exponent of the polynomial of this runtime depends on k and ε, "it remains an interesting question for future research whether this dependence can be removed", that is, whether a $(1+\varepsilon)$-approximation can be obtained in FPT time.

The main result of our paper is a positive resolution of the question of [4]: we show that Steiner Forest admits an *efficient parameterized approximation scheme (EPAS)* for treewidth, that is, a $(1+\varepsilon)$-approximation algorithm with a runtime of the form $f(k,\varepsilon)n^{O(1)}$. In other words, we show that their algorithm can be improved in a way that makes the running time FPT not only in the treewidth, but also in $1/\varepsilon$. More precisely, we show the following:

▶ **Theorem 1.** *The Steiner Forest problem admits an EPAS parameterized by the treewidth k with a runtime of $2^{O(\frac{k^2}{\varepsilon}\log\frac{k}{\varepsilon})} \cdot n^{O(1)}$.*

Moving on from treewidth, we ask what the most general parameter is for which we may hope to obtain an FPT *exact* algorithm for Steiner Forest. We observe that the NP-hardness result of [4, 21] for Steiner Forest on graphs of treewidth 3 actually has some further implications for some even more restricted parameters: the graphs constructed in their reductions also have constant *treedepth* and *feedback vertex set* size, implying that the problem remains hard for both of these parameters (which are incomparable in general). More precisely, known reductions imply the following:

▶ **Theorem 2** ([4, 21])**.** *The Steiner Forest problem is NP-hard on graphs of treewidth* 3*, treedepth* 4*, and feedback vertex set of size* 2*.*

This leads us to consider even more restricted parameters, such as the size of a *vertex cover* and *feedback edge set*, which are not bounded in this reduction. Indeed, not only do we prove that Steiner Forest is FPT for both of these parameters, but we are also able to determine the correct parameter dependence, under the Exponential Time Hypothesis (ETH). For feedback edge set the optimal dependence is single exponential:

▶ **Theorem 3.** *The* Steiner Forest *problem is FPT parameterized by the size k of a feedback edge set and can be solved in $2^{O(k)}n^{O(1)}$ time. Furthermore, no $2^{o(k)}n^{O(1)}$ time algorithm exists, under ETH.*

For the parameterization by the vertex cover size, we obtain a slower runtime for our FPT algorithm. Interestingly, we are also able to prove that this is best possible, under ETH. Our lower bound for Steiner Forest is in contrast to the Steiner Tree problem, for which a faster $2^{O(k)}n^{O(1)}$ time algorithm exists, even if k is the treewidth [6].

▶ **Theorem 4.** *The* Steiner Forest *problem is FPT parameterized by the size k of a vertex cover and can be solved in $2^{O(k \log k)}n^{O(1)}$ time. Furthermore, no $2^{o(k \log k)}n^{O(1)}$ time algorithm exists, under ETH.*

We remark that Bodlaender et al. [8] recently independently showed that Steiner Forest admits a $2^{O(k \log k)}n^{O(1)}$ time algorithm for the size k of a vertex cover (improving an algorithm for the unweighted version of the problem given in [22]). While they develop their own dynamic program to solve this problem, we rely on an existing algorithm by [4] (see Theorem 5). Accordingly our description of the algorithm is very short compared to [8]. The more interesting part of Theorem 4 however is the proof of the lower bound.

1.1 Overview of Techniques

Let us briefly sketch the high level ideas of our results given by Theorems 1, 3, and 4.

EPAS for treewidth. Our algorithm extends the work of [4], so let us briefly recall some key ideas. Given a rooted tree decomposition, a terminal t is called *active* for a bag B if there is a demand $\{s, t\} \in D$ such that t lies in the sub-tree rooted at B while s does not (see Section 2 for formal definitions). It is a standard property of tree decompositions that every bag is a separator. Hence the component of any feasible solution that contains an active terminal must intersect B. The hardness of the problem now inherently stems from the fact that we have to decide for all active terminals of a bag, how the corresponding component intersects the bag, and therefore how the active terminals (whose number is unbounded by k) are partitioned into connected components. Suppose, however, that someone supplied us with this information, that is, suppose that for each bag B we are given a set of partitions Π_B of its active terminals and we are promised that the optimal solution *conforms* to all Π_B. By this we mean that if we look at how the optimal solution partitions the active terminals of B into connected components and call this partition π, then $\pi \in \Pi_B$, that is, the optimal partition is always one of the supplied options. In this case, using this extra information, the problem does become tractable, as shown in [4]:

▶ **Theorem 5** ([4])**.** *For an input graph G on n vertices, let a rooted nice tree decomposition of width k be given, such that all terminals lie in bags of leaf nodes of the decomposition. Also, let a set Π_B of partitions of the active terminals of each bag B of the decomposition be given. If $p = \sum_B |\Pi_B|$ is the total number of partitions, then a minimum cost* Steiner Forest *solution conforming to all Π_B can be computed in $2^{O(k \log k)} \cdot (pn)^{O(1)}$ time.*

The above theorem does not seem immediately helpful, since one would still need to find a small collection of partitions Π_B in order to obtain an efficient algorithm. Note however, that the partition sets may conform to an approximate solution as well, which would let the algorithm compute a solution that is at least as good. The strategy of [4] therefore is to construct a collection of partitions that has size polynomial in n (when k, ε are fixed

constants) by stipulating that when two active terminals are "close" to each other, they should belong in the same set of the partition of some near-optimal solution. In order to bound the resulting approximation ratio, they need to provide a charging scheme: starting from an optimal solution, they merge components which are "close", to obtain a solution that conforms to the Π_B used by the algorithm. They then show that the resulting solution is still near-optimal by charging the extra cost incurred by a merging operation to one of the two merged components.

A blocking point in the above is that we need to make sure we do not "overcharge" any component. This is accomplished in [4] via a partial ordering of the components: we order the components according to the highest bag of the rooted tree decomposition they intersect, and whenever two components are merged we charge this to the *lower* component. As shown in [4], this ensures that no component is charged for more than k merges. Unfortunately, this also implies that the merging procedure is not symmetric, which severely diminishes the contexts in which we can apply it.

Let us now describe how our approach improves upon this algorithm. A key ingredient will be a more sophisticated charging scheme, which will allow us to obtain a better (smaller) collection of partitions Π_B, without sacrificing solution quality. Counter-intuitively, we will achieve this by introducing a *second* parameter: the height h of the tree decomposition. Informally, we will now construct a near-optimal solution by merging two components whenever the connection cost is low compared to the cost of (a part of) *either* component (as opposed to the lower component). As in [4], this runs the risk of charging many merging operations to a higher component, but by performing an accounting by tree decomposition level and using the fact that the decomposition only has h levels, we are able to show that our solution is still near-optimal even though we merge components much more aggressively than [4]. In this way, for each bag we construct one partition of its active terminals into a number of sets that is polynomial in $k + h + \frac{1}{\varepsilon} + \log n$, in a way that guarantees that this partition is a *refinement* of a near-optimal solution. That is, whenever we decide to place two terminals together in our partition, the near-optimal solution does the same. However, this solution does not necessarily conform to the resulting partitions, as two terminals of the same component might end up in different sets of the partition for a bag.

At this point an astute reader may be wondering that since we consider both the width k and the height h of the decomposition as parameters, we are effectively parameterizing by treedepth, rather than treewidth. This is correct, but we then go on to invoke a result of [7] which states that any tree decomposition can be rebalanced to have height $O(\log n)$ without severely increasing its width. Hence, the family of partitions we now have has size polynomial in $k + \frac{1}{\varepsilon} + \log n$. However, we are not done yet, since at this point we can only guarantee that our partitions are refinements of a near-optimal solution. To complete the algorithm, we work from this family of partitions to obtain a collection of partitions *conforming* to our near-optimal solutions using δ-nets (this is similar to the approach of [4]). This leads to a running time of the form $(\log n)^{O(\frac{k^2}{\varepsilon})} n^{O(1)}$, which by standard arguments of parameterized complexity is in fact FPT and can be upper-bounded by a function of the form $2^{O(\frac{k^2}{\varepsilon} \log \frac{k}{\varepsilon})} \cdot n^{O(1)}$. To summarize, our high-level strategy is to show that the approach of [4] can be significantly improved when the input decomposition has small width and height, but then we observe that our new scheme is efficient enough in the height that even if we replace h by a bound that can be obtained for *any* graph, we still have an algorithm with an FPT running time, that is, significantly faster than that of [4].

Vertex Cover. For the parameterization by the vertex cover size, as mentioned we obtain an FPT exact algorithm with dependence $2^{O(k \log k)}$. A similar algorithm was recently independently obtained by [8] via dynamic programming. However, our algorithm is significantly simpler, because our strategy is to show how to construct a tree decomposition and a collection of partitions Π_B such that we only need *one* partition of the active terminals for each bag. As a consequence, $p = O(n)$ and Theorem 5 implies the algorithm of Theorem 4, without the need to formulate a new dynamic program.

Our main result for this parameter is that under ETH the runtime dependence is asymptotically optimal. Note that this also implies that the runtime of the dynamic program given by Theorem 5 cannot be improved with regards to the dependence on the treewidth. To show this, we present a reduction from 3-SAT, where the goal is to compress an n-variable formula into a Steiner Forest instance such that the graph has vertex cover size $O(n/\log n)$. The intuition on why it is possible to achieve such a compression is the following: suppose we have an instance with vertex cover of size k and a demand between two vertices of the independent set. Then the simplest way to satisfy such a demand is to connect both vertices to a common neighbor in the vertex cover. This encodes a choice among k vertices, and hence it is sufficient to encode the assignment for $\log k$ binary variables. The strategy of our reduction is to set up some choice gadgets which allow us to encode the assignments to the original formula taking advantage of the fact that each choice can represent a logarithmic number of variables. Hence we can obtain a construction of slightly sub-linear ($O(n/\log n)$) size. We then of course need to add some verification gadgets, representing the clauses, to check that the formula is indeed satisfied. But even though the number of such gadgets is linear in n, we make sure that they form an independent set, and hence the total vertex cover size remains sufficiently small to obtain our lower bound. We note that this compression strategy is similar to techniques recently used to obtain slightly super-exponential lower bounds for vertex cover for other problems [26, 27], but the constructions we use are new and tailored to Steiner Forest.

Feedback Edge Set. For the parameterization by the size k of a feedback edge set, instead of relying on the dynamic program given by Theorem 5 we go an entirely different route in order to obtain the faster $2^{O(k)}n^{O(1)}$ time FPT algorithm of Theorem 3. First off, it is not hard to reduce the Steiner Forest problem to an instance in which all vertices have degree at least 2. We then consider paths with internal vertices of degree 2, with endpoints that are vertices incident to the feedback edge set or vertices of degree at least 3. We call these paths *topo-edges* and argue that there are only $O(k)$ of these. We then guess for which topo-edges the two endpoints lie in different components of the optimal Steiner Forest solution, which can be done in $2^{O(k)}$ time. If a topo-edge has both its endpoints in the same component of the optimum, we show that it can be easily handled. For the remaining topo-edges, we can decide which edges along the path do not belong to the optimal solution by a reduction to the polynomial-time solvable Min Cut problem.

1.2 Related work

Bateni, Hajiaghayi, and Marx [4] show that one of the consequences of their XP approximation scheme is a PTAS for Steiner Forest on planar graphs, by using the common technique pioneered by Baker [2] of reducing this problem to graphs for which the treewidth is bounded as a function of ε. Because their algorithm is not FPT, their PTAS has a running time of the form $n^{f(\varepsilon)}$. By using our algorithm from Theorem 1 we can improve this runtime to $f(\varepsilon)n^{O(1)}$, i.e., we obtain on EPTAS for planar graphs. However, [18] already showed that

a $(1+\varepsilon)$-approximation algorithm with a runtime of $O(f(\varepsilon) \cdot n \log^3 n)$ exists for Steiner Forest on planar graphs. While they build on the work of [4], and in particular also reduce to graphs of treewidth bounded as a function of ε, interestingly they do not obtain an EPAS parameterized by treewidth. Instead they use a different route and show that given a graph H of treewidth k, in $O(f(k,\varepsilon) \cdot n \log^2 n)$ time it is possible to compute a Steiner Forest solution in H whose cost is at most $\text{cost}(F^\star) + \varepsilon \,\text{cost}(H)$, i.e., there is an additive error that depends on the cost of H compared to the optimum solution $F^\star$. If the input graph G is planar, then a result by [9] implies that from G a so-called *banyan* [3, 30] can be computed, which is a subgraph of G with cost bounded by $O(g(\varepsilon)\,\text{cost}(F^\star))$, and which contains a near-optimal approximation of every Steiner forest (cf. [18, Lemma 2.1]). By applying the framework of [4] on the banyan instead of the input graph, it is then possible to obtain a graph H of treewidth bounded by a function of ε, for which the algorithm of [18] computes a $(1+O(\varepsilon))$-approximation for the input.

If it would be possible to compute a banyan for bounded treewidth graphs, then the algorithm of [18] would also imply an EPAS for treewidth. However, to the best of our knowledge, and as explicitly stated by [3], banyans are only known for planar graphs [9, 18], Euclidean metrics [30], and doubling metrics [3] (in fact, the latter are so-called *forest banyans*, which have weaker properties). Thus it is unclear how to obtain an EPAS for Steiner Forest parameterized by the treewidth via the algorithm of [18]. We leave open whether a banyan exists for bounded treewidth graphs, which could give an alternative algorithm to the one given in Theorem 1. However, a further remark is that the cost of the banyan for planar graphs obtained by [9] has exponential dependence on $1/\varepsilon$, which implies a double exponential runtime dependence on $1/\varepsilon$ for the EPTAS for planar graphs. If a banyan can be obtained for bounded treewidth graphs by generalizing the techniques of [9] to minor-free graphs, then the resulting EPAS parameterized by treewidth would also have double exponential runtime in $1/\varepsilon$. In this case however, our EPAS given by Theorem 1 would be exponentially faster.

A different parameter that is often studied in the context of Steiner problems is the number $p = |R|$ of terminals. The classic result of [15] presents an FPT algorithm for Steiner Tree with a runtime of $3^p n^{O(1)}$. For unweighted graphs, this was improved [5, 29] to $2^p n^{O(1)}$, while the fastest known algorithm for weighted graphs can compute the optimum in $(2+\varepsilon)^p n^{O(\sqrt{\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}})}$ time [20] for any $\varepsilon > 0$. The algorithm of [15] can be generalized to solve Steiner Forest in $2^{O(p)} n^{O(1)}$ time (cf. [12]). A somewhat dual parameter to the number of terminals is the number q of non-terminals (so-called *Steiner vertices*) in the optimum solution. For this parameter, a folklore result states that Steiner Tree (and thus also Steiner Forest) is W[2]-hard (cf. [14, 17]). However, an EPAS with a runtime of $2^{O(q^2/\varepsilon^4)} n^{O(1)}$ was shown to exist for Steiner Tree [17]. For Steiner Forest it is not hard to see that such an EPAS parameterized by q cannot exist unless P=NP (cf. [17]), but if c denotes the number of components of the optimum solution, there is an EPAS with a runtime of $(2c)^{O((q+c)^2/\varepsilon^4)} n^{O(1)}$ [17]. Similar results have been found for related Steiner problems in directed graphs [12]. For further results in the area of parameterized approximations, we refer to the survey in [19].

2 Preliminaries

As mentioned, we assume the reader is familiar with the basics of parameterized complexity, such as the class FPT [14], and approximation algorithms such as a PTAS [34]. A *parameterized approximation scheme (PAS)* is an algorithm that computes a $(1+\varepsilon)$-approximation

for a problem in $f(k,\varepsilon)n^{g(\varepsilon)}$ time for some functions f and g, while an an *efficient parameterized approximation scheme (EPAS)* is a $(1+\varepsilon)$-approximation algorithm running in time $f(k,\varepsilon)n^{O(1)}$ (that is, the running time is FPT in $k+\frac{1}{\varepsilon}$). The distinction between a PAS and an EPAS is similar to the one between a PTAS and an EPTAS.

By $w : E \to \mathbb{R}^+$ we denote an edge-weight function, so that the cost of a solution F to the Steiner Forest problem is $\text{cost}(F) = \sum_{e\in E(F)} w(e)$. We will use $F^\star$ to denote an optimal solution, and for $\alpha \geq 1$ we will say that a solution F is α-approximate if $\text{cost}(F) \leq \alpha\,\text{cost}(F^\star)$. For $u, v \in V$ we use $\text{dist}(u,v)$ to denote the shortest-path distance from u to v in G according to the weight function w.

▶ **Definition 6.** *Given a graph $G = (V,E)$, a* tree decomposition *is a pair $(T, \{B_i\}_{i\in V(T)})$, where T is a tree and each node $i \in V(T)$ of the tree is associated with a* bag *$B_i \subseteq V$, with the following properties:*

1. *$\bigcup_{i\in V(T)} B_i = V$, i.e., all vertices of G are covered by the bags,*
2. *for every edge $uv \in E$ of G there exists a node $i \in V(T)$ of the tree for which $u, v \in B_i$, and*
3. *for every vertex $v \in V$ of G the nodes $\{i \in V(T) \mid v \in B_i\}$ of the tree for which the bags contain v induce a (connected) subtree of T.*

The width *of the tree decomposition is $\max_{i\in V(T)}\{|B_i| - 1\}$ and the* treewidth *of G is the minimum width over all its tree decompositions.*

A rooted tree decomposition is nice *if for every $i \in V(T)$ we have one of the following:*

1. *i has no children*[2] *(i is a* leaf node*),*
2. *i has exactly two children i_1 and i_2 such that $B_i = B_{i_1} = B_{i_2}$ (i is a* join node*),*
3. *i has a single child i' where $B_i = B_{i'} \cup \{v\}$ for some $v \in V$ (i is an* introduce node*), or*
4. *i has a single child i' where $B_i = B_{i'} \setminus \{v\}$ for some $v \in V$ (i is a* forget node*).*

Given a rooted tree decomposition T of a graph G, for a node u of T let B be the bag associated with it. Then V_B is the set of vertices of all bags in the subtree rooted at u. The set $A_B \subseteq R$ denotes the *active terminals* of the bag B: for any demand pair $\{s,t\} \in D$, if $s \in V_B$ and $t \notin V_B$ then $s \in A_B$. For any Steiner Forest solution F, if a connected component C of F contains an active terminal, then we say that C is an *active component* for B. For a fixed solution F, we denote the set of all active components for B by $\mathcal{C}_B$. Note that every active component must intersect the bag B.

If for every bag B a set of partitions Π_B of A_B is given, a Steiner Forest solution F is *conforming* to all Π_B, if for each bag B there exists a partition $\pi \in \Pi_B$ such that any two active terminals in A_B are in the same set $S \in \pi$ if and only if they are part of the same active component C of F, i.e., $S \subseteq V(C)$ and $S' \cap V(C) = \emptyset$ for any $S' \in \pi$ with $S' \neq S$ (note that this implies $|\pi| \leq |B|$). One technicality of Theorem 5 is that the algorithm needs a nice tree decomposition as input, for which the terminals only appear in bags that are leaf nodes of the decomposition. Given any tree decomposition, these conditions are not hard to meet (cf. [4, Lemma 6]). However, for our algorithms we are going to rely on tree decompositions with certain additional properties. Hence we will need to revisit the conditions needed for the algorithm of Theorem 5 when using it for our purposes.

We will also consider the following parameters: The *treedepth* of a graph G can be defined recursively as follows: (i) the treedepth of K_1 is 1 (ii) the treedepth of a disconnected graph is the maximum of the treedepth of any of its components (iii) the treedepth of a connected graph G is $1 + \min_{v\in V(G)} \text{td}(G - v)$. A *feedback vertex set* is a set of vertices whose removal

[2] here we do not demand the leaf nodes to be empty, as is often assumed for this definition.

leaves a forest. A *vertex cover* is a set of vertices such that its removal leaves an edge-less graph. A *feedback edge set* is a set of edges whose removal leaves a forest. In a connected graph with n vertices and m edges the minimum feedback edge set always has size $m - n + 1$.

As part of our approximation algorithm we will use the notion of δ-nets, defined as follows. A well-known fact is that a δ-net exists for any metric and any $\delta \geq 0$, and it can be constructed greedily in polynomial time.

▶ **Definition 7.** *Given a metric* (X, dist)*, a* δ-net *is a subset* $N \subseteq X$ *of points, such that*
1. *any two net points* $u, v \in N$ *are far from one another, i.e.,* $\mathrm{dist}(u, v) > \delta$*, and*
2. *for any node* $u \in X$ *there is some net point* $v \in N$ *close by, i.e.,* $\mathrm{dist}(u, v) \leq \delta$*.*

3 An efficient parameterized approximation scheme for treewidth

In this section we describe the main result of this paper which is an EPAS for Steiner Forest parameterized by treewidth. We begin by giving two preliminary tools (Lemma 8 and Lemma 9) which facilitate the algorithm by ensuring that the given tree decomposition has logarithmic height and that the instance has aspect ratio (ratio of the weights of the heaviest over the lightest edge) bounded by a polynomial in n.

We then go on to Subsection 3.1 where we introduce a second parameter, the height h of the decomposition. Our goal is to fix an almost-optimal solution F_ε and describe an algorithm that produces a partition ζ_B of the active terminals for each bag B of the decomposition, where ζ_B is *a refinement* of the partition implied by F_ε (Lemma 10). In other words, we seek a partition ζ_B of A_B such that if two terminals t_1, t_2 are in the same set of ζ_B, then they are also in the same component of F_ε. Of course, it is trivial to achieve this by giving a ζ_B where each active terminal is in its own set, so the interesting part here is how we group terminals together in a way that in the end allows us to bound $|\zeta_B|$ by a polynomial of $k + h + \frac{1}{\varepsilon} + \log n$, while still ensuring that F_ε is almost optimal.

The partition ζ_B of Lemma 10 is not yet conforming, because two terminals which are in distinct sets of ζ_B may still be in the same component of F_ε, and thus we cannot apply Theorem 5 at this point. Therefore in Subsection 3.2, given ζ_B we focus on how to obtain every possible partition of the set of active terminals, which could be conforming with an almost-optimal solution. By an appropriate use of δ-nets, similar to [4], we are able to "guess" (that is, brute-force) a choice of a small number of net points per active component. Since the number of choices for each point is at most $|\zeta_B|$ and we choose roughly $O(k^2/\epsilon)$ points in total, the total number of produced partitions (and hence the running time given by Theorem 5) is of the form $(\log n + k + \frac{1}{\epsilon})^{O(k^2/\epsilon)} n^{O(1)}$, which is FPT.

Let us now recall a result of Bodlaender and Hagerup [7] which states that a tree decomposition of logarithmic height can always be obtained.

▶ **Lemma 8** ([7])**.** *Given a tree decomposition of width* k *of a graph* G *on* n *vertices, there is a polynomial time algorithm computing a nice tree decomposition of* G *of width* $O(k)$ *and height* $O(k \log n)$*.*

We also need to reduce the aspect ratio of the given graph to a polynomial. This can be done using a standard technique, where however we need to make sure that the treewidth of the given graph remains bounded. Note that the aspect ratio of the resulting graph G' in the following lemma is polynomially bounded in the size of the original graph, but not necessarily in the size of G' (because G' may have significantly fewer vertices).

▶ **Lemma 9.** *Given* $\varepsilon > 0$*, an instance of* Steiner Forest *on a graph* G *with* n *vertices, and a (nice) tree decomposition* T *of width* k *and height* h *for* G*, in polynomial time we can compute an instance on a graph* G' *with at most* n *vertices and a (nice) tree decomposition* T'

of width at most k and height h for G', such that the ratio of the longest to the shortest edge in G' is at most $2n/\varepsilon$, and any α-approximation for G' can be converted into an $(\alpha+\varepsilon)$-approximation for G.

For simplicity, in the following we will scale the edge lengths of any given graph so that the shortest edge has length 1. In particular, after applying Lemma 9, the longest edge has length at most $2n/\varepsilon$.

3.1 Tree decompositions with bounded height

In this section we informally assume that the height h of the given tree decomposition is bounded as well as the width k. Our aim is to prove the following statement, where we restrict ourselves to input graphs of polynomial aspect ratio, which we may do according to Lemma 9 (keeping in mind that n is the number of vertices of the original input graph).

▶ **Lemma 10.** *Let an instance of* Steiner Forest *on a graph G with at most n vertices be given together with a tree decomposition T of width k and height h for G. For any $\varepsilon > 0$, if the ratio between the longest and shortest edge of G is at most $2n/\varepsilon$, then there exists a $(1+\varepsilon)$-approximation F_ε with the following properties. There exists a polynomial time algorithm, which for every bag B of T outputs a partition ζ_B of the active terminals A_B, such that each set of ζ_B belongs to the same component of F_ε and $|\zeta_B| = O(\frac{k^4h^2}{\varepsilon^2}\log\frac{n}{\varepsilon})$.*

To prove Lemma 10 we first identify the solution F_ε, after which we will show how to compute the partitions ζ_B.

3.1.1 A near-optimal solution

The high-level idea to obtain a $(1+\varepsilon)$-approximate solution F_ε is to connect components of the optimum solution $F^\star$ that lie very close to each other. In particular, if the distance between two components C and C' of $F^\star$ is of the form $f(k,h,\varepsilon)\,\mathrm{cost}(C)$ for some small enough function f, then we may hope to add a shortest path between C and C' and charge this additional cost to C, in order to obtain a $(1+\varepsilon)$-approximation. Unfortunately, this approach is not viable, since the number of components that are very close to C may be very large, meaning that the function f in the distance bound would have to linearly depend on the number of vertices in order to result in a $(1+\varepsilon)$-approximation. This in turn would mean that the size of the partition ζ_B would depend polynomially on the number of vertices, making it unsuitable for an FPT time algorithm. This issue lies at the heart of the problem and is the reason for why it is non-trivial to obtain an approximation scheme parameterized by the treewidth. To get around this issue we will measure the distance between components using a modified cost function, which we define next.

Given a bag B of the rooted tree decomposition T, we denote by T_B the subtree of T rooted at the node associated with B, and by $G_B = G[V_B]$ the graph induced by the vertices V_B lying in bags of T_B. We also define the graph $G_B^\downarrow \subseteq G_B$ as the graph spanned by all edges of G_B, except those induced by B, i.e., the edge set of $G_B^\downarrow$ is

$$E(G_B^\downarrow) = \{uv \in E(G_B) \mid u \notin B \vee v \notin B\}.$$

The cost of a component C of some Steiner Forest solution restricted to $G_B^\downarrow$ only counts the edge weights of C in $G_B^\downarrow$, and is denoted by

$$\mathrm{cost}_B^\downarrow(C) = \sum_{e \in E(C) \cap E(G_B^\downarrow)} w(e).$$

Based on these definitions, we fix an optimal solution $F^\star$ and construct a solution F_ε by initially setting $F_\varepsilon = F^\star$, and then connecting components by exhaustively applying the following rule, where we say that two components C and C' *share* a bag B if $V(C) \cap B \neq \emptyset$ and $V(C') \cap B \neq \emptyset$:

▶ **Rule 1.** *If C, C' are components of $F^\star$ sharing a bag B with $\operatorname{dist}(C, C') \leq \frac{\varepsilon}{kh} \cdot \operatorname{cost}^\downarrow_B(C)$ but C and C' are in different components of F_ε, then add a shortest path of length $\operatorname{dist}(C, C')$ between C and C' to the solution F_ε.*

▶ **Lemma 11.** *The cost of the solution F_ε obtained by Rule 1 from $F^\star$ is at most $(1+\varepsilon)\operatorname{cost}(F^\star)$.*

Proof. It suffices to prove that the cost of all paths added to $F^\star$ in order to obtain F_ε according to Rule 1 is at most $\varepsilon \cdot \operatorname{cost}(F^\star)$. For this we use a charging scheme that charges new paths to components of $F^\star$. In particular, we charge a path of length $\operatorname{dist}(C, C') \leq \frac{\varepsilon}{kh} \cdot \operatorname{cost}^\downarrow_B(C)$ to component C.

Fix a component C of $F^\star$ and a bag B with $V(C) \cap B \neq \emptyset$. We define $\operatorname{charge}(C, B)$ to be the cost we charge to C for operations involving other components of $F^\star$ that share B. It is not hard to see that $\operatorname{charge}(C, B) \leq \frac{\varepsilon}{h} \cdot \operatorname{cost}^\downarrow_B(C)$, because there are at most k other components of $F^\star$ that share B.

For $\ell \in \{0, \ldots, h-1\}$, let $\mathcal{B}_\ell$ be the set of bags of the tree decomposition that appear at distance exactly ℓ from the root, i.e., they lie on level ℓ of the tree. We now observe that

$$\sum_{B \in \mathcal{B}_\ell} \operatorname{charge}(C, B) \leq \sum_{B \in \mathcal{B}_\ell} \frac{\varepsilon}{h} \cdot \operatorname{cost}^\downarrow_B(C) \leq \frac{\varepsilon}{h} \operatorname{cost}(C),$$

where the last inequality follows because if we have two bags $B, B' \in \mathcal{B}_\ell$, then $E(G^\downarrow_B) \cap E(G^\downarrow_{B'}) = \emptyset$: note that every edge of $E(G^\downarrow_B)$ must be incident on a vertex v that appears in a descendant of B, but not in B. By the properties of tree decompositions, notably by the fact that B is a separator of G, v cannot appear in B' or any of its descendants. Therefore none of its incident edges are contained in $E(G^\downarrow_{B'})$. Because $\sum_{B \in \mathcal{B}_\ell} \operatorname{cost}^\downarrow_B(C)$ is the sum of costs of C over disjoint sets of edges, the sum is a lower bound on the total cost of C.

To conclude, we observe that the total charge of C is

$$\operatorname{charge}(C) \leq \sum_{\ell=0}^{h-1} \sum_{B \in \mathcal{B}_\ell} \operatorname{charge}(C, B) \leq \varepsilon \operatorname{cost}(C).$$

Therefore, summing over all components of $F^\star$, the total cost of the edges we have added according to Rule 1 is at most $\varepsilon \cdot \operatorname{cost}(F^\star)$. ◀

3.1.2 Partitioning active terminals

We are now ready to prove Lemma 10 for the near-optimal solution F_ε constructed above, for which we will compute the partitions ζ_B for all bags B. We will use the following two claims for the active terminals A_B of the given bag B.

▷ Claim 12. If there exist $t_1, t_2 \in A_B$ such that $\operatorname{dist}(t_1, t_2) \leq \frac{\varepsilon}{kh} \operatorname{dist}(t_1, B)$, then t_1, t_2 are in the same component of F_ε.

▷ Claim 13. Let $A \subseteq A_B$ and $d \geq 0$ be such that (i) there exists $b \in B$ such that for all $t \in A$ we have $\operatorname{dist}(t, B) = \operatorname{dist}(t, b)$ and $d \leq \operatorname{dist}(t, B) \leq 2d$ (ii) for all distinct $t, t' \in A$ we have $\operatorname{dist}(t, t') > \frac{\varepsilon}{kh} d$ (iii) $|A| \geq \frac{8k^2(k+1)h^2}{\varepsilon^2}$. Then, there exists a component of F_ε that contains all terminals of A.

Intuitively, Claim 12 allows us to place terminals of A which are very close to each other into the same set of the partition ζ_B, as placing one terminal in a component forces the placement of the other. Thanks to this claim we can work with an appropriate net. If we find a large collection of such net points which also are roughly the same distance from the bag and closest to the same vertex of the bag, Claim 13 allows us to group them all together in the partition ζ_B. Armed with these tools, we can now prove the main lemma.

Proof of Lemma 10. To compute the partition ζ_B in polynomial time, we first partition the active terminals $A_B \cap B$ contained in the bag B. For this we simply add a set $\{t\}$ for each $t \in A_B \cap B$ to ζ_B, which adds at most $|B| \leq k+1$ sets to ζ_B. Let now $A = A_B \setminus B$ be the remaining active terminals.

To partition A, let $d = \min_{t \in A_B \setminus B} \operatorname{dist}(t, B)$ and $D = \max_{t \in A_B \setminus B} \operatorname{dist}(t, B)$ be the minimum and maximum distances of these active terminals from the bag B. Then partition $A_B \setminus B$ into $|B| \leq k+1$ sets $A_1, A_2, \ldots, A_{|B|}$, depending on the vertex of B that is closest to each $t \in A$ (breaking ties arbitrarily). That is, for each A_i there exists $b \in B$ such that for all $t \in A_i$ we have $\operatorname{dist}(t, B) = \operatorname{dist}(t, b)$. Consider now a set A_i and further partition it into $r = \lceil \log_2 \frac{D}{d} \rceil$ sets $A_{i,0}, A_{i,1}, \ldots, A_{i,r-1}$, where $A_{i,j}$ contains all $t \in A_i$ such that $\operatorname{dist}(t, B) \in [2^j d, 2^{j+1} d)$. Now (greedily) compute an $(\frac{\varepsilon}{kh} 2^j d)$-net $N_{i,j}$ of $A_{i,j}$. We observe that $N_{i,j}$ satisfies the first two conditions of Claim 13 for $2^j d$, so if $|N_{i,j}| \geq \frac{8k^2(k+1)h^2}{\varepsilon^2}$, then we add $A_{i,j}$ as a set of our partition ζ_B, remove the terminals of $N_{i,j}$ from A and continue the algorithm for the remaining terminals. Repeat the previous step for all i, j for which $N_{i,j}$ is sufficiently large. This contributes at most $(k+1)\lceil \log \frac{D}{d} \rceil$ sets to ζ_B. To partition A, let $d = \min_{t \in A_B \setminus B} \operatorname{dist}(t, B)$ and $D = \max_{t \in A_B \setminus B} \operatorname{dist}(t, B)$ be the minimum and maximum distances of these active terminals from the bag B. Then partition $A_B \setminus B$ into $|B| \leq k+1$ sets $A_1, A_2, \ldots, A_{|B|}$, depending on the vertex of B that is closest to each $t \in A$ (breaking ties arbitrarily). That is, for each A_i there exists $b \in B$ such that for all $t \in A_i$ we have $\operatorname{dist}(t, B) = \operatorname{dist}(t, b)$. Consider now a set A_i and further partition it into $r = \lceil \log_2 \frac{D}{d} \rceil$ sets $A_{i,0}, A_{i,1}, \ldots, A_{i,r-1}$, where $A_{i,j}$ contains all $t \in A_i$ such that $\operatorname{dist}(t, B) \in [2^j d, 2^{j+1} d)$. Now (greedily) compute an $(\frac{\varepsilon}{kh} 2^j d)$-net $N_{i,j}$ of $A_{i,j}$. We observe that $N_{i,j}$ satisfies the first two conditions of Claim 13 for $2^j d$, so if $|N_{i,j}| \geq \frac{8k^2(k+1)h^2}{\varepsilon^2}$, then we add $A_{i,j}$ as a set of our partition ζ_B, remove the terminals of $A_{i,j}$ from A and continue the algorithm for the remaining terminals. Repeat the previous step for all i, j for which $N_{i,j}$ is sufficiently large. This contributes at most $(k+1)\lceil \log \frac{D}{d} \rceil$ sets to ζ_B.

Suppose now that we are left with a set of terminals A such that the procedure above fails to construct a sufficiently large net $N_{i,j}$ to apply Claim 13. For every index pair i, j, each remaining terminal $t \in A_{i,j}$ is close enough to some net point $t' \in N_{i,j}$ such that we can apply Claim 12. We therefore create a set in the partition ζ_B for each $t' \in N_{i,j}$, placing into such a set those terminals of $A_{i,j}$ that are closest to t' (breaking ties arbitrarily). Since we cannot apply Claim 13 to the remaining sets $A_{i,j}$, each of the at most $(k+1)\lceil \log \frac{D}{d} \rceil$ nets $N_{i,j}$ has size less than $\frac{8k^2(k+1)h^2}{\varepsilon^2}$, which implies $|\zeta_B| \leq O(\frac{k^4 h^2}{\varepsilon^2} \log \frac{D}{d})$.

Clearly the above procedure can be implemented in polynomial time, and the fact that every set of ζ_B is contained in the same component of F_ε follows from Claim 12 and Claim 13. Finally, any path in a graph with at most n vertices has less than n edges, so that $\frac{D}{d} < 2n^2/\varepsilon$, given that the ratio of the longest to the shortest edge is $2n/\varepsilon$ (note that $d > 0$ by definition). Hence the claimed bound of $|\zeta_B| \leq O(\frac{k^4 h^2}{\varepsilon^2} \log \frac{n}{\varepsilon})$ follows. ◀

3.2 Tree decompositions with logarithmic height

Given a tree decomposition T of logarithmic height, using Lemma 10 we are ready to compute a set of partitions Π_B of FPT size for each bag B, such that a near-optimal solution conforms to Π_B. In particular, by Lemma 8 we may assume that the height of T is $h = O(k \log n)$, which means that the bound on ζ_B in Lemma 10 translates to $O(\frac{k^6}{\varepsilon^2} \log^3 \frac{n}{\varepsilon})$. As in the previous section, we need to apply Lemma 9 in order to bound the aspect ratio of the graph, so that n denotes the number of vertices of the original input graph, while now the graph G has at most n vertices but the ratio between the longest and shortest edge is at most $2n/\varepsilon$. We begin by describing how to obtain the near-optimal solution, after which we will identify the partition sets Π_B.

3.2.1 A near-optimal solution

Bateni, Hajiaghayi, and Marx [4] construct a near-optimal solution by modifying the optimum. We will use similar techniques to obtain our near-optimal solution, but we construct it by instead modifying the $(1+\varepsilon)$-approximate solution F_ε given by Lemma 10. In particular, we construct a near-optimal $(1+\varepsilon)^2$-approximation $\widetilde{F}_\varepsilon$ from F_ε. The main idea to obtain $\widetilde{F}_\varepsilon$ is to connect components of F_ε if they are very close to one another. As before however, doing this naively would incur too much cost for the additional connections.

To make sure that the cost incurred by connecting components of F_ε is not too large, [4] introduced a partial order on the components based on the structure of a given rooted tree decomposition T. Let C_1, C_2 be two components of F_ε that share a bag B of T, i.e., $V(C_1) \cap B \neq \emptyset$ and $V(C_2) \cap B \neq \emptyset$. Since C_1 and C_2 are connected subgraphs of the input graph, a basic property of tree decompositions implies that there are (connected) subtrees T_1 and T_2 of T induced by the respective bags containing vertices of C_1 and C_2. Because these components both contain vertices of B, the node associated with B is part of both T_1 and T_2, and therefore the roots of both subtrees lie on the path from this node to the root of T. This defines an order on C_1 and C_2, and we write $C_1 \leq C_2$ if the root of T_1 is farther from the root of T than the root of T_2 is. This order is defined for any two components of F_ε that share a bag, and thus we obtain a partial order on the components of F_ε, where any components that do not share a bag are incomparable.

Using the defined order, [4] connect components of the optimum solution that are very close to each other. In order to obtain smaller partition sets, we modify the distance bound used in this procedure compared to [4]. In particular, for any value $x > 0$, let $\lfloor x \rfloor_2 = 2^{\lfloor \log_2 x \rfloor}$ denote the largest power of 2 that is at most x. Now, starting with $\widetilde{F}_\varepsilon = F_\varepsilon$ we connect components by exhaustively applying the following rule:

▶ **Rule 2.** *If C, C' are components of F_ε with $C \leq C'$ and $\operatorname{dist}(C, C') \leq \frac{\varepsilon}{k} \lfloor \operatorname{cost}(C) \rfloor_2$ but C and C' lie in different components of $\widetilde{F}_\varepsilon$, then add a shortest path of length $\operatorname{dist}(C, C')$ between C and C' to the solution $\widetilde{F}_\varepsilon$.*

A crucial but subtle observation is that for a component C of F_ε there can be many components $C' \leq C$ at distance at most $\frac{\varepsilon}{k} \lfloor \operatorname{cost}(C) \rfloor_2$ to C, which however are not connected to C in the resulting solution $\widetilde{F}_\varepsilon$ according to Rule 2. This makes it non-trivial to find small partition sets Π_B. Contrary to this however, an important property of the order on the components is that for any component C of F_ε, there are at most k other components C' for which $C \leq C'$, as we will argue for the following lemma to bound the cost of $\widetilde{F}_\varepsilon$. In particular, the lemma implies that $\widetilde{F}_\varepsilon$ is a near-optimal $(1+\varepsilon)^2$-approximation, given that F_ε is a $(1+\varepsilon)$-approximation.

▶ **Lemma 14.** *The cost of the solution $\widetilde{F}_\varepsilon$ obtained by Rule 2 from F_ε is at most $(1+\varepsilon) \operatorname{cost}(F_\varepsilon)$.*

3.2.2 Partitioning active terminals

Given the construction of the $(1+\varepsilon)^2$-approximate solution $\widetilde{F}_\varepsilon$ above, the next step is to find a set of partitions Π_B of the active terminals A_B for each bag B, such that $\widetilde{F}_\varepsilon$ conforms with all sets Π_B. In the following, fix a bag B of the given tree decomposition T. The technique used by [4] is to guess a small net for each active component of bag B,[3] so that every terminal of A_B close to a net point must be part of the same component in the approximate solution, after taking the order on the active components as defined previously into account. Next we choose a net on the terminals of each active component and bound its size.

▶ **Lemma 15.** *Let $N \subseteq A_B \cap C$ be an $\frac{\varepsilon}{k}\lfloor \mathrm{cost}(C)\rfloor_2$-net of the metric induced by the active terminals of some active component C. The size of the net can be bounded by $|N| \leq \lfloor 4k/\varepsilon \rfloor$.*[4]

Following the algorithm of [4], the next step would be to guess such an $\frac{\varepsilon}{k}\lfloor \mathrm{cost}(C)\rfloor_2$-net for each of the at most $k+1$ active components C of the bag B. By Lemma 15, the total number of net points for these at most $k+1$ nets is at most $\lfloor 4k/\varepsilon \rfloor (k+1) = O(k^2/\varepsilon)$. Since however there may be up to n active terminals, guessing these nets for all active components can result in $n^{O(k^2/\varepsilon)}$ many possible choices, which leads to an XP time algorithm. To circumvent this, we instead consider the partition ζ_B of the active terminals as given by Lemma 10, and guess which of the sets of ζ_B contains a net point. We will argue that since the size of ζ_B is $O(\frac{k^6}{\varepsilon^2}\log^3\frac{n}{\varepsilon})$ there are only $(\frac{k}{\varepsilon}\log\frac{n}{\varepsilon})^{O(k^2/\varepsilon)}$ possibilities, leading to a faster algorithm.

More concretely, to compute a set of partitions Π_B that $\widetilde{F}_\varepsilon$ conforms to, our algorithm considers every sequence $((S_1,\delta_1),(S_2,\delta_2),\ldots,(S_\ell,\delta_\ell),\rho)$ of at most $k+1$ pairs (S_j,δ_j) and partitions ρ of the index set $\{1,\ldots,\ell\}$, where each S_j is a subset of the parts of ζ_B such that $|S_j| \leq \lfloor 4k/\varepsilon \rfloor$, and $\delta_j \in \{2^q \mid q \in \mathbb{N}_0 \wedge 0 \leq q \leq \log_2(2n^2/\varepsilon)\}$ is an integer power of 2 between 1 and $2n^2/\varepsilon$, where n is the number of vertices of the original input graph in accordance with Lemma 9. From every such sequence, the algorithm attempts to construct a partition of the active terminals, and if it succeeds adds it to the set Π_B. As we will show, in this process the algorithm will successfully construct one partition π of A_B that $\widetilde{F}_\varepsilon$ conforms to.

Before describing how a partition of the active terminals arises from such a sequence, we bound the number of these sequences, which determines the running time. By Lemma 10, $|\zeta_B| = O(\frac{k^6}{\varepsilon^2}\log^3\frac{n}{\varepsilon})$ if the tree decomposition T has logarithmic height, so that there are at most $\binom{|\zeta_B|}{\lfloor 4k/\varepsilon \rfloor} = (\frac{k}{\varepsilon}\log\frac{n}{\varepsilon})^{O(k/\varepsilon)}$ possible choices for each S_j. Clearly there are $O(\log\frac{n}{\varepsilon})$ choices for each δ_j, and $\ell^\ell = k^{O(k)}$ possible partitions ρ, given that $\ell \leq k+1$. Since a sequence contains ℓ sets S_j, the total number of sequences is bounded by $(\frac{k}{\varepsilon}\log\frac{n}{\varepsilon})^{O(k^2/\varepsilon)}$.

Each sequence may give rise to a partition $\pi \in \Pi_B$ of the active terminals as follows. First, let $\pi = \{Y_1,\ldots,Y_{|\rho|}\}$, i.e., π has the same number of sets as the partition ρ. Let $U_j = \bigcup_{U\in S_j} U$ denote the set of active terminals in S_j, and let $\rho(j)$ be the part of ρ containing j. We distinguish between active terminals $t \in A_B$ that lie in some set U_j and those that do not:

- if $t \in U_j$ for some $j \in [\ell]$ then $t \in Y_{\rho(j)}$ (i.e., $U_j \subseteq Y_{\rho(j)}$), and
- otherwise, if $p_t \in \{1,\ldots,\ell\}$ denotes the smallest index for which $\mathrm{dist}(t,U_{p_t}) \leq \frac{\varepsilon}{k}\delta_{p_t}$, then $t \in Y_{\rho(p_t)}$.

If this π is a partition of A_B we add π to Π_B, and otherwise we dismiss the current sequence. Clearly π can be constructed in polynomial time, given a sequence.

[3] [4] refers to these nets as *groups*.
[4] A slightly worse bound follows from [4, Lemma 19].

▶ **Lemma 16.** *The $(1+\varepsilon)^2$-approximate solution $\widetilde{F}_\varepsilon$ conforms to the set Π_B of partitions constructed above.*

Proof. Consider the $(1+\varepsilon)$-approximate solution F_ε of Lemma 10 from which $\widetilde{F}_\varepsilon$ is constructed according to Rule 2, and the partition ζ_B of A_B as given by Lemma 10. Let the active components of F_ε be $C_1, \ldots, C_\ell$ indexed according to their order, i.e., $C_j \leq C_{j'}$ if and only if $j \leq j'$. For each active component C_j we fix an $\frac{\varepsilon}{k}\lfloor \mathrm{cost}(C_j)\rfloor_2$-net N_j of size at most $\lfloor 4k/\varepsilon \rfloor$ according to Lemma 15. Now, consider the sequence $((S_1,\delta_1),(S_2,\delta_2),\ldots,(S_\ell,\delta_\ell),\rho)$, where

- S_j contains exactly those sets of ζ_B that contain at least one net point of N_j,
- $\delta_j = \lfloor \mathrm{cost}(C_j)\rfloor_2$, and
- ρ is the partition of the index set corresponding to the components of $\widetilde{F}_\varepsilon$, i.e., $\rho(j) = \rho(j')$ if and only if C_j and $C_{j'}$ lie in the same component in $\widetilde{F}_\varepsilon$.

Recall that after applying Lemma 9 to the input, the ratio between the shortest and longest edge is at most $2n/\varepsilon$, where n is the number of vertices of the original input graph. Since we assume that the length of the shortest edge is 1, the cost of any component lies between 1 and $2n^2/\varepsilon$, given that a component is a tree with less than n edges. Therefore $\lfloor \mathrm{cost}(C_j)\rfloor_2 \in \{2^q \mid q \in \mathbb{N}_0 \wedge 0 \leq q \leq \log_2(2n^2/\varepsilon)\}$, which means that the algorithm will consider the above sequence in some iteration.

We now turn to $\pi = \{Y_1, \ldots, Y_{|\rho|}\}$ constructed for this sequence, and show that it is a partition of A_B and that $\widetilde{F}_\varepsilon$ conforms to it. For this, note that no set of ζ_B contains net points of several active components of F_ε, since by Lemma 10 all active terminals in the same set of ζ_B also belong to the same component of F_ε. Thus the sets S_j as defined above (and also the corresponding sets U_j) are pairwise disjoint. This means that, due to the definition of ρ, any two terminals $t \in U_j$ and $t' \in U_{j'}$ end up in the same set of π if and only if t and t' belong to the same component of $\widetilde{F}_\varepsilon$ (as $U_j \subseteq Y_{\rho(j)}$).

Now consider a terminal $t \in A_B$, which does not lie in any U_j, and let q be the index of the active component C_q of F_ε containing t. As $\delta_q = \lfloor \mathrm{cost}(C_q)\rfloor_2$, N_q is an $\frac{\varepsilon}{k}\delta_q$-net of $C_q \cap A_B$. Also, we chose S_q so that $N_q \subseteq U_q$. Hence we get $\mathrm{dist}(t, U_q) \leq \mathrm{dist}(t, N_q) \leq \frac{\varepsilon}{k}\delta_q$, and the definition of p_t implies $p_t \leq q$. Now C_{p_t} is either equal to C_q, or C_q is connected to the component C_{p_t} in the approximate solution $\widetilde{F}_\varepsilon$ according to Rule 2: on one hand we have $C_{p_t} \leq C_q$ due to the order of the indices, and at the same time by Lemma 10 we have $U_{p_t} \subseteq V(C_{p_t}) \cap A_B$, which implies

$$\mathrm{dist}(C_q, C_{p_t}) \leq \mathrm{dist}(t, C_{p_t}) \leq \mathrm{dist}(t, U_{p_t}) \leq \frac{\varepsilon}{k}\delta_{p_t} = \frac{\varepsilon}{k}\lfloor \mathrm{cost}(C_{p_t})\rfloor_2.$$

Hence we can conclude that t lies in the same component as C_{p_t} in $\widetilde{F}_\varepsilon$.

In conclusion, adding U_j to $Y_{\rho(j)}$ and t to $Y_{\rho(p_t)}$ for each terminal t not lying in any U_j, partitions the terminals according to the components of $\widetilde{F}_\varepsilon$. Hence π is a partition of the active terminals A_B that is added to Π_B, and $\widetilde{F}_\varepsilon$ conforms to it. ◀

Using all of the above, we can finally prove our main theorem, stating that there is an EPAS for Steiner Forest parameterized by the treewidth.

Proof of Theorem 1. The first steps of our algorithm are to preprocess the given tree decomposition using Lemma 8 so that it is nice and its height is $O(k \log n)$, and the input graph using Lemma 9 so that the aspect ratio is bounded (which means that n denotes the number of vertices in the original input graph). We then compute the partition sets Π_B for all bags B using the above procedure, resulting in partition sets of size $(\frac{k}{\varepsilon}\log\frac{n}{\varepsilon})^{O(k^2/\varepsilon)} = 2^{O(\frac{k^2}{\varepsilon}\log\frac{k}{\varepsilon})} \cdot n^{o(1)}$. Here, we are using a well-known Win/Win argument: if $k^2/\varepsilon < \sqrt{\log n}$, then $(\log n)^{k^2/\varepsilon} = n^{o(1)}$; otherwise, $\log n \leq k^4/\varepsilon^2$, therefore $(\frac{k}{\varepsilon}\log\frac{n}{\varepsilon})^{O(k^2/\varepsilon)} = (\frac{k}{\varepsilon})^{O(\frac{k^2}{\varepsilon})}$.

Since each partition of a set Π_B can be computed in polynomial time, and the number of bags of the nice tree decomposition is $O(kn)$, this takes $2^{O(\frac{k^2}{\varepsilon}\log\frac{k}{\varepsilon})} \cdot n^{O(1)}$ time. Next we apply Theorem 5 to compute a solution that is at least as good as $\widetilde{F}_\varepsilon$ conforming to all Π_B, in $2^{O(\frac{k^2}{\varepsilon}\log\frac{k}{\varepsilon})} \cdot n^{O(1)}$ time. Hence we obtain a $(1+\varepsilon)^2$-approximation F. According to Lemma 9, F can be converted into a $((1+\varepsilon)^2+\varepsilon)$-approximation to the original input graph. Since for any $\varepsilon' > 0$ we may choose $\varepsilon = \Theta(\varepsilon')$ so that $((1+\varepsilon)^2+\varepsilon) \leq 1+\varepsilon'$, we obtain an EPAS as claimed. ◀

4 Vertex cover

In this section we consider the parameterization by the size of a *vertex cover*, which is a set $S \subseteq V$ of vertices such that every edge is incident on at least one of the vertices of S. In the full version of this paper we present an easy FPT algorithm based on Theorem 5.

Our goal here is to present a reduction showing that the algorithm we have given for Steiner Forest parameterized by vertex cover is essentially optimal, assuming the ETH. Recall that the ETH is the hypothesis that 3-SAT on instances with n variables cannot be solved in time $2^{o(n)}$. We will give a reduction that given a 3-SAT instance ϕ, produces an equivalent Steiner Forest instance with vertex cover at most $O(n/\log n)$. We stress that our reduction works even for unweighted instances.

▶ **Theorem 17.** *If there exists an algorithm which, given an unweighted Steiner Forest instance on n vertices with vertex cover k, finds an optimal solution in time $2^{o(k\log k)}n^{O(1)}$, then the ETH is false.*

Proof. We present a reduction from 3-SAT. Before we proceed, we would like to add to our formula the requirement that the variable set comes partitioned into three sets in a way that each clause contains at most one variable from each set. It is not hard to show that this does not affect the complexity of the instance much, as we demonstrate in the following claim.

▷ Claim 18. Suppose that there exists an algorithm that takes as input a 3-SAT instance ϕ on n variables and a partition of the variables into three sets of equal size, such that each clause contains at most one variable from each set, and decides if ϕ is satisfiable in time $2^{o(n)}$. Then, the ETH is false.

In the remainder we will then assume that we are given a formula ϕ on $3n$ variables which are partitioned into three sets of size n as specified by the previous claim. Without loss of generality, suppose that n is a power of 4 (this can be achieved by adding dummy variables). Note that this ensures that $\frac{\log n}{2}$ and $\sqrt{n}$ are both integers.

We construct an equivalent instance of Steiner Forest as follows. Let $L = \lceil \frac{n}{\log^2 n} \rceil$. We begin by constructing i choice gadgets, i.e., for $i \in \{1, \ldots, 3\log n\}$ we make:

- $2L$ left vertices, labeled ℓ^i_j, for $j \in \{0, \ldots, 2L-1\}$.
- $2L$ right vertices, labeled r^i_j, for $j \in \{0, \ldots, 2L-1\}$.
- $\sqrt{n}$ middle vertices, labeled m^i_j, for $j \in \{0, \ldots, \sqrt{n}-1\}$.
- We connect all middle vertices to all left and right vertices, that is, for all $j \in \{0, \ldots, 2L-1\}$ and $j' \in \{0, \sqrt{n}-1\}$ we connect ℓ^i_j and r^i_j to $m^i_{j'}$.
- For each $j \in \{0, \ldots, 2L-1\}$ we add a demand from ℓ^i_j to r^i_j.

Notice that the graph we have constructed so far contains $3\log n$ choice gadgets, each of which has $4L + \sqrt{n} = O(n/\log^2 n)$ vertices, so the graph at the moment contains $O(n/\log n)$ vertices in total.

Before we proceed, let $X = X_a \cup X_b \cup X_c$ be the set of $3n$ variables of ϕ that was given to us partitioned into three sets of size n. We partition X into $3\log n$ groups $X_1, \ldots, X_{3\log n}$ in a way that (i) $|X_i| \leq \lceil n/\log n \rceil$ for all $i \in \{1, \ldots, \log n\}$ and (ii) for all $i \in \{1, \ldots, \log n\}$ we have X_i is contained in one of X_a, X_b, X_c. This can be done by taking the n variables of X_a and partitioning them arbitrarily into groups $X_1, \ldots, X_{\log n}$ of size as equal as possible (therefore at most $\lceil n/\log n \rceil$), and we proceed similarly for X_b, X_c. Rename the variables of ϕ so that for each i we have that $X_i = \{x_{(i,0)}, \ldots, x_{(i,\lceil n/\log n \rceil - 1)}\}$.

To give some intuition, we will now say that, for $i \in \{1, \ldots, 3\log n\}$, the choice gadget i represents the variables of the set X_i. In particular, for each $j \in \{0, \ldots 2L-1\}$, we will say that the way that the demand $\ell_j^i \to r_j^i$ was satisfied encodes the assignment to the $\frac{\log n}{2}$ variables $\{x_{(i,\frac{j\log n}{2})}, \ldots, x_{(i,\frac{(j+1)\log n}{2}-1)}\}$. More precisely, in our intended solution the demand $\ell_j^i \to r_j^i$ is satisfied by connecting both terminals to a common middle vertex $m_{j'}^i$. We can infer the assignment to the $\frac{\log n}{2}$ variables this represents simply by writing down the binary representation of j', which is a number between 0 and $\sqrt{n}-1$, hence a number with $\frac{\log n}{2}$ bits. Note that this way we represent $2L \cdot \frac{\log n}{2} \geq \lceil \frac{n}{\log n} \rceil$ variables, that is, we can represent the assignment to all the variables of the group.

Armed with this intuition, we can now complete our construction. For each clause c we construct two new vertices, c_1, c_2 and add a demand from c_1 to c_2. For each literal contained in c, suppose that the literal involves the variable $x_{(i,\frac{j\log n}{2}+\alpha)}$ for $i \in \{1, \ldots, 3\log n\}$, $j \in \{0, \ldots, 2L-1\}$, $\alpha \in \{0, \ldots, \frac{\log n}{2} - 1\}$. We then connect c_1 to ℓ_j^i. Furthermore, if $x_{(i,\frac{j\log n}{2}+\alpha)}$ appears positive in c, we connect c_2 to all $m_{j'}^i$ such that the binary representation of j' has a 1 in position α. If on the other hand $x_{(i,\frac{j\log n}{2}+\alpha)}$ appears negative in c, we connect c_2 to all $m_{j'}^i$ such that the binary representation of j' has a 0 in position α. In other words, we connect c_2 to all the middle vertices to which ℓ_j^i could be connected and are consistent with an assignment that satisfies c using the current literal. After repeating the above for all literals of each clause the construction is complete. We set the target cost to be $B = 2m + 12L\log n$.

Before we argue about the correctness of the reduction, let us observe that if the reduction preserves the satisfiability of ϕ, then we obtain the theorem, because the instance we constructed has vertex cover $k = O(n/\log n)$ and size polynomial in the size of ϕ. Indeed, as we argued the choice gadgets have $O(n/\log n)$ vertices in total, and all further edges we added have an endpoint in a choice gadget. If there was an algorithm solving the new instance in time $k^{o(k)}n^{O(1)}$, this would give a $2^{o(n)}$ algorithm to decide ϕ.

Regarding correctness, let us first observe that if ϕ is satisfiable, we can obtain a valid solution using the intuitive translation from assignments to choice gadget solutions we gave above. In particular, for each $i \in \{1, \ldots, 3\log n\}$ and $j \in \{0, \ldots, 2L-1\}$, we consider the assignment to variables $\{x_{(i,\frac{j\log n}{2})}, \ldots, x_{(i,\frac{(j+1)\log n}{2}-1)}\}$ as a binary number, which must have a value j' between 0 and $\sqrt{n}-1$. We then connect both ℓ_j^i, r_j^i to $m_{j'}^i$. Repeating this satisfies all demands internal to choice gadgets and uses $3\log n \cdot 4L = 12L\log n$ edges. Consider now a clause c and the demand from c_1 to c_2. Since we started with a satisfying assignment, c must contain a true literal, say involving the variable $x_{(i,\frac{j\log n}{2}+\alpha)}$. We select the edge from c_1 to ℓ_j^i. Furthermore, we observe that c_2 must be a neighbor of all vertices $m_{j'}^i$ such that the bit in position α of the binary representation of j' agrees with the value of $x_{(i,\frac{j\log n}{2}+\alpha)}$. Since ℓ_j^i is already connected to such a $m_{j'}^i$, we select the edge from that vertex to c_2 to satisfy the demand for this clause. We have therefore spent $2m$ further edges for the clause demands and have used a budget of exactly B.

For the converse direction, suppose we have a solution of cost B. We first observe that each vertex r^i_j must be connected to a middle vertex $m^i_{j'}$, since all right vertices are terminals, but such vertices only have edges connecting them to middle vertices. Recall that, for each i, j, the left vertex ℓ^i_j must be in the same component of the solution as r^i_j, since there is a demand between these two vertices. Hence, each ℓ^i_j is in the same component of the solution as some $m^i_{j'}$. We now slightly edit the solution as follows: suppose there exists a vertex ℓ^i_j which is not directly connected in the solution to any middle vertex $m^i_{j'}$. Since this vertex is in the same component as one such vertex $m^i_{j'}$, we add to the solution the edge connecting them, and since this creates a cycle, remove from the solution another edge incident on ℓ^i_j. Doing this repeatedly ensures that each ℓ^i_j is connected to a middle vertex $m^i_{j'}$ in the solution without increasing the total cost.

We now observe that since each ℓ^i_j and each r^i_j is connected to at least one middle vertex $m^i_{j'}$ in the solution, this already uses a cost of $3 \log n \cdot 4L = 12L \log n$. Furthermore, for each clause we have constructed two terminals, each of which must use at least one of its incident edges, giving an extra cost of $2m$. Since our budget is exactly $2m + 12L \log n$, we conclude that each terminal constructed for a clause is incident on exactly one edge, and each ℓ^i_j and each r^i_j is connected to exactly one middle vertex. Crucially, these observations imply the following fact: if for some i, j, j' we have that ℓ^i_j and $m^i_{j'}$ are in the same component of the solution, then the edge connecting ℓ^i_j and $m^i_{j'}$ is part of the solution. To see this, observe that any path connecting ℓ^i_j and $m^i_{j'}$ that is not a direct edge would need to have length at least 3. However, no clause terminal can be an internal vertex of such a path, since clause terminals have degree 1 in the solution. Furthermore, if we remove clause terminals from the graph, left and right vertices also have degree 1 in the remaining solution, so such vertices also cannot be internal in the path. Finally, middle vertices are an independent set, so it is impossible for all internal vertices of a path of length at least 3 to be middle vertices.

Armed with the observation that ℓ^i_j and $m^i_{j'}$ are in the same connected component of the solution if and only if they are directly connected, we are ready to extract a satisfying assignment from the Steiner forest. For each i, j, if ℓ^i_j is connected to $m^i_{j'}$ we write j' in binary and assign to variable $x_{(i, \frac{j \log n}{2} + \alpha)}$, for $\alpha \in \{0, \ldots, \frac{logn}{2} - 1\}$ the value in position α of the binary representation of j'. We claim that this assignment must be satisfying. Indeed, consider the clause c, and the terminals c_1, c_2 which represent it. Since these terminals have a demand, they must be in the same component. Because c_1 has at most three neighbors which are in different choice gadgets (as each clause contains variables from distinct groups), we can see that c_1 must be connected to some ℓ^i_j and c_2 to some $m^i_{j'}$ in the solution, such that ℓ^i_j and $m^i_{j'}$ are in the same component, and are therefore directly connected. But if ℓ^i_j is directly connected to $m^i_{j'}$ this means that the assignment we extracted from ℓ^i_j gives a value to a variable $x_{(i, \frac{j \log n}{2} + \alpha)}$ which satisfies the clause c, hence we have a satisfying assignment. ◀

5 Feedback Edge Set

A *feedback edge set* of a graph is a set of edges that when removed renders the graph acyclic. It is well-known that if G is a connected undirected graph on n vertices and m edges, then all minimal feedback edge sets of G have size $k = m - n + 1$. Indeed, such a set can be constructed in polynomial time by repeatedly locating a cycle in the graph and selecting an arbitrary edge of the cycle to insert into the feedback edge set.

In this section we will consider Steiner Forest parameterized by the feedback edge set of the input graph, which we will denote by k. Unlike the vertex cover section, here our main result is positive: we show that Steiner Forest can be solved optimally in time

$2^{O(k)}n^{O(1)}$, that is, in time single-exponential in the parameter. Since we are able to achieve a single-exponential dependence, it is straightforward to see that this is optimal under the ETH.

▶ **Theorem 19.** *If there is an algorithm solving* Steiner Tree *in time* $2^{o(k)}n^{O(1)}$*, where* k *is the feedback edge set of the input, then the ETH is false.*

Let us now proceed to the detailed presentation of the algorithm. Suppose that we are given a budget b and we want to decide if there exists a Steiner Forest solution F such that $\text{cost}(F) \leq b$. We start by applying a simple reduction rule.

▶ **Rule 3.** *Suppose we have a* Steiner Forest *instance on graph* G *with weight function* w *and budget* b*, such that a vertex* $u \in V$ *has degree* 1*. If* $u \notin R$*, then delete* u*. If* $u \in R$*, let* v *be the unique neighbor of* u*. Then set* $b' := b - w(uv)$*, delete* u *from the graph and the demand* $\{u, v\}$ *from* D *if it exists, and replace, for each* $x \in V \setminus \{u, v\}$ *such that* $\{u, x\} \in D$ *the demand* $\{u, x\}$ *with the demand* $\{v, x\}$*.*

▶ **Lemma 20.** *Rule 3 is safe.*

Observe that if we apply Rule 3 exhaustively, then the minimum degree of the graph is 2. As we show next, relatively few vertices can have higher degree.

▶ **Lemma 21.** *Suppose we have a* Steiner Forest *instance with feedback edge set of size* k *and minimum degree at least* 2*. Then* G *contains at most* $2k$ *vertices of degree at least* 3*.*

In the remainder we will assume that we have a Steiner Forest instance $G = (V, E)$ with a feedback edge set $H \subseteq E$ of size k, to which 3 can no longer be applied. We will say that a vertex v is *special* if v is incident on an edge of H or v has degree at least 3. By Lemma 21 we know that G contains at most $4k$ special vertices.

We define a *topological edge* (topo-edge for short) as follows: a path P in G is a topological edge if the two endpoints of P are special vertices and all internal vertices of P are non-special. Note that by this definition, all edges of H form topo-edges, since the endpoints of such edges are special. We observe the following:

▶ **Lemma 22.** *Suppose we have a graph* G *with feedback edge set of size* k *and minimum degree at least* 2*. Then* G *contains at most* $5k$ *topological edges.*

We are now ready to state the main algorithmic result of this section.

▶ **Theorem 23.** *There is an algorithm that solves* Steiner Forest *on instances with* n *vertices and a feedback edge set of size* k *in* $2^{O(k)}n^{O(1)}$ *time.*

References

1 Ajit Agrawal, Philip Klein, and Ramamoorthi Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 134–144, 1991.

2 Brenda S Baker. Approximation algorithms for np-complete problems on planar graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994.

3 Yair Bartal and Lee-Ad Gottlieb. Near-linear time approximation schemes for steiner tree and forest in low-dimensional spaces. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1028–1041, 2021.

4 MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Dániel Marx. Approximation schemes for steiner forest on planar graphs and graphs of bounded treewidth. *Journal of the ACM (JACM)*, 58(5):1–37, 2011.

5 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: fast subset convolution. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 67–74, 2007.

6 Hans L Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015.

7 Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM J. Comput.*, 27(6):1725–1746, 1998. doi:10.1137/S0097539795289859.

8 Hans L Bodlaender, Matthew Johnson, Barnaby Martin, Jelle J Oostveen, Sukanya Pandey, Daniel Paulusma, Siani Smith, and Erik Jan van Leeuwen. Complexity framework for forbidden subgraphs iv: The steiner forest problem. *arXiv preprint*, 2023. arXiv:2305.01613.

9 Glencora Borradaile, Philip Klein, and Claire Mathieu. An o (n log n) approximation scheme for steiner tree in planar graphs. *ACM Transactions on Algorithms (TALG)*, 5(3):1–31, 2009.

10 Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *Journal of the ACM (JACM)*, 60(1):1–33, 2013.

11 Xiuzhen Cheng and Ding-Zhu Du. *Steiner trees in industry*, volume 11. Springer Science & Business Media, 2013.

12 Rajesh Chitnis, Andreas Emil Feldmann, and Pasin Manurangsi. Parameterized approximation algorithms for bidirected steiner network problems. *ACM Transactions on Algorithms (TALG)*, 17(2):1–68, 2021.

13 Miroslav Chlebík and Janka Chlebíková. The steiner tree problem on graphs: Inapproximability results. *Theoretical Computer Science*, 406(3):207–214, 2008.

14 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

15 Stuart E Dreyfus and Robert A Wagner. The steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

16 Ding-Zhu Du, JM Smith, and J Hyam Rubinstein. *Advances in Steiner trees*, volume 6. Springer Science & Business Media, 2013.

17 Pavel Dvorák, Andreas E Feldmann, Dusan Knop, Tomás Masarík, Tomas Toufar, and Pavel Vesely. Parameterized approximation schemes for steiner trees with small number of steiner vertices. *SIAM Journal on Discrete Mathematics*, 35(1):546–574, 2021.

18 David Eisenstat, Philip Klein, and Claire Mathieu. An efficient polynomial-time approximation scheme for steiner forest in planar graphs. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 626–638. SIAM, 2012.

19 Andreas Emil Feldmann, Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020.

20 Bernhard Fuchs, Walter Kern, D Molle, Stefan Richter, Peter Rossmanith, and Xinhui Wang. Dynamic programming for minimum steiner trees. *Theory of Computing Systems*, 41(3):493–500, 2007.

21 Elisabeth Gassner. The steiner forest problem revisited. *Journal of Discrete Algorithms*, 8(2):154–163, 2010.

22 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theor. Comput. Sci.*, 918:60–76, 2022. doi:10.1016/J.TCS.2022.03.021.

23 Anupam Gupta and Jochen Könemann. Approximation algorithms for network design: A survey. *Surveys in Operations Research and Management Science*, 16(1):3–20, 2011.

24 Frank K Hwang and Dana S Richards. Steiner tree problems. *Networks*, 22(1):55–89, 1992.

25 Richard M Karp. On the computational complexity of combinatorial problems. *Networks*, 5(1):45–68, 1975.

26 Michael Lampis, Nikolaos Melissinos, and Manolis Vasilakis. Parameterized max min feedback vertex set. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France*, volume 272 of *LIPIcs*, pages 62:1–62:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.MFCS.2023.62.

27 Michael Lampis and Manolis Vasilakis. Structural parameterizations for two bounded degree problems revisited. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 77:1–77:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.77.

28 Ivana Ljubic. Solving steiner trees: Recent advances, challenges, and perspectives. *Networks*, 77(2):177–204, 2021.

29 Jesper Nederlof. Fast polynomial-space algorithms using möbius inversion: Improving on steiner tree and related problems. In *International Colloquium on Automata, Languages, and Programming*, pages 713–725. Springer, 2009.

30 Satish B Rao and Warren D Smith. Approximating geometrical graphs via "spanners" and "banyans". In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 540–550, 1998.

31 R Ravi. A primal-dual approximation algorithm for the steiner forest problem. *Information processing letters*, 50(4):185–189, 1994.

32 Hao Tang, Genggeng Liu, Xiaohua Chen, and Naixue Xiong. A survey on steiner tree construction and global routing for vlsi design. *IEEE Access*, 8:68593–68622, 2020.

33 Stefan Voß. Steiner tree problems in telecommunications. In *Handbook of optimization in telecommunications*, pages 459–492. Springer, 2006.

34 David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

An FPRAS for Two Terminal Reliability in Directed Acyclic Graphs

Weiming Feng
Institute for Theoretical Studies, ETH Zürich, Switzerland

Heng Guo
School of Informatics, University of Edinburgh, UK

Abstract

We give a fully polynomial-time randomized approximation scheme (FPRAS) for two terminal reliability in directed acyclic graphs (DAGs). In contrast, we also show the complementing problem of approximating two terminal unreliability in DAGs is #**BIS**-hard.

2012 ACM Subject Classification Networks → Network reliability; Mathematics of computing → Approximation algorithms; Theory of computation → Generating random combinatorial structures

Keywords and phrases Approximate counting, Network reliability, Sampling algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.62

Category Track A: Algorithms, Complexity and Games

Related Version *Preprint*: `https://arxiv.org/abs/2310.00938`

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 947778). Weiming Feng acknowledges support of Dr. Max Rössler, the Walter Haefner Foundation and the ETH Zürich Foundation.

Acknowledgements We thank Kuldeep S. Meel for bringing the problem to our attention, Antoine Amarilli for explaining their method to us, and Marcelo Arenas for insightful discussions. We also thank Zongchen Chen for suggesting a better presentation of Theorem 1, and Mark Jerrum for some useful insights. This work was done in part while Weiming Feng was visiting the Simons Institute for the Theory of Computing.

1 Introduction

Network reliability is one of the first problems studied in counting complexity. Indeed, $s-t$ reliability is listed as one of the first thirteen complete problems when Valiant [29] introduced the counting complexity class #**P**. The general setting is that given a (directed or undirected) graph G, each edge e of G fails independently with probability q_e. The problem of $s-t$ reliability is then asking the probability that in the remaining graph, the source vertex s can reach the sink t. There are also other variants, where one may ask the probability of various kinds of connectivity properties of the remaining graph. These problems have been extensively studied, and apparently most variants are #**P**-complete [6, 19, 8, 28, 7, 11].

While the exact complexity of reliability problems is quite well understood, their approximation complexity is not. Indeed, the approximation complexity of the first studied $s-t$ reliability is still open in either directed or undirected graphs. One main exception is the *all-terminal* version (where one is interested in the remaining graph being connected or disconnected). A famous result by Karger [21] gives the first fully polynomial-time randomized approximation scheme (FPRAS) for all-terminal *unreliability*, while about two decades later, Guo and Jerrum [16] give the first FPRAS for all-terminal *reliability*. The latter algorithm is under the partial rejection sampling framework [17], and the Markov chain Monte Carlo


(MCMC) method is also shown to be efficient shortly after [3, 12]. See [18, 22, 9], [15], and [4, 10] for more recent results and improved running times along the three lines above for the all-terminal version respectively.

The success of these methods implies that the solution space of all-terminal reliability is well-connected via local moves. However, this is not the case for the two-terminal version (namely the $s-t$ version). Instead, the natural local-move Markov chain for $s-t$ reliability is torpidly mixing. Here the solution space consists of all spanning subgraphs (namely a subset of edges) in which s can reach t. Consider a (directed or undirected) graph composed of two paths of equal length connecting s and t. Suppose we start from one path and leave the other path empty. Then before the other path is all included in the current state, we cannot remove any edge of the initial path. This creates an exponential bottleneck for local-move Markov chains, and it suggests that a different approach is required.

In this paper, we give an FPRAS for the $s-t$ reliability in directed acyclic graphs. Note that the exact version of this problem is #**P**-complete [28, Sec 3], even restricted to planar DAGs where the vertex degrees are at most 3 [27, Theorem 3]. Our result positively resolves an open problem by Zenklusen and Laumanns [30]. Without loss of generality, in the theorem below we assume that any vertex other than s has at least one incoming edge, and thus $|E| \geq |V| - 1$ for the input $G = (V, E)$.

▶ **Theorem 1.** *Let $G = (V, E)$ be a directed acyclic graph (DAG), failure probabilities $\mathbf{q} = (q_e)_{e \in E} \in [0, 1]^E$, two vertices $s, t \in V$, and $\varepsilon > 0$. There is a randomized algorithm that takes $(G, \mathbf{q}, s, t, \varepsilon)$ as inputs and outputs a $(1 \pm \varepsilon)$-approximation to the $s-t$ reliability with probability at least $3/4$ in time $\widetilde{O}(n^6 m^4 \max\{m^4, \varepsilon^{-4}\})$ where $n = |V|$, $m = |E|$, and $\widetilde{O}$ hides polylog(n/ϵ) factors.*

The running time of Theorem 1 is $\widetilde{O}(n^6 m^8)$ when $\varepsilon > 1/m$, and is $\widetilde{O}(n^6 m^4/\varepsilon^4)$ when $\varepsilon < 1/m$. The reason behind this running time is that our algorithm always outputs at least a $(1 \pm 1/m)$-approximation. Thus, when $\varepsilon > 1/m$, it does not matter what ε actually is for the running time. This high level of precision is required for the correctness of the algorithm.

As hinted earlier, our method is a significant departure from the techniques for the all-terminal versions. Indeed, a classical result by Karp and Luby [23, 25] has shown how to efficiently estimate the size of a union of sets. A direct application of this method to $s-t$ reliability is efficient only for certain special cases [24, 30]. Our main observation is to use the Karp-Luby method as a subroutine in dynamic programming using the structure of DAGs. Let $s = v_1, \ldots, v_n = t$ be a topological ordering of the DAG G. (Note that we can ignore vertices before s and after t.) Let R_u be the $u-t$ reliability so that our goal is to estimate R_s. We inductively estimate R_{v_i} from $i = n$ to $i = 1$. For each vertex u, our algorithm maintains an estimator of R_u and a set S_u of samples of subgraphs in which u can reach t. In the induction step, we use the Karp-Luby method to generate the next estimator, and use a self-reduction similar to the Jerrum-Valiant-Vazirani sampling to counting reduction [20] to generate samples. Both tasks can be done efficiently using only what have been computed for previous vertices. Moreover, a naive implementation of this outline would require exponentially many samples. To avoid this, we reuse generated samples and carefully analyze the impact of doing so on the overall error bound. A more detailed overview is given in Section 1.1.

Our technique is inspired by an FPRAS for the number of accepting strings of non-deterministic finite automata (#NFA), found by Arenas, Croquevielle, Jayaram, and Riveros [5], which in turn used some techniques from a quasi-polynomial-time algorithm for sampling words from context-free languages by Gore, Jerrum, Kannan, Sweedyk, and

Mahaney [14]. Their #NFA algorithm runs in time $O\left(\left(\frac{n\ell}{\varepsilon}\right)^{17}\right)$,[1] where n is the number of states and ℓ is the string length. More recently, Meel, Chakraborty, and Mathur claim an improved running algorithm which runs in time $\widetilde{O}(\frac{n^4\ell^{11}}{\varepsilon^4})$ [26, Theorem 3]. These algorithms first normalize the NFA into a particular layered structure. Applying similar methods on the $s-t$ reliability problem can simplify the analysis, but would greatly slow down the algorithm. In contrast, our method works directly on the DAG. This makes our estimation and sampling subroutines interlock in an intricate way. To analyze the algorithm, we have to carefully separate out various sources of randomness. This leads to a considerably more sophisticated analysis, with a reward of a much better (albeit still high) running time.

Independently and simultaneously, Amarilli, van Bremen, and Meel [2] also found an FPRAS for $s-t$ reliability in DAGs. Their method is to reduce the problem to #NFA via a sequence of reductions, and then invoke the algorithm in [5]. Indeed, as Marcelo Arenas subsequently pointed out to us, counting the number of subgraphs of a DAG in which s can reach t belongs to a complexity class **SpanL** [1], where #NFA is **SpanL**-complete under polynomial-time parsimonious reductions. In particular, every problem in **SpanL** admits an FPRAS because #NFA admits one [5], which implies that $s-t$ reliability in DAGs admits an FPRAS if $q_e = 1/2$ for all edges. The method of [2] reduces a reliability instance of n vertices and m edges, where $q_e = 1/2$ for all edges, to estimating length m accepting strings of an NFA with $O(m^2)$ states.[2] As a consequence, their algorithm (even using the faster algorithm for #NFA [26]) has a running time of $O\left(m^{19}\varepsilon^{-4}\right)$. When $q_e \neq 1/2$, their reduction needs to expand the instance further to reduce to the $q_e = 1/2$ case, slowing down the algorithm even more. In contrast, our algorithm deals with all possible probabilities $0 \leq q_e < 1$ in a unified way. In any case, an algorithm via reductions is much slower than the direct algorithm in Theorem 1.

As both all-terminal reliability and unreliability in undirected graphs have FPRASes [21, 16], one may wonder if FPRAS exists for $s-t$ *unreliability* in DAGs. Here $s-t$ unreliability is the probability that s cannot reach t in the random subgraph. In contrast to Theorem 1, we show that this problem is #**BIS**-hard, where #**BIS** is the problem of counting independent sets in bipartite graphs, whose approximation complexity is still open. This is a central problem in the complexity of approximate counting [13], and is conjectured to have no FPRAS.

▶ **Theorem 2.** *There is no FPRAS to estimate* $s-t$ unreliability *in DAGs unless there is an FPRAS for* #**BIS**. *This is still true even if all edges fail with the same probability.*

Theorem 2 is proved in Section 5. The hardness of $s-t$ unreliability does not contradict Theorem 1. This is because a good relative approximation of x does not necessarily provide a good approximation of $1-x$, especially when x is close to 1.

The complexity of estimating $s-t$ reliability in general directed or undirected graphs remains open. We hope that our work sheds some new light on these decades old problems. Another open problem is to reduce the running time of Theorem 1, as currently the exponent of the polynomial is still high.

[1] The running time of the algorithm in [5] is not explicitly given. This bound is obtained by going through their proof.

[2] In fact, [2] first reduces the reliability instance to an nOBDD (non-deterministic ordered binary decision diagram) of size $O(m)$, which can be further reduced to an NFA of size $O(m^2)$. As they are working with a more general context, no explicit reduction is given for the $s-t$ reliability problem in DAGs. We provide a direct (and essentially the same) reduction in the full version of this paper.

1.1 Algorithm overview

Here we give an overview of our algorithm. For simplicity, we assume that $q_e = 1/2$ for all edges. The general case of $0 \le q_e < 1$ can be solved with very small tweaks.

Let $s = v_1 \prec \cdots \prec v_n = t$ be a topological ordering of the DAG G. (Note that we can ignore vertices before s and after t.) Let R_u be the $u - t$ reliability so that our goal is to estimate R_s. Note that R_{v_i} depends only on the subgraph induced by the vertex set $\{v_i, v_{i+1}, \ldots, v_n\}$. We denote this subgraph by $G_{v_i} = (V_{v_i}, E_{v_i})$. As we assumed $q_e = 1/2$, estimating R_{v_i} is equivalent to estimating the number of (spanning) subgraphs of G_{v_i} in which v_i can reach t. Denote the latter quantity by Z_{v_i} so that $R_{v_i} = Z_{v_i}/2^{|E_{v_i}|}$. For each vertex v_i, our algorithm maintains an estimator and a multi-set of random samples:

- $\widetilde{Z}_{v_i}$:[3] an estimator that approximates Z_{v_i} with high probability;
- S_{v_i}: a multi-set of random subgraphs, where each $H \in S_{v_i}$ is an approximate sample from π_{v_i} and π_{v_i} is the uniform distribution over all spanning subgraphs of G_{v_i} in which v_i can reach t.

Our algorithm computes $\widetilde{Z}_{v_i}$ and S_{v_i} for i from n to 1 by dynamic programming. The base case $v_n = t$ is trivial. In the induction step, suppose the vertex v_i has d out-neighbors $u_1, u_2, \ldots, u_d$. Note that each u_j for $j \in [d]$ comes after v_i in the topological ordering. Let us further assume $v_i \prec u_1 \prec, \ldots, \prec u_d$. For any subgraph H of G_{v_i}, if v_i can reach t in H, then there exists a neighbor u_j such that v_i can reach u_j and u_j can reach t in H. We can write Z_{v_i} as the size of the union $\Omega := \cup_{j=1}^{d} \Omega^{(j)}$, where $\Omega^{(j)}$ contains all subgraph of G_{v_i} where v_i can reach t through the neighbor u_j. Note that it is straightforward to estimate the size of $\Omega^{(j)}$ given $\widetilde{Z}_{u_j}$, and to generate uniform random subgraphs from $\Omega^{(j)}$ using samples in S_{u_j}. Given the size and samples from $\Omega^{(j)}$, a classical algorithm by Karp and Luby [23, 25] can be applied to efficiently estimate the size of the union of sets, namely to compute $\widetilde{Z}_{v_i}$.

The more complicated task is to generate the samples of S_{v_i}. We use a sampling to counting self-reduction á la Jerrum-Valiant-Vazirani [20]. To generate a sample H, we go through each edge e in G_{v_i}, deciding if e is added into H according to its conditional marginal probability. The first edge to consider is (v_i, u_1). Its marginal probability depends on how many subgraphs in Ω contain it. This quantity is the same as the number of subgraphs in which Λ can reach t, where Λ is a new vertex after contracting v_i and u_1. To estimate this number, denoted by Z_Λ, we use the Karp-Luby algorithm again. Note that the Karp-Luby algorithm requires estimates and samples from all out-neighbours of Λ, which have been computed already as these vertices are larger in the topological ordering than v_i. Having estimated Z_Λ, we estimate the marginal probability of (v_i, u_1) and decide if it is included in H. The process then continues to consider the next edge. In each step, we contract all vertices that can be reached from v_i into Λ, and keep estimating new Z_Λ using the Karp-Luby algorithm to compute the conditional marginal probabilities, until all edges are considered to generate H.

A naive implementation of the process above is to generate fresh samples every time S_u is used, which would lead to an exponential number of samples required. To maintain efficiency, the key property of our algorithm is to *reuse* random samples. For any vertex u, the algorithm generates the multi-set of samples S_u only once. Whenever the algorithm needs to use random samples for u, it always picks one sample from the same set S_u. Hence, one sample may be used multiple times during the whole algorithm. Reusing samples

[3] Our algorithm actually directly maintains an estimate $\widetilde{R}_{v_i}$ to the reliability R_{v_i}. In this overview, we use $\widetilde{Z}_{v_i}$ instead for simplicity.

introduces very complicated correlation among all $(\widetilde{Z}_u, S_u)$'s, which is a challenge to proving the correctness of the algorithm. Essentially, our analysis shows that as long as the estimates ($\widetilde{Z}_u$'s) and the samples are accurate enough, the overall error can be controlled. Accurate estimates of Z_u's allow us to bound the total variation (TV) distance between our samples and perfect samples. In turn, the small TV distance implies that there is a coupling between them, which helps us bound the errors of the estimates. This way, we circumvent the effect of correlation on the analysis and achieve the desired overall error bound.

For the overall running time, there are two tasks for each vertex, namely the estimation step (based on Karp-Luby) and the sample generation step. As we also need to perform estimation steps as subroutines when generating samples, the running time is dominated by the time for the sampling step. Let ℓ be the number of samples required per vertex, so that the total number of samples generated is $n\ell$. Roughly speaking, because we can only use the union bound due to various correlation, and because errors accumulate throughout dynamic programming, we set the error to be $\delta := n^{-1}\min\{m^{-1}, \varepsilon\}$ for the estimation step. Each estimation has two stages, first getting a constant approximation and then using the crude estimation to tune the parameters and obtain a $1 \pm \delta$ approximation. This succeeds with constant probability using $O(n\delta^{-2})$ samples. The estimator itself requires $O(m)$ time to compute, and thus the total running time for each estimation step with constant success probability is $\widetilde{O}(mn\delta^{-2})$. However, as samples are reused, we need to apply a union bound to control the error over all possible values of the samples, which are exponentially many. This requires us to amplify the success probability of the estimation step to $\exp(-\Omega(m))$, which means we need to repeat the algorithm $O(m)$ times and take the median. Each estimation step thus takes $\widetilde{O}(m^2n\delta^{-2})$ time and $O(mn\delta^{-2})$ samples. Instead of maintaining $O(mn\delta^{-2})$ samples for every vertex, the aforementioned two-stage estimation allows us to spread the cost and maintain $\ell := \frac{O(mn\delta^{-2})}{n} = O(n^2 m \max\{m^2, \varepsilon^{-2}\})$ samples per vertex, which we show suffice with high probability. As we may do up to $O(m)$ estimation steps during each sampling step, the overall running time is then bounded by $\widetilde{O}(n\ell \cdot m \cdot m^2 n\delta^{-2}) = \widetilde{O}\big(n^6 m^4 \max\{m^4, \varepsilon^{-4}\}\big)$.

2 Preliminaries

2.1 Problem definitions

Let $G = (V, E)$ be a directed acyclic graph (DAG). Each directed edge (or arc) $e = (u, v)$ is associated with a failure probability $0 \le q_e < 1$. (Any edge with $q_e = 1$ can be simply removed.) We also assume graph G is simple because parallel edges with failure probabilities $q_{e_1}, q_{e_2}, \ldots, q_{e_k}$ can be replace with one edge with failure probability $q_e = \prod_{i=1}^{m} q_{e_i}$. Given two vertices $s, t \in V$, the $s - t$ reliability problem asks the probability that s can reach t if each edge $e \in E$ fails (namely gets removed) independently with probability q_e. Formally, let $\boldsymbol{q} = (q_e)_{e \in E}$. The $s - t$ reliability problem is to compute

$$R_{G,\boldsymbol{q}}(s, t) := \Pr_{\mathcal{G}}[\,\text{there is a path from } s \text{ to } t \text{ in } \mathcal{G}\,], \tag{1}$$

where $\mathcal{G} = (V, \mathcal{E})$ is a random subgraph of $G = (V, E)$ such that each $e \in E$ is added independently to $\mathcal{E}$ with probability $1 - q_e$.

Closely connected to estimating $s - t$ reliability is a sampling problem, which we call the $s - t$ *subgraph sampling* problem. Here the goal is to sample a random (spanning) subgraph G' conditional on that there is at least one path from s to t in G'. Formally, let $\Omega_{G,s,t}$ be

the set of all subgraphs $H = (V, E_H)$ of G such that $E_H \subseteq E$ and s can reach t in H. The algorithm needs to draw samples from the distribution $\pi_{G,s,t,\boldsymbol{q}}$ whose support is $\Omega_{G,s,t}$ and for any $H = (V, E_H) \in \Omega_{G,s,t}$,

$$\pi_{G,s,t,\boldsymbol{q}}(H) = \frac{1}{R_{G,\boldsymbol{q}}(s,t)} \cdot \prod_{e \in E_H} (1 - q_e) \prod_{f \in E \setminus E_H} q_f. \tag{2}$$

2.2 More notations

Fix a DAG $G = (V, E)$. For any two vertices u and v, we use $u \rightsquigarrow_G v$ to denote that u can reach v in the graph G and use $u \not\rightsquigarrow_G v$ to denote that u cannot reach v in the graph G. It always holds that $u \rightsquigarrow_G u$. Fix two vertices s and t, where s is the source and t is the sink. The failure probabilities $\mathbf{q}$, s, and t will be the same throughout the paper, and thus we omit them from the subscripts. For any vertex $u \in V$, we use $G_u = G[V_u]$ to denote the subgraph of G induced by the vertex set

$$V_u := \{w \in V \mid u \rightsquigarrow_G w \wedge w \rightsquigarrow_G t\}. \tag{3}$$

Without loss of generality, we assume $G_s = G$. This means that all vertices except s have at least one in-neighbour, and thus $m \geq n - 1$. If $G_s \neq G$, then all vertices and edges in $G - G_s$ have no effect on $s - t$ reliability and we can simply ignore them. For the sampling problem, we can first solve it on the graph G_s and then independently add each edge e in $G - G_s$ with probability $1 - q_e$.

Our algorithm actually solves the $u - t$ reliability and $u - t$ subgraph sampling problems in G_u for all $u \in V$. Let $G_u = (V_u, E_u)$. For any subgraph $H = (V_u, E_H)$ of G_u, define the weight function

$$w_u(H) := \begin{cases} \prod_{e \in E_H} (1 - q_e) \prod_{f \in E_u \setminus E_H} q_f & \text{if } u \rightsquigarrow_H t; \\ 0 & \text{if } u \not\rightsquigarrow_H t. \end{cases} \tag{4}$$

Define the distribution π_u by

$$\pi_u(H) := \frac{w_u(H)}{R_u},$$

where the partition function (the normalizing factor)

$$R_u := \sum_{H\text{: subgraph of } G_u} w_u(H)$$

is exactly the $u - t$ reliability in the graph G_u. Finally, let

$$\Omega_u := \{H = (V_u, E_H) \mid E_H \subseteq E_u \wedge u \rightsquigarrow_H t\} \tag{5}$$

be the support of π_u. Also note that R_s and π_s are the probability $R_{G,\boldsymbol{q}}(s,t)$ and the distribution $\pi_{G,s,t,\boldsymbol{q}}$ defined in (1) and (2), respectively. The set Ω_s is the set $\Omega_{G,s,t}$ in Section 2.1.

2.3 The total variation distance and coupling

Let μ and ν be two discrete distributions over Ω. The *total variation distance* between μ and ν is defined by

$$d_{\mathrm{TV}}(\mu, \nu) := \frac{1}{2} \sum_{x \in \Omega} |\mu(x) - \nu(x)|.$$

If $X \sim \mu$ and $Y \sim \nu$ are two random variables, we also abuse the notation and write $d_{\mathrm{TV}}(X, Y) := d_{\mathrm{TV}}(\mu, \nu)$.

A *coupling* between μ and ν is a joint distribution (X, Y) such that $X \sim \mu$ and $Y \sim \nu$. The following coupling inequality is well-known.

▶ **Lemma 3.** *For any coupling $\mathcal{C}$ between two random variables $X \sim \mu$ and $Y \sim \nu$, it holds that*

$$\Pr_{\mathcal{C}}[X \neq Y] \geq d_{TV}(\mu, \nu).$$

Moreover, there exists an optimal coupling that achieves equality.

3 The algorithm

In this section we give our algorithm. We also give intuitions behind various design choices, and give some basic properties of the algorithm along the way. We give a sketch of the analysis in Section 4 and the full analysis is in the full version of this paper.

3.1 The framework of the algorithm

As $G = G_s$ is a DAG, there is a topological ordering of all vertices. There may exist many topological orderings. We pick an arbitrary one, say, $v_1, \ldots, v_n$. It must hold that $v_1 = s$ and $v_n = t$. The topological ordering guarantees that if (u, v) is an edge, u must come before v in the ordering, denoted $u \prec v$.

On a high level, our algorithm is to inductively compute an estimator $\widetilde{R}_u$ of R_u, from $u = v_n$ to $u = v_1$. In addition to $\widetilde{R}_u$, we also maintain a multi-set S_u of samples from π_u over Ω_u. For any vertex $u \in V$, let $\Gamma_{\mathrm{out}}(u) := \{w \mid (u, w) \in E\}$ denote the set of out-neighbours of u. Let

$$\ell := (60n + 150m)(400n + 500\left\lceil 10^4 n^2 \max\{m^2, \varepsilon^{-2}\}\right\rceil) = O((n+m)n^2 \max\{m^2, \epsilon^{-2}\}) \tag{6}$$

be the size of S_v for all $v \in V_u$, where n is the number of vertices in G and m is the number of edges in G. The choice of this parameter is explained in the last paragraph of Section 1.1. Our algorithm is outlined in Algorithm 1. Note that G_t is an isolated vertex. For consistency, we let S_t contain ℓ copies of $\emptyset$.

The base case of $v_n = t$ is trivial. The subroutine $\mathsf{Sample}(\cdot)$ uses $(\widetilde{R}_{v_i}, S_{v_i})$ for all $i > k$ and $\widetilde{R}_{v_k}$ to generate samples in S_{v_k}. The subroutine $\mathsf{ApproxCount}(V, E, u, (\widetilde{R}_w, S_w)_{\Gamma_{\mathrm{out}}(u)})$ takes a graph $G = (V, E)$, a vertex u, and $(\widetilde{R}_w, S_w)$ for all $w \in \Gamma_{\mathrm{out}}(u)$ as the input, and it outputs an approximation of the $u - t$ reliability in the graph G. We describe Sample in Section 3.2 and (a slightly more general version of) $\mathsf{ApproxCount}$ in Section 3.3.

3.2 Generate samples

Let $u = v_k$ where $k < n$. Recall that $G_u = (V_u, E_u)$ is the graph defined in (3). The sampling algorithm aims to output a random spanning subgraph $H = (V_u, \mathcal{E})$ from the distribution π_u. The algorithm is based on the sampling-to-counting reduction in [20]. It scans each edge e in E_u and decides whether to put e into the set $\mathcal{E}$ or not. The algorithm maintains two edge sets:

- $E_1 \subseteq E_u$: the set of edges that have been scanned by the algorithm;
- $\mathcal{E} \subseteq E_1$: the current set of edges sampled by the algorithm.

Algorithm 1 An FPRAS for $s-t$ reliabilities in DAGs.

Input: a DAG $G=(V,E)$, a vector $\boldsymbol{q}=(q_e)_{e\in E}$, the source s, the sink t, and an error bound $0<\varepsilon<1$, where $G=G_s$ and $V=\{v_1,v_2,\ldots,v_n\}$ is topologically ordered with $v_1=s$ and $v_n=t$

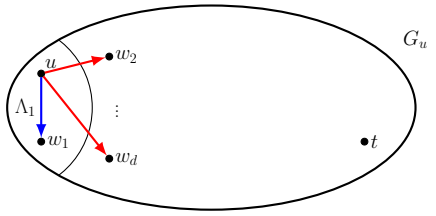
Output: an estimator $\widetilde{R}_s$ of R_s

let $\widetilde{R}_t=1$ and S_t be a multi set of ℓ $\emptyset$'s;
for k *from* $n-1$ *to* 1 **do**
  $\widetilde{R}_{v_k} \leftarrow \mathsf{ApproxCount}\Big(V_{v_k}, E_{v_k}, v_k, (\widetilde{R}_w, S_w)_{w\in\Gamma_{out}(v_k)}\Big)$;
  $S_{v_k} \leftarrow \emptyset$;
  for j *from* 1 *to* ℓ **do**
    $S_{v_k} \leftarrow S_{v_k} \cup \mathsf{Sample}\Big(v_k, (\widetilde{R}_w, S_w)_{w\in\{v_{k+1},v_{k+2},\ldots,v_n\}}, \widetilde{R}_{v_k}\Big)$;
return $\widetilde{R}_s$.

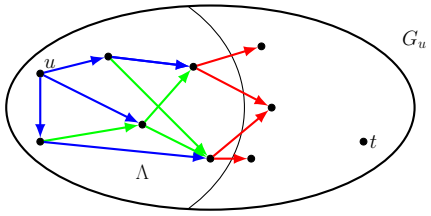
Also, let $E_2 := E_u \setminus E_1$ be the set of edges that have not been scanned yet by the algorithm. Given any $\mathcal{E}$, we can uniquely define the following subset of vertices

$$\Lambda = \Lambda_{\mathcal{E}} := \mathsf{rch}(u, V_u, \mathcal{E}) = \{w \in V_u \mid u \text{ can reach } w \text{ through edges in } \mathcal{E}\}. \tag{7}$$

In other words, let $G'=(V_u,\mathcal{E})$ and Λ is the set of vertices that u can reach in G'. Note that $u\in\Lambda$ for any $\mathcal{E}$. We will keep updating Λ as $\mathcal{E}$ expands. When calculating the marginal probability of the next edge, the path to t can start from any vertex in Λ. Thus we need to estimate the reliability from Λ to t. Instead of having a single source, as called in Algorithm 1, we use a slightly more general version of ApproxCount, described in Section 3.3, to allow a set Λ of sources. This subroutine ApproxCount takes input $(V,E,\Lambda,(\widetilde{R}_w,S_w)_{w\in\partial\Lambda})$ and approximates the $\Lambda-t$ reliability in (V,E), which is the probability that there exists at least one vertex in Λ being able to reach t if each edge $e\in E$ fails independently with probability q_e. An equivalent way of seeing it is to contract all vertices in Λ into a single vertex u first, and then calculate the $u-t$ reliability in the resulting graph. Sample is described in Algorithm 2, and some illustration is given in Figure 1.



(a) At the start, we consider the first edge (u,w_1). To compute its marginal, we estimate two reliability, where the source is either $\Lambda_1=\{u,w_1\}$ (shown in picture) or just u, respectively.



(b) As Algorithm 2 progresses, there are chosen edges $\mathcal{E}$ (blue), not chosen edges $E_1\setminus\mathcal{E}$ (green), and the boundary edges (red). The set Λ contains vertices reachable from u using only $\mathcal{E}$.

Figure 1 An illustration of sampling from π_u.

▶ Remark 4 (Crash of Sample). The subroutine Sample (Algorithm 2) may crash in following cases: (1) in Algorithm 2, $\partial\Lambda=\emptyset$; (2) in Algorithm 2, $q_e c_0+(1-q_e)c_1=0$; (3) in Algorithm 2, $\frac{w_u(H)}{4p_0 p}>1$; (4) in Algorithm 2, $F=0$. If it crashes, we stop Algorithm 1 immediately and output $\widetilde{R}_s=0$.

Algorithm 2 $\mathsf{Sample}\Big(u, (\widetilde{R}_w, S_w)_{w\in\{v_{k+1},v_{k+2},\ldots,v_n\}}, \widetilde{R}_{v_k}\Big)$.

Input: a vertex $u = v_k$, all $(\widetilde{R}_w, S_w)$ for $w \in \{v_{k+1}, v_{k+2}, \ldots, v_n\}$, and $\widetilde{R}_{v_k} = \widetilde{R}_u$
Output: a random subgraph $H = (V_u, \mathcal{E})$

$T \leftarrow \lceil 1000 \log \frac{n}{\varepsilon} \rceil$ and $F \leftarrow 0$;
$p_0 \leftarrow \widetilde{R}_u$;
repeat
  let $p \leftarrow 1$;
  let $E_1 \leftarrow \emptyset$, $E_2 \leftarrow E_u \setminus E_1$ and $\Lambda = \{u\}$;
  while $t \notin \Lambda$ **do**
    let $\partial\Lambda \leftarrow \{w \notin \Lambda \mid \exists w' \in \Lambda \text{ s.t. } (w', w) \in E_2\}$; let $w^* \in \partial\Lambda$ be the smallest vertex in the topological ordering; pick an arbitrary edge $e = (w', w^*) \in E_2$ such that $w' \in \Lambda$;
    let $\Lambda_1 \leftarrow \mathsf{rch}(u, V_u, \mathcal{E} \cup \{e\})$;
    $E_1 \leftarrow E_1 \cup \{e\}$ and $E_2 \leftarrow E_2 \setminus \{e\}$;
    $\partial\Lambda_1 \leftarrow \{w \notin \Lambda_1 \mid \exists w' \in \Lambda_1 \text{ s.t. } (w', w) \in E_2\}$ and $\partial\Lambda \leftarrow \{w \notin \Lambda \mid \exists w' \in \Lambda \text{ s.t. } (w', w) \in E_2\}$;
    $c_0 \leftarrow \mathsf{ApproxCount}(V_u, E_2, \Lambda, (\widetilde{R}_w, S_w)_{w\in\partial\Lambda})$;
    $c_1 \leftarrow \mathsf{ApproxCount}(V_u, E_2, \Lambda_1, (\widetilde{R}_w, S_w)_{w\in\partial\Lambda_1})$;
    let $c \leftarrow 1$ with probability $\frac{(1-q_e)c_1}{q_e c_0 + (1-q_e)c_1}$; otherwise $c \leftarrow 0$;
    if $c = 1$, **then** let $\mathcal{E} \leftarrow \mathcal{E} \cup \{e\}$, $\Lambda \leftarrow \Lambda_1$, $p \leftarrow p\frac{(1-q_e)c_1}{q_e c_0 + (1-q_e)c_1}$;
    if $c = 0$, **then** let $p \leftarrow p\Big(1 - \frac{(1-q_e)c_1}{q_e c_0 + (1-q_e)c_1}\Big)$;
  for *all edges* $e \in E_2$ **do**
    let $c \leftarrow 1$ with probability $1 - q_e$; otherwise $c \leftarrow 0$;
    if $c = 1$, **then** let $\mathcal{E} \leftarrow \mathcal{E} \cup \{e\}$ and $p \leftarrow p(1 - q_e)$;
    if $c = 0$, **then** let $p \leftarrow pq_e$;
  let $F \leftarrow 1$ with probability $\frac{w_u(H)}{4pp_0}$, where $H = (V_u, \mathcal{E})$;
  $T \leftarrow T - 1$;
until $T < 0$ *or* $F = 1$;
if $F = 1$ then **return** $H = (V_u, \mathcal{E})$; **else return** $\perp$;

The algorithm needs c_0 and c_1 in order to compute the marginal probability of e in Algorithm 2. The quantity c_0 is an estimate to the reliability conditional on e not selected and all choices made so far (namely $E_H \cap E_1 = \mathcal{E}$). Similarly, c_1 is an estimate to the reliability conditional on e selected and $E_H \cap E_1 = \mathcal{E}$. Thus, if $\mathsf{ApproxCount}$ returns exact values of c_0 and c_1, then

$$\Pr_{H=(V_u,E_H)\sim\pi_u}\left[e \in \mathcal{E}_H \mid E_H \cap E_1 = \mathcal{E}\right] = \frac{(1-q_e)c_1}{q_e c_0 + (1-q_e)c_1}.$$

However, $\mathsf{ApproxCount}$ can only approximate the reliabilities c_0 and c_1. To handle the error from $\mathsf{ApproxCount}$, our algorithm maintains a number p, which is the probability of selecting the edges in $\mathcal{E}$ and not selecting those in $E_1 \setminus \mathcal{E}$. By the time we reach Algorithm 2, p becomes the probability that H is generated by the algorithm. Then the algorithm uses a filter (with filter probability $\frac{w_u(H)}{4p_0 p}$) to correct the distribution of H. Conditional on passing the filter, H is a perfect sample from π_u. The detailed analysis of the error is given in the full version of this paper.

Before we go to the ApproxCount algorithm, we state one important property of Algorithm 2.

▶ **Lemma 5.** *In Algorithm 2, the following property holds: at the beginning of every while-loop, for any $w \in \partial\Lambda$, $E_1 \cap E_w = \emptyset$, where E_w is the edge set of the graph $G_w = (V_w, E_w)$ defined in* (3)*.*

The above lemma holds because we always pick the smallest vertex in the topological ordering at the beginning of the while-loop. The formal proof is in the full version of this paper.

▶ **Corollary 6.** *In Algorithm 2, in every while-loop, $\Lambda_1 = \Lambda \cup \{w^*\}$.*

Proof. Clearly $\Lambda \cup \{w^*\} \subseteq \Lambda_1$. Suppose there exists $w_0 \neq w^*$ such that $w_0 \in \Lambda_1 \setminus \Lambda$. Then $w_0 \in V_u$ can be reached from w^* through edges in $\mathcal{E}$. The vertex w_0 must be in G_{w^*}, which implies $\mathcal{E} \cap E_{w^*} \neq \emptyset$, and thus $E_1 \cap E_{w^*} \neq \emptyset$. A contradiction to Lemma 5. ◀

3.3 Approximate counting

Our ApproxCount subroutine is used in both Algorithm 1 and Algorithm 2. To suit Algorithm 2, ApproxCount takes input $(V, E, \Lambda, (\widetilde{R}_w, S_w)_{w\in\partial\Lambda})$ and approximates the $\Lambda - t$ reliability $R_\Lambda := \Pr_{\mathcal{G}}[\Lambda \rightsquigarrow_{\mathcal{G}} t]$, where $\mathcal{G}$ is a random spanning subgraph of G obtained by removing each edge e independently with probability q_e, and $\Lambda \rightsquigarrow_{\mathcal{G}} t$ denotes the event that $\exists w \in \Lambda$ s.t. $w \rightsquigarrow_{\mathcal{G}} t$. When called by Algorithm 1, Λ is a single vertex, and when called by Algorithm 2, Λ is the set defined in (7). Moreover, the input satisfies:

- $G = (V, E)$ is a DAG containing the sink t and each edge has a failure probability q_e (for simplicity, we do not write t and all q_e explicitly in the input as they do not change throughout Algorithm 1 and Algorithm 2);
- $\Lambda \subseteq V$ is a subset of vertices that act as sources and $\partial\Lambda := \{w \in V \setminus \Lambda \mid \exists w' \in \Lambda \text{ s.t. } (w', w) \in E\}$;
- for any $w \in \partial\Lambda$, $\widetilde{R}_w$ is an approximation of the $w - t$ reliability in G_w and S_w is a set of ℓ approximate random samples from the distribution π_w, where G_w is defined in (3).

All points above hold when Algorithm 3 is evoked by Algorithm 1 or Algorithm 2. The first two points are easy to verify and the last one is verified in the full version of this paper.

The algorithm first rules out the following two trivial cases:

- if $t \in \Lambda$, the algorithm returns 1;
- if Λ cannot reach t in graph G, the algorithm returns 0.[4]

As we are dealing with the more general set-to-vertex reliability, we need some more definitions. Define

$$\Omega_\Lambda := \{H = (V, E_H) \mid E_H \subseteq E \wedge \Lambda \rightsquigarrow_H t\}. \tag{8}$$

For any subgraph $H = (V, E_H)$ of $G = (V, E)$, define the weight function

$$w_\Lambda(H) := \begin{cases} \prod_{e\in E_H}(1 - q_e)\prod_{f\in E\setminus E_H} q_f & \text{if } \Lambda \rightsquigarrow_H t; \\ 0 & \text{if } \Lambda \not\rightsquigarrow_H t. \end{cases} \tag{9}$$

Define the distribution π_Λ, whose support is Ω_Λ, by

$$\pi_\Lambda(H) := \frac{w_\Lambda(H)}{R_\Lambda}, \quad \text{where } R_\Lambda := \sum_{H\in\Omega_\Lambda} w_\Lambda(H). \tag{10}$$

[4] Since all $q_e < 1$, $R_\Lambda > 0$ if and only if $\Lambda \rightsquigarrow_G t$.

Let $\partial\Lambda$ be listed as $\{u_1, \ldots, u_d\}$ for some $d \in [n]$. Note that $d \geq 1$ because $R_\Lambda > 0$ and $t \notin \Lambda$. To estimate R_Λ, we first write Ω_Λ in (8) as a union of d sets. For each $1 \leq i \leq d$, define

$$\Omega_\Lambda^{(i)} := \{H = (V, E_H) \mid (E_H \subseteq E) \wedge (\exists u \in \Lambda, \text{ s.t. } (u, u_i) \in E) \wedge (u_i \rightsquigarrow_H t)\}.$$

▶ **Lemma 7.** *If $t \notin \Lambda$, $\Omega_\Lambda = \cup_{i=1}^d \Omega_\Lambda^{(i)}$.*

Proof. We first show $\cup_{i=1}^d \Omega_\Lambda^{(i)} \subseteq \Omega_\Lambda$. Fix any $H \in \cup_{i=1}^d \Omega_\Lambda^{(i)}$, say $H \in \Omega_\Lambda^{(i^*)}$ ($i^* \in [d]$ may not be unique, in which case we pick an arbitrary one). Then Λ can reach t in H, because we can first move from Λ to u_{i^*} and then move from u_{i^*} to t. This implies $H \in \Omega_\Lambda$. We next show $\Omega_\Lambda \subseteq \cup_{i=1}^d \Omega_\Lambda^{(i)}$. Fix any $H \in \Omega_\Lambda$. There is a path from Λ to t in H. Say the path is $w_1, w_2, \ldots, w_p = t$. Then $w_1 \in \Lambda$ (hence $w_1 \neq t$) and $w_2 = u_{i^*}$ for some $i^* \in [d]$. Hence, H contains the edge (w_1, u_{i^*}) and $u_{i^*} \rightsquigarrow_H t$. This implies $H \in \cup_{i=1}^d \Omega_\Lambda^{(i)}$. ◀

Similar to (10), we define $\pi^{(i)}$,[5] whose support is $\Omega_\Lambda^{(i)}$, by

$$\pi_\Lambda^{(i)}(H) := \frac{w_\Lambda(H)}{R_\Lambda^{(i)}}, \quad \text{where } R_\Lambda^{(i)} := \sum_{H \in \Omega_\Lambda^{(i)}} w_\Lambda(H). \tag{11}$$

In order to perform the Karp-Luby style estimation, we need to be able to do the following three things:

1. compute the value $R_\Lambda^{(i)}$ for each $i \in [d]$;
2. draw samples from $\pi_\Lambda^{(i)}$ for each $i \in [d]$;
3. given any $i \in [d]$ and $H \in \Omega_\Lambda$, determine whether $H \in \Omega_\Lambda^{(i)}$.

Suppose we can do them for now.[6] Consider the following estimator Z_Λ:

1. draw an index $i \in [d]$ such that i is drawn with probability proportional to $R_\Lambda^{(i)}$;
2. draw an sample H from $\pi_\Lambda^{(i)}$;
3. let $Z_\Lambda \in \{0, 1\}$ indicate whether i is the smallest index $j \in [d]$ satisfying $H \in \Omega_\Lambda^{(j)}$.

It is straightforward to see that

$$\begin{aligned}
\mathbb{E}[Z_\Lambda] &= \sum_{i=1}^d \frac{R_\Lambda^{(i)}}{\sum_{j=1}^d R_\Lambda^{(j)}} \sum_{H \in \Omega_\Lambda^{(i)}} \pi_\Lambda^{(i)}(H) \cdot \mathbf{1}\left[H \in \Omega_\Lambda^{(i)} \wedge \left(\forall j < i, H \notin \Omega_\Lambda^{(j)}\right)\right] \\
&= \frac{\sum_{i=1}^d \sum_{H \in \Omega_\Lambda^{(i)}} w_\Lambda(H) \cdot \mathbf{1}\left[H \in \Omega_\Lambda^{(i)} \wedge \left(\forall j < i, H \notin \Omega_\Lambda^{(j)}\right)\right]}{\sum_{j=1}^d R_\Lambda^{(j)}} \\
\text{(by Lemma 7)} \quad &= \frac{R_\Lambda}{\sum_{j=1}^d R_\Lambda^{(j)}} \geq \frac{1}{d} \geq \frac{1}{n},
\end{aligned} \tag{12}$$

where the first inequality holds because each H belongs to at most d different sets. Since Z_Λ is a 0/1 random variable, $\mathrm{Var}(Z_\Lambda) \leq 1$. Hence, we can first estimate the expectation of Z_Λ by repeating the process above and taking the average, and then use $\mathbb{E}[Z_\Lambda] \sum_{j=1}^d R_\Lambda^{(j)}$ as the estimator $\widetilde{R}_\Lambda$ for R_Λ. However, in the input of ApproxCount, we only have estimates $\widetilde{R}_{u_i}$ and a limited set of samples S_{u_i} for each $u_i \in \partial\Lambda$. Our algorithm will need to handle these imperfections.

[5] One may notice that if $\Omega_\Lambda^{(i)} = \emptyset$, then $\pi_\Lambda^{(i)}$ is not well-defined. In the full version of this paper, we prove that $\Omega_\Lambda^{(i)} = \emptyset$ never happens.

[6] We will do (1) and (2) approximately rather than exactly. This will incur some error that will be controlled later.

The third step of implementing the estimator Z_Λ is straightforward by a BFS. For the first step, $R_\Lambda^{(i)}$ can be easily computed given R_{u_i}, and we will use the estimates $\widetilde{R}_{u_i}$ instead. For the second step, define

$$\delta_\Lambda(u_i) := \{(w, u_i) \in E \mid w \in \Lambda\}.$$

To sample from $\pi_\Lambda^{(i)}$, we shall sample at least one edge in $\delta_\Lambda(u_i)$, sample a subgraph H from π_{u_i}, and add all other edges independently. These are summarized by the next lemma.

▶ **Lemma 8.** *For any $i \in [d]$, it holds that $R_\Lambda^{(i)} = \left(1 - \prod_{u \in \delta_\Lambda(u_i)} q_{(u,u_i)}\right) R_{u_i}$ and a random sample $H' = (V, E_{H'}) \sim \pi_\Lambda^{(i)}$ can be generated by the following procedure:*

- *sample $H = (V_{u_i}, E_H) \sim \pi_{u_i}$;*
- *let $E_{H'} = E_H \cup D$, where $D \subseteq \delta_\Lambda(u_i)$ is a random subset with probability proportional to*

$$\mathbf{1}[D \neq \emptyset] \cdot \prod_{e \in \delta_\Lambda(u_i) \cap D} (1 - q_e) \prod_{f \in \delta_\Lambda(u_i) \setminus D} q_f; \tag{13}$$

- *add each $e \in E \setminus (E_{u_i} \cup \delta_\Lambda(u_i))$ into $E_{H'}$ independently with probability $1 - q_e$.*

All three steps above handle mutually exclusive edge sets and thus are mutually independent.

Proof. If each edge e in G is removed independently with probability q_e, we have a random spanning subgraph $\mathcal{G} = (V, \mathcal{E})$. By the definition of $R_\Lambda^{(i)}$,

$$\begin{aligned} R_\Lambda^{(i)} &= \Pr\left[\exists u \in \Lambda, \text{ s.t. } (u, u_i) \in \mathcal{E} \wedge u_i \rightsquigarrow_{\mathcal{G}} t\right] \\ &= \Pr\left[\exists u \in \Lambda, \text{ s.t. } (u, u_i) \in \mathcal{E}\right] \cdot \Pr\left[u_i \rightsquigarrow_{\mathcal{G}} t \mid \exists u \in \Lambda, \text{ s.t. } (u, u_i) \in \mathcal{E}\right]. \end{aligned}$$

It is easy to see $\Pr[\exists u \in \Lambda, \text{ s.t. } (u, u_i) \in \mathcal{E}] = 1 - \prod_{u \in \delta_\Lambda(u_i)} q_{(u,u_i)}$. For the second conditional probability, note that the event $u_i \rightsquigarrow_{\mathcal{G}} t$ depends only on the randomness of edges in graph G_{u_i}. In other words, for any edges $e \in E \setminus E_{u_i}$, whether or not e is removed has no effect on the $u_i - t$ reachability. Due to acyclicity, all edges in $\delta_\Lambda(u_i)$ are not in the graph G_{u_i}. We have

$$R_\Lambda^{(i)} = \left(1 - \prod_{u \in \delta_\Lambda(u_i)} q_{(u,u_i)}\right) R_{u_i}.$$

By definition (11), $\pi_\Lambda^{(i)}$ is the distribution of $\mathcal{G} = (V, \mathcal{E})$ conditional on $\exists u \in \Lambda, \text{ s.t. } (u, u_i) \in \mathcal{E}$ and $u_i \rightsquigarrow_{\mathcal{G}} t$. For any graph $H' = (V, E_{H'})$ satisfying $\exists u \in \Lambda, \text{ s.t. } (u, u_i) \in E_{H'}$ and $u_i \rightsquigarrow_{H'} t$, we have

$$\begin{aligned} \pi_\Lambda^{(i)}(H') &= \Pr\left[\mathcal{G} = H' \mid \exists u \in \Lambda, \text{ s.t. } (u, u_i) \in \mathcal{E} \wedge u_i \rightsquigarrow_{\mathcal{G}} t\right] \\ &= \frac{\Pr[\mathcal{G} = H']}{R_\Lambda^{(i)}} = \frac{\Pr[\mathcal{G} = H']}{(1 - \prod_{u \in \delta_\Lambda(u_i)} q_{(u,u_i)}) R_{u_i}} \\ &= \prod_{e \in E_{H'} : e \in E \setminus E_{u_i}} (1 - q_e) \prod_{f \notin E_{H'} : f \in E \setminus E_{u_i}} q_f \cdot \frac{w_{u_i}(H'[V_{u_i}])}{(1 - \prod_{u \in \delta_\Lambda(u_i)} q_{(u,u_i)}) R_{u_i}} \\ &= \pi_{u_i}(H'[V_{u_i}]) \cdot \frac{\prod_{e \in \delta_\Lambda(u_i) \cap E_{H'}} (1 - q_e) \prod_{f \in \delta_\Lambda(u_i) \setminus E_{H'}} q_f}{1 - \prod_{u \in \delta_\Lambda(u_i)} q_{(u,u_i)}} \\ &\quad \cdot \prod_{\substack{e \in E_{H'} : \\ e \in E \setminus (E_{u_i} \cup \delta_\Lambda(u_i))}} (1 - q_e) \prod_{\substack{f \notin E_{H'} : \\ f \in E \setminus (E_{u_i} \cup \delta_\Lambda(u_i))}} q_f. \end{aligned}$$

The probability above exactly matches the procedure in the lemma. ◀

The first sampling step in Lemma 8 can be done by directly using the samples from S_{u_i}. We still need to show that the second step in Lemma 8 can be done efficiently. The proof of Lemma 9 is in the full version of this paper.

▶ **Lemma 9.** *There is an algorithm such that given a set $S = \{1, 2, \dots, n\}$ and n numbers $0 \le q_1, q_2, \dots, q_n < 1$, it return a random non-empty subset $D \subseteq S$ with probability proportional to $\mathbf{1}[D \neq \emptyset] \prod_{i \in D}(1 - q_i) \prod_{j \in S \setminus D} q_j$ in time $O(n)$.*

Now, we are almost ready to describe ApproxCount (Algorithm 3). For any u_i, we have an approximate value $\widetilde{R}_{u_i}$ of R_{u_i} and we also have a set S_{u_i} of ℓ approximate samples from the distribution π_{u_i}. By Lemma 8 and Lemma 9, we can efficiently approximate $R^{(i)}_\Lambda$ and generate approximate samples from $\pi^{(i)}_\Lambda$. Hence, we can simulate the process described below Lemma 7 to estimate R_Λ.

However, to save the number of samples, there is a further complication. Our algorithm estimates the expectation of $\mathbb{E}[Z_\Lambda]$ in two rounds and then takes the median of estimators. Recall (6). We further divide ℓ by introducing the following parameters:

$$\begin{aligned} &\ell = B\ell_0, \quad B := 60n + 150m, \quad \ell_0 := \ell_1 + 500\ell_2, \\ &\ell_1 := 400n, \quad \ell_2 := \lceil 10^4 n^2 \max\{m^2, \epsilon^{-2}\} \rceil. \end{aligned} \tag{14}$$

Then we do the following.

- For any $u_i \in \partial\Lambda$, we divide all ℓ samples in S_{u_i} into B blocks, each containing ℓ_0 samples. Denote the B blocks by $S^{(1)}_{u_i}, S^{(2)}_{u_i}, \dots, S^{(B)}_{u_i}$. Each block here is used for one estimatior.
- For each $i \in [d]$ and $j \in [B]$, we further partition $S^{(j)}_{u_i}$ into two multi-sets $S^{(j,1)}_{u_i}$ and $S^{(j,2)}_{u_i}$, where $S^{(j,1)}_{u_i}$ has ℓ_1 samples and $S^{(j,2)}_{u_i}$ has $500\ell_2$ samples, used for the two rounds, respectively.
- For each $j \in [B]$, we do the following two round estimation:
  1. use samples in $(S^{(j,1)}_{u_i})_{i \in [d]}$ to obtain a *constant-error* estimation $\widehat{Z}^{(j)}_\Lambda$ of $\mathbb{E}[Z_\Lambda]$;
  2. use $\widehat{Z}^{(j)}_\Lambda$ and samples in $(S^{(j,2)}_{u_i})_{i \in [d]}$ to obtain a more accurate *estimation* $\widetilde{Z}^{(j)}_\Lambda$ of $\mathbb{E}[Z_\Lambda]$;
  3. let $Q^{(j)}_\Lambda \leftarrow \widetilde{Z}^{(j)}_\Lambda \sum_{i=1}^{d} \widetilde{R}^{(i)}_\Lambda$, where $\widetilde{R}^{(i)}_\Lambda := \big(1 - \prod_{u \in \delta_\Lambda(u_i)} q_{(u,u_i)}\big)\widetilde{R}_{u_i}$ for each $i \in [d]$.
- Return the median number $\widetilde{R}_\Lambda := \text{median}\left\{Q^{(1)}_\Lambda, Q^{(2)}_\Lambda, \dots, Q^{(B)}_\Lambda\right\}$.

A detailed description of ApproxCount is given in Algorithm 3. It uses a subroutine Estimate, which generates the Karp-Luby style estimator and is described in Algorithm 4.

Each time Algorithm 3 finishes, its input (V, E, Λ) and output $\widetilde{R}_\Lambda$ are stored in the memory. If Algorithm 3 is ever evoked again with the same (V, E, Λ), we simply return $\widetilde{R}_\Lambda$ from the memory.

Algorithm 3 first obtains the constant-error estimation $\widehat{Z}^{(j)}_\Lambda$ in Algorithm 3. Next, it puts $\widehat{Z}^{(j)}_\Lambda$ into the parameters and run the subroutine Estimate again to get a more accurate estimation $\widetilde{Z}^{(j)}_\Lambda$. The benefit of this two-round estimation is that we can save the number of samples maintained for each vertex. It costs only a small number of samples to get the crude estiamtion, but the crude estimation carries information of the ratio $\frac{\sum_{t \in [d]} R^{(t)}_\Lambda}{R_\Lambda}$, which allows us to fine tune the number of samples required per vertex for the good estimation. To be more specific, in the second call of the subroutine Estimate, the number of overall samples, namely the parameter T which depends on $\widehat{Z}^{(j)}_\Lambda$, can still be as large as $\Omega(\ell_2 n)$ in the worst case, and yet each $S^{(j,2)}_{u_i}$ has only $O(\ell_2)$ samples in the block. In the analysis, we will show that this many samples per vertex suffice with high probability and Algorithm 4 of Algorithm 4 (the failure case) is executed with low probability. The reason is that, roughly

Algorithm 3 $\mathsf{ApproxCount}\Big(V, E, \Lambda, (\widetilde{R}_w, S_w)_{w\in\partial\Lambda}\Big)$.

Input: a graph $G = (V, E)$, a subset $\Lambda \subseteq V$, all $(\widetilde{R}_w, S_w)$ for $w \in \partial\Lambda$, where $\partial\Lambda = \{w \in V \setminus \Lambda \mid \exists w' \in \Lambda \text{ s.t. } (w', w) \in E\}$;
Output: an estimator $\widetilde{R}_\Lambda$ of R_Λ
if $t \in \Lambda$, **then return** 0; **if** Λ cannot reach t in G, **then return** 1;
for $u_i \in \partial\Lambda$ **do**
  $\widetilde{R}^{(i)}_\Lambda \leftarrow \big(1 - \prod_{u\in\delta_\Lambda(u_i)} q_{(u,u_i)}\big)\widetilde{R}_{u_i}$;
  partition S_{u_i} (arbitrarily) into B multi-sets, denoted by $S^{(j)}_{u_i}$ for $j \in [B]$, where $B = 60n + 150m$ and each $S^{(j)}_{u_i}$ has $\ell_0 = \ell_1 + 500\ell_2$ samples;
  for each $j \in [B]$, partition $S^{(j)}_{u_i}$ further into two multi-sets $S^{(j,1)}_{u_i}$ and $S^{(j,2)}_{u_i}$, where $|S^{(j,1)}_{u_i}| = \ell_1$ and $|S^{(j,2)}_{u_i}| = 500\ell_2$;
for j *from* 1 *to* B **do**
  $\widehat{Z}^{(j)}_\Lambda \leftarrow \mathsf{Estimate}\Big((S^{(j,1)}_{u_i})_{i\in[d]}, \ell_1, \ell_1\Big)$;
  $\widetilde{Z}^{(j)}_\Lambda \leftarrow \mathsf{Estimate}\Big((S^{(j,2)}_{u_i})_{i\in[d]}, 500\ell_2, 25\ell_2 \cdot \min\{2/\widehat{Z}^{(j)}_\Lambda, 4n\}\Big)$;
  $Q^{(j)}_\Lambda \leftarrow \widetilde{Z}^{(j)}_\Lambda \sum_{i=1}^{d} \widetilde{R}^{(i)}_\Lambda$;
return $\widetilde{R}_\Lambda := \text{median}\Big\{Q^{(1)}_\Lambda, Q^{(2)}_\Lambda, \ldots, Q^{(B)}_\Lambda\Big\}$;

speaking, large T means large overlap among $\Omega^{(t)}_\Lambda$'s, and the chance of hitting each vertex is roughly the same, resulting in the number of samples required per vertex close to the average. Conversely, small T means little overlap, and some vertex or vertices may be sampled much more often than other vertices, but in this case the overall number of samples required, namely T, is small anyways. To summarize, with this two-round procedure, $O(\ell)$ overall samples per vertex are enough to obtain an estimation with the desired accuracy and high probability.

4 Sketch of analysis

In this section, we give a sketch of the analysis of our algorithm. The complete analysis is in the full version of this paper. Roughly speaking, what we need to show is that in the induction step, accurate samples (in terms of the TV distance) imply accurate estimates, and accurate estimates imply accurate samples.

First consider $\mathsf{Sample}\Big(v_k, (\widetilde{R}_w, S_w)_{w\in\{v_{k+1},v_{k+2},\ldots,v_n\}}, \widetilde{R}_{v_k}\Big)$ as being called by Algorithm 1. Let $u := v_k$, and the subroutine runs on the graph $G_u = (V_u, E_u)$. We show that in $\mathsf{Sample}(\cdot)$, if every value returned by the $\mathsf{ApproxCount}(\cdot)$ subroutine is accurate, then $\mathsf{Sample}(\cdot)$ also returns accurate samples. Formally, we assume access to an oracle $\mathcal{P}$ satisfying:

- given $u \in V_u$, $\mathcal{P}$ returns p_0 such that $1 - \frac{1}{10m} \le \frac{p_0}{R(V_u,E_u,\{u\})} \le 1 + \frac{1}{10m}$;
- given any $E_2 \subseteq E_u$ and $\Lambda, \Lambda_1 \subseteq V$ in Algorithm 2 and Algorithm 2 of Algorithm 2, $\mathcal{P}$ returns $c_0(V_u, E_2, \Lambda)$ and $c_1(V_u, E_2, \Lambda_1)$ such that $1 - \frac{1}{10m} \le \frac{c_0(V_u,E_2,\Lambda)}{R(V_u,E_2,\Lambda)} \le 1 + \frac{1}{10m}$, and $1 - \frac{1}{10m} \le \frac{c_1(V_u,E_2,\Lambda_1)}{R(V_u,E_2,\Lambda_1)} \le 1 + \frac{1}{10m}$.

Here, we use the convention $\frac{0}{0} = 1$ and $\frac{x}{0} = \infty$ for $x > 0$. For any V, E and $U \subseteq V_u$, $R(V, E, U)$ is the U-t reliability in the graph (V, E). The numbers p_0, c_0 and c_1 returned by $\mathcal{P}$ can be random variables, but we assume that the inequalities above are always satisfied.

Algorithm 4 Estimate$\left((S^{es}_{u_i})_{i\in[d]}, \ell_{es}, T\right)$.

Input: a set of samples $S^{es}_{u_i}$ for each $i \in [d]$, where $|S^{es}_{u_i}| = \ell_{es}$, a threshold T
Output: an estimator Z_{es} of $\mathbb{E}[Z_\Lambda]$

for each $i \in [d]$, let $c_i = 0$;
for k *from* 1 *to* T **do**
  draw an index $i \in [d]$ such that i is drawn with probability proportional to $\widetilde{R}^{(i)}_\Lambda$;
  $c_i \leftarrow c_i + 1$;
  if $c_i > \ell_{es}$ **then return** 0;
  let $H = (V_{u_i}, E_H)$ be the c_i-th sample from $S^{(j)}_{u_i}$;
  do the following transformation on H to get $H' = (V, E_{H'})$;
  - let $E_{H'} \leftarrow E_H$;
  - draw $D \subseteq \delta_\Lambda(u_i)$ with probability proportional to (13), and let $E_{H'} \leftarrow E_{H'} \cup D$;
  - for each $e \in E \setminus (E_{u_i} \cup \delta_\Lambda(u_i))$, add e into $E_{H'}$ independently with probability $1 - q_e$;

  let $Z^{(k)}_{es} \in \{0, 1\}$ indicate whether i is the smallest index $t \in [d]$ satisfying $H' \in \Omega^{(t)}_\Lambda$;
return $Z_{es} := \frac{1}{T}\sum_{k=1}^{T} Z^{(k)}_{es}$;

Consider the following modified sampling algorithm, in which we replace Algorithm 2, Algorithm 2 and Algorithm 2 of Algorithm 2 by calling the oracle $\mathcal{P}$. This modified algorithm has some nice properties, summarized in the next lemma.

▶ **Lemma 10.** *Given any $u = v_k \in V$, the with probability at least $1 - (\varepsilon/n)^{200}$, the modified sampling algorithm does not crash and returns a perfect independent sample from the distribution π_u. The running time is $\widetilde{O}(N(|E_u| + |V_u|))$, where N is the time cost for one oracle call and $\widetilde{O}$ hides polylog(n/ε) factors.*

In the full analysis, we need to handle the real sampling algorithm, the analysis of which also relies on the analysis of ApproxCount. ApproxCount behaves like $\mathcal{P}$ but only with high probability. To analyze the error bounds, we also need to identify several random sources and handle errors from different sources separately.

Next consider ApproxCount$(V, E, \Lambda, (\widetilde{R}_w, S_w)_{w\in\partial\Lambda})$, where $G = (V, E)$ is a DAG and $t \notin \Lambda$. Recall that for any $w \in V$, the graph $G_w = G[V_w]$, where V_w contains all vertices v satisfying $w \rightsquigarrow_G v$ and $v \rightsquigarrow_G t$. Let R_w be the $w - t$ reliability in G_w. Recall that ℓ_2 and B are parameters in ApproxCount, Algorithm 3. The next lemma shows that if in the input, all $\widetilde{R}_w$ are accurate estimators of R_w and all S_w are accurate samples, then ApproxCount$(\cdot)$ outputs an accurate estimator to R_Λ. Let S^{ideal}_w be a multi-set of ℓ *independent* and *prefect* samples from π_w.

▶ **Lemma 11.** *Suppose the following conditions are satisfied*
- *for any $w \in \partial\Lambda$, $1 - \varepsilon_0 \le \frac{\widetilde{R}_w}{R_w} \le 1 + \varepsilon_0$ for some $\varepsilon_0 < 1/2$;*
- *$d_{TV}\left((S_w)_{w\in\partial\Lambda}, (S^{\text{ideal}}_w)_{w\in\partial\Lambda}\right) \le \delta_0$.*

Then with probability at least $1 - \delta_0 - 2^{-B/30}$, it holds that

$$1 - \varepsilon_0 - \frac{2}{\sqrt{\ell_2}} \le \frac{\widetilde{R}_\Lambda}{R_\Lambda} \le 1 + \varepsilon_0 + \frac{2}{\sqrt{\ell_2}}. \tag{15}$$

The running time of ApproxCount *is $O(n\ell(|V| + |E|))$.*

As mentioned before, due to reusing samples, there is correlation among $\widetilde{R}_w$ and S_w for different w's. However, the conditions of Lemma 11 do not rely on correlation, which is key to the correctness of the algorithm.

We note that the failure probability in Lemma 11 is exponentially small (we plug the bound in (16) as δ_0). This is necessary because we eventually need to apply a union bound over the exponential possibilities of spanning subgraphs.

Theorem 1 is proved by combining Lemma 10 and Lemma 11. We use an induction from $v_n = t$ to $v_1 = s$ to show that the following events occur with high probability:

- for any $j \geq i$, let $S_{v_j}^{\text{ideal}}$ be a multi-set of ℓ independent perfect samples from π_{v_j}, it holds that

$$d_{\text{TV}}\left((S_{v_j})_{j\geq i}, (S_{v_j}^{\text{ideal}})_{j\geq i}\right) \leq 2^{-4m}(2^{n-i}-1); \tag{16}$$

- the estimator $\widetilde{R}_{v_i}$ approximates the $v_i - t$ reliability R_{v_i}:

$$1 - \frac{n-i}{50n\max\{m,\varepsilon^{-1}\}} \leq \frac{\widetilde{R}_{v_i}}{R_{v_i}} \leq 1 + \frac{n-i}{50n\max\{m,\varepsilon^{-1}\}}. \tag{17}$$

The base case of $i = n$ is trivial. In the induction step, by using the I.H., we can show that the condition in Lemma 11 is satisfied and thus $\mathsf{ApproxCount}(V, E, \Lambda, (\widetilde{R}_w, S_w)_{w\in\partial\Lambda})$ returns an accurate estimator of R_Λ. Hence, the subroutine $\mathsf{ApproxCount}(\cdot)$ behaves similarly like the oracle $\mathcal{P}$, which, by Lemma 10, implies that the $\mathsf{Sample}(\cdot)$ algorithm also generates accurate samples.

Here, we briefly explain why we set the parameter $\ell_2 = O(n^2\max\{m^2,\varepsilon^{-2}\})$, which controls the number of samples maintained (as $\ell = \Theta(B\ell_2)$, recall (14)) and thus the overall running-time of the algorithm. We want to guarantee that $\widetilde{R}_\Lambda$ returned by $\mathsf{ApproxCount}(\cdot)$ achieves the same accuracy as that returned by $\mathcal{P}$. Note that the relative error in (17) serves as ε_0 in (15). Therefore combining (15) and (17), we need

$$\frac{n-i}{50n\max\{m,\varepsilon^{-1}\}} + \frac{2}{\sqrt{\ell_2}} \leq \frac{1}{50\max\{m,\varepsilon^{-1}\}} + \frac{2}{\sqrt{\ell_2}} \leq \frac{1}{10m}. \tag{18}$$

To inductively prove (17), we also need

$$\frac{n-(i+1)}{50n\max\{m,\varepsilon^{-1}\}} + \frac{2}{\sqrt{\ell_2}} \leq \frac{n-i}{50n\max\{m,\varepsilon^{-1}\}}. \tag{19}$$

The condition in (18) forces us to assume a strong induction hypothesis such that for each v_i, $\widetilde{R}_{v_i}$ has $1 \pm O(\frac{1}{m})$ approximation error. And the condition in (19) is the bottleneck for setting parameter ℓ_2.

In the full analysis, we use a more complicated induction hypothesis in the induction proof. Moreover, we need to define certain bad and good events to carefully analyze where errors come from and how they accumulate in every step.

The approximation guarantee in Theorem 1 follows directly from (17). Note that this guarantee is always at least a $(1 \pm 1/m)$-approximation and is stronger than a $(1 \pm \varepsilon)$-approximation when $\varepsilon > 1/m$. For the running time, recall that the number of samples per vertex is $\ell = O\big(n^2 m\max\{m^2,\varepsilon^{-2}\}\big)$. The running time of $\mathsf{ApproxCount}$ (Algorithm 3) is at most

$$T_{\mathsf{count}} = O(mn\ell) = O\big(n^3m^2\max\{m^2,\varepsilon^{-2}\}\big).$$

The running time of Sample (Algorithm 2) is at most

$$T_{\mathsf{sample}} = \widetilde{O}((n+m)T_{\mathsf{count}}) = \widetilde{O}(mT_{\mathsf{count}}) = \widetilde{O}\big(n^3m^3\max\{m^2,\varepsilon^{-2}\}\big).$$

Hence, the running time of Algorithm 1 is

$$T = O(n(T_{\mathsf{count}} + \ell T_{\mathsf{sample}})) = O(n\ell T_{\mathsf{sample}}) = \widetilde{O}\big(n^6m^4\max\{m^4,\varepsilon^{-4}\}\big).$$

5 #BIS-hardness for $s-t$ unreliability

In this section we show Theorem 2. We first reduce #**BIS** to $s-t$ unreliability where each vertex (other than s and t) fails with $1/2$ probability independently. Note that in this version of the problem no edge would fail. Given a DAG $D=(V\cup\{s,t\},A)$, this is equivalent to counting the number of subsets $S\subseteq V$ such that in the induced subgraph $D[S\cup\{s,t\}]$, s cannot reach t. We call S a $s\not\rightsquigarrow t$ set.

Given a bipartite graph $G=(V,E)$, let its two partitions be L and R. We add two special vertices s and t, and connect, with directed edges, s to all vertices in L and all vertices in R to t. Lastly, for any edge $\{u,v\}\in E$, where $u\in L$ and $v\in R$, we replace it with a directed edge (u,v). Call the new directed graph D_G. Clearly it is a DAG.

For any independent set I in G, we claim that in $D_G[I\cup\{s,t\}]$, s cannot reach t. This is because for any $e\in E$, there is at least one vertex unoccupied. Thus s cannot reach t using the directed version of e, and this holds for any $e\in E$.

In the other direction, let S be a $s\not\rightsquigarrow t$ subset of V. This means that for any edge $\{u,v\}\in E$, either $u\notin S$ or $v\notin S$, as otherwise $s\to u\to v\to t$. This means that S is an independent set of G.

Thus, there is a one-to-one correspondence between independent sets of G and $s\not\rightsquigarrow t$ subsets of V. Namely, $s-t$ unreliability where vertices (other than s and t) fail with $1/2$ probability is #**BIS**-hard.

Next, we reduce $s-t$ unreliability from the vertex version. For this, we can replace each vertex v (other than s and t) by two vertices v,v' and a directed edge $v\to v'$. All incoming edges of v still goes into v, and all outgoing edges of v goes out from v'. Assign to the new edges the failure probabilities of their corresponding vertices, and assign failure probability 0 to all original edges. Clearly the unreliability is the same with these changes. To make failure probabilities uniform, we can replace edges with failure probability 0 by k parallel edges. Effectively, the connection fails only if all the parallel edges fail at the same time. If the failure probability of each edge is q, the probability of all parallel edges failing is q^k. As this probability approaches 0 exponentially fast, it is easy to set a polynomially bounded k so that the new unreliability is a sufficiently good approximation of the original.

As a side note, the last reduction also works for reliability. Thus Theorem 1 also works for $s-t$ reliability in DAGs where vertices rather than edges fail independently.

References

1 Carme Àlvarez and Birgit Jenner. A very hard log-space counting class. *Theor. Comput. Sci.*, 107(1):3–30, 1993.

2 Antoine Amarilli, Timothy van Bremen, and Kuldeep S. Meel. Conjunctive queries on probabilistic graphs: the limits of approximability. *arXiv*, abs/2309.13287, 2023. `arXiv:2309.13287`.

3 Nima Anari, Kuikui Liu, Shayan Oveis Gharan, and Cynthia Vinzant. Log-concave polynomials II: high-dimensional walks and an FPRAS for counting bases of a matroid. In *STOC*, pages 1–12. ACM, 2019.

4 Nima Anari, Kuikui Liu, Shayan Oveis Gharan, Cynthia Vinzant, and Thuy-Duong Vuong. Log-concave polynomials IV: approximate exchange, tight mixing times, and near-optimal sampling of forests. In *STOC*, pages 408–420. ACM, 2021.

5 Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. #NFA admits an FPRAS: efficient enumeration, counting, and uniform generation for logspace classes. *J. ACM*, 68(6):48:1–48:40, 2021.

6 Michael O. Ball. Complexity of network reliability computations. *Networks*, 10(2):153–165, 1980.

7 Michael O. Ball. Computational complexity of network reliability analysis: An overview. *IEEE Trans. Rel.*, 35(3):230–239, 1986.

8 Michael O. Ball and J. Scott Provan. Calculating bounds on reachability and connectedness in stochastic networks. *Networks*, 13(2):253–278, 1983.

9 Ruoxu Cen, William He, Jason Li, and Debmalya Panigrahi. Beyond the quadratic time barrier for network unreliability. *arXiv*, abs/2304.06552, 2023. `arXiv:2304.06552`.

10 Xiaoyu Chen, Heng Guo, Xinyuan Zhang, and Zongrui Zou. Near-linear time samplers for matroid independent sets with applications. *arXiv*, abs/2308.09683, 2023. `arXiv:2308.09683`.

11 Charles J. Colbourn. *The Combinatorics of Network Reliability*. Oxford University Press, 1987.

12 Mary Cryan, Heng Guo, and Giorgos Mousa. Modified log-Sobolev inequalities for strongly log-concave distributions. *Ann. Probab.*, 49(1):506–525, 2021.

13 Martin E. Dyer, Leslie Ann Goldberg, Catherine S. Greenhill, and Mark Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2004.

14 Vivek Gore, Mark Jerrum, Sampath Kannan, Z. Sweedyk, and Stephen R. Mahaney. A quasi-polynomial-time algorithm for sampling words from a context-free language. *Inf. Comput.*, 134(1):59–74, 1997.

15 Heng Guo and Kun He. Tight bounds for popping algorithms. *Random Struct. Algorithms*, 57(2):371–392, 2020.

16 Heng Guo and Mark Jerrum. A polynomial-time approximation algorithm for all-terminal network reliability. *SIAM J. Comput.*, 48(3):964–978, 2019.

17 Heng Guo, Mark Jerrum, and Jingcheng Liu. Uniform sampling through the Lovász local lemma. *J. ACM*, 66(3):18:1–18:31, 2019.

18 David G. Harris and Aravind Srinivasan. Improved bounds and algorithms for graph cuts and network reliability. *Random Struct. Algorithms*, 52(1):74–135, 2018.

19 Mark Jerrum. *On the complexity of evaluating multivariate polynomials*. PhD thesis, The University of Edinburgh, 1981.

20 Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986.

21 David R. Karger. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM J. Comput.*, 29(2):492–514, 1999.

22 David R. Karger. A phase transition and a quadratic time unbiased estimator for network reliability. In *STOC*, pages 485–495. ACM, 2020.

23 Richard M. Karp and Michael Luby. Monte-Carlo algorithms for enumeration and reliability problems. In *FOCS*, pages 56–64. IEEE Computer Society, 1983.

24 Richard M. Karp and Michael Luby. Monte-Carlo algorithms for the planar multiterminal network reliability problem. *J. Complex.*, 1(1):45–64, 1985.

25 Richard M. Karp, Michael Luby, and Neal Madras. Monte-Carlo approximation algorithms for enumeration problems. *J. Algorithms*, 10(3):429–448, 1989.

26 Kuldeep S. Meel, Sourav Chakraborty, and Umang Mathur. A faster FPRAS for #NFA. *arXiv*, abs/2312.13320, 2023. `arXiv:2312.13320`.

27 J. Scott Provan. The complexity of reliability computations in planar and acyclic graphs. *SIAM J. Comput.*, 15(3):694–702, 1986.

28 J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4):777–788, 1983.

29 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

30 Rico Zenklusen and Marco Laumanns. High-confidence estimation of small s-t reliabilities in directed acyclic networks. *Networks*, 57(4):376–388, 2011.

A Note on Approximating Weighted Nash Social Welfare with Additive Valuations

Yuda Feng
Department of Computer Science and Technology, Harbin Institute of Technology, Heilongjiang, China

Shi Li
Department of Computer Science and Technology, Nanjing University, Jiangsu, China

Abstract

We give the first $O(1)$-approximation for the weighted Nash Social Welfare problem with additive valuations. The approximation ratio we obtain is $e^{1/e} + \epsilon \approx 1.445 + \epsilon$, which matches the best known approximation ratio for the unweighted case [3].

Both our algorithm and analysis are simple. We solve a natural configuration LP for the problem, and obtain the allocation of items to agents using a randomized version of the Shmoys-Tardos rounding algorithm developed for unrelated machine scheduling problems [30]. In the analysis, we show that the approximation ratio of the algorithm is at most the worst gap between the Nash social welfare of the optimum allocation and that of an EF1 allocation, for an unweighted Nash Social Welfare instance with identical additive valuations. This was shown to be at most $e^{1/e} \approx 1.445$ by Barman et al. [3], leading to our approximation ratio.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Nash Social Welfare, Configuration LP, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.63

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2404.15607

Funding *Shi Li*: The work of SL was supported by the State Key Laboratory for Novel Software Technology, and the New Cornerstone Science Laboratory.

Acknowledgements YF is an incoming PhD student at Nanjing University, and his work was a part of his undergraduate dissertation supervised by SL.

1 Introduction

In the weighted (or asymmetric) Nash Social Welfare problem with additive valuations, we are given a set $\mathcal{A}$ of n agents, and a set $\mathcal{G}$ of m indivisible items. Every agent $i \in \mathcal{A}$ has a weight $w_i \geq 0$ such that $\sum_{i \in \mathcal{A}} w_i = 1$. There is a value $v_{ij} \in \mathbb{R}_{\geq 0}$ for every $i \in \mathcal{A}$ and $j \in \mathcal{G}$. The goal of the problem is to find an allocation $\sigma : \mathcal{G} \to \mathcal{A}$ of items to agents so as to maximize the following weighted Nash social welfare of σ:

$$\prod_{i \in \mathcal{A}} \left(\sum_{j \in \sigma^{-1}(i)} v_{ij} \right)^{w_i}.$$

In the case where all w_i's are equal to $\frac{1}{n}$, we call the problem the unweighted (or symmetric) Nash Social Welfare problem.

Allocating resources in a fair and efficient manner among multiple agents is a fundamental problem in computer science, game theory, and economics, with applications across diverse domains [19, 33, 4, 28, 25, 2, 29, 5]. The weighted Nash social welfare function is a notable

objective that balances efficiency and fairness. The unweighted (or symmetric) objective was independently proposed by different communities [26, 20, 32], and later the study has been extended to the weighted case [16, 18]. Since then it has been used in a wide range of applications, including bargaining theory [21, 7, 31], water allocation [17, 10], and climate agreements [34].

The unweighted Nash Social Welfare problem with additive valuations is proved to be NP-hard by Nguyen et al. [27], and APX-hard by Lee [22]. Later the hardness of approximation was improved to $\sqrt{8/7} \approx 1.069$ by Garg et al. [12], via a reduction from Max-E3-Lin-2.

On the positive side, Cole and Gkatzelis [9] gave a $(2e^{1/e} + \epsilon \approx 2.889 + \epsilon)$-approximation using a market equilibrium with some spending restrictions. The ratio was improved by Cole et al. [8] to 2 using a tight analysis, and by Anari et al. [1] to e via a connection of the problem to real stable polynomials. Both papers formulated some convex program (CP) relaxations for the problem. In particular, [8] showed that the optimum solution to their CP corresponds to the spending-restricted market equilibrium defined in [9]. The state-of-the-art result for the problem is a combinatorial $(e^{1/e} + \epsilon \approx 1.45 + \epsilon)$-approximation algorithm due to Barman et al.[3]. They showed that when all the valuations of agents are identical, any allocation that is envy-free up to one item (EF1) is $e^{1/e}$-approximate. Their approximation result then follows from a connection between the non-identical and identical valuation settings they established.

All the results discussed above are for the unweighted case. For the weighted case with agent weights $w \in [0,1]^{\mathcal{A}}, |w|_1 = 1$, Brown et al. [6] presented a $5 \cdot \exp(2 \cdot D_{\mathrm{KL}}\left(w||\frac{\vec{1}}{n}\right)) = 5 \cdot \exp(2\log n + 2\sum_{i\in\mathcal{A}} w_i \log w_i)$ approximation algorithm, where D_{KL} denotes the KL divergence of two distributions. This is the first work that studies the weighted version for the additive valuation case. Prior to this work, there is an $O(nw_{\max}) = O(n \max_{i\in\mathcal{A}} w_i)$-approximation for the more general submodular valuation case [13], which we discuss soon. Brown et al. [6] showed that the two CPs from [8] and [1] are equivalent, and their result is based on the CP from [8], generalized to the weighted setting.

The additive valuation setting is a special case of the submodular valuation setting, which is another important setting studied in the literature. In this setting, instead of a v_{ij} value for every ij pair, we are given a monotone submodular function $v_i : 2^{\mathcal{G}} \to \mathbb{R}_{\geq 0}$ for every agent $i \in \mathcal{A}$. Our goal is to find an allocation $\sigma : \mathcal{G} \to \mathcal{A}$ so as to maximize $\prod_{i\in\mathcal{A}} \left(v_i(\sigma^{-1}(i))\right)^{w_i}$. A bulk of the previous work has focused on the unweighted case; that is, $w_i = \frac{1}{n}$ for all $i \in \mathcal{A}$. For this case, Garg et al. [15] proved a hardness of $e/(e-1) \approx 1.5819$ using a reduction from Max-3-Coloring; this is better than the 1.069 hardness for the additive valuation case.

On the positive side, Li and Vondrak [24] extended the techniques of Anari et al. [1], to obtain an $e^3/(e-1)^2$-approximation algorithm for the unweighted Nash Social Welfare problem for a large family of submodular valuations, including coverage functions and linear combinations of matroid rank functions. Later, Garg et al. [14] considered a family of submodular functions called Rado functions, and gave an $O(1)$-approximation for this family using the matching theory and convex program techniques. Li and Vondrak [23] developped the first $O(1)$-approximation for general submodular functions, with an approximation ratio of 380. Recently, Garg et al. [13] presented an elegant 4-approximation local search algorithm for the problem, which is the current best approximation result for the problem. All the results discussed above are for the unweighted case. For the weighted case, Garg et al. [13] gave an $O(nw_{\max})$-approximation, where $n_{\max} = \max_{i\in\mathcal{A}} w_i$. Whether the weighted Nash Social Welfare problem with submodular valuations admits a constant approximation is a big open problem.

Recently, the problem has been studied in an even more general setting, namely, the subadditive valuation setting. Dobzinski et al. [11] gives an $O(1)$-approximation for the unweighted Nash Social Welfare problem in this setting under the demand oracle model.

1.1 Our Result and Techniques

In this note, we give the first $O(1)$-approximation algorithm for the weighted Nash Social Welfare problem with additive valuations:

▶ **Theorem 1.** *For any $\epsilon > 0$, there is a randomized $(e^{1/e} + \epsilon \approx 1.445 + \epsilon)$-approximation algorithm for the weighted Nash Social Welfare problem with additive valuations, with running time polynomial in the size of the input and $\frac{1}{\epsilon}$.*

Our approximation ratio of $e^{1/e} + \epsilon$ matches the best ratio for the unweighted case due to Barman et al. [3]. In contrast, the ratio given by Brown et al. [6] is $5 \cdot \exp(2 \cdot D_{\mathrm{KL}}(w||\frac{\vec{1}}{n}))$, which could be polynomial in n.

Our algorithm is based on a natural configuration LP for the problem, which has not been studied before to the best of our knowledge. The configuration LP contains a $y_{i,S}$ variable for every agent i and subset S of items, indicating if the set of items i gets is S or not. We show that the configuration LP can be solved in polynomial time to any precision, despite having exponential number of variables. Once we obtain the LP solution, we define x_{ij} for every $i \in \mathcal{A}$ and $j \in \mathcal{G}$ to be the fraction of j assigned to i.

We use a randomized version of the Shmoys-Tardos rounding algorithm [30] developed for unrelated machine scheduling problems, to round x into an integral solution. For every agent i, we break the fractional items assigned to i into groups from the most valuable to the least, each containing 1 fractional item. The rounding algorithm maintains marginal probabilities, and the requirement that i gets exactly one item from each group (except for the last one, from which i gets at most one item). In the analysis for each agent i, we construct an instance of the unweighted Nash Social Welfare problem with *identical* additive valuations, that involves many copies of the agent i, along with two alloations $\mathcal{S}$ and $\mathcal{S}'$ to the instance. $\mathcal{S}$ corresponds to the LP solution, and $\mathcal{S}'$ corresponds to the randomized solution given by the rounding algorithm. Thanks to the condition that every group contains one item, the solution $\mathcal{S}'$ is envy-free up to one item (EF1). Using the result of [3] about EF1 allocations, we show that the Nash social welfare of $\mathcal{S}'$ is at least $e^{-1/e}$ times that of $\mathcal{S}$, which eventually leads to our $(e^{1/e} + \epsilon)$-approximation.

We believe the configuration LP could be used in many other settings. We leave as an immediate open problem whether it can give an $O(1)$-approximation for the weighted Nash Social Welfare problem with submodular valuations.

2 $(e^{1/e} + \epsilon)$-Approximation Using Configuration LP

We describe the configuration LP in Section 2.1 and the rounding algorithm in Section 2.2. The analysis is given in Section 2.3.

2.1 The Configuration LP

For convenience, for any value function $v : \mathcal{G} \to \mathbb{R}_{\geq 0}$, we define $v(S) := \sum_{j \in S} v_j$ for every $S \subseteq \mathcal{G}$ to be the total value of items in S according to the value funciton v. In the integer program correspondent to the configuration LP, for every $i \in \mathcal{A}$ and $S \subseteq \mathcal{G}$, we have a variable $y_{i,S} \in \{0, 1\}$ indicating if the set of items assigned to i is S or not. We relax the integer constraint to obtain the following configuration LP:

$$\max \sum_{i\in\mathcal{A},S\subseteq\mathcal{G}} w_i \cdot y_{i,S} \cdot \ln v_i(S) \qquad \text{s.t.} \tag{Conf-LP}$$

$$\sum_{i\in\mathcal{A},S\ni j} y_{i,S} \le 1 \qquad \forall j \in \mathcal{G} \tag{1}$$

$$\sum_{S\subseteq\mathcal{G}} y_{i,S} = 1 \qquad \forall i \in \mathcal{A} \tag{2}$$

$$y_{i,S} \ge 0 \qquad \forall i \in \mathcal{A}, S \subseteq \mathcal{G} \tag{3}$$

It is convenient for us to consider the natural logarithm of the Nash social welfare function as the objective, which is $\sum_{i\in\mathcal{A}} w_i \cdot \ln v_i(\sigma^{-1}(i))$. This leads to the objective in (Conf-LP). (1) requires that every item j is assigned to at most one agent, and (2) requires that every agent i is assigned one set of items.

The configuration LP has exponential number of variables, but it can be solved within an additive error of $\ln(1+\epsilon)$ for any $\epsilon > 0$, in time polynomial in the size of the instance and $\frac{1}{\epsilon}$. We defer the details to Appendix A. Notice that we are considering the logarithm of Nash social welfare, and the typical $(1+\epsilon)$-multiplicative factor becomes an additive error of $\ln(1+\epsilon)$.

2.2 The Rounding Algorithm

From now on, we assume we have obtained a vector y from solving the LP, described using a list of non-zero coordinates; the value of y to (Conf-LP) is at least the optimum value minus $\ln(1+\epsilon)$. We can assume (1) holds with equalities: $\sum_{i\in\mathcal{A},S\ni j} y_{i,S} = 1$ for every $j \in \mathcal{G}$. Then we let $x_{ij} = \sum_{S\ni j} y_{i,S}$ for every $i \in \mathcal{A}$ and $j \in \mathcal{G}$. So $\sum_{i\in\mathcal{A}} x_{ij} = 1$ for every $j \in \mathcal{G}$.

In this paragraph, we fix an agent i and break the fractional items assigned to i into a set G_i of groups, each containing 1 fractional item. They are created in non-increasing order of values, as in the Shmoys-Tardos algorithm for unrelated machine scheduling problems. That is, the first group contains the 1 fractional most valuable items assigned to i, the second group contains the 1 fractional most valuable items assigned to i after removing the first group, and so on. Formally, we sort the items in $\mathcal{G}$ in non-increasing order of v_{ij} values, breaking ties arbitrarily. Let $p_i = \lceil \sum_{j\in\mathcal{G}} x_{ij} \rceil$. Then we can find vectors $g^1, g^2, \cdots, g^{p_i} \in [0,1]^{\mathcal{G}}$ satisfying the following properties:

- **(P1)** For every $t \in [1, p_i-1]$, we have $|g^t|_1 = 1$; for $t = p_i$, we have $|g^t|_1 = \sum_{j\in\mathcal{G}} x_{ij} - (p_i-1) \in (0,1]$.
- **(P2)** $\sum_{t=1}^{p_i} g^t_j = x_{ij}$ for every $j \in \mathcal{G}$.
- **(P3)** For every $1 \le t < t' \le p_i$, and two items j, j' such that j appears before j' in the ordering, it can not happen that $g^t_{j'} > 0$ and $g^{t'}_j > 0$.

It is easy to see that $g^1, g^2, \cdots, g^{p_i}$ are uniquely decided by the three conditions. We say each g^t is a group. Let $G_i = \{g^1, g^2, \cdots, g^{p_i}\}$ be the set of all groups constructed for this agent i.

Now we take all agents i into consideration and let $G = \uplus_{i\in\mathcal{A}} G_i$ be the set of all groups constructed.[1] The representations of groups give a fractional matching between the groups G and items $\mathcal{G}$: an item j is matched to a group $g \in [0,1]^{\mathcal{G}}$ with a fraction of g_j. Then each item is matched to an extent of 1, and every group g is matched to an extent of $|g|_1$. So

[1] It is possible that two groups from different sets G_i and $G_{i'}$ have the same vector representation. So we treat G as a multiset and we assume we know which set G_i each group $g \in G$ belongs to.

a group is matched to an extent of 1 if it is not the last group for an agent, and at most 1 otherwise. Therefore, we can efficiently output a randomized (partial-)matching between the groups G and items $\mathcal{G}$ so that the marginal probabilities are maintained:
($\star$) For every group $g \in G$ and item $j \in \mathcal{G}$, we have $\Pr[j \text{ is matched to } g] = g_j$.

($\star$) implies that an item $j \in \mathcal{G}$ is matched with probability 1. If a group g has $|g|_1 = 1$, then it is matched with probability 1.

The matching naturally gives us an allocation of items to agents: If an item $j \in \mathcal{G}$ is matched to some group $g \in G_i$, then we assign j to i. By ($\star$) we know that the probability that j is assigned to i is precisely x_{ij}. Let S_i be the set of items assigned to i in the algorithm; notice that it is random. This finishes the description of the randomized rounding algorithm.

2.3 The Analysis

To analyze our rounding algorithm, we first formally define an EF1 allocation.

▶ **Definition 2.** *Given an instance of the unweighted Nash Social Welfare problem with agents $\mathcal{A}$, items $\mathcal{G}$, and identical additive valuation $v : \mathcal{G} \to \mathbb{R}_{\geq 0}$ for all agents, an allocation $\sigma : \mathcal{G} \to \mathcal{A}$ is said to be envy-free up to one item (EF1), if for every two distinct agents i, i' with $\sigma^{-1}(i') \neq \emptyset$, there exists some $j \in \sigma^{-1}(i')$, such that $v(\sigma^{-1}(i') \setminus j) \leq v(\sigma^{-1}(i))$.*

We use the following result from [3]:

▶ **Theorem 3** ([3])**.** *For the unweighted Nash Social Welfare problem with identical additive valuations, any EF1-allocation is an $e^{1/e}$-approximate solution.*

With the theorem, we prove the following key lemma:

▶ **Lemma 4.** *For every $i \in \mathcal{A}$, we have*

$$\mathbb{E}\left[\ln v_i(S_i)\right] \geq \sum_{S \subseteq \mathcal{G}} y_{i,S} \cdot \ln v_i(S) \; - \; \frac{1}{e}.$$

Proof. Throughout the proof, we fix the agent i. Let $\Delta > 0$ be an integer, so that every $y_{i,S}$ is an integer multiply of $1/\Delta$, and the probability that $S_i = S$ for any S is also an integer multiply of $1/\Delta$.[2] We consider an instance of the unweighted Nash Social Welfare problem with identical additive valuations. In the instance, there are Δ copies of the agent i, and Δx_{ij} copies of every item $j \in \mathcal{G}$; so all the agents are identical. The $y = (y_{i,S})_{S \subseteq \mathcal{G}}$ vector gives us an allocation $\mathcal{S}$ to the instance: For every $S \subseteq \mathcal{G}$, there are exactly $\Delta y_{i,S}$ agents who get a copy of S. Notice that this is a valid solution, as $\sum_S y_{i,S} = 1$ and $\sum_{S \ni j} y_{i,S} = x_{ij}$ for every item j.

The Nash Social Welfare of the allocation $\mathcal{S}$ is

$$\left(\prod_{S \subseteq \mathcal{G}} v_i(S)^{\Delta y_{i,S}}\right)^{1/\Delta} = \prod_{S \subseteq \mathcal{G}} v_i(S)^{y_{i,S}}.$$

The distribution for S_i also corresponds to an allocation $\mathcal{S}'$ of items to agents: For every $S \subseteq \mathcal{G}$, there are $\Delta \cdot \Pr[S_i = S]$ agents who get a copy of S. Again, this is a valid solution as $\sum_S \Pr[S_i = S] = 1$ and $\sum_{S \ni j} \Pr[S_i = S] = \mathbb{E}[S_i \ni j] = x_{ij}$.

[2] We can assume all $y_{i,S}$ values are rational numbers. Under this condition, it is easy to guarantee that the probabilities are rational numbers.

The Nash Social Welfare of the allocation $\mathcal{S}'$ is

$$\left(\prod_{S\subseteq\mathcal{G}} v_i(S)^{\Delta \Pr[S_i=S]}\right)^{1/\Delta} = \prod_{S\subseteq\mathcal{G}} v_i(S)^{\Pr[S_i=S]}.$$

A crucial property for the solution $\mathcal{S}'$ is that it is EF1. Indeed, if $\Pr[S_i = S] > 0$ for some S, then S contains exactly one item from each group in G_i except for the last one, from which S contains at most one item. Also, the items in the groups G_i are sorted by (P3). So if there are two sets S and S' in the support of the distribution for S_i, and we remove the most valuable item from S', then S beats S' item by item.

Therefore, by Theorem 3, we know that the Nash Social Welfare of $\mathcal{S}'$ is at least $e^{-1/e}$ times that of the optimum allocation for the instance, which is at least that of $\mathcal{S}$. That is,

$$\prod_{S\subseteq\mathcal{G}} v_i(S)^{\Pr[S_i=S]} \geq e^{-1/e} \cdot \prod_{S\subseteq\mathcal{G}} v_i(S)^{y_{i,S}}.$$

Taking logarithm on both sides gives the lemma. ◀

Applying the lemma for every $i \in \mathcal{A}$ and using linearity of expectation, we have

$$\mathbb{E}\left[\sum_{i\in\mathcal{A}} w_i \cdot \ln v_i(S_i)\right] \geq \sum_{i\in S, S\subseteq\mathcal{G}} w_i \cdot y_{i,S} \cdot \ln v_i(S) - \frac{1}{e}.$$

We used that $\sum_{i\in\mathcal{A}} w_i = 1$.

By the convexity of exponential function, we have

$$\mathbb{E}\left[\prod_{i\in\mathcal{A}} v_i(S_i)^{w_i}\right] \geq e^{-1/e} \cdot \exp\left(\sum_{i\in S, S\subseteq\mathcal{G}} w_i \cdot y_{i,S} \cdot \ln v_i(S_i)\right) \geq e^{-1/e} \cdot \frac{\text{opt}}{1+\epsilon},$$

where opt is the Nash Social Welfare of the optimum allocation, and the second inequality used that the value of our solution y to (Conf-LP) is at least its optimum value minus $\ln(1+\epsilon)$. By scaling ϵ down by an absolute constant at the beginning, we can make the right side to be at least $\frac{\text{opt}}{e^{1/e}+\epsilon}$. This finishes the proof of Theorem 1.

Finally, we briefly discuss how to derandomize the rounding algorithm. We round the solution to the configuration LP in iterations, maintaining a fractional assignment $\bar{x}$ of items to agents; $\bar{x} = x$ initially. Let Δ be a large enough integer so that every $y_{i,S}$ is an integer multiply of $1/\Delta$. Focus on a fixed agent $i \in \mathcal{A}$ and consider the Nash Social Welfare instance containing Δ copies of i, and $\Delta\bar{x}_{ij}$ copies of each item $j \in \mathcal{G}$. Group the items as follows: the Δ most valuable items belong to the first group, the next Δ most valuable items belong to the second group, and so on. We define Φ_i to be the logarithm of the Nash Social Welfare of the worst allocation satisfying the following condition: every agent gets at most one item from each group. Fortunately, the worst allocation can be defined naturally: the first agent takes the most valuable item from each group, and the second agent takes the second most valuable item from each group, and so on. Thus Φ_i can be computed efficiently. We define $\Phi = \sum_i w_i\Phi_i$ to be the overall potential function. In the randomized version of the algorithm, one can define the rotation operation over the fractional matching between groups G and items $\mathcal{G}$. In expectation the operation does not decrease Φ. To derandomize the algorithm, we can perform the operation deterministically so that Φ does not decrease. The potential value Φ at the end of the algorithm is at least that at the beginning, which is at least the value of the configuration LP minus $1/e$. On the other hand, the logarithm of the Nash Social Welfare of the integral solution is exactly the final Φ. Therefore, the Nash Social Welfare is at least $e^{-1/e}$ times the exponential of the value of the configuration LP.

References

1 Nima Anari, Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. Nash Social Welfare, Matrix Permanent, and Stable Polynomials. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*, volume 67 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:12, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2017.36.

2 Julius Barbanel and Alan Taylor. The Geometry of Efficient Fair Division. *The Geometry of Efficient Fair Division*, pages 1–462, January 2005. doi:10.1017/CBO9780511546679.

3 Siddharth Barman, Sanath Kumar Krishnamurthy, and Rohit Vaish. Finding Fair and Efficient Allocations. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, EC '18, pages 557–574, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3219166.3219176.

4 Steven J. Brams and Alan D. Taylor. *Fair Division: From Cake-Cutting to Dispute Resolution*. Cambridge University Press, 1996.

5 Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia. *Handbook of Computational Social Choice*. Cambridge University Press, USA, 1st edition, 2016.

6 Adam Brown, Aditi Laddha, Madhusudhan Reddy Pittu, and Mohit Singh. Approximation Algorithms for the Weighted Nash Social Welfare via Convex and Non-Convex Programs. In *Proceedings of the Thirty-Fifth ACM-SIAM Symposium on Discrete Algorithms*, 2024.

7 Suchan Chae and Hervé Moulin. Bargaining Among Groups: An Axiomatic Viewpoint. *International Journal of Game Theory*, 39(1):71–88, 2010. doi:10.1007/s00182-009-0157-6.

8 Richard Cole, Nikhil Devanur, Vasilis Gkatzelis, Kamal Jain, Tung Mai, Vijay V. Vazirani, and Sadra Yazdanbod. Convex Program Duality, Fisher Markets, and Nash Social Welfare. In *Proceedings of the 2017 ACM Conference on Economics and Computation*, EC '17, pages 459–460, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3033274.3085109.

9 Richard Cole and Vasilis Gkatzelis. Approximating the Nash Social Welfare with Indivisible Items. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 371–380, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2746539.2746589.

10 Dagmawi Mulugeta Degefu, Weijun He, Liang Yuan, An Min, and Qi Zhang. Bankruptcy to Surplus: Sharing Transboundary River Basin's Water under Scarcity. *Water Resources Management: An International Journal, Published for the European Water Resources Association (EWRA)*, 32(8):2735–2751, 2018. URL: https://EconPapers.repec.org/RePEc:spr:waterr:v:32:y:2018:i:8:d:10.1007_s11269-018-1955-z.

11 Shahar Dobzinski, Wenzheng Li, Aviad Rubinstein, and Jan Vondrak. A Constant Factor Approximation for Nash Social Welfare with Subadditive Valuations. *ArXiv*, abs/2309.04656, 2023. arXiv:2309.04656.

12 Jugal Garg, Martin Hoefer, and Kurt Mehlhorn. Satiation in Fisher Markets and Approximation of Nash Social Welfare. *Mathematics of Operations Research*, 0(0):null, 2023. doi:10.1287/moor.2019.0129.

13 Jugal Garg, Edin Husić, Wenzheng Li, László A. Végh, and Jan Vondrák. Approximating Nash Social Welfare by Matching and Local Search. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 1298–1310, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3564246.3585255.

14 Jugal Garg, Edin Husić, and László A. Végh. Approximating Nash Social Welfare under Rado Valuations. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 1412–1425, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3406325.3451031.

15 Jugal Garg, Pooja Kulkarni, and Rucha Kulkarni. Approximating Nash Social Welfare under Submodular Valuations through (Un)Matchings. *ACM Trans. Algorithms*, 19(4), September 2023. doi:10.1145/3613452.

16 John C. Harsanyi and Reinhard Selten. A generalized nash solution for two-person bargaining games with incomplete information. *Management Science*, 18(5):P80–P106, 1972. URL: http://www.jstor.org/stable/2661446.

17 Harold Houba, Gerard van der Laan, and Yuyu Zeng. Asymmetric Nash Solutions in the River Sharing Problem. *Game Theory & Bargaining Theory eJournal*, 2013. URL: https://api.semanticscholar.org/CorpusID:17619205.

18 E. Kalai. Nonsymmetric Nash Solutions and Replications of 2-Person Bargaining. *Int. J. Game Theory*, 6(3):129–133, September 1977. doi:10.1007/BF01774658.

19 Mamoru Kaneko and Kenjiro Nakamura. The Nash Social Welfare Function. *Econometrica*, 47(2):423–435, 1979. URL: http://www.jstor.org/stable/1914191.

20 Frank Kelly. Charging and Rate Control for Elastic Traffic. *European Transactions on Telecommunications*, 8(1):33–37, 1997. doi:10.1002/ett.4460080106.

21 Annick Laruelle and Federico Valenciano. Bargaining in Committees as An Extension of Nash's Bargaining Theory. *Journal of Economic Theory*, 132(1):291–305, 2007. doi:10.1016/j.jet.2005.05.004.

22 Euiwoong Lee. APX-Hardness of Maximizing Nash Social Welfare with Indivisible Items. *ArXiv*, abs/1507.01159, 2015. arXiv:1507.01159.

23 W. Li and J. Vondrak. A Constant-Factor Approximation Algorithm for Nash Social Welfare with Submodular Valuations. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 25–36, Los Alamitos, CA, USA, February 2022. IEEE Computer Society. doi:10.1109/FOCS52979.2021.00012.

24 Wenzheng Li and Jan Vondrák. Estimating the Nash Social Welfare for Coverage and Other Submodular Valuations. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '21, pages 1119–1130, USA, 2021. Society for Industrial and Applied Mathematics.

25 Herve Moulin. *Fair Division and Collective Welfare*. The MIT Press, 2004.

26 John F. Nash. The Bargaining Problem. *Econometrica*, 18(2):155–162, 1950. URL: http://www.jstor.org/stable/1907266.

27 Nhan-Tam Nguyen, Trung Thanh Nguyen, Magnus Roos, and Jörg Rothe. Complexity and Approximability of Social Welfare Optimization in Multiagent Resource Allocation. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '12, pages 1287–1288, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.

28 Jack Robertson and William Webb. *Cake-Cutting Algorithms: Be Fair if You Can*. A K Peters/CRC Press, 1998.

29 Jrg Rothe. *Economics and Computation: An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division*. Springer Publishing Company, Incorporated, 1st edition, 2015.

30 David B. Shmoys and Éva Tardos. An Approximation Algorithm for the Generalized Assignment Problem. *Mathematical Programming*, 62(1):461–474, 1993. doi:10.1007/BF01585178.

31 William Thomson. Replication Invariance of Bargaining Solutions. *Int. J. Game Theory*, 15(1):59–63, March 1986. doi:10.1007/BF01769276.

32 Hal R Varian. Equity, Envy, and Efficiency. *Journal of Economic Theory*, 9(1):63–91, 1974. doi:10.1016/0022-0531(74)90075-1.

33 H. Peyton Young. *Equity: In Theory and Practice*. Princeton University Press, 1994. URL: http://www.jstor.org/stable/j.ctv10crfx7.

34 S. Yu, E. C. van Ierland, H. P. Weikard, and X. Zhu. Nash Bargaining Solutions for International Climate Agreements under Different Sets of Bargaining Weights. *International Environmental Agreements: Politics, Law and Economics*, 17(5):709–729, 2017. doi:10.1007/s10784-017-9351-3.

A Solving Configuration LP within an Additive Error of $\ln(1+\epsilon)$

Let $\epsilon > 0$ be upper bounded by a sufficiently small constant (we allow ϵ to be a sub-constant). By only allowing every agent to get one item, we can obtain an m-approximation for the our Nash Social Welfare instance. Then, by making $O\left(\frac{\log m}{\epsilon}\right)$ guesses, we can assume we are given a number o such that the value of (Conf-LP) is in $(o, o+\epsilon/3]$.

We consider the dual of (Conf-LP), with the objective replaced by a constraint.

$$\sum_{j\in\mathcal{G}} \alpha_j + \sum_{i\in\mathcal{A}} \beta_i \le o \tag{4}$$

$$\sum_{j\in S} \alpha_j + \beta_i \ge w_i \cdot \ln v_i(S) \qquad \forall i \in \mathcal{A}, S \subseteq \mathcal{G} \tag{5}$$

$$\alpha_j \ge 0 \qquad \forall j \in \mathcal{G} \tag{6}$$

Since (Conf-LP) has value strictly larger than o, the dual LP (4-6) is infeasible. We design an approximate separation oracle for the LP. Given some $\alpha \in \mathbb{R}^{\mathcal{G}}_{\ge 0}$ and $\beta \in \mathbb{R}^{\mathcal{A}}$ that does not satisfy (5), we can find some $i \in \mathcal{A}$ and $S \subseteq \mathcal{G}$ such that

$$\sum_{j\in S} \alpha_j + \beta_i < w_i \ln\big((1+\epsilon/2)v_i(S)\big).$$

The running time of the oracle is polynomial in the input size and $\frac{1}{\epsilon}$. This can be achieved using the standard dynamic programming technique: For a fixed $i \in \mathcal{A}$, to find the S, we guess the item $j^* \in S$ with the largest v_{ij^*}, coarsen the v_{ij} values based on the guess, and run a dynamic programming to find the S.

So, using the ellipsoid method with the approximate separation oracle, we can find polynomially many half spaces of the form $\sum_{j\in S} \alpha_j + \beta_i \ge w_i \ln\big((1+\epsilon/2)v_i(S)\big)$, whose intersection is empty. Then, we consider the Nash Social Welfare instance where all v_{ij} values are scaled up by $1+\epsilon/2$, and (Conf-LP) to the instance. By solving the LP restricted to the variables $y_{i,S}$ correspondent to the half spaces (that is, we let all other variables be 0), we obtain a solution y whose value is at least o w.r.t the scaled instance. So, the value of the solution y to (Conf-LP) w.r.t the original instance is at least $o - \sum_{i\in\mathcal{A}, S\subseteq\mathcal{G}} y_{i,S} w_i \ln(1+\epsilon/2) = o - \sum_i w_i \ln(1+\epsilon/2) = o - \ln(1+\epsilon/2)$.

As the value of (Conf-LP) is at most $o+\epsilon/3$, we solved the LP up to an additive error of $\epsilon/3 + \ln(1+\epsilon/2)$. For a small enough ϵ, this is at most $\ln(1+\epsilon)$.

Minimizing Tardy Processing Time on a Single Machine in Near-Linear Time

Nick Fischer
Weizmann Institute of Science, Rehovot, Israel

Leo Wennmann
Maastricht University, The Netherlands

Abstract

In this work we revisit the elementary scheduling problem $1||\sum p_jU_j$. The goal is to select, among n jobs with processing times and due dates, a subset of jobs with maximum total processing time that can be scheduled in sequence without violating their due dates. This problem is NP-hard, but a classical algorithm by Lawler and Moore from the 60s solves this problem in pseudo-polynomial time $O(nP)$, where P is the total processing time of all jobs. With the aim to develop best-possible pseudo-polynomial-time algorithms, a recent wave of results has improved Lawler and Moore's algorithm for $1||\sum p_jU_j$: First to time $\widetilde{O}(P^{7/4})$ [Bringmann, Fischer, Hermelin, Shabtay, Wellnitz; ICALP'20], then to time $\widetilde{O}(P^{5/3})$ [Klein, Polak, Rohwedder; SODA'23], and finally to time $\widetilde{O}(P^{7/5})$ [Schieber, Sitaraman; WADS'23]. It remained an exciting open question whether these works can be improved further.

In this work we develop an algorithm in near-linear time $\widetilde{O}(P)$ for the $1||\sum p_jU_j$ problem. This running time not only significantly improves upon the previous results, but also matches conditional lower bounds based on the Strong Exponential Time Hypothesis or the Set Cover Hypothesis and is therefore likely *optimal* (up to subpolynomial factors). Our new algorithm also extends to the case of m machines in time $\widetilde{O}(P^m)$. In contrast to the previous improvements, we take a different, more direct approach inspired by the recent reductions from Modular Subset Sum to dynamic string problems. We thereby arrive at a satisfyingly *simple* algorithm.

2012 ACM Subject Classification Theory of computation → Discrete optimization

Keywords and phrases Scheduling, Fine-Grained Complexity, Dynamic Strings

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.64

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2402.13357

Funding *Nick Fischer*: This work is part of the project CONJEXITY that has received funding from the European Research Council (ERC) under the European Union's Horizon Europe research and innovation programme (grant agreement No. 101078482).
Leo Wennmann: Supported by Dutch Research Council (NWO) project "The Twilight Zone of Efficiency: Optimality of Quasi-Polynomial Time Algorithms" [grant number OCEN.W.21.268].

1 Introduction

Consider the following natural optimization problem: A worker is offered n *jobs*, where each job j requires a *processing time* of p_j days and must be completed before some *due date* d_j. Which jobs should the worker take on in order to maximize their pay, assuming that the worker is paid a fixed amount per day of work? In standard scheduling notation [22], this task is somewhat cryptically called the "$1||\sum p_jU_j$" problem (see Section 2 for a formal definition). The $1||\sum p_jU_j$ problem constitutes an important scheduling task that is arguably among the *simplest* nontrivial scheduling objectives, and has received considerable attention in the literature, especially in recent years.

The $1||\sum p_jU_j$ problem naturally generalizes the famous Subset Sum problem,[1] and is therefore NP-hard. However, it does admit pseudo-polynomial-time algorithms – in 1969, Lawler and Moore [31] pioneered the first such algorithm in time $O(nP)$, where $P = \sum_j p_j$ is the total processing time of all jobs. This result is the baseline in a line of research that, more than 50 years after the initial effort, is finally brought to a close in this paper.

State of the Art. Lawler and Moore originally designed their algorithm for a weighted generalization of the $1||\sum p_jU_j$ problem, and for this generalization the running time $O(nP)$ was proven to be conditionally tight.[2] Even for the $1||\sum p_jU_j$ problem the Lawler-Moore algorithm remained unchallenged for a long time. Only a few years ago, Bringmann, Fischer, Hermelin, Shabtay and Wellnitz [12] managed to solve $1||\sum p_jU_j$ in time[3] $\widetilde{O}(P^{7/4})$, showcasing that improvements over Lawler-Moore are indeed possible in certain parameter regimes (specifically, when $P \ll n^{4/3}$). Their strategy is to design a tight reduction to an intermediate problem called *Skewed Convolution*[4], and to develop an $\widetilde{O}(N^{7/4})$-time algorithm for this intermediate problem.

Their work was later improved in two orthogonal ways. On the one hand, Klein, Polak and Rohwedder [28] improved the running time of Skewed Convolution to $\widetilde{O}(N^{5/3})$. On the other hand, Schieber and Sitaraman [36] improved the algorithmic reduction and established that, if Skewed Convolution is in time $\widetilde{O}(N^\alpha)$, then $1||\sum p_jU_j$ is in time $\widetilde{O}(P^{2-1/\alpha})$. The state-of-the-art algorithm for $1||\sum p_jU_j$ is obtained by combining these two works, resulting in time $\widetilde{O}(P^{7/5})$.

In contrast, fine-grained lower bounds for the Subset Sum problem rule out $1||\sum p_jU_j$ algorithms in time $O(P^{1-\epsilon} \cdot n^{O(1)})$, for any $\epsilon > 0$, conditioned on either the influential Strong Exponential Time Hypothesis [1] or the Set Cover Hypothesis [17]. This leaves a substantial gap between the best known upper bound $\widetilde{O}(P^{7/5})$ and the conceivable optimum $\widetilde{O}(P)$. Closing this gap is the starting point of our paper:

Can the $1||\sum p_jU_j$ *problem be solved in near-linear time* $\widetilde{O}(P)$*?*

In light of the recent algorithmic developments [12, 28, 36], a reasonable strategy appears to aim for even faster algorithms for the Skewed Convolution problem – unfortunately, this approach soon faces a barrier. Namely, improving the running time of Skewed Convolution beyond $O(N^{3/2})$ would entail a similarly fast algorithm for $(\max, \min)$-Convolution, which, while not ruled out under one of the big hypotheses, would be a surprising break-through in fine-grained complexity theory. This leaves us in an uncertain situation. Even if Skewed Convolution could be improved to time $\widetilde{O}(N^{3/2})$, this would mean that the $1||\sum p_jU_j$ problem is in time $\widetilde{O}(P^{4/3})$ [36]. Are further improvements impossible?

Our Results. In this paper we bypass this barrier and develop a new algorithm for $1||\sum p_jU_j$ that avoids the reduction to Skewed Convolution altogether. We thereby successfully resolve our driving question:

---

[1] Indeed, Subset Sum is the special case of $1||\sum p_jU_j$ where all jobs share the same deadline d. In other words, Subset Sum is the $1|d_j = d|\sum p_jU_j$ problem.

[2] In the so-called $1||\sum w_jU_j$ problem each job j is rewarded by a specified pay w_j (instead of p_j). For this generalization the running time $O(nP)$ was proven to be conditionally optimal [18, 30], in the sense that an algorithm in time $O((n+P)^{2-\epsilon})$, for any $\epsilon > 0$, contradicts the well-established $(\min, +)$-Convolution hypothesis. See also the discussion in [12].

[3] We write $\widetilde{O}(T) = T(\log T)^{O(1)}$ to suppress polylogarithmic factors.

[4] Given length-N integer vectors A, B, the Skewed Convolution problem is to compute the length-$(2N-1)$ vector C defined by $C[k] = \min_{i+j=k} \max\{A[i], B[j] - i\}$.

▶ **Theorem 1.** *The* $1||\sum p_jU_j$ *problem can be solved in randomized time* $O(P \log P)$ *and in deterministic time* $O(P \log^{1+o(1)} P)$.

We stress that by the aforementioned lower bounds [1, 17] our new algorithm is *optimal*, up to lower-order factors, conditioned on the Strong Exponential Time Hypothesis or the Set Cover Hypothesis.

As an additional feature, and similar to all previous algorithms, our algorithm not only computes the optimal value of the given instance, but in fact reports for each value $0 \leq x \leq P$ whether there is a feasible schedule with processing time (i.e., pay) x. Moreover, we can compute an optimal schedule (represented as an ordered subset of jobs) in the same running time.

Another benefit of our work is that we managed to distill an astonishingly *simple* algorithm. In fact, our algorithm is basically identical to the original Lawler-Moore algorithm, except that we replace certain naive computations by an appropriate efficient data structure on *strings*, and employ a careful new analysis. This approach is inspired by the recent progress on the Modular Subset Sum problem [5, 4, 14, 34] (see Section 3 for more details). We find it surprising that these conceptually simple ideas lead to near-optimal running times for $1||\sum p_jU_j$.

In particular, in contrast to previous improvements for $1||\sum p_jU_j$ [12, 28, 36], our algorithm is purely combinatorial and does not require the use of the Fast Fourier Transform. Given this simple nature of our algorithm, we are confident that actual implementations of the algorithm would perform well.

Multiple Machines. The "$P_m||\sum p_jU_j$" problem is the straightforward generalization of the $1||\sum p_jU_j$ problem to m workers (or machines) that can partition the jobs arbitrarily among themselves. The goal, as before, is to maximize the total workload across all workers while respecting all due dates. We assume for simplicity that m is a constant.[5]

The Lawler-Moore algorithm generalizes in a straightforward manner to time $O(nP^m)$. For the algorithms based on Skewed Convolution, it seems significantly harder to derive multiple-machine generalizations. Luckily, with some appropriate changes our new algorithm also generalizes to multiple machines:

▶ **Theorem 2.** *The* $P_m||\sum p_jU_j$ *problem can be solved in randomized time* $O(P^m \log P)$ *and in deterministic time* $O(P^m \log^{1+o(1)} P)$.

In particular, Theorem 2 outperforms the Lawler-Moore algorithm by a near-linear factor $\widetilde{\Omega}(n)$. In contrast to the single-machine setting, however, we emphasize that this algorithm is not necessarily optimal. A conditional lower bound for this problem would, most likely, be derived from an analogous lower bound for the multiple-target Subset Sum problem [3]. This appears to be a challenging question which is not resolved yet.

Alternative Parameters. So far we have only mentioned the parameters n and $P = \sum_j p_j$, which have been the main focus in previous work. But there are many other parameters worth considering. Natural candidates include the number of distinct deadlines ($D_\#$), the sum of all distinct deadlines (D), the largest processing time ($p_{\max} = \max_j p_j$) and the largest deadline ($d_{\max} = \max_j d_j$).

[5] When viewing m as an input, it is easy to trace that our algorithms depend only polynomially on m.

There has been research on developing nontrivial $1||\sum p_jU_j$ algorithms (for a single machine) with respect to these parameters, such as an $\widetilde{O}(\min\{P\cdot D_{\#}, P+D\})$-time algorithm due to [12], and an $\widetilde{O}(n+p_{\max}^3)$-time algorithm due to [28]. We remark that the former is subsumed by our new results. The latter algorithm is incomparable to our result (specifically, our algorithm in time $\widetilde{O}(P)=\widetilde{O}(np_{\max})$ performs better if and only if $p_{\max}\gg n^{1/2}$). Both of these results [12, 28] generalize to m machines as well, leading to similar comparisons with our work.

It remains an interesting open question whether our $\widetilde{O}(P)$-time algorithm can be further improved with respect to the parameters $d_{\max}$ and $p_{\max}$. In principle it seems reasonable that time $\widetilde{O}(n+d_{\max})$ can be achieved, as the analogous question for Subset Sum is resolved [8, 26]. We leave this as an open question. An even more exciting question is whether we could possibly achieve time $\widetilde{O}(n+p_{\max})$. However, such an algorithm would entail a break-through for Subset Sum with small items, which currently seems out of reach.

Further Related Work. This work is part of a bigger effort of the fine-grained complexity community to design best-possible pseudo-polynomial time algorithms for a host of optimization problems. This line of research encompasses, besides the aforementioned scheduling problems [12, 23, 28, 36], various variants of Subset Sum [29, 8, 1, 5, 6, 13, 33, 4, 14, 34, 19], Knapsack [38, 18, 30, 7, 20, 33, 11, 16, 10, 9, 25], Integer Programming [20, 24] and many others [15, 19].

2 Preliminaries

Throughout, we write $[n]=\{0,\dots,n-1\}$ and use the interval notation $[i\,..\,j]=\{i,\dots,j\}$, and similarly $[i\,..\,j)$, $(i\,..\,j]$, $(i\,..\,j)$. For two sets of integers S,T and an integer t we employ the sumset notation $S+t=\{s+t : s\in S\}$ and $S+T=\{s+t : s\in S, t\in T\}$.

Scheduling Problems. We begin with a formal definition of the $1||\sum p_jU_j$ problem. The input consists of n *jobs*, where each job $j\in[n]$ has a *processing time* $p_j\in\mathbb{N}_{>0}$ and *due dates* $d_j\in\mathbb{N}_{>0}$. A schedule is a permutation $\sigma:[n]\to[n]$. The *completion time* C_j of a job j in the schedule σ is $C_j=\sum_{i:\sigma(i)\le\sigma(j)}p_j$ (i.e., the total processing time of all jobs preceding j, including j itself). We say that j is *early* if $C_j\le d_j$ and *tardy* otherwise, and let $U_j\in\{0,1\}$ be the indicator variable indicating whether j is tardy. In this notation, our objective is to find a schedule minimizing $\sum_j p_jU_j$ (i.e., the total processing time of all tardy jobs). This explains the description $1||\sum p_jU_j$ in three-field notation.[6] For convenience we have defined the problem in such a way that $p_j>0$, and as a consequence we can always bound $n\le P$.[7]

For the m-machine problem $P_m||\sum p_jU_j$ a schedule is analogously defined as a function $\sigma:[n]\to[n]\times[m]$, where the first coordinate determines the order of the jobs as before, and the second coordinate determines the machine which is supposed to execute the job. The completion time C_j is the total processing time of all jobs preceding j *on j's machine* (including j itself), and the objective of the problem remains unchanged. For simplicity, we assume throughout the paper that m is a constant (it can easily be verified that we only omit $m^{O(1)}$-factors this way).

[6] The 1 in the first field denotes a single machine, the empty second field symbolizes no additional constraints, and the third field gives the objective to minimize $\sum_j p_jU_j$.

[7] If jobs with processing time $p_j=0$ were permitted, all of our algorithms would additionally require $O(n)$ time preprocessing.

Earliest-Due-Date-First Schedules. A key observation about $1||\sum p_jU_j$ dating back to Lawler and Moore [31] is that, without loss of generality, the early jobs are scheduled in increasing order of their due dates. This observation is leveraged as follows: We reorder the jobs such that $d_0 \leq \cdots \leq d_{n-1}$ (we will stick to this ordering for the rest of the paper). Thus, the $1||\sum p_jU_j$ problem is effectively to compute a subset of jobs $J \subseteq [n]$ that maximizes $\sum_{j\in J} p_j$ and is *feasible* in the sense that all jobs in J are early (i.e., $C_j = \sum_{i\in J:i\leq j} p_i \leq d_j$ for all $j \in J$).

Machine Model. We work in the standard Word RAM model with word size $\Theta(\log n + \log P)$ (such that each job can be stored in a constant number of cells). Moreover, all randomized algorithms mentioned throughout are Las Vegas (i.e., zero-error) algorithms running in their claimed time bounds with high probability $1 - 1/n^{\Omega(1)}$.

3 Near-Optimal Algorithm for a Single Machine

In this section, we give the details of our near-optimal algorithm for $1||\sum p_jU_j$. We start with a brief summary of the Lawler-Moore algorithm.

Lawler and Moore's Baseline. The Lawler-Moore algorithm [31] is the natural dynamic programming solution for the $1||\sum p_jU_j$ problem. We present it here by recursively defining the following sets $S_0, \ldots, S_n \subseteq [0\,..\,P]$:

$$\begin{aligned} S_0 &= \{0\}, \\ S'_{j+1} &= S_j + \{0, p_j\} \quad (j \in [n]), \\ S_{j+1} &= S'_{j+1} \cap [0, d_j] \quad (j \in [n]). \end{aligned}$$

(The construction of S_{j+1} is divided into two steps as this will be convenient later on.) Each set S_{j+1} can naively be computed from S_j in time $O(P)$, and thus all sets $S_0, \ldots, S_n$ can be naively computed in time $O(nP)$. We can ultimately read off the optimal value as $\max S_n$, based on the following observation:

▶ **Observation 3** (Lawler and Moore [31]). *There is a feasible schedule of total processing time t if and only if $t \in S_n$.*

More generally, S_{j+1} is the set of processing times of feasible schedules involving the jobs $0, \ldots, j$. To see this, consider any feasible schedule of the jobs $0, \ldots, j-1$ (whose processing time is in S_j). We can either leave out the next job j or append to the schedule. Thereby, the set of processing times becomes $S'_{j+1} = S_j + \{0, p_j\} = \{s, s + p_j : s \in S'_{j+1}\}$. However, this appended schedule is not necessarily feasible as it might not comply with the due date d_j. Hence, all processing times greater than d_j are deleted again in the construction of S_j.

Our Approach. Perhaps surprisingly, our algorithm essentially follows the same approach, i.e., our goal remains to compute the sets $S_0, \ldots, S_n$. However, we will demonstrate that the naive $O(P)$-time computation of each step can be significantly sped up. Our algorithm relies on two ingredients – an algorithmic and a structural one.

Ingredient 1: An Efficient Data Structure. As the sets S'_j and S_j are constructed in a highly structured way, can we compute them faster than time $O(P)$? Specifically, is there a way to (i) compute each set S'_{j+1} in time proportional to the number of *insertions* $|S'_{j+1} \setminus S_j|$, and to (ii) compute S_{j+1} in time proportional to the number of *deletions* $|S'_{j+1} \setminus S_{j+1}|$?

Question (i) is closely related to the Subset Sum problem, and has been successfully resolved in [5] leading to near-optimal algorithms for *Modular* Subset Sum. Their solution based on linear sketching is quite involved [5], but two independent papers [4, 14] provided a significantly simpler proof by replacing linear sketching with a reduction to a dynamic string problem; see also [34][8]. Regarding (ii), it turns out that we can adapt this reduction to the dynamic string problem to efficiently accommodate our deletions. The following lemma summarizes the resulting data structure; we defer its proof to Section 3.1.

▶ **Lemma 4** (Sum-Cap Data Structure). *There is a randomized data structure that maintains a set $S \subseteq [u]$ and supports the following operations:*

- `init`(S)*:* *Initializes the data structure to the given set $S \subseteq [u]$.*
  Runs in time $O(|S| \cdot \log u + \log^2 u)$.
- `query`(s)*:* *Given $s \in [u]$, tests whether $s \in S$.*
  Runs in time $O(\log u)$.
- `sum`(p)*:* *Given $p \in [u]$, updates $S \leftarrow S + \{0, p\}$.*
  Runs in time $O(|(S+p) \setminus S| \cdot \log u)$ (where S is as before the operation).
- `cap`(d)*:* *Given $d \in [u]$, updates $S \leftarrow S \cap [d]$.*
  Runs in time $O(\log u)$.

If at any point during the execution an element $s \notin [u]$ is attempted to be inserted, the data structure becomes invalid. Moreover, the data structure can be made deterministic at the cost of worsening all operations by a factor $\log^{o(1)} u$.

Ingredient 2: A Structural Insight. What have we gained by computing the sets S_j and S'_j with the data structure from Lemma 4? Due to the particularly efficient `cap` operation, the computation of the sets S_j is essentially for free. Computing the sets S'_j using the `sum` operation, however, amounts to time

$$\widetilde{O}\left(\sum_{j\in[n]} |S'_{j+1} \setminus S_j|\right).$$

A priori, it is not clear whether this is helpful. In case of only *inserting* elements, this sum could be conveniently bounded by P (as is the case for Modular Subset Sum). Unfortunately, we additionally have to deal with *deletions*. Specifically, it is possible that some element s is inserted in S'_1, deleted in S_1, inserted again in S'_2, and so on. All in all, s could be inserted up to n times, and so the only immediate upper bound for the sum is nP (which recovers the Lawler-Moore running time).

Our crucial structural insight is that, while the same element can indeed be inserted and deleted multiple times, the *total number of insertions* is nevertheless bounded. More precisely, we show that the overall number of insertions is at most $O(P)$:

[8] In [34], Potepa proposes an improved deterministic data structure with applications to the Modular Subset Sum problem. A priori, it looks like their improvement might similarly apply to our setting. Unfortunately, the data structure is only efficient if we have the freedom to arbitrarily reorder the items, which is prohibitive for us as we have to stick to the order $d_0 \leq \cdots \leq d_{n-1}$.

▶ **Lemma 5** (Bounded Insertions). *It holds that* $\sum_{j\in[n]} |S'_{j+1} \setminus S_j| \leq 2P+1$.

Proof. We split the sum into two parts:

$$\sum_{j\in[n]} \big|(S'_{j+1} \setminus S_j)\big| = \sum_{j\in[n]} \big|(S'_{j+1} \setminus S_j) \cap [0\,..\,d_j]\big| + \sum_{j\in[n]} \big|(S'_{j+1} \setminus S_j) \cap (d_j\,..\,P]\big|.$$

Intuitively, the first sum counts the number of elements that are irreversibly inserted into $S_{j+1}, \dots, S_n$ in the j-th step. The second sum counts the number of elements that are inserted into S'_{j+1} and immediately deleted in S_{j+1}.

For the first sum, consider the following observation: For any $x \in [0\,..\,P]$, if $x \leq d_j$ and $x \in S'_{j+1} \setminus S_j$, then $x \in S_{j+1}, \dots, S_n$ (since $d_j \leq d_{j+1}, \dots, d_{n-1}$). It follows that

$$\big|\{j \in [n] : x \in (S'_{j+1} \setminus S_j) \cap [0\,..\,d_j]\}\big| \leq 1$$

for all $x \in [0\,..\,P]$. Thus,

$$\sum_{j\in[n]} \big|(S'_{j+1} \setminus S_j) \cap [0\,..\,d_j]\big| = \sum_{x\in[0\,..\,P]} \big|\{j \in [n] : x \in (S'_{j+1} \setminus S_j) \cap [0\,..\,d_j]\}\big| \leq P+1.$$

Second, we bound $|(S'_{j+1} \setminus S_j) \cap (d_j\,..\,P]| \leq |S'_{j+1} \cap (d_j\,..\,P]|$. Note that $S_j \subseteq [0\,..\,d_{j-1}]$ and therefore $S'_{j+1} = S_j + \{0, p_j\} \subseteq [0\,..\,d_{j-1} + p_j]$. Consequently,

$$\big|S'_j \cap (d_j\,..\,P]\big| \leq \big|[0\,..\,d_{j-1} + p_j] \cap (d_j\,..\,P]\big| \leq d_{j-1} + p_j - d_j \leq p_j,$$

where the final inequality follows from the assumption that $d_{j-1} \leq d_j$. Hence, the number of overall deletions is

$$\sum_{j\in[n]} \big|(S'_{j+1} \setminus S_j) \cap (d_j\,..\,P]\big| \leq \sum_{j\in[n]} p_j = P.$$

Combining both parts concludes the proof. ◀

The proof for Lemma 4 is provided in Section 3.1. Using Lemma 5 and 4, we are in the position to show our main result.

▶ **Theorem 1.** *The* $1||\sum p_j U_j$ *problem can be solved in randomized time* $O(P \log P)$ *and in deterministic time* $O(P \log^{1+o(1)} P)$.

Proof. In summary, our algorithm works as follows. We compute $S_0, \dots, S_n \subseteq [0\,..\,P]$ using the data structure from Lemma 4 (with $u = P+1$). Specifically, after initializing S with $\texttt{init}(S_0)$, we repeatedly construct the sets S'_j and S_j using the operations $\texttt{sum}(p_j)$ and $\texttt{cap}(d_j)$ for all $j \leftarrow 0, \dots, n-1$. The largest element in the final set $S = S_n$ is the maximal total processing of a feasible schedule of all jobs $0, \dots, n-1$. Finding and returning it is the last step of our algorithm, by repeatedly using the $\texttt{query}(i)$ operation and returning the largest index i for which the query returns yes. The correctness of our algorithm follows from Observation 3.

The running time is composed of the following parts: The initialization runs in time $O(\log^2 P)$, the repeated use of `sum` and `cap` takes time $O(\sum_{j\in[n]}(|S'_{j+1} \setminus S_j| \cdot \log P + \log P))$ and the optimal value is found in time $O(P)$. Using $n \leq P$ and Lemma 5, it holds that

$$O\left(\sum_{j\in[n]} (|S'_{j+1} \setminus S_j| \cdot \log P + \log P)\right) \leq O(P \log P).$$

In total, we have a randomized running time of $O(P \log P)$. Applying the same arguments yields the deterministic running time of $O(P \log^{1+o(1)} P)$. ◀

We stress that the algorithm described in this section only computes the optimal *value*. In Section 3.2, we explain how our algorithm can be easily extended to obtain the optimal *schedule* as well.

3.1 Cap-Sum Data Structure via Dynamic Strings

In this section, we provide the missing proof of Lemma 4 by a reduction to the *dynamic strings* data structure problem. This is the fundamental problem of maintaining a collection of strings that can be concatenated, split, updated, and tested for equality – see [37, 35, 32, 2, 21]. We summarize the state of the art in the following lemma; the fastest randomized (and in fact, optimal) data structure is due to Gawrychowski, Karczmarz, Kociumaka, Lacki and Sankowski [21], and for the fastest deterministic one see [27, Section 8].

Here we use standard string notation for a string x, where $x[i]$ denotes the letter at index i, and $x[i\,..\,j], x[i\,..\,j)$ denote the appropriate substrings.

▶ **Lemma 6** (Dynamic String Data Structure [21, 27]). *There is a data structure that maintains a dynamic collection X of non-empty strings and support the following operations:*

- `make_string`(x)*: Given any string $x \in \Sigma^+$, inserts x into X.*
- `concat`(x_1, x_2)*: Given $x_1, x_2 \in X$, inserts the concatenation x_1x_2 into X.*
- `split`(x, i)*: Given $x \in X$ and $i \in [0\,..\,|x|)$, inserts $x[0\,..\,i]$ and $x(i\,..\,|x|)$ into X.*
- `LCP`(x_1, x_2)*: Given $x_1, x_2 \in X$, returns the length ℓ of their longest common prefix, i.e., returns $\max\{0 \le \ell \le \min\{|x_1|, |x_2|\} : x_1[0\,..\,\ell) = x_2[0\,..\,\ell)\}$.*
- `query`(x, i)*: Given $x \in X$ and $i \in [|x|]$, returns $x[i]$.*

Let n be the maximum of the total length of all strings and the number of executed operations. Then all operations run in randomized time $O(\log n)$ or in deterministic time $O(\log^{1+o(1)} n)$, except for `make_string` *which takes time $O(|x| + \log n)$ and $O(|x| \cdot \log^{o(1)} n)$, respectively.*

For the sake of convenience, we include two more dynamic string operations that are derived from the previous lemma in a black-box fashion. As both are standard operations [21], we only provide their implementations for completeness.

- `update`(x, i, σ): Given $x \in X$, an index $i \in [|x|]$ and $\sigma \in \Sigma$, inserts the string obtained from x by changing the i-th character to σ into the data structure. To implement this, we split the string x twice to separate the letter $x[i]$ from the rest of the string. Specifically, we obtain the substring $x[0\,..\,i]$ using `split`(x, i) and further divide it to get the substring $x[0\,..\,i)$ by `split`$(x[0\,..\,i), i-1)$. Next, the `make_string`(σ) operation creates the string σ. Lastly, we use `concat`$(x[0\,..\,i), \texttt{concat}(\sigma, x(i\,..\,|x|)))$ to reinsert σ between the two substrings.
- `LCE`(x_1, x_2, i_1, i_2): Given $x_1, x_2 \in X$ and $i_1 \in [|x_1|], i_2 \in [|x_2|]$, returns the longest common extension $\max\{0 \le \ell \le \min\{|x_1| - i_1, |x_2| - i_2\} : x_1[i_1\,..\,i_1 + \ell) = x_2[i_2\,..\,i_2 + \ell)\}$. To implement this, using the two operations `split`$(x_1, i-1)$ and `split`$(x_2, j-1)$ we first separate the substrings $x_1[i\,..\,|x_1|)$ and $x_2[j\,..\,|x_2|)$. Observe that the length of the longest common extension of the original strings is exactly the length of the longest common prefix of $x_1[i\,..\,|x_1|)$ and $x_2[j\,..\,|x_2|)$ returned by the operation `LCP`$(x_1[i\,..\,|x_1|), x_2[j\,..\,|x_2|))$.

Both `update` and `LCE` require a constant number of original operations that run in randomized time $O(\log n)$, or deterministic time $O(\log^{1+o(1)} n)$.

Now, we are in the position to provide the proof of Lemma 4. Recall that this proof is in parts borrowed from [4, 14].

Proof of Lemma 4. We maintain the set $S \subseteq [u]$ as an indicator string $x_S \in \{0,1\}^u$ such that $i \in S$ if and only if $x_S[i] = 1$.

- $\texttt{init}(S)$: Using repeated squaring, we construct the string 0^u by inserting $w = 0$ and concatenating w with itself $\log u$ times. Note that 0^u will remain in the data structure and can be used by other operations. We compute x_S by updating 0^u using $\texttt{update}(x_S, S[i], 1)$ for all indices $i \in S$. Since the repeated squaring takes time $O(\log^2 u)$ and updating the elements takes time $O(|S| \cdot \log u)$, the `init` operation runs in time $O(\log^2 u + |S| \cdot \log u)$.
- $\texttt{query}(i)$: As the original data structure already provides a query operation, we use $\texttt{query}(x_S, i)$ that returns $x_S[i]$ in time $O(\log u)$.
- $\texttt{sum}(p)$: We implement `sum` in three steps. First, we will compute the string x_{S+p} that represents the set $S + p$. Observe that x_{S+p} is obtained by shifting x_S by p positions to the right. Thus, we extend x_S using $\texttt{concat}(0^p, x_S)$ where the string 0^p is split off the precomputed string 0^u using $\texttt{split}(0^u, p)$. Then we trim it down to length u with $\texttt{split}(0^p x_S, u)$.

  Second, note that the desired string $x_{S\cup(S+p)}$ is the result of the bit-wise OR of x_S and x_{S+p}. We compute the set $D = \{i \in [u] : x_S[i] \neq x_{S+p}[i]\}$ that contains all indices at which x_S and x_{S+p} differ from each other. To this end, starting with $i \leftarrow 0$, we will repeat the following process as long as $i < u$: Compute $\ell \leftarrow \texttt{LCE}(x_S, x_{S+p}, i, i)$ to determine the next index ℓ at which both strings differ, insert $D \leftarrow D \cup \{i + \ell\}$ and update $i \leftarrow i + \ell + 1$.

  As the third and last step, we compute $x_{S\cup(S+p)}$ by updating x_S using $\texttt{update}(x_S, i, 1)$ for all indices $i \in D$.

  Creating the shifted string x_{S+p} takes time $O(\log u)$. Both the construction of set D and computing the string $x_{S\cup(S+p)}$ require $|(S+p) \setminus S| + |S \setminus (S+p)|$ many operations that each run in time $O(\log u)$. Since $|S + p| = |S|$, we have

  $$|(S+p) \setminus S| = |S+p| - |(S+p) \cap S| = |S| - |(S+p) \cap S| = |S \setminus (S+p)|$$

  and therefore $|(S+p) \setminus S| + |S \setminus (S+p)| = 2 \cdot |(S+p) \setminus S|$. In summary, the `sum` operation takes time $O(|(S+p) \setminus S| \cdot \log u)$.
- $\texttt{cap}(d)$: In order to set $x_S[i] = 0$ for all $i \in (d\,..\,u)$, we separate the substring $x_S[0\,..\,d]$ with $\texttt{split}(x_S, d)$ and split the substring 0^{u-d+1} off the precomputed string 0^u using $\texttt{split}(0^u, u - d + 1)$. Then x_S is assembled using $\texttt{concat}(x_S[0\,..\,d], 0^{u-d+1})$. All three operations take time $O(\log u)$.

Following Lemma 6, the deterministic running times can be obtained by worsening all operations by a factor $\log^{o(1)} u$. ◀

3.2 Obtaining an Optimal Schedule

In the previous sections we have argued that the optimal value OPT (i.e., the maximum total processing of a feasible schedule) can be computed in near-linear time $\widetilde{O}(P)$. In this section we explain how the actual optimal schedule can be computed by a deterministic post-processing routine in time $O(n)$.

The idea is, as is standard for dynamic programming algorithms, to trace through the sets $S_0, \ldots, S_n$ in reverse order. Making this traversal efficient, we slightly modify our algorithm to additionally compute an array $A[0\,..\,P]$ such that $A[s] = \min\{j \in [n] : s \in S_{j+1}\}$. Intuitively, $A[s]$ stores the smallest job j such that there exists a feasible schedule with total processing time s that contains j. Updating A appropriately whenever an element is inserted into S'_{j+1} allows us to easily maintain the array without worsening the asymptotic running time. Then, in order to compute an optimal schedule, we apply the following

algorithm: We initialize $J \leftarrow \emptyset$ and $s \leftarrow \mathrm{OPT}$. We repeatedly retrieve the next job $j \leftarrow A[s]$ and update $J \leftarrow J \cup \{j\}$ and $s \leftarrow s - p_j$, until $s = 0$. In each step, we identify a job j that is contained in the optimal schedule, and thus J is an optimal schedule once the process has terminated. In fact, the same idea can be used to retrieve a feasible schedule for *any* given processing time $s \in S_n$.

4 Generalization to Multiple Machines

In this section, we show that our algorithm for $1||\sum p_j U_j$ can be extended to $P_m||\sum p_j U_j$. Since it follows the same approach as the single machine algorithm, we will keep this section short and concise. For more details refer to Section 3.

Let $e_0, \dots, e_{m-1}$ denote the standard unit vectors of $\mathbb{Z}^m$, then we recursively define the sets $S_0, \dots, S_n \subseteq [0, P]^m$ as follows:

$$\begin{aligned} S_0 &= \{0\}, \\ S'_{j+1} &= S_j + \{0, p_j \cdot e_0, \dots, p_j \cdot e_{m-1}\} && (j \in [n]), \\ S_{j+1} &= S'_{j+1} \cap [0, d_j]^m && (j \in [n]). \end{aligned}$$

As before, the optimal value is the maximum entry in S_n:

▶ **Observation 7** (Lawler and Moore [31]). *There is a feasible schedule of total processing time t if and only if there is some $s \in S_n$ with $s_0 + \dots + s_{m-1} = t$.*

The crucial difference to before is that here all $s \in S_j$ are vectors where their i-th entry corresponds to the i-th machine. Because each job is either scheduled on exactly one machine or not at all, we consider all scheduling possibilities of job j with $S_j + \{0, p_j \cdot e_0, \dots, p_j \cdot e_{m-1}\}$. As our goal is again to bound the total number of insertions, see the following lemma:

▶ **Lemma 8** (Generalized Bounded Insertions). *It holds that*

$$\sum_{j \in [n]} \left| S'_{j+1} \setminus S_j \right| \leq (m+1) \cdot (P+1)^m.$$

Proof. In the following, we consider two parts of the sum separately:

$$\sum_{j \in [n]} \left| (S'_{j+1} \setminus S_j) \right| = \sum_{j \in [n]} \left| (S'_{j+1} \setminus S_j) \cap [0\,..\,d_{j+1}]^m \right| + \sum_{j \in [n]} \left| (S'_{j+1} \setminus S_j) \setminus [0\,..\,d_{j+1}]^m \right|.$$

In other words, in analogy to Lemma 5, we first count the number of elements that are irrevocably inserted into $S_{j+1}, \dots, S_n$ in the j-th step. Second, we count the number of elements that are inserted into S'_{j+1} and instantly deleted in S_{j+1}.

We bound the first sum with the following observation. For any $x \in [0\,..\,P]^m$, it holds that if $x \in (S'_{j+1} \setminus S_j)$ and $x \in [0\,..\,d_j]$, then $x \in S_{j+1}, \dots, S_n$. This follows directly from the assumption that $d_j \leq d_{j+1}, \dots, d_{n-1}$. Therefore, it holds that

$$\left| \{ j \in [n] : x \in (S'_{j+1} \setminus S_j) \cap [0\,..\,d_j]^m \} \right| \leq 1,$$

for all $x \in [0\,..\,P]^m$, and thus

$$\sum_{j \in [n]} \left| (S'_{j+1} \setminus S_j) \cap [0\,..\,d_j]^m \right| = \sum_{x \in [0\,..\,P]^m} \left| \{ j \in [n] : x \in (S'_{j+1} \setminus S_j) \cap [0\,..\,d_j]^m \} \right| \leq (P+1)^m.$$

For the second sum, we bound $|(S'_{j+1} \setminus S_j) \setminus [0\,..\,d_j]^m| \leq |S'_{j+1} \setminus [0\,..\,d_j]^m|$. Using that $S_j \subseteq [0\,..\,d_{j-1}]^m$, we have that

$$\begin{aligned} S'_{j+1} &= S_j + \{0, p_j \cdot e_0, \dots, p_j \cdot e_{m-1}\} \\ &\subseteq [0\,..\,d_{j-1}]^m + \{0, p_j \cdot e_0, \dots, p_j \cdot e_{m-1}\} =: V_{j+1}. \end{aligned}$$

As each job j is scheduled on exactly one machine, we observe that V_{j+1} is the set of vectors where all entries are in $[0\,..\,d_{j-1}]$, except for possibly one entry that is in $[0\,..\,d_{j-1} + p_j]$. Hence, $V_{j+1} \setminus [0\,..\,d_j]^m$ is the set of vectors where all entries are in $[0\,..\,d_{j-1}]$, except for exactly one entry that is in $(d_j\,..\,d_{j-1} + p_j]$. Next, we bound the size of $V_{j+1} \setminus [0\,..\,d_j]^m$: There are m options for the index of the special entry, there are $d_{j-1} + p_j - d_j$ options for the value of the special entry, and finally there are $(P+1)^{m-1}$ options for the other $m-1$ entries. Thus,

$$\begin{aligned} \left|S'_{j+1} \setminus [0\,..\,d_j]^m\right| &\leq \left|V_{j+1} \setminus [0\,..\,d_j]^m\right| \\ &\leq m \cdot (P+1)^{m-1} \cdot (d_{j-1} + p_j - d_j) \\ &\leq m \cdot (P+1)^{m-1} \cdot p_j, \end{aligned}$$

where the final inequality follows from the assumption that $d_{j-1} \leq d_j$. Consequently, the second sum is bounded by

$$\sum_{j \in [n]} \left|S'_{j+1} \setminus [0\,..\,d_j]^m\right| \leq m \cdot (P+1)^{m-1} \cdot \sum_{j \in [n]} p_j \leq m \cdot (P+1)^m.$$

Combining the bounds for both sums yields the overall bound. ◀

Analogous to Section 3, we use the generalized sum-cap data structure to efficiently maintain the generalized sets $S_0, \dots, S_n$.

▶ **Lemma 9** (Generalized Sum-Cap Data Structure). *There is a randomized data structure maintaining a set $S \subseteq [u]^m$ that supports the following operations:*

- `init`(S): *Initializes the data structure to the given set $S \subseteq [u]^m$.*
  Runs in time $O(\log^2 u + |S| \cdot \log u)$.
- `query`(s): *Given $s \in [u]^m$, tests whether $s \in S$.*
  Runs in time $O(\log u)$.
- `sum`(T): *Given $T \subseteq [u]^m$, updates $S \leftarrow S + T$.*
  Runs in time $O(|T| \cdot |(S+T) \setminus S| \cdot \log u)$ (where S is as before the operation).
- `cap`(d): *Given $d \in [u]$, updates $S \leftarrow S \cap [d]^m$.*
  Runs in time $O(u^{m-1} \cdot \log u)$.

If at any point during the execution an element $s \notin [u]^m$ is attempted to be inserted, the data structure becomes invalid. Moreover, the data structure can be made deterministic at the cost of worsening all operations by a factor $\log^{o(1)} u$.

Proof. Let $U = u^m$. Let $\phi : [u]^m \to [U]$ be the bijection defined by $\phi(s) = \sum_{i \in [m]} s_i u^i$. We extend the definition to sets $S \subseteq [u]^m$ via $\phi(S) = \{\phi(s) : s \in S\}$. We maintain the set $S \subseteq [u]^m$ as the indicator string of $\phi(S)$, namely $x_{\phi(S)} \in \{0,1\}^U$, such that $i \in S$ if and only if $x_{\phi(S)}[i] = 1$. In other words, we store for each $s \in [u]^m$, listed in lexicographical order, whether $s \in S$.

- `init`(S): The string 0^U is constructed using repeated squaring by inserting $x = 0$ and concatenating x with itself $m \log u$ times. It will remain in the data structure available to other operations. Repeatedly using `update`$(\cdot, \phi(s), 1)$ for all entries $s \in S$, we update the string 0^U to obtain $x_{\phi(S)}$. As repeated squaring takes time $O(\log^2 u)$ and updating the elements takes time $O(|S| \cdot \log u)$, the `init` operation runs in time $O(\log^2 u + |S| \cdot \log u)$.

- $\texttt{query}(s)$: Using $\texttt{query}(x_{\phi(S)}, s)$ from the string data structure allows us to return $x_{\phi(S)}[s]$ in time $O(\log u)$.
- $\texttt{sum}(T)$: We can assume without loss of generality that $0 \in T$ (as otherwise we can simply shift T and $x_{\phi(S)}$ appropriately such that the smallest element in T becomes 0). Fix an arbitrary nonzero element $p \in T$. Analogously to Lemma 4, we first show how to compute the set $(S+p) \setminus S$ in output-sensitive time. The string $x_{\phi(S+p)}$ representing the set $S+p$ can be computed using the two following facts. If $x, y, x+y \in [u]^m$, then it holds that $\phi(x+y) = \phi(x) + \phi(y)$. Further, if $p \in [u]^m$ and $S, S+p \subseteq [u]^m$, then $\phi(S+p) = \phi(S) + \phi(p)$. Therefore, $x_{\phi(S+p)}$ is $x_{\phi(S)}$ up to a shift of $\phi(p)$, and can be obtained by $\texttt{split}(\texttt{concat}(0^{\phi(p)}, x_{\phi(S)}), U)$.

  Similarly to Lemma 4, repeatedly using $\texttt{LCE}$ queries allows us to first compute the set $\{i \subseteq [U] : x_{\phi(S)}[i] \neq x_{\phi(S+p)}[i]\}$ and then read off the symmetric difference of S and $S+p$, denoted by $D_p = (S \setminus (S+p)) \cup ((S+p) \setminus S)$. We repeat the same for all other nonzero elements $p \in T$. Let $D = \bigcup_{p \in T \setminus \{0\}} D_p$, then we have $D \supseteq (S+T) \setminus S$. Thus, we can update the indicator string $x_{\phi(S)}$ by calling $\texttt{update}(\cdot, \phi(s), 1)$ for all $s \in D$.

  In order to bound the running time, we bound the size of the sets D_p and D. Using that $|S \setminus (S+p)| = |(S+p) \setminus S|$, we have that $|D_p| \leq 2|(S+p) \setminus S| \leq 2|(S+T) \setminus S|$, and therefore $|D| \leq 2|T| \cdot |(S+T) \setminus S|$. In total, we used $O(|D|)$ data structure operations, leading to a running time of $O(|T| \cdot |(S+T) \setminus S| \cdot \log u)$.
- $\texttt{cap}(d)$: We delete all vectors $s \in S$ with at least one entry that is larger than d as follows. We enumerate all $(m-1)$-tuples $(s_1, \ldots, s_{m-1}) \in [u]^{m-1}$. Recall that the vectors are stored in lexicographical order. Thus, the set of vectors $(s_0, s_1, \ldots, s_{m-1})$ where s_0 ranges over $[u]$ and $s_1, \ldots, s_{m-1}$ are fixed, is represented by a length-u substring of $x_{\phi(S)}$. Specifically, the substring $x_{\phi(S)}[\phi(0, s_1, \ldots, s_{m-1}) .. \phi(u-1, s_1, \ldots, s_{m-1})]$. We distinguish two cases: If $\max_{i=1}^{m-1} s_i > d$, then the entire substring is replaced with 0^u. Otherwise, we retain its length-$(d+1)$ prefix and replace its suffix is by 0^{u-d+1}. It takes $O(u^{m-1})$ $\texttt{concat}$ and $\texttt{split}$ operations to perform these modifications, running in total time $O(u^{m-1} \cdot \log u)$.

Again, following Lemma 6 the deterministic running time of the operations differs by replacing $\log u$ with $\log^{1+o(1)} u$. ◀

Based on Lemma 8 and 9, we show the following generalization of our main result.

▶ **Theorem 2.** *The $P_m || \sum p_j U_j$ problem can be solved in randomized time $O(P^m \log P)$ and in deterministic time $O(P^m \log^{1+o(1)} P)$.*

Proof. Analogous to Theorem 1, we use the algorithm: We initialize the data structure from Lemma 8 (used with $u = P+1$) with $\texttt{init}(S_0)$. For all $j \leftarrow 0, \ldots, n-1$, we repeatedly use the operations $\texttt{sum}(\{0, p_j \cdot e_0, \ldots, p_j \cdot e_{m-1}\})$ and $\texttt{cap}(d_j)$ to compute the sets S'_{j+1} and S_{j+1}. We return the optimal value contained in S_n as described in Theorem 1. Our algorithm is correct due to Observation 7. Finally, using Lemmas 8 and 9 and the assumption $n \leq P$, we can bound the dominant term of the running time by

$$O\left(\sum_{j \in [n]} (m \cdot |S'_{j+1} \setminus S_j| \cdot \log P + P^{m-1} \log P)\right) = O(P^m \log P).$$

(Recall that m is constant.) Thus, we obtain a randomized running time of $O(P^m \log P)$, and similarly a deterministic running time of $O(P^m \log^{1+o(1)} P)$. ◀

References

1 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. Seth-based lower bounds for subset sum and bicriteria path. In Timothy M. Chan, editor, *30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 41–57. SIAM, 2019. doi:10.1137/1.9781611975482.3.

2 Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. Pattern matching in dynamic texts. In David B. Shmoys, editor, *11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2000)*, pages 819–828. ACM/SIAM, 2000. URL: http://dl.acm.org/citation.cfm?id=338219.338645.

3 Antonis Antonopoulos, Aris Pagourtzis, Stavros Petsalakis, and Manolis Vasilakis. Faster algorithms for k-subset sum and variations. *J. Comb. Optim.*, 45(1):24, 2023. doi:10.1007/S10878-022-00928-0.

4 Kyriakos Axiotis, Arturs Backurs, Karl Bringmann, Ce Jin, Vasileios Nakos, Christos Tzamos, and Hongxun Wu. Fast and simple modular subset sum. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms (SOSA 2021)*, pages 57–67. SIAM, 2021. doi:10.1137/1.9781611976496.6.

5 Kyriakos Axiotis, Arturs Backurs, Ce Jin, Christos Tzamos, and Hongxun Wu. Fast modular subset sum using linear sketching. In Timothy M. Chan, editor, *30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 58–69. SIAM, 2019. doi:10.1137/1.9781611975482.4.

6 Kyriakos Axiotis and Christos Tzamos. Capacitated dynamic programming: Faster knapsack and graph algorithms. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *LIPIcs*, pages 19:1–19:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ICALP.2019.19.

7 MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Saeed Seddighin, and Cliff Stein. Fast algorithms for knapsack via convolution and prediction. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2018)*, pages 1269–1282. ACM, 2018. doi:10.1145/3188745.3188876.

8 Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In Philip N. Klein, editor, *28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 1073–1084. SIAM, 2017. doi:10.1137/1.9781611974782.69.

9 Karl Bringmann. Knapsack with small items in near-quadratic time. *CoRR*, abs/2308.03075, 2023. doi:10.48550/arXiv.2308.03075.

10 Karl Bringmann and Alejandro Cassis. Faster knapsack algorithms via bounded monotone min-plus-convolution. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *LIPIcs*, pages 31:1–31:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICALP.2022.31.

11 Karl Bringmann and Alejandro Cassis. Faster 0-1-knapsack via near-convex min-plus-convolution. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms (ESA 2023)*, volume 274 of *LIPIcs*, pages 24:1–24:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.24.

12 Karl Bringmann, Nick Fischer, Danny Hermelin, Dvir Shabtay, and Philip Wellnitz. Faster minimization of tardy processing time on a single machine. *Algorithmica*, 84(5):1341–1356, 2022. doi:10.1007/S00453-022-00928-W.

13 Karl Bringmann and Philip Wellnitz. On near-linear-time algorithms for dense subset sum. In Dániel Marx, editor, *32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)*, pages 1777–1796. SIAM, 2021. doi:10.1137/1.9781611976465.107.

14 Jean Cardinal and John Iacono. Modular subset sum, dynamic strings, and zero-sum sets. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms (SOSA 2021)*, pages 45–56. SIAM, 2021. doi:10.1137/1.9781611976496.5.

15 Timothy M. Chan and Qizheng He. More on change-making and related problems. *J. Comput. Syst. Sci.*, 124:159–169, 2022. doi:10.1016/J.JCSS.2021.09.005.

16 Lin Chen, Jiayi Lian, Yuchen Mao, and Guochuan Zhang. Faster algorithms for bounded knapsack and bounded subset sum via fine-grained proximity results. In *35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2024)*, pages 4828–4848. SIAM, 2024. doi:10.1137/1.9781611977912.171.

17 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41:1–41:24, 2016. doi:10.1145/2925416.

18 Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. On problems equivalent to (min, +)-convolution. *ACM Trans. Algorithms*, 15(1):14:1–14:25, 2019. doi:10.1145/3293465.

19 Mingyang Deng, Xiao Mao, and Ziqian Zhong. On problems related to unbounded SubsetSum: A unified combinatorial approach. In Nikhil Bansal and Viswanath Nagarajan, editors, *32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)*, pages 2980–2990. SIAM, 2023. doi:10.1137/1.9781611977554.CH114.

20 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the steinitz lemma. *ACM Trans. Algorithms*, 16(1):5:1–5:14, 2020. doi:10.1145/3340322.

21 Pawel Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Lacki, and Piotr Sankowski. Optimal dynamic strings. In Artur Czumaj, editor, *29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*, pages 1509–1528. SIAM, 2018. doi:10.1137/1.9781611975031.99.

22 Ronald L. Graham, Eugene L. Lawler, Jan K. Lenstra, and Alexander H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979. doi:10.1016/S0167-5060(08)70356-X.

23 Danny Hermelin, Hendrik Molter, and Dvir Shabtay. Minimizing the weighted number of tardy jobs via (max, +)-convolutions. *CoRR*, abs/2202.06841, 2022. arXiv:2202.06841.

24 Klaus Jansen and Lars Rohwedder. On integer programming, discrepancy, and convolution. *Math. Oper. Res.*, 48(3):1481–1495, 2023. doi:10.1287/MOOR.2022.1308.

25 Ce Jin. 0-1 knapsack in nearly quadratic time. *CoRR*, abs/2308.04093, 2023. doi:10.48550/arXiv.2308.04093.

26 Ce Jin and Hongxun Wu. A simple near-linear pseudopolynomial time randomized algorithm for subset sum. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms (SOSA 2019)*, volume 69 of *OASIcs*, pages 17:1–17:6. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASICS.SOSA.2019.17.

27 Dominik Kempa and Tomasz Kociumaka. Dynamic suffix array with polylogarithmic queries and updates. In Stefano Leonardi and Anupam Gupta, editors, *54th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2022)*, pages 1657–1670. ACM, 2022. doi:10.1145/3519935.3520061.

28 Kim-Manuel Klein, Adam Polak, and Lars Rohwedder. On minimizing tardy processing time, max-min skewed convolution, and triangular structured ILPs. In Nikhil Bansal and Viswanath Nagarajan, editors, *34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2023)*, pages 2947–2960. SIAM, 2023. doi:10.1137/1.9781611977554.CH112.

29 Konstantinos Koiliaris and Chao Xu. Faster pseudopolynomial time algorithms for subset sum. *ACM Trans. Algorithms*, 15(3):40:1–40:20, 2019. doi:10.1145/3329863.

30 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.ICALP.2017.21.

31 Eugene L. Lawler and J. Michael Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16(1):77–84, 1969. doi:10.1287/mnsc.16.1.77.

32 Kurt Mehlhorn, R. Sundar, and Christian Uhrig. Maintaining dynamic sequences under equality tests in polylogarithmic time. *Algorithmica*, 17(2):183–198, 1997. doi:10.1007/BF02522825.

33 Adam Polak, Lars Rohwedder, and Karol Wegrzycki. Knapsack and subset sum with small items. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *LIPIcs*, pages 106:1–106:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.106.

34 Krzysztof Potepa. Faster deterministic modular subset sum. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, (ESA 2021)*, volume 204 of *LIPIcs*, pages 76:1–76:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ESA.2021.76.

35 William W. Pugh and Tim Teitelbaum. Incremental computation via function caching. In *16th Annual ACM Symposium on Principles of Programming Languages (POPL 1989)*, pages 315–328. ACM Press, 1989. doi:10.1145/75277.75305.

36 Baruch Schieber and Pranav Sitaraman. Quick minimization of tardy processing time on a single machine. In Pat Morin and Subhash Suri, editors, *18th International Symposium on Algorithms and Data Structures (WADS 2023)*, volume 14079 of *Lecture Notes in Computer Science*, pages 637–643. Springer, 2023. doi:10.1007/978-3-031-38906-1_42.

37 Rajamani Sundar and Robert Endre Tarjan. Unique binary-search-tree representations and equality testing of sets and sequences. *SIAM J. Comput.*, 23(1):24–44, 1994. doi:10.1137/S0097539790189733.

38 Arie Tamir. New pseudopolynomial complexity bounds for the bounded and other integer knapsack related problems. *Oper. Res. Lett.*, 37(5):303–306, 2009. doi:10.1016/J.ORL.2009.05.003.

Optimal Electrical Oblivious Routing on Expanders

Cella Florescu
ETH Zürich, Switzerland

Rasmus Kyng
ETH Zürich, Switzerland

Maximilian Probst Gutenberg
ETH Zürich, Switzerland

Sushant Sachdeva
University of Toronto, Canada

Abstract

In this paper, we investigate the question of whether the electrical flow routing is a good oblivious routing scheme on an m-edge graph $G = (V, E)$ that is a Φ-expander, i.e. where $|\partial S| \geq \Phi \cdot \mathrm{vol}(S)$ for every $S \subseteq V, \mathrm{vol}(S) \leq \mathrm{vol}(V)/2$. Beyond its simplicity and structural importance, this question is well-motivated by the current state-of-the-art of fast algorithms for ℓ_∞ oblivious routings that reduce to the expander-case which is in turn solved by electrical flow routing.

Our main result proves that the electrical routing is an $O(\Phi^{-1} \log m)$-competitive oblivious routing in the ℓ_1- and ℓ_∞-norms. We further observe that the oblivious routing is $O(\log^2 m)$-competitive in the ℓ_2-norm and, in fact, $O(\log m)$-competitive if ℓ_2-localization is $O(\log m)$ which is widely believed.

Using these three upper bounds, we can smoothly interpolate to obtain upper bounds for every $p \in [2, \infty]$ and q given by $1/p + 1/q = 1$. Assuming ℓ_2-localization in $O(\log m)$, we obtain that in ℓ_p and ℓ_q, the electrical oblivious routing is $O(\Phi^{-(1-2/p)} \log m)$ competitive. Using the currently known result for ℓ_2-localization, this ratio deteriorates by at most a sublogarithmic factor for every $p, q \neq 2$.

We complement our upper bounds with lower bounds that show that the electrical routing for any such p and q is $\Omega(\Phi^{-(1-2/p)} \log m)$-competitive. This renders our results in ℓ_1 and ℓ_∞ unconditionally tight up to constants, and the result in any ℓ_p- and ℓ_q-norm to be tight in case of ℓ_2-localization in $O(\log m)$.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Expanders, Oblivious routing for ℓ_p, Electrical flow routing

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.65

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2406.07252`

Funding *Rasmus Kyng*: "Algorithms and complexity for high-accuracy flows and convex optimization" grant (no. 200021 204787) of the Swiss National Science Foundation.
Maximilian Probst Gutenberg: "Algorithms and complexity for high-accuracy flows and convex optimization" grant (no. 200021 204787) of the Swiss National Science Foundation.
Sushant Sachdeva: NSERC Discovery Grant RGPIN-2018-06398, an Ontario Early Researcher Award (ERA) ER21-16-284, and a Sloan Research Fellowship.

Acknowledgements We would like to thank Yang P. Liu for pointing us to the Riesz-Thorin theorem.

1 Introduction

In this paper, we study flow-routing problems on connected, undirected (multi-)graphs $G = (V, E)$. A broad and well-studied class of single-commodity flow problems arises by seeking a flow $\boldsymbol{f} \in \mathbb{R}^E$ that routes given demands $\boldsymbol{\chi} \in \mathbb{R}^V$, while minimizing a ℓ_p-norm of

the flow. Denoting the graph edge-vertex incidence matrix by $\boldsymbol{B} \in \mathbb{R}^{V \times E}$, we can write these optimization problems as

$$\min_{\boldsymbol{B}\boldsymbol{f}=\boldsymbol{\chi}} \|\boldsymbol{f}\|_p. \tag{1}$$

The case $p = \infty$ is known as undirected maximum flow, while $p = 2$ is called electrical flow and $p = 1$ is called transshipment. Here we focus for simplicity on the unweighted setting, but all results in this paper and in related work can in fact be extended to work in weighted graphs.

We can generalize these flow problems to the multi-commodity case by allowing a collection of demands $\{\boldsymbol{\chi}_i\}$ to be routed simultaneously by a collection of flows $\{\boldsymbol{f}_i\}$, while minimizing a single objective on all of them.

$$\min_{\boldsymbol{B}\boldsymbol{f}_i=\boldsymbol{\chi}_i,\forall i} \|\sum_i |\boldsymbol{f}_i|\|_p. \tag{2}$$

For any p, solutions with $(1 + 1/\operatorname{poly}(|E|))$-multiplicative error to these problems can be computed in polynomial time and for the single-commodity setting even in almost-linear time [18, 7]. For the special cases of $p = 1, 2, \infty$, optimal solutions can be computed in polynomial time via linear/convex programming.

However, in many settings, we may want to sacrifice optimality of our routing solutions for simplicity of the routing algorithm. A particularly simple and popular approach is *oblivious routing*, where a collection of routing paths are chosen in advance between every pair of nodes, without knowing the demands that will be eventually routed. Historically, oblivious routings were first studied on specific networks, specifically the hypercube [37, 38]. A deeply influential technique in this area is the work of Rackë [27]. An *oblivious routing* is linear operator $\boldsymbol{A} \in \mathbb{R}^{E \times V}$ that maps any valid[1] demand vector $\boldsymbol{\chi} \in \mathbb{R}^V$ to a flow $\boldsymbol{f} = \boldsymbol{A}\boldsymbol{\chi}$ that routes $\boldsymbol{\chi}$. This extends to routing multiple demands in the multi-commodity setting, $\{\boldsymbol{f}_i = \boldsymbol{A}\boldsymbol{\chi}_i\}$.

Conceptually, a highly attractive feature is that multiple demand vectors can be routed simultaneously without knowing the other demands, and a single demand can be broken down into multiple terms, e.g. source-sink demand pairs, and routings of each pair can be again computed separately. These features make oblivious routings ideal for online routing problems – which was the original motivation for their construction [27].

As mentioned above, using oblivious routing comes at the sacrifice of optimality. To get a quantitative measure of the loss a routing scheme $\boldsymbol{A}$ might incur in the ℓ_p metric, we define the *competitive ratio* of $\boldsymbol{A}$, denoted by $\rho_p(\boldsymbol{A})$, to be the maximal ratio between the objective value achieved by an oblivious routing $\boldsymbol{A}$ and the optimal solution achieved by *any* (multi-commodity) demand.

In a ground-breaking sequence of papers [27, 3, 9], Räcke, Azar, Cohen, Fiat, Kaplan, and Englert showed that for all ℓ_p-norms, oblivious routings with competitive ratio $\tilde{O}(1)$ exist[2]. In fact, for the well-studied setting of $p = \infty$, [28] gave an optimal construction with $O(\log m)$ competitive ratio in polynomial time, matching a $\Omega(\log m)$ lower bound [4, 25].

Fast Algorithms and Applications for ℓ_∞ Oblivious Routing

ℓ_∞ oblivious routings are a fundamental tool in obtaining fast approximate maximum flow algorithms in undirected graphs. Building on the techniques in [35, 24], [16, 33] give algorithms that show that $O(\operatorname{poly}(\alpha/\varepsilon))$ applications of a α-competitive ℓ_∞ oblivious routing

[1] A demand $\boldsymbol{\chi}$ can be routed on a connected graph iff $\sum_v \chi(v) = 0$.
[2] We use $\tilde{O}(\cdot)$ to hide polylogarithmic factors in the graph size m.

yield $(1+\varepsilon)$-approximate maximum flow on undirected graphs by using gradient descent. They then developed almost-linear time algorithms for $m^{o(1)}$-competitive ℓ_∞ oblivious routing. As a result, they obtained approximate undirected maximum flow in time $m^{1+o(1)}\,\mathrm{poly}(1/\varepsilon)$ – one of the major recent breakthroughs in modern graph algorithms.

Later, [29] gave a reduction from computing $\tilde{O}(1)$-competitive ℓ_∞ oblivious routings to approximate maximum flows resulting in a $m^{1+o(1)}$ time algorithm. [26] then showed that combining these approaches recursively yields a $\tilde{O}(m)$ algorithm to compute $\tilde{O}(1)$-competitive ℓ_∞ oblivious routings and a $\tilde{O}(m\,\mathrm{poly}(1/\varepsilon))$ algorithm for $(1+\varepsilon)$-approximate undirected maximum flow [26]. Recently, [12] presented an alternative, simple $\tilde{O}(m)$ algorithm to obtain (and maintain) ℓ_∞ oblivious routings with subpolynomial competitive factor.

While recently the first *exact* maximum flow algorithm with runtime $m^{1+o(1)}$ was given in [7], ℓ_∞ oblivious routings and approximate undirected maximum flow remain important tools with many algorithms crucially relying on them as subroutines to obtain runtime $\tilde{O}(m)$.

We point out that above, for simplicity, we did not properly distinguish between ℓ_∞ oblivious routings which are only constructed in [16], and their weaker counterparts *congestion approximators* which are used in all other constructions. A congestion approximator is a linear operator $\boldsymbol{C}$ that maps each demand $\boldsymbol{\chi}$ to vector $\boldsymbol{c} = \boldsymbol{C}\boldsymbol{\chi}$ such that $\|\boldsymbol{c}\|_\infty$ approximates the objective value of (2). Note, that $\boldsymbol{c}$ is not necessarily a flow.

ℓ_∞ Oblivious Routing on Expanders and in General Graphs

Valiant's trick [38], a popular scheme that routes demands from each source to a set of randomly chosen intermediate nodes before routing them to the destination, establishes the existence of $O(\Phi^{-1}\log n)$-competitive ℓ_∞-oblivious routings in expanders. However, implementing Valiant's trick algorithmically requires computing multi-commodity flows, which are expensive to compute.

To the best of our knowledge, the only fast algorithm that computes an ℓ_∞ oblivious routing on general graphs is given in [16]. In their approach, they first reduce the problem to finding an ℓ_∞ oblivious routing on a Φ-expander with unit-weights (in this case $\Phi = m^{-o(1)}$). They then exploit a simple but striking statement, previously demonstrated by Kelner and Maymounkov [15]: the electrical flow routing, henceforth denoted by $\boldsymbol{A}_{\mathcal{E}}$, on a Φ-expander is a $O(\Phi^{-2}\log m)$-competitive ℓ_∞-routing. It was later observed by Schild-Rao-Srivastava [32] that on unweighted graphs, the statement can be derived from Cheeger's Inequality ([5, 1]). Further, the electrical flow routing can be applied efficiently after $\tilde{O}(m)$ preprocessing, due to the breakthrough result by Spielman and Teng [35] and subsequent work [17, 8, 20, 14][3]. [16] then demonstrates that by assembling and combining these routings on expanders, one obtains an ℓ_∞ oblivious routing of the entire graph that can be evaluated efficiently.

As far as we know, no other fast algorithm is currently known to compute ℓ_∞ oblivious routing, and all fast algorithms that compute congestion approximators again reduce to expanders on which cuts can be approximated by stars. Therefore, to the best of our knowledge, every almost-linear time approach to constructing ℓ_∞ oblivious routing reduces to expanders, and on expanders the only known fast algorithm for obtaining an ℓ_∞ oblivious routing is to use the electrical flow routing.

[3] Technically, only a high-accuracy solution is obtained which suffices for our application.

Oblivious Routing for any ℓ_p

Analogous to the reduction of solving approximate undirected maximum flow via few applications of ℓ_∞ oblivious routing, Sherman later showed in [34], that any ℓ_p-norm minimizing flow on undirected graphs can be computed to an $(1+\varepsilon)$-approximation by applying ℓ_p oblivious routings with α competitive ratio $\tilde{O}(\text{poly}(\alpha/\varepsilon))$ times via gradient descent.

While we are not aware of any article studying fast algorithms for the general ℓ_p-norm, the ℓ_1-norm has received considerable attention and $\tilde{O}(m)$ time algorithms were given with competitive ratio $\tilde{O}(1)$ [23, 39, 30], and adapted to fully-dynamic graphs in [6].

Oblivious Routing for ℓ_1 on Expanders

Further, at least in the unit-capacity setting, the result by Kelner and Maymounkov [15] extends seamlessly to the ℓ_1-norm, i.e. the electric flow routing $\boldsymbol{A}_\mathcal{E}$ has competitive ratio $O(\Phi^{-2}\log m)$. This follows since the electrical flow routing is given by $\boldsymbol{A}_\mathcal{E} = \boldsymbol{B}^\top \boldsymbol{L}^+$, where $\boldsymbol{B}$ is the vertex-edge incidence matrix and $\boldsymbol{L} = \boldsymbol{B}\boldsymbol{B}^\top$ is the Laplacian matrix of the graph and then bounding the ℓ_∞ competitive ratio of the oblivious routing for multicommodity flow problems is equivalent to bounding the quantity $\||\boldsymbol{A}_\mathcal{E}\boldsymbol{B}|\|_{\infty\to\infty}$, where $|\cdot|$ denotes the entrywise absolute value, while the ℓ_1 competitive ratio equals $\||\boldsymbol{A}_\mathcal{E}\boldsymbol{B}|\|_{1\to 1}$. The matrix $\Pi = \boldsymbol{A}_\mathcal{E}\boldsymbol{B} = \boldsymbol{B}^\top\boldsymbol{L}^+\boldsymbol{B}$ is a frequently-studied orthogonal projection matrix and it is a symmetric matrix, since $\boldsymbol{B}^\top\boldsymbol{L}^+\boldsymbol{B} = \boldsymbol{B}^\top\boldsymbol{A}_\mathcal{E}^\top = (\boldsymbol{A}_\mathcal{E}\boldsymbol{B})^\top = (\boldsymbol{B}^\top\boldsymbol{L}^+\boldsymbol{B})^\top$ where we use that $\boldsymbol{L}^+$ is symmetric. But it is further well-known that $\|\boldsymbol{X}\|_{\infty\to\infty} = \|\boldsymbol{X}^\top\|_{1\to 1}$ and thus we have that $\||\Pi|\|_{\infty\to\infty} = \||\Pi|\|_{1\to 1}$, i.e. the competitive ratios achieved by $\boldsymbol{A}_\mathcal{E}$ in ℓ_1- and ℓ_∞-norm are equal.

Beyond Oblivious Routing

The quantity $\||\Pi|\|_{\infty\to 1}$ is important in several other contexts: It captures the so-called *localization* of electrical flow on the graph [32]. Localization measures the ℓ_1-length of the electrical flow corresponding to a demand placed at two endpoints of an edge, averaged over all edges. The bound $\rho_1(\boldsymbol{A}_\mathcal{E}) = O(\Phi^{-2}\log m)$ implies a stronger statement in expanders: For every such edge-demand, the ℓ_1-length is bounded by $O(\Phi^{-2}\log m)$. It is known that in general graphs, localization is bounded by $O(\log^2 m)$ – but it is open whether $O(\log m)$ holds (a lower bound of $\Omega(\log m)$ is known) although it is widely believed. From [15], we see that any graph with expansion $1/o(\sqrt{\log m})$ achieves localization $o(\log^2 m)$.

Localization has been used in a number of contexts, including sampling random spanning trees in almost-linear time [31], computing spectral subspace sparsification [22] of Laplacian matrices, and building oblivious routings using $\tilde{O}(\sqrt{m})$ electrical flows [11].

An interesting message of our paper is that electrical flow on expanders simultaneously is an excellent ℓ_1 and ℓ_∞ oblivious routing, i.e. it uses flow paths are are both short and low congestion. A broad theory of expanders that simultaneously allow for short and low-congestion paths has recently been developed in [10, 13], allowing for other possible trade-offs between length and congestion than those obtained by conventional expanders.

1.1 Main Contributions

In this article, we study a simple but important question:

> *Given* ***any*** $p \in [1,\infty]$*, what is the competitive ratio* $\rho_p(\boldsymbol{A}_\mathcal{E})$ *of the electrical flow routing* $\boldsymbol{A}_\mathcal{E}$ *on a* Φ*-expander?*

We first settle this question for the important cases when $p \in \{1, \infty\}$ by proving the following theorem that proves an upper bound that is tight up to constant factors.

▶ **Theorem 1.1.** *For a Φ-expander multigraph $G = (V, E)$ with edge-vertex incidence matrix $\boldsymbol{B}$ and Laplacian $\boldsymbol{L}$, the electrical routing $\boldsymbol{A}_{\mathcal{E}} = \boldsymbol{B}^\top \boldsymbol{L}^+$ has competitive ratios ρ_∞ and ρ_1 for multi-commodity ℓ_∞ and ℓ_1 routing both bounded by*

$$\rho_\infty(\boldsymbol{A}_{\mathcal{E}}), \rho_1(\boldsymbol{A}_{\mathcal{E}}) \leq 3 \cdot \frac{\log(2m)}{\Phi}$$

The Riesz-Thorin theorem then gives us a way to smoothly interpolate between the upper bounds of any two ℓ_{p_1}- and ℓ_{p_2}-norm competitive ratios to obtain an upper bound on the ℓ_p-norm competitive ratio $\rho_p(\boldsymbol{A}_{\mathcal{E}})$ for any $p_1 < p < p_2$. Using a smooth interpolation between our results for ℓ_1- and ℓ_∞-norm, we thus obtain the following more general result.

▶ **Theorem 1.2.** *For a Φ-expander multigraph $G = (V, E)$, and any $p \in [1, \infty]$, we have that the competitive ratio of $\boldsymbol{A}_{\mathcal{E}}$ is*

$$\rho_p(\boldsymbol{A}_{\mathcal{E}}) \leq 3 \cdot \frac{\log(2m)}{\Phi}.$$

It was proven in [32], that $\Pi = \boldsymbol{B}^\top \boldsymbol{L}^+ \boldsymbol{B}$ satisfies $\||\Pi|\|_{2\to 2} \leq O(\log^2 n)$ (where $|\cdot|$ indicates entry-wise absolute value). We refer to this quantity $\||\Pi|\|_{2\to 2}$ as *ℓ_2-localization*. It is widely believed that $\||\Pi|\|_{2\to 2} \leq O(\log m)$. Implicit in earlier works, albeit perhaps not widely observed, is that $\rho_2(\boldsymbol{A}_{\mathcal{E}}) = \||\Pi|\|_{2\to 2}$ (see Lemma 4.1), i.e. the competitive ratio of multi-commodity ℓ_2 routing is exactly characterized by ℓ_2-localization. By interpolating with this norm bound and our bounds on $\rho_1(\boldsymbol{A}_{\mathcal{E}})$ and $\rho_\infty(\boldsymbol{A}_{\mathcal{E}})$, we obtain potentially much stonger bounds on the competitive ratio $\rho_p(\boldsymbol{A}_{\mathcal{E}})$.

▶ **Corollary 1.3** (implied by Riesz-Thorin)**.** *For a Φ-expander multigraph $G = (V, E)$, and any $p \in [2, \infty]$ and q given by $1/p + 1/q = 1$, we have that the competitive ratios of $\boldsymbol{A}_{\mathcal{E}}$ for the ℓ_p and ℓ_q norms are*

$$\rho_p(\boldsymbol{A}_{\mathcal{E}}), \rho_q(\boldsymbol{A}_{\mathcal{E}}) \leq \||\Pi|\|_{2\to 2}^{2/p} \left(3 \cdot \frac{\log(2m)}{\Phi}\right)^{1-2/p}.$$

We complement our upper bounds with strong, unconditional lower bounds. Remarkably, if $\||\Pi|\|_{2\to 2} \leq O(\log m)$, as widely believed, then our bounds are tight up to constants. Even with the currently known fact $\||\Pi|\|_{2\to 2} \leq O(\log^2 n)$, our lower bounds still prove a sublogarithmic gap in every constant $p \neq 2$ and q and optimal Φ dependency.

▶ **Theorem 1.4.** *For an infinite number of positive integers n and any $\Phi \in [1/\sqrt[3]{n}, 1]$, for any $p \in [2, \infty]$ and q given by $1/p + 1/q = 1$, we have that*

$$\rho_p(\boldsymbol{A}_{\mathcal{E}}), \rho_q(\boldsymbol{A}_{\mathcal{E}}) \geq \Omega\left(\frac{\log m}{\Phi^{1-2/p}}\right).$$

1.2 Oblivious Electric Routing in Weighted Graphs

We can extend Theorem 1.1 to weighted graphs in the following way: Consider a graph $G = (V, E)$ with positive integer edge capacities $\boldsymbol{c} \in \mathbb{R}^E$ and positive integer edge lengths $\boldsymbol{s} \in \mathbb{R}^E$. Letting $\boldsymbol{C}, \boldsymbol{S} \in \mathbb{R}^{E\times E}$ denote diagonal matrices with $\boldsymbol{c}$ and $\boldsymbol{s}$ on the diagonal respectively, we are interested in the optimal weighted ℓ_∞- and ℓ_1-routings for given demands $D = \{\boldsymbol{\chi}_1, \ldots, \boldsymbol{\chi}_k\}$

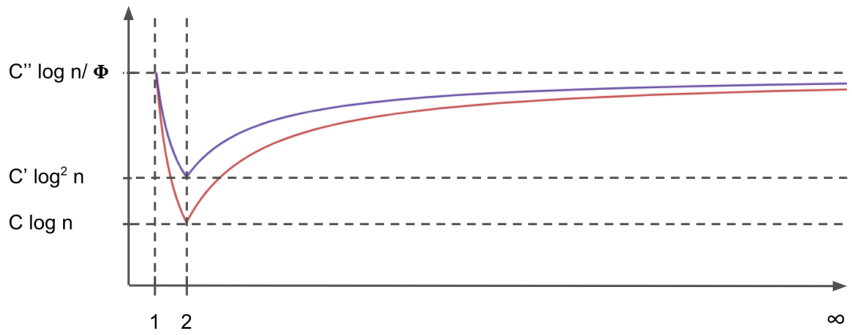


Figure 1 An illustration of the result given in Corollary 1.3 of the different competitive ratios achieved with respect to each ℓ_p-norm, where n and Φ are fixed and $\Phi \ll 1/\log m$. The red curve shows the optimal ratios, achieved if localization is in $O(\log m)$, which is also obtained up to constant factors by our lower bound in Theorem 1.4. The lilac curve shows the current trade-off where we use the known result that localization is in $O(\log^2 n)$. For $\Phi \gg 1/\log m$ the upper bound of Theorem 1.2 achieves values between the two curves.

$$\mathrm{opt}_\infty(D) = \min_{\boldsymbol{B}\boldsymbol{f}_i=\boldsymbol{\chi}_i,\forall i} \|\boldsymbol{C}^{-1}\sum_i |\boldsymbol{f}_i|\|_\infty \text{ and } \mathrm{opt}_1(D) = \min_{\boldsymbol{B}\boldsymbol{f}_i=\boldsymbol{\chi}_i,\forall i} \|\boldsymbol{S}\sum_i |\boldsymbol{f}_i|\|_1 \tag{3}$$

Now consider defining an electrical routing by choosing resistances $\boldsymbol{R} = \boldsymbol{S}\boldsymbol{C}^{-1}$ and defining the electrical routing $\boldsymbol{A}_\mathcal{E} = \boldsymbol{R}^{-1}\boldsymbol{B}^\top(\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^\top)^+$. Let $\hat{G}$ denote the multigraph with edge e replaced by $\boldsymbol{c}(e)$ unit-weight paths of length $\boldsymbol{s}(e)$. Now, one can easily show that the electrical routing in G according to $\boldsymbol{A}_\mathcal{E}$ is equal to the unit-weight electrical routing in $\hat{G}$, when mapping flows on a capacitated edge to a collection of flows on multi-edge paths.

▶ **Corollary 1.5** ((Informal) Electrical Oblivious ℓ_∞- and ℓ_1-Routing on Weighted Expanders)**.** *For (multi-)graph $G = (V, E)$ with edge-vertex incidence matrix $\boldsymbol{B}$ and positive integer edge weights and lengths given as diagonal matrices $\boldsymbol{C}, \boldsymbol{S} \in \mathbb{R}^{E\times E}$, the electrical routing $\boldsymbol{A}_\mathcal{E} = \boldsymbol{R}^{-1}\boldsymbol{B}^\top(\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^\top)^+$, where $\boldsymbol{R} = \boldsymbol{S}\boldsymbol{C}^{-1}$, has competitive ratios ρ_∞ and ρ_1 for multicommodity ℓ_∞ and ℓ_1 routing both bounded by*

$$\rho_\infty(\boldsymbol{A}_\mathcal{E}), \rho_1(\boldsymbol{A}_\mathcal{E}) \le 3 \cdot \frac{\ln(2|E_{\hat{G}}|)}{\Phi(\hat{G})}$$

where $\hat{G}$ denotes the multigraph with edge e replaced by a path of length $\boldsymbol{S}(e,e)$ with $\boldsymbol{C}(e,e)$ unit-weight multi-edges across each hop of the path.

▶ Remark 1.6. When we take all edge lengths $\boldsymbol{S}(e,e)$ to be 1, the expansion of $\hat{G}$ equals the usual definition of expansion in graph G with edge weight equal to capacity.

1.3 New Implications for Localization

From the connection to localization outlined earlier, we also immediately conclude that localization of graphs with $1/o(\log m)$ expansion improve over the general localization bound of $O(\log^2 m)$.

▶ **Corollary 1.7** ((Informal) Localization of Electrical Flow)**.** *For a multigraph $G = (V, E)$, the average over multi-edges of the ℓ_1-norm the electrical flow routing 1 unit of flow across the multi-edge is bounded by $O\left(\min\{\Phi^{-1}\log m, \log^2 n\}\right)$, and hence graphs with expansion $\Phi = 1/o(\log m)$ have localization $o(\log^2 m)$.*

1.4 Roadmap

We next give a Preliminary section to set up the necessary notation for the article. We then prove Theorem 3 in Section 3. We use this result together with the Riesz-Thorin theorem to obtain Theorem 1.2 and Corollary 1.3 in Section 4. Finally, in Section 5, we give our lower bounds as stated in Theorem 1.4.

2 Preliminaries

General Definitions

For any $n \in \mathbb{N}^*$, we let $[n]$ denote the set $\{1, 2, \ldots, n\}$. We let $\mathbf{1}$ denote the all ones vector and $\mathbf{1}_S$ denote the vector that has ones in the positions indexed by the elements of the set S and zeros otherwise. For any $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, we let $|\boldsymbol{A}|$ denote the matrix where the absolute value operator has been applied entrywise.

Graphs

Although our results in the contribution section are for unweighted graphs, we also prove stronger statements in the article that also work on weighted graphs. Therefore, we define various notions with respect to weighted graphs.

Given an input graph $G = (V, E, w)$ with positive weights which we all assume to be at least 1, we define $n = |V|$ and $m = |E|$. We assume an arbitrary underlying direction assigned to each edge of G. We define the edge-vertex incidence matrix $\boldsymbol{B} \in \mathbb{R}^{V \times E}$ of G as

$$\boldsymbol{B}(w, e) = \begin{cases} -1, & \text{if } e = (w, v) \\ 1, & \text{if } e = (v, w) \\ 0, & \text{otherwise} \end{cases}.$$

We define the Laplacian $\boldsymbol{L} = \boldsymbol{B}\boldsymbol{W}\boldsymbol{B}^\top$ where $\boldsymbol{W}$ is the diagonal matrix given by the weights w and denote by $\boldsymbol{L}^+$ the pseudo-inverse of the Laplacian. We call $\Pi = \boldsymbol{B}^\top \boldsymbol{L}^+ \boldsymbol{B}$ the *unweighted* projection matrix of G.

Expanders

We say G is a Φ-expander if $|\partial S| \geq \Phi \cdot \mathrm{vol}(S)$ for every $S \subseteq V, \mathrm{vol}(S) \leq \mathrm{vol}(V)/2$, where we define $|\partial S|$ to be the weight of all edges with exactly one endpoint in S, and $\mathrm{vol}(S)$ to be the sum of weighted degrees of vertices in S.

Flows and Congestion

We say $\boldsymbol{\chi} \in \mathbb{R}^V$ is a demand vector if $\mathbf{1}^\top \boldsymbol{\chi} = 0$. We let $\boldsymbol{\chi}_{(a,b)} \in \mathbb{R}^V$ for every $(a, b) \in E$ to be the unitary demand on the edge (a, b), that is $\boldsymbol{\chi}_{(a,b)} = \mathbf{1}_a - \mathbf{1}_b$. We say a vector $\boldsymbol{f} \in \mathbb{R}^E$ is a flow that routes demand $\boldsymbol{\chi}$ if $\boldsymbol{B}\boldsymbol{f} = \boldsymbol{\chi}$. Given an arbitrary norm $\|\cdot\|$ on $\mathbb{R}^E$, we define the congestion of a multi-set of flows $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_k\}$ to be:

$$\mathrm{cong}\left(\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_k\}\right) = \left\| \boldsymbol{W}^{-1} \sum_{i=1}^{k} |\boldsymbol{f}_i| \right\|.$$

Oblivious Routings

We define an oblivious routing on G to be a linear operator $\boldsymbol{A} \in \mathbb{R}^{E \times V}$ such that $\boldsymbol{B}\boldsymbol{A}\boldsymbol{\chi} = \boldsymbol{\chi}$ for all demand vectors $\boldsymbol{\chi} \in \mathbb{R}^V$, i.e. to be a flow that routes the demand $\boldsymbol{\chi}$.

Given a multiset of demands $D = \{\boldsymbol{\chi}_1, \ldots, \boldsymbol{\chi}_k\}$, we define the optimal congestion achievable by

$$\mathrm{opt}(D) = \min_{\{\boldsymbol{f}_i\}_{i\in[k]} \text{ multiset} \,:\, \boldsymbol{B}\boldsymbol{f}_i = \boldsymbol{\chi}_i, \forall i} \mathrm{cong}(\{\boldsymbol{f}_i\}_{i\in[k]}).$$

This allows us to define the competitive ratio of an oblivious routing, which we define

$$\rho(\boldsymbol{A}) = \max_{\{\boldsymbol{\chi}_i\}_{i\in[k]} \text{ multiset} \,:\, \boldsymbol{\chi}_i \perp \mathbf{1}, \forall i} \frac{\mathrm{cong}\left(\{\boldsymbol{A}\boldsymbol{\chi}_i\}_{i\in[k]}\right)}{\mathrm{opt}\left(\{\boldsymbol{\chi}_i\}_{i\in[k]}\right)}.$$

Note that whenever we use the subscript "p" for the competitive ratio ρ, we mean that the norm used in defining the congestion in that special case is the ℓ_p-norm.

Electrical Flows and Voltages

In this article, we define the electric flow routing operator $\boldsymbol{A}_{\mathcal{E}} = \boldsymbol{W}\boldsymbol{B}^\top\boldsymbol{L}^+$. Right-applying the operator $\boldsymbol{A}$ to any demand $\boldsymbol{\chi}$ yields the electric flow $\boldsymbol{f} = \boldsymbol{A}\boldsymbol{\chi}$ that routes the demand $\boldsymbol{\chi}$. We define the electrical energy associated with the flow vector $\boldsymbol{f}$ by $\mathcal{E}(\boldsymbol{f}) = \boldsymbol{f}^\top\boldsymbol{W}^{-1}\boldsymbol{f}$.

We define the electric voltage vector $\boldsymbol{v} \in \mathbb{R}^V$ with respect to a demand $\boldsymbol{\chi}$ by $\boldsymbol{v} = \boldsymbol{L}^+\boldsymbol{\chi}$. We define the electrical energy associated with the voltage vector $\boldsymbol{v}$ as $\mathcal{E}(\boldsymbol{v}) = \boldsymbol{v}^\top\boldsymbol{L}\boldsymbol{v}$. Note in particular that the energy of voltages induced by a certain demand coincides with the energy of the respective flow.

We introduce the notion of *"fractional" volume* at given a threshold $t \in \mathbb{R}$ with respect to a given voltage vector $\boldsymbol{v} \in \mathbb{R}^V$. We first define the fractional volume per edge and then for the whole graph. For an edge $(a, b) \in E$:

$$\mathrm{vol}_{\geq t}(a,b) = \begin{cases} 2 \cdot w(a,b), & \text{if } \boldsymbol{v}(a) > t \\ 2 \cdot w(a,b) \cdot \frac{\boldsymbol{v}(b) - t}{\boldsymbol{v}(b) - \boldsymbol{v}(a)}, & \text{if } \boldsymbol{v}(b) \leq t \\ 0, & \text{otherwise} \end{cases}.$$

For the whole graph G:

$$\mathrm{vol}_{\geq}(t) = \sum_{(a,b)\in E} \mathrm{vol}_{\geq t}(a,b),$$

$$\mathrm{vol}^+_{\geq}(t) = \mathrm{vol}_{\geq}(t) + 1.$$

Note that in our notation we omit specifying which voltage vector the "fractional" volume function is tied to, as it will be clearly specified upon usage during the proofs.

We define $S_t = \{a \in V \mid v(a) \geq t\}$ as the set of vertices whose voltages are greater or equal to the arbitrary threshold $t \in \mathbb{R}$ and the cut determined by the voltage threshold $t \in \mathbb{R}$ to be $C_t = (S_t, V \setminus S_t)$. For convenience, we let the weight of the cut C_t be $\delta(t) = |\partial C_t| = \sum_{e\in C_t} w(e)$.

3 An Upper Bound on the Competitive Ratio of Electrical Flow Routing for ℓ_∞

In this section, we prove our main technical result, Theorem 1.1, by establishing a tight upper bound on the competitive ratio when the congestion is defined in terms of ℓ_∞ which then by the symmetry of Π immediately gives the same competitive ratio in ℓ_1.

However, while Theorem 1.1 only claims a result for unweighted (multi-)graphs, we show in this section that $\boldsymbol{A}_{\mathcal{E}}$ even has good competitive ratio in weighted graphs with respect to ℓ_∞. However, in the weighted setting, we cannot use the bound for ℓ_∞ to derive a bound on ℓ_1, as the matrix-norms that exactly characterize the competitive ratio are not equal in general. Nonetheless, the "multi-graph" trick from Corollary 1.5 can be used to transform the weighted setting into an unweighted instance and derive bounds.

Intuition for our Proof

Kelner and Maymounkov showed that in order to bound the congestion of the electrical routing, it suffices, via a duality (or transposition) argument, to bound the worst case ℓ_1-norm of the flow induced by routing 1 unit of flow electrically across any edge. We adopt the same approach, but give a more precise analysis.

Suppose $e = (x, y)$ is the edge such that routing one unit of flow between the endpoints causes the highest overall congestion. We let $\boldsymbol{v}$ be the associated voltage vector that induces the electrical flow routing one unit from x to y. The overall congestion then equals $\sum_{(a,b)\in E} w(a,b)|\boldsymbol{v}(a) - \boldsymbol{v}(b)|$. We can express this by integrating with respect to voltage along a voltage threshold cut with respect to $\boldsymbol{v}$, where the function being integrated at point t is exactly $Y_t = \sum_{(a,b)\in C_t} w(a,b)$, where C_t is the cut at voltage threshold t. This ensures that after integrating Y_t over the entire voltage range, each edge (a, b) contributes exactly $w(a,b)|\boldsymbol{v}(a) - \boldsymbol{v}(b)|$, as desired. Our proof proceeds by leveraging that the flow crossing the cut C_t at threshold t is exactly $\sum_{(a,b)\in C_t} w(a,b)|\boldsymbol{v}(a) - \boldsymbol{v}(b)|$.

As we are sending one unit of flow from x to y, and all electrical flow goes one way across a voltage cut, this quantity is exactly 1. At each threshold t, this creates an "on average" relationship between voltage difference $|\boldsymbol{v}(a) - \boldsymbol{v}(b)|$ and weight $w(a, b)$ for edges being cut. This in turn allows us to establish a pointwise relationship at each threshold voltage t between the growth in congestion and the change in volume at t. Armed with this relationship, we can bound the accumulated congestion of the integrated cuts in terms of the accumulated volume, and this yields our result.

Contrast with the Kelner-Maymounkov Proof

It is instructive to consider why the Kelner-Maymounkov congestion bound loses an additional factor Φ compared to our bound. For concreteness, consider the graph given by a direct edge from x to y and an additional k disjoint paths of length k from x to y. It can be shown that in this example, the edge that governs the congestion bound in the strategy above is in fact the direct (x, y) edge.

Kelner-Maymounkov upper bound the true competitive ratio $\rho' = \sum_{(a,b)\in E} w(a,b)|\boldsymbol{v}(a) - \boldsymbol{v}(b)|$ by the quantity $\rho'' = \sum_{a\in V} \boldsymbol{d}(a)|\boldsymbol{v}(a) - c|$ for some constant c (see Equation (4.3) in [15]). On this concrete graph, ρ' can be explicitly evaluated and is $\Theta(k)$. As the graph has expansion $1/k$, we can think of this as a bound of $\Theta(1/\Phi)$. But, ρ'' is $\Theta(k^2)$ i.e. $\Theta(1/\Phi^2)$.

However, Kelner and Maymoukov's strategy makes it difficult to directly bound ρ' as they first measure changes in volume over a (discrete) sequence of threshold cuts, and then changes in voltage over the same sequence of cuts. Their discrete sequence of cuts skips entirely over some edges, i.e. there will be edges that are not crossing any of their cuts. This makes it difficult to establish an estimate for each edge of the pointwise relation between its contribution to volume growth versus voltage growth or congestion growth. Hence, they work with summed bounds on voltage and compare these with summed bounds on volume, which naturally yields bounds on ρ''. But, as we have seen, a bound on ρ'' must inherently be loose as there is a gap between ρ' and ρ''.

▶ **Theorem 3.1** (ℓ_∞ Competitive bound of electrical flows)**.** *For a weighted Φ-expander multigraph $G = (V, E, w)$, the following holds:*

$$\rho_\infty(\boldsymbol{A}_\mathcal{E}) \leq 3 \cdot \frac{\ln(\mathrm{vol}(V))}{\Phi}.$$

Proof. We first use that

$$\rho_\infty(\boldsymbol{A}_\mathcal{E}) = \max_{e \in E} \left\| \boldsymbol{W}\boldsymbol{B}^\top \boldsymbol{L}^+ \boldsymbol{\chi}_e \right\|_1. \tag{4}$$

as shown in [16] as part of the proof of Lemma 26 (by using primarily Lemmas 10 and 11). In order to prove the desired inequality, we then fix an $e \in E$ such that the quantity in (4) gets maximized, and let $\boldsymbol{v} = \boldsymbol{L}^+\boldsymbol{\chi}_e$ be the voltage induced by setting a unitary demand on this edge. Thus, we equivalently aim to bound:

$$\rho_\infty(\boldsymbol{A}_\mathcal{E}) = \left\| \boldsymbol{W}\boldsymbol{B}^\top \boldsymbol{L}^+ \boldsymbol{\chi}_e \right\|_1 = \left\| \boldsymbol{W}\boldsymbol{B}^\top \boldsymbol{v} \right\|_1 = \sum_{(a,b)\in E} w(a,b) \cdot |\boldsymbol{v}(a) - \boldsymbol{v}(b)|. \tag{5}$$

Observe now that shifting all the values of $\boldsymbol{v}$ by the same constant does not change the value expressed in (5), and therefore we can assume without loss of generality that the voltages are centered around 0, that is we can assume $\mathrm{vol}\left(\{i \in V \mid \boldsymbol{v}(i) \geq 0\}\right) \geq \mathrm{vol}(V)/2$ and $\mathrm{vol}\left(\{i \in V \mid \boldsymbol{v}(i) \leq 0\}\right) \geq \mathrm{vol}(V)/2$.

Note that, by convention, the electrical flow $\boldsymbol{f}_\mathcal{E} = \boldsymbol{A}_\mathcal{E}\boldsymbol{\chi}_e$ induces an orientation on the edges in the set E. Henceforth, we assume without loss of generality that orientations of edges in E align with the direction of the electrical flow $\boldsymbol{f}_\mathcal{E}$, that is $\boldsymbol{f}_\mathcal{E}(a,b) = w(a,b)\cdot(\boldsymbol{v}(b) - \boldsymbol{v}(a)) \geq 0$ for any $(a,b) \in E$.

In the following, we will employ the definitions introduced in Section 2. These concepts give rise to the notion of "fractional" volume, which will ultimately allow us to bound the quantity of Equation (5).

It can easily be proven that $\mathrm{vol}_\geq$ is continuous in $\mathbb{R}$ and differentiable at any threshold level $t \in \mathbb{R}$ for which there does not exist a node $a \in V$ such that $\boldsymbol{v}(a) = t$. Furthermore, if $t_{\min} = \min\{\boldsymbol{v}(a) \mid a \in V\}$ and $t_{\max} = \max\{\boldsymbol{v}(a) \mid a \in V\}$, then $\mathrm{vol}_\geq(t_{\min}) = \mathrm{vol}(G)$ and $\mathrm{vol}_\geq(t_{\max}) = 0$. We can even assume without loss of generality a more precise centering of the voltages around 0, namely that $\mathrm{vol}_\geq(0) = \mathrm{vol}(V)/2$.

A voltage threshold level $t \in \mathbb{R}$ can be seen as naturally determining a cut C_t in G. Note that the assumption about centering the voltages around 0 ensures that $\mathrm{vol}(S_t) \leq \mathrm{vol}(V)/2$ for any $t > 0$, so it holds that $\min\{\mathrm{vol}(S_t), \mathrm{vol}(V \setminus S_t)\} = \mathrm{vol}(S_t)$.

By taking the orientation of the edges into account, we can drop the absolute value operator and rewrite Equation (5) as:

$$\begin{aligned}
\sum_{(a,b)\in E} w(a,b) \cdot |\boldsymbol{v}(a) - \boldsymbol{v}(b)| &= \sum_{(a,b)\in E} w(a,b) \cdot (\boldsymbol{v}(b) - \boldsymbol{v}(a)) \\
&= \sum_{(a,b)\in E} w(a,b) \cdot \int_{t_{\min}}^{t_{\max}} \mathbb{1}_{\boldsymbol{v}(a) < t \leq \boldsymbol{v}(b)}\, dt \\
&= \int_{t_{\min}}^{t_{\max}} \sum_{(a,b)\in E} w(a,b) \cdot \mathbb{1}_{\boldsymbol{v}(a) < t \leq \boldsymbol{v}(b)}\, dt \\
&= \int_{t_{\min}}^{t_{\max}} \delta(t)\, dt \\
&= \int_{t_{\min}}^{0} \delta(t)\, dt + \int_{0}^{t_{\max}} \delta(t)\, dt.
\end{aligned} \tag{6}$$

We will bound the quantity in (6) by separately bounding each of the two terms in the last equality. Only the proof for the integral over the non-negative values of t will be presented, the one for the non-positive values proceeds in an analogous manner. Assume thus for the rest of the proof that $t \geq 0$ holds.

In order to obtain the bound on $\delta(t)$ for non-negative t, we will inspect the rate of change of the "fractional" volume with respect to the voltage threshold of the fixed voltage vector $\boldsymbol{v}$. Intuitively, we grow an electrical threshold ball, and directly relate the change in volume to the stretch accumulated at the current voltage threshold. In more precise terms, we compute a bound on the derivative of $\mathrm{vol}^{+}_{\geq}$ with respect to t on the domain of differentiability as follows:

$$\begin{aligned}
-\frac{d}{dt}\mathrm{vol}^{+}_{\geq}(t) &= -\frac{d}{dt}\mathrm{vol}_{\geq}(t) \\
&= -\frac{d}{dt}\left(\sum_{(a,b)\in E} \mathrm{vol}_{\geq t}(a,b)\right) \\
&= -\frac{d}{dt}\left(\sum_{(a,b)\in C_t} \mathrm{vol}_{\geq t}(a,b)\right) \\
&= \sum_{(a,b)\in C_t} -\frac{d}{dt}\mathrm{vol}_{\geq t}(a,b) \\
&= \sum_{(a,b)\in C_t} 2\frac{w(a,b)}{\boldsymbol{v}(b)-\boldsymbol{v}(a)}.
\end{aligned} \tag{7}$$

By the construction based on the voltage levels, all edges of the cut C_t have their head in the set S_t. Therefore, the flow carried by these edges has to be the unit flow, since this is the demand of $\boldsymbol{\chi}_e$:

$$1 = \sum_{(a,b)\in C_t} f_{\mathcal{E}}(a,b) = \sum_{(a,b)\in C_t} w(a,b)\cdot(\boldsymbol{v}(b)-\boldsymbol{v}(a)). \tag{8}$$

Hereafter, we show that the negative volume change must exceed the square of the cut size. Informally, the change in volume per edge is relatively large whenever the voltage gap across the edge is small (volume change scales as inversely proportional to the gap compared to the cut-value of the edge). But, a "typical" edge in the cut must have a fairly small voltage gap, as we otherwise route too much flow across the gap. Formally, since the voltage drops among the edges in the cut C_t are non-negative, we can use (8) and the definition of the conductance of G to further bound (7) using the Cauchy–Bunyakovsky–Schwarz inequality:

$$
\begin{aligned}
-\frac{d}{dt}\mathrm{vol}^{+}_{\geq}(t) &= 2\sum_{(a,b)\in C_t}\frac{w(a,b)}{\boldsymbol{v}(b)-\boldsymbol{v}(a)} \\
&= 2\left(\sum_{(a,b)\in C_t}\frac{w(a,b)}{\boldsymbol{v}(b)-\boldsymbol{v}(a)}\right)\cdot 1 \\
&= 2\left(\sum_{(a,b)\in C_t}\frac{w(a,b)}{\boldsymbol{v}(b)-\boldsymbol{v}(a)}\right)\left(\sum_{(a,b)\in C_t} w(a,b)\cdot(\boldsymbol{v}(b)-\boldsymbol{v}(a))\right) \\
&\geq 2\left(\sum_{(a,b)\in C_t}\sqrt{\frac{w(a,b)}{\boldsymbol{v}(b)-\boldsymbol{v}(a)}\cdot w(a,b)\cdot(\boldsymbol{v}(b)-\boldsymbol{v}(a))}\right)^2 \\
&\geq 2\left(\sum_{(a,b)\in C_t} w(a,b)\right)^2 \\
&= 2\cdot\delta(t)^2 \\
&\geq 2\cdot\delta(t)\cdot\Phi\cdot\mathrm{vol}(S_t).
\end{aligned}
\tag{9}
$$

Denote by $\mathrm{vol}_{\mathrm{int}}(t) = \mathrm{vol}(S_t) - \delta(t)$ twice the weight of the edges that have both endpoints in the set S_t. Recall that we assumed all of the edges to have weights at least 1, therefore it holds that $\delta(t) \geq 1$. The definition of "fractional" volume implies $\mathrm{vol}_{\geq}(t) \leq \mathrm{vol}_{\mathrm{int}} + 2\delta(t)$, which can be used to further bound (9):

$$
\begin{aligned}
-\frac{d}{dt}\mathrm{vol}^{+}_{\geq}(t) &\geq 2\cdot\delta(t)\cdot\Phi\cdot\mathrm{vol}(S_t) \\
&= 2\cdot\delta(t)\cdot\Phi\cdot(\mathrm{vol}_{\mathrm{int}}(t)+\delta(t)) \\
&= \delta(t)\cdot\frac{2\Phi}{3}\cdot(3\mathrm{vol}_{\mathrm{int}}(t)+3\delta(t)) \\
&\geq \delta(t)\cdot\frac{2\Phi}{3}\cdot(\mathrm{vol}_{\mathrm{int}}(t)+2\delta(t)+1) \\
&\geq \delta(t)\cdot\frac{2\Phi}{3}\cdot(\mathrm{vol}_{\geq}(t)+1) \\
&= \delta(t)\cdot\frac{2\Phi}{3}\cdot\mathrm{vol}^{+}_{\geq}(t).
\end{aligned}
$$

Observe that we can rewrite the inequality above to obtain a bound on $\delta(t)$:

$$
\delta(t) \leq -\frac{3}{2\Phi}\cdot\frac{1}{\mathrm{vol}^{+}_{\geq}(t)}\cdot\frac{d}{dt}\mathrm{vol}^{+}_{\geq}(t). \tag{10}
$$

We can now use equation (10) to bound the integral over the interval $[0, t_{\max}]$ in (6):

$$
\int_0^{t_{\max}}\delta(t)\,dt \leq \int_0^{t_{\max}} -\frac{3}{2\Phi}\cdot\frac{1}{\mathrm{vol}^{+}_{\geq}(t)}\cdot\frac{d}{dt}\mathrm{vol}^{+}_{\geq}(t)\,dt.
$$

The latter integral can easily be computed via the change of variable $u = \mathrm{vol}^{+}_{\geq}(t)$, yielding the integration bounds $\mathrm{vol}^{+}_{\geq}(t_{\max}) = \mathrm{vol}_{\geq}(t_{\max})+1 = 1$ and $\mathrm{vol}^{+}_{\geq}(0) = \mathrm{vol}_{\geq}(0)+1 = \mathrm{vol}(V)/2+1$:

$$\begin{aligned}\int_0^{t_{\max}} \delta(t)\,dt &\leq \int_0^{t_{\max}} -\frac{3}{2\Phi}\cdot\frac{1}{\mathrm{vol}^+_{\geq}(t)}\cdot\frac{d}{dt}\mathrm{vol}^+_{\geq}(t)\,dt \\ &= \int_{t_{\max}}^{0} \frac{3}{2\Phi}\cdot\frac{1}{\mathrm{vol}^+_{\geq}(t)}\cdot\frac{d}{dt}\mathrm{vol}^+_{\geq}(t)\,dt \\ &= \frac{3}{2\Phi}\int_1^{\mathrm{vol}(V)/2+1}\frac{1}{u}\,du.\end{aligned}$$

Since the graph has by assumption at least two nodes connected by an edge with weight at least 1, it follows that $\mathrm{vol}(V) \geq 2 \iff \mathrm{vol}(V) \geq \mathrm{vol}(V)/2 + 1$. Coupled with the fact that $u > 0$ for $u \in [1, \mathrm{vol}(V)]$, this gives us the final bound for the integral:

$$\begin{aligned}\int_0^{t_{\max}} \delta(t)\,dt &\leq \frac{3}{2\Phi}\int_1^{\mathrm{vol}(V)/2+1}\frac{1}{u}\,du \\ &\leq \frac{3}{2\Phi}\int_1^{\mathrm{vol}(V)}\frac{1}{u}\,du \\ &= \frac{3}{2\Phi}\cdot\ln(\mathrm{vol}(V)).\end{aligned}$$

As already mentioned, the same bound can be obtained for the other term in (6) in an analogous manner.

Combining the aforementioned result with the relations given by (5) and (6) gives the desired inequality:

$$\rho_\infty(\boldsymbol{A}_\mathcal{E}) = \int_{t_{\min}}^{0}\delta(t)\,dt + \int_0^{t_{\max}}\delta(t)\,dt \leq 2\cdot\frac{3\ln(\mathrm{vol}(V))}{2\Phi} = \frac{3\ln(\mathrm{vol}(V))}{\Phi}.$$ ◀

4 An Upper Bound on the Competitive Ratio of Electrical Flow Routing for ℓ_p (for any p)

In this section, we prove two generalizations of Theorem 3.1. Previously, we gave a bound on the competitive ratio when the congestion was defined in terms of the ℓ_∞-norm. This result can be extended to ℓ_p-norms for an arbitrary $p \in [1, \infty]$ by using Theorem 3.1, and instantiating a special case of the Riesz-Thorin theorem.

This establishes both the results in Theorem 1.2 and in Corollary 1.3. We stress that the results obtained in this section crucially exploit the symmetry of Π and therefore only hold for unweighted graphs.

A Toolbox for ℓ_p-Norms

We first use the following lemma. Its proof is fairly standard and follows the structure of the proof of Lemma 24 in [16]. We include a full proof in an extended version of this paper on arXiv.

▶ **Lemma 4.1** (Competitive ratio of ℓ_p-norms). *Let $G = (V, E)$ be a multigraph. For any $p \in [1, \infty]$ and oblivious routing $\boldsymbol{A}$, we have*

$$\rho_p(\boldsymbol{A}) = |||\boldsymbol{AB}|||_{p\to p}.$$

Our second tool is the Riesz-Thorin theorem. We explicitly state the two relevant special cases of the theorem that we require in the next section for the convenience of the reader.

▶ **Theorem 4.2** (Special cases of the Riesz-Thorin theorem, see [36, Theorem 1.3]). *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ be a matrix with non-negative entries. For any $p \in (1,\infty)$ it holds that:*

$$\|\boldsymbol{A}\|_{p\to p} \leq \|\boldsymbol{A}\|_{1\to 1}^{\frac{1}{p}} \cdot \|\boldsymbol{A}\|_{\infty\to\infty}^{1-\frac{1}{p}}.$$

Furthermore, for $p \in (2,\infty)$,

$$\|\boldsymbol{A}\|_{p\to p} \leq \|\boldsymbol{A}\|_{2\to 2}^{\frac{2}{p}} \cdot \|\boldsymbol{A}\|_{\infty\to\infty}^{1-\frac{2}{p}}.$$

Proof of Theorem 1.2

We have that the theorem already holds for ℓ_1 and ℓ_∞ since we have proven Theorem 1.1 in the previous section. Consider therefore any $p \in (1,\infty)$. Then, we have

$$\begin{aligned}
\rho_p(\boldsymbol{A}_{\mathcal{E}}) &\overset{\text{Lemma 4.1}}{=} \||\boldsymbol{A}_{\mathcal{E}}\boldsymbol{B}|\|_{p\to p} \\
&\overset{\text{Theorem 4.2}}{\leq} \||\boldsymbol{A}_{\mathcal{E}}\boldsymbol{B}|\|_{1\to 1}^{\frac{1}{p}} \cdot \||\boldsymbol{A}_{\mathcal{E}}\boldsymbol{B}|\|_{\infty\to\infty}^{1-\frac{1}{p}} \\
&\overset{\text{Lemma 4.1}}{=} (\rho_1(\boldsymbol{A}_{\mathcal{E}}))^{\frac{1}{p}} \cdot (\rho_\infty(\boldsymbol{A}_{\mathcal{E}}))^{1-\frac{1}{p}} \\
&\overset{\text{Theorem 1.1}}{\leq} \left(3\cdot\frac{\ln(2m)}{\Phi}\right)^{\frac{1}{p}} \cdot \left(3\cdot\frac{\ln(2m)}{\Phi}\right)^{1-\frac{1}{p}} \\
&= 3\cdot\frac{\ln(2m)}{\Phi}.
\end{aligned}$$

Proof of Corollary 1.3

Consider next any $p \in (1,\infty)$. Since we have again that for q given by $1/p + 1/q = 1$, we have $\|\boldsymbol{X}\|_{p\to p} = \|\boldsymbol{X}^\top\|_{q\to q}$, we can assume w.l.o.g. that $p \geq 2$. Similarly to [21], we obtain

$$\begin{aligned}
\rho_p(\boldsymbol{A}_{\mathcal{E}}) &\overset{\text{Lemma 4.1}}{=} \||\boldsymbol{A}_{\mathcal{E}}\boldsymbol{B}|\|_{p\to p} \\
&\overset{\text{Theorem 4.2}}{\leq} \||\boldsymbol{A}_{\mathcal{E}}\boldsymbol{B}|\|_{2\to 2}^{\frac{2}{p}} \cdot \||\boldsymbol{A}_{\mathcal{E}}\boldsymbol{B}|\|_{\infty\to\infty}^{1-\frac{2}{p}} \\
&\overset{\text{Lemma 4.1}}{=} (\rho_2(\boldsymbol{A}_{\mathcal{E}}))^{\frac{2}{p}} \cdot (\rho_\infty(\boldsymbol{A}_{\mathcal{E}}))^{1-\frac{2}{p}} \\
&\overset{\text{Lemma 4.1, Theorem 1.1}}{\leq} (\||\Pi|\|_{2\to 2})^{\frac{2}{p}} \cdot \left(3\cdot\frac{\ln(2m)}{\Phi}\right)^{1-\frac{2}{p}}.
\end{aligned}$$

5 A Lower Bound for Competitive Ratio of Electric Flow Routing

Finally, in this section, we provide a strong lower bound on the competitive ratio of the electrical routing scheme in any ℓ_p-norm.

▶ **Theorem 5.1** (Restatement of Theorem 1.4). *For an infinite number of positive integers n and any $\Phi \in [1/\sqrt[3]{n}, 1]$, for any $p \in [2,\infty]$ and q given by $1/p + 1/q = 1$, we have that*

$$\rho_p(\boldsymbol{A}_{\mathcal{E}}), \rho_q(\boldsymbol{A}_{\mathcal{E}}) \geq \Omega\left(\frac{\log m}{\Phi^{1-2/p}}\right).$$

In our proof, we use the following theorem given in [2]. We remind the reader that the *girth* of a graph G is the weight of the smallest weight cycle of G.

▶ **Theorem 5.2** ([2, Theorem 1.2]). *There are infinitely many positive integer Δ and n, for which an n-vertex unweighted graph $G_{\Delta,n} = (V,E)$ exists such that G is Φ_{const}-expander that is Δ-regular with $\Phi_{const} = \Theta(1)$ such that G has girth $\Omega(\log_\Delta n)$.*

In our proof, we use the existential result behind the statement to refine a proof technique previously used by Englert and Räcke [9] to give a lower bound on the competitive ratio of any ℓ_p oblivious routing scheme. Our refinement can also be used to strengthen their result by a $\Theta(\log \log n)$ factor.

In our proof, we crucially exploit the following facts about effective resistance. Recall that the effective resistance of a graph G for a pair $(s,t) \in V^2$ is the minimum energy required to route one unit of demand from s to t in G, or alternatively the difference in voltages of s and t induced by routing this unit of demand via an electrical flow which is given by $\chi_{(s,t)}\boldsymbol{L}^{+}\chi_{(s,t)}$. The facts below can be derived straightforwardly from Cheeger's Inequality, mixing of random walks, and characterization of effective resistance by commute times (see for example [19]).

▶ **Fact 5.3.** *For G being a constant-degree $\Omega(1)$-expander, we have that the effective resistance of any pair $(s,t) \in V$ is in $\Theta(1)$.*

▶ **Fact 5.4.** *Given two constant-degree graphs G and H over the same vertex set V. If the effective resistance for a pair $(s,t) \in V^2$ is in $\Theta(1)$ in both G and H, then the electrical flow routing one unit of demand from s to t on the union of graphs $G \cup H$ sends at least a constant fraction of the flow over G and a constant fraction of the flow over H.*

Let us now give a lower bound for any $p \geq 2$ and any parameter $\Phi \in [1/n, 1]$ such that $1/\Phi$ is integer. We start by considering the electrical routing $\boldsymbol{A}_{\mathcal{E}}$ for a large constant Δ and any n and $G_{\Delta,n}$ of the multi-commodity demand that is given by routing for each edge $e = (u,v)$ in $G_{\Delta,n}$ one unit of a commodity from u to v, i.e. we consider the demand $\boldsymbol{\chi} = \left\{\boldsymbol{\chi}_{(u,v)}\right\}_{e=(u,v)\in E(G_{\Delta,n})}$. Towards understanding the electrical routing, we prove the following simple claim.

▷ Claim 5.5. For any edge e in $G_{\Delta,n}$ where Δ is a large constant, we have that the electrical flow $\boldsymbol{f} = \boldsymbol{A}_{\mathcal{E}}\boldsymbol{\chi}_{(u,v)}$ routing the demand $\boldsymbol{\chi}_{(u,v)}$ has $\|\boldsymbol{f}\|_1 = \Omega(\log n)$.

Proof. The claim follows from showing that $\boldsymbol{f}(e)$ carries only $(1-\varepsilon)$ units of flow for some constant $\varepsilon > 0$. This is because it implies that a constant fraction of the flow is not routed via the edge e. But since each path between the endpoints of e that does not use the edge e is of length $\Omega(\log n)$ (by the girth bound in Theorem 5.2), we have that this ε-fraction adds $\Omega(\varepsilon \log n) = \Omega(\log n)$ units of flow to the network $G_{\Delta,n}$.

To prove the claim, it suffices to observe that the graph $G_{\Delta,n} \setminus \{e\}$ is a $\Omega(1)$-expander. But to this end, it suffices to observe that since the conductance of $G_{\Delta,n}$ does not depend on Δ by Fact 5.3, by choosing Δ sufficiently large (i.e. at least twice the inverse of the conductance), we have that each cut contains at least 2 edges and thus the conductance of $G_{\Delta,n} \setminus \{e\}$ is at least half of the conductance of $G_{\Delta,n}$, and thus still constant.

Using that the trivial graph consisting only of the edge e is a constant-degree $\Omega(1)$-expander, we thus have that the effective resistance of the pair (u,v) in both the graph $G_{\Delta,n} \setminus e$ and e is constant by Fact 5.3. Thus, by Fact 5.4, we have that a constant fraction of the demand $\boldsymbol{\chi}_{(u,v)}$ is not routed into e, as desired. ◁

Using that multi-commodity flows do not cancel, we thus have that each edge in $G_{\Delta,n}$ carries on average $\Omega(\log n)$ units of flow. We next transform the graph $G_{\Delta,n}$ to then obtain our final gadget on which we can prove the lower bound.

▶ **Definition 5.6.** *Let $G^{\Phi}_{\Delta,n}$ be the graph obtained from $G_{\Delta,n}$ by replacing each edge with $1/\Phi$ vertex-disjoint paths of length $1/\Phi$ between the endpoints of the vertices. Thus, $G^{\Phi}_{\Delta,n}$, for Δ being a constant, has $\Theta(n/\Phi^2)$ vertices.*

Next, we claim that the effective resistance of our demand pairs is the same up to a constant in $G_{\Delta,n}$ and $G^{\Phi}_{\Delta,n}$.

▷ Claim 5.7. For each edge $(u,v) \in E(G_{\Delta,n})$, the effective resistance of the pair (u,v) in the graph $G^{\Phi}_{\Delta,n}$ is $\Theta(1)$.

Proof. To show this result, we give an explicit mapping of the electrical flow routing $\boldsymbol{\chi_{(u,v)}}$ in $G_{\Delta,n}$ to routing the flow in $G^{\Phi}_{\Delta,n}$ whose energy is at most constant. Let $\boldsymbol{f}$ be this electrical flow routing on $G_{\Delta,n}$, then we map the flow on each edge e' in $G_{\Delta,n}$ uniformly through the $1/\Phi$ disjoint paths between the endpoints of e' in $G^{\Phi}_{\Delta,n}$. Since each path now routes only a Φ-fraction of the original flow on the edge e', we have that the energy used to route through each edge on the disjoint paths replacing e'_{middle} is $(\boldsymbol{f}(e')\Phi)^2 = \boldsymbol{f}(e')^2\Phi^2$. We thus have that the energy incurred by routing through the $1/\Phi$ disjoint paths each consisting of $1/\Phi$ edges is $1/\Phi^2 \cdot \boldsymbol{f}(e')^2\Phi^2 = \boldsymbol{f}(e')^2$. Thus, the effective resistance of (u,v) in $G^{\Phi}_{\Delta,n}$ is at most the resistance in $G_{\Delta,n}$ which implies it is in $O(1)$.

A lower bound of $\Omega(1)$ is observed by inversing this mapping to collect the amount of flow pushed through the disjoint paths replacing edge e' together and adding it to e' in $G_{\Delta,n}$. The proof is straightforward and therefore omitted. ◁

Before we can carry out the proof of our lower bound, it remains to show for our lower bound gadget which is the graph $G = G_{\Delta,n} \cup G^{\Phi}_{\Delta,n}$ that it is a $\Theta(\Phi)$-expander.

▷ Claim 5.8. G is $\Theta(\Phi)$-expander with $\Theta(n/\Phi^2)$ edges.

Proof. The number of edges is straightforward from our construction of G. To see that G is $O(\Phi)$-expander, observe that we can take the internal vertices of any path in $G^{\Phi}_{\Delta,n}$ replacing an edge in $G_{\Delta,n}$ which has volume $\Omega(1/\Phi)$ but only two edges leaving (the once to the endpoints of the replaced edges). To observe that it is an $\Omega(\Phi)$-expander, it suffices to show that each cut in S is maximized by assigning all internal vertices of each such path to one side of the cut. It is then not hard to show from $G_{\Delta,n}$ being a $\Omega(1)$-expander that the claim follows. ◁

Let us now give the proof of the main result. We take the graph under consideration to be $G = G_{\Delta,n} \cup G^{\Phi}_{\Delta,n}$. We take as demand, the vector $\boldsymbol{\chi}^{\Phi} = \frac{1}{\Phi} \cdot \boldsymbol{\chi} = \frac{1}{\Phi} \cdot \{\boldsymbol{\chi}_{(u,v)}\}_{e=(u,v)\in E(G_{\Delta,n})}$. Let $\boldsymbol{A_{\mathcal{E}}}$ denote the electrical flow routing on this graph G. Let us look at each edge $e = (u,v) \in E(G_{\Delta,n})$. From Fact 5.3, Claim 5.7 and Fact 5.4, we have that the flow $\boldsymbol{f}_e = \boldsymbol{A_{\mathcal{E}}} \cdot \frac{1}{\Phi}\boldsymbol{\chi}_{(u,v)}$ restricted to the edges in $E(G_{\Delta,n})$ routes in total at least $\log n/\varphi$ units of flow along all of these edges. By linearity of $\boldsymbol{A_{\mathcal{E}}}$ and the fact that flows do not cancel, we have that when routing $\boldsymbol{\chi}^{\Phi}$, an average edge in $E(G_{\Delta,n})$ carries $\Omega(\log n/\Phi)$ units of flow. Thus, the ℓ_p-norm of this flow is at least $\sqrt[p]{n \cdot (\log n/\Phi)^p} = n^{1/p} \cdot \log n \cdot \Phi$. But observe that we can route the flow with demand with congestion 1 in $G^{\Phi}_{\Delta,n}$ by routing for each demand $\boldsymbol{\chi}_{(u,v)}$ exactly 1 unit of flow through each of the disjoint paths corresponding to the edge $e' = (u,v)$ in $G^{\Phi}_{\Delta,n}$. The ℓ_p-norm of this flow is $\Theta((n/\Phi^2)^{1/p}) = \Theta(n^{1/p}\Phi^{-2/p})$ (using Claim 5.8). We thus have that $\rho_p(\boldsymbol{A_{\mathcal{E}}}) = \Omega(\log n \cdot \Phi^{p/(p-2)})$.

To obtain the result for $p < 2$, we use that for q given by $1/p + 1/q = 1$, we have $\|\boldsymbol{X}\|_{p\to p} = \|\boldsymbol{X}^{\top}\|_{q\to q}$, and for the electrical routing, $\boldsymbol{A_{\mathcal{E}}B} = (\boldsymbol{A_{\mathcal{E}}B})^{\top}$ since $\boldsymbol{L}^{+}$ is symmetric.

We note that in the construction above the number of vertices in the final graph G might be much larger than n. By considering all possible parameters for Φ in $[1/n, 1]$ (i.e. all such numbers were $1/\Phi$ is integer), we obtain a family of n'-vertex graphs with conductances in $[1, 1/\sqrt[3]{n'}]$, as claimed. Since every Φ-expander is also a Φ'-expander for every $\Phi' \leq \Phi$, we do further not need to restrict the domain of Φ' further than in range. We point out that by considering parameters Φ in our construction that are even smaller than $1/n$, one can get up to an arbitrarily small polynomial factor close to conductances as small as $1/\sqrt{n'}$.

References

1 N Alon and V. D Milman. Λ1, Isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory, Series B*, 38(1):73–88, February 1985. doi:10.1016/0095-8956(85)90092-9.

2 Noga Alon, Shirshendu Ganguly, and Nikhil Srivastava. High-girth near-ramanujan graphs with localized eigenvectors. *Israel Journal of Mathematics*, 246(1):1–20, 2021.

3 Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Racke. Optimal oblivious routing in polynomial time. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, pages 383–388, 2003.

4 Yair Bartal and Stefano Leonardi. On-line routing in all-optical networks. In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Automata, Languages and Programming*, Lecture Notes in Computer Science, pages 516–526, Berlin, Heidelberg, 1997. Springer. doi:10.1007/3-540-63165-8_207.

5 Jeff Cheeger. A lower bound for the smallest eigenvalue of the Laplacian, Problems in analysis (Papers dedicated to Salomon Bochner, 1969), 1970.

6 Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, and Maximilian Probst Gutenberg. Almost-linear time algorithms for incremental graphs: Cycle detection, sccs, s-t shortest path, and minimum-cost flow. *CoRR*, abs/2311.18295, 2023. To appear at STOC'24. doi:10.48550/arXiv.2311.18295.

7 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum Flow and Minimum-Cost Flow in Almost-Linear Time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623, October 2022. doi:10.1109/FOCS54457.2022.00064.

8 Michael B Cohen, Rasmus Kyng, Gary L Miller, Jakub W Pachocki, Richard Peng, Anup B Rao, and Shen Chen Xu. Solving sdd linear systems in nearly m log1/2 n time. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 343–352, 2014.

9 Matthias Englert and Harald Räcke. Oblivious routing for the Lp-norm. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 32–40. IEEE, 2009.

10 Mohsen Ghaffari, Bernhard Haeupler, and Goran Zuzic. Hop-constrained oblivious routing. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 1208–1220, New York, NY, USA, June 2021. Association for Computing Machinery. doi:10.1145/3406325.3451098.

11 Gramoz Goranci, Monika Henzinger, Harald Räcke, Sushant Sachdeva, and A. R. Sricharan. Electrical flows for polylogarithmic competitive oblivious routing, 2023. arXiv:2303.02491.

12 Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2212–2228. SIAM, 2021.

13 Bernhard Haeupler, Harald Räcke, and Mohsen Ghaffari. Hop-constrained expander decompositions, oblivious routing, and distributed universal optimality. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 1325–1338, New York, NY, USA, June 2022. Association for Computing Machinery. doi:10.1145/3519935.3520026.

14 Arun Jambulapati and Aaron Sidford. Ultrasparse ultrasparsifiers and faster laplacian system solvers. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 540–559. SIAM, 2021.

15 Jonathan Kelner and Petar Maymounkov. Electric routing and concurrent flow cutting. *Theoretical computer science*, 412(32):4123–4135, 2011.

16 Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations, 2013. arXiv:1304.2338.

17 Jonathan A Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 911–920, 2013.

18 Rasmus Kyng, Richard Peng, Sushant Sachdeva, and Di Wang. Flows in almost linear time via adaptive preconditioning. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 902–913, 2019.

19 Rasmus Kyng and Maximilian Probst. Advanced graph algorithms and optimization, 2021.

20 Rasmus Kyng and Sushant Sachdeva. Approximate gaussian elimination for laplacians-fast, sparse, and simple. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 573–582. IEEE, 2016.

21 Gregory Lawler and Hariharan Narayanan. Mixing times and ℓ_p bounds for oblivious routing. In *2009 Proceedings of the Sixth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 66–74. SIAM, 2009.

22 Huan Li and Aaron Schild. Spectral subspace sparsification. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 385–396. IEEE, 2018.

23 Jason Li. Faster parallel algorithm for approximate shortest path. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 308–321, 2020.

24 Aleksander Madry. Fast Approximation Algorithms for Cut-Based Problems in Undirected Graphs. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 245–254, October 2010. doi:10.1109/FOCS.2010.30.

25 B.M. Maggs, F. Meyer auf der Heide, B. Vocking, and M. Westermann. Exploiting locality for data management in systems of limited bandwidth. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 284–293, October 1997. doi:10.1109/SFCS.1997.646117.

26 Richard Peng. Approximate undirected maximum flows in o (m polylog (n)) time. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1862–1867. SIAM, 2016.

27 H. Racke. Minimizing congestion in general networks. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 43–52, November 2002. doi:10.1109/SFCS.2002.1181881.

28 Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, pages 255–264, 2008.

29 Harald Räcke, Chintan Shah, and Hanjo Täubig. Computing Cut-Based Hierarchical Decompositions in Almost Linear Time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 227–238. Society for Industrial and Applied Mathematics, January 2014. doi:10.1137/1.9781611973402.17.

30 Václav Rozhoň, Christoph Grunau, Bernhard Haeupler, Goran Zuzic, and Jason Li. Undirected (1+ ε)-shortest paths via minor-aggregates: near-optimal deterministic parallel and distributed algorithms. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 478–487, 2022.

31 Aaron Schild. An almost-linear time algorithm for uniform random spanning tree generation. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 214–227, 2018.

32 Aaron Schild, Satish Rao, and Nikhil Srivastava. Localization of electrical flows. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 1577–1584, USA, January 2018. Society for Industrial and Applied Mathematics.

33 Jonah Sherman. Nearly maximum flows in nearly linear time. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 263–269. IEEE, 2013.

34 Jonah Sherman. Generalized preconditioning and undirected minimum-cost flow. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 772–780. SIAM, 2017.

35 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 81–90, New York, NY, USA, June 2004. Association for Computing Machinery. doi:10.1145/1007352.1007372.

36 Elias M. Stein and Guido Weiss. *The Fourier Transform*, pages 1–36. Princeton University Press, 1971. URL: http://www.jstor.org/stable/j.ctt1bpm9w6.4.

37 L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 263–277, New York, NY, USA, 1981. Association for Computing Machinery. doi:10.1145/800076.802479.

38 Leslie G. Valiant. A scheme for fast parallel communication. *SIAM journal on computing*, 11(2):350–361, 1982.

39 Goran Zuzic, Gramoz Goranci, Mingquan Ye, Bernhard Haeupler, and Xiaorui Sun. Universally-optimal distributed shortest paths and transshipment via graph-based $_1$-oblivious routing. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 2549–2579. SIAM, 2022. doi:10.1137/1.9781611977073.100.

Problems in NP Can Admit Double-Exponential Lower Bounds When Parameterized by Treewidth or Vertex Cover

Florent Foucaud
Université Clermont Auvergne, CNRS, Mines Saint-Étienne, Clermont Auvergne INP, LIMOS, 63000 Clermont-Ferrand, France

Esther Galby
Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden
University of Gothenburg, Sweden

Liana Khazaliya
Technische Universität Wien, Austria

Shaohua Li
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Fionn Mc Inerney
Technische Universität Wien, Austria

Roohani Sharma
University of Bergen, Norway

Prafullkumar Tale
Indian Institute of Science Education and Research Pune, India

Abstract

Treewidth serves as an important parameter that, when bounded, yields tractability for a wide class of problems. For example, graph problems expressible in Monadic Second Order (MSO) logic and Quantified SAT or, more generally, Quantified CSP, are fixed-parameter tractable parameterized by the treewidth of the input's (primal) graph plus the length of the MSO-formula [Courcelle, Information & Computation 1990] and the quantifier rank [Chen, ECAI 2004], respectively. The algorithms generated by these (meta-)results have running times whose dependence on treewidth is a tower of exponents. A conditional lower bound by Fichte, Hecher, and Pfandler [LICS 2020] shows that, for Quantified SAT, the height of this tower is equal to the number of quantifier alternations. These types of lower bounds, which show that at least *double-exponential* factors in the running time are *necessary*, exhibit the extraordinary level of computational hardness for such problems, and are rare in the current literature: there are only a handful of such lower bounds (for treewidth and vertex cover parameterizations) and all of them are for problems that are #NP-complete, Σ_2^p-complete, Π_2^p-complete, or complete for even higher levels of the polynomial hierarchy.

Our results demonstrate, for the first time, that it is not necessary to go higher up in the polynomial hierarchy to achieve double-exponential lower bounds: we derive double-exponential lower bounds in the treewidth (tw) and the vertex cover number (vc), for natural, important, and well-studied NP-*complete* graph problems. Specifically, we design a *technique* to obtain such lower bounds and show its *versatility* by applying it to three different problems: Metric Dimension, Strong Metric Dimension, and Geodetic Set. We prove that these problems do not admit $2^{2^{o(\mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$-time algorithms, even on bounded diameter graphs, unless the ETH fails (here, n is the number of vertices in the graph). In fact, for Strong Metric Dimension, the double-exponential lower bound holds even for the vertex cover number. We further complement all our lower bounds with matching (and sometimes non-trivial) upper bounds.

For the conditional lower bounds, we design and use a novel, yet simple technique based on *Sperner families* of sets. We believe that the amenability of our technique will lead to obtaining such lower bounds for many other problems in NP.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 66; pp. 66:1–66:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized Complexity, ETH-based Lower Bounds, Double-Exponential Lower Bounds, Kernelization, Vertex Cover, Treewidth, Diameter, Metric Dimension, Strong Metric Dimension, Geodetic Sets

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.66

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2307.08149

Funding *Florent Foucaud*: ANR project GRALMECO (ANR-21-CE48-0004), French government IDEX-ISITE initiative 16-IDEX-0001 (CAP 20-25), International Research Center "Innovation Transportation and Production Systems" of the I-SITE CAP 20-25.
Liana Khazaliya: Vienna Science and Technology Fund (WWTF) [10.47379/ICT22029]; Austrian Science Fund (FWF) [Y1329]; European Union's Horizon 2020 COFUND programme [LogiCS@TUWien, grant agreement No. 101034440].
Shaohua Li: European Research Council (ERC) consolidator grant No. 725978 SYSTEMATIC-GRAPH.
Fionn Mc Inerney: Austrian Science Fund (FWF, project Y1329).

1 Introduction

Many interesting computational problems turn out to be intractable. In these cases, identifying parameters under which the problems become tractable is desirable. In the area of parameterized complexity, treewidth is a cornerstone parameter since a large class of problems become tractable on graphs of bounded treewidth.

Courcelle's celebrated theorem [13] states that the class of graph problems expressible in Monadic Second-Order Logic (MSOL) of constant size is fixed-parameter tractable (FPT) when parameterized by the treewidth of the graph. That is, such problems admit algorithms whose running time is of the form $f(\mathtt{tw}) \cdot \mathsf{poly}(n)$, where $\mathtt{tw}$ is the treewidth of the input, n is the size of the input, and f is a function that depends only on $\mathtt{tw}$. Similarly, a result by Chen [12] shows that the Quantified SAT (Q-SAT) problem can also be solved in time $f(\mathtt{tw}) \cdot \mathsf{poly}(n)$, where $\mathtt{tw}$ is the treewidth of the primal graph of the input formula and f is a function that depends only on $\mathtt{tw}$ and the number of quantifier alternations in the input formula. Q-SAT is a generalization of SAT that allows universal and existential quantifications over the variables. Note that Q-SAT with k quantifier alternations is Π_k^p-complete or Σ_k^p-complete. Unfortunately, in both of the aforementioned results, the function f is a tower of exponents whose height depends roughly on the size of the MSOL and input formulas, respectively. For Q-SAT, the height of this tower equals the number of quantifier alternations in the Q-SAT instance [12].

Over the years, the focus shifted to making such FPT algorithms as efficient as possible. Thus, a natural question is to ask when this higher-exponential dependence on treewidth is necessary. There is a rich literature that provides (conditional) lower bounds on this dependency for many problems, and these bounds are commonly of the form $2^{o(\mathtt{tw})}$ or, in some unusual cases, $2^{o(\mathtt{tw}\log\mathtt{tw})}$ (e.g., [15, 50]) and even $2^{o(\mathsf{poly}(\mathtt{tw}))}$ (e.g., [14, 55]). Most notably, these lower bounds are far from the tower of exponents upper bounds given by the (meta-)results discussed above. In this work, we develop a simple technique that allows to prove double-exponential dependence on the treewidth $\mathtt{tw}$ and the vertex cover number $\mathtt{vc}$,

two of the most fundamental graph parameters. Notably, these are the first such results for problems in NP, and we believe that the amenability of our technique will lead to many more similar results for other problems in NP.

Indeed, after a preprint of this paper appeared on arxiv, our technique was also used to prove double-exponential dependence on vc for an NP-complete machine learning problem [11] and double-exponential dependence on the solution size and tw for NP-complete *identification problems* like Test Cover and Locating-Dominating Set [10].

Double-exponential lower bounds: treewidth and vertex cover parameterizations. Fichte, Hecher, and Pfander [25] recently proved that, assuming the Exponential Time Hypothesis[1] (ETH), Q-SAT with k quantifier alternations cannot be solved in time significantly better than a tower of exponents of height k in the treewidth. This exemplifies an interesting but expected trait of this problem: its complexity, in terms of the height of the exponential tower in tw, increases with each quantifier alternation. It strengthened the result that appeared in [48], where conditional double-exponential lower bounds for $\exists\forall$SAT and $\forall\exists$SAT were given. The results in [48] also yield a double-exponential lower bound in vc of the primal graph for both problems. Besides these results, there are only a handful of other problems known to require higher-exponential dependence in the treewidth of the input graph (or the primal graph of the input formula). Specifically, the Π_2^p-complete k-Choosability problem and the Σ_3^p-complete k-Choosability Deletion problem admit a double-exponential and a triple-exponential lower bound in treewidth [52], respectively. Recently, the Σ_p^2-complete problems Cycle HitPack and H-HitPack, for a fixed graph H, were shown to admit tight algorithms that are double-exponential in the treewidth [26]. Further, the Σ_p^2-complete problem Core Stability was shown to admit a tight double-exponential lower bound in the treewidth, even on graphs of bounded degree [32]. Lastly, the #NP-complete counting problem Projected Model Counting admits a double-exponential lower bound in tw [23, 24]. For other double-exponential lower bounds, see [1, 16, 27, 32, 37, 41, 43, 44, 47, 51, 56, 59].

All the double- (or higher) exponential lower bounds in treewidth mentioned so far are for problems that are #NP-complete, Σ_2^p-complete, Π_2^p-complete, or complete for even higher levels of the polynomial hierarchy. To quote [52]: *"Π_2^p-completeness of these problems already gives sufficient explanation why double- [...] exponential dependence on treewidth is needed. [...] the quantifier alternations in the problem definitions are the common underlying reasons for being in the higher levels of the polynomial hierarchy and for requiring unusually large dependence on treewidth."*

As mentioned above, we develop a technique that allows to demonstrate, for the first time, that it is not necessary to go to higher levels of the polynomial hierarchy to achieve double-exponential lower bounds in the treewidth or the vertex cover number of the graph.

We prove that three natural and well-studied NP-complete problems admit double-exponential lower bounds in tw or vc, under the ETH. These are the first problems in NP known to admit such lower bounds.[2]

[1] The Exponential Time Hypothesis roughly states that n-variable 3-SAT cannot be solved in time $2^{o(n)}$.

[2] While it may be possible to artificially engineer a graph problem or graph representation of a problem in NP that admits such lower bounds (although, to the best of our knowledge, this has not been done), we emphasize that this is not the case for these three natural and well-established graph problems in NP.

NP-complete metric-based graph problems. We study three *metric-based graph problems*. These problems are Metric Dimension [34, 58], Strong Metric Dimension [57], and Geodetic Set [33], and they arise from network design and network monitoring. Apart from serving as examples for double-exponential dependence on treewidth and the amenability of our technique, these problems are of interest in their own right, and possess a rich literature both in the algorithms and discrete mathematics communities (see, e.g., [2, 3, 4, 5, 7, 8, 17, 19, 20, 21, 31, 38, 45, 53] and the references below). Their non-local nature has posed interesting algorithmic challenges and our results, as we explain later, supplement the already vast literature on the structural parameterizations of these problems. Below we define the three above-mentioned problems formally, and particularly focus on Metric Dimension as it is the most popular and well-studied of the three.

Metric Dimension
Input: A graph G and a positive integer k.
Question: Does there exist $S \subseteq V(G)$ such that $|S| \leq k$ and, for any pair of vertices $u, v \in V(G)$, there exists a vertex $w \in S$ with $d(w, u) \neq d(w, v)$?

The Metric Dimension problem dates back to the 70s [34, 58]. As in geolocation problems, the aim is to distinguish the vertices of a graph via their distances to a solution set. Metric Dimension was first shown to be NP-complete in general graphs in Garey and Johnson's book [30, GT61], and this was later extended to many restricted graph classes [18, 22, 28], including graphs of diameter 2 [28] and graphs of pathwidth 24 [49]. In a seminal paper, Metric Dimension was proven to be W[2]-hard parameterized by the solution size k, even in subcubic bipartite graphs [35]. This drove the subsequent meticulous study of the problem under structural parameterizations.

In particular, the complexity of Metric Dimension parameterized by treewidth remained an intriguing open problem for a long time. Recently, it was shown that Metric Dimension is para-NP-hard parameterized by pathwidth (`pw`) [49] (an earlier result [6] showed that it is W[1]-hard for pathwidth). A subsequent paper showed that the problem is W[1]-hard parameterized by the combined parameter feedback vertex set number (`fvs`) plus pathwidth of the graph [29].

We conclude this part with the definitions of the remaining two problems, both of which are known to be NP-Complete [9, 54]. Geodetic Set is also W[1]-hard parameterized by the solution size, feedback vertex set number, and pathwidth, combined [39].

Strong Metric Dimension
Input: A graph G and a positive integer k.
Question: Does there exist $S \subseteq V(G)$ such that $|S| \leq k$ and, for any pair of vertices $u, v \in V(G)$, there exists a vertex $w \in S$ such that either u lies on some shortest path between v and w, or v lies on some shortest path between u and w?

Geodetic Set
Input: A graph G and a positive integer k.
Question: Does there exist $S \subseteq V(G)$ such that $|S| \leq k$ and, for any vertex $u \in V(G)$, there are two vertices $s_1, s_2 \in S$ such that a shortest path from s_1 to s_2 contains u?

Our technical contributions. As Metric Dimension and Geodetic Set are NP-complete on bounded diameter graphs *or* on bounded treewidth graphs, we study their parameterized complexity with $\mathtt{tw} + \mathtt{diam}$ as the parameter and prove the following results.

1. Metric Dimension and Geodetic Set do not admit algorithms running in time $2^{f(\mathtt{diam})^{o(\mathtt{tw})}} \cdot n^{\mathcal{O}(1)}$, for any computable function f, unless the ETH fails.
2. Strong Metric Dimension does not even admit an algorithm with a running time of $2^{2^{o(\mathtt{vc})}} \cdot n^{\mathcal{O}(1)}$, unless the ETH fails. This also implies the problem does not admit a kernelization algorithm that outputs an instance with $2^{o(\mathtt{vc})}$ *vertices*, unless the ETH fails.

The above lower bounds for $\mathtt{tw}+\mathtt{diam}$, in particular, imply that Metric Dimension and Geodetic Set on graphs of bounded diameter cannot admit $2^{2^{o(\mathtt{tw})}} \cdot n^{\mathcal{O}(1)}$-time algorithms, unless the ETH fails. The reduction for Metric Dimension (sketched in Section 2.3) also works for fvs and td, and the one for Geodetic Set (Section 3) also works for td.

We show that all our lower bounds are tight by providing algorithms (kernelization algorithms, respectively) with matching running times (guarantees, respectively).

1. Metric Dimension and Geodetic Set admit algorithms running in time $2^{\mathtt{diam}^{\mathcal{O}(\mathtt{tw})}} \cdot n^{\mathcal{O}(1)}$.
2. Strong Metric Dimension admits an algorithm running in time $2^{2^{\mathcal{O}(\mathtt{vc})}} \cdot n^{\mathcal{O}(1)}$ and a kernel with $2^{\mathcal{O}(\mathtt{vc})}$ vertices.

The (kernelization) algorithm for the vc parameterization is very simple, whereas the algorithms for the $\mathtt{tw}+\mathtt{diam}$ parameter are highly non-trivial and require showing interesting locality properties in the instance. Further, for our $\mathtt{tw}+\mathtt{diam}$ parameterized algorithms, the (double-exponential) dependency of treewidth in the running time is unusual (and rightly so, as exhibited by our lower bounds), as most natural graph problems in NP for which a dedicated algorithm (i.e., *not* relying on Courcelle's theorem) parameterized by treewidth is known, can be solved in time $2^{\mathcal{O}(\mathtt{tw})} \cdot n^{\mathcal{O}(1)}$, $2^{\mathcal{O}(\mathtt{tw}\cdot\log(\mathtt{tw}))} \cdot n^{\mathcal{O}(1)}$ or $2^{\mathcal{O}(\mathsf{poly}(\mathtt{tw}))} \cdot n^{\mathcal{O}(1)}$.

Finally, our reductions rely on a novel, yet simple technique based on *Sperner families* of sets that allows to encode particular SAT relations across large sets of variables and clauses into relatively small vertex-separators. As mentioned before, we believe that this technique is the key to obtaining such lower bound results for other problems in NP. In particular, as witnessed by our results, our technique has the additional features that it even allows to prove such lower bounds in very restricted cases, such as bounded diameter graphs, and is not specific to any one structural parameter, as it also works for, e.g., the feedback vertex set number and treedepth.

Due to space constraints, we cannot discuss all of our results in depth, and refer the reader to the full version for full proofs, formal details, and more related work. Nonetheless, we elaborate on our technique and present an overview of the results for Metric Dimension in the next section. Then, in Section 3, we present a formal proof for the lower bound for Geodetic Set. Finally, we conclude the paper in Section 4.

2 Technical Overview

In this section, we present an overview of our lower bound techniques. We first exhibit our technique to obtain the double-exponential lower bounds in its most general setting. Then, we continue with the problem-specific tools we developed that are required for the reductions.

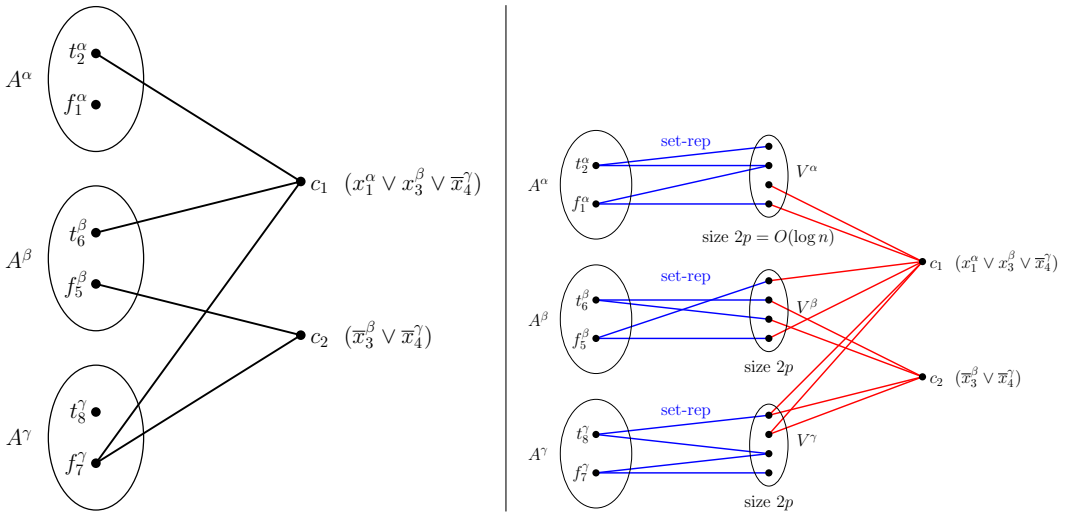


Figure 1 Graph representations of 3-Partitioned-3-SAT. (Left) incidence graph representation. (Right) representation with small separators using our technique. Note, for example, that x_1^α appears as a positive literal in the clause C_1. Thus, on the left, t_2^α is the only literal vertex in A^α incident to c_1, while on the right, t_2^α is the only literal vertex in A^α that does not share a common neighbor with c_1 in V^α. The edges from c_2 to each vertex in V^α are omitted for clarity.

2.1 General Technique for Double-Exponential Lower Bounds

The first integral part of our technique is to reduce from a variant of 3-SAT known as 3-Partitioned-3-SAT that was introduced in [46]. In this problem, the input is a formula ψ in 3-CNF form, together with a partition of the set of its variables into three disjoint sets X^α, X^β, X^γ, with $|X^\alpha| = |X^\beta| = |X^\gamma| = n$, and such that no clause contains more than one variable from each of X^α, X^β, and X^γ. The objective is to determine whether ψ is satisfiable. Unless the ETH fails, 3-Partitioned-3-SAT does not admit an algorithm running in time $2^{o(n)}$ [46, Theorem 3].

Typical reductions from satisfiability problems to graph problems usually entail representing the satisfiability problem by its incidence graph, in which each variable is represented by two vertices corresponding to its positive and negative literals. In this representation, a clause vertex is adjacent to a literal vertex if and only if it contains that literal in ψ (see Figure 1 (left) for an illustration). However, this naive approach does not lead to any structural parameters of the incidence graph being of bounded size. The *core idea* of our technique is to instead represent the relationships between clause and literal vertices via edges from these two sets of vertices to "small" separators (three separators in the case of 3-Partitioned-3-SAT) that encode these relationships.

Formally, this is achieved as follows. For a positive integer p, define $\mathcal{F}_p$ as the collection of subsets of $[2p]$ that contains exactly p integers. We critically use the fact that no set in $\mathcal{F}_p$ is contained in any other set in $\mathcal{F}_p$ (such a collection of sets are called a *Sperner family*). Let ℓ be a positive integer such that $\ell \leq \binom{2p}{p}$. We define $\mathsf{set\text{-}rep} : [\ell] \mapsto \mathcal{F}_p$ as a one-to-one function by arbitrarily assigning a set in $\mathcal{F}_p$ to an integer in $[\ell]$. By the asymptotic estimation of the central binomial coefficient, $\binom{2p}{p} \sim \frac{4^p}{\sqrt{\pi \cdot p}}$ [36]. To get the upper bound of p, we scale down the asymptotic function and have $\ell \leq \frac{4^p}{2^p} = 2^p$. Thus, $p = \mathcal{O}(\log \ell)$.

Let ψ be an instance of 3-Partitioned-3-SAT on $3n$ variables, and let p be the smallest integer such that $2n \leq \binom{2p}{p}$. In particular, $p = \mathcal{O}(\log n)$. Define $\mathsf{set\text{-}rep} : [2n] \mapsto \mathcal{F}_p$ as above. Rename the variables in X^α to x_i^α for all $i \in [n]$. For each variable x_i^α, add two vertices t_{2i}^α

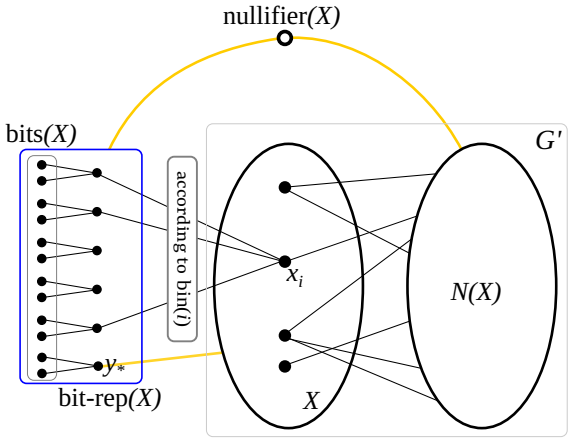


Figure 2 Set Identifying Gadget. The blue box represents bit-rep(X) and the yellow lines represent that nullifier(X) is adjacent to each vertex in bit-rep$(X) \cup N(X)$, and $y_\star$ is adjacent to each vertex in X. Also, G' is not necessarily restricted to the graph induced by the vertices in $X \cup N(X)$.

and f^{α}_{2i-1} corresponding to the positive and negative literals of x^{α}_i, respectively. Let $A^{\alpha} = \{t^{\alpha}_{2i}, f^{\alpha}_{2i-1} | \ i \in [n]\}$. Add a *validation portal* with $2p$ vertices, denoted by $V^{\alpha} = \{v^{\alpha}_1, \ldots, v^{\alpha}_{2p}\}$. For each $i \in [n]$, add the edge $t^{\alpha}_{2i}v^{\alpha}_{p'}$ for each $p' \in$ set-rep$(2i)$. Similarly, for each $i \in [n]$, add the edge $f^{\alpha}_{2i-1}v^{\alpha}_{p'}$ for each $p' \in$ set-rep$(2i-1)$. Repeat the above steps for β and γ.

Now, for each clause C_j ($j \in [m]$) in ψ, add a clause vertex c_j. Let $\delta \in \{\alpha, \beta, \gamma\}$. For all $i \in [n]$ and $j \in [m]$, if the variable x^{δ}_i appears as a positive (negative, respectively) literal in the clause C_j in ψ, then add the edge $c_j v^{\delta}_{p'}$ for each $p' \in [2p] \setminus$ set-rep$(2i)$ ($p' \in [2p] \setminus$ set-rep$(2i-1)$, respectively). For all $j \in [m]$, if no variable from X^{δ} appears in C_j in ψ, then make c_j adjacent to all the vertices in V^{δ}. See Figure 1 (right) for an illustration.

As a clause contains at most one variable from X^{δ} in ψ, c_j and t^{δ}_{2i} (f^{δ}_{2i-1}, respectively) do not share a common neighbor in V^{δ} if and only if the clause C_j contains x^{δ}_i as a positive (negative, respectively) literal in ψ. For the reductions, we use this representation of the relationship between clause and literal vertices. Since $p = \mathcal{O}(\log n)$, this ensures that $\mathtt{tw}(G) = \mathcal{O}(\log n)$, which we exploit along with the fact that, unless the ETH fails, 3-Partitioned-3-SAT does not admit an algorithm running in time $2^{o(n)}$.

2.2 Basic Tools for Lower Bounds

For brevity, we focus on Metric Dimension and explain our problem-specific tools in this context. We use two such simple tools: the *bit representation gadget* and the *set representation gadget*. The set representation gadget is the problem-specific implementation of the above technique, and it uses the bit representation gadget.

Before going further, we need to define some terms related to Metric Dimension. The set S defined in the problem statement of Metric Dimension is called a *resolving set* of G. A subset of vertices $S' \subseteq V(G)$ *resolves* a pair of vertices $u, v \in V(G)$ if there exists a vertex $w \in S'$ such that $d(w, u) \neq d(w, v)$. Lastly, a vertex $u \in V(G)$ is *distinguished* by a subset of vertices $S' \subseteq V(G)$ if, for any $v \in V(G) \setminus \{u\}$, there exists a vertex $w \in S'$ such that $d(w, u) \neq d(w, v)$.

Bit Representation Gadget to Identify Sets. Suppose we are given a graph G' and a subset $X \subseteq V(G')$ of its vertices. Further, suppose that we want to add a vertex set X^+ to G' to obtain a new graph G with the following properties. We want that each vertex in $X \cup X^+$ is

distinguished by vertices in X^+ that must be in any resolving set S of G, and that no vertex in X^+ can resolve any "critical pair" of vertices in G. Roughly, a pair of vertices is critical if it forces certain "types" of vertices to be in any resolving set S of G, and the selection of the specific vertices of those types depends on the solution to the problem being reduced from (which, in our case, is 3-PARTITIONED-3-SAT [46]). We refer to the graph induced by the vertices of X^+, along with the edges connecting X^+ to G', as the *Set Identifying Gadget* for the set X. Given a graph G' and a non-empty subset $X \subseteq V(G')$ of its vertices, to construct such a graph G, we add vertices and edges to G' as follows (see Figure 2):

- The vertex set X^+ that we are aiming to add is the union of a set $\mathsf{bit\text{-}rep}(X)$ and a special vertex denoted by $\mathsf{nullifier}(X)$.
- First, let $X = \{x_i \mid i \in [|X|]\}$, and set $q := \lceil \log(|X|+2) \rceil + 1$. We select this value for q to *(1)* uniquely represent each integer in $[|X|]$ by its bit-representation in binary (note that we start from 1 and not 0), *(2)* ensure that the only vertex whose bit-representation contains all 1's is $\mathsf{nullifier}(X)$, and *(3)* reserve one spot for an additional vertex $y_\star$.
- For every $i \in [q]$, add three vertices y_i^a, y_i, y_i^b, and add the path (y_i^a, y_i, y_i^b).
- Add 3 vertices $y_\star^a, y_\star, y_\star^b$ and the path $(y_\star^a, y_\star, y_\star^b)$. Add edges to make $\{y_i \mid i \in [q]\} \cup \{y_\star\}$ a clique. Make $y_\star$ adjacent to each vertex in X. Let $\mathsf{bit\text{-}rep}(X) = \{y_i, y_i^a, y_i^b \mid i \in [q]\} \cup \{y_\star, y_\star^a, y_\star^b\}$ and denote its subset by $\mathsf{bits}(X) = \{y_i^a, y_i^b \mid i \in [q]\} \cup \{y_\star^a, y_\star^b\}$.
- For every integer $j \in [|X|]$, let $\mathsf{bin}(j)$ denote the binary representation of j using q bits. Connect x_j with y_i if the i^{th} bit (going from left to right) in $\mathsf{bin}(j)$ is 1.
- Add a vertex, denoted by $\mathsf{nullifier}(X)$, and connect it to each vertex in $\{y_i \mid i \in [q]\} \cup \{y_\star\}$.
- For every vertex $u \in V(G) \setminus (X \cup X^+)$ such that u is adjacent to some vertex in X, add an edge between u and $\mathsf{nullifier}(X)$. We add this vertex to ensure that vertices in $\mathsf{bit\text{-}rep}(X)$ do not resolve critical pairs in $V(G)$.

Set Representation Gadget. We define $\mathsf{set\text{-}rep} : [\ell] \mapsto \mathcal{F}_p$ as in Section 2.1, and recall that $p = \mathcal{O}(\log \ell)$. Suppose we have a "large" collection of vertices, say $A = \{a_1, a_2, \ldots, a_\ell\}$, and a "large" collection of critical pairs $C = \{\langle c_1^\circ, c_1^\star\rangle, \langle c_2^\circ, c_2^\star\rangle, \ldots, \langle c_m^\circ, c_m^\star\rangle\}$. Moreover, we are given an injective function $\phi : [m] \mapsto [\ell]$. The objective is to design a gadget such that only $a_{\phi(q)} \in A$ can resolve a critical pair $\langle c_q^\circ, c_q^\star\rangle \in C$ for any $q \in [m]$, while keeping the treewidth of this part of the graph of order $\mathcal{O}(\log(|A|))$. With this in mind, we do the following.

- Add vertices and edges to identify the set A and to add critical pairs in C (for each critical pair in C, both vertices share the same bit-representation in the Set Identifying Gadget for C).
- Add a *validation portal*, a clique on $2p$ vertices, denoted by $V = \{v_1, v_2, \ldots, v_{2p}\}$, and vertices and edges to identify it.
- For every $i \in [\ell]$ and for every $p' \in \mathsf{set\text{-}rep}(i)$, add the edge $(a_i, v_{p'})$.
- For every critical pair $\langle c_q^\circ, c_q^\star\rangle$, make c_q° adjacent to every vertex in V, and add every edge of the form $(c_q^\star, v_{p'})$ for $p' \in [2p] \setminus \mathsf{set\text{-}rep}(\phi(q))$. Note that the vertices in V that are indexed using integers in $\mathsf{set\text{-}rep}(\phi(q))$ are *not* adjacent with $c_q^\star$.

See Figure 3 for an illustration. Now, consider a critical pair $\langle c_q^\circ, c_q^\star\rangle$ and suppose $i = \phi(q)$.

- By the construction, $N(a_i) \cap N(c_q^\circ) \neq \emptyset$, whereas $N(a_i) \cap N(c_q^\star) = \emptyset$. Hence, a_i resolves the critical pair $\langle c_q^\circ, c_q^\star\rangle$ as $d(a_i, c_q^\circ) = 2$ and $d(a_j, c_q^\star) > 2$.
- For any other vertex in A, say a_j, $\mathsf{set\text{-}rep}(j) \setminus \mathsf{set\text{-}rep}(i)$ is a non-empty set. So, there are paths from a_j to c_q° and a_j to $c_q^\star$ through vertices in V with indices in $\mathsf{set\text{-}rep}(j) \setminus \mathsf{set\text{-}rep}(i)$. This implies that $d(a_j, c_q^\circ) = d(a_j, c_q^\star) = 2$ and a_j cannot resolve the pair $\langle c_q^\circ, c_q^\star\rangle$.

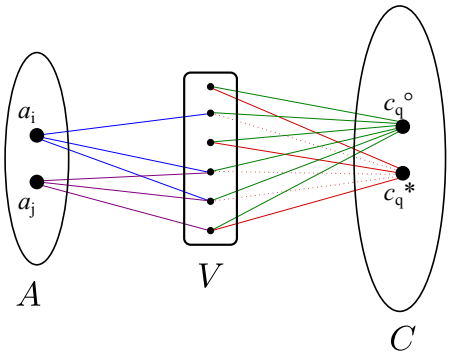


Figure 3 Set Representation Gadget. Let $\phi(q) = i$, i.e., only a_i in A can resolve the critical pair $\langle c_q^\circ, c_q^\star \rangle$. Let the vertices in V be indexed from top to bottom and let $\mathsf{set\text{-}rep}(i) = \{2, 4, 5\}$. By construction, the only vertices in V that $c_q^\star$ is *not* adjacent to are v_2, v_4, and v_5 (this is highlighted by red-dotted edges). Thus, $\mathrm{dist}(a_i, c_q^\circ) = 2$ and $\mathrm{dist}(a_i, c_q^\star) > 2$, and hence, a_i resolves $\langle c_q^\circ, c_q^\star \rangle$. For any other vertex in A, say a_j, $\mathsf{set\text{-}rep}(j) \setminus \mathsf{set\text{-}rep}(i)$ is non-empty, and thus, a_j cannot resolve $\langle c_q^\circ, c_q^\star \rangle$.

2.3 Sketch of the Lower Bound Proof for Metric Dimension

With these tools in hand, we present an overview of the reduction from 3-Partitioned-3-SAT used to prove Theorem 1, which we restate here for convenience.

▶ **Theorem 1.** *Unless the ETH fails,* Metric Dimension *does not admit an algorithm running in time* $2^{f(\mathtt{diam})^{o(\mathtt{tw})}} \cdot n^{\mathcal{O}(1)}$ *for any computable function* $f : \mathbb{N} \mapsto \mathbb{N}$.

The reduction in the proof of Theorem 1 takes as input an instance ψ of 3-Partitioned-3-SAT on $3n$ variables and returns (G, k) as an instance of Metric Dimension such that $\mathtt{tw}(G) = \mathcal{O}(\log(n))$ and $\mathtt{diam}(G) = \mathcal{O}(1)$. In the following, we mention a crude outline of the reduction, omitting some technical details.

2.3.1 Reduction

- We rename the variables in X^α to x_i^α for $i \in [n]$. For every variable x_i^α, we add a critical pair $\langle x_i^{\alpha,\circ}, x_i^{\alpha,\star} \rangle$ of vertices. We denote $X^\alpha = \{x_i^{\alpha,\circ}, x_i^{\alpha,\star} \mid i \in [n]\}$.
- For each variable x_i^α, we add the vertices t_{2i}^α and f_{2i-1}^α. Let $A^\alpha = \{t_{2i}^\alpha, f_{2i-1}^\alpha | \ i \in [n]\}$.
- For every $i \in [n]$, we add the edges $(x_i^{\alpha,\circ}, t_{2i}^\alpha)$ and $(x_i^{\alpha,\circ}, f_{2i-1}^\alpha)$ which will ensure that any resolving set contains at least one vertex in $\{t_{2i}^\alpha, f_{2i-1}^\alpha, x_i^{\alpha,\circ}, x_i^{\alpha,\star}\}$ for every $i \in [n]$.
- Let p be the smallest integer such that $2n \leq \binom{2p}{p}$. In particular, $p = \mathcal{O}(\log n)$. Define $\mathsf{set\text{-}rep} : [2n] \mapsto \mathcal{F}_p$ as in Section 2.1.
- We add a *validation portal*, a clique on $2p$ vertices, denoted by $V^\alpha = \{v_1^\alpha, v_2^\alpha, \ldots, v_{2p}^\alpha\}$.
- For each $i \in [n]$, we add the edge $(t_{2i}^\alpha, v_{p'}^\alpha)$ for every $p' \in \mathsf{set\text{-}rep}(2i)$. Similarly, for each $i \in [n]$, we add the edge $(f_{2i-1}^\alpha, v_{p'}^\alpha)$ for every $p' \in \mathsf{set\text{-}rep}(2i-1)$.

We repeat the above steps to construct $X^\beta, A^\beta, V^\beta$, $X^\gamma, A^\gamma, V^\gamma$.

- For every clause C_q in ψ, we introduce a pair $\langle c_q^\circ, c_q^\star \rangle$ of vertices. Let C be the collection of vertices in such pairs.
- We add edges across C and the portals as follows. Consider a clause C_q in ψ and the corresponding critical pair $\langle c_q^\circ, c_q^\star \rangle$ in C. Let $\delta \in \{\alpha, \beta, \gamma\}$. As ψ is an instance of 3-Partitioned-3-SAT, at most one variable in X^δ appears in C_q, say x_i^δ for some $i \in [n]$. We add all edges of the form $(v_{p'}^\delta, c_q^\circ)$ for every $p' \in [2p]$. If x_i^δ appears as a positive literal

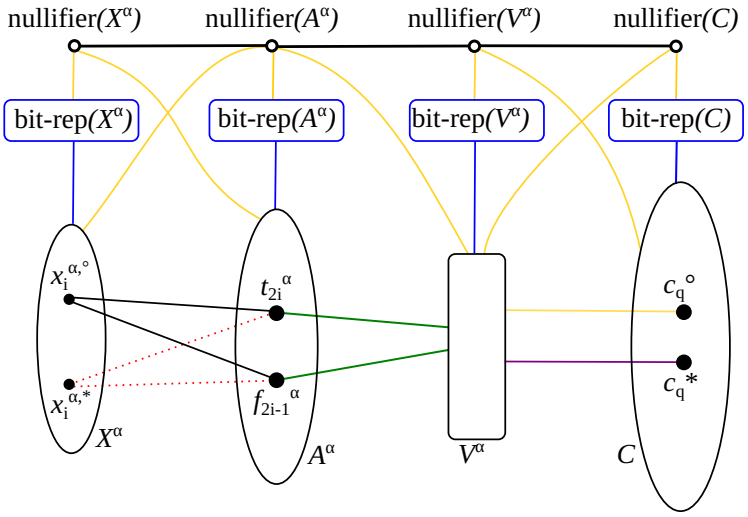


Figure 4 Reduction for proof of Theorem 1. Yellow lines represent that vertex is connected to every vertex in the set the edge goes to. Green edges denote adjacencies with respect to set-rep, e.g., t^{α}_{2i} is adjacent to $v_j \in V^{\alpha}$ if $j \in \textsf{set-rep}(2i)$. Purple lines also indicate adjacencies with respect to set-rep, but in a complementary way, i.e., if $x_i \in c_q$, then, for every $p' \in [2p] \setminus \textsf{set-rep}(2i)$, we have $(v^{\alpha}_{p'}, c^{\star}_q) \in E(G)$, and if $\overline{x}_i \in c_q$, then, for all $p' \in [2p] \setminus \textsf{set-rep}(2i-1)$, we have $(v^{\alpha}_{p'}, c^{\star}_q) \in E(G)$.

in C_q, then we add the edge $(v^{\delta}_{p'}, c^{\star}_q)$ for every $p' \in [2p] \setminus \textsf{set-rep}(2i)$ (which corresponds to t^{δ}_{2i}). If x^{δ}_i appears as a negative literal in C_q, then we add the edge $(v^{\delta}_{p'}, c^{\star}_q)$ for every $p' \in [2p] \setminus \textsf{set-rep}(2i-1)$ (which corresponds to f^{δ}_{2i-1}). Note that if x^{δ}_i appears as a positive (negative, respectively) literal in C_q, then the vertices in V^{δ} whose indices are in $\textsf{set-rep}(2i)$ ($\textsf{set-rep}(2i-1)$, respectively) are *not adjacent* to $c^{\star}_q$. If no variable in X^{δ} appears in C_q, then we make each vertex in V^{δ} adjacent to both c°_q and $c^{\star}_q$.

For all the sets mentioned above, we add vertices and edges to identify them as shown in Figure 4 (for each critical pair, both vertices share the same bit-representation in their Set Identifying Gadget). This concludes the construction of G. The reduction returns (G, k) as an instance of Metric Dimension for some appropriate value of k.

2.3.2 Correctness of the Reduction

We give an informal description of the proof of correctness of the reverse direction here. Fix $\delta \in \{\alpha, \beta, \gamma\}$. For all $i \in [n]$, the only vertices that can resolve the critical pair $\langle x^{\delta,\circ}_i, x^{\delta,\star}_i \rangle$ are the vertices in $\{x^{\delta,\circ}_i, x^{\delta,\star}_i\} \cup \{t^{\delta}_{2i}, f^{\delta}_{2i-1}\}$. This fact and the budget k ensure that any resolving set of G contains exactly one vertex from $\{t^{\delta}_{2i}, f^{\delta}_{2i-1}\} \cup \{x^{\delta,\circ}_i, x^{\delta,\star}_i\}$ for all $i \in [n]$. This naturally corresponds to an assignment of the variable x^{δ}_i if a vertex from $\{t^{\delta}_{2i}, f^{\delta}_{2i-1}\}$ is in the resolving set. However, if a vertex from $\{x^{\delta,\circ}_i, x^{\delta,\star}_i\}$ is in the resolving set, then we can see this as giving an arbitrary assignment to the variable x^{δ}_i. Suppose the clause C_q contains the variable x^{δ}_i as a positive literal. By the construction, every vertex in V^{δ} that is adjacent to t^{δ}_{2i} is *not* adjacent to $c^{\star}_q$. However, c°_q is adjacent to every vertex in V^{δ}. Hence, $d(t^{\delta}_{2i}, c^{\circ}_q) = 2$, whereas $d(t^{\delta}_{2i}, c^{\star}_q) > 2$. Thus, t^{δ}_{2i} resolves the critical pair $\langle c^{\circ}_q, c^{\star}_q \rangle$. Consider any other vertex in A^{δ}, say t^{δ}_{2j}. Since $\textsf{set-rep}(2i)$ is not a subset of $\textsf{set-rep}(2j)$ (as both have the same cardinality), there is at least one integer, say p', in $\textsf{set-rep}(2j) \setminus \textsf{set-rep}(2i)$. The vertex $v^{\delta}_{p'} \in V^{\delta}$ is adjacent to t^{δ}_{2j}, c°_q, and $c^{\star}_q$. Hence, t^{δ}_{2j} cannot resolve the critical pair $\langle c^{\circ}_q, c^{\star}_q \rangle$ as both these vertices are at distance 2 from it. Also, as ψ is an instance of

3-Partitioned-3-SAT, C_q contains at most one variable in X^δ, which is x_i^δ in this case. This also helps to encode the fact that at most one vertex from A^δ should be able to resolve the critical pair $\langle c_q^\circ, c_q^\star \rangle$. Since vertices in X^δ cannot resolve critical pairs $\langle c_q^\circ, c_q^\star \rangle$ in C, then finding a resolving set in G corresponds to finding a satisfying assignment for ψ.

2.3.3 Lower Bounds Obtained from the Reduction

Let $Z = \{V^\delta \cup X^+ \mid X \in \{X^\delta, A^\delta, V^\delta, C\}, \delta \in \{\alpha, \beta, \gamma\}\}$. Note that $|Z| = \mathcal{O}(\log(n))$ and $G - Z$ is a collection of P_3's and isolated vertices. Hence, $\mathtt{tw}(G)$, $\mathtt{fvs}(G)$, and $\mathtt{td}(G)$ are upper bounded by $\mathcal{O}(\log(n))$. Also, G has constant diameter. Thus, if there is an algorithm for Metric Dimension that runs in time $2^{f(\mathtt{diam})^{o(\mathtt{tw})}}$ (or $2^{f(\mathtt{diam})^{o(\mathtt{fvs})}}$ or $2^{f(\mathtt{diam})^{o(td)}}$), then there is an algorithm solving 3-Partitioned-3-SAT in time $2^{o(n)}$, contradicting the ETH.

2.4 High-Level Description of the Dynamic Programming Algorithm for Metric Dimension

The aim of this subsection is to give an informal description of how we prove the upper bound concerning the parameter $\mathtt{tw} + \mathtt{diam}$ for Metric Dimension. To this end, we give a dynamic programming algorithm on a tree decomposition for Metric Dimension. The algorithm is inspired by the one from [7] for chordal graphs, though there are some non-trivial differences. We will assume that a tree decomposition of the input graph G of width w is given to us. Note that one can compute a tree decomposition of width $w \leq 2\mathtt{tw}(G) + 1$ in time $2^{\mathcal{O}(\mathtt{tw}(G))}n$ [42], and it can be transformed into a *nice tree decomposition* of the same width with $\mathcal{O}(wn)$ bags in time $\mathcal{O}(w^2 n)$ [40]. We now give a high-level overview of the dynamic programming algorithm used to prove the following theorem.

▶ **Theorem 2.** *Metric Dimension admits an algorithm running in time* $2^{\mathtt{diam}^{\mathcal{O}(\mathtt{tw})}} \cdot n^{\mathcal{O}(1)}$.

In [7], as the diameter of the graph was unbounded, it was crucial to restrict the computations for each step of the dynamic programming to vertices "not too far" from the current bag. This was possible due to the metric properties of chordal graphs. In our case, as we consider the diameter of the graph as a parameter, we do not need such restrictions, which makes the proof a little bit simpler.

We now give an intuitive description of the dynamic programming scheme. At each step of the algorithm, we consider a bounded number of *solution types*, depending on the properties of the solution vertices with respect to the current bag. At a given dynamic programming step, we will assume that the current solution resolves all vertex pairs in G_i. Such a vertex pair may be resolved by a vertex from $G - G_i$, or by a vertex in G_i itself.

Any bag X_i of the tree decomposition whose node i lies on a path between two join nodes in T, forms a separator of G: there are no edges between the vertices of $G_i - X_i$ and $G - G_i$. For a vertex v not in X_i, we consider its distance-vector to the vertices of X_i; the distance-vectors induce an equivalence relation on the vertices of $G - X_i$, whose classes we call *X_i-classes*. Consider the two subgraphs G_i and $G - G_i$. Any two solution vertices x, y from $G - G_i$ that are in the same X_i-class, resolve the exact same pairs of vertices from G_i. Thus, for this purpose, it is irrelevant whether x or y will be in a resolving set, and it is sufficient to know that a vertex of their X_i-class will eventually be chosen. In this way, one can check whether a vertex pair from G_i is resolved by a solution vertex of $G - G_i$.

The same idea is used to "remember" the previously computed solution: it is sufficient to remember the X_i-classes of the vertices in the previously computed resolving set, rather than the vertices themselves.

It is slightly more delicate to make sure that vertex pairs in G_i are resolved in the case where such a pair is resolved by a vertex in G_i. Indeed, this must be ensured, in particular when processing a join node i, for vertex pairs belonging to bags in the two sub-trees corresponding to the children i_1, i_2 of i. Such pairs may be resolved by four types of solution vertices: from $G - G_i$, X_i, $G_{i_1} - X_i$, or $G_{i_2} - X_i$. To ensure this, the dynamic programming scheme makes sure that, at each step, for any possible pair C_1, C_2 of X_i-classes, all vertex pairs $\langle u, v\rangle$ consisting of a vertex u of G_i with class C_1 and a vertex v of $G - G_i$ with class C_2 are resolved. The crucial step here is that when a new vertex v is introduced (i.e., added to a bag X_i to form $X_{i'}$), depending on its X_i-class, it must be made sure that it is resolved from all other vertices depending on their X_i-classes, as described above. To ensure that v is distinguished from all other vertices of G_i, we keep track of vertex pairs of $G_i \times (G - G_i)$ that are already resolved by the partial solution, and enforce that, when processing bag $X_{i'}$, for every vertex x of G_i, the pair $\langle x, v\rangle$ is already resolved. As v belongs to the new bag $X_{i'}$, we know its distances to all resolving vertices (indeed, $X_{i'}$-classes of solution vertices can be computed from their X_i-classes), and thus, the information can be updated accurately.

For a bag X_i and a vertex v not in X_i, the number of possible distance vectors to the vertices of X_i is at most $\texttt{diam}(G)^{|X_i|}$. Thus, a solution for bag X_i will consist of: (i) the subset of vertices of X_i selected in the solution; (ii) a subset of the $\texttt{diam}(G)^{|X_i|}$ possible vectors to denote the X_i-classes from which the currently computed solution (for G_i) contains at least one vertex in the resolving set; (iii) a subset of the $\texttt{diam}(G)^{|X_i|}$ possible vectors denoting the X_i-classes from which the future solution needs at least one vertex of $G - G_i$ in the resolving set; (iv) a subset of the $\texttt{diam}(G)^{|X_i|} \times \texttt{diam}(G)^{|X_i|}$ possible pairs of vectors representing the X_i-classes of the pairs of vertices in $G_i \times (G - G_i)$ that are already resolved by the partial solution.

3 Geodetic Set: Lower Bound Regarding Diameter plus Treewidth

The aim of this section is to prove the following theorem.

▶ **Theorem 3.** *Unless the ETH fails,* Geodetic Set *does not admit an algorithm running in time* $2^{f(\texttt{diam})^{o(\texttt{tw})}} \cdot n^{\mathcal{O}(1)}$ *for any computable function* $f : \mathbb{N} \mapsto \mathbb{N}$.

Here, we present a different reduction from 3-Partitioned-3-SAT to Geodetic Set. The reduction takes as input an instance ψ of 3-Partitioned-3-SAT on $3n$ variables and returns (G, k) as an instance of Geodetic Set such that $\texttt{tw}(G) = \mathcal{O}(\log(n))$ and $\texttt{diam}(G) = \mathcal{O}(1)$. We rely on the tool of set representation from Section 2.2, that, for convenience, we reintroduce in the context of Geodetic Set in the next subsection.

3.1 Preliminary Tool: Set Representation

For a positive integer p, define $\mathcal{F}_p$ as the collection of subsets of $[2p]$ that contains exactly p integers. We critically use the fact that no set in $\mathcal{F}_p$ is contained in any other set in $\mathcal{F}_p$ (such a collection of sets is called a *Sperner family*). Let ℓ be a positive integer such that $\ell \leq \binom{2p}{p}$. We define $\mathsf{set\text{-}rep} : [\ell] \mapsto \mathcal{F}_p$ as a one-to-one function by arbitrarily assigning a set in $\mathcal{F}_p$ to an integer in $[\ell]$. By the asymptotic estimation of the central binomial coefficient, $\binom{2p}{p} \sim \frac{4^p}{\sqrt{\pi \cdot p}}$ [36]. To get the upper bound of p, we scale down the asymptotic function and have $\ell \leq \frac{4^p}{2^p} = 2^p$. Thus, $p = \mathcal{O}(\log \ell)$.

We will apply the existence of such a function in the context of Geodetic Set. Suppose we have a "large" collection of vertices, say $A = \{a_1, a_2, \ldots, a_\ell\}$, and a "large" collection of vertices $C = \{c_1, c_2, \ldots, c_m\}$. Moreover, we are given a function $\phi : [m] \mapsto [\ell]$. The basic

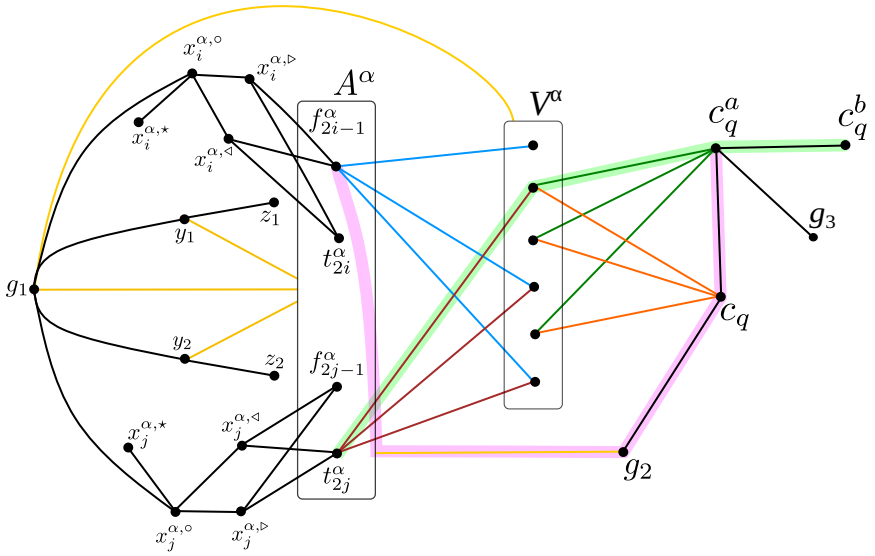


Figure 5 Overview of the reduction. We only draw A^α and V^α here, as A^β, A^γ, V^β, and V^γ are similar. The yellow lines joining g_1, g_2, y_1, and y_2 to sets indicate that the corresponding vertex is adjacent to all the vertices of the corresponding set. Suppose that f_{2i-1}^α and t_{2j}^α are in the geodetic set and $\overline{x}_i$ appears in the clause c_q. The thick green path is a shortest path between t_{2j}^α and c_q^b which does not cover c_q. The thick violet path plus the edge (c_q^a, c_q^b) is a shortest path between f_{2i-1}^α and c_q^b covering c_q.

idea is to design gadgets such that c_q is only covered by the shortest path from $a_{\phi(q)} \in A$ to c_q^b (c_q^b is forced to be chosen in the geodetic set) for any $q \in [m]$, while keeping the treewidth of this part of the graph of order $\mathcal{O}(\log(|A|))$. To do so, we create a "small" intermediate set V (of size $\mathcal{O}(\log(|A|))$) through which will go the shortest paths between vertices in A and C, and we connect a_i to the vertices of V corresponding to the bit-representation of $\mathsf{set\text{-}rep}(i)$, and c_q (with $i = \phi(q)$) to all the other vertices of V. In this way, the construction will ensure that c_q is covered by a shortest path between $a_{\phi(q)}$ and c_q^b, but is not covered by any other shortest path between a vertex of A and a vertex of C. We give the details in the following subsection.

3.2 Reduction

Consider an instance ψ of 3-Partitioned-3-SAT, with $X^\alpha, X^\beta, X^\gamma$ the partition of the variable set. From ψ, we construct the graph G as follows. We describe the construction of X^α, with the constructions for X^β and X^γ being analogous. See Figure 5 for an illustration. We rename the variables in X^α to x_i^α for $i \in [n]$.

- For every variable x_i^α, we add the vertices t_{2i}^α and f_{2i-1}^α. Formally, $A^\alpha = \{t_{2i}^\alpha, f_{2i-1}^\alpha \mid i \in [n]\}$, and hence, $|A^\alpha| = 2n$.
- For every variable x_i^α, we add four vertices: $x_i^{\alpha,\triangleleft}, x_i^{\alpha,\triangleright}, x_i^{\alpha,\circ}, x_i^{\alpha,\star}$. We make $x_i^{\alpha,\triangleleft}$ and $x_i^{\alpha,\triangleright}$ adjacent to both t_{2i}^α and f_{2i-1}^α. We make $x_i^{\alpha,\circ}$ adjacent to both $x_i^{\alpha,\triangleleft}$ and $x_i^{\alpha,\triangleright}$. We make $x_i^{\alpha,\star}$ adjacent to $x_i^{\alpha,\circ}$.
- We add the vertices y_1, y_2, z_1, z_2. We make y_1 and y_2 adjacent to every vertex of A^α. We make y_i adjacent to z_i for $i \in \{1, 2\}$. Note that y_1, y_2, z_1, z_2 are common to X^β and X^γ.
- We add the vertex g_1 and make it adjacent to y_1, y_2, and $x_i^{\alpha,\circ}$ for each $i \in [n]$. Note that g_1 is common to X^β and X^γ. We add edges between g_1 and every vertex of A^α.

- Let p be the smallest positive integer such that $2n \leq \binom{2p}{p}$. In particular, $p = \mathcal{O}(\log n)$. We add a *validation portal*, a clique on $2p$ vertices, denoted by $V^\alpha = \{v_1^\alpha, v_2^\alpha, \ldots, v_{2p}^\alpha\}$. For each $\delta \in \{\alpha, \beta, \gamma\}$, we add edges between g_1 and every vertex of V^δ.
- For every clause C_q in ψ, we introduce three vertices: c_q, c_q^a, c_q^b. We add the edges (c_q, c_q^a) and (c_q^a, c_q^b).
- Define $\mathsf{set\text{-}rep} : [2n] \mapsto \mathcal{F}_p$ as an arbitrary injective function, where $\mathcal{F}_p$ is the Sperner family (and p is as defined two items above). Add the edge $(t_{2i}^\alpha, v_{p'}^\alpha)$ for every $p' \in \mathsf{set\text{-}rep}(2i)$ and the edge $(f_{2i-1}^\alpha, v_{p'}^\alpha)$ for every $p' \in \mathsf{set\text{-}rep}(2i-1)$. If the variable x_i^α appears positively in the clause C_q, then we add the edges $(c_q, v_{p'}^\alpha)$ and $(c_q^a, v_{p'}^\alpha)$ for every $p' \in [2p] \setminus \mathsf{set\text{-}rep}(2i)$. If the variable x_i^α appears negatively in the clause C_q, then we add the edges $(c_q, v_{p'}^\alpha)$ and $(c_q^a, v_{p'}^\alpha)$ for every $p' \in [2p] \setminus \mathsf{set\text{-}rep}(2i-1)$.
- Add a vertex g_2 and make g_2 adjacent to every vertex of A^α and every vertex of $\{c_q : q \in [m]\}$. Note that g_2 is common to X^β and X^γ.
- Add a vertex g_3 and make it adjacent to every vertex of $\{c_q^a : q \in [m]\}$. Note that g_3 and the vertices of $\{c_q, c_q^a, c_q^b : q \in [m]\}$ are common to X^β and X^γ.

This concludes the construction of G. The reduction returns (G, k) as an instance of Geodetic Set where $k = 6n + m + 2$.

3.3 Correctness of the Reduction

Suppose, given an instance ψ of 3-Partitioned-3-SAT, that the reduction above returns (G, k) as an instance of Geodetic Set.

▶ **Lemma 4.** *If ψ is a satisfiable* 3-Partitioned-3-SAT *formula, then G admits a geodetic set of size k.*

Proof. Suppose that $\pi : X^\alpha \cup X^\beta \cup X^\gamma \mapsto \{\texttt{True}, \texttt{False}\}$ is a satisfying assignment for ψ. We construct a geodetic set S of size k for G using this assignment.

For every $\delta \in \{\alpha, \beta, \gamma\}$ and $i \in [n]$, if $\pi(x_i^\delta) = \texttt{True}$, then let $t_{2i}^\delta \in S$, and otherwise, $f_{2i-1}^\delta \in S$. We also put z_1, z_2, $x_i^{\delta,\star}$, and c_q^b into S for all $i \in [n]$, $\delta \in \{\alpha, \beta, \gamma\}$, and $q \in [m]$. Note that $|S| = k$.

Now, we show that S is indeed a geodetic set of G. First, y_1, y_2, z_1, z_2, g_1, and all the vertices of $A^\alpha, A^\beta, A^\gamma$ are covered by a shortest path between z_1 and z_2. Then, for each $\delta \in \{\alpha, \beta, \gamma\}$ and $i \in [n]$, $x_i^{\delta,\triangleleft}$, $x_i^{\delta,\triangleright}$, $x_i^{\delta,\circ}$, and $x_i^{\delta,\star}$ are covered by a shortest path between $S \cap \{t_{2i}^\delta, f_{2i-1}^\delta\}$ and $x_i^{\delta,\star}$. The vertex g_3 is covered by any shortest path between c_q^b and $c_{q'}^b$, where C_q and $C_{q'}$ are two clauses of ψ. Suppose that $\pi(x_i^\delta)$, for some $i \in [n]$ and $\delta \in \{\alpha, \beta, \gamma\}$, satisfies some clause C_q. By our construction, if x_i^δ appears positively (negatively, respectively) in C_q, then t_{2i}^δ (f_{2i-1}^δ, respectively) and c_q^b are at distance four since t_{2i}^δ (f_{2i-1}^δ, respectively) and c_q^a have no common neighbor in V^δ. Moreover, there is a shortest path from t_{2i}^δ (f_{2i-1}^δ, respectively) to c_q^b of length four, covering g_2, c_q, c_q^a, and c_q^b; there is also a shortest path from t_{2i}^δ (f_{2i-1}^δ, respectively) to c_q^b of length four, covering $v_j^\delta, v_h^\delta, c_q^a$, and c_q^b, where $v_j^\delta \in V^\delta$ is a vertex adjacent to t_{2i}^δ (f_{2i-1}^δ, respectively) and v_h^δ is any vertex of V^δ that is not adjacent to t_{2i}^δ (f_{2i-1}^δ, respectively). Thus, every vertex of V^δ for $\delta \in \{\alpha, \beta, \gamma\}$ is covered by a shortest path between two vertices of S. Since every clause of ψ is satisfied by π, it follows that every vertex of $\{c_q, c_q^a, c_q^b : q \in [m]\}$ is covered by a shortest path between two vertices of S. As a result, S is a geodetic set of G. ◀

▶ **Lemma 5.** *If G admits a geodetic set of size k, then ψ is a satisfiable* 3-Partitioned-3-SAT *formula.*

Proof. Suppose that G has a geodetic set S of size at most k. Since they all have degree 1, z_1, z_2, $x_i^{\delta,\star}$, and c_q^b for all $i \in [n]$, $\delta \in \{\alpha, \beta, \gamma\}$, and $q \in [m]$ must be in any geodetic set S of G.

▷ Claim 6. For each $i \in [n]$ and $\delta \in \{\alpha, \beta, \gamma\}$, exactly one of t_{2i}^δ and f_{2i-1}^δ must be in S.

Proof. Since S is a geodetic set, for each $i \in [n]$ and $\delta \in \{\alpha, \beta, \gamma\}$ $x_i^{\delta,\triangleleft}$ and $x_i^{\delta,\triangleright}$ must be covered by shortest paths between two vertices of S. If $t_{2i}^\delta \in S$ ($f_{2i-1}^\delta \in S$, respectively), $x_i^{\delta,\triangleleft}$ and $x_i^{\delta,\triangleright}$ are covered by shortest paths between $t_{2i}^\delta \in S$ ($f_{2i-1}^\delta \in S$, respectively) and $x_i^{\delta,\star}$. Suppose that, for some $i' \in [n]$ and $\delta' \in \{\alpha, \beta, \gamma\}$, neither of $t_{2i'}^{\delta'}$ and $f_{2i'-1}^{\delta'}$ is in S. Moreover, if neither of $x_{i'}^{\delta',\triangleleft}$ and $x_{i'}^{\delta',\triangleright}$ is in S, then, due to the edges incident with g_1, no vertices in S have a shortest path containing any of these two vertices. Similarly, if only one of $x_{i'}^{\delta',\triangleleft}$ and $x_{i'}^{\delta',\triangleright}$ is in S, then the other is not covered by S. Thus, if neither of $t_{2i'}^{\delta'}$ and $f_{2i'-1}^{\delta'}$ is in S, then both $x_{i'}^{\delta',\triangleleft}$ and $x_{i'}^{\delta',\triangleright}$ must be in S. Since $k - |\{z_1, z_2\} \cup \{x_i^{\delta,\star} : i \in [n], \delta \in \{\alpha, \beta, \gamma\}\} \cup \{c_q^b : q \in [m]\}| = 3n$, we conclude that exactly one of t_{2i}^δ and f_{2i-1}^δ must be in S for each $i \in [n]$ and $\delta \in \{\alpha, \beta, \gamma\}$. ◁

By Claim 6 and earlier arguments, we now have that $|S| = k$.

▷ Claim 7. For each $q \in [m]$, the vertex c_q is covered either by a shortest path between c_q^b and t_{2i}^δ, where the variable x_i^δ appears positively in the clause C_q, or by a shortest path between c_q^b and f_{2i-1}^δ, where the variable x_i^δ appears negatively in the clause C_q. Moreover, c_q is covered by no other type of shortest path between two vertices in S.

Proof. By the construction of G, if the variable x_i^δ appears positively in the clause C_q, then there is a shortest path from t_{2i}^δ to c_q^b of length four covering g_2, c_q, c_q^a, and c_q^b. If the variable x_i^δ appears negatively in the clause C_q, then there is a shortest path from f_{2i-1}^δ to c_q^b of length four covering g_2, c_q, c_q^a, and c_q^b.

Next, we show that c_q is not covered by any shortest path between any other two vertices of S. We can check that c_q is not covered by any of the shortest paths between z_1 and z_2, between z_j ($j \in \{1, 2\}$) and $x_i^{\delta,\star}$ ($i \in [n], \delta \in \{\alpha, \beta, \gamma\}$), and between z_j ($j \in \{1, 2\}$) and $S \cap \{t_{2i}^\delta, f_{2i-1}^\delta\}$ ($i \in [n], \delta \in \{\alpha, \beta, \gamma\}$). Note that any shortest path from z_j ($j \in \{1, 2\}$) to c_q^b ($q \in [m]$) is of length five, covering y_j, some vertex of A^δ ($\delta \in \{\alpha, \beta, \gamma\}$), some vertex of V^δ, c_q^a, and c_q^b.

We can check that c_q is not covered by any of the shortest paths between $x_i^{\delta,\star}$ and $x_{i'}^{\delta',\star}$ ($i, i' \in [n], \delta, \delta' \in \{\alpha, \beta, \gamma\}$), and between $x_i^{\delta,\star}$ and $S \cap \{t_{2i'}^{\delta'}, f_{2i'-1}^{\delta'}\}$ ($i, i' \in [n], \delta, \delta' \in \{\alpha, \beta, \gamma\}$). Note that any shortest path from $x_i^{\delta,\star}$ ($i \in [n], \delta \in \{\alpha, \beta, \gamma\}$) to c_q^b ($q \in [m]$) is of length five, covering $x_i^{\delta,\circ}$, g_1, some vertex of V^δ, c_q^a, and c_q^b.

Note that any shortest path between c_q^b and $c_{q'}^b$ ($q, q' \in [m]$) is of length four, covering c_q^a, g_3, and $c_{q'}^a$.

We can check that c_q is not covered by any shortest paths between $S \cap \{t_{2i}^\delta, f_{2i-1}^\delta\}$ and $S \cap \{t_{2i'}^{\delta'}, f_{2i'-1}^{\delta'}\}$ ($i, i' \in [n], \delta, \delta' \in \{\alpha, \beta, \gamma\}$).

If the variable x_i^δ does not appear positively in the clause C_q, then any shortest path between c_q^b and t_{2i}^δ is of length three (because c_q^a and t_{2i}^δ have a common neighbour in V^δ), covering some vertex of V^δ and c_q^a, but not c_q. Similarly, if x_i^δ does not appear negatively in C_q, then any shortest path between c_q^b and f_{2i-1}^δ is of length three and does not cover c_q.

By the case analysis above, the claim is true. ◁

By Claim 6, exactly one vertex of t_{2i}^δ and f_{2i-1}^δ belongs to S for each $i \in [n]$ and $\delta \in \{\alpha, \beta, \gamma\}$. We define an assignment π to the variables of ψ as follows. For each $i \in [n]$ and $\delta \in \{\alpha, \beta, \gamma\}$, if $t_{2i}^\delta \in S$, then $\pi(x_i^\delta) = \texttt{True}$. Otherwise, $\pi(x_i^\delta) = \texttt{False}$. Since S is a geodetic

set for G, every vertex c_q ($q \in [m]$) is covered by a shortest path between two vertices of S. By Claim 7, every vertex c_q ($q \in [m]$) is covered by a shortest path between $S \cap \{t^\delta_{2i}, f^\delta_{2i-1}\}$ and c^b_q, where the variable x^δ_i appears in the clause C_q. It follows that every clause C_q is satisfied by $\pi(x^\delta_i)$. As a result, ψ is a satisfiable 3-Partitioned-3-SAT formula. ◀

Proof of Theorem 3. First, it is not hard to check that the diameter of G is at most 5. Then, let $X = V^\alpha \cup V^\beta \cup V^\gamma \cup \{g_1, g_2, g_3, y_1, y_2\}$. We can check that every component of $G \setminus X$ has at most six vertices and $|X| = \mathcal{O}(\log n)$. Thus, the treewidth $\mathtt{tw}(G)$ – in fact, even the treedepth $\mathtt{td}(G)$ – of G is bounded by $\mathcal{O}(\log n)$. By the description of the reduction, it takes polynomial time to compute the reduced instance. Hence, if there is an algorithm for Geodetic Set that runs in time $2^{f(\mathtt{diam})^{o(\mathtt{tw})}}$ (or $2^{f(\mathtt{diam})^{o(\mathtt{td})}}$), then there is an algorithm running in time $2^{o(n)}$ for 3-Partitioned-3-SAT, which contradicts the ETH. ◀

4 Conclusion

We have shown (under the ETH) that three natural metric-based graph problems, Metric Dimension, Geodetic Set, and Strong Metric Dimension, exhibit tight (double-) exponential running times for the standard structural parameterizations by treewidth and vertex cover number. This includes tight double-exponential running times for treewidth plus diameter (Metric Dimension and Geodetic Set) and for vertex cover (Strong Metric Dimension).

Such tight double-exponential running times for FPT structural paramaterizations of graph problems had previously been observed only for counting problems and problems complete for classes above NP. Thus, surprisingly, our results show that some natural problems can be in NP and still exhibit such a behavior.

It would be interesting to see whether this phenomenon holds for other graph problems in NP, and for other structural parameterizations. Perhaps one can determine certain properties shared by these metric-based graph problems, that imply such running times, with the goal of generalizing our approach to a broader class of problems. In particular, concerning the general versatile technique that we designed to obtain the double-exponential lower bounds, it would be intriguing to see for which other problems in NP our technique works.

In fact, after this paper appeared online, our technique was successfully applied to an NP-complete problem in machine learning [11] (for vc) as well as NP-complete identification problems [10] (for tw).

References

1 A. Achilleos, M. Lampis, and V. Mitsou. Parameterized modal satisfiability. *Algorithmica*, 64(1):38–55, 2012.

2 J. Ahn, L. Jaffke, O. Kwon, and P. T. Lima. Well-partitioned chordal graphs. *Discrete Mathematics*, 345(10):112985, 2022.

3 F. Barbero, L. Isenmann, and J. Thiebaut. On the distance identifying set meta-problem and applications to the complexity of identifying problems on graphs. *Algorithmica*, 82(8):2243–2266, 2020. doi:10.1007/S00453-020-00674-X.

4 R. Belmonte, F. V. Fomin, P. A. Golovach, and M. S. Ramanujan. Metric dimension of bounded tree-length graphs. *SIAM J. Discrete Math.*, 31(2):1217–1243, 2017.

5 B. Bergougnoux, O. Defrain, and F. Mc Inerney. Enumerating minimal solution sets for metric graph problems. In *Proc. of the 50th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2024)*, Lecture Notes in Computer Science. Springer, 2024.

6 E. Bonnet and N. Purohit. Metric dimension parameterized by treewidth. *Algorithmica*, 83:2606–2633, 2021.

7 N. Bousquet, Q. Deschamps, and A. Parreau. Metric dimension parameterized by treewidth in chordal graphs. In *Proc. of the 49th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2023)*, volume 14093 of *Lecture Notes in Computer Science*, pages 130–142. Springer, 2023.

8 D. Chakraborty, S. Das, F. Foucaud, H. Gahlawat, D. Lajou, and B. Roy. Algorithms and complexity for geodetic sets on planar and chordal graphs. In *31st International Symposium on Algorithms and Computation (ISAAC 2020)*, volume 181 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

9 D. Chakraborty, F. Foucaud, H. Gahlawat, S. K. Ghosh, and B. Roy. Hardness and approximation for the geodetic set problem in some graph classes. In *Proc. of the 6th International Conference on Algorithms and Discrete Applied Mathematics (CALDAM 2020)*, volume 12016 of *Lecture Notes in Computer Science*, pages 102–115, Cham, 2020. Springer.

10 D. Chakraborty, F. Foucaud, D. Majumdar, and P. Tale. Tight (double) exponential bounds for identification problems: Locating-dominating set and test cover, 2024. `arXiv:2402.08346`.

11 J. Chalopin, V. Chepoi, F. Mc Inerney, and S. Ratel. Non-clashing teaching maps for balls in graphs. *Arxiv:2309.02876*, 2023.

12 H. Chen. Quantified constraint satisfaction and bounded treewidth. In *Proc. of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004*, pages 161–165. IOS Press, 2004.

13 B. Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

14 M. Cygan, D. Marx, M. Pilipczuk, and M. Pilipczuk. Hitting forbidden subgraphs in graphs of bounded treewidth. *Inf. Comput.*, 256(C):62–82, 2017.

15 M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS 2011)*, pages 150–159, 2011.

16 M. Cygan, M. Pilipczuk, and M. Pilipczuk. Known algorithms for edge clique cover are probably optimal. *SIAM J. Comput.*, 45(1):67–83, 2016. `doi:10.1137/130947076`.

17 B. DasGupta and N. Mobasheri. On optimal approximability results for computing the strong metric dimension. *Discrete Applied Math.*, 221:18–24, 2017.

18 J. Díaz, O. Pottonen, M. J. Serna, and E. J. van Leeuwen. Complexity of metric dimension on planar graphs. *J. Comput. Syst. Sci.*, 83(1):132–158, 2017.

19 M. C. Dourado, F. Protti, D. Rautenbach, and J. L. Szwarcfiter. Some remarks on the geodetic number of a graph. *Discrete Mathematics*, 310(4):832–837, 2010.

20 T. Ekim, A. Erey, P. Heggernes, P. van't Hof, and D. Meister. Computing minimum geodetic sets of proper interval graphs. In *Proc. of the 10th Latin American Symposium on Theoretical Informatics (LATIN 2012)*, volume 7256 of *Lecture Notes in Computer Science*, pages 279–290. Springer, 2012.

21 D. Eppstein. Metric dimension parameterized by max leaf number. *Journal of Graph Algorithms and Applications*, 19(1):313–323, 2015.

22 L. Epstein, A. Levin, and G. J. Woeginger. The (weighted) metric dimension of graphs: Hard and easy cases. *Algorithmica*, 72(4):1130–1171, 2015.

23 J. K. Fichte, M. Hecher, M. Morak, P. Thier, and S. Woltran. Solving projected model counting by utilizing treewidth and its limits. *Artif. Intell.*, 314:103810, 2023.

24 J. K. Fichte, M. Hecher, M. Morak, and S. Woltran. Exploiting treewidth for projected model counting and its limits. In *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Proc.*, volume 10929 of *Lecture Notes in Computer Science*, pages 165–184. Springer, 2018.

25 J. K. Fichte, M. Hecher, and A. Pfandler. Lower bounds for QBFs of bounded treewidth. In *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 410–424. ACM, 2020.

26 J. Focke, F. Frei, S. Li, D. Marx, P. Schepper, R. Sharma, and K. Wegrzycki. Hitting meets packing: How hard can it be? *CoRR*, abs/2402.14927, 2024. doi:10.48550/arXiv.2402.14927.

27 F. V. Fomin, P. A. Golovach, D. Lokshtanov, S. Saurabh, and M. Zehavi. Clique-width III: hamiltonian cycle and the odd case of graph coloring. *ACM Trans. Algorithms*, 15(1):9:1–9:27, 2019.

28 F. Foucaud, G. B. Mertzios, R. Naserasr, A. Parreau, and P. Valicov. Identification, location-domination and metric dimension on interval and permutation graphs. II. Algorithms and complexity. *Algorithmica*, 78(3):914–944, 2017.

29 E. Galby, L. Khazaliya, F. Mc Inerney, R. Sharma, and P. Tale. Metric dimension parameterized by feedback vertex set and other structural parameters. *SIAM J. Discrete Math.*, 37(4):2241–2264, 2023.

30 M. R. Garey and D. S. Johnson. *Computers and Intractability - A guide to NP-completeness*. W.H. Freeman and Company, 1979.

31 T. Gima, T. Hanaka, M. Kiyomi, Y. Kobayashi, and Y. Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theoretical Computer Science*, 918:60–76, 2022. doi:10.1016/J.TCS.2022.03.021.

32 T. Hanaka, H. Köhler, and M. Lampis. Core stability in additively separable hedonic games of low treewidth, 2024. arXiv:2402.10815.

33 F. Harary, E. Loukakis, and C. Tsouros. The geodetic number of a graph. *Mathematical and Computer Modelling*, 17(11):89–95, 1993.

34 F. Harary and R. A. Melter. On the metric dimension of a graph. *Ars Combinatoria*, 2:191–195, 1976.

35 S. Hartung and A. Nichterlein. On the parameterized and approximation hardness of metric dimension. In *Proc. of the 28th Conference on Computational Complexity, CCC 2013*, pages 266–276. IEEE Computer Society, 2013.

36 A. Ian. *Combinatorics of Finite Sets*. Oxford University Press, 1987.

37 K. Jansen, KM. Klein, and A. Lassota. The double exponential runtime is tight for 2-stage stochastic ILPs. *Math. Program.*, 197:1145–1172, 2023.

38 M. M. Kanté and L. Nourine. Polynomial time algorithms for computing a minimum hull set in distance-hereditary and chordal graphs. *SIAM J. Discrete Math.*, 30(1):311–326, 2016.

39 L. Kellerhals and T. Koana. Parameterized complexity of geodetic set. *Journal of Graph Algorithms and Applications*, 26(4):401–419, 2022.

40 T. Kloks. *Treewidth, Computations and Approximations*. Springer, 1994.

41 D. Knop, M. Pilipczuk, and M. Wrochna. Tight complexity lower bounds for integer linear programming with few constraints. *ACM Trans. Comput. Theory*, 12(3):19:1–19:19, 2020.

42 T. Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS 2021)*, pages 184–192, 2022.

43 L. Kowalik, A. Lassota, K. Majewski, M. Pilipczuk, and M. Sokołowski. Detecting points in integer cones of polytopes is double-exponentially hard. In *2024 Symposium on Simplicity in Algorithms (SOSA)*, pages 279–285, 2024.

44 M. Künnemann, F. Mazowiecki, L. Schütze, H. Sinclair-Banks, and K. Węgrzycki. Coverability in VASS Revisited: Improving Rackoff's Bound to Obtain Conditional Optimality. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 131:1–131:20, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

45 D. Kuziak, M. L. Puertas, J. A. Rodríguez-Velázquez, and I. G. Yero. Strong resolving graphs: The realization and the characterization problems. *Discrete Applied Math.*, 236:270–287, 2018.

46 M. Lampis, N. Melissinos, and M. Vasilakis. Parameterized max min feedback vertex set. In *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023*, volume 272 of *LIPIcs*, pages 62:1–62:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

47 M. Lampis, S. Mengel, and V. Mitsou. QBF as an alternative to courcelle's theorem. In *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018*, volume 10929 of *Lecture Notes in Computer Science*, pages 235–252. Springer, 2018.

48 M. Lampis and V. Mitsou. Treewidth with a quantifier alternation revisited. In *12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*, volume 89 of *LIPIcs*, pages 26:1–26:12, 2017.

49 S. Li and M. Pilipczuk. Hardness of metric dimension in graphs of constant treewidth. *Algorithmica*, 84(11):3110–3155, 2022.

50 D. Lokshtanov, D. Marx, and S. Saurabh. Slightly superexponential parameterized problems. *SIAM J. Comput.*, 47(3):675–702, 2018.

51 D. Lokshtanov, S. Saurabh, S. Suri, and J. Xue. An ETH-tight algorithm for multi-team formation. In *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021*, volume 213 of *LIPIcs*, pages 28:1–28:9. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

52 D. Marx and V. Mitsou. Double-exponential and triple-exponential bounds for choosability problems parameterized by treewidth. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *LIPIcs*, pages 28:1–28:15, 2016.

53 M. Mezzini. Polynomial time algorithm for computing a minimum geodetic set in outerplanar graphs. *Theoretical Computer Science*, 745:63–74, 2018.

54 O. R. Oellermann and J. Peters-Fransen. The strong metric dimension of graphs and digraphs. *Discrete Applied Mathematics*, 155(3):356–364, 2007.

55 M. Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: A logical approach. In *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Proc.*, volume 6907 of *Lecture Notes in Computer Science*, pages 520–531. Springer, 2011.

56 M. Pilipczuk and M. Sorge. A double exponential lower bound for the distinct vectors problem. *Discret. Math. Theor. Comput. Sci.*, 22(4), 2020.

57 A. Sebő and E. Tannier. On metric generators of graphs. *Mathematics of Operations Research*, 29(2):383–393, 2004.

58 P. J. Slater. Leaves of trees. In *Proc. of the 6th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, pages 549–559. Congressus Numerantium, No. XIV. Utilitas Mathematica, 1975.

59 P. Tale. Double exponential lower bound for telephone broadcast, 2024. `arXiv:2403.03501`.

Subexponential Parameterized Directed Steiner Network Problems on Planar Graphs: A Complete Classification

Esther Galby
Chalmers University of Technology, Gothenburg, Sweden

Sándor Kisfaludi-Bak
Department of Computer Science, Aalto University, Finland

Dániel Marx
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Roohani Sharma
Department of Informatics, University of Bergen, Norway

Abstract

In the Directed Steiner Network problem, the input is a directed graph G, a set $T \subseteq V(G)$ of k terminals, and a demand graph D on T. The task is to find a subgraph $H \subseteq G$ with the minimum number of edges such that for every $(s,t) \in E(D)$, the solution H contains a directed $s \to t$ path. The goal of this paper is to investigate how the complexity of the problem depends on the demand pattern in *planar graphs*. Formally, if $\mathcal{D}$ is a class of directed graphs, then the $\mathcal{D}$-Steiner Network ($\mathcal{D}$-DSN) problem is the special case where the demand graph D is restricted to be from $\mathcal{D}$. We give a complete characterization of the behavior of every $\mathcal{D}$-DSN problem on planar graphs. We classify every class $\mathcal{D}$ closed under transitive equivalence and identification of vertices into three cases: assuming ETH, either the problem is

1. solvable in time $2^{O(k)} \cdot n^{O(1)}$, i.e., FPT parameterized by the number k of terminals, but not solvable in time $2^{o(k)} \cdot n^{O(1)}$,
2. solvable in time $f(k) \cdot n^{O(\sqrt{k})}$, but cannot be solved in time $f(k) \cdot n^{o(\sqrt{k})}$, or
3. solvable in time $f(k) \cdot n^{O(k)}$, but cannot be solved in time $f(k) \cdot n^{o(k)}$.

Our result is a far-reaching generalization and unification of earlier results on Directed Steiner Tree, Directed Steiner Network, and Strongly Connected Steiner Subgraph on planar graphs. As an important step of our lower bound proof, we discover a rare example of a genuinely planar problem (i.e., described by a planar graph and two sets of vertices) that cannot be solved in time $f(k) \cdot n^{o(k)}$: given two sets of terminals S and T with $|S| + |T| = k$, find a subgraph with minimum number of edges such that every vertex of T is reachable from every vertex of S.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → Graph algorithms analysis

Keywords and phrases Directed Steiner Network, Sub-exponential algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.67

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2208.06015` [16]

1 Introduction

Finding Steiner trees and related network design problems were intensively studied in undirected graphs, directed graphs, and planar graphs, from the viewpoint of approximation and parameterized algorithms [1–3, 5–8, 11–14, 17–19, 22, 23, 25, 27, 29, 31–33]. The simplest problem of this type is Steiner Tree, where given an undirected graph G and set $T \subseteq V(G)$ of terminals, the task is to find a tree with smallest number of edges that contains every

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 67; pp. 67:1–67:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

terminal. This problem models a network-design scenario where the terminals need to be connected to each other with a network of minimum cost. Steiner Forest is the generalization where we do not require connection between every pair of terminals, but have to satisfy a given set of demands. Formally, the input of Steiner Forest is a graph G with pairs of vertices $(s_1, t_1), \ldots, (s_d, t_d)$, the task is to find a subgraph with the minimum number of edges that satisfies every request, that is, s_i and t_i are in the same component of the solution for every $i \in [d]$.

On directed graphs, Directed Steiner Tree (DST) is defined by specifying one of the terminals in T to be the root and the task is to find a subgraph with the smallest number of edges such that there is a path from the root to every terminal in the solution. This problem models a scenario where we need to construct a network where the root can broadcast to every other terminal. An equally natural network design problem on directed graphs is the Strongly Connected Steiner Subgraph (SCSS) problem, where given a directed graph G and a set $T \subseteq V(G)$ of terminals, the task is to find a subgraph with the smallest number of edges where T is in a single strongly connected component, or in other words, the solution contains a path from every terminal to every other terminal. The directed variant of Steiner Forest generalizes both of these problems: in Directed Steiner Network (DSN), the input is a digraph G with pairs of vertices $(s_1, t_1), \ldots, (s_d, t_d)$, and the task is to find a subgraph with the minimum number of edges that has an $s_i \to t_i$ path for every $i \in [d]$.

Planar graphs. A well-known phenomenon on planar graphs is that the running time of parameterized algorithms for typical NP-hard problems have exponential dependence on $O(\sqrt{k})$, where k is the parameter, and this dependence is best possible assuming the Exponential-Time Hypothesis (ETH) [8–10, 15, 20, 21, 24, 26–28, 30]. All three of DST, SCSS, and DSN remain NP-hard on planar graphs. However, they behave very differently from the viewpoint of parameterized complexity: the dependence of the running time on the number k of terminals is very different.

Our starting point

1. Planar DST can be solved in time $2^k \cdot n^{O(1)}$ [4], but cannot be solved in time $2^{o(k)} \cdot n^{O(1)}$ [27], assuming the ETH.
2. Planar SCSS can be solved in time $2^{O(k \log k)} \cdot n^{O(\sqrt{k})}$ [8], but has no algorithm with running time $f(k) \cdot n^{o(\sqrt{k})}$ for any function f, assuming the ETH [8].
3. Planar DSN can be solved in time $f(k) \cdot n^{O(k)}$ [12], but has no algorithm with running time $f(k) \cdot n^{o(k)}$ for any function f, assuming the ETH [8].

Using the terminology of parameterized complexity, Planar DST is *fixed-parameter tractable (FPT)* parameterized by the number k of terminals, but does not admit a *subexponential* FPT algorithm, assuming the ETH. For Planar SCSS and Planar DSN, it is already a highly nontrivial result to show that there is an algorithm that runs in polynomial time for fixed values of k; such an algorithm is called an *XP algorithm*. Furthermore, Planar SCSS admits a *subexponential* XP algorithm (i.e., the exponent of n is $o(k)$), while Planar DSN has no such algorithm, assuming the ETH.

As these results show, there has been significant interest in parameterized directed connectivity problems on planar graphs and in particular tight bounds were obtained for the three problems Planar DST, Planar SCSS, Planar DSN. But what can we say about other natural variants of connectivity requirements? For example, already with the simple extension that the input contains two sets of terminals T_1 and T_2, we can define three different natural connectivity requirements:

(1) The solution has to contain a directed path from any $t \in T_i$ to any $t' \in T_i$.
(2) The solution has to contain a directed path from any $t \in T_1$ to any $t' \in T_1 \cup T_2$.
(3) The solution has to contain a directed path from any $t \in T_1$ to any $t' \in T_2$.

Do these problems behave similarly to one of the three problems listed above? The goal of this paper is to answer such questions by putting the previous results into the context of a wider landscape of directed network design problems. We systematically explore other special cases of Planar DSN and determine their behavior on planar graphs. Our main result is showing that every special case defined in a formal setting behaves similarly to one of the three problems Planar DST, Planar SCSS, Planar DSN: assuming the ETH, the best possible running time is of the form $2^{O(k)} \cdot n^{O(1)}$, $f(k) \cdot n^{O(\sqrt{k})}$, or $f(k) \cdot n^{O(k)}$. Furthermore, we provide an exact combinatorial characterization of the problems belonging to the three classes. In particular, we can use these results to show that variants (1) and (2) behave similarly to Planar SCSS, while variant (3) behaves similarly to Planar DSN. Therefore, (3) is a rare (perhaps first) example of a planar problem with k terminals where the best possible running time is $n^{O(k)}$ and the input can be described in a purely planar way (by the two sets T_1 and T_2) that does not contain any extra information violating the planarity of the instance. This is in stark contrast with the general Planar DSN problem, where the lower bound showing the optimality of the $n^{O(k)}$ running time requires that the input contain an arbitary list of pairs of vertices, giving a highly nonplanar input.

Dichotomy for general graphs. We explore the different special cases of Directed Steiner Network on planar graphs in a framework similar to how Feldmann and Marx [14] treated the problem on general graphs. We can define various special cases of Directed Steiner Network by looking at what kind of graph the connection demands define on the terminals: it is an out-star for Directed Steiner Tree, a bidirected clique for Strongly Connected Steiner Subgraph, and a matching for Directed Steiner Network. More generally, for every class $\mathcal{D}$ of directed graphs, we investigate the problem where the pattern of demands has to belong to the class $\mathcal{D}$. Our goal is to understand how the graph-theoretic properties of the members of $\mathcal{D}$ influence the resulting special case of Directed Steiner Network.

Formally, for every class $\mathcal{D}$, Feldmann and Marx [14] defined the restriction of the problem in the following way.

$\mathcal{D}$-Steiner Network
Input: Digraph G, a set of k terminals $T \subseteq V(G)$, and a demand digraph $D \in \mathcal{D}$ with vertex set T.
Question: What is the minimum number of edges in a subgraph H of G where for each $(u, v) \in E(D)$ there is a $u \to v$ path in H?

One can define also the weighted version of the problem: the input contains weights on the edges and the goal is to minimize the total weight of the subgraph H. Typically, the weighted generalization does not make the problem harder (polynomially bounded integer weights can be easily simulated by subdivided edges, but the algorithmic results in this paper and earlier work allows integer weights in binary as well). Feldmann and Marx [14] characterized those classes $\mathcal{D}$ where $\mathcal{D}$-Steiner Network is fixed-parameter tractable (FPT) parameterized by the number of terminals, that is, can be solved in time $f(k) \cdot n^{O(1)}$. The characterization can be stated in a clean way in terms of five hard families of patterns if we observe the following closure properties of the problem. Observe first that only the transitive closure of D matters for the problem: if D_1 and D_2 have the same transitive closure, then having D_1 or D_2 in the

input results in exactly the same problem. Therefore, it makes sense to consider only classes $\mathcal{D}$ that are *closed under transitive equivalence,* that is, if D_1 and D_2 have the same transitive closure and $D_1 \in \mathcal{D}$, then $D_2 \in \mathcal{D}$ as well. Moreover, we may assume that $\mathcal{D}$ is *closed under identifying vertices:* that is, if $G \in \mathcal{D}$ and G' is obtained by merging two vertices $x, y \in V(G)$ to a single vertex whose in- and out-neighbors are the union of the in- and out-neighbors of x and y, respectively, then G' is also in $\mathcal{D}$. Feldmann and Marx [14, Lemma 5.2] showed that if $\mathcal{D}$-Steiner Network is FPT, then it is FPT also for the closure $\mathcal{D}'$ of $\mathcal{D}$ under identifying vertices, that is, adding further demand patterns obtained by identifying vertices does not make the problem any harder. Intuitively, if D' is obtained from $D \in \mathcal{D}$ by identifying x and y to w, then an instance with demand pattern D' can be simulated by an instance with demand pattern D if we replace w with the two terminals x and y connected by 0-weight edges in both direction (or something similar in case of unweighted graphs[1]).

These arguments show that it is sufficient to obtain a characterization for classes closed under transitive equivalence and identifying vertices. Under these assumptions, Feldmann and Marx [14] identified five graph classes that prevent the problem from being FPT. A *pure out-diamond* is a complete bipartite graph $K_{2,t}$ directed from the 2-element side to the t-element side. A *flawed out-diamond* has in addition a vertex v and edges going from v to the 2-element side. The pure in-diamond and flawed in-diamond are defined similarly by reversing edge orientations. Let us denote by $\mathcal{A}_1$, $\mathcal{A}_2$, ..., $\mathcal{A}_5$ the class of all pure out-diamonds, flawed out-diamonds, pure in-diamonds, flawed in-diamonds, and directed cycles, respectively.

▶ **Theorem 1** (Feldmann and Marx [14]). *Let $\mathcal{D}$ be a class of graphs closed under transitive equivalence and identifying vertices.*

1. ***FPT:*** *If $\mathcal{A}_i \not\subseteq \mathcal{D}$ for any $i \in [5]$, then $\mathcal{D}$-Steiner Network can be solved in time $2^{O(k)}n^{O(1)}$, where k is the number of terminals.*
2. ***Hard:*** *If $\mathcal{A}_i \subseteq \mathcal{D}$ for some $i \in [5]$, then $\mathcal{D}$-Steiner Network is W[1]-hard parameterized by the number k of terminals.*

The first part of Theorem 1 was proved by a combination of an algorithm that solves the problem in time $2^{O(kw \log w)} \cdot n^{O(w)}$ if there is an optimum solution with treewidth w and a combinatorial result showing that if $\mathcal{D}$ is not the superset of $\mathcal{A}_i$ for any $i \in [5]$, then there is a constant bound on the treewidth of optimum solutions. The second part follows from a W[1]-hardness result for each of the five classes $\mathcal{A}_i$.

Our result: trichotomy for planar graphs. Our main result classifies Planar $\mathcal{D}$-Steiner Network (the special case of the problem restricted to planar digraphs G) into three levels of complexity: $2^{O(k)} \cdot n^{O(1)}$, $f(k) \cdot n^{O(\sqrt{k})}$, or $f(k) \cdot n^{O(k)}$ time. In light of Theorem 1 and the earlier results on planar graphs, there are three natural questions that arise:

1. Are there cases that are FPT on planar graphs, but W[1]-hard on general graphs?
2. Are there subexponential FPT cases on planar graphs, that is, where the running time is $2^{o(k)} \cdot n^{O(1)}$?
3. Are there W[1]-hard cases where the optimal running time is neither $f(k) \cdot n^{O(\sqrt{k})}$ nor $f(k) \cdot n^{O(k)}$? If not, where is the boundary line between these two cases?

[1] As mentioned above, polynomially bounded integer weights can be simulated by subdivision of edges. If there are C edges of weight 0, then let us consider every original weight-1 edge to have weight $C+1$ (i.e., a path of length $C+1$) and every weight-0 to have weight 1. Then the original instance has a solution of weight at most x if and only if the new instance has a solution of weight at most $x(C+1)+C$.

We answer the first question negatively: the hard cases remain hard on planar graphs. The answer to the second question is also negative: we show that every (nontrivial) case of Planar $\mathcal{D}$-DSN is at least as hard as Directed Steiner Tree, hence a known lower bound [27] shows that there is no subexponential FPT algorithm, assuming the ETH.

The answer to the third question is positive, but arguably for the wrong reason. Consider the following artificial case. Let $\mathcal{D} = \{D_1, D_2, \dots\}$ be defined the following way: D_i consists of the disjoint union of an out-star of i edges and a directed matching of $\lceil \log_2 i \rceil$ edges. Using the results of Feldmann and Marx [14, Theorems 1.4 and 1.5], it can be shown that Planar $\mathcal{D}$-DSN is solvable in time $f(k)n^{O(\log k)}$, as the graph D_i can be interpreted as a "1-caterpillar with $\lceil \log_2 i \rceil$ extra edges" (see the definition in [14]) and hence there is an optimum solution of treewidth $O(\log k)$. On the other hand, a Planar DSN problem with $\lceil \log_2 i \rceil$ terminal pairs can be reduced to Planar $\mathcal{D}$-DSN with pattern D_i: we can effectively "ignore" the terminals in the out-star by putting these terminals on a directed cycle of 0-weight edges. Thus the lower bound ruling out $f(k)n^{o(k)}$ algorithms for Planar DSN [8] translates into a lower bound ruling out $f(k)n^{o(\log k)}$ algorithms for Planar $\mathcal{D}$-DSN, assuming the ETH. Therefore, the optimal exponent of n in the running time is $O(\log k)$. This is somewhat counterintuitive: as there is a simple reduction from Planar DSN to Planar $\mathcal{D}$-DSN, it feels that the latter problem should be harder. Formally, however, this is not the case: the reduction introduced a new, irrelevant, trivial part of the problem (the terminals on the cycle of 0-weight edges), which increased the parameter significantly.

One could argue that such trivial features of the instance should not influence the way we measure the complexity of the problem. Terminals that are in the same strongly connected component of 0-weight edges can be effectively treated as a single terminal. Therefore, instead of the parameter k being the number of terminals, one could consider the parameter to be the number of strongly connected components of the 0-weight edges that contain terminals, or in other words, the number of terminals after contracting every directed cycle of weight 0. With this parameterization, the reduction from Planar DSN increases the parameter only by 1 and shows that Planar $\mathcal{D}$-DSN has no $f(k)n^{o(k)}$ time algorithm, satisfying the expectation that the problem should be at least as hard as Planar DSN. Equivalently, we can consider the closure $\mathcal{D}'$ of $\mathcal{D}$ under identifying vertices: then $\mathcal{D}'$ contains every directed matching, hence the problem is clearly at least as hard as Planar SCSS. Assuming that the pattern class $\mathcal{D}$ is closed under identifying vertices is a clean way of formalizing the intention that we want to consider terminals in a strongly connected component of weight-0 edges as a single terminal that contributes only 1 to the parameter. In order to obtain meaningful classifications, we assume in the rest of the paper that the class of patterns has this closure property. This way, we avoid pathological examples similar to the one described above.

Under the assumption that $\mathcal{D}$ is closed under transitive equivalence and identifying vertices, we can answer the third question in the negative and map the boundary line between the $f(k)\cdot n^{O(\sqrt{k})}$ and the $f(k)\cdot n^{O(k)}$ cases. We define a finite number $\kappa \leq 300000$ of classes $\mathcal{C}_i$, $i \in [\kappa]$, and show that these are precisely the classes of patterns that prevent subexponential $f(k) \cdot n^{O(\sqrt{k})}$ time algorithms. Note that for every $i \in [\kappa]$, it is easy to show that arbitrary large strongly connected graphs can be obtained from $\mathcal{C}_i$ by identifying vertices. That is, if $\mathcal{D}$ is closed under transitive equivalence and identifying vertices, and $\mathcal{C}_i \subseteq \mathcal{D}$, then $\mathcal{A}_5 \subseteq \mathcal{C}_i \subseteq \mathcal{D}$, i.e., $\mathcal{D}$ contains every directed cycle. Thus we have three different cases depending on whether $\mathcal{D}$ contains (1) none of the $\mathcal{A}_i$'s, (2) some $\mathcal{A}_i$, but no $\mathcal{C}_i$, or (3) some $\mathcal{C}_i$.

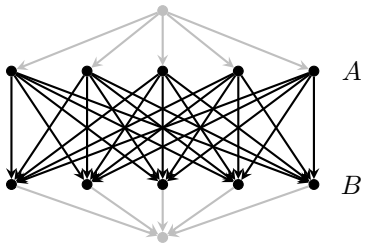


Figure 1 The 5-hard biclique patterns: each gray vertex may or may not be present.

Our main result

▶ **Theorem 2.** *Let $\mathcal{D}$ be a class of directed graphs closed under transitive equivalence and identifying vertices where the number of edges is not bounded.*

1. ***FPT:*** *If $\mathcal{A}_i \not\subseteq \mathcal{D}$ for any $i \in [5]$, then* Planar $\mathcal{D}$-Steiner Network
   - **(i)** *can be solved in time $2^{O(k)} \cdot n^{O(1)}$,*
   - **(ii)** *but has no $2^{o(k)} \cdot n^{O(1)}$ time algorithm assuming the ETH.*
2. ***Subexponential XP:*** *If $\mathcal{A}_i \subseteq \mathcal{D}$ for some $i \in [5]$, but $\mathcal{C}_i \not\subseteq \mathcal{D}$ for any $i \in [\kappa]$, then* Planar $\mathcal{D}$-Steiner Network
   - **(iii)** *can be solved in time $f(k) \cdot n^{O(\sqrt{k})}$,*
   - **(iv)** *but has no $f(k) \cdot n^{o(\sqrt{k})}$ time algorithm assuming the ETH.*
3. ***Hard XP:*** *If $\mathcal{C}_i \subseteq \mathcal{D}$ for some $i \in [\kappa]$, then* Planar $\mathcal{D}$-Steiner Network
   - **(v)** *can be solved in time $f(k) \cdot n^{O(k)}$,*
   - **(vi)** *but has no $f(k) \cdot n^{o(k)}$ time algorithm assuming the ETH.*

We remark that the algorithms work also for weighted graphs, while the lower bounds hold already for unweighted graphs.

Hard classes. Let us define now the graph classes $\mathcal{C}_i$ representing the hard-patterns. Given a digraph G and a set $X \subseteq V(G)$, an *X-source* is a vertex $s \in V(G) \setminus X$ such that $N^+(s) = X$. Similarly, an *X-sink* is a vertex $t \in V(G) \setminus X$ such that $N^-(t) = X$. The first 4 classes $\mathcal{C}_1$, $\ldots$, $\mathcal{C}_4$ are defined by extending a biclique.

▶ **Definition 3** (t-hard-biclique-pattern). *A* t-hard-biclique-pattern *is an (acyclic) digraph D constructed in the following way. We start with two disjoint sets A and B with $|A| = |B| = t$ and introduce every edge from A to B. Furthermore, we introduce into D any combination of the following items (see Figure 1):*

1. *an A-source;*
2. *a B-sink.*

In particular, there are $2 \cdot 2$ types of t-hard-biclique patterns: we let $\mathcal{C}_1, \ldots, \mathcal{C}_4$ be the 4 classes that each contain all the t-hard-biclique-patterns of a specific type for every t.

The following definition specifies the remaining classes.

▶ **Definition 4** (t-hard-matching-pattern). *A* t-hard-matching-pattern *is an (acyclic) digraph D constructed the following way. We start with disjoint vertex sets $W = \{w_1, \ldots, w_t\}$, $X = \{x_1, \ldots, x_t\}$, $Y = \{y_1, \ldots, y_t\}$ and $Z = \{z_1, \ldots, z_t\}$ and introduce the edges $w_i x_i$ and $y_i z_i$ for every $i \in [t]$. Furthermore, we introduce into D any combination of the following items:*

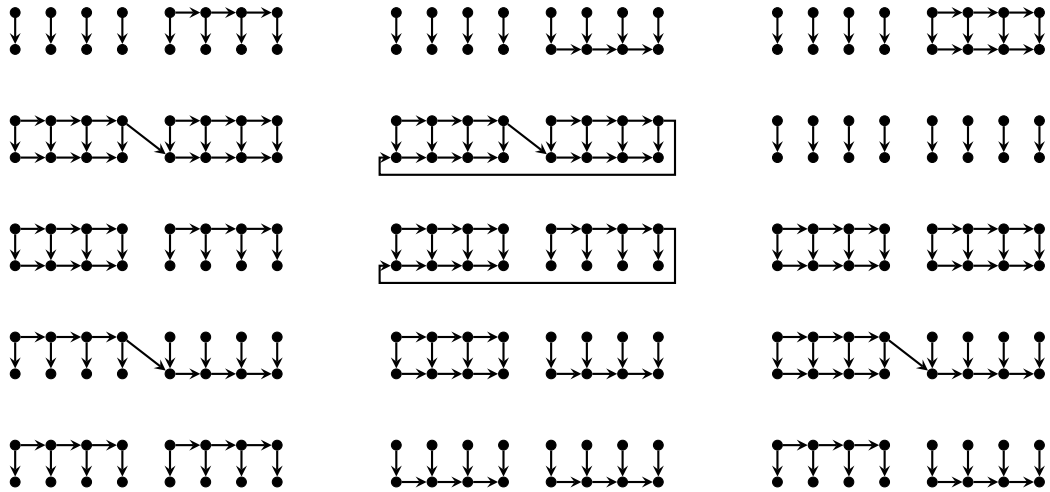


Figure 2 The 4-hard matching patterns (without source, sink, r_{WZ}, or r_{YX}).

1. *either the directed path* $w_1 \to w_2 \to \ldots \to w_t \to z_1 \to z_2 \to \ldots \to z_t$*, or any of the directed paths* $w_1 \to w_2 \to \ldots \to w_t$ *and* $z_1 \to z_2 \to \ldots \to z_t$*;*
2. *either the directed path* $y_1 \to y_2 \to \ldots \to y_t \to x_1 \to x_2 \to \ldots \to x_t$*, or any of the directed paths* $x_1 \to x_2 \to \ldots \to x_t$ *and* $y_1 \to y_2 \to \ldots \to y_t$*;*
3. *a vertex* s *such that for exactly one* $S \in \{W, X, Y, Z, W \cup Y, X \cup Z, X \cup Y, W \cup Z\}$*,* $N^+(s) \cap (W \cup X \cup Y \cup Z) = S$*;*
4. *a vertex* t *such that for exactly one* $S \in \{W, X, Y, Z, W \cup Y, X \cup Z, X \cup Y, W \cup Z\}$*,* $N^-(t) \cap (W \cup X \cup Y \cup Z) = S$*;*
5. *a vertex* r_{WZ} *such that* $N^-(r_{WZ}) \setminus \{s\} = W$ *and* $N^+(r_{WZ}) \setminus \{t\} = Z$*;*
6. *a vertex* r_{YX} *such that* $N^-(r_{YX}) \setminus \{s\} = Y$ *and* $N^+(r_{YX}) \setminus \{t\} = X$*;*
7. *arc* $s \to r_{WZ}$ *if* $N^+(s) \cap W = \emptyset$*, or arc* $s \to r_{YX}$ *if* $N^+(s) \cap Y = \emptyset$*, or both if* $N^+(s) \cap (W \cup Y) = \emptyset$*;*
8. *arc* $r_{WZ} \to t$ *if* $N^-(t) \cap Z = \emptyset$*, or arc* $r_{YX} \to t$ *if* $N^-(t) \cap X = \emptyset$*, or both if* $N^-(t) \cap (Z \cup X) = \emptyset$
9. *arc* $s \to t$ *if* s *cannot already reach* t*.*

In particular, there are less than $5 \cdot 5 \cdot 9 \cdot 9 \cdot 2 \cdot 2 \cdot 4 \cdot 4 \cdot 2$ *types of* t*-hard matching patterns: we let* $\mathcal{C}_5, \ldots, \mathcal{C}_\kappa$ *, where* $\kappa \leq 259200$*, be the classes that each contain all the* t*-hard-matching-patterns of a specific type for every* t*.*

Note that some of these classes are isomorphic. For example, adding the path $x_1 \to x_t$ or the path $z_1 \to z_t$ lead to isomorphic graphs. If we just consider the graph classes where we choose not to add a source, sink, vertex r_{WZ}, or vertex r_{YX}, then we have 15 non-isomorphic classes, as shown in Figure 2. One could think of t-hard-matching-patterns as (the transitive closure of) one of these graphs, potentially extended by appropriate sources, sinks, and r_{WZ}/r_{YX} vertices.

Finally, we define a t-hard pattern as any of the patterns defined above.

▶ **Definition 5** (t-hard-pattern). *A* t-hard-pattern *is either a* t*-hard-biclique-pattern or* t*-hard-matching-pattern, that is, a pattern that belongs to one of the* κ *classes* $\mathcal{C}_1, \ldots, \mathcal{C}_\kappa$ *defined in Definitions 3 and 4.*

1.1 Overview of our main result

Observe that Theorem 2 consists of six statements. Let us briefly discuss how these six statements are proved. Note that some of these statements follow from known results, while for others we need to do a substantial amount of new technical work. The proofs of statements (iii) and (vi) form the main technical part of the paper (see Figure 4).

The main problem studied in this paper, obtaining tight upper and lower bounds for different families of network design problems, is a genuine computer science question. Due to the nature of the question we are asking, our results are at the intersection of computational complexity, algorithms, and combinatorics (graph theory). The lower bounds are obtained using the standard method of computational complexity: by reductions from problems for which (conditional) lower bounds were already established. In particular, we are using the known lower bounds for Planar DSN and Planar SCSS on planar graphs [8]. However, the majority of the hardness proofs we present contain significant new ideas, new gadget constructions, and use new non-obvious global structures when connecting the gadgets.

The upper bounds are obtained using an algorithm of Feldmann and Marx [14] (see Theorem 7 below) solving the problem in time depending on the treewidth of an optimum solution. Therefore, in this paper the main technical effort is spent on the combinatorial question of bounding the treewidth of optimum solutions. Our proof uses planarity in a geometric way (arguing about faces, invoking Sperner's Lemma, etc.) and hence completely different from earlier proofs that relied only on bidimensionality of planar graphs [8]. The treewidth-based algorithm offers an abstraction that allows us to treat the upper bounds in a clean, modular manner. Thus we can focus our efforts on understanding the patterns that allow faster solutions, without having to develop details of algorithmic steps.

Furthermore, we have the purely combinatorial task of connecting the obstructions that prevent the treewidth upper bound and the hard structures. In the full version [16], we establish this connection with a heavy use of case analysis and Ramsey-theoretical arguments. The type and amount of combinatorial effort is very different from what was needed in earlier work on general graphs [14], where more elementary arguments were sufficient.

Statement (i): FPT algorithms

The FPT result (i) follows directly from Theorem 1 (here the surprising aspect is that, by statement (iv), there are no further FPT cases).

Statement (ii): no subexponential FPT algorithms

The lower bound (ii) follows by observing that every relevant class contains either all in-stars or all out-stars, hence the lower bound for Directed Steiner Tree [27] applies. To avoid triviality, we need to assume that the class contains graphs with arbitrarily large number of edges.

▶ **Lemma 6.** *Let $\mathcal{D}$ be a class of graphs closed under identifying vertices and transitive closure where the number of edges of the graphs is not bounded. Then one of the following holds:*

- *$\mathcal{D}$ contains every directed cycle,*
- *$\mathcal{D}$ contains every out-star, or*
- *$\mathcal{D}$ contains every in-star.*

In statement (ii) of Theorem 2, we assume that $\mathcal{A}_i \not\subseteq \mathcal{D}$, and $\mathcal{A}_5$ is the class of all directed cycles. Thus $\mathcal{D}$ contains either every out-star or every in-star.

Statement (iii): $f(k)n^{O(\sqrt{k})}$ algorithms

Our main technical result is proving statement (iii): the existence of an $f(k) \cdot n^{O(\sqrt{k})}$ time algorithm if $\mathcal{C}_i \not\subseteq \mathcal{D}$ for any $i \in [\kappa]$ (in the following subsection, we give a more detailed description of the proof). This algorithm is obtained by showing that the treewidth of the optimal solution is always $O(\sqrt{k})$ under these conditions. Then we can use the following result of Feldmann and Marx [14].

▶ **Theorem 7** (Theorem 1.5 of [14]). *If an instance (G, T, D) of* Directed Steiner Network *has an optimum solution H of treewidth w, then it can be solved in $2^{O(kw \log w)} \cdot n^{O(w)}$ time.*

Note that this is a slightly weaker form of the statement, with a simplified bound on the running time. With Theorem 7 at hand, our main goal is to prove that every optimum solution of Planar $\mathcal{D}$-DSN has treewidth $O(\sqrt{k})$ if $\mathcal{C}_i \not\subseteq \mathcal{D}$ for any $i \in [\kappa]$.

Towards proving this bound, we first translate the question to a problem on acyclic graphs: it is sufficient to show that if the solution is acyclic, then the total degree of the branch vertices (i.e., of degree > 2) is $O(k)$. More formally, for a vertex v of a digraph, let $d^*(v)$ denote the *branch degree* of v, defined as

$$d^*(v) = \max(d^+(v) + d^-(v) - 2, 0),$$

where $d^+(v)$ and $d^-(v)$ denotes the out- and in-degree of v, respectively. The total branch degree of a graph G is the sum of all branch degrees of the vertices of G.

We say that a feasible solution H of (G, T, D) is *edge-minimal* if for all edges $e \in E(H)$ the graph $H - e$ is not feasible. An edge e is *essential* for some demand edge $(t, t') \in E(D)$ if there is no $t \to t'$ path in $H - e$. Note that all edges of an edge-minimal graph H are essential for some demand edge of D. We say that a pattern class $\mathcal{D}$ is *c-acyclic-bounded* for some $c = O(1)$ if for any instance (G, T, D) of Planar $\mathcal{D}$-Steiner Network where G, D are acyclic, and any edge-minimal solution H, the total branch degree of H is at most $c|T|$.

The next theorem moves the problem to the domain of acyclic digraphs: what we need now is a linear bound on the total branch degree of acyclic solutions.

▶ **Theorem 8.** *If the pattern class $\mathcal{D}$ is c-acyclic-bounded for some $c = O(1)$, then for any instance of* Planar $\mathcal{D}$-Steiner Network *with $|T| = k$, the solution graph H has treewidth $O(\sqrt{k})$.*

Applying Theorem 7 implies that c-acyclic-bounded classes have the desired subexponential algorithm, but we still need to establish a link between non-c-acyclic-bounded classes and t-hard-patterns. First, we argue that if the total branch degree is too large, then a grid-like structure can be found in the solution. The grid-like structure appears in the solution to satisfy a set of edges in the demand graph D, and this set of demands forms a certain hard structure in the demand pattern that we call a t-tough-pair which we define informally here (see Definition 14 for a formal definition). We say that two edges e_1 and e_2 are *weakly independent* if there is no directed path from the head of one to the tail of the other. Edges e_1 and e_2 are *strongly independent* if, in addition to being weakly independent, there is no directed path containing the heads of both edges and there is no directed path containing the tails of both edges. An edge e is *minimal* in a digraph D if there is no path from the tail of e to the head of e avoiding e. Let $E_1 \cup E_2$ be a vertex-disjoint set of minimal edges with $|E_1| = |E_2| = t$. We say that (E_1, E_2) is a *t-tough-pair* if

- any two edges $e, e' \in E_1$ are weakly independent,
- any two edges $e, e' \in E_2$ are weakly independent, and
- any two edges $e_1 \in E_1$ and $e_2 \in E_2$ are strongly independent.

Observe that in particular the two matchings in a t-hard-matching-pattern form (vertical edges in Figure 2) a t-tough-pair. Similarly, taking two vertex-disjoint matchings of size t each in a $2t$-hard-biclique-pattern is also a t-tough-pair.

Our main structure theorem connects the total branch degree to the existence of these kind of hard structures.

▶ **Theorem 9** (Structure Theorem). *Let $\mathcal{D}$ be a class of graphs closed under identifying vertices and transitive equivalence. Then either $\mathcal{D}$ has a pattern with a t-tough-pair for each positive integer t, or it is c-acyclic-bounded for some constant c.*

Theorems 8 and Theorem 9 show that the existence of arbitrarily large t-tough-pairs is the canonical reason why the treewidth is not $O(\sqrt{k})$. The lower bounds ruling out $f(k) \cdot n^{o(k)}$ time algorithms essentially rely on the existence of t-tough-pairs. However, the existence of a t-tough-pair in a demand pattern $D \in \mathcal{D}$ is not sufficient for the lower bound: the t-tough-pair could be only a small part of the pattern D, and hence the lower bounds may not apply. We show, with heavy use of Ramsey's Theorem and other combinatorial arguments, that whenever a large t-tough-pair appears in a graph, then the graph can be "cleaned": we can identify vertices to obtain one of the t-hard-patterns. Therefore, if arbitrary large t-tough-pairs appear in the members of a class $\mathcal{D}$ closed under identifying vertices, then the class is a superset of one of the hard classes $\mathcal{C}_i$.

▶ **Theorem 10.** *Let $\mathcal{D}$ be a class of graphs closed under transitive equivalence and identifying vertices. The following two are equivalent:*

1. *For every t, there is a $D \in \mathcal{D}$ that has a t-tough pair.*
2. *$\mathcal{C}_i \subseteq \mathcal{D}$ for some $i \in [\kappa]$.*

We can conclude that if $\mathcal{D}$ is not the superset of $\mathcal{C}_i$ for any $i \in [\kappa]$, then the treewidth of the optimum solution is $O(\sqrt{k})$, implying that Planar $\mathcal{D}$-DSN can be solved in time $f(k) \cdot n^{O(\sqrt{k})}$.

Statement (iv): no $f(k)n^{o(\sqrt{k})}$ algorithms

If $\mathcal{D}$ contains $\mathcal{A}_5$ (directed cycles), then lower bounds ruling out $f(k) \cdot n^{o(\sqrt{k})}$ time algorithms follow from the known lower bound for Strongly Connected Steiner Subgraph [8]. When $\mathcal{D}$ contains one of $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4$ (pure or flawed dimonds), the problem is known to be W[1]-hard on general graphs [14]. We reprove the hardness of diamonds, this time restricted to planar graphs, and observe that this W[1]-hardness proof actually rules out $f(k) \cdot n^{o(\sqrt{k})}$ time algorithms. Compared to the W[1]-hardness on general graphs, the proof for planar graphs is more involved. As it is very usual for planar problems, we establish these lower bounds by reducing from $k \times k$-Grid Tiling, which cannot be solved in time $f(k) \cdot n^{o(k)}$, assuming the ETH [9]. For statement (iv), we need to reduce from $\sqrt{k} \times \sqrt{k}$ Grid Tiling to a Planar $\mathcal{D}$-DSN with $O(k)$ terminals forming a pure/flawed in/out-diamond pattern, ruling out $f(k) \cdot n^{o(\sqrt{k})}$ algorithms for such patterns.

In all these reductions, we are reusing and extending the gadget constructions from earlier work [8]. However, the high-level structure of the reduction is substantially different and depends on the pattern class we are considering. In light of Theorem 7, we should first verify, as a sanity check, that the treewidth of the solution can be sufficiently large, that is, it can be $\Omega(\sqrt{k})$ in case of diamonds. Typically, one can expect that examples with sufficiently large treewidth shed some light on how the high-level structure of the hardness proof could look like. Figure 3 shows that treewidth can be indeed sufficiently large: a $\sqrt{k} \times \sqrt{k}$ grid can be obtained from two "interlocking combs."

Statement (v): $f(k)n^{O(k)}$ algorithms

The upper bound $f(k) \cdot n^{O(k)}$ (statement (v)) follows from the work of Eiben et al. [12], who showed that Planar DSN with k terminals can always be solved within this running time. Note that Feldman and Ruhl [13] presented a $n^{O(p)}$ time algorithm for DSN on general graphs where p is the number of demands. However, as the number of demands on k terminals can be $\Omega(k^2)$, their algorithm *does not* give an $f(k) \cdot n^{O(k)}$ algorithm where k is the number of terminals.

Statement (vi): no $f(k)n^{o(k)}$ algorithms

To prove statement (vi) ruling out $f(k) \cdot n^{o(k)}$ algorithms, we provide such a lower bound for each class $\mathcal{C}_i$ for $i \in [\kappa]$. Analogously to statement (iv), the proof is by reduction from $k \times k$ Grid Tiling to a Planar $\mathcal{D}$-DSN instance with a k-hard-matching-pattern or a k-hard-biclique-pattern, ruling out $f(k) \cdot n^{o(k)}$ algorithms. Again, let us verify that the treewidth can be sufficiently large: Figure 3 shows how a $k \times k$ grid can appear in the solution to an instance with k terminals.

For t-hard-matching-patterns, the simplest case is when we have two induced matchings of size t. Then a $t \times t$ grid can arise very easily in the solution if the terminals are on the boundary of a grid. The crucial point here is that the t-hard-matching-pattern was defined in a way that all the additional paths, sources etc. do not interfere with the grid, see the figure for an example. For the t-hard-biclique-pattern, there is a non-obvious and highly delicate way of constructing an instance with $2t$ terminals where a $t \times t$ grid appears. Combining these constructions gives the lower bound.

▶ **Theorem 11.** *Let $\mathcal{D}$ be a class of graphs closed under identifying vertices and transitive equivalence. If $\mathcal{C}_i \subseteq \mathcal{D}$ for some $i \in [\kappa]$, then* Planar $\mathcal{D}$-Steiner Network *has no $f(k) \cdot n^{o(k)}$ time algorithm assuming the ETH.*

Let us observe that if $\mathcal{D}$ consists of bicliques directed from one side to the other, then Planar $\mathcal{D}$-DSN corresponds to the following problem: given a planar digraph G with two sets $S, T \subseteq V(G)$ of terminals with $|S| + |T| = k$, find a subgraph with minimum number of edges such that there is a path from every vertex of S to every vertex of T. Our result shows that, assuming the ETH, this problem has no $f(k) \cdot n^{o(k)}$ time algorithm. This result is surprising, as the problem can be considered to be *genuinely planar* in the sense that the input is a planar graph with k terminals and a single bit of annotation at each terminal (and there is no extra information, such as terminal pairs, that can disregard the planarity of the instance). To our knowledge, this is the first example of a relatively natural planar problem where $f(k) \cdot n^{O(k)}$ is best possible and cannot be improved to $f(k) \cdot n^{O(\sqrt{k})}$.

1.2 Details of Statement (iii): the $f(k) \cdot n^{O(\sqrt{k})}$ algorithm

In this section, we give a more detailed overview of the technical steps of the proof of (iii) sketched above.

From treewidth to total branch degree. Theorem 8 translates the question about the treewidth of the solution in general graphs to a question about the total branch degree of the solution in acyclic graphs. Suppose that we have an edge-minimal solution H in a (not necessarily acyclic) graph G with k terminals. Let us contract the strongly connected

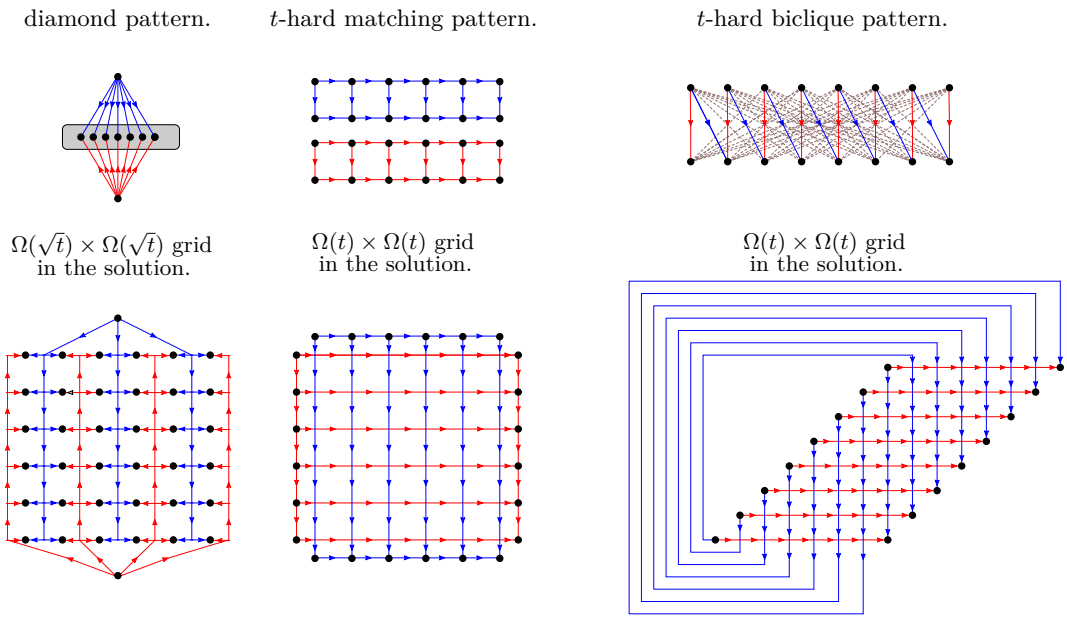


Figure 3 Pattern graphs (top row) and example minimal solution graphs with large grid patterns and large treewidth (bottom row). The red/blue edges show how (some of the) demands are connected in the solution.

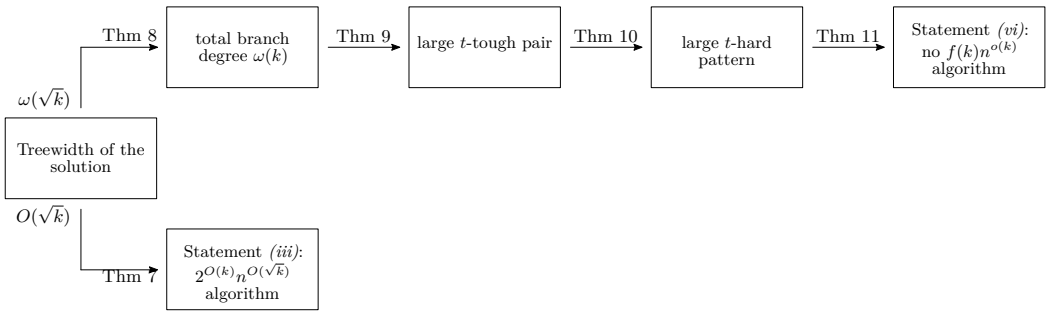


Figure 4 The structure of the proofs of statements (iii) and (vi).

components of H in both G and H to obtain G' and H', respectively. We can observe that H' is an acyclic graph that is the optimum solution to an instance in G' with at most k terminals. Our goal is to show that if H' has total branch degree d, then H has treewidth $O(\sqrt{d+k})$. Therefore, in the later steps of the proof, we bound the total branch degree of H' by $O(k)$, giving an $O(\sqrt{k})$ bound on the treewidth of H.

We say that a vertex of a strongly connected component of H is a *portal* if it is incident to an edge connecting it to some other component. For simplicity of discussion, let us assume here that every strongly connected component of H has at least 3 edges incident to the portals, that is, every vertex of H' has at least 3 incident edges. (If a component has less than 3 such edges and has no terminal, then it consists only of a single vertex and does not affect treewidth anyway; if it has terminals, then it can be taken into account with additional calculations.) By this assumption, the set P of portals has size at most $6d$, where d is the total branch degree of H'.

We want to bound the treewidth of H by showing that there is a set W of $O(d+k)$ vertices such that $H - W$ has treewidth at most 2. It is known that if removing a set W of vertices from a *planar* graph reduces treewidth to a constant, then the planar graph has treewidth $O(\sqrt{|W|})$. Thus the treewidth bound $O(\sqrt{d+k})$ follows from the existence of such a set W.

Let $H[V_i]$ be a strongly connected component of H that has p_i portals and contains k_i terminals. The key observation is that the only role of $H[V_i]$ in the solution is to fully connect the terminals and portals in $H[V_i]$. That is, we can assume that $H[V_i]$ is an optimum solution of a Strongly Connected Steiner Subgraph instance with $p_i + k_i$ terminals. Chitnis et al. [8] showed that we can remove a set W_i of $O(p_i + k_i)$ vertices from such an optimum solution to reduce its treewidth to 2. Therefore, taking the union of P and every W_i, we get a set W of size $O(d) + O(\sum(p_i + k_i)) = O(d + k)$ whose removal reduces treewidth to 2 (as removing P breaks the graph in a way that each component is a subset of some V_i, and the removal of W_i breaks $H[V_i]$ into components of treewidth at most 2).

Building a skeleton. Towards the proof of Theorem 9, our goal is to bound the total branch degree by $O(k)$ in an edge-minimal acyclic solution H. At some step of the proof, it will be important to assume that H is a triangulated planar graph (every face has exactly three vertices and edges), which is of course not true in general. Therefore, we introduce artificial undirected edges in the graph H to make it triangulated. As these edges do not play any role in the directed problem, it does not change the nature of the solution. Another simplification step is that we assume that there is no vertex $v \notin T$ with $d^-(v) = d^+(v) = 1$. Such a vertex has branch degree 0 and hence suppressing it (i.e., removing it and adding an edge from its in-neighbor to its out-neighbor) has no effect on the total branch degree and on the connectivity of the terminals.

We start by building a *skeleton* of the solution: a connected subgraph that contains every terminal. The skeleton is composed from *segments* of two types. A *long segment* is a directed path of H of length at least some constant L. A *short segment* is any path in the undirected sense of length at most L, possibly containing both undirected or directed edges of any orientation. Furthermore, we require that any two long segments in the skeleton are *distant,* that is, have distance at least L in the undirected sense.

A skeleton tree consisting of $O(k)$ segments and containing all the terminals can be built the following way. Initially, we start with an edgeless subgraph R containing only the k terminals. For simplicity of discussion, let us assume that the demand pattern is connected (in the undirected sense). Then there has to be a demand t_it_j such that t_i and t_j are in two different components C_i and C_j of R, respectively. This means that H has a directed path P connecting two different components of R. If P has length at most L, then we can introduce it as short segment to reduce the number of components of R. Otherwise, we can shorten P to P' such that every vertex of P' is at distance at least L from R and the two endpoints are at distance exactly L from two different components C and C' of R. Then we can reduce the number of components of R by introducing P' as a long segment and two short segments connecting the endpoints of P' to C and C'. By repeating these steps, we can reduce the number of components to 1 by introducing $O(k)$ segments in total.

Refining the faces. Our next goal is to further refine the skeleton such that every face of the skeleton has at most 35 segments on its boundary, and it is still true that the skeleton consists of $O(k)$ segments. We achieve this goal by iteratively dividing a face into two by introducing to the skeleton a new path consisting of at most 5 segments. We argue below that if the division is not very skewed in a certain sense, then the bound $O(k)$ on the number of segments can be achieved even after iterative applications of this step.

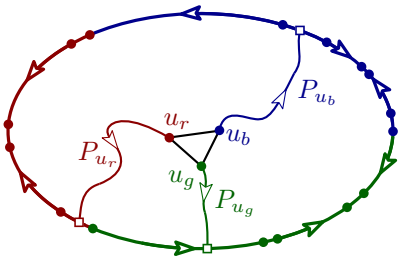


Figure 5 Finding a division that is not skewed.

Suppose that we have a face F where $x \geq 36$ segments appear on the boundary. Let P be a path between two segments of the boundary and assume that P consists of at most 5 segments. Introducing the path P into the skeleton creates two new faces F_1 and F_2 that see some number x_1 and x_2 segments on the boundary of F, plus the 5 new segments of P. We have $x_1 + x_2 \leq x + 2$: if the endpoints of P are internal vertices of segments, then we may have up to 2 segments that are now on the boundary of both F_1 and F_2.

For a face seeing $x \geq 13$ segments of the skeleton, let us define $x - 13 \geq 0$ to be the potential of the face. If we chose the path P such that $x_1, x_2 \geq 13$, then the potential of the two new faces F_1 and F_2 are defined. Moreover, the total potential of the two faces is at most

$$(x_1 + 5 - 13) + (x_2 + 5 - 13) \leq x - 14,$$

strictly less than the potential of F. This means that if we start with a face F that sees x segments of the skeleton, then repeated applications of this step can introduce only $O(x)$ new segments.

Finding a division that is not skewed. Next we show that if face F sees $x \geq 36$ segments of the skeleton, then we can find a division with $x_1, x_2 \geq 13$. Then as we have seen above, repeated applications of this step introduces $O(x)$ segments and divide F into faces that see at most 35 segments each.

Let us divide the boundary of F into three parts, red, green, and blue, each containing at least 12 segments (see Figure 5). As every vertex v inside the face F is essential for the solution, there is a directed path P_v from v to some vertex of the boundary; let us fix such a P_v for each v. This defines a color of v according to which of the three parts of the boundary contains the head of P_v. Then by Sperner's Lemma and fact that the graph is triangulated, there is a triangle u_r, u_g, u_b inside F where the three vertices have three different colors. From the assumptions that u_r, u_g, u_b are on three different parts, and each part has length at least 13, it follows that there are two vertices, say u_r and u_b, such that both subpaths of the boundary between the heads of P_{u_r} and P_{u_b} have at least 12 segments. Then putting together P_{u_r} and P_{u_b} creates a path P that divides the face F in the required way. This argument needs to be refined a bit further: as we said earlier, we want a skeleton where the long segments are distant, i.e., are at distance at least L from each other. But this can be easily achieved by appropriately shortening the long segments P_{u_r}, P_{u_b}, and then extending them by three short segments.

Many edges incident to a long path. We assume now that the skeleton has $O(k)$ faces, each seeing at most 35 segments. If we can show that the total branch degree (of the orginal solution H without the artificial edges) is a constant in each face, then we can bound by $O(k)$ the total branch degree of the solution. We can observe, using the acyclicity of the edges inside the face, that we need to bound only the number of edges incident on the boundary.

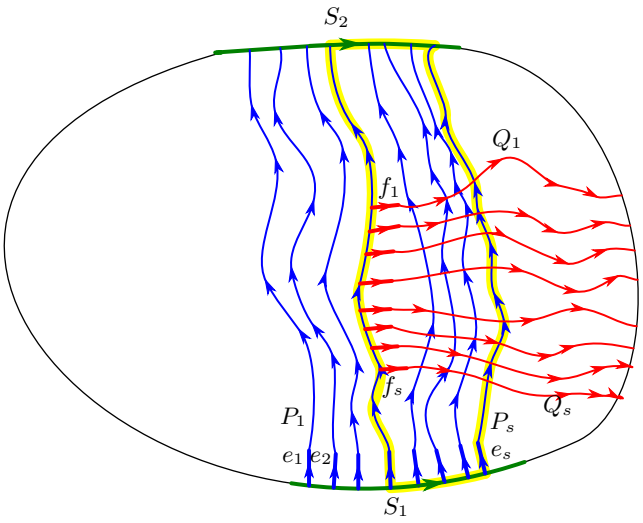


Figure 6 Finding a grid.

Let e be an edge inside the face incident to vertex v of the boundary. We say that e is *essential* for demand t_it_j if removing e breaks every path from t_i to t_j. Then we can define a path P_e the following way: let us take any path P from t_i to t_j, and let P_e be the subpath of P starting from e (which has to appear on P) to the first vertex on the boundary of F. Let us consider two edges e_1, e_2 starting from the same vertex v of the boundary. Let us observe that P_{e_1} and P_{e_2} cannot intersect: then we could bypass e.g. e_1 by starting on P_{e_2} and following it until intersection. By a similar argument, P_{e_1} and P_{e_2} cannot go to the same long segment: then one of P_{e_1} and P_{e_2} could be avoided by using the other path and part of the long segment. From these observations, it follows that the only way the boundary can have many edges incident to it is that if there are edges $e_1, \ldots, e_s$ incident to distinct vertices of a long segment S_1, with paths $P_1, \ldots, P_s$ going to distinct vertices of some other long segment S_2 (see Figure 6).

Finding a grid and a t-tough pair. Now comes the point where we use the assumption that long segments are distant. In particular, this means that the "middle path" $P_{e_{s/2}}$ is long. The internal vertices of this path have no terminals (as all the terminals are on the skeleton), hence it is not possible that $d^+(v) = d^-(v) = 1$ for any such internal vertex. Thus either there are many vertices on this path that have an edge leaving the path, or many vertices that have an edge entering the path. Assume without loss of generality the former, let $f_1, \ldots, f_s$ be these edges. Again, each edge is essential for some demand, hence the path satisfiying the demand has a subpath Q_i starting with f_i and going to the boundary. We can observe again that these paths have to be disjoint. Therefore, we can obtain a grid-like structure in the region surrounded by S_1, $P_{s/2}$, S_2, and P_s, see the region highlighted by yellow in Figure 6. (There are some other cases to consider, which we ignore here. For example, the paths Q_i may go to S_1 or S_2.) This region has $s/2 - 1$ "vertical" paths $P_{s/2}$, $\ldots$, P_{s-1}, intersected by the s "horizontal paths" $Q_1, \ldots, Q_s$.

We observe that if this grid has t horizontal and vertical paths, then we can use it to discover a t-tough pair. Each edge e_i is essential for some minimal demand; let E_1 be the set of these t demands. Similarly, we define E_2 based on choosing a minimal demand for which f_i is essential. Then we can carefully verify that (E_1, E_2) is a t-tough-pair: if there is an edge in the demand graph that is not allowed, then a careful analysis shows that there is a way of

bypassing some e_i or f_i in the grid, contradicting the fact that it is essential. This concludes the proof that if we have an upper bound on the size of the largest t-tough pair appearing in the graphs of class $\mathcal{D}$, then we can bound the treewidth of the solution by $O(\sqrt{k})$.

Cleaning. To prove Theorem 10, we need to show that if arbitrary large t-tough-pairs appear in the graphs of $\mathcal{D}$, then $\mathcal{C}_i \subseteq \mathcal{D}$ for some $i \in [\kappa]$. The proof is a long combinatorial argument to show that we can find t-tough-pairs that are canonical in some sense, and then we use the assumption that $\mathcal{D}$ is closed under identifying vertices to contract the vertices outside the t-tough-pair into a small constant number of well-behaved vertices.

Suppose that there is a t-tough-pair (E_1, E_2) in a digraph D. The minimality of the edges in E_1 and the fact that they do not appear in directed cycles (as they are weakly independent to themselves) imply that for any two edges $x_iy_i, x_jy_j \in E_1$, at least one of the following holds:

1. exactly the edges x_iy_j, x_jy_i appear between $\{x_i, y_i\}$ to $\{x_j, y_j\}$,
2. there is no edge from $\{x_i, y_i\}$ to $\{x_j, y_j\}$, or
3. there is no edge from $\{x_j, y_j\}$ to $\{x_i, y_i\}$.

Let us consider a complete graph on t vertices $w_1, \ldots, w_t$, and for every $i < j$, color the edge w_iw_j according to which of the three statements hold for the edges x_iy_i and x_jy_j (if more than one statement is true, we can choose arbitrarily). By Ramsey's Theorem, there is a large subset $E_1' \subseteq E_1$ where the same statement holds for any pair of edges. We can find a similar subset $E_2' \subseteq E_2$. We consider two main cases. The first case is when Statement 1 holds either in E_1' or E_2'. Then what we have is a matching x_iy_i of minimal edges that is part of a complete bipartite graph, that is, every x_i is adjacent to every y_j (but note that x_iy_j does not have to be a minimal edge). The second case is where we have Statement 2 or 3 in both E_1' and E_2'. Then we can reorder E_1 and E_2 to have a further ordering property: there is no edge from $\{x_i, y_i\}$ to $\{x_j, y_j\}$ for $j < i$. We handle the two cases separately. With further Ramsey arguments and case distinctions, we show that identifications can be used to find a t'-hard biclique pattern or a t'-hard matching pattern appearing in a graph in $\mathcal{D}$, where t' is some unbounded function of t. It follows that if arbitrarily large t-tough pairs appear in $\mathcal{D}$, then $\mathcal{D}$ is a superclass of some $\mathcal{C}_i$.

For full details and proofs, as well as a concluding discussion and open problems, please see the full version of this article [16].

2 Formal definition of a t-tough-pair

In this section we give the formal definition of a t-tough-pair. Further definitions, that are specific to the sections are defined in the beginning of the respective sections.

Given a digraph D and an edge $e = (u, v) \in E(D)$, we say that e is a *minimal* edge of D if D has no (u, v)-path of length strictly greater than 1 in D, where the length of the path is the number of edges in it. We say that a digraph D is *reachability-minimal* if each edge of D is minimal. For an edge $e = (u, v)$ in a directed graph D, v is called the *head* of e and u is called the *tail* of e. For any $E' \subseteq E(D)$, $\texttt{head}(E')$ (resp. $\texttt{tail}(E')$) denotes the set of heads (resp. tails) of the edges in E'. Next we define weak independence and strong independence that are crucial to define the t-tough-pair formally.

▶ **Definition 12** (Weakly independent edges). *Given a digraph D and edges $e_1 = (u_1, v_1), e_2 = (u_2, v_2) \in E(D)$, we say that the pair of edges (e_1, e_2) is weakly independent in D, if $u_1 \neq v_1 \neq u_2 \neq v_2$, and D has neither a (v_1, u_2)-path nor a (v_2, u_1)-path. A set of edges $E' \subseteq E(D)$ is weakly independent if every pair of distinct edges in E' are pairwise weakly independent and for each edge $(u_i, v_i) \in E'$, there is no (v_i, u_i)-path in D.*

Informally, a pair of edges is weakly independent, if the head of one edge cannot reach the tail of the other. Therefore, if a pair of edges is weakly independent, then they cannot lie on a directed path.

▶ **Definition 13** (Strongly independent edges). *Given a digraph D and edges $e_1 = (u_1, v_1), e_2 = (u_2, v_2) \in E(D)$, we say that the pair of edges (e_1, e_2) is strongly independent in D, if they are weakly independent in D, and additionally D has no (u_1, u_2)-path, no (u_2, u_1)-path, no (v_1, v_2)-path and no (v_2, v_1)-path.*

Informally, a pair of edges is strongly independent, if they are weakly independent, and the head of one cannot reach the head of the other, and the tail of one cannot reach the tail of the other. That is, the vertices of the heads (resp. vertices of tails) do not lie on any directed path.

▶ **Definition 14** (t-tough-pair). *Given a digraph D, $E_1, E_2 \subseteq E(D)$, we say that (E_1, E_2) is a tough-pair in D if:*

1. $|E_1| = |E_2|$,
2. *each edge of $E_1 \cup E_2$ is a minimal edge in D,*
3. *all edges in E_i are pairwise weakly independent in D, for both $i \in \{1, 2\}$, and*
4. *for each $e_1 \in E_1$ and $e_2 \in E_2$, (e_1, e_2) are strongly independent in D.*

Further, for a positive integer t, we say that (E_1, E_2) is a t-tough-pair if $|E_1| = |E_2| = t$.

References

1 MohammadHossein Bateni, Chandra Chekuri, Alina Ene, Mohammad Taghi Hajiaghayi, Nitish Korula, and Dániel Marx. Prize-collecting Steiner Problems on Planar Graphs. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1028–1049, 2011. doi:10.1137/1.9781611973082.79.

2 MohammadHossein Bateni, Mohammad Taghi Hajiaghayi, and Dániel Marx. Approximation Schemes for Steiner Forest on Planar Graphs and Graphs of Bounded Treewidth. *J. ACM*, 58(5):21:1–21:37, 2011. doi:10.1145/2027216.2027219.

3 Piotr Berman, Arnab Bhattacharyya, Konstantin Makarychev, Sofya Raskhodnikova, and Grigory Yaroslavtsev. Approximation algorithms for spanner problems and directed steiner forest. *Inf. Comput.*, 222:93–107, 2013. doi:10.1016/j.ic.2012.10.007.

4 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: fast subset convolution. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 67–74. ACM, 2007. doi:10.1145/1250790.1250801.

5 Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation Algorithms for Directed Steiner Problems. *J. Algorithms*, 33(1):73–91, 1999. doi:10.1006/jagm.1999.1042.

6 Rajesh Chitnis, Hossein Esfandiari, Mohammad Taghi Hajiaghayi, Rohit Khandekar, Guy Kortsarz, and Saeed Seddighin. A Tight Algorithm for Strongly Connected Steiner Subgraph on Two Terminals with Demands. *Algorithmica*, 77(4):1216–1239, 2017. doi:10.1007/s00453-016-0145-8.

7 Rajesh Chitnis, Andreas Emil Feldmann, and Pasin Manurangsi. Parameterized Approximation Algorithms for Bidirected Steiner Network Problems. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, pages 20:1–20:16, 2018. doi:10.4230/LIPIcs.ESA.2018.20.

8 Rajesh Hemant Chitnis, Andreas Emil Feldmann, Mohammad Taghi Hajiaghayi, and Dániel Marx. Tight bounds for planar strongly connected steiner subgraph with fixed number of terminals (and extensions). *SIAM J. Comput.*, 49(2):318–364, 2020. doi:10.1137/18M122371X.

9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

10 Éric Colin de Verdière. Multicuts in planar and bounded-genus graphs with bounded number of terminals. *Algorithmica*, 78(4):1206–1224, 2017. doi:10.1007/s00453-016-0258-0.

11 Pavel Dvorák, Andreas Emil Feldmann, Dusan Knop, Tomás Masarík, Tomas Toufar, and Pavel Veselý. Parameterized Approximation Schemes for Steiner Trees with Small Number of Steiner Vertices. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 26:1–26:15, 2018. doi:10.4230/LIPIcs.STACS.2018.26.

12 Eduard Eiben, Dusan Knop, Fahad Panolan, and Ondrej Suchý. Complexity of the Steiner Network Problem with Respect to the Number of Terminals. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, volume 126 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:17, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2019.25.

13 Jon Feldman and Matthias Ruhl. The Directed Steiner Network Problem is Tractable for a Constant Number of Terminals. *SIAM J. Comput.*, 36(2):543–561, 2006. doi:10.1137/S0097539704441241.

14 Andreas Emil Feldmann and Dániel Marx. The complexity landscape of fixed-parameter directed steiner network problems. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, volume 55 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.27.

15 Fedor V. Fomin, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Subexponential parameterized algorithms for planar and apex-minor-free graphs via low treewidth pattern covering. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 515–524. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.62.

16 Esther Galby, Sándor Kisfaludi-Bak, Dániel Marx, and Roohani Sharma. Subexponential parameterized directed steiner network problems on planar graphs: a complete classification, 2022. arXiv:2208.06015.

17 Jiong Guo, Rolf Niedermeier, and Ondrej Suchý. Parameterized complexity of arc-weighted directed steiner problems. *SIAM J. Discrete Math.*, 25(2):583–599, 2011. doi:10.1137/100794560.

18 S. Louis Hakimi. Steiner's problem in graphs and its implications. *Networks*, 1(2):113–133, 1971. doi:10.1002/net.3230010203.

19 Richard M. Karp. Reducibility Among Combinatorial Problems. In *Proceedings of a Symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, pages 85–103, 1972. doi:10.1007/978-1-4684-2001-2_9.

20 Philip N. Klein and Dániel Marx. Solving planar k-terminal cut in $O(n^{c\sqrt{k}})$ time. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, volume 7391 of *Lecture Notes in Computer Science*, pages 569–580. Springer, 2012. doi:10.1007/978-3-642-31594-7_48.

21 Philip N. Klein and Dániel Marx. A subexponential parameterized algorithm for subset TSP on planar graphs. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1812–1830. SIAM, 2014. doi:10.1137/1.9781611973402.131.

22 A Levin. Algorithm for the shortest connection of a group of graph vertices. *Soviet Math. Dokl.*, 12:1477–1481, 1971. doi:10.4086/toc.2010.v006a005.

23 Chung-Lun Li, S. Thomas McCormick, and David Simchi-Levi. The point-to-point delivery and connection problems: complexity and algorithms. *Discrete Applied Mathematics*, 36(3):267–292, 1992. doi:10.1016/0166-218X(92)90258-C.

24 Daniel Lokshtanov, Saket Saurabh, and Magnus Wahlström. Subexponential parameterized odd cycle transversal on planar graphs. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, volume 18 of *LIPIcs*, pages 424–434. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.424.

25 Dániel Marx. On the Optimality of Planar and Geometric Approximation Schemes. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 338–348, 2007. doi:10.1109/FOCS.2007.50.

26 Dániel Marx. A tight lower bound for planar multiway cut with fixed number of terminals. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, volume 7391 of *Lecture Notes in Computer Science*, pages 677–688. Springer, 2012. doi:10.1007/978-3-642-31594-7_57.

27 Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. On subexponential parameterized algorithms for steiner tree and directed subset TSP on planar graphs. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 474–484. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00052.

28 Dániel Marx and Michal Pilipczuk. Optimal parameterized algorithms for planar facility location problems using voronoi diagrams. *ACM Trans. Algorithms*, 18(2):13:1–13:64, 2022. doi:10.1145/3483425.

29 Madan Natu and Shu-Cherng Fang. The Point-to-point Connection Problem - Analysis and Algorithms. *Discrete Applied Mathematics*, 78(1-3):207–226, 1997. doi:10.1016/S0166-218X(97)00010-3.

30 Jesper Nederlof. Detecting and counting small patterns in planar graphs in subexponential parameterized time. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1293–1306. ACM, 2020. doi:10.1145/3357713.3384261.

31 S. Ramanathan. Multicast tree generation in networks with asymmetric links. *IEEE/ACM Trans. Netw.*, 4(4):558–568, 1996. doi:10.1109/90.532865.

32 Hussein F. Salama, Douglas S. Reeves, and Yannis Viniotis. Evaluation of Multicast Routing Algorithms for Real-Time Communication on High-Speed Networks. *IEEE Journal on Selected Areas in Communications*, 15(3):332–345, 1997. doi:10.1109/49.564132.

33 Pawel Winter. Steiner problem in networks: A survey. *Networks*, 17(2):129–167, 1987. doi:10.1002/net.3230170203.

A Tight Subexponential-Time Algorithm for Two-Page Book Embedding

Robert Ganian
Algorithms and Complexity Group, TU Wien, Austria

Haiko Müller
School of Computing, University of Leeds, UK

Sebastian Ordyniak
School of Computing, University of Leeds, UK

Giacomo Paesani
School of Computing, University of Leeds, UK

Mateusz Rychlicki
School of Computing, University of Leeds, UK

Abstract

A book embedding of a graph is a drawing that maps vertices onto a line and edges to simple pairwise non-crossing curves drawn into "pages", which are half-planes bounded by that line. Two-page book embeddings, i.e., book embeddings into 2 pages, are of special importance as they are both NP-hard to compute and have specific applications. We obtain a $2^{\mathcal{O}(\sqrt{n})}$ algorithm for computing a book embedding of an n-vertex graph on two pages – a result which is asymptotically tight under the Exponential Time Hypothesis. As a key tool in our approach, we obtain a single-exponential fixed-parameter algorithm for the same problem when parameterized by the treewidth of the input graph. We conclude by establishing the fixed-parameter tractability of computing minimum-page book embeddings when parameterized by the feedback edge number, settling an open question arising from previous work on the problem.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases book embedding, page number, subexponential algorithms, subhamiltonicity, feedback edge number

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.68

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2404.14087 [24]

Funding *Robert Ganian*: Project No. Y1329 of the Austrian Science Fund (FWF), Project No. ICT22-029 of the Vienna Science Foundation (WWTF).
Sebastian Ordyniak: Project EP/V00252X/1, EPSRC.

1 Introduction

Book embeddings of graphs are drawings centered around a line, called the *spine*, and half-planes bounded by the spine, called *pages*. In particular, a k-page book embedding of a graph G is a drawing which maps vertices to distinct points on the spine and edges to simple curves on one of the k pages such that no two edges on the same page cross [6]. These embeddings have been the focus of extensive study to date [16, 20, 21, 22, 25, 38, 47], among others due to their classical applications in VLSI, bio-informatics, and parallel computing [11, 20, 31].

Every n-vertex graph is known to admit an $\lceil \frac{n}{2} \rceil$-page book embedding [6, 11, 30], but in many cases it is possible to obtain book embeddings with much fewer pages. Particular attention has been paid to two-page embeddings, which have specifically been used, e.g.,

to represent RNA pseudoknots [31, 42]. The class of graphs that can be embedded on two pages was studied by Di Giacomo and Liotta [27], Heath [32] as well as by other authors [1], and was shown to be a superclass of planar graphs with maximum degree at most 4 [5].

While two-page book embeddings are a special class of planar embeddings, they are not polynomial-time computable unless $\mathsf{P} = \mathsf{NP}$. Indeed, a graph admits a two-page book embedding if and only if it is *subhamiltonian* (*i.e.*, is a subgraph of a planar Hamiltonian graph) [6] and testing subhamiltonicity is an NP-hard problem [11]. On the other hand, the aforementioned problem of constructing a two-page book embedding (or determining that none exists) – which we hereinafter call Two-Page Book Embedding – becomes linear-time solvable if one is provided with a specific ordering of the n vertices of the input graph along the spine [31]. While Two-Page Book Embedding can be seen to admit a trivial brute-force $2^{\mathcal{O}(n \cdot \log n)}$ algorithm, it has also been shown to be solvable in $2^{\mathcal{O}(n)}$ time – in particular, one can branch to determine the allocation of edges into the two pages and then solve the problem via dynamic programming on SPQR trees [2, 33, 34].

Contribution. As our main contribution, we break the single-exponential barrier for Two-Page Book Embedding by providing an algorithm that solves the problem in $2^{\mathcal{O}(\sqrt{n})}$ time. Our algorithm is exact and deterministic, and avoids the single-exponential overhead of branching over edge allocations to pages by instead attacking the equivalent subhamiltonicity testing formulation of the problem. It is also asymptotically optimal under the Exponential Time Hypothesis [35]: there is a well-known quadratic reduction that excludes any $2^{o(\sqrt{n})}$ algorithm for Hamiltonian Cycle on cubic planar graphs [26], and a linear reduction from that problem (under the same restrictions) to subhamiltonicity testing [46] then excludes any $2^{o(\sqrt{n})}$ algorithm for our problem of interest.

The central component of our result is a non-trivial dynamic programming procedure that solves Two-Page Book Embedding in time $2^{\mathcal{O}(tw)} \cdot n$, where tw is the treewidth of the input graph. The desired subexponential algorithm then follows by the well-known fact that n-vertex planar graphs have treewidth at most $\mathcal{O}(\sqrt{n})$ [28, 39, 44]. But in addition to that, we believe our single-exponential treewidth-based algorithm to be of independent interest also in the context of parameterized algorithmics [13, 19].

Indeed, while Two-Page Book Embedding was already shown to be fixed-parameter tractable w.r.t. treewidth (i.e., to admit an algorithm running in time $f(tw) \cdot n$) by Bannister and Eppstein [3], that result crucially relied on Courcelle's Theorem [12]. More specifically, they showed that the required property can be encoded via a constant-size sentence in Monadic Second Order logic, which suffices for fixed-parameter tractability – but unfortunately not for a single-exponential algorithm, and a direct dynamic programming algorithm based on the characterization employed there seems to necessitate a parameter dependency that is more than single-exponential. Moreover, it is not at all obvious how one could employ convolution-based tools – which have successfully led to $2^{\mathcal{O}(tw)} \cdot n$ algorithms for, e.g., Hamiltonian Cycle [10, 14, 15] – for our problem of interest here.

Instead, we obtain our results by employing dynamic programming along a *sphere-cut decomposition* – a type of branch decomposition specifically designed for planar graphs of small treewidth [18]. However, unlike in previous applications of sphere-cut decompositions [36, 40], our algorithm requires the nooses delimiting the bags in the sphere-cut decomposition to admit a fixed drawing since our arguments rely on constructing a hypothetical solution (a subhamiltonian curve) that is "well-behaved" w.r.t. a fixed set of curves. While this would typically lead to extensive case analysis to compute the records of a parent noose from the records of the children, we introduce a generic framework that allows us to transfer records

from child to parent nooses via XOR operations. We believe that this technique may be of broader interest, specifically when working with problems which require one to enhance the embedding or drawing of an input graph.

In the final part of the article, we turn our attention to the parameterized complexity of computing book embeddings. While Two-Page Book Embedding is fixed-parameter tractable when parameterized by the treewidth of the input graph, the only graph parameter which has been shown to yield fixed-parameter algorithms for computing ℓ-page book embeddings for $\ell > 2$ is the *vertex cover number*[1] [7]. Whether this tractability result also holds for other structural graph parameters such as treewidth, *treedepth* [41] or the *feedback edge number* [45] has been stated as an open question in the field[2]. We conclude by providing a novel fixed-parameter algorithm for computing ℓ-page book embeddings (or determining that one does not exist) under the third parameterization mentioned above – the feedback edge number, i.e., the edge deletion distance to acyclicity. This result is complementary to the known vertex-cover based fixed-parameter algorithm, and can be seen as a necessary stepping stone towards eventually settling the complexity of computing ℓ-page book embeddings parameterized by treewidth. Moreover, since the obtained kernel is linear in the case of $\ell = 2$, the obtained kernel allows us to generalize our main algorithmic result to a run-time of $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$ where k is the feedback edge number of the input graph.

2 Preliminaries

Basic Notions. We use basic terminology for graphs and multi-graphs [17], and assume familiarity with the basic notions of parameterized complexity and fixed-parameter tractability [13, 19]. The *feedback edge number* of G, denoted by $\mathsf{fen}(G)$, is the minimum size of any *feedback edge set* of G, *i.e.*, a set $F \subseteq E(G)$ such that $G - F = (V(G), E(G) \setminus F)$ is acyclic.

For a face f of a plane graph, we use $\sigma(f)$ to denote the cyclic sequence of the vertices obtained by traversing the closed curve representing the border of f in a clock-wise manner.

Book Embeddings and Subhamiltonicity. An ℓ-page book embedding of a multi-graph $G = (V, E)$ will be denoted by a pair $\langle \prec, \sigma \rangle$, where $\prec$ is a linear order of V, and $\sigma \colon E \to [\ell]$ is a function that maps each edge of E to one of ℓ pages $[\ell] = \{1, 2, \ldots, \ell\}$. In an ℓ-page book embedding $\langle \prec, \sigma \rangle$ it is required that for no pair of edges $uv, wx \in E$ with $\sigma(uv) = \sigma(wx)$ the vertices are ordered as $u \prec w \prec v \prec x$, i.e., each page must be crossing-free. The *page number* of a graph G is the minimum number ℓ such that G admits an ℓ-page book embedding. The general problem of computing the page number of an input graph is thus:

Book Thickness	
Instance:	A multi-graph G with n vertices and a positive integer ℓ.
Question:	Does G admit a ℓ-page book embedding?

It is known that a multi-graph admits a 2-page book embedding if and only if it is *subhamiltonian*, i.e., if it has a planar Hamiltonian supergraph on the same vertex set [6]; an illustration is provided in Figure 1. Hence, the problem of deciding whether a multi-graph has page number 2 can be equivalently stated as:

[1] The vertex cover number is the minimum size of a vertex cover, and represents a much stronger restriction on the structure of the input graphs than, e.g., treewidth.

[2] E.g., at **Advances in Parameterized Graph Algorithms** (Spain, May 2–7 2022) and also at Dagstuhl seminar 21293 **Parameterized Complexity in Graph Drawing** [23].

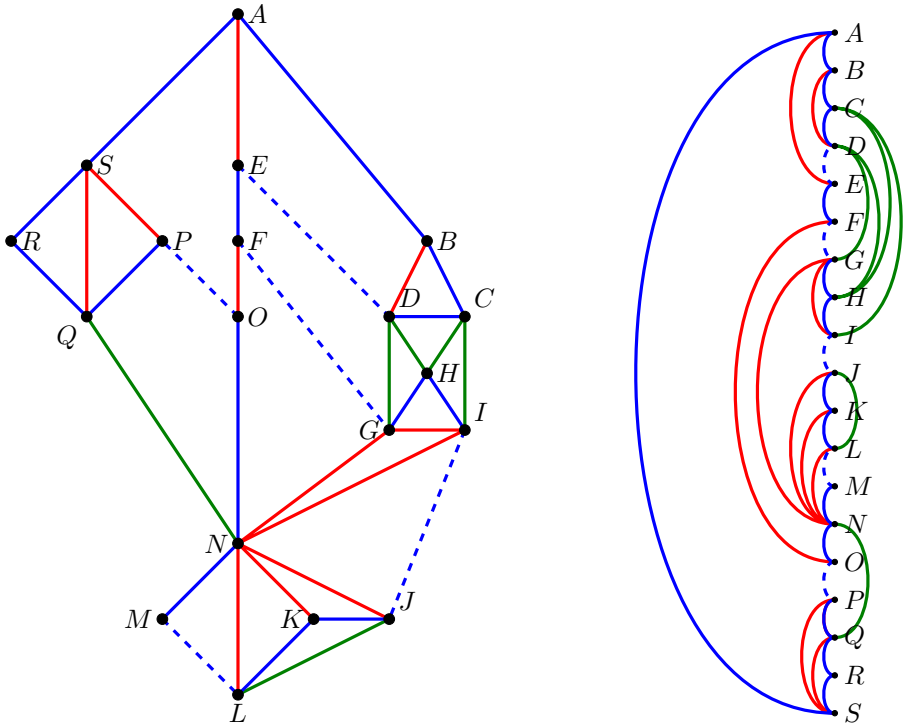


Figure 1 A drawing of a subhamiltonian graph G, made of the full-edges, which is completed by the dashed edges to one of its Hamiltonian supergraphs G_H (left) and the same graph drawn as a two-page book embedding (right). In both drawings the Hamiltonian cycle H is colored in blue and the edges belonging to page 1 and 2 are colored with green and red, respectively.

Subhamiltonicity (SUBHAM)	
Instance:	A multi-graph G with n vertices.
Question:	Is G subhamiltonian?

Since the transformation between 2-page book embeddings and Hamiltonian cycles of supergraphs is constructive in both directions, a constructive algorithm for SUBHAM (such as the one presented here) allows us to also output a 2-page book embedding for the graph.

Let G be subhamiltonian. For a Hamiltonian cycle H on $V(G)$ (where H is not necessarily a subgraph of G), we denote by G_H the graph obtained from G after adding the edges of H and say that H is a *witness* for G if G_H is planar. A drawing D of G *respects* H if D can be completed to a planar drawing of G_H by only adding the edges of H. We extend the notion of "witness" to include all the information defining the solution as follows: a tuple (D, D_H, G_H, H) is a *witness* for G if G_H is a planar supergraph of G containing the Hamiltonian cycle H, D_H is a planar drawing of G_H, and D is the restriction of D_H to G; note that D_H witnesses that D respects H.

SPQR-Trees. We assume familiarity with the SPQR-tree data structure for biconnected multi-graphs which decomposes a graph into (S)eries, (P)arallel, (R)igid and (Q) nodes (leaf nodes and root node), following the formalism used by Gutwenger et al. [29], see also [4, 8, 9]. For a node b in an SPQR-tree, we use $\textsc{Sk}(b)$ and $\textsc{Pe}(B)$ to denote the *skeleton* and *pertinent graph* of b, respectively. SPQR-trees can be computed in linear time, and an illustration of the data structure is provided in Figure 2.

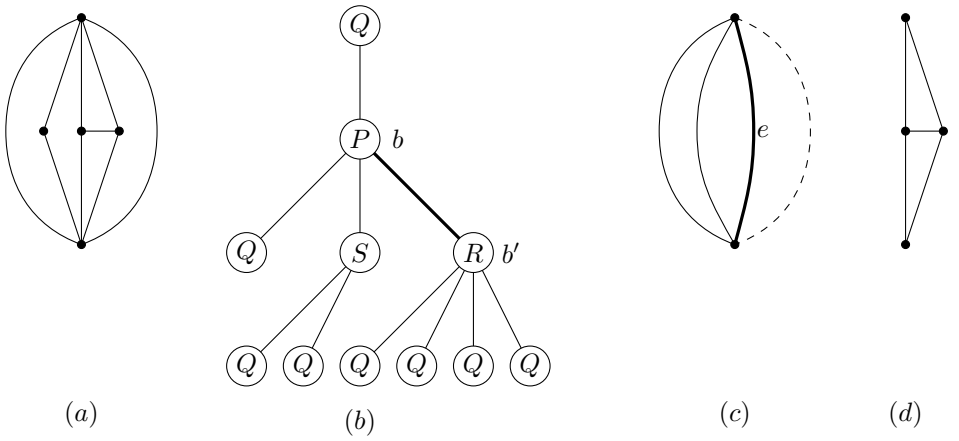


Figure 2 (a) shows a biconnected multi-graph G. (b) shows the SPQR-tree $\mathcal{B}$ of G. (c) shows the skeleton of b, $\textsc{Sk}(b)$, where the edge e that corresponds to the child (with pertinent node) b' is in bold and the dashed edge represents the reference edge. Finally, (d) shows $\textsc{Pe}(b')$.

Sphere-Cut Decompositions. A branch decomposition $\langle T, \lambda \rangle$ of a graph G consists of an unrooted ternary tree T (meaning that each node of T has degree one or three) and of a bijection $\lambda : \mathcal{L}(T) \leftrightarrow E(G)$ from the leaf set $\mathcal{L}(T)$ of T to the edge set $E(G)$ of G; to distinguish $E(T)$ from $E(G)$, we call the elements of the former *arcs* (as was also done in previous work [18]). For each arc a of T, let T_1 and T_2 be the two connected components of $T - a$, and, for $i = 1, 2$, let G_i be the subgraph of G that consists of the edges corresponding to the leaves of T_i, i.e., the edge set $\{\lambda(\mu) : \mu \in \mathcal{L}(T) \cap V(T_i)\}$. The middle set $mid(a) \subseteq V(G)$ is the intersection of the vertex sets of G_1 and G_2, i.e., $mid(a) := V(G_1) \cap V(G_2)$. The width $\beta(\langle T, \lambda \rangle)$ of $\langle T, \lambda \rangle$ is the maximum size of the middle sets over all arcs of T, i.e., $\beta(\langle T, \lambda \rangle) := max\{|mid(a)| : a \in E(T)\}$. An optimal branch decomposition of G is a branch decomposition with minimum width; this width is called the branchwidth $\beta(G)$ of G. We will need the following well-known relation between treewidth and branchwidth.

▶ **Lemma 1** ([43, Theorem 5.1]). *Let G be a graph. Then, $bw(G) - 1 \leq tw(G) \leq \frac{3}{2} bw(G) - 1$, where $bw(G)$ is the branchwidth and $tw(G)$ is the treewidth of G.*

Let D be a plane drawing of a connected planar graph G. A noose of D is a closed simple curve that (i) intersects D only at vertices and (ii) traverses each face at most once, i.e., its intersection with the region of each face forms a connected curve. The length of a noose is the number of vertices it intersects, and every noose O separates the plane into two regions δ_1 and δ_2. A *sphere-cut decomposition* $\langle T, \lambda, \Pi = \{ \pi_a \mid a \in E(T) \} \rangle$ of (G, D) is a branch decomposition$\langle T, \lambda \rangle$ of G together with a set Π of circular orders π_a of $mid(a)$ – one for each arc a of T – such that there exists a noose O_a whose closed discs δ_1 and δ_2 enclose the drawing of G_1 and of G_2, respectively. Observe that O_a intersect G exactly at $mid(a)$ and its length is $|mid(a)|$. Note that the fact that G is connected together with Conditions (i) and (ii) of the definition of a noose implies that the graphs G_1 and G_2 are both connected and that the set of nooses forms a laminar set family, that is, any two nooses are either disjoint or nested. A clockwise traversal of O_a in the drawing of G defines the cyclic ordering π_a of $mid(a)$. We always assume that the vertices of every middle set $mid(a)$ are enumerated according to π_a. A sphere-cut decomposition of a given planar graph with n vertices can be constructed in $\mathcal{O}(n^3)$ time [18].

We say that a biconnected planar multi-graph G equipped with an SPQR-tree $\mathcal{B}$ is *associated* with a set $\mathcal{T}$ of sphere-cut decompositions if $\mathcal{T}$ contains a sphere-cut decomposition of $\textsc{Sk}(b)$ for every R-node and every S-node b of $\mathcal{B}$.

▶ **Lemma 2.** *Let G be biconnected planar multi-graph with planar drawing D and SPQR-tree $\mathcal{B}$ of G together with the associated set $\mathcal{T}$ of sphere-cut decompositions. Then, D can be extended to a planar drawing D' of G together with all nooses in $\{ O_a \mid a \in E(T_b) \wedge \langle T_b, \lambda_b, \Pi_b \rangle \in \mathcal{T} \}$ as well as a noose N_b for every node b of $\mathcal{B}$ satisfying:*

- *N_b intersects with D only at s_b and t_b.*
- *N_b separates $P_E(b)$ from $G \setminus P_E(b)$ in D.*

Moreover, if any of the subcurves of the nooses O_a and the nooses N_b connect the same two vertices in the same face of D, then the two subcurves are identical in D'.

Non-Crossing Matchings. Let K_n be the complete graph on vertices $\{1, \ldots, n\}$ and let $<$ be a cyclic ordering of the elements in $\{1, \ldots, n\}$. A *non-crossing matching* is a matching M in the graph K_n such that for every two edges $\{a, b\}, \{c, d\} \in M$ it is not the case that $a < c < b < d$.

3 Solution Normal Form

Our first order of business is to show that we can assume that the solution (Hamiltonian cycle) to the SUBHAM problem interacts with the drawing in a restricted manner. In particular, we aim to show that every subhamiltonian graph G has a witness (D, D_H, G_H, H) in *normal form*, i.e., with the following property: it is possible to draw a curve in D_H between any two vertices occurring in a common face of D such that this curve only crosses the Hamiltonian cycle at most twice. Note that this property will allow us to bound the number of possible interactions of the Hamiltonian cycle with any subgraph corresponding to either a node in the SQPR-tree or an arc in a sphere-cut decomposition and is crucial to bound the number of types in our dynamic programming algorithm. The following lemma is the main technical lemma behind our normal form. An illustration of the main ideas behind the proof is provided in Figure 3.

▶ **Lemma 3.** *Let G be a subhamiltonian graph with witness (D, D_H, G_H, H), let f be a face of D and let c be a curve drawn inside f between two vertices $u, v \in V(f)$. Then, there is a witness $(D, D_{H'}, G_{H'}, H')$ for G such that:*

(1) *$D_{H'}$ and D_H differ only inside f.*
(2) *c crosses at most two curves corresponding to the edges of H'.*
(3) *c crosses each curve corresponding to an edge of H' at most once.*

We are now ready to define our normal form for the Hamiltonian cycle. Essentially, we show that if there is a Hamiltonian cycle, then there is one which crosses each subcurve that is either part of the border of a node in the SPQR-tree or that is a subcurve of some noose in a sphere-cut decomposition of an R-node or an S-node at most twice.

Let G be a biconnected subhamiltonian multi-graph with SPQR-tree $\mathcal{B}$ and the associated set $\mathcal{T}$ of sphere-cut decompositions $\langle T_b, \lambda_b, \Pi_b \rangle$ of $\textsc{Sk}(b)$ for every R-node and S-node b of $\mathcal{B}$. We say that a witness $W = (D, D_H, G_H, H)$ for G *respects* the sphere-cut decompositions in $\mathcal{T}$, if there is a planar drawing of all nooses in the sphere-cut decompositions of $\mathcal{T}$ into D such that every subcurve c in $\bigcup_{a \in E(T_b)} O_a$ crosses the curves corresponding to the edges of H at most twice in D_H. We say that the witness W for G *respects* $\mathcal{B}$ if it respects the sphere-cut decompositions in $\mathcal{T}$ and for every node b of $\mathcal{B}$ with reference edge (s_b, t_b), it holds that there is a noose N_b that can be drawn into D_H such that:

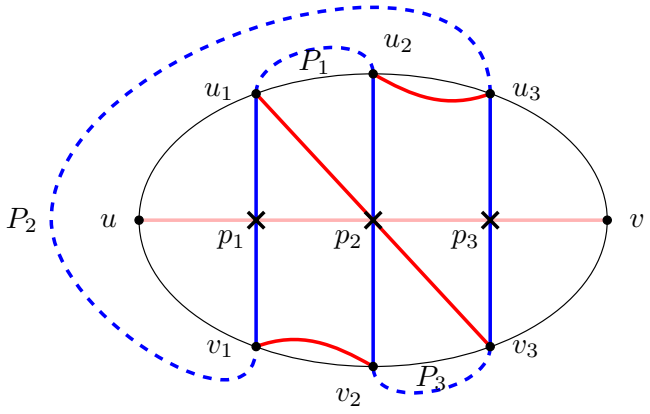


Figure 3 The cycle $H = (u_2, P_1, u_1, v_1, P_2, u_3, v_3, P_3, v_2, u_2)$ represents a Hamiltonian cycle that crosses the uv-curve at least three times (in p_1, p_2 and p_3). Thanks to Lemma 3, we obtain a Hamiltonian cycle $H' = (u_2, P_1, u_1, v_3, P_3, v_2, v_1, P_2, u_3, u_2)$ that differs from H only inside the face $f = (u, u_1, u_2, u_3, v, v_3, v_2, v_1)$ and crosses the uv-curve two fewer times than H does. Finally, note that the vertices u and v are part of either P_1, P_2, or P_3.

- N_b touches D only at s_b and t_b.
- N_b separates $\text{PE}(b)$ from $G \setminus \text{PE}(b)$ in D.
- Each of the two subcurves L_b and R_b obtained from N_b by splitting N_b at s_b and t_b crosses the curves corresponding to the edges of H at most twice.
- Moreover, if any of the subcurves of the nooses O_a and the nooses N_b connect the same two vertices in the same face of D, then the two subcurves are identical.

The following lemma allows us to assume our normal form and follows easily from a repeated application of Lemma 3.

▶ **Lemma 4.** *Let G be a biconnected subhamiltonian multi-graph with SPQR-tree $\mathcal{B}$ and the associated set $\mathcal{T}$ of sphere-cut decompositions. Then, there is a witness $W = (D, D_H, G_H, H)$ for G that respects $\mathcal{B}$.*

4 Setting Up the Framework

In this section we provide the foundations for our algorithm. That is, in Subsection 4.1, we show that it suffices to consider biconnected graphs allowing us to employ SPQR-trees. We then define the types for nodes in the SPQR-tree, which we compute in our dynamic programming algorithm on SPQR-trees, in Subsection 4.2. Finally, in Subsection 4.3 we introduce our general framework for simplifying dynamic programming algorithms on sphere-cut decompositions and introduce the types for nodes of a sphere-cut decomposition.

4.1 Reducing to the Biconnected Case

We begin by showing that any instance of SUBHAM can be easily reduced to solving the same problem on the biconnected components of the same instance. It is well-known that SUBHAM can be solved independently on each connected component of the input graph, the following theorem now also shows that the same holds for the biconnected components of the graph and allows us to employ SPQR-trees for our algorithm.

▶ **Theorem 5.** *Let G be a graph and let $C \subseteq V(G)$ such that $N(C) = \{n\}$, where $N(C) = \{ v \in V(G) \setminus C \mid \exists c \in C \ \{v, c\} \in E(G) \}$ is the set of neighbors of any vertex of C in $V(G) \setminus C$. Then G is subhamiltonian if and only if both $G^- = G - C$ and $G^C = G[C \cup \{n\}]$ are subhamiltonian.*

4.2 Defining the Types for Nodes in the SPQR-tree

Here, we define the types for nodes in the SPQR-tree that we will later compute using dynamic programming. In the following, we assume that G is a biconnected multi-graph with SPQR-tree $\mathcal{B}$ and the associated set $\mathcal{T}$ of sphere-cut decompositions. Let b be a node of $\mathcal{B}$ with pertinent graph $\textsc{Pe}(b)$ and reference edge $e = (s, t)$. A *type* of b is a triple (ψ, M, S) such that (please refer also to Figure 4 for an illustration of some types):

- ψ is a function from $\{L, R\}$ to subsets of $\{l, l', r, r'\}$ such that $\psi(L) \in \{\emptyset, \{l\}, \{l, l'\}\}$ and $\psi(R) \in \{\emptyset, \{r\}, \{r, r'\}\}$. We denote by $V(\psi)$ the set $\psi(L) \cup \psi(R)$. Informally, ψ captures how many times the Hamiltonian cycle enters and exits the graph $\textsc{Pe}(b)$ from the left (L) and from the right (R).
- $M \subseteq \{ \{u, v\} \mid u, v \in \{s, t\} \cup V(\psi) \wedge u \neq v \}$ and M is a non-crossing matching w.r.t. the circular ordering (s, r, r', t, l', l) that matches all vertices in $V(\psi)$ (i.e. $V(\psi) \subseteq V(M)$), where $V(M) = \bigcup_{e \in M} e$. Informally, M captures the maximal path segments of the Hamiltonian cycle inside $\textsc{Pe}(b) \cup V(\psi)$ with endpoints in $\{s, t\} \cup V(\psi)$.
- $S \subseteq \{s, t\} \setminus V(M)$. Informally, S captures whether s or t are contained as inner vertices on path segments corresponding to M.

We now provide the formal semantics of types; see Figure 4 for an illustration. Let $\mathcal{X}$ be the set of all types and $\textsc{Pe}^*(b)$ be the graph obtained from $\textsc{Pe}(b)$ after adding the dummy vertices l, l', r, and r' together with the edges sl, ll', $l't$, sr, rr', and $r't$. We say that b has type $X = (\psi, M, S)$ if there is a set $\mathcal{P}$ of vertex-disjoint paths or a single cycle in the complete graph with vertex set $V(\textsc{Pe}^*(b))$ such that:

- $\mathcal{P}$ consists of exactly one path P_e between u and v for every $e = \{u, v\} \in M$ or $\mathcal{P}$ is a cycle and $M = \emptyset$.
- $\{ \text{IN}(P) \mid P \in \mathcal{P} \}$ is a partition of $(V(\textsc{Pe}(b)) \setminus \{s, t\}) \cup S$, where $\text{IN}(P)$ denotes the set of inner vertices of P.
- there is a planar drawing $D(b, X)$ of $\textsc{Pe}^*(b) \cup \bigcup_{P \in \mathcal{P}} P$ with outer-face f such that $\sigma(f) = \{s, r, r', t, l', l\}$.

The way we define the types $X = (\psi, M, S)$ of a node b allows us to associate each witness $W = (D, D_H, G_H, H)$ with a type, denoted by $\Gamma_W(b)$, based on the restriction of the witness to the respective pertinent graph.

4.3 Framework for Sphere-cut Decomposition

Here, we introduce our framework to simplify the computation of records via bottom-up dynamic programming along a sphere-cut decomposition. Since the framework is independent of the type of records one aims to compute, we believe that the framework is widely applicable and therefore interesting in its own right. In particular, we introduce a simplified framework for computing the types of arcs (or, equivalently, nooses) in sphere-cut decompositions.

Indeed, the central ingredient of any dynamic programming algorithm on sphere-cut decompositions is a procedure that given an inner node with parent arc a_P and child arcs a_L and a_R computes the set of types for the noose O_{a_P} from the set of types for the nooses O_{a_L} and O_{a_R}. Unfortunately, there is no simple way to obtain O_{a_P} from O_{a_L} and O_{a_R} and this is why computing the set of types for O_{a_P} from the set of types for O_{a_L} and O_{a_R} usually involves a technical and cumbersome case distinction [18]. To circumvent this issue, we

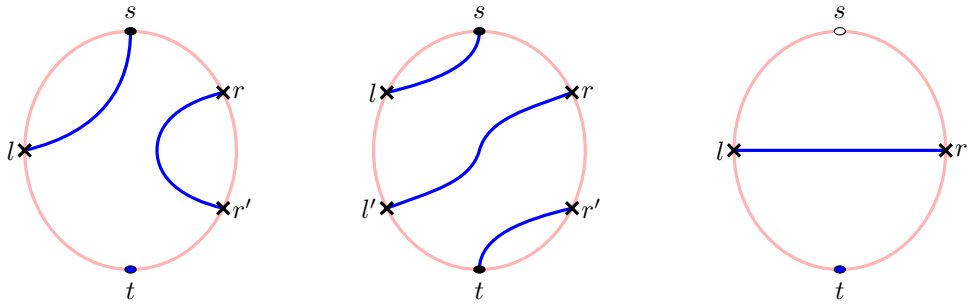


Figure 4 The figure shows three different types of a node in an SPQR-tree with reference edge (s,t), i.e., the types shown are (from left to right): $(\{\{L \to \{l\}\}, \{R \to \{r,r'\}\}, \{\{l,s\},\{r,r'\}\}, \{t\})$, $(\{\{L \to \{l,l'\}\}, \{R \to \{r,r'\}\}, \{\{l,s\},\{l',r\},\{t,r'\}\}, \emptyset)$, and $(\{\{L \to \{l\}\}, \{R \to \{r\}\}, \{\{l,r\}\}, \{t\})$. The subset of $\{l,l'\}$ and $\{r,r'\}$ that appears corresponds to $\psi(L)$ and $\psi(R)$ respectively. The blue edges correspond to the matching M and the blue vertices corresponds to S.

introduce a simple operation, i.e., the $\oplus$ (**XOR**) operation defined below, and show that the noose O_{a_p} can be obtained from the nooses O_{a_L} and O_{a_R} using merely a short sequence – one of length at most four – of $\oplus$ operations.

Central to our framework is the notion of *weak nooses*, which are defined below and can be seen as intermediate results in the above-mentioned sequence of simple operations from the child nooses to the parent noose; in particular, weak nooses are made up of subcurves of the nooses in the sphere-cut decomposition. Let G be a biconnected multi-graph and let $\mathcal{B}$ be an SPQR-tree of G. Let b be an R-node or S-node of $\mathcal{B}$ with pertinent graph $\textsc{Pe}(b)$. Let $\langle T_b, \lambda_b, \Pi_b\rangle$ be a sphere-cut decomposition of $\textsc{Sk}(b)$ and a be an arc of T_b with pertinent graph $\textsc{Pe}(b,a)$. Let $C(T_b)$ be the set of all subcurves of all nooses occurring in T_b, i.e., $C(T_b) = \bigcup_{a \in E(T_b)} O_a$ where O_a is seen as a set of subcurves. We say O is a *weak noose* if O is a noose consisting only of subcurves from $C(T_b)$. For each $O \subseteq C(T_b)$, let $V(O)$ be equal to the vertices of G touched by the noose O.

Having defined weak nooses, we will now define our simplified operation. Let $A \oplus B$ be an exclusive or for two sets A and B, i.e. $A \oplus B = (A \cup B) \setminus (A \cap B)$. We will apply the $\oplus$-operation to weak nooses, whose $\oplus$ is again a weak noose. The following lemma, whose setting is illustrated in Figure 5, is central to our framework as it shows that we can always obtain the noose for the parent arc a_P from the nooses of the child arcs a_L and a_R using a short sequence of $\oplus$-operations such that every intermediate result is a weak noose.

▶ **Lemma 6.** *Let a_P be a parent arc with two child arcs a_L and a_R in a sphere-cut decomposition $\langle T, \lambda, \Pi\rangle$ of a biconnected multi-graph G with the drawing D. There exists a sequence Q of at most 3 $\oplus$-operations such that:*

- *each step generates a weak noose O with $|O| \leq 1 + \max\{|mid(a_P)|, |mid(a_L)|, |mid(a_R)|\}$ as the $\oplus$-operation of two weak nooses O_1 and O_2, whose inside region contains all subcurves in $(O_1 \cap O_2)$,*
- *the last step generates the noose O_{a_P},*
- *Q contains O_{a_L} and O_{a_R} and at most two new weak nooses, each of them bounds the edge-less graph of size 3.*

We are now ready to define the types of weak nooses, which informally can be seen as a generalization of the types of nodes in an SPQR-tree introduced in Subsection 4.2. An illustration of the types is also provided in Figure 7. In the following we fix an arbitrary order π_G of the vertices in G. A type of a weak noose O is a triple (ψ, M, S) such that:

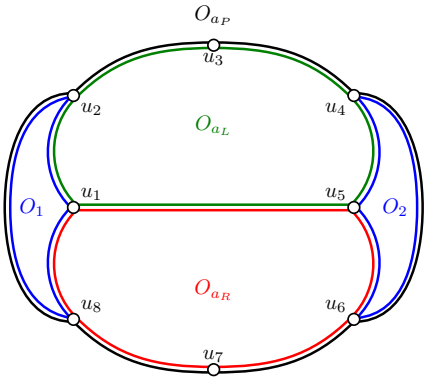


Figure 5 An illustration of the relationship of the parent noose O_{a_P} and the child nooses O_{a_L} and O_{a_R}. The illustration represents the case of Lemma 6 where $O' = O_{a_P} \oplus O_{a_L} \oplus O_{a_R}$ consists of two disjoint weak nooses (triangles) O_1 and O_2.

(1) ψ is a function that for each subcurve $c = (\{u, v\}, f)$ in O, i.e., the subcurve of O between u and v in face f, returns a sequence of at most two new nodes, (2) S is a subset of $V(O)$, and (3) $M \subseteq \{\, \{u, v\} \mid u, v \in V(\psi) \cup (V(O) \setminus S) \wedge u \neq v \,\}$, $V(\psi) \subseteq V(M)$, and M is a non-crossing matching w.r.t. the circular order $\pi^\circ(\psi)$ defined as follows. $\pi^\circ(\psi)$ is the circular order obtained from the circular order $\pi^\circ(O)$ of $V(O)$ after adding $\psi(c)$ between u and v, for every $c = (\{u, v\}, f) \in O$ assuming that $\pi_G(u) < \pi_G(v)$.

The semantics for the types as well as the definition of a type given a witness are now defined in a similar way as in the case of types for SPQR-tree nodes.

5 An FPT-algorithm for SUBHAM using Treewidth

In this section we show that SUBHAM admits a constructive single-exponential fixed-parameter algorithm parameterized by treewidth.

▶ **Theorem 7.** SUBHAM *can be solved in time* $2^{\mathcal{O}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$*, where* tw *is the treewidth of the input graph.*

Since the treewidth of an n-vertex planar graph is upper-bounded by $\mathcal{O}(\sqrt{n})$ [28, 39, 44] and there are single-exponential constant-factor approximation algorithms for treewidth [37], Theorem 7 immediately implies the following corollary.

▶ **Corollary 8.** SUBHAM *can be solved in time* $2^{\mathcal{O}(\sqrt{n})}$.

The main component used towards proving Theorem 7 is the following lemma, from which Theorem 7 follows as an easy consequence .

▶ **Lemma 9.** *Let G be a biconnected multi-graph with n vertices and m edges and SPQR-tree $\mathcal{B}$. Then, we can decide in time $\mathcal{O}(315^\omega n + n^3)$ whether G is subhamiltonian, where ω is the maximum branchwidth of $S\kappa(b)$ over all R-nodes and S-nodes b of $\mathcal{B}$.*

The remainder of this section is therefore devoted to a proof of Lemma 9, which we show by providing a bottom-up dynamic programming algorithm along the SPQR-tree of the graph. That is, let G be a biconnected multi-graph, $\mathcal{B}$ be an SPQR-tree of G with associated set $\mathcal{T}$ of sphere-cut decompositions for every R-node and S-node of $\mathcal{B}$. Using a dynamic programming algorithm starting at the leaves of $\mathcal{B}$, we will compute a set $\mathcal{R}(b)$ of all types X satisfying the following two conditions:

(R1) If $X \in \mathcal{R}(b)$, then b has type X.
(R2) If there is a witness $W = (D, D_H, G_H, H)$ for G that *respects* $\mathcal{B}$ such that b has type $X = \Gamma_W(b)$, then $X \in \mathcal{R}(b)$.

Interestingly, we do not know whether it is possible to compute the set of all types X such that b has type X as one would usually expect to be able to do when looking at similar algorithms based on dynamic programming. That is, we do not know whether one can compute the set of types that also satisfies the reverse direction of **(R1)**. While we do not know, we suspect that this is not the case because b might have a type that can only be achieved by crossing some sub-curves of nooses inside of $\textsc{Pe}(b)$ more than twice. Indeed Lemma 3, which allows us to avoid more than two crossings per sub-curve, requires the property that the type of b can be extended to a Hamiltonian cycle of the whole graph, which is clearly not necessarily the case for every possible type of b.

5.1 Handling P-nodes

In this part, we show how to compute the set of types for any P-node in the given SPQR-tree by establishing the following lemma.

▶ **Lemma 10.** *Let b be a P-node of $\mathcal{B}$ such that $\mathcal{R}(c)$ has already been computed for every child c of b in $\mathcal{B}$. Then, we can compute $\mathcal{R}(b)$ in time $\mathcal{O}(\ell)$, where ℓ is the number of children of b in $\mathcal{B}$.*

In the following, let b be a P-node of $\mathcal{B}$ with reference edge (s,t) and let C with $|C| = \ell$ be the set of all children of b in $\mathcal{B}$. Informally, $\mathcal{R}(b)$ is the set of types X such that there is an ordering $\rho = (c_1, \ldots, c_\ell)$ of the children in C and an assignment $\tau : C \to \mathcal{X}$ of children to types with $\tau(c) \in \mathcal{R}(c)$ for every child $c \in C$ that "realizes" the type X for b. The main challenge is to compute $\mathcal{R}(b)$ efficiently, i.e., without having to enumerate all possible orderings ρ and assignments τ. Below, we make this intuition more precise before proceeding.

For a type $X = (\psi, M, S)$ of b and $A \in \{L, R\}$, we let $\#_A(X) = |\psi(A)|$. Moreover, for every $A \in \{s,t\}$, we set $\#_A(X)$ to be equal to 2 if $A \in S$, equal to 1 if $A \in V(M)$ and equal to 0 otherwise. Next, let $\rho = (X_1, \ldots, X_\ell)$ be a sequence of types, where $X_i = (\psi_i, M_i, S_i)$ for every i with $1 \leq i \leq \ell$. We say that ρ is *weakly compatible* if the following holds:
(C1) for every i with $1 \leq i < \ell$, $\#_R(X_i) = \#_L(X_{i+1})$, and
(C2) $\sum_{i=1}^{\ell} \#_s(X_i) \leq 2$ and $\sum_{i=1}^{\ell} \#_t(X_i) \leq 2$.

Note that **(C1)** corresponds to our assumption made in Lemma 4 that we can add the nooses N_b to any planar drawing D of G such that every face of D contains at most one subcurve of any N_b. This in particular means that if $\textsc{Pe}(c)$ is drawn immediately to the left of $\textsc{Pe}(c')$ for two children c and c' of b, then the subcurves R_c and $L_{c'}$ are identical. Please also refer to Figure 6 for an illustration of these subcurves.

Let ρ be weakly compatible. We define the following auxiliary graph $H(\rho)$. $H(\rho)$ has two vertices s and t and additionally for every i with $1 \leq i \leq \ell$ and every vertex $v \in V(\psi)$, $H(\rho)$ has a vertex v_i. For convenience, we also use s_i and t_i to refer to s and t, respectively. Moreover, $H(\rho)$ has the following edges:

- for every $1 \leq i \leq \ell$ if $M_i = \emptyset$ and $S_i = \{s_i, t_i\}$, $H(\rho)$ has a cycle on s_i and t_i,
- for every $1 \leq i \leq \ell$ if $M_i \neq \emptyset$ then for every $e = \{u, v\} \in M_i$, $H(\rho)$ has the edge $\{u_i v_i\}$,
- for every $1 \leq i < \ell$, $H(\rho)$ contains the edge $\{r_i, l_{i+1}\}$ if $r \in \psi_i(R)$ and $l \in \psi_{i+1}(L)$,
- for every $1 \leq i < \ell$, $H(\rho)$ contains the edge $\{r'_i, l'_{i+1}\}$ if $r' \in \psi_i(R)$ and $l' \in \psi_{i+1}(L)$.

We say that ρ is *compatible* if it is weakly compatible and furthermore either $H(\rho)$ is acyclic, or $H(\rho) - (\bigcup_{i=1}^{\ell} S_i)$ is a single (Hamiltonian) cycle.

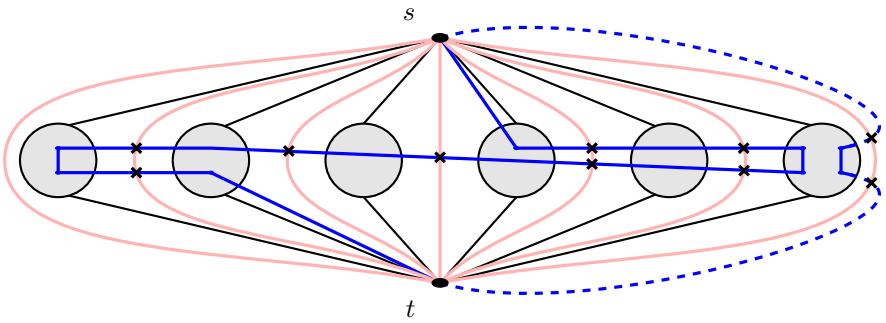


Figure 6 An illustration of how a Hamiltonian Cycle in normal form can interact with a drawing of $\text{PE}(b)$ for a P-node b. Here, the pertinent graphs $\text{PE}(c)$ for all children c of b (without the nodes s and t of the common reference edge (s,t)) are represented by gray ellipses. The Hamiltonian cycle is given in blue with dashed segments representing path segments outside of $\text{PE}(b)$. The red curves represent the subcurves of N_c for every child c of b. In this figure all but the types of the second and fourth pertinent graph are clean. Moreover, the type of the third and fifth pertinent graphs are 1-good and 2-good, respectively, and the types of all other pertinent graphs are bad.

In the following let $\rho = (X_1, \dots, X_\ell)$ be compatible. We now define the type X associated with ρ, which we denote by $X(\rho)$, as follows. If $H(\rho)$ is a single cycle and $\{s,t\} \subseteq \bigcup_{i=1}^{\ell} S_i$, then we set $X(\rho) = (\psi, \emptyset, \{s,t\})$, where $\psi(L) = \psi(R) = \emptyset$. Otherwise, let $\mathcal{P}(\rho)$ be the set of paths in $H(\rho)$, which can be shown to have their endpoints in $\{s,t,l_1,l'_1,r_\ell,r'_\ell\}$. Then, we set $X(\rho) = (\psi, M, S)$, where ψ, M, and S are defined as follows. M contains the set $\{u,v\}$ for every path in $\mathcal{P}(\rho)$ with endpoints u and v; for brevity, we denote l_1, l'_1, r_ℓ, r'_ℓ as l, l', r, r', respectively. Moreover, $\psi(L) = V(M) \cap \{l,l'\}$, $\psi(R) = V(M) \cap \{r,r'\}$, and S contains s (t) if $\sum_{i=1}^{\ell} \#_s(X_i) = 2$ ($\sum_{i=1}^{\ell} \#_t(X_i) = 2$).

We say that ρ is *realizable* if there is an ordering $\pi = (c_1, \dots, c_\ell)$ of the children in C and an assignment $\tau : C \to \mathcal{X}$ from children to types with $\tau(c) \in \mathcal{R}(c)$ for every $c \in C$ such that $\rho = \tau(\pi) = (\tau(c_1), \dots, \tau(c_\ell))$. The following lemma now allows us to focus on finding the set of all types X for which there is a compatible and realizable ρ such that $X = X(\rho)$.

▶ **Lemma 11.** *The set R containing every type $X \in \mathcal{X}$ such that there is a compatible and realizable ρ with $X = X(\rho)$ satisfies the properties* **(R1)** *and* **(R2)**.

We will now show that this can be achieved very efficiently because only a constant number, i.e., at most 8 types (and their ordering) need to be specified in order to infer the type of a sequence ρ. Let $X = (\psi, M, S) \in \mathcal{X}$ be a type. We say that X is *dirty* if $\#_s(X) + \#_t(X) > 0$ and otherwise we say that X is *clean*. We say that X is 0-*good*, 1-*good*, and 2-*good*, if X is clean and additionally $M = \emptyset$, $M = \{\{l,r\}\}$, and $M = \{\{l,r\},\{l',r'\}\}$, respectively. We say that X is *good* if it is x-good for some $x \in \{0,1,2\}$ and otherwise we say that X is *bad*. We denote by $\mathcal{X}_G$ and $\mathcal{X}_B$ the subset of $\mathcal{X}$ consisting only of the good respectively bad types. An illustration of these notions is provided in Figure 6.

▶ **Lemma 12.** *Let $\rho = (X_1, \dots, X_\ell)$ be compatible, then ρ contains at most 8 bad types.*

Next, we will show that any compatible sequence contains at most 8 bad types and that the type $X(\rho)$ is already determined by looking only at the sequence of bad types that occur in ρ. This will then allow us to simulate the enumeration of all possible sequences, by enumerating merely all sequences of at most 8 bad types.

We say that a sequence ρ' is an extension of ρ if ρ is a (not necessarily consecutive) sub-sequence of ρ'. We call a compatible sequence ρ (X,i)-*extendable* for some $X \in \mathcal{X}$ and

integer i, if there is a compatible extension ρ' of ρ such that ρ' is obtained by adding i elements of type X to ρ and $X(\rho) = X(\rho')$. We call ρ *X-extendable* if ρ is (X,i)-extendable for any integer i. We say that ρ' is an *(X,i)-extension* of ρ if ρ' is a compatible sequence obtained after adding i elements of type X to ρ and $X(\rho) = X(\rho')$.

▶ **Lemma 13.** *Let $\rho = (X_1, \ldots, X_\ell)$ with $X_i = (\psi_i, M_i, S_i)$ and $X \in \mathcal{X}_G$. Then, ρ is $(X,1)$-extendable if and only if ρ is X-extendable. Moreover, deciding whether ρ is $(X,1)$-extendable and if so computing an (X,i)-extension ρ' of ρ can be achieved in time $\mathcal{O}(\ell + i)$ for every integer i.*

▶ **Lemma 14.** *Let ρ be a compatible sequence and let ρ' be the sub-sequence of ρ consisting only of the bad types in ρ. Then, ρ' is compatible and $X(\rho) = X(\rho')$.*

At this point, we are ready to describe the algorithm we will use to compute $\mathcal{R}(b)$ (and argue its correctness). The algorithm first enumerates all possible compatible sequences ρ of at most 8 bad types, i.e., $\rho = (Y_1, \ldots, Y_r)$ with $r \leq 8$ and $Y_i \in \mathcal{X}_B$ for every i. Note that there are at most $(|\mathcal{X}_B| + 1)^8$ (and therefore constantly many) such sequences and those can be enumerated in constant time. Given one such sequence $\rho = (Y_1, \ldots, Y_r)$, the algorithm then tests whether the sequence can be realized given the types available for the children in C as follows. It first uses Lemma 13 to test whether ρ allows for adding a 0-good, 1-good or 2-good type in constant time. Let $A_\rho \subseteq \mathcal{X}_G$ be the set of all good types that can be added to ρ and let C_ρ be the subset of C containing all children c such that $A_\rho \cap \mathcal{R}(c) \neq \emptyset$.

Consider the following bipartite graph Q_ρ having one vertex y_i for every i with $1 \leq i \leq r$ representing the type Y_i on one side and one vertex v_c for every $c \in C$ representing the child c on the other side of the bipartition. Moreover, Q_ρ has an edge between y_i and v_c if $Y_i \in \mathcal{R}(c)$. We claim that ρ can be extended to a compatible and realizable sequence if and only if Q_ρ has a matching that saturates $\{y_1, \ldots, y_r\} \cup \{ v_c \mid c \in C \setminus C_\rho \}$. This problem can be solved using a simple reduction to the well-known maximum flow problem. The following lemma now establishes the correctness (i.e., the soundness and completeness) of the algorithm.

▶ **Lemma 15.** *Let $X \in \mathcal{X}$. Then, there is a compatible and realizable sequence ρ with $X = X(\rho)$ if and only if there is a compatible sequence $\rho = (Y_1, \ldots, Y_r)$ of bad types with $r \leq 8$ with $X = X(\rho)$ such that the bipartite graph H_ρ has a matching that saturates $\{y_1, \ldots, y_r\} \cup \{ v_c \mid c \in C \setminus C_\rho \}$.*

5.2 Handling R-nodes and S-nodes

Here, we will show how to compute a set of types satisfying **(R1)** and **(R2)** for every R-node and S-node of $\mathcal{B}$. To achieve this we will again use a dynamic programming algorithm albeit on a sphere-cut decomposition of $\textsc{Sk}(b)$ instead of on the SPQR-tree. The aim of this subsection is therefore to show the following lemma.

▶ **Lemma 16.** *Let b be an R-node or S-node of $\mathcal{B}$ such that $\mathcal{R}(c)$ has already been computed for every child c of b in $\mathcal{B}$. Then, we can compute $\mathcal{R}(b)$ in time $\mathcal{O}((84\sqrt{14})^\omega \omega \ell + \ell^3)$, where ω is the branchwidth of the graph $\textsc{Sk}(b)$ and ℓ is the number of children of b in $\mathcal{B}$.*

In the following, let b be an R-node or S-node of $\mathcal{B}$ with reference edge (s_b, t_b) and let $\langle T_b, \lambda_b, \Pi_b \rangle$ be a sphere-cut decomposition of $\textsc{Sk}(b)$ that is rooted in $r = \lambda_b^{-1}((s_b, t_b))$. For a weak noose $O \subseteq C(T_b)$, let $\mathcal{A}(O)$ be the set of all types of O satisfying the following two natural analogs of **(R1)** and **(R2)**, i.e.:

(RO1) if $X \in \mathcal{A}(O)$, then O has type X, and **(RO2)** if there is a witness (D, D_H, G_H, H) for G that *respects* $\mathcal{B}$ such that $\Gamma_W(b, O) = X$, where $\Gamma_W(b, O)$ is defined analogously to $\Gamma_W(b)$ for the graph $\textsc{Pe}(b, O)$, then $X \in \mathcal{A}(O)$.

Our aim is to compute $\mathcal{A}(O_{a^r})$ for the arc a^r incident to the root r of T_b. This is achieved by computing $\mathcal{A}(O_a)$ for every inner arc a of T_b via a bottom-up dynamic programming algorithm along T_b; after initially calculating $\mathcal{A}(O_a)$ from $\mathcal{R}(c)$ for every leaf-arc a corresponding to the child c of b. Employing our framework introduced in Subsection 4.3, we only have to show how to compute $\mathcal{A}(O_1 \oplus O_2)$ from $\mathcal{A}(O_1)$ and $\mathcal{A}(O_2)$ for any weak nooses O_1 and O_2.

Let O_1 and O_2 be two weak nooses having type $X_1 = (\psi_1, M_1, S_1)$ and type $X_2 = (\psi_2, M_2, S_2)$, respectively. We say that X_1 and X_2 are *compatible* if

(1) $O = O_1 \oplus O_2$ is a weak noose,
(2) the inside region of the noose O contains all subcurves in $(O_1 \cap O_2)$,
(3) $\forall c \in O_1 \cap O_2$, it holds $\psi_1(c) = \psi_2(c)$,
(4) for every $u \in V(O_1 \cap O_2) \setminus V(O_1 \oplus O_2)$, it holds that u is only in one of following sets: S_1, S_2 or $V(M_1) \cap V(M_2)$, and
(5) the multi-graph obtained from the union of M_1 and M_2 is acyclic, or is one cycle and $V(O) \subseteq S_1 \cup S_2 \cup (V(M_1) \cap V(M_2))$,
(6) if X_1 is the full type, then X_2 is the empty type and $V(O_2) \subseteq V(O_1)$, and vice versa.

We denote by $X_1 \circ X_2$ the *combined type* $X = (\psi, M, S)$ of $X_1 = (\psi_1, M_1, S_1)$ and $X_2 = (\psi_2, M_2, S_2)$ for the weak noose $O = O_1 \oplus O_2$ that is defined as follows and also illustrated in Figure 7. For each $c \in O$, if $c \in O_1$ then $\psi(c)$ is equal to $\psi_1(c)$, otherwise $\psi(c)$ is equal to $\psi_2(c)$ and the set S is equal to $(S_1 \cup S_2 \cup (V(M_1) \cap V(M_2))) \cap V(O)$, i.e., any vertex with degree two w.r.t. X must be in $V(O)$ and have degree two already w.r.t. X_1 or X_2, or it must be in both matchings M_1 and M_2. If either X_1 or X_2 is a full type, then by **(6)** we get that $M_1 = M_2 = M = \emptyset$ and $X_1 \circ X_2$ is the full type. If the multi-graph $M_1 \cup M_2$ is one cycle, then by **(5)** we get that $M = \emptyset$ and $X_1 \circ X_2$ is the full type. Otherwise, due to **(5)**, the multi-graph $M_1 \cup M_2$ is acyclic and corresponds to a set of paths. Therefore, the matching M is the set containing the two endpoints for every path in $M_1 \cup M_2$.

▶ **Observation 17.** *Let X_1 and X_2 be two types defined on the weak nooses O_1 and O_2, respectively. Then, we can check whether X_1 and X_2 are compatible and if so compute the type $X_1 \circ X_2$ in time $\mathcal{O}(|O_1| + |O_2|)$.*

To show the correctness of our approach it now remains to show that: (1) if there is a witness W for G that respects $\mathcal{B}$, then for every two weak nooses O_1 and O_2 it holds that $\Gamma_W(b, O_1)$ and $\Gamma_W(b, O_2)$ are compatible types and $\Gamma_W(b, O) = \Gamma_W(b, O_1) \circ \Gamma_W(b, O_2)$ and (2) if O_1 and O_2 have compatible types X_1 and X_2, then $O = O_1 \oplus O_1$ has type $X_1 \circ X_2$.

5.3 Putting Everything Together

Finally, we show how to compute the set of types for every leaf (Q-node) l of $\mathcal{B}$ in time $\mathcal{O}(1)$; informally, since $\textsc{Pe}(b)$ is just an edge (s, t), $\mathcal{R}(l)$ contains all types that do not allow the Hamiltonian cycle to cross from left to right without using either s or t. Together with Lemma 10 and 16, this then concludes the proof of Lemma 9.

6 An Algorithm Using the Feedback Edge Number

In this section, we establish the following theorem:

▶ **Theorem 18.** Book Thickness *is fixed-parameter tractable when parameterized by the feedback edge number of the input graph.*

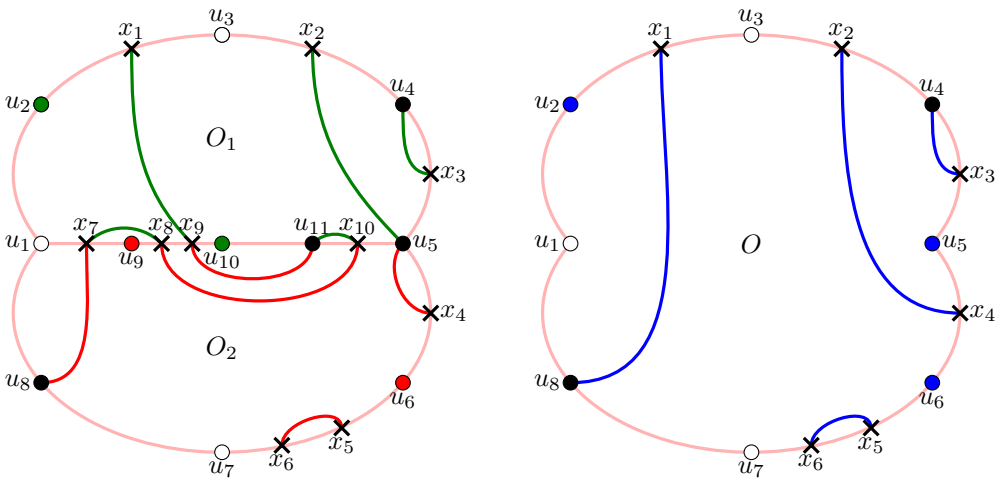


Figure 7 An illustration of combining two compatible types $X_1 = (\psi_1, M_1, S_1)$ and $X_2 = (\psi_2, M_2, S_2)$ for two weak nooses O_1 and O_2 into the combined type $X = (\psi, M, S) = X_1 \circ X_2$ for $O = O_1 \oplus O_2$. Vertices of the graph are represented as circles and vertices subdividing the nooses, i.e., vertices in $V(\psi_1) \cup V(\psi_2)$, are represented as crosses. Black vertices are the vertices that are within a matching, i.e., the vertices in $V(M_1) \cup V(M_2)$, green (red) vertices are the vertices in S_1 (S_2) and all other vertices of the graph are white.

The result is achieved by separately handling two cases: one where the targeted number of pages is greater than 2, or where it is precisely 2. Both cases are handled by a kernelization procedure, and in both cases it is easy to show that pendant vertices can be safely removed. At this point, the target graph consists of a tree plus k edges, whereas the only part that may remain large in this tree are paths of degree-2 vertices. In the former case, we obtain a non-trivial proof that allows us to reduce the maximum length of such a path to length that is bounded by an exponential function of the feedback edge number. In the latter case (which is equivalent to solving SUBHAM), the reduction step is easier and we in fact obtain a linear kernel for the problem:

▶ **Theorem 19.** SUBHAM *parameterized by the feedback edge number k admits a kernel with at most $12k - 8$ vertices and at most $14k - 9$ edges.*

Moreover, by combining Theorem 19 with the subexponential algorithm of Corollary 8, we can slightly strengthen our main result as follows.

▶ **Corollary 20.** SUBHAM *can be solved in time $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$, where k is the feedback edge number of the input graph.*

7 Concluding Remarks

While our main algorithmic result settles the complexity of computing 2-page book embeddings under the exponential time hypothesis, many questions remain when one aims at computing k-page book embeddings for a fixed k greater than 2. To the best of our knowledge, even the existence of a single-exponential algorithm for this problem is open.

In terms of the problem's parameterized complexity, it is natural to ask whether one can obtain a generalization of Theorem 7 for computing k-page book embeddings when $k > 2$. In fact, it is entirely open whether computing, e.g., 4-page book embeddings is even in XP

when parameterized by the treewidth. In this sense, our positive result for the feedback edge number can be seen as a natural step on the way towards finally settling the structural boundaries of tractability for computing page-optimal book embeddings.

References

1 Bernardo M. Ábrego, Oswin Aichholzer, Silvia Fernández-Merchant, Pedro Ramos, and Gelasio Salazar. The 2-page crossing number of K_n. *Discrete & Computational Geometry*, 49(4):747–777, 2013. doi:10.1007/S00454-013-9514-0.

2 Patrizio Angelini, Marco Di Bartolomeo, and Giuseppe Di Battista. Implementing a partitioned 2-page book embedding testing algorithm. *Proc. GD 2012*, 7704:79–89, 2012. doi:10.1007/978-3-642-36763-2_8.

3 Michael J. Bannister and David Eppstein. Crossing minimization for 1-page and 2-page drawings of graphs with bounded treewidth. *Journal of Graph Algorithms and Applications*, 22(4):577–606, 2018. doi:10.7155/jgaa.00479.

4 Giuseppe Di Battista and Roberto Tamassia. Incremental planarity testing. *Proc. FOCS 1989*, pages 436–441, 1989. doi:10.1109/SFCS.1989.63515.

5 Michael A. Bekos, Martin Gronemann, and Chrysanthi N. Raftopoulou. Two-page book embeddings of 4-planar graphs. *Algorithmica*, 75(1):158–185, 2016. doi:10.1007/s00453-015-0016-8.

6 Frank Bernhart and Paul C. Kainen. The book thickness of a graph. *Journal of Combinatorial Theory, Series B*, 27(3):320–331, 1979. doi:10.1016/0095-8956(79)90021-2.

7 Sujoy Bhore, Robert Ganian, Fabrizio Montecchiani, and Martin Nöllenburg. Parameterized algorithms for book embedding problems. *Journal of Graph Algorithms and Applications*, 24(4):603–620, 2020. doi:10.7155/jgaa.00526.

8 Daniel Bienstock and Clyde L. Monma. Optimal enclosing regions in planar graphs. *Networks*, 19(1):79–94, 1989. doi:10.1002/NET.3230190107.

9 Daniel Bienstock and Clyde L. Monma. On the complexity of embedding planar graphs to minimize certain distance measures. *Algorithmica*, 5(1):93–109, 1990. doi:10.1007/BF01840379.

10 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015. doi:10.1016/J.IC.2014.12.008.

11 F. Chung, F. Leighton, and A. Rosenberg. Embedding graphs in books: a layout problem with applications to VLSI design. *SIAM Journal on Algebraic Discrete Methods*, 8(1):33–58, 1987. doi:10.1137/0608002.

12 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

13 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

14 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *Journal of the ACM*, 65(3):12:1–12:46, 2018. doi:10.1145/3148227.

15 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Transactions on Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

16 Hubert de Fraysseix, Patrice Ossona de Mendez, and János Pach. A left-first search algorithm for planar graphs. *Discrete & Computational Geometry*, 13:459–468, 1995. doi:10.1007/BF02574056.

17 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

18 Frederic Dorn, Eelko Penninkx, Hans L. Bodlaender, and Fedor V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. *Algorithmica*, 58(3):790–810, 2010. doi:10.1007/S00453-009-9296-1.

19 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

20 Vida Dujmović and David R. Wood. On linear layouts of graphs. *Discrete Mathematics & Theoretical Computer Science*, 6(2):339–358, 2004. doi:10.46298/dmtcs.317.

21 Vida Dujmovic and David R. Wood. Graph treewidth and geometric thickness parameters. *Discrete & Computational Geometry*, 37(4):641–670, 2007. doi:10.1007/s00454-007-1318-7.

22 Toshiki Endo. The pagenumber of toroidal graphs is at most seven. *Discrete Mathematics*, 175(1):87–96, 1997. doi:10.1016/S0012-365X(96)00144-6.

23 Robert Ganian, Fabrizio Montecchiani, Martin Nöllenburg, and Meirav Zehavi. Parameterized complexity in graph drawing (dagstuhl seminar 21293). *Dagstuhl Reports*, 11(6):82–123, 2021. doi:10.1016/j.artint.2017.12.006.

24 Robert Ganian, Haiko Mueller, Sebastian Ordyniak, Giacomo Paesani, and Mateusz Rychlicki. A tight subexponential-time algorithm for two-page book embedding, 2024. arXiv:2404.14087.

25 Joseph L. Ganley and Lenwood S. Heath. The pagenumber of k-trees is $O(k)$. *Discrete Applied Mathematics*, 109(3):215–221, 2001. doi:10.1016/S0166-218X(00)00178-5.

26 M. R. Garey, D. S. Johnson, and R. Endre Tarjan. The planar hamiltonian circuit problem is np-complete. *SIAM Journal on Computing*, 5(4):704–714, 1976. doi:10.1137/0205049.

27 Emilio Di Giacomo and Giuseppe Liotta. The hamiltonian augmentation problem and its applications to graph drawing. *Proc. WALCOM 2010, LNCS*, 5942:35–46, 2010. doi:10.1007/978-3-642-11440-3_4.

28 Qian-Ping Gu and Hisao Tamaki. Improved bounds on the planar branchwidth with respect to the largest grid minor size. *Algorithmica*, 64(3):416–453, 2012. doi:10.1007/S00453-012-9627-5.

29 Carsten Gutwenger, Petra Mutzel, and René Weiskircher. Inserting an edge into a planar graph. *Algorithmica*, 41(4):289–308, 2005. doi:10.1007/S00453-004-1128-8.

30 András Gyárfás and Jenö Lehel. Covering and coloring problems for relatives of intervals. *Discrete Mathematics*, 55(2):167–180, 1985. doi:10.1016/0012-365X(85)90045-7.

31 Christian Haslinger and Peter F. Stadler. RNA structures with pseudo-knots: Graph-theoretical, combinatorial, and statistical properties. *Bulletin of Mathematical Biology*, 61(3):437–467, 1999. doi:10.1006/bulm.1998.0085.

32 Lenwood S. Heath. Embedding outerplanar graphs in small books. *SIAM Journal on Algebraic Discrete Methods*, 8(2):198–218, 1987. doi:10.1137/0608018.

33 Seok-Hee Hong and Hiroshi Nagamochi. Two-page book embedding and clustered graph planarity. Technical report, Citeseer, 2009.

34 Seok-Hee Hong and Hiroshi Nagamochi. Simpler algorithms for testing two-page book embedding of partitioned graphs. *Theoretical Computer Science*, 725:79–98, 2018. doi:10.1016/J.TCS.2015.12.039.

35 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/JCSS.2001.1774.

36 Hugo Jacob and Marcin Pilipczuk. Bounding twin-width for bounded-treewidth graphs, planar graphs, and bipartite graphs. *Proc. WG 2022*, 13453:287–299, 2022. doi:10.1007/978-3-031-15914-5_21.

37 Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. *Proc. FOCS 2021*, pages 184–192, 2021. doi:10.1109/FOCS52979.2021.00026.

38 Seth M. Malitz. Genus g graphs have pagenumber $O(\sqrt{g})$. *Journal of Algorithms*, 17(1):85–109, 1994. doi:10.1006/jagm.1994.1028.

39 Dániel Marx. Four shorts stories on surprising algorithmic uses of treewidth. *Treewidth, Kernels, and Algorithms*, 12160:129–144, 2020. doi:10.1007/978-3-030-42071-0_10.

40 Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. A subexponential parameterized algorithm for directed subset traveling salesman problem on planar graphs. *SIAM Journal on Computing*, 51(2):254–289, 2022. doi:10.1137/19M1304088.

41 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity – Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

42 Malgorzata Nowicka, Vinay K. Gautam, and Pekka Orponen. Automated rendering of multi-stranded dna complexes with pseudoknots, 2023. arXiv:2308.06392.

43 Neil Robertson and Paul D. Seymour. Graph minors. x. obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991. doi:10.1016/0095-8956(91)90061-N.

44 Neil Robertson, Paul D. Seymour, and Robin Thomas. Quickly excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 62(2):323–348, 1994. doi:10.1006/JCTB.1994.1073.

45 Johannes Uhlmann and Mathias Weller. Two-layer planarization parameterized by feedback edge set. *Theoretical Computer Science*, 494:99–111, 2013. doi:10.1016/J.TCS.2013.01.029.

46 Avi Wigderson. The complexity of the hamiltonian circuit problem for maximal planar graphs. *Technical Report*, 1982.

47 Mihalis Yannakakis. Embedding planar graphs in four pages. *Journal of Computer and System Sciences*, 38(1):36–67, 1989. doi:10.1016/0022-0000(89)90032-9.

Quantum Algorithms for Graph Coloring and Other Partitioning, Covering, and Packing Problems

Serge Gaspers
UNSW Sydney, Australia

Jerry Zirui Li
James Ruse Agricultural High School, Carlingford, Australia
UNSW Sydney, Australia

Abstract

Let U be a universe on n elements, let k be a positive integer, and let $\mathcal{F}$ be a family of (implicitly defined) subsets of U. We consider the problems of partitioning U into k sets from $\mathcal{F}$, covering U with k sets from $\mathcal{F}$, and packing k non-intersecting sets from $\mathcal{F}$ into U. Classically, these problems can be solved via inclusion–exclusion in $2^n n^{O(1)}$ time [8]. Quantumly, there are faster algorithms for graph coloring with running time $O(1.9140^n)$ [26] and for Set Cover with a small number of sets with running time $O\left(1.7274^n |\mathcal{F}|^{O(1)}\right)$ [1]. In this paper, we give a quantum speedup for Set Partition, Set Cover, and Set Packing whenever there is a classical enumeration algorithm that lends itself to a quadratic quantum speedup, which, for any subinstance on a set $X \subseteq U$, enumerates at least one member of a k-partition, k-cover, or k-packing (if one exists) restricted to (or projected onto, in the case of k-cover) the set X in $c^{|X|}n^{O(1)}$ time with $c < 2$. Our bounded-error quantum algorithm runs in time $(2+c)^{n/2}n^{O(1)}$ for Set Partition, Set Cover, and Set Packing. It is obtained by combining three algorithms that have the best running time for some values of c. When $c \leq 1.147899$, our algorithm is slightly faster than $(2+c)^{n/2}n^{O(1)}$; when c approaches 1, it matches the $O\left(1.7274^n |\mathcal{F}|^{O(1)}\right)$ running time of [1] for Set Cover when $|\mathcal{F}|$ is subexponential in n.

For covering, packing, and partitioning into maximal independent sets, maximal cliques, maximal bicliques, maximal cluster graphs, maximal triangle-free graphs, maximal cographs, maximal claw-free graphs, maximal trivially-perfect graphs, maximal threshold graphs, maximal split graphs, maximal line graphs, and maximal induced forests, we obtain bounded-error quantum algorithms with running times ranging from $O(1.8554^n)$ to $O(1.9629^n)$. Packing and covering by maximal induced matchings can be done quantumly in $O(1.8934^n)$ time.

For Graph Coloring (covering with k maximal independent sets), we further improve the running time to $O(1.7956^n)$ by leveraging faster algorithms for coloring with a small number of colors to better balance our divide-and-conquer steps. For Domatic Number (packing k minimal dominating sets), we obtain a $O((2-\varepsilon)^n)$ running time for some $\varepsilon > 0$.

Several of our results should be of interest to proponents of classical computing:

- We present an inclusion-exclusion algorithm with running time $O^*\left(\sum_{i=0}^{\lfloor \alpha n \rfloor} \binom{n}{i}\right)$, which determines, for each $X \subseteq U$ of size at most αn, $0 \leq \alpha \leq 1$, whether $(X, \mathcal{F})$ has a k-cover, k-partition, or k-packing. This running time is best-possible, up to polynomial factors.
- We prove that for any linear-sized vertex subset $X \subseteq V$ of a graph $G = (V, E)$, the number of minimal dominating sets of G that are subsets of X is $O((2-\varepsilon)^{|X|})$ for some $\varepsilon > 0$.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Quantum complexity theory

Keywords and phrases Graph algorithms, quantum algorithms, graph coloring, domatic number, set cover, set partition, set packing

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.69

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2311.08042 [20]

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 69; pp. 69:1–69:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Graph Coloring is an example of a problem requiring to partition an n-element set U into k sets from a family $\mathcal{F}$. In this case U is the vertex set of a graph G and $\mathcal{F}$ is implicitly defined as the independent sets of G. We can also view Graph Coloring as a covering problem where the vertex set needs to be covered with k maximal independent sets.

In 2006, Björklund and Husfeldt [4] and Koivisto [25] independently solved Graph Coloring in $O^*(2^n)$ time via a new inclusion–exclusion approach, along with other partitioning and covering problems. The approach has been used for packing problems as well, and has been generalized to solve more generic subset convolution problems [5, 7, 8].

In this work, we give faster quantum algorithms for a range of partitioning, covering, and packing problems, including Graph Coloring and Domatic Number. To do this, we use the framework of Ambainis et al. [1] where a preprocessing step computes solutions to small subinstances and stores them in QRAM. These solutions are then accessed by a divide-and-conquer algorithm which enjoys a quadratic speedup in quantum models of computation via techniques such as Grover's search [22]. For the preprocessing step (Section 3), we adapt the afore-mentioned inclusion-exclusion approach [8] to compute the solutions to all subinstances induced by a small subset of U up to roughly $n/4$ elements. For the divide-and-conquer step (Section 4) one would ideally like to divide U into two halves; unfortunately, optimal partitions[1] may not allow for such a balanced split. However, we can restrict the divide-and-conquer step to divide U into two parts where in the larger part (equivalently, in both parts) the removal of one set of an optimal partition results in at most $n/2$ elements. Finding one set of the optimal partition is done via an algorithm that enumerates all relevant candidate sets; for Graph Coloring, it enumerates the maximal independent sets in the graph induced by the subset $X \subseteq U$ under consideration. Importantly, we need that this enumeration can be done in $O^*\left(c^{|X|}\right)$ time, for some $c < 2$ by a classical algorithm that has a quadratic quantum speedup, so that after two levels of divide-and-conquer, the overall quadratic quantum speedup outperforms the classical $O^*(2^n)$ running time.

Our algorithm differs from the previously fastest quantum algorithm for Graph Coloring by Shimizu and Mori [26] in both the preprocessing step and the divide-and-conquer step. Our preprocessing step is deterministic and its running time is optimal, matching the size of the output up to polynomial factors; the preprocessing step of [26] is a bounded-error quantum algorithm whose running time is a multiplicative factor of $3^{|X|/6}$ slower than ours for each small (up to size roughly $n/4$) subset $X \subseteq U$. For the divide-and-conquer step, our divide-then-enumerate strategy is described above; [26] employ an enumerate-then-divide strategy, where the enumeration is done on the set to be divided and the remainder is then divided into two sets of size at most half the original set. It turns out that blending the the divide-then-enumerate and the enumerate-then-divide strategy gives faster algorithms when $c \leq 1.147899$ (Section 5). For $c \leq 1.0872$ case, we also use a third level of divide-and-conquer, and when c approaches 1, our $O(1.7274^n)$ running time matches the running time for Set Cover with a subexponential number of sets of Ambainis et al. [1].

This gives improved algorithms for a range of partitioning, covering, and packing problems (Section 6). We further improve the running time for Graph Coloring (Section 7) to $O(1.7956^n)$ by leveraging faster algorithms for a small number of colors [26]. Our algorithm considers large subsets of vertices ($\geq 0.48n$) and checks whether they are 5-colorable, 6-colorable with

[1] For conciseness, we will mainly discuss partitions in the introduction. The treatment of covers and packings is similar.

no large 5-colorable subset, and in some cases 7-colorable via a new 7-Coloring algorithm that relies on the preprocessing step. The advantage of excluding such cases from further consideration is that we can make the divide-and-conquer steps more balanced.

For Domatic Number, at first glance it seems that our approach cannot be used. The issue is that when considering a vertex subset X, even though there is an algorithm that enumerates all minimal dominating sets of $G[X]$ in $O(1.7159^n)$ time [16], this is insufficient for our purposes: we need to enumerate minimal vertex subsets of X that dominate all of G, not just $G[X]$, in $O^*\left(c^{|X|}\right)$ time for some $c < 2$. In Section 8, we show that such an enumeration algorithm (with a quadratic quantum speedup) indeed exists provided that $|X| = \Omega(n)$. This then also gives a bounded-error quantum algorithm for Domatic Number running in $O((2-\varepsilon)^n)$ time.

2 Preliminaries

For a proposition P, the *Iverson bracket* $[P]$ is a function that returns 1 if P is true and 0 otherwise.

Asymptotic notation

The O^*-notation is similar to the usual O-notation but allows to hide polynomial factors in the input size. The $\tilde{O}$-notation hides polylogarithmic factors. We make heavy use of Stirling's approximation for factorials, which implies that $\binom{n}{k} = O^*\left(\left(\frac{n}{k}\right)^k \cdot \left(\frac{n}{n-k}\right)^{n-k}\right)$, and of the binomial theorem, $\sum_{k=0}^{n}\binom{n}{k}x^k y^{n-k} = (x+y)^n$.

Set Systems

An *implicit set system* [15] is a function Φ that takes as input a string $I \in \{0,1\}^*$ and outputs a set system $(U_I, \mathcal{F}_I)$, where U_I is a universe and $\mathcal{F}_I$ is a collection of subsets of U_I. The string I is referred to as an *instance* and we denote by $|U_I| = n$ the size of the universe and by $|I| = N$ the size of the instance. We assume that $N \geq n$. The implicit set system Φ is said to be *polynomial time computable* if (a) there exists a polynomial time algorithm that given I, produces U_I, and (b) there exists a polynomial time algorithm that given I, U_I and a subset S of U_I, determines whether $S \in \mathcal{F}_I$. Throughout this paper, we consider only polynomial time computable implicit set systems. We define a *subset polynomial time computable* implicit set system Φ to be a polynomial time computable set system, where (c) there exists a polynomial time algorithm that given I, U_I and a subset S of U_I, determines whether $S \subseteq S'$ for some $S' \in \mathcal{F}_I$. This is equivalent to determining whether $S \in \mathcal{F}\downarrow$, where the downward closure $\mathcal{F}\downarrow$ of $\mathcal{F}$ contains all sets in $\mathcal{F}$ and their subsets.

For any subset of elements $X \subseteq U$, an ordered tuple $(S_1, \ldots, S_k)$ of k sets from $\mathcal{F}$ is a *k-cover* for X if the union of these sets is X; it is a *k-packing* for X if the S_i's are contained in X and are pairwise non-intersecting; it is a *k-partition* for X if it is both a k-cover and a k-packing for X.

For a subset polynomial time computable implicit set system Φ, the input of the Φ-Set Cover problem is an instance I and an integer k, and the question is whether the set system $\Phi(I) = (U_I, \mathcal{F}_I)$ has a k-cover. This is equivalent to asking whether $(U_I, \mathcal{F}_I\downarrow)$ has a k-cover and therefore, we assume that $\mathcal{F}_I = \mathcal{F}_I\downarrow$ whenever discussing k-covers. For a polynomial time computable implicit set system Φ, the input of the Φ-Set Partition and Φ-Set Packing problem is an instance I and an integer k, and the question is whether the set system $\Phi(I) = (U_I, \mathcal{F}_I)$ has a k-partition or k-packing, respectively. We generally omit Φ and the subscript I when they are clear from context.

Graphs

In a graph $G = (V, E)$, the *open neighborhood* of a vertex v, denoted $N_G(v)$ is the set containing all vertices adjacent to v in G. The *closed neighborhood* of v in G also contains v itself, and is denoted $N_G[v] = \{v\} \cup N_G(v)$. Again, we may omit the subscript G. For a vertex subset $X \subseteq V$, the graph $G - X$ is obtained from G by removing the vertices in X and all their incident edges; the graph $G[X]$ induced on X is the graph $G - (V \setminus X)$.

Quantum Algorithms

It is known that most classical branching algorithms have a quadratic speedup on quantum machines. As [26], we also rely on the following results.

▶ **Theorem 1** ([22, 9]). *Let $A : \{1, 2, \ldots, N\} \to \{0, 1\}$ be a bounded-error quantum algorithm with running time T. Then, there is a bounded-error quantum algorithm computing $\bigvee_{x\in\{1,\ldots,N\}} A(x)$ with running time $\tilde{O}(\sqrt{N}T)$.*

▶ **Theorem 2** ([13]). *Let $A : \{1, 2, \ldots, N\} \to \{0, 1\}$ be a bounded-error quantum algorithm with running time T. Then, there is a bounded-error quantum algorithm computing $\min_{x\in\{1,\ldots,N\}} A(x)$ with running time $\tilde{O}(\sqrt{N}T)$.*

In our context, A is an algorithm exploring paths in superposition from the root to the leaves of the search tree of a classical branching algorithm. The amplitudes of this exploration depend on estimates of the sizes of the subtrees, either by relying on an analysis of the classical branching algorithm [18, 26] or by on-the-fly estimations [2]. We speak of a *simple* branching algorithm when the exploration of one root-to-leaf path is independent of the other paths; this excludes, for example, algorithms relying on clause learning, re-use of computation done in earlier branches, and branch-and-bound. For a simple branching algorithm with running time $O^*(c^n)$, one obtains a bounded-error quantum algorithm with running time $O^*(c^{n/2})$ in this way; we simply say that we apply Grover's search to the branching algorithm.

3 Preprocessing small subsets

For $\alpha \in [0, 1]$, a subset X of U is *α-small* if $|X| \leq \alpha n$. Denote by $s(n, \alpha) = \sum_{i=0}^{\lfloor \alpha n \rfloor} \binom{n}{i}$ the number of α-small subsets of U. In this section, we consider the problem of counting the number of k-covers, k-packings, and k-partitions for each α-small subset of U. When considering k-covers, we assume that $\mathcal{F}$ has been replaced by $\mathcal{F}\downarrow$. This is because when we would like to cover a subset of elements of X, we may use a set from $\mathcal{F}$ that also contains elements outside of X. Since Φ is subset polynomial time computable in the Φ-Set Cover problem, we may as well replace $\mathcal{F}$ by $\mathcal{F}\downarrow$; this makes the discussion of covering, partitioning, and packing problems more uniform. Our algorithms run in $O^*(s(n, \alpha))$ time, which is best possible, since the output is a list of $s(n, \alpha)$ integers.

This section heavily relies on previous $O^*(2^n)$ inclusion-exclusion approaches [7, 8] to compute the number of k-covers, k-packings, and k-partitions for U and these results are well-known when $\alpha = 1$. The work by [6] is also related, but their running times depends on the number of supersets of $\mathcal{F}$.

We start by defining the α-small zeta transform, which is central to this section.

▶ **Definition 3.** *Let f be a function from subsets of the universe U to an algebraic ring R. The* α-small zeta transform *of f, denoted $f\zeta_\alpha$ is*

$$f\zeta_\alpha(X) = \sum_{Y\subseteq X} f(Y)$$

for any α-small $X \subseteq U$.

The 1-small zeta transform is also called the *zeta transform* and the α-small zeta transform is precisely the restriction of the zeta transform to α-small sets. Throughout this paper we assume that arithmetic operations in the ring R take $O^*(1)$ time and each ring element is represented using $O^*(1)$ space.

▶ **Definition 4.** *Let f be a function from subsets of the universe U to an algebraic ring R. The* α-small Möbius transform *of f, denoted $f\mu_\alpha$ is*

$$f\mu_\alpha(X) = \sum_{Y \subseteq X} (-1)^{|X \setminus Y|} f(Y)$$

for any α-small $X \subseteq U$.

It is well-known (see, e.g., [17]) that $f\zeta_\alpha\mu_\alpha = f\mu_\alpha\zeta_\alpha = f$ when $\alpha = 1$, and the same is true when $\alpha \neq 1$.

▶ **Lemma 5.** *The α-small zeta transform $f\zeta_\alpha$ and the α-small Möbius transform $f\mu_\alpha$ can be computed in $O^*(s(n, \alpha))$ time.*

Proof. We start with $f\zeta_\alpha$ and proceed as in Yates's method [27]. Consider an arbitrary ordering of the elements of $U = \{v_1, \ldots, v_n\}$. The algorithm considers each α-small $X \subseteq U$ by increasing order of cardinality.

Set $g_0(X) = f(X)$. Then, iterate over the elements of U in the ordering fixed above. When processing element v_i, set

$$g_i(X) = g_{i-1}(X) + [v_i \in X] \cdot g_{i-1}(X \setminus \{v_i\}).$$

Finally, set $f\zeta_\alpha(X) = g_n(X)$.

Correctness can be shown by induction on i by observing that

$$g_i(X) = \sum_{\{v_{i+1}, \ldots, v_n\} \cap X \subseteq Y \subseteq X} f(Y).$$

For each set X the computation takes $O^*(1)$ time, and the number of sets X to be considered is $s(n, \alpha)$.

To compute $f\mu_\alpha$, we use the fact that $\mu_\alpha = \sigma_\alpha\zeta_\alpha\sigma_\alpha$, where the α-small odd-negation transform is

$$f\sigma_\alpha(X) = (-1)^{|X|} f(X),$$

defined for any α-small $X \subseteq U$. Indeed,

$$\begin{aligned} f\sigma_\alpha\zeta_\alpha\sigma_\alpha(X) &= (-1)^{|X|} \cdot \sum_{Y \subseteq X} (-1)^{|Y|} \cdot f(Y) \\ &= \sum_{Y \subseteq X} (-1)^{|X|+|Y|} f(Y) \\ &= \sum_{Y \subseteq X} (-1)^{|X \setminus Y|} f(Y) \end{aligned}$$

since $|X|+|Y|$ and $|X|-|Y| = |X \setminus Y|$ have the same parity. The result now follows, because, for a function g and a set X, $g\sigma_\alpha(X)$ can be computed in $O^*(1)$ time. ◀

We refer to these algorithms as the *fast α-small zeta transform* and the *fast α-small Möbius transform*.

By inclusion-exclusion, the number of k-covers for a subset $X \subseteq U$ is [7]

$$c_k(\mathcal{F}, X) = \sum_{Y \subseteq X} (-1)^{|X \setminus Y|} a(Y)^k, \tag{1}$$

where $a(Y)$ is the number of subsets $Z \subseteq Y$ that belong to $\mathcal{F} = \mathcal{F}\downarrow$.

For the number of k-partitions, we use an indeterminate z in the ring R that allows us to keep track of the sum of the cardinalities of the sets in the cover. The number of k-partitions for a subset $X \subseteq U$ is given [7] by the coefficient of the monomial $z^{|X|}$ in the polynomial

$$d_k(\mathcal{F}, X) = \sum_{Y \subseteq X} (-1)^{|X \setminus Y|} \left(\sum_{j=0}^{|Y|} a_j(Y) z^j \right)^k, \tag{2}$$

where $a_j(Y)$ is the number of size-j subsets $Z \subseteq Y$ that belong to $\mathcal{F}$.

For the number of k-packings of X, we compute the number of $(k+1)$-partitions where the first k members of the $(k+1)$-tuple belong to $\mathcal{F}$ and the last member is an arbitrary subset of X. Noting that $(1+z)^{|Y|} = \sum_{i=0}^{|Y|} \binom{|Y|}{i} z^i$, the number of k-packings for $X \subseteq U$ is [7] the coefficient of $z^{|X|}$ in

$$p_k(\mathcal{F}, X) = \sum_{Y \subseteq X} (-1)^{|X \setminus Y|} (1+z)^{|Y|} \left(\sum_{j=0}^{|Y|} a_j(Y) z^j \right)^k. \tag{3}$$

The first algorithmic step is to compute the values for $a(Y)$ in (1) and the polynomials $\sum_{j=0}^{|Y|} a_j(Y) z^j$ in (2) and (3). Observe that $|Y| \leq |X| \leq \alpha n$. To compute the values $a(Y)$ for all α-small $Y \subseteq U$, observe that $a(Y) = \sum_{Z \subseteq Y} [Z \in \mathcal{F}]$ is the α-small zeta transform of the indicator function of $\mathcal{F}$. Since the implicit set system is polynomial time computable, the indicator function can be evaluated in polynomial time. Therefore, the fast α-small zeta transform allows us to compute all relevant values of $a(\cdot)$ in $O^*(s(n,\alpha))$ time. Similarly, the polynomial $\sum_{j=0}^{|Y|} a_j(Y) z^j$ equals $h\zeta_\alpha(Y)$ where $h(Z) = [Z \in \mathcal{F}] \cdot z^{|Z|}$ and can be computed in the same time bound by the fast α-small zeta transform.

The second algorithmic step is to to use the fast α-small Möbius transform and apply it to the functions that associate with each α-small $Y \subseteq U$ the values $a(Y)^k$; $\left(\sum_{j=0}^{|Y|} a_j(Y) z^j\right)^k$; and $(1+z)^{|Y|} \left(\sum_{j=0}^{|Y|} a_j(Y) z^j\right)^k$, respectively.

We conclude that $c_k(\mathcal{F}, X)$, $d_k(\mathcal{F}, X)$, and $p_k(\mathcal{F}, X)$ for each α-small $X \subseteq U$ can be computed in $O^*(s(n,\alpha))$ time.

▶ **Theorem 6.** *Given a polynomial time computable implicit set family $\Phi(I) = (U, \mathcal{F})$ with $|U| = n$, there is a $O^*(s(n,\alpha))$ time algorithm which determines, for all $k \leq \alpha n$ and all α-small $X \subseteq U$, whether $(X, \mathcal{F})$ has a k-cover (if we assume that $\mathcal{F}$ is closed under subsets), k-partition, or k-packing.*

4 Divide-and-conquer algorithm

Let $\Phi(I) = (U, \mathcal{F})$ be a polynomial time computable implicit set family for which we would like to determine whether there is a k-cover (assuming $\mathcal{F} = \mathcal{F}\downarrow$), k-partition, or k-packing.

We say that a simple branching algorithm *enumerates* a family $\mathcal{F}'$ of subsets of U if, at each leaf of its search tree it finds at most one member of $\mathcal{F}'$, and collectively the leaves find all members of $\mathcal{F}'$ (duplicates are allowed).

Our algorithm uses a divide-and-conquer strategy where the universe is twice partitioned into two.

▶ **Definition 7.** *For a family of subsets $\mathcal{F}$ of a universe U, and a subset $X \subseteq U$, the* restriction *of $\mathcal{F}$ to X is $r(\mathcal{F}, X) = \{S \subseteq X : S \in \mathcal{F}\}$.*

Observe that $(U, \mathcal{F})$ has a k-partition (resp., a k-packing or a k-cover) for $k \geq 2$ iff there is a set $L \subseteq U$ and a positive integer $k_l < k$ such that $(L, r(\mathcal{F}, L))$ has a k_l-partition (resp., a k_l-packing or a k_l-cover) and $(R, r(\mathcal{F}, R))$ has a k_r-partition (resp., a k_r-packing or a k_r-cover), where $R = U \setminus L$ and $k_r = k - k_l$. We say that a k-partition, k-packing, or k-cover $(S_1, \ldots, S_k)$ does not *straddle* X if for each $i \in \{1, \ldots, k\}$, either $S_i \subseteq X$ or $S_i \cap X = \emptyset$.

In this section we prove the following theorem.

▶ **Theorem 8.** *Suppose there is a simple (classical) branching algorithm A, which, given an instance I with $\Phi(I) = (U, \mathcal{F})$ and a subset $X \subseteq U$, enumerates a family $e(X, \mathcal{F}_X)$ of subsets of X from $\mathcal{F}_X$ such that*

- *if $(U, \mathcal{F})$ has a k-cover (resp., a k-partition or a k-packing) that does not straddle X, then $(U, \mathcal{F})$ has a k-cover (resp., a k-partition or a k-packing) $(S_1, \ldots, S_k)$ with $S_1 \in e(X, \mathcal{F}_X)$, and*
- *the algorithm runs in $O^*(c^{|X|})$ time for some $c \leq 2$.*

Then, there is a bounded-error quantum algorithm, which determines whether $(U, \mathcal{F})$ has a k-cover (resp., a k-partition or a k-packing) in $O^*\left(\binom{n}{n/4} + (2+c)^{n/2}\right)$ time, where $n = |U|$.

From now on, we focus on Set Cover; the discussion of Set Partition and Set Packing is analogous. The first step is to use the algorithm from Theorem 6 with $\alpha = \frac{1}{4}$ and store the result in QRAM.

Ideally, we would want to divide the universe U into equal sized sets L and R and compute a k-cover where each set of the cover is responsible for covering elements of either L or R. However, we cannot guarantee that such a k-cover exists. Instead, we can focus on partitions of U into L and R with $|L| \geq n/2$ where the removal of one member of the k-cover decreases the size of L to at most $n/2$.

▶ **Lemma 9.** *For any $x \in \{0, \ldots, n\}$, $(U, \mathcal{F})$ has a k-cover, $k \geq 2$, iff there is a partition of U into (L, R) with $|L| \geq x$ and integers $k_L, k_R \geq 1$ with $k = k_L + k_R$ such that $(L, r(\mathcal{F}, L))$ has a k_L-cover $(S_1, \ldots, S_{k_L})$ and $(R, r(\mathcal{F}, R))$ has a k_r-cover such that $|S_i| \geq |L| - x$ for every $i \in \{1, \ldots, k_L\}$.*

Proof. For the backward direction, assume that $(L, r(\mathcal{F}, L))$ has a k_l-cover and $(R, r(\mathcal{F}, R))$ has a k_r-cover. Then $(L \cup R, r(\mathcal{F}, L) \cup r(\mathcal{F}, R))$ has a k-cover where $k = k_l + k_r$. This k-cover is of the form $(S_1, S_2, \cdots, S_k)$, where $S_i \subseteq r(\mathcal{F}, L) \cup r(\mathcal{F}, R)$. To turn it into a k-cover for $(U, \mathcal{F})$, we replace each S_i by a set S'_i with $S_i \subseteq S'_i \in \mathcal{F}$, and we note that such a set S'_i exists in $\mathcal{F}$, since S_i is the restriction of some set in $\mathcal{F}$ to either L or R. Hence, $(U, \mathcal{F})$ has a k-cover.

For the forward direction, assume that $(U, \mathcal{F})$ has a k-cover $(S_1, S_2, \cdots, S_k)$ and assume, w.l.o.g., that $|S_1| \geq |S_2| \geq \cdots \geq |S_k|$. Let $L = \bigcup_{i=1}^{k_L} S_i$ and $R = U \setminus L$ where k_L is the smallest value such that $|L| > x$. Obviously, $(S_1, S_2, \cdots, S_{k_L})$ is a k_L-cover of $(L, r(\mathcal{F}, L))$ and $(S_{k+1} \cap R, S_{k+2} \cap R, \cdots, S_k \cap R)$ is a k_R-cover of $(R, r(\mathcal{F}, R))$ where $k_R = k - k_L$.

Focusing on S_{k_L}, the smallest set in the cover of L, we have that

$$\begin{aligned}
&\bigcup_{i=1}^{k_L} S_i \setminus S_{k_L} \subseteq \bigcup_{i=1}^{k_L-1} S_i \\
\implies &|\bigcup_{i=1}^{k_L} S_i| - |S_{k_L}| \leq |\bigcup_{i=1}^{k_L-1} S_i| \\
\implies &\quad |L| - |S_{k_L}| \leq x \\
\implies &\quad |S_i| \geq |S_{k_L}| \geq |L| - x \ \text{ for all } i \in \{1, \ldots, k_L\}.
\end{aligned}$$

Therefore, $(U, \mathcal{F})$ has a k-cover if and only if the conditions are satisfied. ◀

In particular, this means that removing any member S_i of the k_L-cover gives a $(k_L - 1)$-cover of $L \setminus S_i$ and $|L \setminus S_i| \leq x$.

We use Grover's search to divide the elements into two sets (L, R) where $|L| \geq n/2$ and k into $k_L + k_R$. Then, we solve each of these two instances independently, again using Grover's search on $(L, r(\mathcal{F}, L))$ (resp., on $(R, r(\mathcal{F}, R))$), dividing it into (LL, LR) where $|LL| \geq n/4$ and $k_{LL} + k_{LR} = k_L$ (similar for R). For $(LL, r(\mathcal{F}, LL))$ (and, similarly, on the corresponding instances for LR, RL, RR), we use Grover's search on the branching algorithm A to enumerate candidate sets $S_1 \in e(LL, r(\mathcal{F}, LL))$; if $|LL| - |S_1| > n/4$, then this branch is unsuccessful; otherwise, look up whether $LL \setminus S_1$ has a $(k_{LL} - 1)$-cover in QRAM.

The running time of this algorithm is

$$O^*\left(\binom{n}{n/4}\right)$$

for running the algorithm from Theorem 6; the steps using Grover's search take

$$O^*\left(\sqrt{\sum_{l=\lceil n/2 \rceil}^{n} \binom{n}{l} \sum_{l'=\lceil n/4 \rceil}^{l} \binom{l}{l'} c^{l'}}\right) = O^*\left((2+c)^{n/2}\right)$$

time to achieve constant success probability. This proves Theorem 8.

Discussion

When $c \geq \frac{16}{3^{3/2}} - 2 \approx 1.079201$, then the running time is $O^*\left((2+c)^{n/2}\right)$. When $c \leq \frac{16}{3^{3/2}} - 2$, then the running time is dominated by the term $\binom{n}{n/4} \approx 1.7548^n$. For $c \leq 1.147899$, the next section gives faster algorithms.

5 Divide-and-conquer algorithms for small c

In this section, we again assume that there is an enumeration algorithm A, as in Theorem 8, with running time $O^*\left(c^{|X|}\right)$. We present two divide-and-conquer algorithms which are faster for small values of c.

Throughout this section, we focus on Set Cover; the discussion of Set Partition and Set Packing is analogous.

▶ **Theorem 10.** *There is a bounded-error quantum algorithm, which determines whether $(U, \mathcal{F})$ has a k-cover (resp., a k-partition or a k-packing) in $O^*\left(\binom{n}{n/4} + (1+c)^{\frac{3}{4} \cdot n}\right)$ time, where $n = |U|$.*

The first step is to use the algorithm from Theorem 6 with $\alpha = \frac{1}{4}$ and store the result in QRAM. Our algorithm is similar to the algorithm in Theorem 8, but we use the branching algorithm A to remove a subset from L before dividing it into (LL, LR). Again, Lemma 9 is central to our algorithm.

We use Grover's Search to divide the elements into two sets (L, R) where $|L| \geq n/2$ and k into $k_L + k_R$. Then, we solve each of these two instances independently, again using Grover's search on $(R, r(\mathcal{F}, R))$, dividing it into (RL, RR) where $|RL| \geq n/4$ and $k_{RL} + k_{RR} = k_R$. For the instance $(L, r(\mathcal{F}, L))$, we use Grover's search on the branching algorithm A to enumerate candidate sets $S_1 \in e(L, r(\mathcal{F}, L))$; if $|L| - |S_1| > n/2$ then this branch is unsuccessful; otherwise, we use Grover's search on the subinstance $(L \setminus S_1, r(\mathcal{F}, L \setminus S_1))$, dividing it into (LL, LR) where $|LL| \geq n/4$ and $k_{LL} + k_{LR} = k_L - 1$.

We process $(LL, r(\mathcal{F}, LL))$ (and, similarly, the corresponding instances for LR, RL, RR) as we did in Theorem 8 – we use Grover's search on the branching algorithm A to enumerate candidate sets $S_1 \in e(LL, r(\mathcal{F}, LL))$; if $|LL| - |S_1| > n/4$, then this branch is unsuccessful; otherwise, look up whether $LL \setminus S_1$ has a $(k_{LL} - 1)$-cover in QRAM.

The running time of this algorithm is

$$O^*\left(\binom{n}{n/4}\right)$$

for the running the algorithm from Theorem 6; the steps using Grover's search take

$$O^*\left(\sqrt{\sum_{l=\lceil n/2\rceil}^{n}\binom{n}{l}c^l\sum_{l'=\lceil n/4\rceil}^{\lfloor n/2\rfloor}\binom{n/2}{l'}c^{l'}}\right) = O^*\left((1+c)^{\frac{3}{4}\cdot n}\right)$$

time to achieve constant success probability. This proves Theorem 10.

Discussion

When $c \geq \frac{2^{8/3}}{3} - 1 \approx 1.11653$, then the running time is $O^*\left((1+c)^{\frac{3}{4}\cdot n}\right)$. When $c \leq \frac{2^{8/3}}{3} - 1$, then the running time is dominated by the term $\binom{n}{n/4} \approx 1.7548^n$. When $c \leq 1.0872$, then the following algorithm is faster.

▶ **Theorem 11.** *If $c \leq 1.0872$, there is a bounded-error quantum algorithm, which determines whether $(U, \mathcal{F})$ has a k-cover (resp., a k-partition or a k-packing) in*

$$O^*\left(\left(\min_{0.1303\leq\alpha\leq 0.25}\left((1+c)^{3/4}c^{\alpha/2}\left(1-4\cdot\alpha\right)^{\frac{4\cdot\alpha-1}{8}}\left(4\cdot\alpha\right)^{-\frac{\alpha}{2}},\ \alpha^{-\alpha}\cdot(1-\alpha)^{\alpha-1}\right)\right)^n\right)$$

time.

The first step is to use the algorithm from Theorem 6 with some $0.1303 \leq \alpha < 0.25$ and store the result in QRAM. Our algorithm is identical to the one presented in Theorem 10 except that we cannot directly look up whether a subset $X \subseteq U$ has a k-cover in QRAM for $|X| \leq n/4$. Instead, we use the following approach.

▶ **Lemma 12.** *If $c \leq 1.0872$ and $0.1303 \leq \alpha < 0.25$, after running the algorithm from Theorem 6, there exists a bounded-error quantum algorithm that checks for a k-cover of a subset $X \subseteq U$ where $|X| \leq n/4$ in $O^*\left(\sqrt{\binom{n/4}{\alpha\cdot n}\cdot c^{\alpha\cdot n}}\right)$ time.*

Proof. We use Grover's search to divide the elements of X into two sets (XL, XR) where $|XL| \geq \alpha \cdot n$ and k into $k_{XL} + k_{XR}$. We use Grover's search on the branching algorithm A to enumerate candidate sets $S_1 \in e(XL, r(\mathcal{F}, XL))$; if $|XL| - |S_1| > \alpha \cdot n$, then this branch is unsuccessful; otherwise, look up whether $XL \setminus S_1$ has a $(k_{XL} - 1)$-cover and XR has a k_{XR}-cover in QRAM. The correctness of this approach follows from Lemma 9.

As the function $f(x) = \binom{n/4}{x} c^x$ is strictly decreasing for $x > \frac{c}{c+1} \cdot \frac{n}{4}$ and

$$\alpha \cdot n \geq 0.1303 \cdot n > \frac{1.0872 \,/\, 4}{1.0872 + 1} \cdot n \geq \frac{c}{c+1} \cdot \frac{n}{4},$$

the running time of this algorithm is

$$O^*\left(\sqrt{\sum_{i=\lceil \alpha \cdot n \rceil}^{\lfloor n/4 \rfloor} \binom{n/4}{i} c^i}\right) = O^*\left(\sqrt{\binom{n/4}{\alpha \cdot n} c^{\alpha \cdot n}}\right).$$

This proves the lemma. ◀

Our algorithm is the same as the one in Theorem 8, except when we check for a $(k_{LL} - 1)$-cover for $LL \setminus S_1$ (resp., on LR, RL, RR) we use Lemma 12 instead of a direct lookup in QRAM.

Running the algorithm from Theorem 6 takes

$$O^*\left(\binom{n}{\alpha \cdot n}\right) = O^*\left(\alpha^{-\alpha n} \cdot (1-\alpha)^{(\alpha-1)n}\right)$$

time. The steps using Grover's search take

$$O^*\left(\sqrt{\sum_{l=\lceil n/2 \rceil}^{n} \binom{n}{l} c^l \sum_{l'=\lceil n/4 \rceil}^{\lfloor n/2 \rfloor} \binom{n/2}{l'} c^{l'}} \sqrt{\binom{0.25 \cdot n}{\alpha \cdot n} c^{\alpha \cdot n}}\right)$$
$$= O^*\left(\left((1+c)^{3/4} c^{\alpha/2} \left(\frac{1}{1-4\cdot\alpha}\right)^{\frac{1-4\cdot\alpha}{8}} \left(\frac{1}{4\cdot\alpha}\right)^{\frac{\alpha}{2}}\right)^n\right) \text{ time.}$$

The running time of the first part increases with α while the running time of the second part decreases with α. We can therefore optimise the running time by balancing the value of α for a given c. To determine when this approach is faster than $O^*\left(\binom{n}{n/4}\right)$ we compute the value of c when α is balanced at 0.25:

$$(1+c)^{3/4} c^{1/8} = \frac{4}{3^{3/4}}$$
$$\implies \qquad c \approx 1.08724 \ .$$

Therefore, this algorithm outperforms the algorithms of Theorem 8 and Theorem 10 when $c \leq 1.08723$.

Combining Theorem 8, Theorem 10, and Theorem 11, we obtain the following corollary.

▶ **Corollary 13.** *There is a bounded-error quantum algorithm, which determines whether $(U, \mathcal{F})$ has a k-cover (resp., a k-partition or a k-packing) in $O^*\left((2+c)^{n/2}\right)$ time, where $n = |U|$.*

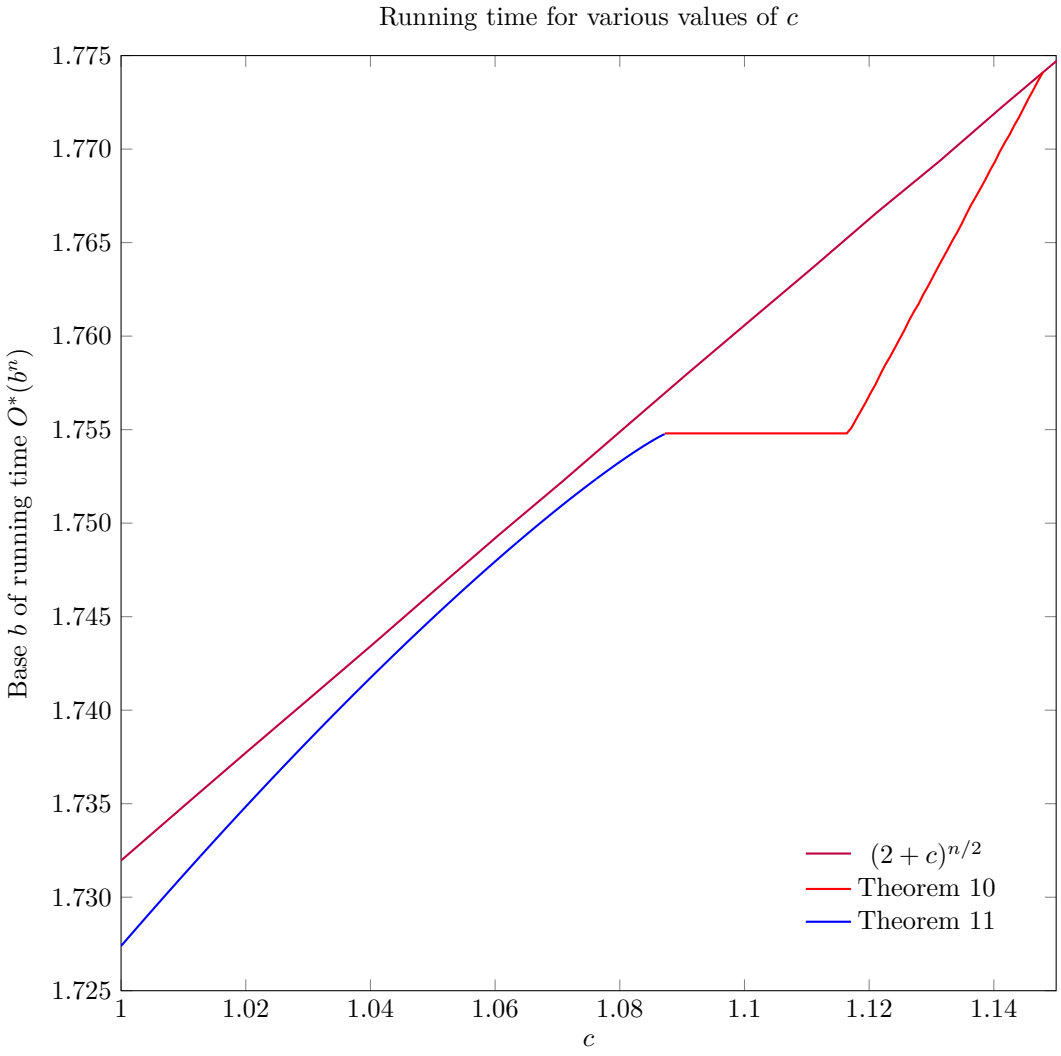


Figure 1 Visual presentation of running times of Theorem 10 and Theorem 11.

When $c \leq 1.147899$, then the running time provided by the best among Theorem 10 and Theorem 11 is slightly faster. At $c = 1$, the running time of Theorem 11 matches the running time of the Set Cover algorithm of [1] when $|\mathcal{F}|$ is subexponential in n.

When $c < -\frac{2}{3} + \sqrt[3]{\frac{47}{54} - \frac{\sqrt{93}}{18}} + \sqrt[3]{\frac{47}{54} + \frac{\sqrt{93}}{18}} \approx 1.147899$, then choosing the best algorithm among Theorem 10 and Theorem 11 gives an algorithm running in $O^*(b^n)$ time for some $b < \sqrt{c+2}$. This can be seen visually in Figure 1 and proved rigorously, e.g., by interleaving a stepwise function between the function $(c+2)^{n/2}$ and the function from Theorem 11 when $c \leq 1.08$:

$$\mathsf{steps}(c,n) = \begin{cases} 3^{n/2} & \text{if } c \leq 1.0123 \\ 1.735595^n & \text{if } 1.0123 \leq c \leq 1.0221 \\ \dots & \\ 1.7533^n & \text{if } 1.0742 \leq c \leq 1.08. \end{cases}$$

c	α	running time
1.0	0.236159	$O^*(1.7274^n)$
1.01	0.238036	$O^*(1.7312^n)$
1.02	0.239858	$O^*(1.7349^n)$
1.03	0.241622	$O^*(1.7384^n)$
1.04	0.243320	$O^*(1.7418^n)$
1.05	0.244946	$O^*(1.7450^n)$
1.06	0.246488	$O^*(1.7480^n)$
1.07	0.247928	$O^*(1.7508^n)$
1.08	0.249227	$O^*(1.7533^n)$

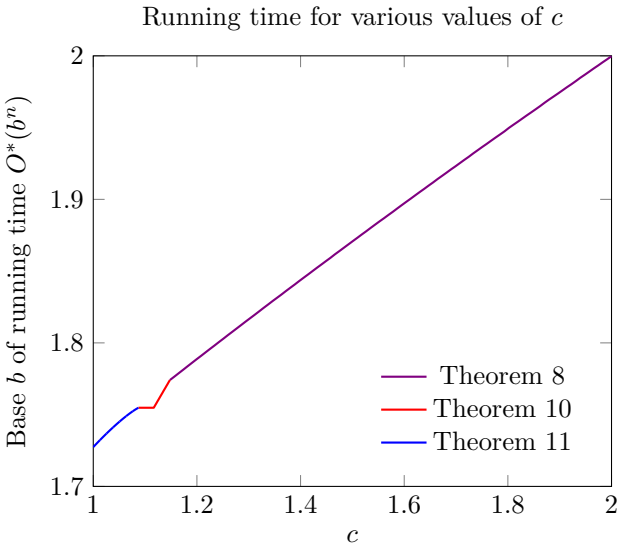


Figure 2 Running time for various values of c.

When $1.08 \leq c \leq 1.147899$, then Theorem 10 gives an algorithm with running time $O^*(d^n)$ for some $d < \sqrt{c+2}$. We list precise running times for certain values of c in Figure 2.

6 Applications

We can now use Corollary 13 in combination with simple branching algorithms enumerating various vertex sets in graphs, in particular algorithms enumerating maximal independent sets (equivalently, maximal cliques in the complement graph) in $O^*(3^{n/3})$ time [17] (we note that the algorithm of [24] is not a *simple* branching algorithm), maximal bicliques in $O^*(3^{n/3})$ time, maximal induced matchings in $O^*(10^{n/5})$ time [23], maximal induced forests in $O(1.8527^n)$ time [19], and minimal k-hitting sets in $O^*((2-1/k)^n)$ time [15]. The last result is used to enumerate maximal $\mathcal{H}$-free subgraphs, which have no induced subgraph isomorphic to any graph from the family $\mathcal{H}$ of graphs, all of which have at most k vertices. Some well-known $\mathcal{H}$-free graph classes are

- *cluster* graphs with $\mathcal{H} = \{P_3\}$, where P_k denotes the path on k vertices,
- *triangle-free* graphs with $\mathcal{H} = \{K_3\}$, where K_k denotes the complete graph on k vertices,
- *cographs* with $\mathcal{H} = \{P_4\}$ [12],
- *claw-free* graphs with $\mathcal{H} = \{K_{1,3}\}$, where $K_{k,\ell}$ denotes the complete bipartite graph with partite sets of size k and ℓ,
- *trivially-perfect* graphs with $\mathcal{H} = \{P_4, C_4\}$ [21], where C_k denotes the cycle on k vertices,
- *threshold* graphs with $\mathcal{H} = \{P_4, C_4, \overline{C_4}\}$ [11], where $\overline{G}$ denotes the complement of G,
- *split* graphs with $\mathcal{H} = \{C_4, \overline{C_4}, C_5\}$ [14], and
- *line* graphs, where $\mathcal{H}$ is a set of 9 graphs on at most 6 vertices each [3].

▶ **Theorem 14.** *There are bounded-error quantum algorithms, which, for a graph on n vertices and integer k, determine whether there is a k-packing, k-partitioning, or k-covering of G*

- *with maximal independent sets, maximal cliques, or maximal bicliques in $O(1.8554^n)$ time,*
- *with maximal cluster graphs or maximal triangle-free graphs in $O(1.9149^n)$ time,*

- *with maximal cographs, maximal claw-free graphs, maximal trivially-perfect graphs, or maximal threshold graphs in* $O(1.9365^n)$ *time,*
- *with maximal split graphs in* $O(1.9494^n)$ *time,*
- *with maximal line graphs in* $O(1.9579^n)$ *time, and*
- *with maximal induced forests in* $O(1.9629^n)$ *time.*

There are bounded-error quantum algorithms, which, for a graph on n *vertices and integer* k*, determine whether there is a* k*-packing or* k*-partitioning of* G

- *with maximal induced matchings in* $O(1.8934^n)$ *time*[2]*.*

In particular, this leads to a bounded-error quantum algorithm computing the chromatic number of an input graph in $O(1.8554^n)$ time; a graph can be covered with k maximal independent sets iff it has chromatic number at most k. In Section 7, we expedite this algorithm by exploiting fast algorithms for coloring a graph with a small number of colors and this will enable us to partition the vertex set in a more balanced way in the divide-and-conquer steps.

Even though there is a simple branching algorithm that enumerates all minimal dominating sets in $O(1.7159^n)$ time [16], this algorithm cannot be readily used for computing the domatic number using Theorem 8. This is because when we consider a subset X of the vertex set of $G = (V, E)$, we need to enumerate vertex subsets from X that are minimal dominating sets for G, and not $G[X]$. In Section 8, we prove that such minimal dominating sets can be enumerated in $O^*\left((2-\varepsilon)^{|X|}\right)$ time if $|X|$ is linear in $|V|$.

7 Faster Computation of the Chromatic Number

Assume the vertex set of input graph $G = (V, E)$ can be partitioned into independent sets $C = (I_1, I_2, \ldots, I_\chi)$ where χ is the chromatic number of G. Denote by n the number of vertices of G.

Using the algorithm from Theorem 8, we can compute the chromatic number of G in $O(1.8554^n)$ time. The family of subsets $\mathcal{F}$ corresponds to the independent sets of G and the family $e(X, r(\mathcal{F}, X))$ corresponds to the maximal independent sets of $G[X]$, which can be enumerated by a simple branching algorithm A in $O(3^{|X|/3})$ time [17]. We now prove a stronger result.

A proof of Lemma 17 along with a more detailed complexity analysis can be found in the full version of the paper [20].

▶ **Theorem 15.** *There is a bounded-error quantum algorithm, which for a graph on* n *vertices, computes the chromatic number in* $O(1.7956^n)$ *time.*

Assume, w.l.o.g., that $|I_1| \geq |I_2| \geq |I_3| \geq |I_4| \geq \cdots \geq |I_\chi|$.

In the first step, we use the algorithm from Theorem 6 with $\alpha = 0.27$ and store the result in QRAM. That is, for each α-small subset $X \subset V$, we find the chromatic number of X by finding the smallest k such that there exists a k-partition of X into independent sets. This takes $O^*(\binom{n}{0.27 \cdot n}) = O^*(1.79187^n)$ time.

We then consider a few different possibilities for the large sets in C and present an algorithm which finds a partition into a smallest number of independent sets for each of these cases. These possibilities cover all configurations of C so one result is guaranteed to find the chromatic number by detecting the partition C above, or an equivalent partition. The cases are as follows:

[2] Note that it is NP-hard to determine, for a graph $G = (V, E)$ and vertex subset $X \subseteq V$, whether there is a superset $Y \supseteq X$ that induces a 1-regular subgraph of G. This can be seen by a simple reduction from Independent Set.

1. G contains a vertex subset of size at least $0.48 \cdot n$ that is the union of at most five sets in C.
   We check for k-coloring for all $1 \leq k \leq n$ and take the minimum valid value of k as the chromatic number. To check for a certain k, we use Grover's search over all $Q \subset V$ where $|Q| \leq 0.52 \cdot n$ and check whether $G - Q$ is 5-colorable. If it is 5-colorable, we compute its chromatic number by checking its k'-colorability for $k' < 5$. We then solve the instance $(Q, r(\mathcal{F}, Q))$ to find a k_Q-coloring, where $k_Q = k - \chi(G - Q)$, using Grover's search to divide Q into (QL, QR) where $|QL| \geq 0.27 \cdot n$ and $k_{QL} + k_{QR} = k_Q$. For $(QL, r(\mathcal{F}, QL))$ (and similarly for QR) we use Grover's search on the branching algorithm A to enumerate candidate sets $S_1 \in e(QL, r(\mathcal{F}, QL))$; if $|QL| - |S_1| > 0.27 \cdot n$, then this branch is unsuccessful; otherwise, find $\chi(QL \setminus S_1)$ in QRAM.
   The running time of this case with quantum 5-coloring in $O^*(1.4695^n)$ [26] is $O^*(1.7831^n)$.
2. G contains a vertex subset of size at least $0.48 \cdot n$ that is the union of six sets in C, but does not contain a vertex subset of size at least $0.48 \cdot n$ that is the union of at most five sets in C.
   In this case we have

$$\begin{aligned} & |I_1| + |I_2| + \cdots + |I_5| < 0.48 \cdot n \\ \implies & 5 \cdot |I_6| \leq 5 \cdot |I_5| < 0.48 \cdot n \\ \implies & |I_1| + |I_2| + \cdots + |I_6| < 0.576 \cdot n, \end{aligned}$$

   which, along with the condition $0.48 \cdot n \leq |\bigcup_{i=1}^{6} I_i|$ gives

$$0.424 \cdot n < |V \setminus \bigcup_{i=1}^{6} I_i| \leq 0.52 \cdot n.$$

   For this case we check for k-coloring for all $1 \leq k \leq n$ and take the minimum valid value of k as the chromatic number. To check for a certain k, we use Grover's search over all $Q \subset V$ where $0.424 \cdot n < |Q| \leq 0.52 \cdot n$ and check whether $G - Q$ is 6-colorable. If so, we check whether the instance $(Q, r(\mathcal{F}, Q))$ is $(k-6)$-colorable in the same way as in item 1.
   The running time of this case with quantum 6-coloring in $O^*(1.5261^n)$ [26] is $O^*(1.7937^n)$.
3. G does not contain a vertex subset of size at least $0.48 \cdot n$ that is the union of at most six sets in C. That is $|\bigcup_{i=1}^{6} I_i| < 0.48 \cdot n$.
   Let $T = \bigcup_{i=1}^{q} I_i$ where q is the maximum index such that $|T| < \frac{n}{2}$. As $|\bigcup_{i=1}^{6} I_i| < 0.48 \cdot n$, we have $q \geq 6$. We consider two possibilities for the size of T and present an algorithm which finds a partition into a smallest number of independent sets for each case. As a valid coloring is computed in each case, the smallest partition gives the chromatic number if $|\bigcup_{i=1}^{6} I_i| < 0.48 \cdot n$.

   3.1. Consider the case where $|T| < \frac{6 \cdot n}{13}$:
   Let $L = \bigcup_{i=1}^{q+1} I_i$ and $R = \bigcup_{q+2}^{\chi} I_i$. We have

$$\begin{aligned} & |I_1| + |I_2| + \cdots + |I_q| < \frac{6 \cdot n}{13} \\ \implies & |I_1| + |I_2| + \cdots + |I_{q+1}| < \frac{7 \cdot n}{13} \\ \iff & |L| < \frac{7 \cdot n}{13} \end{aligned}$$

   and $0.5 \cdot n \leq |L|$ from the definition of q.

Assume L is not 7-colorable. There must be more than 7 independent sets in the construction of of L, which means

$$\begin{aligned} & q+1>7 \iff q \geq 7 \\ \implies & 7 \cdot |I_{q+1}| < \frac{6 \cdot n}{13} \\ \implies & |L| < \frac{48 \cdot n}{91} \end{aligned}$$

By contraposition, $|L| \geq \frac{48 \cdot n}{91}$ implies that L is 7-colorable.

▶ **Lemma 16.** *After running the algorithm presented in Theorem 6, there exists a bounded-error quantum algorithm to check the 7-colorability of a subset $X \subseteq U$ where $|X| \leq \frac{7}{3} \cdot \alpha \cdot n$ in $O^*(1.5622^{|X|})$ time.*

Proof. Assume that X is 7-colorable and there is a partition $D = (J_1, J_2, \ldots, J_7)$ of X into independent sets. Assume, w.l.o.g., that $|J_1| \geq |J_2| \geq \cdots \geq |J_7|$. Let $TL = J_1 \cup J_2 \cup J_3$ and $TR = J_4 \cup J_5 \cup J_6 \cup J_7$. We have

$$\begin{aligned} & \frac{|TL|}{3} \geq |J_3| \geq |J_4| \geq \frac{|TR|}{4} \\ \implies & |TL| + \frac{4}{3} \cdot |TL| \geq |X| \\ \iff & |TL| \geq \frac{3}{7} \cdot |X|. \end{aligned}$$

Consider $TR \setminus J_4$,

$$\begin{aligned} & |TR| \leq \frac{4}{7} \cdot |X| \text{ and } |J_4| \geq \frac{1}{4} \cdot |TR| \\ \implies & |TR \setminus J_4| \leq \frac{3}{7} \cdot |X| \leq \alpha \cdot n. \end{aligned}$$

So there exists a subset $S_1 \in e(TR, r(\mathcal{F}, TR))$ such that $|TR \setminus S_1| \leq \alpha \cdot n$.
We use Grover's search to divide X into two sets (XL, XR) where $|XL| \geq \frac{3 \cdot |X|}{7}$. We check XL for 3-colorability using a fast quantum algorithm. If it is 3-colorable, we use Grover's search on the branching algorithm A to enumerate candidate sets $S_1 \in e(XR, r(\mathcal{F}, XR))$; if $|XR| - |S_1| > \alpha \cdot n$, then this branch is unsuccessful; otherwise, check whether $\chi(XR \setminus S_1) \leq 3$ in QRAM. If X is 7-colorable and $(XL, XR) = (TL, TR)$, we indeed detect 7-colorability.
The running time of this algorithm with quantum 3-coloring in $O^*(1.1528^n)$ time [18] is $O^*(1.5622^{|X|})$. ◀

We check for k-coloring for all $1 \leq k \leq n$ and take the minimum valid value of k as the chromatic number. To check for a certain k, we use Grover's search to divide V into two sets (L, R) where $0.5 \cdot n \leq |L| < \frac{7 \cdot n}{13}$ and k into $k_L + k_R$. When $|L| < \frac{48 \cdot n}{91}$, we solve the subinstances $(L, r(\mathcal{F}, L))$ and $(R, r(\mathcal{F}, R))$ as in Theorem 8. When $|L| \geq \frac{48 \cdot n}{91}$, we solve the subinstance $(R, r(\mathcal{F}, R))$ as in Theorem 8 and run the algorithm in Lemma 16 on L; if L isn't 7-colorable, then this branch is unsuccessful; otherwise the subinstance $(L, r(\mathcal{F}, L))$ is k_L-colorable for $k_L \geq 7$. This algorithm finds the chromatic number of G when (L, R) are equal to the ones we constructed from C. The running time of this algorithm is $O^*(1.7956^n)$.

3.2. Otherwise, consider the case where $\frac{6 \cdot n}{13} \leq |T| < \frac{n}{2}$:
Let $L = T = \bigcup_{i=1}^{q} I_i$ and $R = \bigcup_{q+1}^{\chi} I_i$. We partition R into independent sets $C' = (I'_1, I'_2, \ldots, I'_{\chi'})$ where $I'_i = I_{q+i}$ for all $1 \leq i \leq \chi'$. By the definition of C, this is an optimal partition of R. As $q \geq 6$, we get $\frac{|L|}{6} \geq |I_q| \geq |I_{q+i}| = |I'_i|$ for all $1 \leq i \leq \chi'$.

▶ **Lemma 17.** *Assume we have a partition $D = (J_1, J_2, \ldots, J_m)$ of a set X where $t \geq |J_1| \geq |J_2| \geq \cdots \geq |J_m|$. Define $p = \left\lceil \frac{|X|}{t} \right\rceil$ and $r = \frac{|X|}{p}$. For any non-negative integer a, $1 \leq a \leq p-2$, there exists an integer k such that $a \cdot r \leq |\bigcup_{i=1}^{k} J_i| \leq (a+1) \cdot r$.*
Let $T_a = \bigcup_{i=1}^{q'} I'_i$ and $T_b = \bigcup_{i=1}^{q'+1} I'_i$ where q' is the maximum index such that $|T_a| < \frac{|R|}{2}$. Note that $\frac{|R|}{2} \leq |T_b|$. When we apply Lemma 17 with $X = R$, $D = C'$ and $t = \frac{|L|}{6}$, we get

$$\begin{aligned} \frac{|X|}{t} = \frac{6 \cdot |R|}{|L|} &\leq \frac{6 \cdot \frac{7 \cdot n}{13}}{\frac{6 \cdot n}{13}} = 7 \\ \text{and} \quad \frac{6 \cdot |R|}{|L|} &> \frac{6 \cdot \frac{n}{2}}{\frac{n}{2}} = 6 \\ \implies p = \left\lceil \frac{|X|}{t} \right\rceil &= 7. \end{aligned}$$

If we let $a = 3$, the lemma states that there exists a k such that $\frac{3 \cdot |R|}{7} \leq |\bigcup_{i=1}^{k} I'_i| \leq \frac{4 \cdot |R|}{7}$. As q' and $q'+1$ differ by 1, it is not possible that $|\bigcup_{i=1}^{q'} I'_i| < \frac{3 \cdot |R|}{7} \leq |\bigcup_{i=1}^{k} I'_i| \leq \frac{4 \cdot |R|}{7} < |\bigcup_{i=1}^{q'+1} I'_i|$. So, either $\frac{3 \cdot |R|}{7} \leq |T_a| < \frac{|R|}{2}$ or $\frac{|R|}{2} \leq |T_b| \leq \frac{4 \cdot |R|}{7}$. We let $TL = T_a$ or $TL = T_b$ such that $\frac{3 \cdot |R|}{7} \leq |TL| \leq \frac{4 \cdot |R|}{7}$ and $TR = R \setminus TL$. Note that in both cases, removing the independent set $I'_{q'+1}$ from either TL or TR would leave both subsets with a size $\leq \frac{|R|}{2}$. Therefore, there exists a subset $S_1 \in e(TL, r(\mathcal{F}, TL))$ (resp. for TR) such that $|TL \setminus S_1| \leq \frac{|R|}{2} \leq 0.27$ (and similarly for TR).

We check for k-coloring for all $1 \leq k \leq n$ and take the minimum valid value of k as the chromatic number. To check for a certain k, we use Grover's search to divide V into two sets (L, R) where $\frac{6 \cdot n}{13} \leq |L| < 0.5 \cdot n$ and k into $k_L + k_R$. We solve the subinstance $(L, r(\mathcal{F}, L))$ as in Theorem 8. We also solve the subinstance $(R, r(\mathcal{F}, R))$ as in Theorem 8, except that R is divided into (RL, RR) with $\frac{3 \cdot |R|}{7} \leq |RL| \leq \frac{4 \cdot |R|}{7}$ instead of $|RL| > 0.5 \cdot |R|$. This algorithm finds the chromatic number of G when (L, R) are equal to the ones we constructed from C and $(RL, RR) = (TL, TR)$. The running time of this algorithm is $O^*(1.7956^n)$.

We observe that the overall running time of the algorithm is

$$\begin{aligned} &O^*\left(\sqrt{\binom{n}{\frac{7}{13} \cdot n}\binom{\frac{7}{13} \cdot n}{\frac{4}{13} \cdot n} 3^{\frac{1}{3} \cdot \frac{4}{13} \cdot n}}\right) \\ &= O^*\left(\left(\frac{\sqrt{13}}{2^{7/13} \cdot 3^{23/78}}\right)^n\right) \\ &= O^*(1.7956^n) \end{aligned}$$

We reach this worst case when $\chi(G) = 13$ and $|I_1| = |I_2| = |I_3| = \cdots = |I_{13}|$. An example of such a graph is the disjoint union of $\frac{n}{13}$ complete graphs on 13 vertices. If the universe is partitioned into 2, one part must have at least 7 independent sets which is then partitioned into two parts where one part has at least 4 independent sets. Hence we cannot improve on the running time with a different twice-partitioning strategy.

We also note that the current best known quantum algorithms for checking 13-colorability, 7-colorability and 4-colorability [26] do not improve our running time. When iterating through $S_1 \in (X, r(\mathcal{F}, X))$, we only need to consider S_1 with $|S_1| \geq |X| - \alpha \cdot n$ so we can use an improved upper bound [10] when we only need to consider $|S_1| > |X| / 3$. However, this is only the case when $|X| > \frac{3}{2} \cdot \alpha \cdot n$; so it does not affect our overall running time as $|X| = \frac{4}{13} \cdot n$ in the worst case.

The pseudocode for the algorithm in this section can be found in [20].

8 Enumeration of Minimal Subset Dominating Sets

In this section, we prove that the number of minimal dominating sets of a graph that are subsets of some linear-sized subset of vertices X is at most $O^*\left((2-\varepsilon)^{|X|}\right)$. Moreover, they can be enumerated by a simple branching algorithm whose running time is within a polynomial factor of this bound.

▶ **Theorem 18.** *There is a simple branching algorithm, which, given any graph $G = (V, E)$ and any subset of vertices $X \subseteq V$ with $|X| \geq d \cdot |V|$ for some $d > 0$, enumerates all minimal dominating sets of G that are subsets of X in $O^*\left((2-\varepsilon_d)^{|X|}\right)$ time, for some $\varepsilon_d > 0$.*

The theorem will follow from a slightly more general theorem about minimal set covers of a set system. From a graph $G = (V, E)$ and a vertex subset $X \subseteq V$, we obtain a set system $(U, \mathcal{F})$ where $U = V$ and a set $S_x \in \mathcal{F}$ for each $x \in X$ that contains the closed neighborhood of vertex x in the graph G: $S_x = N_G[x]$. Then, there is a 1-to-1 correspondence between inclusion-wise minimal dominating sets in G that are subsets of X and inclusion-wise minimal set covers of $(U, \mathcal{F})$.

▶ **Theorem 19.** *There is a simple branching algorithm, which, given any set system $(U, \mathcal{F})$ with $|U| \leq r \cdot |\mathcal{F}|$ for some $r > 0$, enumerates all minimal set covers of $(U, \mathcal{F})$ in $O^*\left(2^{(1-\varepsilon_r)\cdot|\mathcal{F}|}\right)$ time, for some $\varepsilon_r > 0$.*

Fomin et al. [16] proved Theorem 18 for $X = V$ and Theorem 19 for $r = 1$. In particular, their algorithm enumerates all minimal dominating sets of a graph on n vertices in $O(1.7159^n)$ time.

Proof. Let $r > 0$. Let $\varepsilon = \frac{3(r+1)-\log(2^{3(r+1)}-1)}{3(r+1)^2}$. Consider the measure

$$\mu(U, \mathcal{F}) = (1 - (r+1)\varepsilon)|\mathcal{F}| + \varepsilon|U|$$

which associates a weight of $1 - (r+1)\varepsilon > 0$ to each set and a weight of $\varepsilon > 0$ to each element of a set system $(U, \mathcal{F})$. We will show that every set system $(U, \mathcal{F})$ has at most $2^{\mu(U,\mathcal{F})}$ minimal set covers by induction on $|\mathcal{F}|$. For a set system with $|U| \leq r \cdot |\mathcal{F}|$, this number is $2^{(1-(r+1)\varepsilon)|\mathcal{F}|+\varepsilon|U|} \leq 2^{(1-\varepsilon)\cdot|\mathcal{F}|}$. The proof can easily be turned into a simple branching algorithm enumerating all minimal set covers whose search tree is the induction tree of this proof.

The statement trivially holds when $|\mathcal{F}| = 0$ since such an instance has at most 1 minimal set cover. For the induction, we consider three cases.

1. $\mathcal{F}$ contains a set S with $|S| \geq 3(r+1)$. Any minimal set cover either contains S or not. Those that do not contain S are also minimal set covers of $(U, \mathcal{F} \setminus \{S\})$ and those that contain S are made up of S and a minimal set cover of $(U \setminus S, \mathcal{F} \setminus \{S\})$. For the induction, we would like that

$$\begin{aligned} & 2^{\mu(U,\mathcal{F}\setminus\{S\})} + 2^{\mu(U\setminus S,\mathcal{F}\setminus\{S\})} \leq 2^{\mu(U,\mathcal{F})} \\ \iff \quad & 2^{\mu(U,\mathcal{F})-(1-(r+1)\varepsilon)} + 2^{\mu(U,\mathcal{F})-(1-(r+1)\varepsilon)-|S|\cdot\varepsilon} \leq 2^{\mu(U,\mathcal{F})} \\ \iff \quad & 2^{-1+(r+1)\varepsilon} + 2^{-1+(r+1)\varepsilon-|S|\cdot\varepsilon} \leq 1 \\ \impliedby \quad & 2^{-1+(r+1)\varepsilon} + 2^{-1-2(r+1)\varepsilon} \leq 1, \end{aligned}$$

   and this inequality holds because $\varepsilon \leq \frac{1}{r+1} \log\left(\frac{1+\sqrt{5}}{2}\right)$.
2. There is an element $u \in U$ with frequency at most $3(r+1)$, i.e., u occurs in at most $3(r+1)$ sets of $\mathcal{F}$. Denote the sets that contain u by $\mathcal{S} = \{S \in \mathcal{F} : u \in S\}$. Since u needs to be covered, each set cover contains at least one set from $\mathcal{S}$. We use induction on all $2^{|\mathcal{S}|} - 1$ choices of including at least one set from $\mathcal{S}$ into the set covers and excluding the remaining sets from $\mathcal{S}$; each such choice leads to a set cover instance where we remove all sets in $\mathcal{S}$, and we remove all elements covered by the sets that are included in the set covers. Each such choice reduces the measure μ by more than $|\mathcal{S}| \cdot (1-(r+1)\varepsilon)$. Now, we would therefore like that

$$\begin{aligned} & (2^{|\mathcal{S}|}-1) \cdot 2^{\mu(U,\mathcal{F})-|\mathcal{S}|\cdot(1-(r+1)\varepsilon)} \leq 2^{\mu(U,\mathcal{F})} \\ \iff \quad & (2^{|\mathcal{S}|}-1) \cdot 2^{-|\mathcal{S}|\cdot(1-(r+1)\varepsilon)} \leq 1 \\ \iff \quad & 2^{-|\mathcal{S}|\cdot(1-(r+1)\varepsilon)} \leq (2^{|\mathcal{S}|}-1)^{-1} \\ \iff \quad & -|\mathcal{S}| \cdot (1-(r+1)\varepsilon) \leq -\log(2^{|\mathcal{S}|}-1) \\ \iff \quad & (r+1)\varepsilon - 1 \leq \frac{-\log(2^{|\mathcal{S}|}-1)}{|\mathcal{S}|} \\ \iff \quad & \varepsilon \leq \frac{|\mathcal{S}| - \log(2^{|\mathcal{S}|}-1)}{(r+1)|\mathcal{S}|} \end{aligned}$$

   Note that $\frac{|\mathcal{S}|-\log(2^{|\mathcal{S}|}-1)}{(r+1)|\mathcal{S}|}$ decreases when $|\mathcal{S}|$ increases, and for the maximum possible value of $|\mathcal{S}|$, which is $3(r+1)$, the inequality holds with equality for the value of ε given in the beginning of the proof.
3. It remains to consider the case where all sets have size less than $3(r+1)$ and all elements have frequency more than $3(r+1)$. Since the sum of set sizes equals the sum of element frequencies, we have that $|\mathcal{F}| \geq |U|$. Here, we use the result of Fomin et al. [16] who proved that the number of minimal set covers is at most $1.7159^{|\mathcal{F}|}$. For the induction, we would like that

$$\begin{aligned} & 1.7159^{|\mathcal{F}|} \leq 2^{\mu(U,\mathcal{F})} \\ \iff \quad & 2^{|\mathcal{F}|\log 1.7159} \leq 2^{(1-(r+1)\varepsilon)|\mathcal{F}|+\varepsilon|U|} \\ \impliedby \quad & 0.779|\mathcal{F}| \leq (1-(r+1)\varepsilon)|\mathcal{F}| \\ \iff \quad & \varepsilon \leq \frac{0.221}{r+1} \end{aligned}$$

   and this inequality holds for the value of ε given in the beginning of the proof.

This concludes the proof of the theorem. ◀

Theorem 8 now lets us conclude the following.

▶ **Corollary 20.** *There is a bounded-error quantum algorithm which computes the domatic number of any graph on n vertices in $O((2-\varepsilon)^n)$ time for some constant $\varepsilon > 0$.*

References

1 Andris Ambainis, Kaspars Balodis, Janis Iraids, Martins Kokainis, Krisjanis Prusis, and Jevgenijs Vihrovs. Quantum speedups for exponential-time dynamic programming algorithms. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 1783–1793. SIAM, 2019. doi:10.1137/1.9781611975482.107.

2 Andris Ambainis and Martins Kokainis. Quantum algorithm for tree size estimation, with applications to backtracking and 2-player games. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2017)*, pages 989–1002. ACM, 2017. doi:10.1145/3055399.3055444.

3 Lowell W. Beineke. Characterizations of derived graphs. *Journal of Combinatorial Theory, Series B*, 9:129–135, 1970. doi:10.1016/S0021-9800(70)80019-9.

4 Andreas Björklund and Thore Husfeldt. Inclusion–exclusion algorithms for counting set partitions. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006)*, pages 575–582. IEEE Computer Society, 2006. doi:10.1109/FOCS.2006.41.

5 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC 2007)*, pages 67–74. ACM, 2007. doi:10.1145/1250790.1250801.

6 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Trimmed Moebius inversion and graphs of bounded degree. *Theory of Computing Systems*, 47(3):637–654, 2010. doi:10.1007/S00224-009-9185-7.

7 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Covering and packing in linear space. *Information Processing Letters*, 111(21-22):1033–1036, 2011. doi:10.1016/J.IPL.2011.08.002.

8 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009. doi:10.1137/070683933.

9 Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics*, 46(4-5):493–505, 1998. doi:10.1002/(SICI)1521-3978(199806)46:4/5\%3C493::AID-PROP493\%3E3.0.CO;2-P.

10 Jesper Makholm Byskov. Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters*, 32(6):547–556, 2004. doi:10.1016/j.orl.2004.03.002.

11 V. Chvátal and P. L. Hammer. Set-packing problem and threshold graphs. Technical report, University of Waterloo Research Report, 1973. CORR 73-21.

12 D. G. Corneil, H. Lerchs, and L. Stewart Burlingham. Complement reducible graphs. *Discrete Applied Mathematics*, 3(3):163–174, 1981. doi:10.1016/0166-218X(81)90013-5.

13 Christoph Dürr and Peter Høyer. A quantum algorithm for finding the minimum. *CoRR*, quant-ph/9607014, 1996. arXiv:quant-ph/9607014.

14 Stephan Földes and Peter L. Hammer. Split graphs. In *Proceedings of the 8th Southeastern Conference on Combinatorics, Graph Theory and Computing*, pages 311–315, 1977.

15 Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. *Journal of the ACM*, 66(2):8:1–8:23, 2019. doi:10.1145/3284176.

16 Fedor V. Fomin, Fabrizio Grandoni, Artem V. Pyatkin, and Alexey A. Stepanov. Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. *ACM Transactions on Algorithms*, 5(1):9:1–9:17, 2008. doi:10.1145/1435375.1435384.

17 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010. doi:10.1007/978-3-642-16533-7.

18 Martin Fürer. Solving NP-complete problems with quantum search. In *Proceedings of the 8th Latin American Symposium on Theoretical Informatics (LATIN 2008)*, volume 4957 of *Lecture Notes in Computer Science*, pages 784–792. Springer, 2008. doi:10.1007/978-3-540-78773-0_67.

19 Serge Gaspers and Edward J. Lee. Exact algorithms via multivariate subroutines. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *LIPIcs*, pages 69:1–69:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.69.

20 Serge Gaspers and Jerry Zirui Li. Quantum algorithms for graph coloring and other partitioning, covering, and packing problems. *CoRR*, abs/2311.08042, 2023. doi:10.48550/arXiv.2311.08042.

21 Martin C. Golumbic. Trivially perfect graphs. *Discrete Mathematics*, 24(1):105–107, 1978. doi:10.1016/0012-365X(78)90178-4.

22 Lov K. Grover. A fast quantum mechanical algorithm for database search. In Gary L. Miller, editor, *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC 1996)*, pages 212–219. ACM, 1996. doi:10.1145/237814.237866.

23 Sushmita Gupta, Venkatesh Raman, and Saket Saurabh. Maximum r-regular induced subgraph problem: Fast exponential algorithms and combinatorial bounds. *SIAM Journal on Discrete Mathematics*, 26(4):1758–1780, 2012. doi:10.1137/09077850X.

24 David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988. doi:10.1016/0020-0190(88)90065-8.

25 Mikko Koivisto. An $O^*(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion–exclusion. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006)*, pages 583–590. IEEE Computer Society, 2006. doi:10.1109/FOCS.2006.11.

26 Kazuya Shimizu and Ryuhei Mori. Exponential-time quantum algorithms for graph coloring problems. *Algorithmica*, 84(12):3603–3621, 2022. doi:10.1007/S00453-022-00976-2.

27 F. Yates. The design and analysis of factorial experiments. *Imperial Bureau of Soil Science, Harpenden, England*, Technical Communication No. 35, 1937.

BQP, Meet NP: Search-To-Decision Reductions and Approximate Counting

Sevag Gharibian
Department of Computer Science and Institute for Photonic Quantum Systems (PhoQS), Paderborn University, Germany

Jonas Kamminga
Department of Computer Science and Institute for Photonic Quantum Systems (PhoQS), Paderborn University, Germany

Abstract

What is the power of polynomial-time quantum computation with access to an NP oracle? In this work, we focus on two fundamental tasks from the study of Boolean satisfiability (SAT) problems: search-to-decision reductions, and approximate counting. We first show that, in strong contrast to the classical setting where a poly-time Turing machine requires $\Theta(n)$ queries to an NP oracle to compute a witness to a given SAT formula, quantumly $\Theta(\log n)$ queries suffice. We then show this is tight in the black-box model – any quantum algorithm with "NP-like" query access to a formula requires $\Omega(\log n)$ queries to extract a solution with constant probability.

Moving to approximate counting of SAT solutions, by exploiting a *quantum* link between search-to-decision reductions and approximate counting, we show that existing classical approximate counting algorithms are likely optimal. First, we give a lower bound in the "NP-like" black-box query setting: Approximate counting requires $\Omega(\log n)$ queries, even on a quantum computer. We then give a "white-box" lower bound (i.e. where the input formula is not hidden in the oracle) – if there exists a randomized poly-time classical or quantum algorithm for approximate counting making $o(\log n)$ NP queries, then $\mathsf{BPP}^{\mathsf{NP}[o(n)]}$ contains a P^{NP}-complete problem if the algorithm is classical and $\mathsf{FBQP}^{\mathsf{NP}[o(n)]}$ contains an $\mathsf{FP}^{\mathsf{NP}}$-complete problem if the algorithm is quantum.

2012 ACM Subject Classification Theory of computation → Quantum complexity theory

Keywords and phrases Approximate Counting, Search to Decision Reduction, BQP, NP, Oracle Complexity Class

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.70

Category Track A: Algorithms, Complexity and Games

Related Version *Previous Version*: https://arxiv.org/abs/2401.03943

Funding *Sevag Gharibian*: supported by the DFG under grant numbers 432788384 and 450041824, the BMBF within the funding program "Quantum Technologies - from Basic Research to Market" via project PhoQuant (grant number 13N16103), and the project "PhoQC" from the programme "Profilbildung 2020", an initiative of the Ministry of Culture and Science of the State of Northrhine Westphalia.
Jonas Kamminga: supported by the DFG under grant number 450041824.

Acknowledgements The authors would like to thank Ronald de Wolf, Dieter van Melkebeek, Osamu Watanabe, Henry Yuen, Scott Aaronson, William Kretschmer and Eric Allender for helpful comments and remarks.

1 Introduction

A fundamental direction of study in classical complexity theory is: What can P or BPP achieve with access to an NP oracle? Here, the study of relational-problems, i.e. where the output is not a single bit, but a string, has proven particularly fruitful. (Formally, this refers

to the classes FunctionP (FP) and FunctionBPP (FBPP); see Section 2.) A first direction here has been *search-to-decision* reductions. Namely, given a SAT formula φ and an NP oracle, it is well-known that $O(n)$ queries suffice to extract a solution to φ (assuming one exists). Moreover, this is likely classically optimal: an $o(n)$-query algorithm would violate the Exponential Time Hypothesis [17][1]. (Before ETH was posited, Krentel showed that if $O(\log n)$ queries suffice, then $\mathsf{P} = \mathsf{NP}$ [21].)

A second key direction has been *approximate counting* of the number of solutions of a Boolean formula, first studied by Stockmeyer [22]. Approximate counting has proven widely influential, even having applications in quantum advantage frameworks such as Aaronson and Arkhipov's Boson Sampling [2]. Stockmeyer showed [22] that an FBPP machine making $O(\log n \log\log n)$ NP queries suffices to approximate the number of solutions within a constant multiplicative factor, and that at least $\Omega(\log n)$ queries are required. This gap was closed by Chakraborty, Meel, and Vardi, who improved the upper bound to $O(\log n)$ queries [13]. Thus, the NP-query complexity of these two tasks is now well understood.

The quantum setting. The guiding question of this work is the next natural frontier: *Can* quantum *access to an NP oracle reduce the number of required queries?* For relation problems, this is a particularly intriguing question: Intuitively, a single classical NP query yields only 1 bit of information, suggesting that if an FP machine wishes to produce an n-bit output, then $\Theta(n)$ queries are necessary. (Indeed, as mentioned above, this is the case for search-to-decision reduction of SAT, assuming ETH.) *Quantum* access to an oracle, however, can sometimes bypass this obstacle, producing n bit outputs with just a *single* query. A notable instance of this is the Bernstein-Vazirani algorithm [11], which requires just a single query to an oracle encoding an affine function $f(x) = a \cdot x + b$ to output string $x \in \{0,1\}^n$. We thus ask: *Can a FunctionBQP (*FBQP*) machine make fewer queries to an* NP *oracle to extract a* SAT *solution or approximately count the number of solutions?*

Our results

In this work, we give tight resolutions to this question for both tasks.

Search-to-decision reductions. As mentioned earlier, classically, $\Theta(n)$ NP queries are necessary and sufficient for search-to-decision reduction for SAT, assuming ETH. Before proceeding to our main results, the lower bounds, we show that $O(\log n)$ queries suffice quantumly.

▶ **Theorem 1.** $\mathsf{FNP} \subseteq \mathsf{FBQP}^{\mathsf{NP}[\log]}$.

Here, FunctionNP (FNP) asks to produce a witness to an NP relation (Section 2), and NP[log] in the exponent denotes $O(\log n)$ NP queries. We remark that independently and prior to this work, Irani, Natarajan, Nirkhe, Rao and Yuen [18] showed that for SAT formulae with a *unique* satisfying assignment, a single NP query suffices to extract said solution (see Related Work).

[1] If there is an $\mathsf{FP}^{\mathsf{NP}[o(n)]}$ algorithm outputting a satisfying assignment then SAT can be decided in time $2^{o(n)}$ as follows: Enumerate through all possible strings y of $o(n)$ NP query answers, which takes $2^{o(n)}$ time. For each y, run the FP machine on y to obtain candidate solution x, and check if $\varphi(x) = 1$.

Is Theorem 1 tight? Unfortunately, since we are in the *white-box model* for search-to-decision reductions (i.e. the input formula φ is given as input to the FBQP machine, rather then hidden in the oracle), even a single-query lower bound would imply[2] NP $\not\subseteq$ BQP, and is thus likely out of reach. We hence move to the *black-box model* in order to prove a lower bound. For this, note that the standard quantum query model does not capture the power of "existential" or NP queries. Rather, we introduce the (straightforward quantum reformalization of) Stockmeyer's [22] existential query model:

▶ **Definition 2** (Existential query model). *An algorithm in the existential query model has access to the input string $x \in \{0,1\}^N$ via the following existential query gate:*

$$O_x^{\exists}\colon |z\rangle \mapsto (-1)^{\mathsf{overlap}(x,z)} |z\rangle \tag{1}$$

where $z \in \{0,1\}^N$ and $\mathsf{overlap}(x,z) = 1$ if there is an i such that $x_i = z_i = 1$ and 0 otherwise.

In this model, we show a matching lower bound for Theorem 1.

▶ **Theorem 3.** *Any quantum algorithm with existential query access to $x \in \{0,1\}^N$ that outputs a i with $x_i = 1$ with constant probability needs to make $\Omega(\log\log N) = \Omega(\log n)$ existential queries.*

Approximate counting. Recall that, classically, approximate counting requires $\Theta(\log n)$ NP queries. We next exploit the fact that the technique behind the proof of Theorem 1 (c.f. [18]) reveals a genuinely *quantum* link between search-to-decision reduction and approximate counting. This allows us to show the following tight black-box lower bound on *quantum* algorithms in the existential query model:

▶ **Corollary 4.** *Any quantum algorithm with existential query access to a string $x \in \{0,1\}^N$ which outputs an estimate c such that $2^{|x|-1} \leq c < 2^{|x|}$, where $|x|$ is the Hamming weight of x, requires at least $\Omega(\log\log N)$ queries to the oracle.*

Above, x denotes the truth table of the SAT formula, and so $N = 2^n$ for n the number of variables. Thus, for approximate counting, quantum algorithms do not outperform classical algorithms.

Finally, we prove a tight *white-box* lower bound for *both* classical or quantum algorithms which approximately count using $o(\log n)$ NP queries. As far as we are aware, no white-box lower bounds existed for either setting prior to this work.

▶ **Corollary 5.** *If there exists a* classical *randomized poly-time algorithm for approximate counting, making $o(\log n)$ NP queries, then $\mathsf{BPP}^{\mathsf{NP}[o(n)]}$ contains a P^{NP}-complete problem* [3]*. Similarly, if there is a poly-time* quantum *algorithm for approximate counting making $o(\log n)$ NP queries, then $\mathsf{FBQP}^{\mathsf{NP}[o(n)]}$ contains an $\mathsf{FP}^{\mathsf{NP}}$-complete problem.*

While the complexity theoretic implications above are not as standard as P = NP or the collapse of PH, they nevertheless would arguably be striking if true. This is because an $\mathsf{FP}^{\mathsf{NP}}$-complete problem is finding a satisfying assignment of *smallest lexicographical ordering* [21]. Thus, using $o(n)$ queries would seem to require resolving the lex-ordering in sublogarithmic time (in the search space size), whereas classical and quantum algorithms for the closely related task of binary search cannot achieve sublogarithmic time [5].

[2] If one could show that any FBQP machine requires at least 1 NP query for search-to-decision, then NP $\not\subseteq$ BQP. This is because if NP $\subseteq$ BQP, then FNP $\subseteq$ FBQP via the standard search-to-decision reduction for SAT.

[3] Note that this is not quite as strong as $\mathsf{P}^{\mathsf{NP}} \subseteq \mathsf{BPP}^{\mathsf{NP}[o(n)]}$ as overheads in the reduction to the P^{NP}-complete problem may erase the reduction in the number of queries.

Proof techniques

We now sketch our proof techniques, organized by topic.

Search-to-decision. Theorem 1 follows rather straightforwardly from prior results. We first note that, quantumly, the solution of a formula with a unique satisfying assignment can be found with a single NP query using the Bernstein Vazirani algorithm (c.f. [18]). Therefore, it remains to reduce an arbitrary formula to a uniquely satisfiable one. Valiant and Vazirani showed this can be done with probability $O\left(\frac{1}{n}\right)$ [23]. If, however, the approximate cardinality of the set of solutions is known, then this reduction succeeds with constant probability. Since approximately counting this cardinality can be done with $\log(n)$ NP queries [22, 13] classically, this gives a quantum algorithm for search-to-decision reduction using $O(\log n)$ queries and with *constant* success probability. The success probability can now be boosted to any constant by running the algorithm a constant number of times, checking the outputs and outputting one of the satisfying assignments.

Next, we discuss the first of our main results, the black-box lower bound (Theorem 3). Here the proof requires more work. Most quantum query lower bounds fall in one of two groups: polynomial methods and adversary methods [8, 10, 6]. Unfortunately, these methods are tailored to the standard query model, and it is not clear how to effectively utilize them in our *existential* query model. Another complicating factor for us is that search-to-decision is a *relational* problem, not a function problem. That is, for a given input formula φ there are *multiple* correct outputs: all solutions of φ.

To overcome this, we instead give a reduction from "(unstructured) search with existential queries" to "binary search with standard queries", so we may invoke Ambainis' binary search lower bound [5]. We show that an algorithm for search on strings of length $N = 2^n$ using q existential queries induces an algorithm for binary search on a space of size n with the same number of *standard* queries. The basic idea for this is as follows. If we can find a solution, then we can also sample a random solution by randomly permuting the solution space. Furthermore, a binary search instance, which is the task of finding the index of the first 1 in a monotonically increasing binary string $x \in \{0,1\}^n$, can be modified in the following way. We make a new exponentially longer string $y \in \{0,1\}^N$ where the first 2^{n-1} entries of y are set to x_1, the next 2^{n-2} to x_2 and so on. The index of a uniformly random 1 in y corresponds to the index of the least 1 in x with probability $> \frac{1}{2}$. Therefore, transforming x into y and running the random solution sampling algorithm on it solves binary search on x using q existential queries. We now note that because x is monotonically increasing, any existential query can be simulated by a standard query. The results of an existential query with string z will be the same as simply querying the largest index i where $z_i = 1$. As the last step of the proof, we invoke Ambainis result that binary search on a space of size n takes $\Omega(\log n)$ queries to complete the proof [5].

Approximate counting. Our black-box lower bound (Corollary 4) follows by combining the proof of Theorem 1 with Theorem 3. If there is an approximate counter making q existential queries, then an index with $x_i = 1$ can be found with constant success probability and $q + 1$ existential queries using the algorithm from Theorem 1. By Theorem 3 this is only possible if $q = \Omega(\log n)$.

Finally, we discuss our white-box lower bound (Corollary 5). We assume the existence of an approximate counter making $o(\log n)$ queries and show that we can, with $o(\log n)$ queries, find the lexicographically smallest solution of a formula φ, which is an $\mathsf{FP}^{\mathsf{NP}}$-complete problem [21]. The main idea is as follows. The algorithm from Theorem 1 samples approximately from

the uniform distribution on the set of solutions of φ. We run this algorithm on the AND of n^2 copies of φ, where we pick new sets of variables for each instance. This will give us n^2 solutions of φ picked almost uniformly at random. We find the least x_{min} and repeat the process on the formula $\varphi(x) \wedge (x \leq x_{min})$. After every round the number of solutions will be divided by at least n with high probability. Therefore, after $O(\log_n(|\operatorname{sol}(\varphi)|)) = O\left(\frac{n}{\log n}\right)$ rounds we will have found the lexicographically smallest solution of φ. As every round takes $o(\log n)$ queries, we have found the lexicographically smallest solution using $o(n)$ queries, completing the proof.

Related work

As previously mentioned, Irani, Natarajan, Nirkhe, Rao and Yuen already independently showed that the Bernstein-Vazirani algorithm can be used to find the solution of a *uniquely* satisfiable formula (our Lemma 17) [18]. They combine this with the Valiant-Vazirani theorem to do search-to-decision reduction for QCMA (and NP) with a single query and *inverse polynomial* success probability. However, they do not further study the case of a *constant* success probability as done here. They also show that there exists a quantum polynomial time algorithm that makes a single query to a PP oracle and generates a witness for a QMA problem up to polynomial precision. Additionally, they show that there is an oracle such that QMA search does not reduce to QMA decision relative to that oracle.

In their work [12], Buhrman and van Dam study the difference between classical and quantum access to an NP-oracle. They show that an EQP machine, that is, a quantum computer that is not allowed to err, can save on the number of queries compared to classical. Among other results they prove the inclusions $\mathsf{P}^{\|\,\mathsf{NP}[2k]} \subseteq \mathsf{EQP}^{\|\,\mathsf{NP}[k]}$ and $\mathsf{FP}^{\|NP} \subseteq \mathsf{FEQP}^{\mathsf{NP}(O(\log n))}$. Note that while $\mathsf{P}^{\|NP} = \mathsf{P}^{\mathsf{NP}(O(\log n))}$, a similar equality for FP would collapse the polynomial hierarchy.

Search-to-decision reduction has been studied in other settings. If only parallel (i.e. non-adaptive) queries to the NP oracle are allowed, then the standard $O(n)$-query search-to-decision reduction for NP does not work. Nevertheless, it has been shown that $O(n^2)$ parallel oracle queries suffice for classical randomized algorithms [9]. Kawachi, Rossman and Watanabe showed that this is optimal in a black-box model and give an algorithm with improved error tolerance [19]. In a later work they also consider more general black-box models and show that $O(n^2)$ parallel classical queries are still needed [20].

The class BQP with access to various resources has been studied before. Aaronson, Ingram and Kretschmer [4] study oracle separations between various complexity classes involving BQP as an oracle or BQP with access to an oracle. Among other results the authors prove that there is an oracle relative to which $\mathsf{BQP}^{\mathsf{NP}} \not\subseteq \mathsf{PH}^{\mathsf{BQP}}$ and an oracle relative to which $\mathsf{NP}^{\mathsf{BQP}} \not\subseteq \mathsf{BQP}^{\mathsf{PH}}$. Aaronson, Buhrman and Kretschmer [3] investigate BQP when given various types of advice. There it is shown, among other results, that $\mathsf{FBQP/qpoly} \neq \mathsf{FBQP/poly}$ (not relative to an oracle!).

Isolation algorithms, i.e., algorithms reducing the number of solutions of a Boolean formula to 1, have been studied by Dell, Kabanets, van Melkebeek and Watanabe [15]. They show that, unless $\mathsf{NP} \subseteq \mathsf{P/poly}$, no randomized polynomial time isolation algorithm with success probability better than $\frac{2}{3}$ can exist.

Discussion and open questions

Our paper characterizes the quantum NP-query complexity of search-to-decision reductions and approximate counting (and additionally gives a white-box lower bound for classical approximate counting algorithms). For this, some of our results utilized a quantum reformulation of Stockmeyer's classical existential query model. Can quantum query lower bound

methods like polynomial methods and adversary methods be adapted to apply directly to existential queries? An obstacle here is the fact that, for example, adversary methods often keep track of how a "progress measure" increases with each query made. Existential queries, however, seem to allow arbitrarily large jumps in such "progress measures".

Second, we prove that $\mathsf{BPP}^{\mathsf{NP}[o(n)]}$ and $\mathsf{FBQP}^{\mathsf{NP}[o(n)]}$ containing P^{NP}- and $\mathsf{FP}^{\mathsf{NP}}$-complete problems, respectively, are consequences to very efficient approximate counting. It would be interesting to see if there are further consequences to these conclusions or if our results can be strengthened. For example, can our results be strengthened to a contradiction of a common complexity theoretic hypothesis such as the (strong) Exponential Time Hypothesis? Third, what other tasks might a $\mathsf{BQP}^{\mathsf{NP}}$ or $\mathsf{FBQP}^{\mathsf{NP}}$ machine be good for? Finally, we close with a simple-to-state, concrete open question, which captures much of the difficulty of working with $\mathsf{BQP}^{\mathsf{NP}}$: Let φ be a SAT formula. Classically, it is easy to see that a solution to φ cannot be produced by an FBPP machine with a single NP query, for this would imply $\mathsf{BPP} = \mathsf{NP}$. This is because one can simply plug each possible answer, 0 or 1, from the NP machine into the FBPP machine, and check if the string x produced by the latter satisfies φ. Unfortunately, this approach completely breaks down for an FBQP machine making a single NP query, since the query may involve exponentially many inputs in superposition! Can one nevertheless show that $\mathsf{FNP} \subseteq \mathsf{FBQP}^{\mathsf{NP}[1]}$ implies $\mathsf{NP} \subseteq \mathsf{BQP}$?

Organization

In Section 2 we cover definitions and prior results used in this article. In Section 3, we will give the proof of Theorem 1, the upper bound on search-to-decision reduction for NP with quantum access to the oracle. Following that, in Section 4 we give the proofs of our results in the existential query model, Theorem 3 and Corollary 5. Finally, we prove Theorem 5 in Section 5.

2 Preliminaries

2.1 Notation

Throughout the paper we use n for the number of variables of the formulae and $N = 2^n$ for the size of their truth tables. We write $\mathrm{sol}(\varphi)$ for the set $\{x \in \{0,1\}^n : \varphi(x)\}$ of solutions of the Boolean formula φ. If $x \in \{0,1\}^*$ is a binary string we write $|x|$ for its Hamming weight.

2.2 Function classes

We briefly recall the definition the relevant function classes. Contrary to what the name suggests, function classes are actually classes of relations. We will require all relations R in these classes to be *p-bounded*.

▶ **Definition 6.** *A relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ is called p-bounded if there is some polynomial p such that, for each x, if $\exists y.R(x,y)$, then $\exists z$ such that $\mathrm{len}(z) \leq p(\mathrm{len}(x))$ and $R(x,z)$. Here $\mathrm{len}(x)$ is the length of the string x.*

▶ **Definition 7.** FP *is the class of polynomial time computable*[4] *p-bounded relations $R \subseteq \{0,1\}^* \times \{0,1\}^*$ such that there is a deterministic poly-time algorithm that on input x does the following:*

1. *If $\exists y$ such that $R(x,y)$ then the algorithm outputs one such y*
2. *If $\forall y.(x,y) \notin R$ then the algorithm outputs $\perp$.*

[4] With a poly-time computable relation we mean that there is a poly-time algorithm for evaluating $R(x,y)$ when given x and y as inputs.

▶ **Definition 8.** *The class* FNP *consists of all poly-time computable p-bounded relations* $R \subseteq \{0,1\}^* \times \{0,1\}^*$.

To see the similarities with the definition of FP we note that this condition implies the existence of a poly-time non-deterministic algorithm for computing R in the following sense. The algorithm takes as input a string z, and each branch of the (non-deterministic) computation outputs either a string z or $\perp$ and satisfies the following properties:

1. If $\exists y$ such that $R(x,y)$, then all branches either output $\perp$ or a string z such that $R(x,z)$ (not necessarily the same one). Furthermore, at least one branch does not output $\perp$.
2. If $\forall y\ (x,y) \notin R$, then all branches output $\perp$.

An FNP-complete problem is FunctionSAT. It is the relation $R(\varphi, x)$ where φ is a (binary encoding of) a Boolean formula and $(\varphi, x) \in R$ iff x is a satisfying assignment of φ. FunctionSAT is FNP-complete for the same reasons that SAT is NP-complete.

For the definition of FBQP we follow Aaronson [1].

▶ **Definition 9.** FBQP *is the class of p-bounded relations* $R \subseteq \{0,1\}^* \times \{0,1\}^*$ *for which are computable by a quantum algorithm in the following sense. There exists a poly-time quantum algorithm that takes as input* x *and* $0^{1/\epsilon}$ *and outputs a* y*. This is such that* $R(x,y)$ *with probability at least* $1-\epsilon$ *(assuming a* y *with* $R(x,y)$ *exists). If* $\forall y\ (x,y) \notin R$ *then it outputs* $\perp$ *with probability at least* $1-\epsilon$.

2.3 Witness isolation

We consider algorithms that reduce an arbitrary formula to a uniquely satisfying one.

▶ **Definition 10** (Isolation algorithm)**.** *An isolation algorithm with success probability* p *is a randomized algorithm that maps a Boolean formula* φ *on* n *variables to a formula* u *on the same variables such that the formula* $\varphi \wedge u$ *has a unique solution with probability at least* p.

A celebrated result by Valiant and Vazirani states that isolation algorithms exist.

▶ **Theorem 11** (Valiant Vazirani Theorem ([23]))**.** *There exists an isolation algorithm with success probability* $\frac{1}{O(n)}$.

The main idea of Valiant and Vazirani is to use cleverly chosen pairwise independent hash function to reduce the size of the solutions space.

▶ **Definition 12** (Pairwise independent hash functions ([7, Definition 8.14]))**.** *A collection* $\mathcal{H}_{n,k}$ *of functions from* $\{0,1\}^n$ *to* $\{0,1\}^k$ *is a collection of pairwise independent hash functions if for every* $x \neq x' \in \{0,1\}^n$ *and* $y, y' \in \{0,1\}^k$ *we have*

$$\Pr[h(x) = y \wedge h(x') = y'] = 2^{-2k} \tag{2}$$

where the probability is over h *being drawn uniformly at random from* $\mathcal{H}_{n,k}$.

The proof of Valiant and Vazirani's theorem follows from the following lemma which will also be of independent interest for us.

▶ **Lemma 13** ([7, Lemma 17.19])**.** *Let* $\mathcal{H}_{n,k}$ *be a collection of pairwise-independent hash functions from* $\{0,1\}^n$ *to* $\{0,1\}^k$ *and suppose that* $\mathrm{sol}(\varphi) \subseteq \{0,1\}^n$ *is such that* $2^{k-2} \leq |\mathrm{sol}(\varphi)| \leq 2^{k-1}$. *Then*

$$\Pr_{h \sim \mathcal{H}_{n,k}} \left[\left| \left\{ x \in \mathrm{sol}(\varphi) : h(x) = 0^k \right\} \right| = 1 \right] \geq \frac{1}{8}. \tag{3}$$

It follows that witness isolation can be performed with constant success probability if the approximate size of the set of solutions is known.

We will also consider witness isolation algorithms with the added requirement that all solutions of φ will be the unique solution of $\varphi \wedge u$ with approximately equal probability. We call such algorithms almost-uniform isolation algorithms.

▶ **Definition 14.** *An ϵ-almost-uniform isolation algorithm A_{iso} with success probability p takes as input a Boolean formula φ, and efficiently produces a Boolean formula u on the same variables such that:*

- *p-Completeness: if $\varphi \in \mathrm{SAT}$, then, with probability at least p, $\varphi(x) \wedge u(x)$ has a unique satisfying assignment. In this case we say that the isolation succeeds.*
- *ϵ-almost-uniformity: for all $x \in \mathrm{sol}(\varphi)$ we have:*

$$\Pr\left[\varphi(x) \wedge u(x) \middle| |\mathrm{sol}(\varphi(x) \wedge u(x))| = 1\right] \leq \frac{1+\epsilon}{|\mathrm{sol}(\varphi)|}.$$

Note that we do not require a lower bound on this probability.

2.4 Approximate counting

Stockmeyer was the first to realize that an NP oracle can be used for approximate counting. It was shown by Chakraborty, Meel and Vardi that a logarithmic number of NP queries suffice: [22, 13].

▶ **Theorem 15** (Approximate counting, [22, 13])**.** *Given a formula φ on n variables and parameters $\delta, \epsilon > 0$, there exists a randomized poly-time algorithm, making $O\left(\frac{\log n \log(1/\delta)}{\epsilon^2}\right)$ queries to an NP-oracle, that outputs a value c such that:*

$$\Pr\left(\frac{|\mathrm{sol}(\varphi)|}{1+\epsilon} \leq c \leq (1+\epsilon)|\mathrm{sol}(\varphi)|\right) \geq 1-\delta. \tag{4}$$

2.5 Query complexity

In query complexity one studies how often an algorithm needs to query an input string $x \in \{0,1\}^N$ in order to compute some function of x. In this paper we will consider three different types of oracles: standard oracles, succinct existential oracles and non-succinct existential oracles.

With the *standard oracle model* we will refer to the oracle model that is usually used in quantum query complexity. In this model, the queries give access to a string $x \in \{0,1\}^N$ using the following query gate:

$$O_x \colon |i\rangle \mapsto (-1)^{x_i} |i\rangle . \tag{5}$$

We will call an application of the oracle *classical* if it is applied to a computational basis state.

In this paper, the string x will usually be the truth table of a hidden formula φ (i.e. $x_i = 1 \iff \varphi(i)$). Then, querying the oracle on index i corresponds to computing $\varphi(i)$.

Standard queries do not satisfactorily capture the power of NP queries. For example, it only takes one NP query to determine if a formula is satisfiable (i.e. if its truth table is not all 0s), but it is well known that determining if there is an i with $x_i = 1$ takes $\Theta(\sqrt{N})$ standard queries ([16, 10]). Therefore, we will also consider other query models that better capture the power of NP queries.

The first of these query models we will call the *succinct existential query model*. Here, the oracle hides a formula φ. A query consists of a different poly-size formula ψ and the result to this query will be whether or not $\varphi \wedge \psi$ is satisfiable. Specifically, the oracle can be queried using the query gate

$$O_\varphi^{\mathsf{NP}} \colon |\psi\rangle \mapsto (-1)^{\mathsf{sat}(\varphi \wedge \psi)} |\psi\rangle \tag{6}$$

where $\mathsf{sat}(\varphi \wedge \psi)$ is 1 if $\varphi \wedge \psi$ is satisfiable and 0 otherwise. The succinct $\exists$-query model captures the power of an actual NP oracle well. It is strong enough to be used for the most common, if not all, well-known algorithms computing properties of Boolean formulae, such as finding the lexicographically-least solution of the hidden formula φ (a P^{NP}-complete problem) and approximately count the number of solutions of φ (e.g. with [13]).

We will also consider a non-succinct version of the existential query model, which we will also simply call the *existential query model*. Essentially this model is a reformulation of a model originally introduced by Stockmeyer [22] and used in e.g. [19]. We restated it in a manner closer to the standard query model. Existential queries ($\exists$-queries) are of the form

$$O_x^{\exists} \colon |z\rangle \mapsto (-1)^{\mathsf{overlap}(x,z)} |z\rangle \,, \tag{7}$$

where $\mathsf{overlap} \colon \{0,1\}^N \times \{0,1\}^N \to \{0,1\}$ is the function

$$\mathsf{overlap}(x,z) = \begin{cases} 1 & \text{if } \exists i, x_i = z_i = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

Again, the hidden string will usually be the truth table of a formula φ. At first glance, this query model may look rather useless. The query register has size exponential in n (the number of variables of φ). Therefore, even performing a single query will take exponential time. However, *if* one is only concerned with the number of queries, and not with other resources such as time and space, then non-succinct existential queries are more powerful than succinct ones. Instead of making a succinct $\exists$-query with formula ψ, an algorithm can make a non-succinct existential query with z the truth table of ψ. Furthermore, not all truth tables correspond to poly-size formulae. In this paper we will prove lower bounds on the number of existential queries an algorithm needs to make. The result will allow the algorithms unbounded time and space and hence these lower bounds will in particular hold for efficient algorithms making succinct $\exists$-queries.

We are interested in the number of these queries needed to find a solution of φ. In the existential query model this corresponds to solving the search problem on the truth table x of ϕ, that is, outputting an index i such that $x_i = 1$. It should be noted that, unlike in the case of standard queries, the existential query complexity of this function search problem is not necessarily the same as that of the decision search problem (i.e. determining if there is such an index). For example, an information theoretic argument shows that classically $\Theta(n)$ existential queries are needed for function search, but a single existential query with $z = 1^N$ solves decision search.[5]

We will also be interested in a slight variation of the search problem which we call the *index sampling* problem. Here the task is to sample according to the uniform distribution on the support of x. Formally, we define it as

[5] The string 1^N has overlap with any non-zero string. Hence it has overlap with the truth table of φ iff φ is satisfiable.

▶ **Definition 16** (Index sampling). *An algorithm solves the index sampling problem if it, for all $x \in \{0,1\}^N \setminus \{0^N\}$, outputs $s \in [N] \cup \{\perp\}$ such that:*

- *for all $i \in [N]$ with $x_i = 1$, $\Pr[s = i | s \neq \perp] = \frac{1}{|x|}$,*
- *there is a constant c such that $\Pr[s \neq \perp] \geq c$.*

3 Quantum algorithm for search-to-decision reduction

We are now ready to prove our results. For pedagogical reasons we start with the proof of the upper bound (Theorem 1) before proving our main results: the lower bounds in Theorem 3 and Theorem 5.

▶ **Theorem 1.** $\mathsf{FNP} \subseteq \mathsf{FBQP}^{\mathsf{NP}[\log]}$. *Furthermore, all queries made to the oracle are of the form $\varphi \wedge \chi$ where φ is the input formula and χ some other formula.*

Proof. We will show that there exists an $\mathsf{FBQP}^{\mathsf{NP}[\log]}$ algorithm that, when given a SAT instance φ, outputs a satisfying assignment $x \in \{0,1\}^n$ of φ if one exists, and outputs "no solution" otherwise. The algorithm succeeds with constant probability. This success probability can be boosted by running the algorithm a constant number of times, checking for each output if it is indeed a satisfying assignment and then outputting one that is. Hence, the success probability can be taken to be any arbitrary constant.

The existence of a satisfying assignment can be checked using a single query to the NP-oracle. Therefore, we will restrict our attention to the case where a satisfying assignment exists. We will proceed in two steps. First, we show how the satisfying assignment of a formula with *exactly* one satisfying assignment can be found with only a single query to the NP-oracle using the Bernstein Vazirani algorithm. Then we show how we can use $O(\log n)$ queries to reduce any formula to a uniquely satisfying one with constant probability.

▶ **Lemma 17** ("Bernstein-Vazirani (BV) trick" [18]). *Let φ be a formula with exactly one solution. There exists a* BQP *algorithm that makes a single query to an* NP*-oracle and finds this unique solution with probability 1.*

Proof of BV trick. Let the unique solution of φ be denoted by s and consider the formula

$$\psi_a \coloneqq \varphi(x) \wedge (x \cdot a = 1), \tag{9}$$

where $a \in \{0,1\}^n$ and $x \cdot a$ denotes the inner product of the two binary strings x and a modulo 2. (Note that ψ_a is of the form $\varphi \wedge \chi$.) We now have that ψ_a is satisfiable (i.e. there is a y such that $\varphi(y) \wedge y \cdot a = 1$) if and only if[6] $a \cdot s = 1$ because s is, by assumption, the only solution of φ. Now we run the Bernstein-Vazirani algorithm, where to evaluate $a \cdot s$ we ask the NP machine whether ψ_a is true. That is, we start with the state $|0^n\rangle$ and apply $H^{\otimes n}$ to get the uniform superposition on n qubits. The next step is to query the oracle on input ψ_a where a is in a uniform superposition:

$$\frac{1}{\sqrt{2^n}} \sum_{a \in \{0,1\}^n} |a\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{a \in \{0,1\}^n} (-1)^{a \cdot s} |a\rangle . \tag{10}$$

Now another application of $H^{\otimes n}$ gives:

$$\frac{1}{2^n} \sum_{a,y \in \{0,1\}^n} (-1)^{a \cdot s + a \cdot y} |y\rangle = \frac{1}{2^n} \sum_{a,y \in \{0,1\}^n} (-1)^{a \cdot (s \oplus y)} |y\rangle = |s\rangle . \tag{11}$$

Hence measuring the final state in the computational basis gives the unique satisfying assignment s of φ. ◁

[6] Note this does not imply $a = s$.

To deal with an arbitrary number of solutions, we first use Theorem 15 and $O(\log n)$ queries to the NP oracle to find k such that $2^{k-2} \leq |\mathrm{sol}(\varphi)| < 2^{k-1}$. (All queries made by the approximate counting algorithm in [13] are of the form $\varphi \wedge \chi$.) Then, we invoke Lemma 13 to obtain $u(x) := h(x) = 0^k$ such that $\varphi \wedge u$ has a unique solution with probability $> \frac{1}{8}$. Applying the BV trick to $\varphi \wedge u$ then completes the proof. Furthermore, all queries made were of the claimed form. ◀

4 Lower bound for existential query complexity of search

We will prove that all quantum algorithms for the search problem need $\Omega(\log n)$ existential queries, even if we allow the algorithm to additionally make $\mathrm{poly}(n)$ classical standard queries. To do so, we will reduce binary search to this problem in order to use Ambainis' lower bound for binary search [5]. A binary search problem consists of a monotonic binary string $x = 00\ldots01\ldots1$ and the task is to find the index of the first 1.

▶ **Theorem 3** (Restated). *Any quantum algorithm with existential query access to $x \in \{0,1\}^N$ needs to make $\Omega(\log\log N) = \Omega(\log n)$ existential queries to find an i such that $x_i = 1$. This remains true even if the algorithm is allowed to make an additional $\mathrm{poly}(n)$ classical standard queries.*

The proof will follow from the following lemma.

▶ **Lemma 18.** *Consider the following statements:*

1. *There exists a quantum algorithm for search on strings of size $N = 2^n$ that makes q $\exists$-queries and $\mathrm{poly}(n)$ classical standard queries and succeeds with constant probability.*
2. *There exists a quantum algorithm for search on strings of size N using $q + O(\log\log n)$ $\exists$-queries and no additional standard queries which succeeds with constant probability.*
3. *There exists a quantum algorithm for index sampling on strings of size N using $q + O(\log\log n)$ $\exists$-queries and no additional standard queries which succeeds with constant probability.*
4. *There exists a quantum algorithm for binary search on strings of size n using $q+O(\log\log n)$ $\exists$-queries and no additional standard queries. The algorithm succeeds with constant probability.*
5. *There exists a quantum algorithm for binary search on strings of size n using $q+O(\log\log n)$ standard quantum queries and no $\exists$-queries. The algorithm succeeds with constant probability.*

Then, $1 \implies 2 \implies 3 \implies 4 \implies 5$.

It is worth noting that all success probabilities can be boosted to be bigger than any constant $c < 1$.

Proof. $1 \implies 2$: consider an algorithm for search using q $\exists$-queries and $\mathrm{poly}(n)$ classical standard queries. We will modify the algorithm to get rid of the standard queries. Our new modified algorithm will act exactly the same as the original algorithm, except it does not actually perform the classical standard queries. Instead, it assumes the answer to those queries is 0 and keeps track of the positions that should have been queried in a set A. At the end of the algorithm, it checks if its assumptions were correct using an $\exists$-query. That is, it performs an $\exists$-query with string z defined by $z_i = 1$ iff the i-th index should have been queried by a classical standard query at some point.

Now there are now two options. Either the result of this $\exists$-query is 0, in which case the assumptions that all indices queried by classical standard queries were 0 is correct, and hence the modified algorithm and the original algorithm coincide. Alternatively, the result of

the ∃-query is 1. Then at least one of the assumptions was wrong. But now the algorithm has determined that the set $A \subseteq [N]$ contains some $i \in A$ with $x_i = 1$. Furthermore, A contains at most $\mathrm{poly}(n)$ elements since only $\mathrm{poly}(n)$ classical queries were made. The search algorithm from Theorem 1 makes only (succinct) existential queries, so we can now use it to search, within A, for an i with $x_i = 1$.[7] Because we know that $|A| = \mathrm{poly}(n)$, approximately counting the number of solutions *within* A will take $O(\log\log|A|) = O(\log\log n)$ ∃-queries using the algorithm from [13].[8] Therefore, finding a solution within A given that there is one will take $O(\log\log n)$ existential queries.

2 $\implies$ 3: consider a permutation $\sigma \in S_N$ mapping $[N]$ to itself drawn uniformly at random. With slight abuse of notation we define $\sigma(x)$ by $\sigma(x)_i = x_{\sigma(i)}$. Our algorithm for index sampling will apply the algorithm for search to $\sigma(x)$, undo the permutation, check if it is indeed a solution and output the result if it is, and abort (i.e. output $\perp$) if it is not. The probability of aborting is exactly the failure probability of the search algorithm. In the following we condition on the sampling algorithm not aborting and assume $x \neq 0^N$ (this case can be checked with 1 ∃-query).

Denote by $\mathsf{search}(x)$ the output of the search algorithm on input x. Consider the probability $p(i) = \Pr[\sigma^{-1}(\mathsf{search}(\sigma(x))) = i] = \Pr[\mathsf{search}(\sigma(x)) = \sigma(i)]$. We will show that $p(i) = \frac{1}{|x|}$ if $x_i = 1$ and 0 otherwise. Note that there are two sources of randomness: the random choice of σ and potentially random behavior of the search algorithm. We can write:

$$p(i) = \sum_{y\in\{0,1\}^N} \sum_{k\in[N]} \Pr[\sigma(x) = y \wedge \sigma(i) = k \wedge \mathsf{search}(y) = k] \tag{12}$$

$$= \sum_{y,k} \Pr[\sigma(x) = y] \cdot \Pr[\sigma(i) = k | \sigma(x) = y] \cdot \Pr[\mathsf{search}(y) = k | \sigma(x) = y \wedge \sigma(i) = k]. \tag{13}$$

Because the algorithm for search does not depend on σ, we have that

$$p_y(k) \coloneqq \Pr[\mathsf{search}(y) = k | \sigma(x) = y \wedge \sigma(i) = k] = \Pr[\mathsf{search}(y) = k]. \tag{14}$$

The $p_y(k)$ are unknown, but because the algorithm solves the search problem we do know that $\Pr[\mathsf{search}(y) = k | y_k = 0] = 0$ and $\sum_{k:y_k=1} p_y(k) = 1$. Furthermore, we have

$$\Pr[\sigma(x) = y] = \begin{cases} 1/\binom{N}{|x|} & \text{if } |x| = |y| \\ 0 & \text{if } |x| \neq |y| \end{cases} \tag{15}$$

and

$$\Pr[\sigma(i) = k | \sigma(x) = y] = \begin{cases} \frac{1}{|x|} & \text{if } x_i = y_k = 1 \\ \frac{1}{N-|x|} & \text{if } x_i = y_k = 0 \\ 0 & \text{if } x_i \neq y_k. \end{cases} \tag{16}$$

We can now put everything together to get

$$p(i) = \sum_{y:|x|=|y|} \frac{1}{\binom{N}{|x|}} \sum_{k:y_k=1} \frac{1}{|x|} p_y(k) = \frac{1}{|x|} \tag{17}$$

[7] We can add $\wedge x \in A$ to all formulae to restrict the search to within A.

[8] Essentially, the algorithm from [13] uses binary search to find $k \in [n]$ such that $2^{k-1} < |\mathrm{sol}(\varphi)| < 2^k$. Because $|A| = O(\mathrm{poly}(n))$, it is already known that $|\mathrm{sol}(\varphi(x) \wedge x \in A)| \leq O(\mathrm{poly}(n)) = 2^{O(\log n)}$. Therefore, the binary search is sped up.

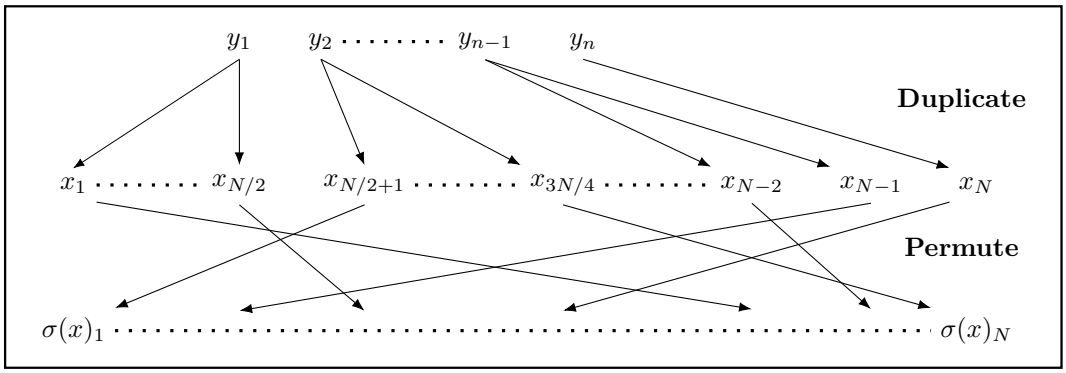


Figure 1 Modification of the binary-search oracle y in steps 2 $\implies$ 3 and 3 $\implies$ 4. Queries to $\sigma(x)$ are made by "going up the arrows".

if $x_i = 1$. On the other hand, if $x_i = 0$, we have that $p_y(k) = 0$ if $y_k = 0$ and that $\Pr[\sigma(i) = k|\sigma(x) = y] = 0$ if $y_k = 1$. Therefore, we have $p(i) = 0$ if $x_i = 0$.

3 $\implies$ 4: let $y \in \{0,1\}^n$ be a binary search instance. That is, $y = 00\ldots011\ldots1$ is monotonically increasing. For binary search, we want to find the smallest index i such that $y_i = 1$. We will now define another (exponentially longer) binary string $x \in \{0,1\}^N$ by $x_1 = x_2 = \cdots = x_{N/2} = y_1$, $x_{N/2+1} = \cdots = x_{3N/4} = y_2$ and so on, ending with $x_{N-2} = x_{N-1} = y_{\log N - 1}$ and $x_n = y_{\log N}$, i.e.

$$x = \underbrace{y_1 y_1 \ldots y_1}_{\frac{N}{2} \text{ times}} \underbrace{y_2 y_2 \ldots y_2}_{\frac{N}{4} \text{ times}} \cdots \underbrace{y_i y_i \ldots y_i}_{\frac{N}{2^i} \text{ times}} \cdots y_{\log N - 1} y_{\log N - 1} y_{\log N}. \tag{18}$$

If we sample an i with $x_i = 1$ uniformly at random, it will correspond to the smallest j with $y_j = 1$ with probability $> \frac{1}{2}$. This is because by construction each y_j appears more in x than all y_k for $k > j$ together. Furthermore, each query to y (standard or existential) can be simulated by a single query of the same kind to x (see Fig. 1). Therefore, statement 3. allows us to solve binary search with constant success probability and $q + O(\log\log n)$ $\exists$-queries.

4 $\implies$ 5: because the strings considered in a binary search problem are monotonically increasing we can simulate an $\exists$-query on such a string using only a single standard query. Instead of an $\exists$-query with z, we find the largest i such that $z_i = 1$ and use a standard query to query the i-th bit. The claimed implication follows. ◀

With this lemma in hand, Theorem 3 is easily proven.

Proof of Theorem 3. Suppose there exists an algorithm for search on $x \in \{0,1\}^N$ making $q = o(\log n)$ existential queries and poly(n) classical standard queries. Then, by Lemma 18 there is an algorithm for binary search on strings of length n making $q + O(\log\log n) = o(\log n)$ standard queries. This contradicts Ambainis' lower bound on binary search, which states that $\Omega(\log n)$ queries are required for binary search on size n strings [5]. Therefore, any search algorithm needs to make at least $q = \Omega \log n$ existential queries, even if it also makes poly(n) classical standard queries. ◀

Corollary 4 is an easy consequence of the previous theorem.

▶ **Corollary 4.** *Any quantum algorithm that is given existential query access to a string $x \in \{0,1\}^N$ and outputs an estimate c such that $2^{|x|-1} \leq c < 2^{|x|}$, where $|x|$ is the Hamming weight of x, needs to make at least $\Omega(\log\log N)$ queries to the oracle.*

Proof. We show that the existence of such an approximate counter making $o(\log\log N) = o(\log n)$ $\exists$-queries implies the existence of an algorithm for search making $o(\log n)$ $\exists$-queries. Using the approximate counter and Lemma 13 we can reduce x to be of Hamming weight 1 with constant probability and $o(\log n)$ $\exists$-queries. The index of the unique 1 can then be extracted using the BV trick (Lemma 17) and a single $\exists$-query. ◀

5 Conditional lower bound on number of NP queries for approximate counting

In order to prove Corollary 5 we will first concern ourselves with almost-uniform isolation algorithms. We will first prove the following theorem stating consequences of the existence of almost-uniform isolation algorithms making $o(\log n)$ NP queries. Thereafter, we will show that approximate counting with $o(\log n)$ NP queries implies almost-uniform isolation with $o(\log n)$ NP queries.

▶ **Theorem 19.** *Let $\epsilon > 0$ and $p \in (0, 1]$ be constants such that there exists an ϵ-almost-uniform isolation algorithm A_{iso} with success probability p making $Q(n) = o(\log n)$ queries to an* NP*-oracle. Then,* $\mathsf{BPP}^{\mathsf{NP}[o(n)]}$ *contains a* P^{NP}*-complete problem if this algorithm is classical and* $\mathsf{FBQP}^{\mathsf{NP}[o(n)]}$ *contains an* $\mathsf{FP}^{\mathsf{NP}}$*-complete problem if it is quantum.*

Proof. We will give a $\mathsf{BPP}^{\mathsf{NP}[o(n)]}$ algorithm to isolate the lexicographically least solution of an input formula φ, as outputting the last bit of this solution is P^{NP}-complete [21]. The idea of the algorithm will be as follows. We work in rounds. In the first round we sample n^2 almost-uniformly random solutions of φ. We will not explicitly know what these solutions are, but we can find a formula to which they are the unique solution. Next, we sample n^2 solutions of φ among all solutions that are lexicographically smaller than the all previously found solutions. We keep going like this until only one solution remains, which will be the lexicographically least solution of φ. We show that with high probability, the number of solutions that are smaller than the least solution found yet decreases by at least a factor $\frac{1}{n}$ every round. Therefore, we will, with high probability, need at most $\log_n(|sol(\varphi)|) = \frac{\log(|sol(\varphi)|)}{\log n} \leq \frac{n}{\log n}$ rounds to isolate the lexicographically least solution of φ.

In the first round we apply A_{iso} to $\Psi_1(\vec{x}_1, \ldots, \vec{x}_{n^2}) \coloneqq \varphi(\vec{x}_1) \wedge \cdots \wedge \varphi(\vec{x}_{n^2})$. Here the $\vec{x}_i$ denote fresh sets of variables and we use the vector notation to emphasize that they are n-bit strings and not bits. The result of this application will be $u_1(\vec{x}_1, \ldots, \vec{x}_{n^2})$ such that $\Psi_1 \wedge u_1$ has a unique solution. This unique solution will be the concatenation of n^2 solutions of φ.

In round $r+1$ we will do the following. From the previous round we have already constructed

$$\Psi_r(\vec{x}_{r,1}, \ldots, \vec{x}_{r,n^2}, \ldots, \vec{x}_{1,1}, \ldots, \vec{x}_{1,n^2})$$

with a unique solution. In this unique solution, $\vec{x}_{i,1}, \ldots, \vec{x}_{i,n^2}$ will be the solutions to φ picked in round i. We set

$$\begin{aligned}\chi_{r+1}(\vec{x}_{r+1,1}, \ldots, \vec{x}_{r+1,n^2}, \ldots, \vec{x}_{1,1}, \ldots, \vec{x}_{1,n^2}) \coloneqq\ & \varphi(\vec{x}_{r+1,1}) \wedge \cdots \wedge \varphi(\vec{x}_{r+1,n^2}) \\ & \wedge \Psi_r(\vec{x}_{r,1}, \ldots, \vec{x}_{r,n^2}, \ldots, \vec{x}_{1,1}, \ldots, \vec{x}_{1,n^2}) \\ & \wedge \vec{x}_{r+1,1} <_{lex} \vec{x}_{r,1} \wedge \cdots \wedge \vec{x}_{r+1,1} <_{lex} \vec{x}_{r,n^2} \\ & \wedge \vec{x}_{r+1,n^2} <_{lex} \vec{x}_{r,n^2} \wedge \cdots \wedge \vec{x}_{r+1,1} <_{lex} \vec{x}_{r,n^2}. \quad (19)\end{aligned}$$

In any satisfying assignment of χ_{r+1}, the first line of the RHS enforces that $\vec{x}_{r+1,1}, \ldots, \vec{x}_{r+1,n^2}$ are solutions of φ. The second line makes sure that $\vec{x}_{r,1}, \ldots, \vec{x}_{r,n^2}, \ldots, \vec{x}_{1,1}, \ldots, \vec{x}_{1,n^2}$ are set to the unique solutions picked in previous rounds. The third and fourth lines make sure that the new solutions are lexicographically strictly smaller than any solution picked in a previous round.[9]

We now pick the new solutions of round $r+1$ by applying A_{iso} to χ_{r+1}. We call the round is successful if A_{iso} succeeds, i.e. if its output u_{r+1} is such that $\chi_{r+1} \wedge u_{r+1}$ has a unique solution. We can check if A_{iso} succeeded by spending 2 NP queries.[10] In case of a success $\Psi_{r+1} \coloneqq \chi_{r+1} \wedge u_{r+1}$ will have a unique solution, which is picked almost uniformly at random from all solutions of χ_{r+1} (by definition of A_{iso}). In this unique solution $\vec{x}_{r+1,1}, \ldots, \vec{x}_{r+1,n^2}$ will be the newly picked solutions. By construction they will be smaller than the solutions found in the previous rounds. The previously picked solutions $\vec{x}_{r,1}, \ldots, \vec{x}_{r,n^2}, \ldots, \vec{x}_{1,1}, \ldots, \vec{x}_{1,n^2}$ will be the same as in previous rounds because the second line of Equation 19 has a unique solution. Finally, we check if there are still smaller solutions available by checking if

$$\varphi(y) \wedge \Psi_{r+1}(\vec{x}_{r+1,1}, \ldots, \vec{x}_{r+1,n^2}, \ldots, \vec{x}_{1,1}, \ldots, \vec{x}_{1,n^2}) \wedge y <_{lex} \vec{x}_{r+1,1} \wedge \cdots \wedge y <_{lex} \vec{x}_{r+1,n^2} \quad (20)$$

is still satisfiable. If it is satisfiable we proceed to the next round and if it is not then, in the unique solution of Ψ_{r+1}, one of the $\vec{x}_{r+1,i}$ will be the minimal solution of φ. Therefore,

$$\varphi(y) \wedge \Psi_{r+1}(\vec{x}_{r+1,1}, \ldots, \vec{x}_{r+1,n^2}, \ldots, \vec{x}_{1,1}, \ldots, \vec{x}_{1,n^2}) \wedge y \leq_{lex} \vec{x}_{r+1,1} \wedge \cdots \wedge y \leq_{lex} \vec{x}_{r+1,n^2} \quad (21)$$

will have a unique solution where y is the lexicographically least assignment of φ (note the use of $\leq$ instead of $<$). Asking the oracle if this formula is still satisfiable with the last bit of y set to 1 will then tell us the last bit of the lexicographically least solution of φ. Alternatively, a BQP-machine can use the BV trick to obtain the *entire* lexicographically least solution with one query and solve the $\mathsf{FP}^{\mathsf{NP}}$-complete problem of outputting the lexicographically least solution of φ [21].

We will proceed by proving that this algorithm succeeds with probability at least $\frac{2}{3}$ and makes at most $o(n)$ NP queries using the following claims. The proofs of these claims can be found in Appendix A.1.

▷ Claim 20 (Number of successful rounds needed). For sufficiently large n, the probability that the algorithm described above has not terminated after $\frac{n}{\log n}$ *successful* rounds is less than $\frac{1}{6}$ (a successful round is a round in which a unique solution remains after the application of A_{iso}).

▷ Claim 21 (Probability of successful rounds). The probability that after $\frac{2n}{p \log n}$ rounds there have not been $\frac{n}{\log n}$ *successful* rounds is at most $\frac{1}{6}$ for sufficiently large n. Here, recall p is the success probability of A_{iso}.

From these claims it follows that with probability at least $\frac{2}{3}$, the algorithm will, after at most $\frac{2n}{p \log n}$ rounds, have terminated. In every round, $Q(\mathrm{poly}(n))$ queries are made because only a polynomial amount of terms are added to Ψ_r every round (recall that $Q(n)$ is the number of queries made by A_{iso}). Since $Q(n) = o(\log n)$ by assumption, we have $Q(n^c) = o(c \cdot \log n) = o(\log n)$. Hence the algorithm makes at most $\frac{2n}{p \log n} o(\log n) = o(n)$ queries. ◀

[9] Note that comparing only to the previous rounds solutions suffices.

[10] One query is used to check if the formula is satisfiable and the other is used to check if there are two or more distinct solutions.

Corollary 5 follows by showing that an algorithm for approximate counting induces an algorithm for almost-uniform isolation with the same number of NP queries.

▶ **Corollary 5.** *If there exists a classical randomized poly-time algorithm for approximate counting making $o(\log n)$ NP queries, then $\mathsf{BPP}^{\mathsf{NP}[o(n)]}$ contains a P^{NP}-complete problem. Similarly, if there is a poly-time quantum algorithm for approximate counting making $o(\log n)$ NP queries, then $\mathsf{BQP}^{\mathsf{NP}[o(n)]}$ contains an $\mathsf{FP}^{\mathsf{NP}}$-complete problem.*

The idea of the proof is that we can make a almost-uniform isolation algorithm by first approximately counting the number of solutions (using $o(\log n)$ NP queries by assumption) and then picking a suitable hash function. Using ideas from Dellanoy and Meel [14, Lemma 3] it can be shown that this procedure indeed gives an almost-uniform isolation algorithm. The full proof of the corollary can be found in Appendix A.2.

References

1 Scott Aaronson. The equivalence of sampling and searching. *Theory of Computing Systems*, 55(2):281–298, 2014. doi:10.1007/s00224-013-9527-3.

2 Scott Aaronson and Alex Arkhipov. The computational complexity of linear optics. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 333–342, 2011. doi:10.1145/1993636.1993682.

3 Scott Aaronson, Harry Buhrman, and William Kretschmer. A qubit, a coin, and an advice string walk into a relational problem. *arXiv preprint arXiv:2302.10332*, 2023. doi:10.48550/arXiv.2302.10332.

4 Scott Aaronson, DeVon Ingram, and William Kretschmer. The Acrobatics of BQP. In Shachar Lovett, editor, *37th Computational Complexity Conference (CCC 2022)*, volume 234 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:17, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CCC.2022.20.

5 Andris Ambainis. A better lower bound for quantum algorithms searching an ordered list. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 352–357. IEEE, 1999. doi:10.1109/SFFCS.1999.814606.

6 Andris Ambainis. Quantum lower bounds by quantum arguments. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 636–643, 2000. doi:10.1145/335305.335394.

7 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

8 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM (JACM)*, 48(4):778–797, 2001. doi:10.1145/502090.502097.

9 Shai Ben-David, Benny Chor, and Oded Goldreich. On the theory of average case complexity. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 204–216, 1989. doi:10.1145/73007.73027.

10 Charles H Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM journal on Computing*, 26(5):1510–1523, 1997. doi:10.1137/S0097539796300933.

11 Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 11–20, 1993. doi:10.1145/167088.167097.

12 Harry Buhrman and Wim van Dam. Quantum bounded query complexity. In *Proceedings. Fourteenth Annual IEEE Conference on Computational Complexity (Formerly: Structure in Complexity Theory Conference)(Cat. No. 99CB36317)*, pages 149–156. IEEE, 1999. doi:10.1109/CCC.1999.766273.

13 Supratik Chakraborty, Kuldeep S Meel, and Moshe Y Vardi. Algorithmic improvements in approximate counting for probabilistic inference: from linear to logarithmic SAT calls. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 3569–3576, 2016. URL: `http://www.ijcai.org/Abstract/16/503`.

14 Remi Delannoy and Kuldeep S Meel. On almost-uniform generation of SAT solutions: The power of 3-wise independent hashing. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–10, 2022. `doi:10.1145/3531130.3533338`.

15 Holger Dell, Valentine Kabanets, Dieter van Melkebeek, and Osamu Watanabe. Is Valiant–Vazirani's isolation probability improvable? *computational complexity*, 22:345–383, 2013. `doi:10.1007/s00037-013-0059-7`.

16 Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996. `doi:10.1145/237814.237866`.

17 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. `doi:10.1006/jcss.2000.1727`.

18 Sandy Irani, Anand Natarajan, Chinmay Nirkhe, Sujit Rao, and Henry Yuen. Quantum Search-To-Decision Reductions and the State Synthesis Problem. In Shachar Lovett, editor, *37th Computational Complexity Conference (CCC 2022)*, volume 234 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.CCC.2022.5`.

19 Akinori Kawachi, Benjamin Rossman, and Osamu Watanabe. Query complexity and error tolerance of witness finding algorithms. In *Electron. Colloquium Comput. Complex.*, volume 19, page 2, 2012. URL: `https://eccc.weizmann.ac.il/report/2012/002`, `arXiv:TR12-002`.

20 Akinori Kawachi, Benjamin Rossman, and Osamu Watanabe. The query complexity of witness finding. *Theory of Computing Systems*, 61:305–321, 2017. `doi:10.1007/s00224-016-9708-y`.

21 Mark W Krentel. The complexity of optimization problems. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 69–76, 1986. `doi:10.1145/12130.12138`.

22 Larry Stockmeyer. The complexity of approximate counting. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 118–126, 1983. `doi:10.1145/800061.808740`.

23 Leslie G Valiant and Vijay V Vazirani. NP is as easy as detecting unique solutions. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 458–463, 1985. `doi:10.1145/22145.22196`.

A Omitted proofs

A.1 Claims in proof of Theorem 19

▷ Claim 20 (Number of successful rounds needed). For sufficiently large n, the probability that the algorithm described above has not terminated after $\frac{n}{\log n}$ *successful* rounds is less than $\frac{1}{6}$ (a successful round is a round in which a unique solution remains after the application of A_{iso}).

Proof of Claim 20. Let $\vec{y}_1 < \cdots < \vec{y}_k$ denote all solutions to φ smaller than $\vec{y}_{min,r-1}$, the smallest solution found in round $r-1$. Define $g = \lceil \frac{k}{n} \rceil$. We now compute the probability that, in a single round r, $\vec{y}_{min,r}$ is larger than $\vec{y}_g$ as

$$\Pr[\vec{y}_{min,r} >_{lex} \vec{y}_g] = \Pr\left[\vec{x}_{r,1} >_{lex} \vec{y}_g \wedge \cdots \wedge \vec{x}_{r,n^2} >_{lex} \vec{y}_g\right] \tag{22}$$

$$= \Pr[(\vec{x}_{r,1}, \ldots, \vec{x}_{r,n^2}) \in \{\vec{y}_{g+1}, \ldots, \vec{y}_k\}^{n^2}] \tag{23}$$

$$\leq |\{\vec{y}_{g+1}, \ldots, \vec{y}_k\}^{n^2}| \cdot \frac{1+\epsilon}{|\{\vec{y}_1, \ldots, \vec{y}_k\}^{n^2}|} \tag{24}$$

$$= (k-g)^{n^2} \frac{1+\epsilon}{k^{n^2}} \tag{25}$$

$$\leq \left(k\left(1-\frac{1}{n}\right)\right)^{n^2} \frac{1+\epsilon}{k^{n^2}} \tag{26}$$

$$= (1+\epsilon)\left(1-\frac{1}{n}\right)^{n^2}, \tag{27}$$

where Equation 25 follows from a union bound and the definition of an ϵ-almost-uniform isolation algorithm Hence, by a union bound, the probability that in at least one of $\frac{n}{\log n}$ successful rounds $\vec{y}_{min,r} >_{lex} \vec{y}_{goal,r}$, i.e., the probability that in at least on of the rounds the search space is not cut down by at least a factor $\frac{1}{n}$ is

$$\Pr[\exists r \leq \frac{n}{\log n} \text{ s.t. } \vec{y}_{min,r} >_{lex} \vec{y}_{g,r}] \leq (1+\epsilon)\frac{n}{\log n}\left(1-\frac{1}{n}\right)^{n^2}, \tag{28}$$

of which the right-hand side goes to 0 as n goes to ∞. The claim follows. ◁

▷ Claim 21 (Probability of successful rounds). The probability that after $\frac{2n}{p \log n}$ rounds there have not been $\frac{n}{\log n}$ *successful* rounds is at most $\frac{1}{6}$ for sufficiently large n. Here, recall p is the success probability of A_{iso}.

Proof of claim 21. Each round succeeds with probability p. After $\frac{2n}{p \log n}$ rounds the expected number of successful rounds is $\frac{2n}{\log n}$. By a Chernoff bound we have:

$$\Pr[\#\text{successes} < \frac{n}{\log n}] \leq \exp\left(-\frac{n}{4p \log n}\right). \tag{29}$$

For sufficiently large n, the right-hand side will indeed be at most $\frac{1}{6}$. ◁

A.2 Full proof of Corollary 19

▶ **Corollary 5.** *If there exists a classical randomized poly-time algorithm for approximate counting making $o(\log n)$ NP queries, then $\mathsf{BPP}^{\mathsf{NP}[o(n)]}$ contains a P^{NP}-complete problem. Similarly, if there is a poly-time quantum algorithm for approximate counting making $o(\log n)$ NP queries, then $\mathsf{BQP}^{\mathsf{NP}[o(n)]}$ contains an $\mathsf{FP}^{\mathsf{NP}}$-complete problem.*

Proof of Corollary 5. The approximate counting algorithms can be used to make an almost-uniform isolation algorithm as follows. First, use the approximate counting algorithm to find k such that $2^{k-2} \leq |\operatorname{sol}(\varphi)| \leq 2^{k-1}$. Next, choose a random hash function h from a set of pairwise independent hash functions from $\{0,1\}^n$ to $\{0,1\}^k$. By Lemma 13, the formula $\varphi(x) \wedge h(x) = 0^k$ will then have a unique solution with probability at least $\frac{1}{8}$.[11] We claim that in this case the unique solution will be distributed almost-uniformly at random among all solutions of φ. Our proof of this claim is based on work by Dellanoy and Meel [14, Lemma 3].

[11] We only require almost-uniformity in the case that there is a unique solution.

Before we prove the claim let us first introduce some notation. Let the random variable N denote the number of solutions of $\varphi(x) \wedge h(x) = 0^k$ and let SC denote the event that the approximate counting was successful, i.e. $2^{k-2} \leq |\operatorname{sol}(\varphi)| \leq 2^{k-1}$. We assume that SC occurs with probability $1 - \delta$. For any fixed x, we are interested in

$$\Pr[h(x) = 0^k | N = 1] = \frac{\Pr[h(x) = 0^k \wedge N = 1]}{\Pr[N = 1]} \tag{30}$$

$$= \frac{\Pr[h(x) = 0^k \wedge N = 1]}{\sum_{y \in \operatorname{sol}(\varphi)} \Pr[h(y) = 0^k \wedge N = 1]} \tag{31}$$

$$= \frac{\Pr[N = 1 | h(x) = 0^k] \Pr[h(x) = 0^k]}{\sum_{y \in \operatorname{sol}(\varphi)} \Pr[N = 1 | h(y) = 0^k] \Pr[h(y) = 0^k]} \tag{32}$$

$$= \frac{\Pr[N = 1 | h(x) = 0^k]}{\sum_{y \in \operatorname{sol}(\varphi)} \Pr[N = 1 | h(y) = 0^k]} \tag{33}$$

and want to show that this probability is $\leq \frac{\epsilon}{|\operatorname{sol}(\varphi)|}$. Let $\chi(y)$ be 1 if $h(y) = 0^k$ and 0 otherwise. For any $x \in \operatorname{sol}(\varphi)$ we have:

$$\mathbb{E}\left[N \middle| h(x) = 0^k \wedge SC\right] = \mathbb{E}\left[\sum_{y \in \operatorname{sol}(\varphi)} \chi(y) \middle| h(x) = 0^k \wedge SC\right] \tag{34}$$

$$= \sum_{y \in \operatorname{sol}(\varphi)} \mathbb{E}\left[\chi(y) \middle| h(x) = 0^k \wedge SC\right] \tag{35}$$

$$= 1 + \frac{|\operatorname{sol}(\varphi)| - 1}{2^{-k}} \tag{36}$$

where we use that h is a 2-wise independent hash function. By Markov's inequality this means that for all $x \in \operatorname{sol}(\varphi)$

$$\Pr[N = 1 | h(x) = 0^k \wedge SC] = 1 - \Pr[N \geq 2 | h(x) = 0^k \wedge SC] \tag{37}$$

$$\geq 1 - \frac{\mathbb{E}[N | h(x) = 0 \wedge SC]}{2} \tag{38}$$

$$\geq \frac{1}{2} - \frac{|\operatorname{sol}(\varphi)| - 1}{2^{k+1}} \tag{39}$$

$$\geq \frac{1}{4}. \tag{40}$$

Combining with Equation 33 and using that $\Pr[N = 1 | h(y) = 0^k] \geq \Pr[N = 1 \wedge SC | h(y) = 0^k]$ and $\Pr[N = 1 | h(x) = 0^k] \leq 1$ gives

$$\Pr[h(x) = 0^k | N = 1] \leq \frac{1}{\sum_{y \in \operatorname{sol}(\varphi)} \Pr[N = 1 \wedge SC | h(y) = 0^k]} \tag{41}$$

$$\leq \frac{1}{\sum_{y \in \operatorname{sol}(\varphi)} \Pr[N = 1 | h(y) = 0^k \wedge SC] \Pr[SC]} \tag{42}$$

$$\leq \frac{4}{(1 - \delta)|\operatorname{sol}(\varphi)|}. \tag{43}$$

Note that this holds for *any* x. Hence the existence of an approximate counter gives us an almost-uniform isolation algorithm with $p = \frac{1-\delta}{8}$ and $\epsilon = \frac{3+\delta}{1-\delta}$. Furthermore, the approximate counter and isolation algorithm will make the same amount of NP queries (i.e. $o(\log n)$). ◀

Low-Memory Algorithms for Online Edge Coloring

Prantar Ghosh
Georgetown University, Washington, DC, USA

Manuel Stoeckl
Dartmouth College, Hanover, NH, USA

Abstract

For edge coloring, the online and the W-streaming models seem somewhat orthogonal: the former needs edges to be assigned colors immediately after insertion, typically without any space restrictions, while the latter limits memory to be sublinear in the input size but allows an edge's color to be announced any time after its insertion. We aim for the best of both worlds by designing small-space online algorithms for edge coloring.

Our online algorithms significantly improve upon the memory used by prior ones while achieving an $O(1)$-competitive ratio. We study the problem under both (adversarial) edge arrivals and vertex arrivals. Under vertex arrivals of any n-node graph with maximum vertex-degree Δ, our online $O(\Delta)$-coloring algorithm uses only semi-streaming space (i.e., $\tilde{O}(n)$ space, where the $\tilde{O}(.)$ notation hides polylog(n) factors). Under edge arrivals, we obtain an online $O(\Delta)$-coloring in $\tilde{O}(n\sqrt{\Delta})$ space. We also achieve a smooth color-space tradeoff: for any $t = O(\Delta)$, we get an $O(\Delta t(\log^2 \Delta))$-coloring in $\tilde{O}(n\sqrt{\Delta/t})$ space, improving upon the state of the art that used $\tilde{O}(n\Delta/t)$ space for the same number of colors.

The improvements stem from extensive use of random permutations that enable us to avoid previously used colors. Most of our algorithms can be derandomized and extended to multigraphs, where edge coloring is known to be considerably harder than for simple graphs.

2012 ACM Subject Classification Mathematics of computing → Graph coloring; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Edge coloring, streaming model, online algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.71

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2304.12285

Funding *Prantar Ghosh*: Supported in part by NSF under award 1918989. Part of this work was done while the author was at DIMACS, Rutgers University, supported in part by a grant (820931) to DIMACS from the Simons Foundation.
Manuel Stoeckl: This work was supported in part by the National Science Foundation under award 2006589.

1 Introduction

A proper edge-coloring of a graph or a multigraph colors its edges such that no two adjacent edges share the same color. The goal is to use as few colors as possible. Any graph with maximum vertex-degree Δ trivially requires Δ colors to be properly edge-colored. Vizing's celebrated theorem [46] asserts that $\Delta + 1$ colors suffice for any simple graph.[1] Constructive polynomial-time algorithms exist for $(\Delta + 1)$-edge-coloring in the classical offline setting [39], and these are likely to be optimal with respect to the number of colors: distinguishing between whether the edge-chromatic number (i.e., the minimum number of colors needed to edge-color a graph) of a simple graph is Δ or $\Delta + 1$ is NP-hard [33].

[1] For multigraphs, $3\Delta/2$ colors are necessary and sufficient. [44]

The edge-coloring problem finds applications in various practical scenarios, including switch routing [2], round-robin tournament scheduling [34], call scheduling [27], optical networks [42], and link scheduling in sensor networks [30]. In applications like switch routing, where the underlying graph evolves through edge insertions, the color assignments must be made instantly and irrevocably. This is modeled by the *online* edge coloring problem. Due to its restrictions, an online algorithm cannot obtain a $(\Delta+1)$-coloring [11]. Nevertheless, the simple greedy algorithm, which colors every edge with the first available color not assigned to any neighbor, obtains a $(2\Delta-1)$-coloring since each edge can have at most $2\Delta-2$ neighboring edges. Thus, the competitive ratio achieved is $2-o(1)$ (since the optimum is Δ or $\Delta+1$). Bar-Noy, Motwani, and Naor [11] showed that no online algorithm can outperform this greedy algorithm. However, they proved this only for graphs with max-degree $\Delta = O(\log n)$. They conjectured that for $\Delta = \omega(\log n)$, it is possible to get better bounds – specifically, a $(1+o(1))\Delta$-coloring. For this regime of Δ, several works [2, 10, 23, 18, 43, 35, 41] have studied online edge coloring with the goal of surpassing the greedy algorithm and, even further, of resolving the said conjecture. Additionally, other variants of the online problem have been investigated [29, 38, 28]. However, all prior works assume full storage of all graph edges and their colors in memory.

With the ubiquity of big data in the modern world, the assumption of storing entire graphs in memory becomes impractical. Even graphs like communication and internet routing networks that motivate the study of edge coloring often turn out to be large-scale networks. This challenge has given rise to big-graph processing models such as *graph streaming* which, like the online model, sequentially accesses the graph edges, but only retains a small summary. There is an immediate barrier for edge coloring in this setting: reporting all edge colors at the end of the stream would use space linear in the number of edges. To remedy this, a natural extension of the model, called the *W-streaming model*, allows reporting the output in streaming fashion. Here, an algorithm with limited working memory stores information about both the input graph and the output coloring and periodically streams or announces edge colors. Unlike the online model, here we don't need to assign a color to an incoming edge right away, and can defer it. However, due to space constraints, we are not able to remember all the previously announced colors. Note that this makes even the greedy $(2\Delta-1)$-coloring algorithm hard, if not impossible, to implement in this model.

In this work, we aim to get the best of both worlds of the online and the streaming models by designing *low-memory* online algorithms for edge coloring. This addresses the need for immediate color assignment in modern scenarios while optimizing space usage. We achieve an $O(1)$-competitive ratio, i.e., a color bound of $O(\Delta)$. Note that no prior work in the sublinear-space setting has achieved an $O(\Delta)$-coloring W-streaming algorithm, let alone an online algorithm. For adversarial edge-arrival streams, we get an online $O(\Delta)$-coloring in $\widetilde{O}(n\sqrt{\Delta})$ space[2], significantly reducing the space compared to prior online algorithms with only a constant factor increase in colors. We can smoothly trade off space with colors, obtaining an $\widetilde{O}(\Delta t)$-coloring in $\widetilde{O}(n\sqrt{\Delta/t})$ space. This improves upon the state of the art [21, 5] which used $\widetilde{O}(n\Delta/t)$ space for the same color bound. Furthermore, for natural and well-studied settings of vertex-arrival in general graphs and one-sided vertex arrival in bipartite graphs, we enhance the space usage to $\widetilde{O}(n)$, the prevalent *semi-streaming* memory regime for graph streaming problems. Most of our algorithms generalize to multigraphs and can be made deterministic.

[2] Throughout the paper, the $\widetilde{O}(.)$ notation hides polylog(n) and polylog(Δ) factors.

We remark that apart from being interesting online algorithms on their own, our results can be also viewed as strengthening W-streaming algorithms with the guarantee of online output-reporting. In particular, they contribute an important conceptual message: the state-of-the-art W-streaming space bound for $O(\Delta)$ edge coloring can be matched *without* the exclusive power of the W-streaming model (allowing delayed assignment of edge colors); we can report edge colors online.

1.1 Our Results and Contributions

We study edge-coloring in the online model with sublinear (i.e., $o(n\Delta)$) memory, under (adversarial) edge-arrivals as well as vertex-arrivals. These results are summarized in Table 1, and their corollaries for the W-streaming model in Table 2. The tables mention the state of the art, for comparison.

Table 1 Our results in the online model. Here, $t \leq \Delta$ is any positive integer. Algorithms marked with a $\star$ require oracle randomness for randomized algorithms and an advice string computable in exponential time for deterministic.

Arrival	Algorithm	Colors	Space	Graph	Reference
Edge	Randomized	$\left(\frac{e}{e-1}+o(1)\right)\Delta$	$\widetilde{O}(n\Delta)$	Simple	[35]
	Randomized	$O(\Delta)$	$\widetilde{O}(n\sqrt{\Delta})$	Simple	Theorem 1
	Deterministic	$(2\Delta-1)t$	$O(n\Delta/t)$	Multigraph	[5]
	Deterministic	$\widetilde{O}(\Delta t)$	$\widetilde{O}(n\sqrt{\Delta/t})\star$	Multigraph	Theorem 3 + Lemma 5
Vertex	Randomized	$(1.9+o(1))\Delta$	$\widetilde{O}(n\Delta)$	Simple	[43]
	Randomized	$O(\Delta)$	$\widetilde{O}(n)\star$	Multigraph	Theorem 6
	Deterministic	$2\Delta-1$	$O(n\Delta)$	Multigraph	Greedy folklore
	Deterministic	$O(\Delta)$	$\widetilde{O}(n)\star$	Multigraph	Theorem 7
One sided vertex	Randomized	$(1+o(1))\Delta$	$\widetilde{O}(n\Delta)$	Simple	[23]
	Randomized	1.533Δ	$\widetilde{O}(n\Delta)$	Multigraph	[41]
	Randomized	5Δ	$\widetilde{O}(n)\star$	Multigraph	Theorem 6
	Deterministic	$2\Delta-1$	$O(n\Delta)$	Multigraph	Greedy folklore
	Deterministic	$O(\Delta)$	$\widetilde{O}(n)\star$	Multigraph	Theorem 7

Table 2 Our results in the W-streaming edge-arrival model. Here, $s \leq \Delta/2$ is any positive integer. Results marked with $\star$ require oracle randomness for randomized algorithms and an advice string computable in exponential time for deterministic.

Algorithm	Colors	Space	Graph	Reference
Randomized	$O(\Delta^2/s)$	$\widetilde{O}(ns)$	Simple	[21]
Randomized	$O(\Delta)$	$\widetilde{O}(n\sqrt{\Delta})$	Simple	Corollary 2
Deterministic	$(1-o(1))\Delta^2/s$	$O(ns)$	Simple	[5]
Deterministic	$\widetilde{O}(\Delta^2/s)$	$\widetilde{O}(n\sqrt{s})\star$	Multigraph	Corollary 4 + Lemma 5

Edge-arrival model

Here we design both online and W-streaming algorithms.

▶ **Theorem 1.** *Given any adversarial edge-arrival stream of a simple graph, there is a randomized algorithm for online $O(\Delta)$-edge-coloring using $\widetilde{O}(n\sqrt{\Delta})$ bits of space.*

Previously, there was no sublinear space online algorithm known for $O(\Delta)$-coloring. As observed in Table 1, all prior algorithms need $\Theta(n\Delta)$ space in the worst case to achieve a color bound of $O(\Delta)$.

Note that Theorem 1 immediately implies a randomized W-streaming algorithm with the same space and color bounds. Although immediate, we believe that it is important to note it as a corollary.

▶ **Corollary 2.** *Given an adversarially ordered edge stream of any simple graph, there is a randomized W-streaming algorithm for $O(\Delta)$-edge-coloring using $\widetilde{O}(n\sqrt{\Delta})$ bits of space.*

The above result improves upon the state of the art algorithms of [21, 5] which, as implied by Table 2, only obtain $\omega(\Delta)$-colorings for $o(n\Delta)$ space (the non-trivial memory regime in W-streaming). In fact, we improve upon them by a factor of $\Omega(\sqrt{\Delta})$ in space for $O(\Delta)$-coloring.

Further, we prove that we can make the above algorithms deterministic, and able to handle multigraphs, at the cost of only a polylogarithmic factor in the number of colors used. Once again, the online algorithm is also a W-streaming algorithm.

▶ **Theorem 3.** *Given an adversarially ordered edge-arrival stream of any multigraph, there is a deterministic algorithm for online $O(\Delta \log^2 \Delta)$-edge-coloring using $\widetilde{O}(n\sqrt{\Delta})$ bits of space.*

▶ **Corollary 4.** *Given an adversarially ordered edge stream of any multigraph, there is a deterministic W-streaming algorithm for $O(\Delta \log^2 \Delta)$-edge-coloring using $\widetilde{O}(n\sqrt{\Delta})$ bits of space.*

Furthermore, in each case, we can achieve a smooth tradeoff between the number of colors and the memory used. This is implied by a framework captured in the following lemma.

▶ **Lemma 5.** *Suppose that we are given an online $f(n, \Delta)$-space streaming algorithm $\mathcal{A}$ for $O(\Delta)$-coloring any n-node multigraph with max-degree Δ under adversarial edge arrivals. Then, for any $t \geq 1$, there is a online streaming algorithm $\mathcal{B}$ for $O(\Delta t)$-coloring the same kind of graphs under adversarial edge arrivals using $f(n/t, \Delta t) + \widetilde{O}(n)$ bits of space.*

For the online model, the above lemma combined with Theorem 3 immediately gives the tradeoff of $\widetilde{O}(\Delta t)$ colors and $\widetilde{O}(n\sqrt{\Delta/t})$ space for any $t = O(\Delta)$, as claimed in Table 1. In other words, combined with Corollary 4, it implies the W-streaming bounds of $\widetilde{O}(\Delta^2/s)$ colors and $O(n\sqrt{s})$ space for any $s = O(\Delta)$, as claimed in Table 2.[3] Note that our results match the tradeoff obtained by the state of the art for $t = \Theta(\Delta)$ and $s = O(1)$, and strictly improve upon them for $t = o(\Delta)$ and $s = \omega(1)$.

[3] We use the parameter t for online algorithms and s for W-streaming, where $t \cdot s = \Delta$, so that a reader can easily compare our bounds with the "ideal" bounds of $O(\Delta)$ colors and $\widetilde{O}(n)$ space in the respective models by smoothly growing t or s from 1 to Δ.

Vertex-arrival Model

We now turn to the weaker vertex-arrival model. The online edge-coloring problem has been widely studied in this setting as well (see Section 1.2 for a detailed discussion). Our online algorithms obtain significantly better space bounds than the edge-arrival setting.

▶ **Theorem 6.** *Given any adversarial vertex-arrival stream of a multigraph, there is a randomized online $O(\Delta)$-edge coloring algorithm using $\widetilde{O}(n)$ bits of space. It works even against an adaptive adversary and uses $\widetilde{O}(n\Delta)$ oracle random bits.*[4] *In particular, for one-sided bipartite vertex arrivals, there is an algorithm using 5Δ colors.*

Thus, at the cost of only a constant factor in the number of colors, we can improve the memory usage from $\widetilde{O}(n\Delta)$ to $\widetilde{O}(n)$ for vertex-arrival streams. Since this algorithm immediately implies a W-streaming algorithm with the same bounds, we see that for vertex-arrival streams, $O(\Delta)$-coloring can be achieved in semi-streaming space, the most popular space regime for graph streaming. Behnezhad et al. [12] mentioned that "a major open question is whether [the number of colors for W-streaming edge-coloring] can be improved to $O(\Delta)$ while also keeping the memory near-linear in n." Our results answer the question in the affirmative for the widely studied model of vertex-arrival streams.

Further, we show that the algorithm can be made deterministic using $\widetilde{O}(n)$ bits of *advice* instead of $\widetilde{O}(n\Delta)$ bits of oracle randomness. By picking a uniformly random advice string, the updated algorithm can alternatively be used as a robust algorithm (see Definition 12) with $1/\operatorname{poly}(n)$ error; the advice can also be computed in exponential time.

▶ **Theorem 7.** *Given any adversarial vertex-arrival stream of a multigraph, there is a deterministic online $O(\Delta)$-edge-coloring algorithm using $\widetilde{O}(n)$ bits of space, using $\widetilde{O}(n)$ bits of advice.*

An interesting special case of the vertex-arrival model is the one-sided vertex-arrival setting for bipartite graphs. Here, the vertices on one side of the bipartite graph are fixed, while the vertices on the other side arrive in a sequence along with their incident edges. A couple of works [23, 41] have studied online edge-coloring specifically in this model. We design low-memory online algorithms in this setting and use them as building blocks for our algorithms in the more general settings of vertex-arrival and edge-arrival. These algorithms may be of independent interest due to practical applications of the one-sided vertex-arrival model; moreover, the randomized algorithm here uses only 5Δ colors (as opposed to our other algorithms where the hidden constants in $O(\Delta)$ are rather large).

Finally, we present a lower bound on the space requirement of a deterministic online edge-coloring algorithm. To the best of our knowledge, this is the first non-trivial space lower bound proven for an online edge-coloring algorithm.

▶ **Theorem 8.** *For $\Delta \leq \varepsilon n$ for a sufficiently small constant ε, any deterministic online algorithm that edge-colors a graph using $(2 - o(1))\Delta$ colors requires $\Omega(n)$ space.*

[4] The use of oracle randomness here is not a big deal. In practice (where we assume cryptographic pseudo-random number generators exist), it is straightforward to generate the bits of the oracle random string on demand, ensuring that computationally bounded systems essentially cannot produce hard inputs for the algorithm. Also, against oblivious adversaries, one can easily modify the randomized vertex arrival algorithm to use only $\widetilde{O}(n)$ random bits, by choosing its random permutations from almost $O(\log n)$-wise independent families, but we skip this to keep the proof simple.

1.2 Related Work

Online model

Online edge-coloring has a rich literature [2, 5, 11, 10, 18, 19, 23, 28, 29, 37, 38, 41, 35, 43]. The seminal work of Bar-Noy, Motwani, and Naor [11] ruled out any online algorithm outperforming the $(2\Delta - 1)$-coloring greedy algorithm that assigns each edge the first available color not used by any adjacent edge. However, this lower bound applies only to graphs with $\Delta = O(\log n)$. They conjectured that for $\Delta = \omega(\log n)$, there exist online $(1 + o(1))\Delta$-coloring algorithms. Although this conjecture remains unresolved, there has been significant progress on it over the years. A number of works [2, 10, 18] considered the problem under *random-order* edge arrivals: Aggarwal et al. [2] showed that if $\Delta = \omega(n^2)$, then a tight $(1 + o(1))\Delta$-coloring is possible. For $\Delta = \omega(\log n)$ (the bound in the said conjecture), Bahmani et al. [10] obtained a 1.26Δ-coloring. Bhattacharya et al. [18] then achieved the best aspect of each result as they attained the tight color bound of $(1 + o(1))\Delta$ for the broad range of $\Delta = \omega(\log n)$, essentially resolving the conjecture for random-order arrivals.

More relevant to our work is the setting of *adversarial-order* edge arrivals. Cohen et al. [23] were the first to make progress on [11]'s conjecture in this regard, obtaining a $(1 + o(1))\Delta$-coloring for bipartite graphs under one-sided vertex arrivals (i.e., the nodes on one side are fixed, and the nodes on the other side arrive one by one with all incident edges). Their algorithm assumes a priori knowledge of the value of Δ. For unknown Δ, they rule out any online algorithm using fewer than $(e/(e-1))\Delta$ colors and also complement this result with a $(e/(e-1) + o(1))\Delta$-coloring algorithm. For bipartite *multigraphs* with one-sided vertex arrivals, Naor et al. [41] very recently proved that 1.533Δ colors suffice, while at least 1.207Δ colors are necessary even for $\Delta = 2$. Saberi and Wajc [43] showed that it is possible to beat the greedy algorithm for $\Delta = \omega(\log n)$ under vertex arrivals in general graphs: they designed a $(1.9 + o(1))\Delta$-coloring algorithm. Recently, Kulkarni et al. [35] made the first progress on the said conjecture for fully general adversarial edge arrivals: they obtained a $(e/(e-1) + o(1))\Delta$-coloring in this model. Note that the focus of all these works was on resolving [11]'s conjecture without any space limitations. Our focus is on designing low-memory online algorithms while staying within a constant competitive ratio. The only prior sublinear-space online edge-coloring algorithm we know was given by Ansari et al. [5]: a (deterministic) online $2\Delta t$-coloring in $O(n\Delta/t)$ space for any $t \leq \Delta$.

Several works [29, 26, 28] have addressed the variant where, given a fixed number of colors, the goal is to color as many edges as possible. Mikkelsen [37, 38] considered online edge coloring with limited advice for the future.

W-Streaming model

The W-streaming model [25] is a natural extension of the classical streaming model for studying problems where the output size is very large, possibly larger than our memory. While the W-streaming literature has considered several graph problems [25, 24, 36, 31], we are aware of only three past works [12, 21, 5] that have studied edge-coloring here. Behnezhad et al. [12] initiated the study of W-streaming edge coloring. They considered the problem for both adversarial-order and random-order streams: using $\widetilde{O}(n)$ bits of working memory, they gave an $O(\Delta^2)$-coloring in the former setting, and a $(2e\Delta)$-coloring in the latter setting. Charikar and Liu [21] improved these results: for adversarial-order streams, for any $s \in [\Omega(\log n), \Delta]$, they presented an $O(\Delta^2/s)$-coloring algorithm that uses $\widetilde{O}(ns)$ space; and for random-order streams, they gave a $(1 + o(1))\Delta$-coloring algorithm using $\widetilde{O}(n)$ space. Both of the said adversarial-order streaming algorithms are, however, randomized. Ansari

et al. [5] designed simple deterministic algorithms achieving the same bounds of $O(\Delta^2/s)$ colors and $\widetilde{O}(ns)$ space. Their algorithm can also be made online at the cost of a factor of 2 in the number of colors. Note that, parameterizing our results in Table 2 appropriately, our algorithms achieve $O(\Delta^2/s)$-colorings in $\widetilde{O}(n\sqrt{s})$ space, matching the state of the art for $s = O(1)$, and strictly improving upon it for $s = \omega(1)$.

The related problem of vertex coloring has a more mature literature in the streaming model [1, 3, 6, 7, 8, 9, 15, 16, 17, 20, 32]. However, due to fundamental differences between the classical streaming and W-streaming models and between the two problems, not many techniques seem to carry over.

Concurrent work

In an independent and parallel work, Behnezhad and Saneian [13] have designed a randomized $\widetilde{O}(n\sqrt{\Delta})$-space W-streaming algorithm for $O(\Delta)$-edge-coloring for edge-arrival streams in simple general graphs. This matches our Corollary 2. Their result generalizes to give, for any $s \in [\sqrt{\Delta}]$, an $O(\Delta^{1.5}/s)$ coloring algorithm in $\widetilde{O}(ns)$ space, while we achieve an $O(\Delta^2/s^2)$-coloring in the same space. They also get an $O(\Delta)$-edge-coloring algorithm for vertex-arrival streams using $\widetilde{O}(n)$ space, similar to our Theorem 6. Note that a crucial difference between the two papers is that most of our algorithms have the additional strong feature of being online, while it is not clear if their W-streaming edge-arrival algorithm can also be implemented in the online setting. Thus, conceptually, our results affirm that to obtain an $O(\Delta)$-coloring in sublinear space, the only advantage of W-streaming over online – enabling delay of color assignment – is not necessary.

In terms of techniques, while both works have some high level ideas in common (e.g., using random offsets/permutations to keep track of colors, or designing a one-sided vertex-arrival algorithm first and building on it to obtain the edge-arrival algorithm), the final algorithms and analyses in the two papers are fairly different.

Another independent work by Chechik, Mukhtar, and Zhang [22] obtains a randomized W-streaming algorithm that colors an edge-arrival stream on general multigraphs using $O(\Delta^{1.5}\log\Delta)$ colors in expectation[5], and $\widetilde{O}(n)$ space in expectation. Unlike us, they make no claims in the online model.

2 Notation and Definitions

Throughout the paper, logarithms are in base 2. The notation $[t]$ indicates the set of integers $\{1, \ldots, t\}$. The notation $\widetilde{O}(x)$ ignores $\mathrm{poly}(\log(n), \log(\Delta))$ factors in x. $A \sqcup B$ gives the disjoint union of A and B. $\mathbb{S}_t$ is the set of permutations over $[t]$, and for any permutation $\sigma \in \mathbb{S}_t$ and $X \subseteq [t]$, we denote $\sigma[X] := \{\sigma_i : i \in X\}$. For any set X, $\binom{X}{k}$ denotes the set of all k-sized subsets of X.

If not otherwise stated, n is the number of vertices in a graph G, V the set of vertices (or $A \sqcup B$ if the graph is bipartite), E the (multi-)set of edges, and Δ is the maximum degree of the graph (counting multiplicity, for multigraphs).

[5] While [22] does not claim this, one can prove their algorithm uses $O(\Delta^{1.5}\log\Delta)$ colors with $\geq 1 - 1/\mathrm{poly}(n)$ probability.

2.1 Models

We consider the following models for presenting edges (to be colored) to an algorithm. In all cases, the set of vertices for the graph is known in advance.

▶ **Definition 9** (Edge-Arrival Model). *With an* edge-arrival stream*, the algorithm is given a sequence of edges in the graph. Each edge is provided as an unordered pair $\{x, y\}$ of vertices in V. In this paper, online algorithms processing edge arrival streams will implement a method* Process$(\{x, y\})$ *which returns the color assigned to the edge. W-streaming algorithms may assign the color for an edge at any time, although all edges must be given a color at the end of the stream. We permit algorithms to output not just integers but also tuples of integers as "colors", since users of the algorithms can easily remap these colors to whatever space they wish.*

▶ **Definition 10** (Vertex-Arrival Model). *In a* vertex-arrival stream*, the algorithm is given a sequence of (vertex, edge-set) pairs (v, M_v), where the edge (multi-)set M_v contains all edges from v to vertices that have been seen earlier in the stream. Online algorithms should report colors for all edges in M_v when (v, M_v) is processed.*

A one-sided vertex-arrival stream *on a bipartite graph with partite sets A, B is a vertex arrival stream where the vertices in one partite set (B) are fixed, and then all the (vertex, edge-set) pairs for the other partite set (A) are given. This means that the stream consists of pairs (v, M_v), where each $v \in A$, and M_v contains all edges from v to vertices in B.*

▶ Remark 11 (Assumption of prior knowledge of Δ). We assume that the maximum degree Δ of G is known in advance. An edge-coloring algorithm for which Δ is not known in advance can be converted to one which is, although one way to do this conversion (by running a new 2Δ-coloring algorithm with a fresh set of colors whenever the maximum degree of graph formed by the input stream doubles) increases the total number of colors used by a constant factor, and uses $O(n \log \Delta)$ bits of space to keep track of the maximum degree. Since the algorithms in this paper already have large constant factors on number of colors used, it is not worth it to optimize the algorithms for the case where Δ is not known in advance.

▶ **Definition 12** (Robust Streaming). *An algorithm is said to be* robust *or* adversarially robust *if it works with $\geq 1 - \delta$ probability even when its input streams are adaptively generated. By "adaptively generated", we mean that the input is produced by an adaptive adversary that sees all outputs of the online (or W-streaming) algorithm, and repeatedly chooses the next element of the stream based on what the algorithm has output so far. See [14] for a more detailed explanation.*

2.2 Basic Definitions

▶ **Definition 13** ((ϵ, k)-wise independent permutation). *A random permutation σ is (ϵ, k)-wise independent if for all distinct $a_1, \ldots, a_k$ in $[n]$, the distribution of σ on $a_1, \ldots, a_k$ has total variation distance $\leq \epsilon$ from uniform. In other words,*

$$\frac{1}{2} \sum_{\text{distinct } b_1, \ldots, b_k \in [n]} \left| \Pr\left[\bigwedge_{i \in [k]} \{\sigma(a_i) = b_i\} \right] - \frac{1}{\prod_{i \in [k]} (n - i + 1)} \right| \leq \epsilon \,.$$

Per [4], while it is not known if there are nontrivial $(0, k)$-wise independent families of permutations for large k and n, one can always construct weighted distributions which have support of size $n^{O(k)}$ and provide $(0, k)$-wise independence. However, sampling from these may not be efficient.

We say a random permutation is almost k-wise independent *when it is (ϵ, k)-wise independent for sufficiently small ϵ.*

We, using a result by [40] on switching networks, give a short lemma describing an efficient construction of (ϵ, k)-wise independent permutations.

▶ **Lemma 14** (Random permutations through switching networks). *Let C be a power of 2, let s be a natural number $\leq C$, and let $\epsilon > 0$. For $r = O(s(\log C)^4 \log \frac{1}{\epsilon})$, there is a map f from $\{0,1\}^r$ to $\mathbb{S}_C$ so that, if U is a uniformly random string of r bits, then $\sigma = f(U)$ is an (ϵ, s) wise independent random permutation.*

Furthermore, for any $i \in [C]$ and $u \in \{0,1\}^r$, with $\sigma = f(u)$, we can evaluate $\sigma(i)$ and $\sigma^{-1}(i)$ in $O(s(\log C)^4 \log \frac{1}{\epsilon} \log C)$ time.

3 Technical Overview

In this section, we give a high-level overview of our algorithms and techniques. We see these techniques as a major contribution of the paper since many of them are used for the first time in the context of W-streaming and online edge coloring. The proofs of our results are given in the full version of this paper.

3.1 General Reductions

Reducing to bipartite case

We show that it is essentially enough to consider bipartite graphs. Suppose that we can partition a general graph into $O(\log n)$ bipartite graphs, each of which has max-degree roughly $\Delta/\log n$, where Δ is the max-degree of the original graph. Then, if we run our algorithm on these bipartite graphs with disjoint palettes, we use colors roughly proportional to Δ. It is known (see [21] or for a similar result, [43, Lemma 2.1]) that such a partition can be done in a randomized way, incurring a multiplicative overhead of just $1 + o(1)$ in the number of colors. In this work, we show that if we are willing to tolerate an $O(1)$ blowup in the number of colors, then this partition can be done deterministically. Since such a primitive is used in multiple edge-coloring algorithms, this deterministic version might be of independent interest. One advantage of this version is that it works against adaptive adversaries, unlike the randomized version which can be shown to be breakable by such an adversary.

We construct this partition using appropriate binary codes for the vertices: the codes are of length $O(\log n)$ and we have a bipartite graph corresponding to each bit, where the bipartition is given by whether the bit is 0 or 1. Now, we need to ensure that (a) every edge goes to "some" bipartition, and (b) the max-degree of a single bipartite graph is not much higher than $\Delta/\log n$. This can be done using codes with constant rate and relative distance, like the expander codes described by [45]. Now we can focus on getting $O(\Delta)$-colorings for bipartite graphs, which would give us asymptotically same number of colors for general graphs.

Note that for the vertex-arrival model, we can go one step farther and assume "one-sided" vertex arrival, i.e., vertices along with their incident edges arrive on only one side of the bipartite graph. This is because we can run two copies of the algorithm, one each for the vertices arriving on either side, with disjoint palettes. This incurs only a factor of 2 in the number of colors.

Space-color tradeoff for multigraphs

We show with Lemma 5 that if an algorithm can handle multigraphs, then we can smoothly tradeoff colors with space. This is one of our motivations behind extending our algorithms to multigraphs. Recall that we reduce the problem to just bipartite graphs. Now the idea for the tradeoff is simple: we arbitrarily group t nodes (for some parameter t) from the same partite set together as a single supernode. Since the vertices on the same partite set do not share any edges, there are no edges inside a supernode. Then, the resulting multigraph has no self-loops, but any pair of supernodes can have multiple edges between them. Observe that the max-degree can now increase to Δt, where Δ is the max-degree of the original graph. Thus, if we have an $S(n, \Delta)$-space $f(\Delta)$-coloring algorithm for multigraphs, then we can turn it into an $S(n/t, \Delta t)$-space $f(\Delta t)$-coloring algorithm. In particular, our $\widetilde{O}(n\sqrt{\Delta})$-space $\widetilde{O}(\Delta)$-coloring algorithm from Theorem 3 generalizes to an $\widetilde{O}(n\sqrt{\Delta/t})$-space $\widetilde{O}(\Delta t)$-coloring for any $1 \leq t \leq \Delta$. We generalize most of our algorithms to multigraphs and establish such a tradeoff.

3.2 Randomized Online Algorithms

Randomized online algorithm for vertex arrivals

Recall the simple greedy algorithm for online $(2\Delta - 1)$-coloring: we assign each incoming edge a color that is not already taken up by any of its adjacent edges. However, even in the one-sided vertex arrival model, to naively implement this algorithm, we need to remember the colors assigned to edges incident on *each* vertex on the "fixed" side, and hence, essentially colors assigned to all previous edges. This needs $O(n\Delta)$ space, and hence, the greedy algorithm doesn't seem to help in getting low-memory algorithms.

Algorithm 1 Randomized algorithm for 5Δ-coloring under one sided vertex arrivals.

Input: Stream of vertex arrivals of n-vertex graph $G = (A \sqcup B, E)$

Initialize:

Let $C = 5\Delta$.
for $v \in B$ **do**
    Let σ_v be a uniformly random permutation over $[C]$ ▷ *constructed on demand from random oracle bits*
    $h_v \leftarrow 1$. ▷ *counter for vertex v*

Process(vertex $x \in A$ with multiset M_x of edges)

Let $S_x \leftarrow \emptyset$ ▷ *set of colors M_x will have used so far*
for $e = \{x, y\}$ in M_x, in arbitrary order **do**
    ▷ *Increase counter for y to next value with color $\sigma_y[h_y]$ not already used, if possible*
    while $(h_y \leq C) \wedge (\sigma_y[h_y] \in S_x)$ **do**
        $h_y \leftarrow h_y + 1$ ▷ *increment counter for y*
    if $h_y > C$ **then** ▷ *counter exceeded 5Δ*
        abort
    Assign color $\sigma_y[h_y]$ to e ▷ *if this line is reached, current color must be unused; assign to e*
    $S_x \leftarrow S_x \cup \{\sigma_y[h_y]\}$ ▷ *add assigned color to set of used colors*
    $h_y \leftarrow h_y + 1$ ▷ *increment counter for y*

We observe, however, that for the vertex-arriving side, it is enough to remember only the colors assigned to edges on the "current" vertex so as to ensure no conflict among these edges. We shoot for a semi-streaming, i.e., $\widetilde{O}(n)$ space algorithm, and hence can afford to store the entire edge set of the current vertex with the assigned colors. To ensure that there is no color-conflict on the fixed side, we resort to random permutations. On each such vertex v, we have a random permutation σ_v of $[5\Delta]$ and a counter h_v. When an edge $\{a, b\}$ arrives with b on the fixed side, we look at the color at the h_bth index of σ_b. If that color is already taken by any edge incident on a (whose colors we explicitly store), then we increment the counter h_b. We continue this until we find an available color and increment the counter. The random permutations ensure (i) no color will be repeated on any fixed vertex (since a permutation takes distinct counter values to distinct colors) and (ii) with high probability, none of the counters can exceed 5Δ.[6] Intuitively, the slack in the number of colors ensure that for a single edge, an available color is reached within a constant counter increment in expectation. Hence, the Δ edges incident on a vertex can increase its counter to at most $O(\Delta)$ in expectation. Since we only store a counter for each vertex whose value can go up to $O(\Delta)$, the space usage is $\widetilde{O}(n)$. Thanks to the reductions discussed above, we can extend this to a semi-streaming[7] $O(\Delta)$-coloring for the general vertex arrival case, even for general graphs.

Randomized algorithm for online edge arrivals

When handling edge arrivals for simple graphs, our goal on receiving an edge $\{u, v\}$ is to assign a color that has not been used by any edge incident on u or on v. As precisely tracking the set of available colors for any given vertex requires $\Omega(\Delta)$ space, we will instead keep track of a subset of the available colors for a vertex, and choose colors for edges in a way that limits the amount of information we must store.

Algorithm 2 Storing free regions from a permutation.

$F \leftarrow$InitFreeTracker(C,s,Δ,σ)**:** ▷ *Assume C,s,Δ are powers of two, $C \geq \Delta$, $C \geq s$, and $\sigma \in S_C$*

$H \leftarrow [s]$ be a subset of $[s]$
$b \leftarrow 1$ be a counter with range from 1 to C/s

Interpreting F as subset of $[C]$
return $\sigma[H + (b-1)s]$

F**.RemoveAndUpdate**(c) ▷ *This will only be called with $c \in F$*
$H \leftarrow H \setminus \{\sigma^{-1}(c)\}$
if $|H| \leq s - s\Delta/C$ **then** ▷ *Switch to next block*
    $H \leftarrow [s]$
    $b \leftarrow b + 1$

Specifically, for each vertex v in the graph, we associate a uniform and independently chosen random permutation σ_v over the set $[C]$ of colors, where $C = O(\Delta)$ is the number of colors used. When edges incident on v arrive, we will choose colors by very roughly increasing

[6] This algorithm essentially discards colors that it skips over. We suspect one can obtain a semi-streaming $(2\Delta - 1)$-edge-coloring, or better, by retaining and promptly using the skipped colors in some fashion.
[7] Not counting random oracle bits. To implement without oracle randomness, choose each permutation independently from an almost $O(\log n)$-wise independent distribution over permutations.

order in σ_v. Specifically, we split σ_v up into a sequence of disjoint blocks $P_{v,1}, \ldots, P_{v,C/s}$ of size $s = O(\sqrt{\Delta \log n})$ each. At any given time, the algorithm have an integer b_v so that, all colors in blocks from $P_{v,1}$ up to P_{v,b_v-1} are assumed unsafe to use; none of the colors in $P_{v,b_v+1}, \ldots, P_{v,C/s}$ have been used; and the set F_v of colors in P_{v,b_v} that are still available is tracked exactly. Pseudocode for updating this state is given in Algorithm 2; it will ensure that $|F_v| \geq s(1 - \Delta/C)$. Whenever an edge $\{x, y\}$ arrives, the algorithm (Algorithm 3) will assign it a random color in $F_x \cap F_y$. We will prove that, since F_x and F_y have size close to s, and only a Δ/C fraction of (biased) random elements from P_{x,b_x} and P_{y,b_y} will have been used/excluded from F_x and F_y, the intersection of $F_x \cap F_y$ will have size $\Omega(s^2/C) = \Omega(\log n)$ with high probability. Thus, our algorithm will w.h.p. be able to pick a color for the edge $\{x, y\}$.

Algorithm 3 Randomized algorithm for $O(\Delta)$ edge coloring for simple graph edge arrival streams.

Input: Stream of vertex arrivals n-vertex graph $G = (V, E)$
Assume Δ is a power of two, and $\Delta = \Omega(\log(n/\delta))$
δ is the maximum failure probability over all input streams

Initialize:
Let $C = 128\Delta$ and $\epsilon = \frac{\delta}{16n^3}$
Let s be the least power of two which is $\geq 128\sqrt{\Delta \log(n/\delta)}$
for $v \in V$ **do**
    $\sigma_v \leftarrow$ permutation drawn from the (ϵ, s)-wise independent distribution over $\mathbb{S}_C$ from Lemma 14
    $F_v \leftarrow$ InitFreeTracker(C, s, Δ, σ_v)

Process(edge $\{x, y\}$) $\rightarrow$ **color**
if $F_x \cap F_y = \emptyset$ **then**
    abort
Let c be chosen uniformly at random from $F_x \cap F_y$.
F_x.RemoveAndUpdate(c)
F_y.RemoveAndUpdate(c)
return color c

Keeping track of the available colors for each vertex will use $\widetilde{O}(\sqrt{\Delta})$ space per vertex. We will choose the random permutations from an (ε, s)-wise independent distributions (see Definition 13). We show in a technical lemma that there is a particular such distribution whose permutations can be encoded using $\widetilde{O}(\sqrt{\Delta})$ space each, and for which sampling and permutation evaluation are efficient. In total over all vertices, we will use $\widetilde{O}(n\sqrt{\Delta})$ space for our algorithm.

3.3 Deterministic Online Algorithms

Derandomization of online vertex arrival

When a vertex a and its neighboring edges are processed, the randomized online vertex arrival algorithm is greedily finding an assignment for each edge incident on a to the next few available colors on the "fixed" vertex at the other endpoint. While this greedy selection works acceptably in the *average* case, it does not provide strong worst-case guarantees: with $\Omega(1/\operatorname{poly}(n))$ probability, *some* vertex adjacent to a will increase its counter by $\Omega(\log n)$,

thereby marking ("consuming") $\Omega(\log n)$ colors from its random permutation as unavailable. Consequently, there is small but nonzero risk that a vertex will use up too many colors from its random permutation and run out. In contrast, our deterministic algorithm for online edge coloring for one-sided vertex arrivals (Algorithm 4) is designed to ensure that, for any fixed vertex, the *amortized* number of colors consumed per edge incident on a fixed vertex is always bounded by a constant.

First, instead of greedily coloring the edges incident to a, we explicitly construct a matching in a bipartite graph between the set M_a of edges incident to a and the set of all colors, where each edge $\{a, b\}$ in M_a is linked to some of the colors that are known to be still available for vertex b. This avoids the problem of the greedy color selection, where there was always a small risk that for some vertex, all the next few colors in its random permutation were used; although it has the risk that the matching might not exist. Fortunately, as the number of color candidates per edge in M_a increases, the probability of there being no matching shrinks rapidly. If there are no repeated edges in M_a, and if each edge has t uniformly randomly chosen color candidates, the probability of there being no matching is $\exp(-\Omega(t|M_a|))$.

Second, instead of using random permutations, the deterministic algorithm uses a certain "good" array of permutations for which we are guaranteed that whenever we look for a matching, one will exist. This relies on an additional modification: instead of having each "fixed" vertex v keep track of a single counter h_v, we store a set of $\geq t = \Omega(\log n)$ colors known to still be available for vertex v, and periodically replace this set with a range of new colors from the permutation σ_v. The exact details of the encoding ensure that each fixed vertex has only $O(\operatorname{poly} n)$ possible states. Now, consider what happens when a vertex a (with incident edge set M_a) arrives, assuming for simplicity that M_a has no repeated edges. The number of possible configurations of states of vertices in $N(a)$ will be $\exp(|M_a| \log n)$. If each vertex had a random permutation, the probability of there being no matching would be $\exp(-\Omega(|M_a| \log n)$. By carefully adjusting parameters, we can ensure the product of these two is exponentially *small*. Then by a union bound we can show that the probability (over the random permutations) of *any* matching failing, for *any* set M_a and any associated vertex state, is at most $\frac{1}{2}$. In other words, for a good choice of permutations, our algorithm would *always* find a matching, in any state.

A notable problem is that storing each permutation would require $\widetilde{O}(\Delta)$ bits each.[8] To avoid this, we select the permutations from small, almost-k-wise independent families of permutations, instead. This works, but requires a more careful analysis to prove correct, and a large fraction of our proof is spent dealing with the interaction of this and support for multigraphs.

Derandomization of online edge arrival

We did not find a way to directly derandomize the randomized algorithm for online edge arrival. Instead, we created an algorithm (Algorithm 5) that manages to *partially* solve the edge coloring problem, only assigning a color to a $\geq 1/3$ fraction of the incoming edges, and leaving the rest uncolored. Now, say we run $O(\log \Delta)$ instances of this algorithm in parallel, each using a distinct palette of $O(\Delta)$ colors. When an edge arrives, we pass it to the first instance of the algorithm, and if it wasn't assigned a color, pass it to the second instance;

[8] One could recompute individual "good" permutations on demand, instead of storing all of the permutations, but this has the risk of the computation itself requiring $\widetilde{O}(n\Delta)$ bits of scratch space.

Algorithm 4 Deterministic $O(\Delta)$ edge coloring algorithm for one sided bipartite vertex arrivals.

Input: Stream of vertex arrivals for n-vertex graph $G = (A \sqcup B, E)$ of max degree Δ, where Δ is a power of 2

Initialize(δ):
Let $C = 2^{18}\Delta$.
Let $s = \lceil 2^{18} \log \frac{n\Delta}{\delta} \rceil$.
▷ *To get a deterministic algorithm, fix "good" values of these permutations*
▷ *When the permutations are chosen randomly, with probability* $\geq 1 - \delta$ *the algorithm will succeed on* all *inputs*
Let $(\sigma_v)_{v \in B}$ be permutations drawn from an $(\epsilon = C^{-s-1}, s)$-wise independent distribution over $\mathbb{S}_C$, using Lemma 14

for $v \in B$ **do**
  $b_v \leftarrow 1$.
  $Q_v \leftarrow [s]$.

Process(vertex x with multiset M_x of edges to B)
Let $d_{x,y}$ be the number of times edge $\{x, y\}$ is in M_x
for each $y \in B$ with $d_{x,y} > 0$ **do**
  if $d_{x,y} < \frac{1}{16}s$ **then**
    Let $F_y = (b_y - 1)s + Q_y$
  else
    Let $F_y = ((b_y - 1)s + Q_y) \sqcup \left[b_y s + 1, b_y s + \left\lceil \frac{64 d_{x,y}}{s} \right\rceil s\right]$
Construct bipartite graph H from M_x to $[C]$, edge $e \in M_x$ is linked to all $c \in \sigma_y[F_y]$.
Compute an M_x-saturating matching P of H.
for each $e \in M_x$ **do**
  Assign color $P(e)$ to e
  if $d_{x,y} < \frac{1}{16}s$ **then**
    Remove $\sigma_y^{-1}(P(e)) - (b_y - 1)s$ from Q_y
for each $y \in B$ with $d_{x,y} > 0$ **do**
  if $d_{x,y} < \frac{1}{16}s$ **then**
    if $|Q_y| \leq s - \frac{1}{2^{17}}s$ **then**
      $b_y \leftarrow b_y + 1$
      $Q_y \leftarrow [s]$
  else
    $Q_y \leftarrow [s]$
    $b_y \leftarrow b_y + \left\lceil \frac{64 d_{x,y}}{s} \right\rceil + 1$

and if that didn't assign a color, pass it to the third instance, and so on. All in all, only a $\leq 1/\operatorname{poly}(\Delta)$ fraction of the input stream will fail to be colored by this process, and there are few enough of these leftover edges that one can store them all and color them using $O(\Delta)$ colors. This approach uses $O(\Delta \log \Delta)$ colors in total.

The partial edge coloring algorithm itself uses an interesting trick. Let C be the number of colors allowed. Each vertex v has an associated permutation $\sigma_v \in S_C$, which is partitioned into a number of blocks $P_{v,1}, P_{v,2}, \ldots$ of $\widetilde{O}(\sqrt{\Delta})$ colors each. Whenever an edge $\{u, v\}$ arrives, it must be colored using a color in the set $P_{v,i} \cap P_{u,j}$, where i and j depend on the degrees of vertices v and u at the time. Parameters are set up so this set has size $\Omega(\log n)$, and the

Algorithm 5 (1/3)-partial $O(\Delta)$ edge coloring algorithm for graph edge arrival streams.

Input: Stream of edge arrivals in n-vertex graph $G = (V, E)$ of max degree Δ, where Δ is a power of two

Initialize(Δ,C,s):
▷ *C: number of colors used; s: block size parameter*
▷ *As with Algorithm 4, a "good" set of permutations will exist, if C/Δ is large*
$(\sigma_v)_{v\in V}$ are specific s-wise almost independent permutations
1: **for** $v \in V$ **do**
2: $F_v \leftarrow$ InitFreeTracker(C, s, Δ, σ_v). ▷ *Defined in: Algorithm 2*

Process(edge $\{x, y\}$) $\rightarrow$ **Option<color>** $\in \{\perp\} \cup [C]$
3: **if** $F_x \cap F_y \neq \emptyset$ **then**
4: Choose $c \in F_x \cap F_y$ arbitrarily
5: F_x.RemoveAndUpdate(c, $\{x, y\}$)
6: F_y.RemoveAndUpdate(c, $\{x, y\}$)
7: **return** color c
8: **else**
9: **return** $\perp$

algorithm knows which colors in $P_{v,i}$ and $P_{u,j}$ have been used so far. We prove that, if the algorithm could preview the future of the stream, it could always pick the "right" color in $P_{v,i} \cap P_{u,j}$, and thereby find a valid edge coloring. On the other hand, without being able to look at future edges, if one just greedily picks valid colors in $P_{v,i} \cap P_{u,j}$ that don't conflict with colors chosen earlier – assuming there are any – then the algorithm will color at least a 1/3 fraction of the edges.

Handling multigraphs in online edge arrival

Let us now turn to multigraphs. Note that the degree of a vertex takes into consideration the multiplicity of each incident edge. Both the randomized online edge arrival algorithm and what was described so far of the deterministic algorithm will fail when faced with input streams that repeat edges. But looking more closely, both can tolerate some amount of repeated edges – a given edge e could be repeated up to $\widetilde{O}(\sqrt{\Delta})$ times, as long as it doesn't arrive too frequently. Specifically, as long as, for a given endpoint x of e, the substream of edges incident on x does not contain e more than $\widetilde{O}(1)$ times in any interval of $\widetilde{O}(\sqrt{\Delta})$ edges. This is a consequence of the way the edge arrival algorithms rotate between blocks of colors for each vertex.

On the other hand, the edges that would break the sketch, which repeat many times within the last $\widetilde{O}(\sqrt{\Delta})$ edges incident on a given vertex, are easy to detect using $\widetilde{O}(n\sqrt{\Delta})$ space. All one must do is keep track of the edges which recently arrived at a vertex, and detect duplicates.

To handle the "bad" type of repeated edges, we maintain $O(\log \Delta)$ modified instances of an edge arrival algorithm. (In this paper, we use the basic deterministic online edge arrival algorithm, but the same argument would work for the randomized one.) The first instance processes all edges in the stream, filtering out the edges which it detects to have repeated at least once. The repeated edges are sent to the second instance, which filters out edges that it finds to have repeated twice, and sends those to the third instance. In general, the

ith instance will receive edges which have been seen $\Theta(2^i)$ times in the stream. (The exact condition is a bit more complicated.) The ith instance is also modified to handle edges with high repetition rates, assigning batches of colors to each edge type that it processes.

3.4 A Lower Bound

Space lower bound for online deterministic edge coloring

Our last notable result is a lower bound on the space needed for deterministic online edge coloring algorithms, which use $\beta\Delta$ colors, for a constant $\beta < 2$. It applies in the one sided bipartite vertex arrival setting, and thus automatically gives a lower bound for general vertex and edge arrivals. Let B be the "fixed" set of vertices, and A the "arriving" set of vertices. We prove the lower bound by reducing a deterministic, Δ-player, one way, communication game to (deterministic, one-sided, bipartite, vertex arrival) online edge coloring.

In this game, each player receives a set of edges to color, and must immediately output a coloring for the edges, before sending a message to the next player. Say that there are only $2^{o(n)}$ possible messages. Each message corresponds to a collection of inputs which the player could have received. One can show that each message "rules out" some of the colors for each vertex v in B, so that, if the protocol is correct, future players cannot mark edges going to v with one of the ruled out colors. Furthermore, there must be some message which has a large number of associated inputs, and which rules out $> \beta$ colors for each fixed vertex, on average. As this can happen for each of the Δ players, by the end of the protocol it is possible to have ruled out $> \beta\Delta$ colors per vertex, on average, which contradicts the assumption that the algorithm uses at most $\beta\Delta$ colors. Thus, there must be $2^{\Omega(n)}$ possible messages.

References

1. Amir Abboud, Keren Censor-Hillel, Seri Khoury, and Ami Paz. Smaller cuts, higher lower bounds. *CoRR*, abs/1901.01630, 2019. doi:10.48550/arXiv.1901.01630.
2. Gagan Aggarwal, Rajeev Motwani, Devavrat Shah, and An Zhu. Switch scheduling via randomized edge coloring. In *44th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2003,*, pages 502–512. IEEE, 2003. doi:10.1109/SFCS.2003.1238223.
3. Noga Alon and Sepehr Assadi. Palette sparsification beyond (Δ+1) vertex coloring. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2020, August 17-19, 2020, Virtual Conference*, volume 176 of *LIPIcs*, pages 6:1–6:22, 2020. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.6.
4. Noga Alon and Shachar Lovett. Almost k-wise vs. k-wise independent permutations, and uniformity for general group actions. In *Proc. 16th International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 350–361. Springer, 2012. doi:10.1007/978-3-642-32512-0_30.
5. Mohammad Ansari, Mohammad Saneian, and Hamid Zarrabi-Zadeh. Simple Streaming Algorithms for Edge Coloring. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms (ESA 2022)*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:4, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2022.8.
6. Sepehr Assadi, Amit Chakrabarti, Prantar Ghosh, and Manuel Stoeckl. Coloring in graph streams via deterministic and adversarially robust algorithms. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2023, Seattle, WA, USA, June 18-23, 2023*, pages 141–153. ACM, 2023. doi:10.1145/3584372.3588681.
7. Sepehr Assadi, Andrew Chen, and Glenn Sun. Deterministic graph coloring in the streaming model. In *Proc. 54th Annual ACM Symposium on the Theory of Computing*, pages 261–274, 2022. doi:10.1145/3519935.3520016.

8 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for (Δ+ 1) vertex coloring. In *Proc. 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 767–786, 2019. doi:10.1137/1.9781611975482.48.

9 Sepehr Assadi, Pankaj Kumar, and Parth Mittal. Brooks' theorem in graph streams: a single-pass semi-streaming algorithm for Δ-coloring. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 234–247. ACM, 2022. doi:10.1145/3519935.3520005.

10 Bahman Bahmani, Aranyak Mehta, and Rajeev Motwani. Online graph edge-coloring in the random-order arrival model. *Theory of Computing*, 8(1):567–595, 2012. doi:10.4086/toc.2012.v008a025.

11 Amotz Bar-Noy, Rajeev Motwani, and Joseph Naor. The greedy algorithm is optimal for on-line edge coloring. *Information Processing Letters*, 44(5):251–253, 1992. doi:10.1016/0020-0190(92)90209-E.

12 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Marina Knittel, and Hamed Saleh. Streaming and massively parallel algorithms for edge coloring. In *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 15:1–15:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.15.

13 Soheil Behnezhad and Mohammad Saneian. Streaming edge coloring with asymptotically optimal colors. *arXiv preprint arXiv:2305.01714*, 2023. doi:10.48550/arXiv.2305.01714.

14 Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. In *Proc. 39th ACM Symposium on Principles of Database Systems*, pages 63–80, 2020. doi:10.1145/3375395.3387658.

15 Suman K. Bera, Amit Chakrabarti, and Prantar Ghosh. Graph coloring via degeneracy in streaming and other space-conscious models. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 11:1–11:21, 2020. doi:10.4230/LIPIcs.ICALP.2020.11.

16 Suman Kalyan Bera and Prantar Ghosh. Coloring in graph streams. *CoRR*, abs/1807.07640, 2018. doi:10.48550/arXiv.1807.07640.

17 Anup Bhattacharya, Arijit Bishnu, Gopinath Mishra, and Anannya Upasana. Even the easiest(?) graph coloring problem is not easy in streaming! In *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, pages 15:1–15:19, 2021. doi:10.4230/LIPIcs.ITCS.2021.15.

18 Sayan Bhattacharya, Fabrizio Grandoni, and David Wajc. Online edge coloring algorithms via the nibble method. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2830–2842. SIAM, 2021. doi:10.1137/1.9781611976465.168.

19 Joakim Blikstad, Ola Svensson, Radu Vintan, and David Wajc. Simple and asymptotically optimal online bipartite edge coloring. In *2024 Symposium on Simplicity in Algorithms (SOSA)*, pages 331–336, 2024. doi:10.1137/1.9781611977936.30.

20 Amit Chakrabarti, Prantar Ghosh, and Manuel Stoeckl. Adversarially robust coloring for graph streams. In *Proc. 13th Conference on Innovations in Theoretical Computer Science*, pages 37:1–37:23, 2022. doi:10.4230/LIPIcs.ITCS.2022.37.

21 Moses Charikar and Paul Liu. Improved algorithms for edge colouring in the W-streaming model. In *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 181–183. SIAM, 2021. doi:10.1137/1.9781611976496.20.

22 Shiri Chechik, Doron Mukhtar, and Tianyi Zhang. Streaming edge coloring with subquadratic palette size. *arXiv preprint arXiv:2305.07090*, 2023. doi:10.48550/arXiv.2305.07090.

23 Ilan Reuven Cohen, Binghui Peng, and David Wajc. Tight bounds for online edge coloring. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1–25. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00010.

24 Camil Demetrescu, Bruno Escoffier, Gabriel Moruz, and Andrea Ribichini. Adapting parallel algorithms to the W-stream model, with applications to graph problems. *Theoretical Computer Science*, 411(44):3994–4004, 2010. doi:10.1016/j.tcs.2010.08.030.

25 Camil Demetrescu, Irene Finocchi, and Andrea Ribichini. Trading off space for passes in graph streaming problems. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006*, pages 714–723. ACM Press, 2006. doi:10.1145/1644015.1644021.

26 Martin R. Ehmsen, Lene M. Favrholdt, Jens S. Kohrt, and Rodica Mihai. Comparing first-fit and next-fit for online edge coloring. *Theor. Comput. Sci.*, 411(16-18):1734–1741, 2010. doi:10.1016/j.tcs.2010.01.015.

27 Thomas Erlebach and Klaus Jansen. The complexity of path coloring and call scheduling. *Theoretical Computer Science*, 255(1):33–50, 2001. doi:10.1016/S0304-3975(99)00152-8.

28 Lene M. Favrholdt and Jesper W. Mikkelsen. Online edge coloring of paths and trees with a fixed number of colors. *Acta Informatica*, 55(1):57–80, 2018. doi:10.1007/s00236-016-0283-0.

29 Lene M. Favrholdt and Morten N. Nielsen. On-line edge-coloring with a fixed number of colors. *Algorithmica*, 35(2):176–191, 2003. doi:10.1007/s00453-002-0992-3.

30 S. Gandham, M. Dawande, and R. Prakash. Link scheduling in sensor networks: distributed edge coloring revisited. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 4, pages 2492–2501 vol. 4, 2005. doi:10.1109/INFCOM.2005.1498534.

31 Christian Glazik, Jan Schiemann, and Anand Srivastav. A one pass streaming algorithm for finding euler tours. *Theory of Computing Systems*, 67(4):1–23, December 2022. doi:10.1007/s00224-022-10077-w.

32 Magnus M. Halldorsson, Fabian Kuhn, Alexandre Nolin, and Tigran Tonayan. Near-optimal distributed degree+1 coloring. In *Proc. 54th Annual ACM Symposium on the Theory of Computing*, pages 450–463, 2022. doi:10.1145/3519935.3520023.

33 Ian Holyer. The np-completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, 1981. doi:10.1137/0210055.

34 Tiago Januario, Sebastián Urrutia, Celso C. Ribeiro, and Dominique de. Werra. Edge coloring: A natural model for sports scheduling. *European Journal of Operational Research*, 254(1):1–8, 2016. doi:10.1016/j.ejor.2016.03.038.

35 Janardhan Kulkarni, Yang P. Liu, Ashwin Sah, Mehtaab Sawhney, and Jakub Tarnawski. Online edge coloring via tree recurrences and correlation decay. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 104–116. ACM, 2022. doi:10.1145/3519935.3519986.

36 Luigi Laura and Federico Santaroni. Computing strongly connected components in the streaming model. In *Theory and Practice of Algorithms in (Computer) Systems*, pages 193–205. Springer Berlin Heidelberg, 2011. doi:10.1007/978-3-642-19754-3_20.

37 Jesper W. Mikkelsen. Optimal online edge coloring of planar graphs with advice. In *Algorithms and Complexity - 9th International Conference, CIAC 2015, Paris, France, May 20-22, 2015. Proceedings*, volume 9079 of *Lecture Notes in Computer Science*, pages 352–364. Springer, 2015. doi:10.1007/978-3-319-18173-8_26.

38 Jesper W. Mikkelsen. Randomization can be as helpful as a glimpse of the future in online computation. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 39:1–39:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.39.

39 Jayadev Misra and David Gries. A constructive proof of Vizing's theorem. *Information Processing Letters*, 41(3):131–133, 1992. doi:10.1016/0020-0190(92)90041-S.

40 Ben Morris. Improved mixing time bounds for the thorp shuffle. *Combinatorics, Probability and Computing*, 22(1):118–132, 2013. doi:10.1017/S0963548312000478.

41 Joseph Naor, Aravind Srinivasan, and David Wajc. Online dependent rounding schemes. *CoRR*, abs/2301.08680, 2023. doi:10.48550/arXiv.2301.08680.

42 Prabhakar Raghavan and Eli Upfal. Efficient routing in all-optical networks. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing (STOC)*, pages 134–143, 1994. doi:10.1145/195058.195119.

43 Amin Saberi and David Wajc. The greedy algorithm is not optimal for on-line edge coloring. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 109:1–109:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.109.

44 Claude E. Shannon. A theorem on coloring the lines of a network. *Journal of Mathematics and Physics*, 28(1-4):148–152, 1949. doi:10.1002/sapm1949281148.

45 Michael Sipser and Daniel A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996. doi:10.1109/18.556667.

46 V. G. Vizing. On an estimate of the chromatic class of a p-graph. *Discret Analiz*, 3:25–30, 1964.

On the Smoothed Complexity of Combinatorial Local Search

Yiannis Giannakopoulos
School of Computing Science, University of Glasgow, UK

Alexander Grosz
School of Computation, Information and Technology, Technical University of Munich, Germany

Themistoklis Melissourgos
School of Computer Science and Electronic Engineering, University of Essex, UK

Abstract

We propose a unifying framework for smoothed analysis of combinatorial local optimization problems, and show how a diverse selection of problems within the complexity class PLS can be cast within this model. This abstraction allows us to identify key structural properties, and corresponding parameters, that determine the smoothed running time of local search dynamics. We formalize this via a black-box tool that provides concrete bounds on the expected maximum number of steps needed until local search reaches an exact local optimum. This bound is particularly strong, in the sense that it holds for any starting feasible solution, any choice of pivoting rule, and does not rely on the choice of specific noise distributions that are applied on the input, but it is parameterized by just a global upper bound ϕ on the probability density. The power of this tool can be demonstrated by instantiating it for various PLS-hard problems of interest to derive efficient smoothed running times (as a function of ϕ and the input size).

Most notably, we focus on the important local optimization problem of finding pure Nash equilibria in Congestion Games, that has not been studied before from a smoothed analysis perspective. Specifically, we propose novel smoothed analysis models for general and Network Congestion Games, under various representations, including explicit, step-function, and polynomial resource latencies. We study PLS-hard instances of these problems and show that their standard local search algorithms run in polynomial smoothed time.

Further applications of our framework to a wide range of additional combinatorial problems can be found in the full version of our paper.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory; Mathematics of computing → Combinatorial optimization; Theory of computation → Computational complexity and cryptography

Keywords and phrases Smoothed Analysis, local search, better-response dynamics, PLS-hardness, combinatorial local optimization, congestion games, pure Nash equilibria

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.72

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2211.07547` [12]

Funding *Alexander Grosz*: Supported by the Alexander von Humboldt Foundation with funds from the German Federal Ministry of Education and Research (BMBF).

Acknowledgements Y. Giannakopoulos is grateful to Diogo Poças for many useful discussions and inspiration during the early stages of this project.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 72; pp. 72:1–72:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Local search heuristics are some of the most prominent, and widely used in practice, algorithms for solving computationally hard, combinatorial problems [1, 15]. Their appeal stems not only from their simplicity and theoretical elegance, but also from the fact that, for many applications, they seem to perform remarkably well both in terms of their running time and the quality of the solutions they produce.

Theoreticians have long tried to rigorously study the performance of local search, but also explain its prevalence in practice. Johnson, Papadimitriou, and Yannakakis [13] introduced the class PLS to capture the complexity of local optimization problems; since then, many important such problems have been shown to be PLS-complete, implying that they most likely cannot be solved (exactly) in polynomial time. This hardness applies not only to local search algorithms, but to arbitrary local optimization methods. For local search, in particular, PLS-hardness (under tight reductions) implies the provable existence of instances leading to exponentially slow convergence [26]. Examples include the Travelling Salesman problem under the k-Opt heuristic [14, 5] (TSP/k-Opt in the following), Local Maximum Cut on weighted graphs [23] (LocalMaxCut), and the problem of finding pure Nash equilibria in Congestion Games [10] (PNE-Congestion).

On the other hand, Orlin, Punnen, and Schulz [17] designed a local-search-based polynomial-time scheme for efficiently computing *approximately* locally-optimal solutions for general combinatorial optimization problems (with linear objectives). Although this result provides concrete justification for the practical tractability of local optimization, there are still many important aspects that call for further investigation. First, if one requires exponential accuracy, their FPTAS still cannot provide polynomial running times. Secondly, approximate solutions do not always make sense for all local optimization problems; there are problems in PLS that are inherently exact and they are not derived by simply considering the local version of some "master" global optimization problem. A notable example is PNE-Congestion [19]. Finally, we would like to be able to argue about the more general family of "vanilla" local search, and to ideally get positive results that do not depend on additional details and specific choices of pivoting rules. Addressing these points is a key objective of the present paper.

Smoothed analysis was introduced by Spielman and Teng [24] as a more realistic alternative to traditional worst-case analysis, where now the adversarially selected input is submitted to small random shocks of its numerical parameters, *before* being presented to the algorithm; the running time is then measured *in expectation* with respect to these perturbations. Under this model, Spielman and Teng were able to show that Simplex, the archetypical method for solving linear programs, is guaranteed to terminate in polynomial time (under a shadow pivoting rule) – as opposed to its exponential complexity under worst-case analysis. This remarkable result established smoothed analysis as a canonical framework for studying the performance of algorithms beyond the worst-case (see [20] for an overview of this field).

In particular, smoothed analysis has been applied successfully to important local search algorithms, providing thus a theoretical basis for the justification of their good performance in practice; these include, e.g., the k-Opt heuristic for TSP and the Flip heuristic for LocalMaxCut. (A more detailed exposition of related work on this front is deferred to the following sections of this paper, where each of our local optimization problems of interest is explicitly studied; see, namely, Section 4 and [12, Sec. 5].) A common characteristic of this prior work, though, is that running-time analysis is usually tailored specifically to the local optimization problem at hand. This naturally creates the need for technically

heavy derivations, from which it is generally not clear how to pin down the core properties of the underlying local-search structure that allow for the efficient smoothed complexity. Furthermore, as a result, it is often not easy to immediately generalize these results to capture interesting extensions, e.g., argue about the *asymmetric* version of TSP, or go from TSP/2-Opt to TSP/k-Opt. Finally, this lack of sufficient abstraction is one of the reasons that smoothed analysis has not been yet considered at all for prominent PLS-hard problems, including, e.g., PNE-Congestion. Dealing with this set of challenges is another driving force behind our paper.

Our Results and Outline

We start by proposing an abstract model for smoothed analysis of combinatorial local optimization (CLO) in Section 2. Our family of CLO problems includes problems in PLS that have an arbitrary combinatorial neighbourhood structure and linear objective functions; essentially, our model generalizes that of [17] beyond binary configurations. In Section 2.1, we add our smoothness layer that introduces probabilistic noise (independently) to the cost parameters of the CLO problem. No further assumptions are made on the distributions of the perturbed costs; their densities are only parameterized by a global upper bound of ϕ. This is a standard model (employed, e.g., in [3, 22, 8, 9]) that makes positive smoothed-analysis results even stronger, and extends the seminal model of Gaussian perturbations from [24].

Section 3 contains our key technical result for deriving upper bounds on the expected (under smoothness) number of local-search steps, until an *exact* local optimum is reached. Our black-box tool (Theorem 2) can be readily applied to an abstract CLO problem, once its underlying neighbourhood structure is appropriately captured; this is formalized through the notion of *separability* (see Definition 1), quantified by a collection of three parameters that are critical for the upper bound given by Theorem 2. We note here that our bounds are robust against the specific choice of a starting point for the local search dynamics, as well as the pivoting rule utilized at every step to transition to an improving neighbour. In other words, our main black-box tool establishes bounds for the entire *family* of local search heuristics of a local optimization problem. At a technical level, our proof works by lower-bounding the probability that *all* steps of the local-search sequence improve sufficiently the objective.

In Section 4 we demonstrate the applicability of our general framework by instantiating it for PNE-Congestion, a prominent PLS-complete problem which has not been studied before from a smoothed analysis perspective. First, we propose smoothed analysis formulations for various representations of interest for the problem, namely explicit, step-function, or polynomial resource latencies. Next, after formally establishing how PNE-Congestion is indeed a CLO problem in Section 4.1, we identify a natural parameterization of the problem that we call B-*restrained games* (Section 4.2), where B is an upper bound on the number of resources that can be changed during a single-player deviation (see Definition 5). Interestingly enough, the case of constant B is still rich enough to encode the full PLS-hardness of PNE-Congestion ([12, Appendix B]), while at the same time it can be shown (see the proof of Theorem 6) to be appropriately separable in order to immediately provide polynomial smoothed running time bounds via our black-box tool developed in Section 3. Similarly, in Section 4.3 we also study a special class of *network* congestion games, which we term (A, B)-*compact* (Definition 9); we establish polynomial smoothed complexity for various families of instances, including the one where A is polynomial and B is constant (Corollary 11), which we pair with a complementing PLS-hardness proof ([12, Appendix C]).

Finally, we apply our high-level framework to various other local optimization problems of interest, by first formally establishing that they can be viewed as CLO problems and then identifying the proper separability structures that can be plugged into our black-box tool to

provide good smoothed bounds for local search. This not only unifies, and greatly simplifies, prior existing positive results, but also allows us to extend or improve them. Notable examples include: rederiving and strengthening the polynomial smoothed time for local Max-Cut, for graphs with up to logarithmic degree, first given by Elsässer and Tscheuschner [7]; extending the polynomial smoothed time, of Englert, Röglin, and Vöcking [8], for the 2-Opt heuristic for the general (symmetric) Travelling Salesman problem (TSP), to k-Opt neighbourhoods and to *asymmetric* TSP (ATSP); and improving the quasipolynomial smoothed time for Network Coordination Games, by Boodaghians, Kulkarni, and Mehta [4], to polynomial for constant-degree graphs.

Due to space constraints, the study of all these additional applications of our framework is deferred to the full version of our paper [12].

2 Smoothed Combinatorial Local Optimization

In this section we formalize our model and fix the necessary notation.

We denote by $\mathbb{N}$ and $\mathbb{R}$, the natural and real numbers, respectively. We also denote $\mathbb{N}^* \coloneqq \mathbb{N} \setminus \{0\}$ and $[k] \coloneqq \{1, 2, \ldots, k\}$, $[\![k]\!] \coloneqq \{0\} \cup [k]$ for $k \in \mathbb{N}$. We will use boldface notation for vectors, $\boldsymbol{s} = (s_1, s_2, \ldots, s_n) \in \mathbb{R}^n$. For an index $i \in [n]$, we use $\boldsymbol{s}_{-i}$ to denote the $(n-1)$-dimensional vector that results from an n-dimensional vector $\boldsymbol{s}$ if we remove its i-component; in that way, we can express $\boldsymbol{s}$ as $(s_i, \boldsymbol{s}_{-i})$ in order to easily denote deviations in the i-th component. More generally, for a set of indices $I \subseteq [n]$, $\boldsymbol{s}_I$ denotes the $|I|$-dimensional vector that we get if we keep only the components of $\boldsymbol{s}$ whose indices are in I. For finite sets $I, \mathcal{I}$, we say that $\mathcal{I}$ is a *cover* of I if $I \subseteq \bigcup_{I' \in \mathcal{I}} I'$. All logarithms appearing in our paper are of base 2.

Combinatorial local optimization (CLO)

An instance of a *combinatorial local optimization (CLO)* problem is composed of:

- A set of feasible *configurations* $\boldsymbol{S} \subseteq [\![M]\!]^\nu \times \{0,1\}^{\bar{\nu}}$, where $M, \nu \in \mathbb{N}^*$, $\bar{\nu} \in \mathbb{N}$. A configuration $\boldsymbol{s}$ can be expressed as $\boldsymbol{s} = (\boldsymbol{s}^\bullet, \boldsymbol{s}^\circ)$, where $\boldsymbol{s}^\bullet = (s_1, \ldots, s_\nu) \in [\![M]\!]^\nu$ is called its *cost part* and $\boldsymbol{s}^\circ = (s_{\nu+1}, \ldots, s_{\nu+\bar{\nu}}) \in \{0,1\}^{\bar{\nu}}$ its *non-cost part*.
- A vector of *costs* $\boldsymbol{c} = (c_1, c_2, \ldots, c_\nu) \in [-1, 1]^\nu$. We call the c_i *cost coefficients*.[1]
- A *neighbourhood* function $N : \boldsymbol{S} \longrightarrow 2^{\boldsymbol{S}}$. For $\boldsymbol{s} \in \boldsymbol{S}$, any configuration $\boldsymbol{s}' \in N(\boldsymbol{s})$ will be called a *neighbour* of $\boldsymbol{s}$.

For the special case of $M = 1$ we will call our problem *binary*.

The *cost* of a configuration $\boldsymbol{s}$ (with respect to a fixed cost vector $\boldsymbol{c}$) is given by

$$C(\boldsymbol{s}) \coloneqq \boldsymbol{c} \cdot \boldsymbol{s}^\bullet = \sum_{i=1}^{\nu} c_i s_i.$$

[1] Note that the restriction of costs to the range of $[-1, 1]$ has been made to facilitate the translation to smoothed CLO problems below (see Section 2.1). In fact, due to the linearity of our objective, scaling $\boldsymbol{c}$ by any positive constant does not change the structure of the problem, and we will implicitly use this fact when formulating other problems as CLO problems.

A configuration $\boldsymbol{s}$ is said to be a *local optimum (minimum*[2]*)* if there are no neighbours with better costs; formally,

$$C(\boldsymbol{s}) \leq C(\boldsymbol{s}') \qquad \text{for all } \ \boldsymbol{s}' \in N(\boldsymbol{s}). \tag{1}$$

For every CLO problem we can define its *configuration* (or *neighbourhood*) *graph*, made up by all edges pointing from configurations to their neighbours. Formally, it is the directed graph $G = (V, E)$ with node set $V = \boldsymbol{S}$ and edges $E = \{(\boldsymbol{s}, \boldsymbol{s}') \mid \boldsymbol{s} \in \boldsymbol{S},\ \boldsymbol{s}' \in N(\boldsymbol{s})\}$. Observe that the configuration graph does not depend on the costs $\boldsymbol{c}$, but only on the combinatorial structure of the problem. Taking the cost vector $\boldsymbol{c}$ into consideration, we can now restrict the configuration graph edges to those that correspond to locally improving moves, i.e. take $E' = \{(\boldsymbol{s}, \boldsymbol{s}') \in E \mid C(\boldsymbol{s}') < C(\boldsymbol{s})\}$. The resulting acyclic subgraph $G' = (V, E')$ is called *transition graph*.

The transition graph provides an elegant and concise interpretation of local optimization: finding a local optimum of a combinatorial (local optimization) problem translates to finding a sink of its transition graph. This task is exactly the object of interest of our paper: we study (the running time of) algorithms that find local optima of such problems.

Complexity

If the configuration set and the neighbourhoods $N(\cdot)$ were given explicitly in the input of a local optimization algorithm, then finding a locally optimal solution would have been a computationally trivial task: we could afford (in polynomial time) to exhaustively go over all nodes of the transition graph, until we find a sink. Notice that the existence of a sink is guaranteed, by the fact that $\boldsymbol{S}$ is finite. However, most problems of interest (including all the problems that we study here) have exponential-size configuration graphs, with respect to the dimension $\nu + \bar{\nu}$ of the problem. Therefore, $\boldsymbol{S}$ and N are usually instead described *implicitly*, via some succinct representation (that is polynomial on ν, $\bar{\nu}$ and M). Then, the computational complexity of our algorithms is naturally measured as a function of the critical parameters ν, $\bar{\nu}$ and M (and the bit representation of the costs $\boldsymbol{c}$).

Polynomial local optimization (PLS)

In this paper we want to study problems contained in PLS, the "canonical" complexity class for local optimization problems, introduced by Johnson, Papadimitriou, and Yannakakis [13]. Therefore, without further mention, from now on we will assume that our CLO problems further satisfy the following properties:

- An *initial* configuration $\boldsymbol{s}_0 \in \boldsymbol{S}$ can be computed in polynomial time (on the size of the input).
- There exists a polynomial-time algorithm that, given as input any configuration $\boldsymbol{s}$, decides whether $\boldsymbol{s}$ is a local optimum and, if not, returns an improving neighbour $\boldsymbol{s}' \in N(\boldsymbol{s})$ with $C(\boldsymbol{s}') < C(\boldsymbol{s})$. Such an algorithm is called *pivoting rule*.

It is important to clarify that the polynomial time algorithms of the bullets above are formally *not* part of the description of the CLO problem, neither are they required to be specified in the input of a local optimization algorithm. The *existence* of these algorithms is

[2] Here we choose to write the local optimality condition with respect to minimization problems. All the theory developed in this paper applies immediately to local maximization problems as well: just flip the inequality in (1) or, equivalently, negate the cost vector $\boldsymbol{c}$.

merely a requirement for the *membership* of the problem in the class PLS (similarly to the existence of short certificates and an efficient verifier for membership in NP). As a result, PLS can be interpreted as the class of problems that correspond to looking for a sink in an (implicitly given, possibly exponentially large) directed acyclic graph[3], where at least one node can be found efficiently and, for any given node, at least one neighbour (if such exists) can be found efficiently [6, 11].

This interpretation naturally gives rise to the *standard local search* heuristic: start from an arbitrary initial configuration, and at every iteration perform an (arbitrary[4]) locally improving move until no such move exists anymore. For a fixed pivoting rule, this corresponds to traversing a single path of the transition graph. Due to the definition of PLS membership above, the running time of such a process is thus determined by the *number* of these local search iterations. In the worst case, this amounts to bounding the length of the *longest path* in the transition graph.

It is important to emphasize that, regardless of being a very natural heuristic, standard local search is definitely not the only method for finding local optima: as a matter of fact, we know that there exist local optimization problems that are efficiently solvable via more involved, "centralized" methods (e.g., by using linear programming), but for which standard local search would provably require exponential time (see, e.g., [2]): more precisely, there exist nodes in their transition graphs from which *all* paths have exponential length [23].

To assert the intractability of a problem, the common argumentation is via complexity theory techniques that prove the conditional inexistence of polynomial time algorithms. If a problem is PLS-hard, then unless PLS = P, there is no algorithm that solves it in polynomial time under traditional worst-case analysis. However, since we are studying the performance of local search heuristics, we are in fact interested in specifically *their* running time. Schäffer and Yannakakis [23] introduced the notion of a stronger reduction among problems in PLS, named *tight* PLS*-reduction*, which preserves key structural properties of the initial instance to the target-problem's instance. An important implication of an initial-problem P having a tight PLS-reduction to target-problem Q is the following: for any instance I of P, a path in the transition graph of the corresponding reduced instance J of Q induces a path of I of length no larger than its own. In particular, if for an instance I there exists a starting configuration from which all paths to solutions are exponentially long, this also applies to the reduced instance J. We show that all the problems we study have this property, either by referring to explicit bad instances for the problem at hand, or through (chains of) tight reductions, which we construct if they haven't been established before.

For a more thorough treatment of the complexity of local optimization and the class PLS, the interested reader is referred to the excellent monograph of Yannakakis [26].

2.1 Smoothed Combinatorial Local Search

Under traditional, worst-case algorithmic analysis, the running time of an algorithm for a CLO problem would be evaluated against an adversarially selected cost vector $\boldsymbol{c} = (c_1, \ldots, c_\nu)$. Instead, our goal here is to propose a systematic *smoothed analysis* [25, 21, 20] framework for local optimization. Therefore, for the remainder of this paper we assume that $\boldsymbol{c}$ is not fixed, but drawn randomly from a product distribution. More specifically, each cost coefficient c_i is drawn independently from a continuous probability distribution with density

[3] This is the transition graph mentioned earlier.

[4] Thus, standard local search is essentially a *family* of algorithms; different pivoting rules can give rise to different local search algorithms.

$f_i : [-1,1] \longrightarrow [0,\phi]$, where $\phi \geq \frac{1}{2}$. These distributions can be adversarially selected, but their realizations c_i are not; the running time is then computed *in expectation* with respect to the random cost vector $\boldsymbol{c}$. An efficient algorithm runs in time polynomial in the combinatorial-structure parameters ν, $\bar{\nu}$ and M of the problem, and in the smoothness parameter ϕ. We will sometimes refer to this model as *smoothed CLO*, if we want to give particular emphasis to the fact that we are performing a smoothed running time analysis (as opposed to worst-case analysis).

In that sense, smoothed analysis can be seen as interpolating between two extremes: an average-case analysis setting where all c_i's are drawn i.i.d. from a uniform distribution over $[-1,1]$, derived for $\phi = \frac{1}{2}$; and traditional worst-case analysis that can be derived in the limit, via $\phi \to \infty$, as distributions f_i approximate adversarial, single-point-mass instances.

In this paper we focus on smoothed analysis for standard local search, and so our quantity of interest will be the expected number of improving moves (for any set of adversarially given input distributions) until a local optimum is reached; that is, the expected length of the longest path in the transition graph. This allows us to also avoid some delicate representation issues that are typical of smoothed analysis, and which have to do with how the realizations c_i of *continuous* distributions (which therefore produce irrational numbers almost surely) can be handled as inputs to a Turing-machine-based computational model. For a careful discussion about this topic we point to the papers of Beier and Vöcking [3] or Röglin and Vöcking [22]. In a nutshell, for our purposes it is safe to think of the polynomial-time improving-local-move oracle (from the PLS definition) as having access to real-number arithmetic.

3 Smoothed Analysis of Local Search

In this section we present our first main result, which is a black-box tool for upper-bounding the number of improving moves of standard local search, under smoothed analysis. To achieve this, we first highlight an appropriate underlying structure of CLO problems and identify key parameters that characterize it (see Definition 1). Then, our bounds for standard local search are directly expressed as a function of these parameters (see Theorem 2). As we will demonstrate in Sections 4.2 and 4.3, and [12, Sec. 4.2], for various CLO problems of interest these parameters are well-behaved enough to result in polynomial smoothed running times.

We now introduce some terminology and notation that will be necessary for stating our main result in Theorem 2. Fix an instance of a CLO problem with cost coordinates $[\nu]$ and configurations $\boldsymbol{S}$ (see Section 2), and let $G = (\boldsymbol{S}, E)$ be its neighbourhood graph. A *covering* $(\mathcal{E}, \mathcal{I})$ of this instance consists of a cover $\mathcal{E}$ of the edges of its neighbourhood graph, and a cover $\mathcal{I}$ of its cost coordinates. That is, $\mathcal{E} \subseteq 2^E$ and $\mathcal{I} \subseteq 2^{[\nu]}$ such that $E = \bigcup_{E' \in \mathcal{E}} E'$ and $[\nu] = \bigcup_{I \in \mathcal{I}} I$. We call $\mathcal{E}$ the *transition cover* and $\mathcal{I}$ the *coordinate cover* and they contain *transition clusters* and *coordinate clusters* respectively. Recall that, under our previous discussion (see Section 2), $\mathcal{E}$ can be simply interpreted as covering all potential configuration transitions that can be made by standard local search, clustered appropriately in different groups $T \in \mathcal{E}$. For an arbitrary set of such transitions $T \subseteq E$, we also define its *core* to be the set of coordinates affected by any of the transitions:

$$\mathsf{core}(T) \coloneqq \{ i \in [\nu] \mid \boldsymbol{s}_i^\bullet \neq \boldsymbol{s}_i'^\bullet \text{ for some } (\boldsymbol{s}, \boldsymbol{s}') \in T \},$$

and its *diversity* with respect to a given set of coordinates $I \subseteq [\nu]$ to be the number of different configuration changes when projected to I:

$$\delta_I(T) \coloneqq |\mathsf{range}_I(T)|, \quad \text{where } \mathsf{range}_I(T) \coloneqq \{ \boldsymbol{s}_I^\bullet - \boldsymbol{s}_I'^\bullet \mid (\boldsymbol{s}, \boldsymbol{s}') \in T \}.$$

▶ **Definition 1** ((λ, β, μ)-separable instances)**.** *An instance of a CLO problem is called* (λ, β, μ)-separable *if it has a covering* $(\mathcal{E}, \mathcal{I})$ *with* $|\mathcal{E}| \leq \lambda$*, such that any transition cluster* $T \in \mathcal{E}$*:*

(a) *has a core that can be covered by using* β *many coordinate clusters from the cover* $\mathcal{I}$*; formally, there exists a* $\mathcal{I}_T \subseteq \mathcal{I}$ *with* $|\mathcal{I}_T| \leq \beta$ *such that* $\mathsf{core}(T) \subseteq \bigcup_{I \in \mathcal{I}_T} I$*, and*

(b) *has at most* μ *diversity with respect to all coordinate clusters; formally,* $\max_{I \in \mathcal{I}} \delta_I(T) \leq \mu$.

▶ **Theorem 2.** *On any* (λ, β, μ)*-separable smoothed combinatorial local optimization instance, standard local search terminates after at most*

$$3 \cdot \mu^\beta \lambda \cdot \nu^2 M \log(M+1) \cdot \phi$$

many steps (in expectation).

For the proof of Theorem 2 we will need the following technical lemmas. Their proofs can be found in [12, Appendix A]. We would like to highlight the fact that the structural quantities (λ, β, μ) appear as the expression $\mu^\beta \lambda$ in the statement of the theorem and are the deciding quantities for the running time complexity of the problem, as evidenced in the applications further below. As the choice of the covering is not unique (and often even allows for more than just one *natural* choice), this exact expression helps to explain the quantitative interaction between the properties of the covering.

▶ **Lemma 3.** *Let* $I \subseteq [\nu]$ *be a set of cost coordinates and* $\mathcal{J} \subseteq 2^{[\nu]}$ *be a cover of* I*, i.e.* $I \subseteq \bigcup_{J \in \mathcal{J}} J$*. Then, for any set of transitions* $T \subseteq E$*,*

$$\delta_I(T) \leq \prod_{J \in \mathcal{J}} \delta_J(T).$$

▶ **Lemma 4.** *Fix some* $\phi > 0$ *and let* $\boldsymbol{X} = (X_1, X_2, \ldots, X_m)$ *be a random real vector, where each component* X_i *is drawn independently from a continuous distribution with density* $f_i : \mathbb{R} \longrightarrow [0, \phi]$*. Then, for any nonzero vector* $\boldsymbol{\xi} \in \mathbb{R}^m$ *and any* $\varepsilon \geq 0$*,*

$$\mathrm{Prob}\left[0 \leq \boldsymbol{\xi} \cdot \boldsymbol{X} \leq \varepsilon\right] \leq \min\left(\frac{1}{\|\boldsymbol{\xi}\|_\infty}, \frac{\sqrt{2}}{\|\boldsymbol{\xi}\|_2}\right) \cdot \varepsilon\phi, \tag{2}$$

where $\|\cdot\|_2$ *and* $\|\cdot\|_\infty$ *denote the Euclidean and maximum norms, respectively. For the special*[5] *case where* $\boldsymbol{\xi}$ *is a (nonzero) integer vector we get*

$$\mathrm{Prob}\left[0 \leq \boldsymbol{\xi} \cdot \boldsymbol{X} \leq \varepsilon\right] \leq \varepsilon\phi. \tag{3}$$

Proof of Theorem 2. Fix a (λ, β, μ)-separable smoothed CLO instance, with neighbourhood graph $G = (\boldsymbol{S}, E)$, and let $(\mathcal{E}, \mathcal{I})$ be a covering satisfying the conditions of Definition 1.

We introduce the following notation:

$$\Delta(\boldsymbol{s}, \boldsymbol{s'}) \coloneqq \begin{cases} C(\boldsymbol{s}) - C(\boldsymbol{s'}), & \text{if this is positive,} \\ \infty, & \text{otherwise,} \end{cases}$$

[5] In this paper we will be actually making use of Lemma 4 only via its weaker bound (3), rather than the stronger form (2). This is sufficient for our purposes because, as it turns out, this has an asymptotically negligible effect on our bounds. However, we still choose to state (and prove) Lemma 4 in its full generality, since we expect it to be of potential independent interest for future extensions, especially if one considers more involved structures, or non-integral configurations.

for all $(\boldsymbol{s}, \boldsymbol{s'}) \in E$, and

$$\Delta \coloneqq \min_{(\boldsymbol{s},\boldsymbol{s'})\in E} \Delta(\boldsymbol{s}, \boldsymbol{s'}).$$

Notice that these are random variables, depending on the realizations of the cost vector $\boldsymbol{c}$.

Our first goal is to give a bound on the probability that there exists a local move that improves the cost only by (at most) $\varepsilon > 0$, as a function of this improvement bound ε. If this quantity is sufficiently small, then with high probability, standard local search will achieve improvements more than ε at *every* step, thus resulting in faster convergence. To upper bound this probability $\text{Prob}\left[\Delta \leq \varepsilon\right]$, we first use a union bound over the cover $\mathcal{E}$ of all transitions $(\boldsymbol{s}, \boldsymbol{s'}) \in E$ to get

$$\text{Prob}\left[\Delta \leq \varepsilon\right] = \text{Prob}\left[\bigcup_{(\boldsymbol{s},\boldsymbol{s'})\in E} \left[\Delta(\boldsymbol{s}, \boldsymbol{s'}) \leq \varepsilon\right]\right] \leq \sum_{T\in\mathcal{E}} \text{Prob}\left[\bigcup_{(\boldsymbol{s},\boldsymbol{s'})\in T} \left[\Delta(\boldsymbol{s}, \boldsymbol{s'}) \leq \varepsilon\right]\right]. \quad (4)$$

Next, for a fixed transition cluster $T \in \mathcal{E}$ we can express the inner union of events in (4) as

$$\begin{aligned}\bigcup_{(\boldsymbol{s},\boldsymbol{s'})\in T} \left[\Delta(\boldsymbol{s}, \boldsymbol{s'}) \leq \varepsilon\right] &= \bigcup_{(\boldsymbol{s},\boldsymbol{s'})\in T} \left[0 < \boldsymbol{c}_{\mathsf{core}(T)} \cdot \left(\boldsymbol{s}_{\mathsf{core}(T)} - \boldsymbol{s'}_{\mathsf{core}(T)}\right) \leq \varepsilon\right] \\ &= \bigcup_{\boldsymbol{x}\in\mathsf{range}_{\mathsf{core}(T)}(T)} \left[0 < \boldsymbol{c}_{\mathsf{core}(T)} \cdot \boldsymbol{x} \leq \varepsilon\right]. \quad (5)\end{aligned}$$

By the separability assumption of our CLO instance (see Definition 1), for any cluster $T \in \mathcal{E}$ there exists a subset $\mathcal{I}_T$ of $\mathcal{I}$ with $|\mathcal{I}_T| \leq \beta$ that covers $\mathsf{core}(T)$ such that, additionally, $\max_{J\in\mathcal{I}_T} \delta_J(T) \leq \max_{I\in\mathcal{I}} \delta_I(T) \leq \mu$. So, from Lemma 3 we can deduce that

$$\left|\mathsf{range}_{\mathsf{core}(T)}(T)\right| = \delta_{\mathsf{core}(T)}(T) \leq \prod_{J\in\mathcal{I}_T} \delta_J(T) \leq \mu^\beta.$$

Furthermore, observe that each $\boldsymbol{x} \in \mathsf{range}_{\mathsf{core}(T)}(T)$ is a nonzero integral vector. Thus, applying (3) of Lemma 4 we can derive that, for a fixed $\boldsymbol{x} \in \mathsf{range}_{\mathsf{core}(T)}(T)$, we have

$$\text{Prob}\left[0 < \boldsymbol{c}_{\mathsf{core}(T)} \cdot \boldsymbol{x} \leq \varepsilon\right] \leq \phi\varepsilon.$$

Therefore, using again a union bound, this time on event (5), we can see that

$$\text{Prob}\left[\bigcup_{(\boldsymbol{s},\boldsymbol{s'})\in T} \left[\Delta(\boldsymbol{s}, \boldsymbol{s'}) \leq \varepsilon\right]\right] \leq \sum_{\boldsymbol{x}\in\mathsf{range}_{\mathsf{core}(T)}(T)} \text{Prob}\left[0 < \boldsymbol{c}_{\mathsf{core}(T)} \cdot \boldsymbol{x} \leq \varepsilon\right] \leq \mu^\beta\phi\varepsilon.$$

Plugging this into (4), and using again the separability of our instance, we can finally bound our desired probability

$$\text{Prob}\left[\Delta \leq \varepsilon\right] \leq \lambda\mu^\beta\varepsilon\phi. \quad (6)$$

Now we continue with the second part of the proof, in which we utilize the probability bound in (6) to derive a concrete bound on the expected number of iterations of standard local search. Let $\mathcal{T}$ denote the random variable of that maximum length among all paths in the transition graph of our instance. Then, our goal is to bound $\mathbb{E}\left[\mathcal{T}\right]$.

Recall that the number of different possible cost-part configurations is trivially upper-bounded by $(M+1)^\nu$. Also, at every step of standard local search, the configuration cost is *strictly* decreasing. Thus $\mathcal{T} \leq (M+1)^\nu$ and, furthermore, since the range of configuration costs is within $[-M\nu, M\nu]$, and the minimum cost improvement of any iteration is Δ, we also know that $\mathcal{T} \leq \frac{2M\nu}{\Delta}$ holds. Using these, we get

$$\begin{aligned}
\mathbb{E}[\mathcal{T}] &= \sum_{t=1}^{(M+1)^\nu} \text{Prob}[\mathcal{T} \geq t] \\
&\leq \sum_{t=1}^{(M+1)^\nu} \text{Prob}\left[\Delta \leq \frac{2M\nu}{t}\right] \\
&\leq 2\lambda\mu^\beta\phi M\nu \sum_{t=1}^{(M+1)^\nu} \frac{1}{t}, && \text{due to (6)}, \\
&\leq 2\lambda\mu^\beta\phi M\nu \frac{3}{2}\log(M+1)^\nu, && \text{since } 1+\frac{1}{2}+\cdots+\frac{1}{n} \leq \frac{3}{2}\log n \ \ \forall n \geq 2, \\
&= 3\mu^\beta\lambda\phi\nu^2 M\log(M+1).
\end{aligned}$$

◀

4 Smoothed Analysis for Congestion Games

Congestion games are composed of finite nonempty sets of *players* $\mathcal{N} = [n]$ and *resources* $\mathcal{R}$. Each player $i \in \mathcal{N}$ has a *strategy set* $\Sigma_i \subseteq 2^{\mathcal{R}}$ and each resource $r \in \mathcal{R}$ has a *cost* (or *latency*) *function* $\kappa_r : [n] \longrightarrow \mathbb{R}_{\geq 0}$. Each (pure) *strategy profile* (or *outcome*) $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n) \in \boldsymbol{\Sigma} \coloneqq \Sigma_1 \times \cdots \times \Sigma_n$ induces a *load* on each resource r, equal to the number of players that use it:

$$\ell_r(\boldsymbol{\sigma}) \coloneqq |\{i \in \mathcal{N} \mid r \in \sigma_i\}|.$$

Then, the *cost of player* i is the total cost she experiences from all resources that she is using:

$$C_i(\boldsymbol{\sigma}) \coloneqq \sum_{r \in \sigma_i} \kappa_r(\ell_r(\boldsymbol{\sigma})).$$

An (exact) pure *Nash equilibrium (PNE)* is an outcome $\boldsymbol{\sigma}^*$ from which no player can improve her cost by unilaterally deviating. Formally, for any player $i \in \mathcal{N}$ and any deviation $\sigma_i' \in \Sigma_i$:

$$C_i(\boldsymbol{\sigma}^*) \leq C_i(\sigma_i', \boldsymbol{\sigma}^*_{-i}).$$

Thus, if a strategy profile $\boldsymbol{\sigma}$ is *not* a PNE, there has to exist a player i and a deviation $\sigma_i' \in \Sigma_i$ that reduces her cost, i.e.

$$C_i(\sigma_i', \boldsymbol{\sigma}_{-i}) < C_i(\boldsymbol{\sigma}).$$

Such a strategy σ_i' is then called a *better-response* (of player i with respect to the profile $\boldsymbol{\sigma}$).

By the seminal work of Rosenthal [18] we know that function $\Phi : \boldsymbol{\Sigma} \longrightarrow \mathbb{R}_{\geq 0}$ defined by

$$\Phi(\boldsymbol{\sigma}) = \sum_{r \in \mathcal{R}} \sum_{\ell=1}^{\ell_r(\boldsymbol{\sigma})} \kappa_r(\ell) \tag{7}$$

and commonly referred to as *Rosenthal's potential*, has the property that

$$C_i(\sigma_i', \boldsymbol{\sigma}_{-i}) - C_i(\boldsymbol{\sigma}) = \Phi(\sigma_i', \boldsymbol{\sigma}_{-i}) - \Phi(\boldsymbol{\sigma}) \qquad \forall \boldsymbol{\sigma} \in \boldsymbol{\Sigma}\ \forall i \in \mathcal{N}\ \forall \sigma_i' \in \Sigma_i.$$

In other words, function Φ is an (exact) *potential* [16] of the corresponding congestion game. This implies that PNE of a congestion game correspond *exactly* to the set of *local* minimizers of its Rosenthal's potential (7), meaning that $\Phi(\boldsymbol{\sigma}^*) \leq \Phi(\sigma_i', \boldsymbol{\sigma}^*_{-i})$ for any player i and any deviation $\sigma_i' \in \Sigma_i$. This also immediately establishes the existence of PNE in *all* congestion games, since the potential function $\Phi(\boldsymbol{\sigma})$ can only take finitely many different values.

Depending on the type and representation of the cost functions, different classes of congestion games can arise. Below, we describe three prominent ones which we will focus on in this paper:

- *General.* The cost functions are given explicitly as a list of nonnegative values, one for each possible load on the resource; $(\kappa_r(1), \kappa_r(2), \ldots, \kappa_r(n))$. Notice how in this model we do not impose a monotonicity constraint; this is deliberate, to maintain full generality. If one wants to focus on nondecreasing cost functions (as is many times the case in the literature), then the step-function representation (see below) can easily be used instead.
- *Polynomials (of degree d).* The cost functions are polynomials of maximum degree $d \in \mathbb{N}$ with nonnegative coefficients. More specifically, the cost functions are given implicitly by the coefficients $\{\alpha_{r,j}\}_{r\in\mathcal{R}, j\in[\![d_r]\!]} \subseteq \mathbb{R}_{\geq 0}$, where $d_r \leq d$, via

$$\kappa_r(\ell) = \sum_{j=0}^{d_r} \alpha_{r,j}\ell^j \qquad \text{for all } \ \ell \in [n]. \tag{8}$$

- *Step functions (with d break-points).* The cost functions are nondecreasing, piecewise constant, given by pairs of break-points and value-increases. More specifically, for each resource $r \in \mathcal{R}$ there is a list of break-points $\mathcal{B}_r \subseteq [n]$ and associated jumps $\{\alpha_{r,j}\}_{j\in\mathcal{B}_r}$. We denote the number of break-points of a resource r by $d_r \coloneqq |\mathcal{B}_r|$, and we set $d \coloneqq \max_{r\in\mathcal{R}} d_r$. Then, the cost functions are given via

$$\kappa_r(\ell) = \sum_{j\in\mathcal{B}_r\cap[\ell]} \alpha_{r,j} \qquad \text{for all } \ \ell \in [n]. \tag{9}$$

To the best of our knowledge, no smoothed analysis model has been established so far for congestion games. In this paper, we propose and study the following perturbation semantics for the aforementioned classes:

- *General congestion games.* The costs $\kappa_r(\ell)$ are independently distributed according to densities $f_{r,\ell} : [0,1] \longrightarrow [0,\phi]$, for all $r \in \mathcal{R}$ and $\ell \in [n]$.
- *Polynomial games.* The coefficients $\alpha_{r,j}$ are independently distributed according to densities $f_{r,j} : [0,1] \longrightarrow [0,\phi]$, for all $r \in \mathcal{R}$ and $j \in [\![d_r]\!]$.
- *Step-function games.* The jump increases $\alpha_{r,j}$ are independently distributed according to densities $f_{r,j} : [0,1] \longrightarrow [0,\phi]$, for all $r \in \mathcal{R}$ and $j \in \mathcal{B}_r$. Notice, however, that the break-points $\mathcal{B}_r$ themselves are not subjected to any noise, and they are assumed to be fixed (and adversarially selected).

4.1 Nash Equilibria as Combinatorial Local Optimization Problems

We now show how PNE-Congestion, the problem of finding a pure Nash equilibrium in congestion games, can actually be interpreted as a combinatorial local optimization problem (with respect to our definitions in Section 2.1), for any of the cost models described above. For all models, their randomness semantics translate directly to the respective randomness of cost coefficients in the smoothed CLO problem.

- *General congestion games.* By (7) we know that PNE correspond exactly to local minimizers of the potential function $\Phi(\boldsymbol{\sigma}) = \sum_{r\in\mathcal{R}} \sum_{j=1}^{\ell_r(\boldsymbol{\sigma})} \kappa_r(j)$. Therefore, finding a PNE of a general congestion game can be viewed as a binary ($M = 1$) CLO problem, with cost dimension $\nu = |\mathcal{R}|\, n$ (the cost coordinates are given by $\mathcal{R} \times [n]$), where each strategy profile $\boldsymbol{\sigma}$ is mapped to a cost configuration $\boldsymbol{s}^\bullet = (s_{r,j})_{r\in\mathcal{R}, j\in[n]}$ given by the indicator functions:

$$s_{r,j} = \mathbb{1}\left[j \leq \ell_r(\boldsymbol{\sigma})\right]. \tag{10}$$

The CLO cost coefficients are given by $c_{r,j} = \kappa_r(j)$.

Furthermore, we want to establish a one-to-one correspondence between configurations $\boldsymbol{s}$ of the CLO problem and strategy profiles $\boldsymbol{\sigma}$ of our congestion games where, in particular, the neighbours of $\boldsymbol{s}$ are exactly the configurations corresponding to single-player deviations $\{(\sigma_i', \boldsymbol{\sigma}_{-i})\}_{i\in\mathcal{N}, \sigma_i'\in\Sigma_i}$. In that way, better-responses of the congestion game would correspond exactly to local improvements in the CLO formulation. However, this cannot be achieved by using just the cost part defined above by (10), since in that case, different strategy profiles may end up being mapped to the same cost part configuration (when they induce the same resource loads). To overcome this technical pitfall, we also maintain a non-cost part $\boldsymbol{s}^\circ$, which keeps track of the actual strategies of the players: this can be easily achieved, with only an additional polynomial size burden.[6]

Finally, notice that the neighbourhoods of the CLO problem we created can be explicitly listed and efficiently searched for a better (smaller cost) value: they have a maximum size of $n \cdot \max_{i\in\mathcal{N}} |\Sigma_i|$, which is polynomial in the description of the original congestion game.

- *Polynomial games.* Using (8), Rosenthal's potential (7) can now be written as

$$\Phi(\boldsymbol{\sigma}) = \sum_{r\in\mathcal{R}} \sum_{\ell=1}^{\ell_r(\boldsymbol{\sigma})} \sum_{j=0}^{d_r} \alpha_{r,j}\ell^j = \sum_{r\in\mathcal{R}} \sum_{j=0}^{d_r} \alpha_{r,j} \sum_{\ell=1}^{\ell_r(\boldsymbol{\sigma})} \ell^j = \sum_{r\in\mathcal{R}} \sum_{j=0}^{d_r} \alpha_{r,j}\mathfrak{S}_j(\ell_r(\boldsymbol{\sigma})),$$

where

$$\mathfrak{S}_j(\ell) \coloneqq \sum_{k=1}^{\ell} k^j \leq \ell^{j+1} \leq \ell^{d+1} \leq n^{d+1}$$

for any $\ell \in \mathbb{N}$. This induces a CLO problem with parameters

$$\nu = \sum_{r\in\mathcal{R}} (d_r + 1) \leq |\mathcal{R}|\,(d+1) \qquad \text{and} \qquad M = n^{d+1},$$

with each strategy profile $\boldsymbol{\sigma}$ corresponding to a cost configuration $\boldsymbol{s}^\bullet = (s_{r,j})_{r\in\mathcal{R}, j\in[\![d_r]\!]}$ given by:

$$s_{r,j} = \mathfrak{S}_j(\ell_r(\boldsymbol{\sigma})).$$

The costs coefficients are given by $c_{r,j} = \alpha_{r,j}$. Notice that again all neighbourhoods are efficiently searchable since they have a polynomial maximum size of $n \cdot \max_{i\in\mathcal{N}} |\Sigma_i|$.

- *Step-function games.* Using (9), Rosenthal's potential (7) can now be written as

$$\begin{aligned}\Phi(\boldsymbol{\sigma}) = \sum_{r\in\mathcal{R}} \sum_{\ell=1}^{\ell_r(\boldsymbol{\sigma})} \sum_{j\in\mathcal{B}_r\cap[\ell]} \alpha_{r,j} &= \sum_{r\in\mathcal{R}} \sum_{j\in\mathcal{B}_r\cap[\ell_r(\boldsymbol{\sigma})]} (\ell_r(\boldsymbol{\sigma}) - j + 1)\alpha_{r,j} \\ &= \sum_{r\in\mathcal{R}} \sum_{j\in\mathcal{B}_r} \max(0, \ell_r(\boldsymbol{\sigma}) - j + 1)\alpha_{r,j}.\end{aligned}$$

This induces a CLO problem with dimension

$$\nu = \sum_{r\in\mathcal{R}} |\mathcal{B}_r| = \sum_{r\in\mathcal{R}} d_r \leq |\mathcal{R}|\,d \qquad \text{and} \qquad M = n,$$

with each strategy profile $\boldsymbol{\sigma}$ corresponding to configuration $\boldsymbol{s}^\bullet = (s_{r,j})_{r\in\mathcal{R}, j\in[d_r]}$ given by

$$s_{r,j} = \max(0, \ell_r(\boldsymbol{\sigma}) - \mathcal{B}_r(j) + 1),$$

[6] E.g. we can choose $\bar{\nu} = |\mathcal{N}|$ and let $s_i^\circ \in [|\Sigma_i|]$ be the index of strategy σ_i deployed by player i in profile $\boldsymbol{\sigma}$. To simplify our exposition, in the remaining congestion game classes studied below, we will avoid explicitly discussing these non-cost parts; they are identical to the general congestion game model.

where we use $\mathcal{B}_r(j)$ to denote the j-th break-point[7] of resource r. The costs are given by $c_{r,j} = \alpha_{r,\mathcal{B}_r(j)}$. Again, it is straightforward to see that all neighbourhoods are efficiently searchable.

4.2 Restrained Congestion Games

Although congestion games are guaranteed to have (at least one) PNE, the computational problem of actually finding one is considered hard; as a matter of fact, the problem PNE-Congestion is one of the most prominent PLS-complete problems. Our goal in this section is to investigate whether this computational barrier can be bypassed, under the lens of the more optimistic complexity model of smoothed analysis.

To achieve this, we establish an upper bound (Theorem 6) on the expected number of better-responses that need to be performed until a PNE is found in a congestion game, as a function of a critical structural parameter of its action space that we identify (see Definition 5). Then, we can deduce that for congestion games in which this parameter is appropriately bounded, the smoothed running time becomes tractable (Corollary 7). At the same time, we show how such a restriction does not make the problem trivially tractable, by proving that PNE-Congestion remains PLS-complete even for this subclass of games (Theorem 8).

▶ **Definition 5** (Restrained Congestion Games). *A congestion game will be called* B-restrained, *where* $B \in \mathbb{N}$, *if the maximum number of resources changed by any single-player deviation is at most* B. *Formally,*

$$\max_{i \in \mathcal{N}} \max_{\sigma, \sigma' \in \Sigma_i} |\sigma \triangle \sigma'| \leq B,$$

where $\triangle$ *denotes the standard symmetric difference operator (recall that* σ, σ' *are subsets of resources).*

▶ **Theorem 6.** *Consider a* B*-restrained* n*-player congestion game, under any of the smoothed-analysis models described in Section 4 (namely general, polynomial, or step-function latencies), with maximum density parameter* ϕ. *Then, performing any better-response dynamics, starting from an arbitrary strategy profile, converge to an (exact) PNE of the congestion game in an expected number of iterations that is bounded by*

- $\mathcal{O}\left(n^{B+3}k^2m^2\phi\right)$ *for general games,*
- $\mathcal{O}\left(n^{B+d+2}\log(n)(d+1)^3k^2m^2\phi\right)$ *for polynomial games with degree at most* d, *and*
- $\mathcal{O}\left((d+1)^{B+2}n^2\log(n)k^2m^2\phi\right)$ *for step-function games with at most* d *break-points,*

where $m = |\mathcal{R}|$ *is the number of resources and* $k = \max_{i \in \mathcal{N}} |\Sigma_i|$ *is the maximum strategy set size.*

Proof. In Section 4.1 we already showed how congestion games can be interpreted as CLO problems. In particular, we established a one-to-one correspondence between better-responses of the players to local improvements of the CLO cost objective (which corresponds to the value of Rosenthal's potential). Using this interpretation, we can now make use of our main black-box tool from Section 3: bounding the expected number of local search steps will induce the same bound in the expected iterations of better-response dynamics in the original congestion game. Therefore, the gist of our proof is to construct coverings $(\mathcal{E}, \mathcal{I})$ so that the induced CLO problem can be shown to be (λ, β, μ)-separable (see Definition 1) for parameters with appropriately small magnitude.

[7] That is, if $\mathcal{B}_r = \{b_1, b_2, \ldots, b_{d_r}\} \subseteq [n]$, then $\mathcal{B}_r(j) = b_j$, for any $j \in [d_r]$.

We start by establishing some properties that will be shared across all three different cost models. Fix a congestion game and its corresponding CLO problem, as described in Section 4.1.[8] For convenience, we denote by $k_i \coloneqq |\Sigma_i|$ the size of the strategy set of a player i, and let $k \coloneqq \max_{i\in\mathcal{N}} k_i$. Also, in all of the models we have a similar index structure in the cost part; the cost-part is given by $(s_{r,j})_{r\in\mathcal{R},j\in J_r}$, where J_r depends on the cost function model ($J_r = [n]$ for general, $J_r = [\![d_r]\!]$ for polynomial, and $J_r = [d_r]$ for step-function costs).

The transition cover $\mathcal{E}$ is constructed from clusters that collect all edges in the neighbourhood graph (of the CLO problem) that correspond to a fixed deviation of a player, regardless of the configuration of the remaining players. Formally, we let

$$\mathcal{E} \coloneqq \{E(i,\sigma_i,\sigma_i') \mid i\in\mathcal{N},\ \sigma_i,\sigma_i'\in\Sigma_i\}, \quad \text{where}$$
$$E(i,\sigma_i,\sigma_i') \coloneqq \{(\boldsymbol{s}(\sigma_i,\boldsymbol{\sigma}_{-i}),\boldsymbol{s}(\sigma_i',\boldsymbol{\sigma}_{-i}))\in E \mid \boldsymbol{\sigma}_{-i}\in\boldsymbol{\Sigma}_{-i}\},$$

and $\boldsymbol{s}(\boldsymbol{\sigma})$ is used to denote the CLO configuration corresponding to strategy profile $\boldsymbol{\sigma}$ in the congestion game. Now we immediately get the bound

$$|\mathcal{E}| \le nk(k-1),$$

which will be used as the value for our separability parameter λ (see Definition 1).

Next, for the coordinate cover $\mathcal{I}$, we cluster the indices with respect to each resource, i.e. we choose

$$\mathcal{I} \coloneqq \{I_r \mid r\in\mathcal{R}\}, \quad \text{where} \quad I_r = \{(r,j) \mid j\in J_r\}.$$

In congestion games, a deviation $\sigma_i \to \sigma_i'$ only affects the resources $r \in \sigma_i \triangle \sigma_i'$. Their loads are changed to increase by 1 for $r \in \sigma_i' \setminus \sigma_i$ and decrease by 1 for $r \in \sigma_i \setminus \sigma_i'$; the load of all other resources does not change. Our choice of the cover $\mathcal{I}$, therefore, will allow us to settle β for all models due to the B-restrain assumption on the size of $\sigma_i \triangle \sigma_i'$. In more detail, recall that the cost parts of a configuration depend only on the loads of the resources, thus all components associated with resources $r \notin \sigma_i \triangle \sigma_i'$ (specifically, the components with coordinates I_r) remain unchanged during the transition, since the load of r does not change either. The fact that the size of those sets $\sigma_i \triangle \sigma_i'$ is universally bounded, by assumption, will let us use $\beta = B$ as a separability parameter (see Definition 1).

The remaining parameter μ depends on the structure of the configurations in the cost part. Again, we emphasize that for all cost models in Section 4.1, the sub-configuration $\boldsymbol{s}_{I_r}(\boldsymbol{\sigma})$, which comprises all components of $\boldsymbol{s}(\boldsymbol{\sigma})$ that correspond to a resource r, depends only on the load $\ell_r(\boldsymbol{\sigma})$ of resource r (under strategy profile $\boldsymbol{\sigma}$). We can therefore represent it as a function $\boldsymbol{h}_r : [\![n]\!] \to [\![M]\!]^{J_r}$, i.e., $\boldsymbol{s}_{I_r}(\boldsymbol{\sigma}) = \boldsymbol{h}_r(\ell_r(\boldsymbol{\sigma}))$. For ease of notation we write $\boldsymbol{s}_r = \boldsymbol{s}_{I_r}$ in the following.

To discuss $\mathsf{range}_{I_r}(E(i,\sigma_i,\sigma_i'))$, we need to consider the configuration changes given by

$$\boldsymbol{s}_r(\boldsymbol{\sigma}) - \boldsymbol{s}_r(\boldsymbol{\sigma}') = \begin{cases} \boldsymbol{h}_r(\ell_r(\boldsymbol{\sigma})) - \boldsymbol{h}_r(\ell_r(\sigma')) = \boldsymbol{h}_r(\ell_r(\boldsymbol{\sigma})) - \boldsymbol{h}_r(\ell_r(\boldsymbol{\sigma})+1), & r\in\sigma_i'\setminus\sigma_i, \\ \boldsymbol{h}_r(\ell_r(\boldsymbol{\sigma})) - \boldsymbol{h}_r(\ell_r(\sigma')) = \boldsymbol{h}_r(\ell_r(\boldsymbol{\sigma})) - \boldsymbol{h}_r(\ell_r(\boldsymbol{\sigma})-1), & r\in\sigma_i\setminus\sigma_i'. \end{cases}$$

The only variable in this expression is therefore the initial load $\ell_r(\boldsymbol{\sigma})$. In either case, there are n possible initial loads for each resource[9] and therefore also at most n difference-vectors within $\mathsf{range}_{I_r}(E(i,\sigma_i,\sigma_i'))$; thus $\mu \le n$. In the following, we will discuss the actual configuration difference structure for each model and whether we can improve μ over this basic bound.

[8] We will use standard notation for the various components and parameters of the game and the CLO problem, as introduced above in Section 4.

[9] Note that although there are $n+1$ different loads $0,\dots,n$, a load-increasing resource cannot already have load n and a decreasing one cannot have 0.

- *General.* The CLO representation follows a binary model with $M = 1$ and $\nu = mn$: each resource r corresponds to components $s_{r,j}$, $j = 1, \ldots, n$ (the indices from I_r), with a value of $s_{r,j} = 1$ for $j \leq \ell_r(\boldsymbol{\sigma})$ and $s_{r,j} = 0$ otherwise. The function $\boldsymbol{h}_r : [\![n]\!] \longrightarrow \{0,1\}^n$ is thus given by

$$(\boldsymbol{h}_r(\ell))_j = \mathbb{1}[j \leq \ell], \qquad \text{for all } \; j \in [n].$$

  In particular, the vectors $\boldsymbol{\delta} \in \mathsf{range}_{I_r}(E(i, \sigma_i, \sigma_i'))$ are a result of moving the rightmost entry with value 1 within the vector $\boldsymbol{s}_r$ to the next larger or smaller load in the configuration, i.e. they are given by: $\delta_{\ell_r(\boldsymbol{\sigma})+1} = -1$ for $r \in \sigma_i' \setminus \sigma_i$; $\delta_{\ell_r(\boldsymbol{\sigma})} = +1$ for $r \in \sigma_i \setminus \sigma_i'$; and zeroes elsewhere. Since there are n many such vectors for every resource r, we cannot improve over $\mu = n$.
  In this case, thus, Theorem 2 yields an expected running time of at most

$$3 \cdot n^B nk(k-1) \cdot (mn)^2 \cdot \phi = \mathcal{O}\left(n^{B+3} k^2 m^2 \phi\right).$$

- *Polynomial games.* For this model, each (cost-part) configuration component $s_{r,j}$ is given by accumulated monomials $\mathfrak{S}_j(\ell_r(\boldsymbol{\sigma}))$, for degrees $j = 0, 1, \ldots, d_r$. Thus, $\nu = m(d+1)$, and also $M = n^{d+1}$, in order to capture all possible values of these functions. Therefore we now get $\boldsymbol{h}_r : [\![n]\!] \longrightarrow [\![n^{d+1}]\!]^{d_r+1}$ with

$$\boldsymbol{h}_r(\ell) = (\mathfrak{S}_0(\ell), \ldots, \mathfrak{S}_{d_r}(\ell))$$

  for the configuration component of I_r. Again, we cannot do better than the basic bound, so we use $\mu = n$.
  Similarly to the previous case for general latency functions, using Theorem 2 we can bound the expected number of better-response iterations by

$$3 \cdot n^B nk(k-1) \cdot (m(d+1))^2 n^{d+1} \log(n^{d+1}) \cdot \phi = \mathcal{O}\left(n^{B+d+2} \log(n)(d+1)^3 k^2 m^2 \phi\right).$$

- *Step-function games.* The configuration mapping is now given by

$$s_{r,j} = \max\left(0, \ell_r(\boldsymbol{\sigma}) - \mathcal{B}_r(j) + 1\right)$$

  and $M = n, \nu = m \cdot d$. Therefore, the (cost-part) configuration components of a resource r are represented by the function $\boldsymbol{h}_r : [\![n]\!] \longrightarrow [\![n]\!]^{d_r}$ given by

$$\boldsymbol{h}_r(\ell) = (\max(0, \ell - \mathcal{B}_r(1) + 1), \ldots, \max(0, \ell - \mathcal{B}_r(d_r) + 1)).$$

  In this case, we can do even better than the basic $\mu = n$ bound. We investigate the structure of the differences with respect to each coordinate $j \in J_r$, for an increase in the load of a resource r, i.e. for $r \in \sigma_i' \setminus \sigma_i$ (the decreasing case follows analogously):

$$\begin{aligned}(\boldsymbol{h}_r(\ell) - \boldsymbol{h}_r(\ell+1))_j &= \max\left(0, \ell - \mathcal{B}_r(j) + 1\right) - \max\left(0, (\ell+1) - \mathcal{B}_r(j) + 1\right) \\ &= \begin{cases} 0 - 0 = 0, & \text{if } \; \ell < \mathcal{B}_r(j) - 1, \\ -\left((\ell+1) - \mathcal{B}_r(j) + 1 - 0\right) = -1, & \text{if } \; \ell = \mathcal{B}_r(j) - 1, \\ (\ell - \mathcal{B}_r(j) + 1) - ((\ell+1) - \mathcal{B}_r(j) + 1) = -1, & \text{if } \; \ell > \mathcal{B}_r(j) - 1. \end{cases}\end{aligned}$$

  Because the jump points $\mathcal{B}_r(j)$ are ordered increasingly with respect to j, the resulting vectors $\boldsymbol{h}_r(\ell) - \boldsymbol{h}_r(\ell+1)$ are of the form $(-1, \ldots, -1, 0, \ldots, 0)$, including the zero vector. Therefore, with respect to the coordinate cluster I_r, there are at most $d_r + 1$ possible distinct vectors for the (cost-part) configuration differences, and we choose $\mu = \max_r d_r + 1 = d + 1$. By Theorem 2 we can thus bound the expected number of iterations by

$$3 \cdot (d+1)^B nk(k-1) \cdot (m \cdot d)^2 n \log(n) \cdot \phi = \mathcal{O}\left((d+1)^{B+2} n^2 \log(n) k^2 m^2 \phi\right). \qquad \blacktriangleleft$$

An immediate corollary of Theorem 6 is that, when congestion games are sufficiently restrained, PNE can be found efficiently via better-response dynamics. In what follows, by a *constantly-* and *polylogarithmically*-restrained congestion game we mean a B-restrained game with $B \in \mathcal{O}(1)$ and $B \in \mathcal{O}(\log^c N)$ for some constant $c > 0$, respectively, where N is the size of the input.

▶ **Corollary 7.** *Better-response dynamics terminate in polynomial smoothed time for the class of constantly-restrained congestion games, and in quasipolynomial smoothed time for polylogarithmically-restrained games, under any cost model. Under the step-function cost model, in $\mathcal{O}(\log N)$-restrained congestion games with a constant number of steps d, better-response dynamics terminate in polynomial smoothed time.*

We now show that the class of constantly-restrained congestion games, for which Corollary 7 provides efficient smooth running time, constitutes a computationally meaningful restriction of arbitrary congestion games, since they can still encode the PLS-completeness of the original problem. The hardness is a straightforward adaptation of that in [19], and so the proof of Theorem 8 is deferred to [12, Appendix B] for completeness. It makes use of the fact that LocalMaxCut is PLS-complete even for constant-degree graphs.

▶ **Theorem 8.** *The problem of computing a PNE of a constantly-restrained congestion game is* PLS*-complete, for all the input models described in Section 4 (namely general, polynomial, or step-function cost representations).*

In addition to the conditional intractability that PLS-hardness implies for these families of congestion games, we show an unconditional lower bound on the worst-case running time of the standard local search algorithm of the problem. We do this by using the notion of a *tight* PLS*-reduction*, as discussed in Section 2. Since we reduce from the problem LocalMaxCut-d for $d \geq 5$ (defined in [12, Appendix B]), which admits a configuration starting from which the standard local search algorithm needs exponentially many iterations (see discussion after proof of [12, Theorem 5.10]), our tight PLS-reduction implies that standard local search of our families of congestion games – under *any* pivoting rule – takes exponential time in the worst case.

4.3 Network Congestion Games

An interesting, and very well studied, variation on the vanilla representation model for congestion games (which we presented at the start of Section 4 above) is that of *network* congestion games. In such games, the strategy sets Σ_i of the players are not given explicitly in the input, but implicitly via an underlying directed graph G whose edges constitute the resources of the game. More precisely, for each player i we are given an origin o_i and a destination d_i node of G. Then, Σ_i is defined (implicitly) as the set of all (simple) $o_i \to d_i$ paths in G. Importantly, this means that now players may have exponentially many strategies available to them.

A critical implication is that Σ_i cannot be searched *exhaustively* for better-responses. However, a better-response can still be found efficiently: keeping all other players fixed, a minimum-cost strategy of player i is a shortest path on graph G with edge costs equal to the cost $c_r(\ell_r(\boldsymbol{\sigma}_{-i}) + 1)$ of an edge/resource r when used by player i. This means that actually a polynomial-time *best*-response oracle does exist. This immediately places network congestion games in the complexity class PLS (since neighbourhoods can be searched efficiently for a local cost improvement; see our discussion in Section 2) and thus it constitutes a valid CLO

problem (via a similar interpretation as we did for general congestion games in Section 4.1). To emphasize this, we will refer to these "canonical" best-response dynamics of network congestion games as *shortest-path* dynamics.

Finding PNE of network congestion games remains a PLS-complete problem [2]. Given the prominence of these games, both in the theoretical and applied literature, in this section we want to identify conditions under which network congestion games inherit the desirable properties of their general counterparts that allow them to be tractable under smoothed analysis. More precisely, can our positive result from Theorem 6 be applied to network congestion games in a straightforward way? And which structural parameters are now relevant for the running time bound? At the same time, the network congestion game instances that allow for efficient smoothed solutions should still be interesting enough to remain PLS-hard under traditional worst-case analysis. We introduce the following family of network congestion games that are defined by two parameters.

▶ **Definition 9** (Compact Network Congestion Games). *For A, B positive integers, a network congestion game is called* (A, B)-compact *if (a) each player has at most A different best-response strategies, and (b) all such strategies are paths of length at most B. Formally, there exist strategy sets $\Sigma_i^* \subseteq \Sigma_i$, such that:*
(a) $|\Sigma_i^*| \leq A$ *for all* $i \in \mathcal{N}$ *and* $\operatorname{argmin}_{\sigma_i \in \Sigma_i} C_i(\sigma_i, \boldsymbol{\sigma}_{-i}) \subseteq \Sigma_i^*$ *for all* $i \in \mathcal{N}, \boldsymbol{\sigma}_{-i} \in \boldsymbol{\Sigma}_{-i}$, *and*
(b) $|\sigma_i| \leq B$ *for all* $i \in \mathcal{N}, \sigma_i \in \Sigma_i^*$.

Property (b) above will serve the purpose of imposing the restraint condition (see Definition 5) needed to deploy Theorem 6. Property (a) will help us constrain the exponentiality of the strategy space of network games, in order to be able to handle them using tools designed for general games. Intuitively, this property can be related to classical vehicle routing settings in which players can a priori exclude unreasonable detours or paths that involve a road with a construction site with large delay. Both properties are illuminated within the proof of our following positive result for network congestion games.

▶ **Theorem 10.** *Consider an (A, B)-compact n-player network congestion game, under any of the smoothed-analysis models described in Section 4 (namely general, polynomial, or step-function latencies), with maximum density parameter ϕ. Then, performing shortest-path dynamics, starting from an arbitrary strategy profile, converges to a PNE of the game in an expected number of iterations that is polynomial in ϕ, $(d+1)^B$, A, and the description of the game, where the parameter d depends on the cost function representation. In particular: for general latencies, $(d+1)$ can be replaced by n; for polynomial latencies, d is the maximum degree; and for step-functions, d is the maximum number of break-points.*

An immediate consequence of Theorem 10 is that (analogously to Corollary 7 for general congestion games) in (A, B)-compact network games with sufficiently small parameters A, B, a PNE can be found efficiently under smoothness.

▶ **Corollary 11.** *Let N be the size of the input, and A be a polynomial in N. Shortest-path dynamics on (A, B)-compact network congestion games under any cost model terminate in polynomial smoothed time when $B \in \mathcal{O}(1)$ and in quasipolynomial smoothed time when $B \in \mathcal{O}(\log^c N)$ for some constant $c > 0$. Under the step-function cost model, when A is a polynomial in N, $B \in \mathcal{O}(\log N)$, and the number of steps d is constant, shortest-path dynamics terminate in polynomial smoothed time.*

The following hardness result establishes that such games, even for $A, B \in \mathcal{O}(1)$, are PLS-hard. The proof is deferred to [12, Appendix C]. It is based on the reduction constructed by Ackermann, Röglin, and Vöcking [2], with special care taken in order to incorporate constant-length paths that can be established by making use of the fact that LocalMaxCut is PLS-complete even for constant degree graphs.

▶ **Theorem 12.** *The problem of computing a PNE of an (A, B)-compact network congestion game is* PLS*-complete, even for $A, B \in \mathcal{O}(1)$, for all the input models described in Section 4 (namely general, polynomial, or step-function latencies).*

Similarly to our PLS-hardness reduction of Theorem 8, the above theorem's PLS-reduction is tight, in the sense of [23] (see Section 2). The chain of tight PLS-reductions that leads to Network Congestion Games starts from LocalMaxCut with maximum degree 5, and includes Congestion Games. As we discuss after the proof of [12, Theorem 5.10], in such LocalMaxCut instances there is a starting configuration from which all improvement sequences of standard local search have exponential length in the worst case. Therefore, shortest-path dynamics on our family of network congestion games – under *any* pivoting rule – need exponentially many iterations in the worst case. In contrast, our Theorem 10 and Corollary 11 show that under smoothness, even for significantly wider instance families that include these problematic cases, shortest-path dynamics terminate after polynomially many steps in expectation.

References

1 Emile Aarts and Jan Karel Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley, 1997.

2 Heiner Ackermann, Heiko Röglin, and Berthold Vöcking. On the impact of combinatorial structure on congestion games. *Journal of the ACM*, 55(6):1–22, 2008. doi:10.1145/1455248.1455249.

3 Rene Beier and Berthold Vöcking. Typical properties of winners and losers in discrete optimization. *SIAM Journal on Computing*, 35(4):855–881, 2006. doi:10.1137/s0097539705447268.

4 Shant Boodaghians, Rucha Kulkarni, and Ruta Mehta. Smoothed efficient algorithms and reductions for network coordination games. In *11th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 151 of *LIPIcs*, pages 73:1–73:15, 2020. doi:10.4230/LIPIcs.ITCS.2020.73.

5 Barun Chandra, Howard Karloff, and Craig Tovey. New results on the old k-opt algorithm for the traveling salesman problem. *SIAM Journal on Computing*, 28(6):1998–2029, 1999. doi:10.1137/s0097539793251244.

6 Constantinos Daskalakis and Christos Papadimitriou. Continuous local search. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2011. doi:10.1137/1.9781611973082.62.

7 Robert Elsässer and Tobias Tscheuschner. Settling the complexity of local max-cut (almost) completely. In Luca Aceto, Monika Henzinger, and Jiří Sgall, editors, *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 171–182, 2011. doi:10.1007/978-3-642-22006-7_15.

8 Matthias Englert, Heiko Röglin, and Berthold Vöcking. Smoothed analysis of the 2-opt algorithm for the general TSP. *ACM Transactions on Algorithms*, 13(1):1–15, 2016. doi:10.1145/2972953.

9 Michael Etscheid and Heiko Röglin. Smoothed analysis of local search for the maximum-cut problem. *ACM Transactions on Algorithms*, 13(2):25:1–25:12, 2017. doi:10.1145/3011870.

10 Alex Fabrikant, Christos Papadimitriou, and Kunal Talwar. The complexity of pure nash equilibria. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 604–612, 2004. doi:10.1145/1007352.1007445.

11 John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *Journal of Computer and System Sciences*, 114:1–35, 2020. doi:10.1016/j.jcss.2020.05.007.

12 Yiannis Giannakopoulos, Alexander Grosz, and Themistoklis Melissourgos. On the smoothed complexity of combinatorial local search, 2022. arXiv:2211.07547v3.

13 David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988. doi:10.1016/0022-0000(88)90046-3.

14 Mark W. Krentel. Structure in locally optimal solutions. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 216–221, 1989. doi:10.1109/SFCS.1989.63481.

15 Wil Michiels, Jan Korst, and Emile Aarts. *Theoretical Aspects of Local Search*. Springer, 2007. doi:10.1007/978-3-540-35854-1.

16 Dov Monderer and Lloyd S. Shapley. Potential games. *Games and Economic Behavior*, 14(1):124–143, 1996. doi:10.1006/game.1996.0044.

17 James B. Orlin, Abraham P. Punnen, and Andreas S. Schulz. Approximate local search in combinatorial optimization. *SIAM Journal on Computing*, 33(5):1201–1214, 2004. doi:10.1137/s0097539703431007.

18 Robert W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2(1):65–67, 1973. doi:10.1007/BF01737559.

19 Tim Roughgarden. *Pure Nash Equilibria and PLS-Completeness*, pages 261–278. Cambridge University Press, 2016. doi:10.1017/CBO9781316779309.020.

20 Tim Roughgarden, editor. *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 2021. doi:10.1017/9781108637435.

21 Heiko Röglin. *The Complexity of Nash Equilibria, Local Optima, and Pareto-Optimal Solutions*. PhD thesis, RWTH Aachen University, 2008. URL: https://publications.rwth-aachen.de/record/50046.

22 Heiko Röglin and Berthold Vöcking. Smoothed analysis of integer programming. *Mathematical Programming*, 110(1):21–56, 2007. doi:10.1007/s10107-006-0055-7.

23 Alejandro A. Schäffer and Mihalis Yannakakis. Simple local search problems that are hard to solve. *SIAM J. Comput.*, 20(1):56–87, 1991. doi:10.1137/0220004.

24 Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms. *Journal of the ACM*, 51(3):385–463, 2004. doi:10.1145/990308.990310.

25 Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis: An attempt to explain the behavior of algorithms in practice. *Commun. ACM*, 52(10):76–84, 2009. doi:10.1145/1562764.1562785.

26 Mihalis Yannakakis. Computational complexity. In Emile Aarts and Jan Karel Lenstra, editors, *Local Search in Combinatorial Optimization*, chapter 2. John Wiley, 1997.

A Characterization of Complexity in Public Goods Games

Matan Gilboa
University of Oxford, UK

Abstract

We complete the characterization of the computational complexity of equilibrium in public goods games on graphs. In this model, each vertex represents an agent deciding whether to produce a public good, with utility defined by a "best-response pattern" determining the best response to any number of productive neighbors. We prove that the equilibrium problem is NP-complete for every finite non-monotone best-response pattern. This answers the open problem of [Gilboa and Nisan, 2022], and completes the answer to a question raised by [Papadimitriou and Peng, 2021], for all finite best-response patterns.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory; Theory of computation → Exact and approximate computation of equilibria; Theory of computation → Problems, reductions and completeness

Keywords and phrases Nash Equilibrium, Public Goods, Computational Complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.73

Category Track A: Algorithms, Complexity and Games

Related Version *arXiv Version*: `https://arxiv.org/abs/2301.11580`

Acknowledgements I would like to thank Noam Nisan for many useful conversations, and for suggesting the Copy Gadget in the proof of Theorem 3.2. I would like to thank Roy Gilboa for many useful conversations, and for adjusting the Copy Gadget in the proof of Theorem 3.2. I would like to thank Noam Nisan for communicating to me the alternative solution to the monotone case (see footnote 2), which was suggested by Sigal Oren. I would like to thank the anonymous ICALP reviewers for their helpful feedback.

1 Introduction

Public goods games describe scenarios where multiple agents face a decision of whether or not to produce some "good", such that producing this good benefits not only themselves, but also other (though not necessarily all) agents. Typically, we consider the good to be costly to produce, and therefore an agent might choose not to produce it, depending on the actions of the agents that affect her. This type of social scenarios can be found in various real-life examples, such as vaccination efforts (an individual pays some personal cost for being vaccinated but she and other people in her proximity gain from it) and research efforts (a research requires many resources, but the researcher benefits from the results along with other researchers in similar areas). As is common in the literature, to model this we use an undirected graph, where each node represents an agent and an edge between two nodes captures the fact that these nodes directly affect one another by their strategy. As in [4, 6, 7, 9, 10], in our model the utility of an agent is completely determined by the number of productive neighbors she has, as well as by her own action. We focus on a specific version of the game which has the following characteristics. Firstly, our strategy space is binary, i.e. an agent can only choose whether or not to produce the good, rather than choose a quantity (we call an agent who produces the good a *productive* agent); secondly, our game is *fully-homogeneous*, meaning that all agents share the same utility function and cost of

producing the good; and thirdly, our game is *strict*, which means that an agent has a single best response to any number of productive neighbors she might have (i.e. we do not allow indifference between the actions).

The game is formally defined by some fixed cost c of producing the good, and by some "social" function $X(s_i, n_i)$, which takes into account the boolean strategy of agent i and the number of productive neighbors she has (marked as s_i and n_i respectively), and outputs a number representing how much the agent gains. The utility u_i of agent i is then given by the social function $X(s_i, n_i)$, reduced by the cost c if the agent produces the good, i.e. $u_i(s_i, n_i) = X(s_i, n_i) - c \cdot s_i$. However, since any number of productive neighbors yields a unique best response (i.e. the game is *strict*), we can capture the essence of the utility function and the cost using what we call (as in [3]), a Best-Response Pattern $T : \mathbb{N} \to \{0, 1\}$. We think of the Best-Response Pattern as a boolean vector in which the k^{th} entry represents the best response to exactly k productive neighbors. We are interested in the problem of determining the existence of a non-trivial pure Nash equilibrium in these games, which is defined as follows.

Equilibrium decision problem in a public goods game. For a fixed Best-Response Pattern $T : \mathbb{N} \to \{0, 1\}$, and with an undirected graph $G = (V, E)$ given as input, determine whether there exists a pure non-trivial Nash equilibrium of the public goods game defined by T on G, i.e. an assignment $s : V \to \{0, 1\}$ that is not all 0, such that for every $1 \leq i \leq |V|$ we have that

$$s_i = T[\sum_{j \in N(i)} s_j],$$

where $N(i)$ is the set of neighbors of agent i.

The first Best-Response Pattern for which this problem was studied was the so-called Best-Shot pattern (where an agent's best response is to produce the good only if she has no productive neighbors, namely $T = [1, 0, 0, 0, ...]$), which was shown in [1] to have a pure Nash equilibrium in any graph. In [1], they also show algorithmic results for "convex" patterns, which are monotonically increasing (best response is 1 if you have at least k productive neighbors). The question of characterizing the complexity of this problem for all possible patterns was first raised in [7], where they manage to fully answer an equivalent problem on directed graphs: They show tractability for the All-1 pattern, the infinite alternating $[1, 0, 1, 0, ...]$ pattern, and all patterns beginning with 0, and NP-completeness for all other patterns. The open question on undirected graphs was then partially answered in [3], where an efficient algorithm is shown for the pattern $[1, 1, 0, 0, 0, ...]$, and NP-completeness is established for several classes of non-monotone patterns: Those beginning with 0 or 11, or have a prefix of the form $1, 0, 0, ..., 0, 1, 1$. There have been several studies concerning other versions of this problem as well. In [9], the general version of this problem (where the pattern is part of the input rather than being fixed) was shown to be NP-complete when removing the strictness assumption, (i.e. allowing indifference between actions, such that both 0 and 1 are best responses in certain cases) [1]. In [10], NP-completeness is shown for the general version of the problem in the heterogeneous public goods game, in which the utility function varies between agents. In [4], they show NP-completeness of the equilibrium problem when restricting the equilibrium to have at least k productive agents, or at least some specific subset of agents. In [6], the parameterized complexity of the equilibrium problem is studied, for a number of parameters of the graph on which the game is defined.

[1] The paper [11] had an earlier version [10] which presented a proof for this case as well, but an error in the proof was pointed out in [9], who then also provided an alternative proof.

In [3], two open problems are suggested regarding the two following patterns: $T_1 = [1, 1, 1, 0, 0, 0, ...]$, $T_2 = [1, 0, 1, 0, 0, 0, ...]$. T_1 has been recently solved in [5], where they show that all monotonically decreasing patterns can be viewed as potential games, and thus always have a pure Nash equilibrium[2].

There are various instances where non-monotonic patterns are of interest. For example, consider work that necessitates collaboration from n agents or a financial effort that is irrelevant if too few agents contribute but if too many do so it becomes redundant from an agent's perspective. Our main contribution is completing the characterization of the equilibrium decision problem for all finite patterns, by showing that for all non-monotone patterns the problem is NP-complete.

Theorem. For any Best-Response Pattern that is non-monotone and finite (i.e., has a finite number of entries with value 1), the equilibrium decision problem in a public goods game is NP-complete (under Turing reductions).

The first step along this way was to prove NP-completeness for the above pattern T_2 (which we call the 0-Or-2-Neighbors pattern), namely the second open problem by [3]. An alternative proof to this specific problem was obtained independently and concurrently in [5].

We note that we only focus on finite patterns, which we believe to be more applicable to real-life problems that can be modeled by this game. Nonetheless, we find the characterization of all infinite patterns to be of interest, and this topic remains open, though some initial results can be found in Corollary 3.9. Another interesting open problem is to obtain a similar characterization for the non-strict version of the game, where agents are allowed to be indifferent between the two possible actions.

The rest of this paper is organized as follows. In Section 2 we introduce the formal model and some relevant definitions. We then set out to show hardness of all remaining patterns, dividing them into classes. In Section 3 we present a solution for the open question from [3], showing hardness of the 0-Or-2-Neighbors Best Response Pattern, and expanding the result to a larger sub-class of patterns that begin with 1,0,1. In Section 4 we show hardness of all patterns beginning with 1,0,0 (where we also have a slightly more subtle division into sub-classes), and in Section 5 we show hardness of all patterns beginning with 1,0,1 that were not covered in Section 3, thus completing the characterization for all finite patterns. The outline of this paper is also depicted[3] in Figure 1.

2 Model and Definitions

A *Public Goods Game* (PGG) is defined on an undirected graph $G = (V, E)$, $V = \{v_1, ..., v_n\}$, where each node represents an agent. The strategy space, which is identical for all agents, is $S = \{0, 1\}$, where 1 represents producing the good and 0 represents not producing it. The utility of node v_i (which is assumed to be the same for all agents) is completely determined by the number of productive neighbors v_i has, as well as by v_i's own strategy. Moreover, our model is restricted to utility functions where an agent always has a single best response to the strategies of its neighbors, i.e. there is no indifference between actions in the game. Therefore, rather than defining a PGG with an explicit utility function and cost for producing

[2] Alternatively, known results about k-Dominating and k-Independent sets (Theorem 19 in [2]) can be used to prove this.

[3] Some patterns which start with 1,0 were solved in [3], though for simplicity we omit them from Figure 1.

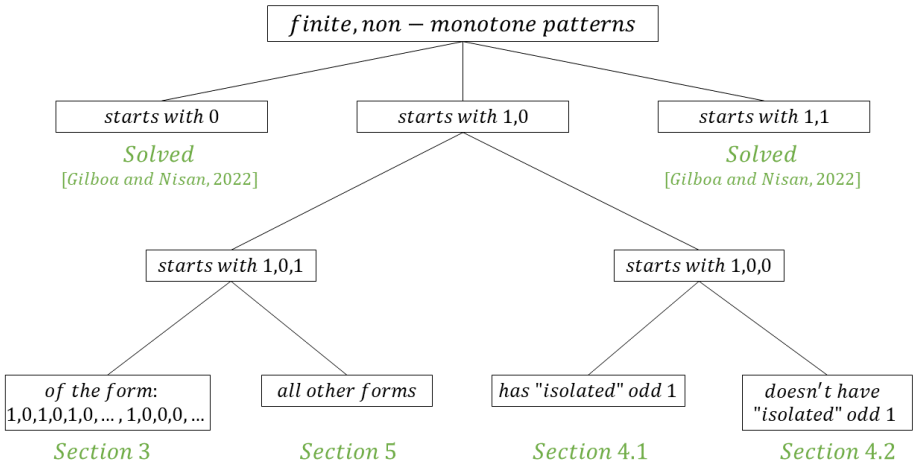


Figure 1 Outline of this paper.

the good, we can simply consider the best response of an agent for any number of productive agents in its neighborhood. Essentially, this can be modeled as a function $T : \mathbb{N} \to \{0, 1\}$, which, as in [3], we represent in the form of a *Best Response Pattern*:

▶ **Definition 2.1.** *A Best-Response Pattern (BRP) of a PGG, denoted by T, is an infinite boolean vector in which the k^{th} entry indicates the best response for each agent v_i given that exactly k neighbors of v_i (excluding v_i) produce the good:*

$$\forall k \geq 0 \;\; T[k] = \textit{best response to } k \textit{ productive neighbors.}$$

▶ **Definition 2.2.** *Given a Public Goods Game defined on a graph $G = (V, E)$ with respect to a BRP T, a strategy profile $s = (s_1, ..., s_n) \in S^n$ (where $s_i \in \{0, 1\}$ represents the strategy of node $v_i \in V$) is a pure Nash equilibrium (PNE) if all agents play the best response to the strategies of their neighbors:*

$$\forall 1 \leq i \leq n \;\; s_i = T[\sum_{j \in N(i)} s_j],$$

where $N(i) = \{j : \{v_i, v_j\} \in E\}$. In addition, if there exists $1 \leq i \leq n$ s.t. $s_i = 1$, then s is called a non-trivial pure Nash equilibrium (NTPNE).

We note that throughout the paper we also use the notation $v_i = 0$ and $v_i = 1$ to indicate the strategy of some node v_i, rather than use $s_i = 0$ and $s_i = 1$, respectively.

▶ **Definition 2.3.** *For a fixed BRP T, the non-trivial[4] pure Nash equilibrium decision problem corresponding to T, denoted by NTPNE(T), is defined as follows: The input is an undirected graph G. The output is 'True' if there exists an NTPNE in the PGG defined on G with respect to T, and 'False' otherwise.*

[4] In this paper, we only study BRPs where the best response for zero productive neighbors is 1, for which there never exists a trivial all-zero PNE (as these are the only BRPs left to solve). However, we sometimes reduce from patterns where this is not the case, and therefore include the non-triviality restriction in our problem definition, in order to correspond with the literature.

▶ **Definition 2.4.** *A BRP T is called monotonically increasing (resp. decreasing) if* $\forall k \in \mathbb{N}$, $T[k] \leq T[k+1]$ *(resp.* $T[k] \geq T[k+1]$*).*

▶ **Definition 2.5.** *A BRP T is called finite if it has a finite number of entries with value 1:*

$$\exists N \in \mathbb{N} \;\; s.t. \;\; \forall n > N \;\; T[n] = 0$$

As seen in Figure 1, the only patterns for which the equilibrium decision problem remains open are patterns that begin with 1,0. We divide those into the two following classes of patterns.

▶ **Definition 2.6.** *A BRP T is called semi-sharp if:*
1. $T[0] = 1$
2. $T[1] = T[2] = 0$

i.e. T begins with $1, 0, 0$.

▶ **Definition 2.7.** *A BRP T is called spiked if:*
1. $T[0] = T[2] = 1$
2. $T[1] = 0$

i.e. T begins with $1, 0, 1$.

We note that given any finite BRP T, the NTPNE(T) problem is in NP, since an assignment to the nodes can be easily verified as an NTPNE by iterating over the nodes and checking whether they all play their best response. Therefore, we only prove NP-hardness of the problems throughout the paper.

3 Hardness of the 0-Or-2-Neighbors Pattern

In this section we show that the equilibrium problem is NP-complete for the 0-Or-2-Neighbors pattern, and provide some intuition about the problem. This result answers an open question from [3]. We then expand this to show hardness of a slightly more general class of patterns. In the 0-Or-2-Neighbors BRP the best response is 1 only to zero or two productive neighbors, as we now define.

▶ **Definition 3.1.** *The 0-Or-2-Neighbors Best Response Pattern is defined as follows:*

$$\forall k \in \mathbb{N} \;\; T[k] = \begin{cases} 1 & \text{if } k = 0 \;\; or \;\; k = 2 \\ 0 & otherwise \end{cases}$$

i.e.

$$T = [1, 0, 1, 0, 0, 0, ...].$$

▶ **Theorem 3.2.** *Let T be the 0-Or-2 Neighbors BRP. Then NTPNE(T) is NP-complete.*

Before proving the theorem, we wish to provide basic intuition about the 0-Or-2-Neighbors BRP, by examining several simple graphs. Take for example a simple cycle. Since $T[2] = 1$ (i.e. best response for two productive neighbors is 1), we have that any simple cycle admits a pure Nash equilibrium[5], assigning 1 to all nodes (see Figure 2). However, looking at a simple path with n nodes, we see that the all-ones assignment is never a pure Nash equilibrium.

[5] In this pattern, any pure Nash equilibrium must also be non-trivial, since $T[0] = 1$.

The reason for this is that $T[1] = 0$ (i.e. best response for one productive neighbors is 0), and so the two nodes at both ends of the path, having only one productive neighbor, do not play best response. Nevertheless, any simple path does admit a pure Nash equilibrium. To see why, let us observe the three smallest paths, of length 2, 3 and 4. Notice that in a path of length two a PNE is given by the assignment 0,1; in a path of length three a PNE is given by the assignment 0,1,0; and in a path of length four a PNE is given by the assignment 1,0,0,1. We can use these assignment to achieve a PNE in any simple path: given a simple path of length n, if $n \equiv 0 \pmod 3$ we use the path of length three as our basis, adding 0,1,0 to it as many times as needed; if $n \equiv 1 \pmod 3$ we use the path of length four as our basis, adding 0,0,1 to it as many times as needed; and if $n \equiv 2 \pmod 3$ we use the path of length two as our basis, adding 0,0,1 to it as many times as needed (see example in Figure 3).

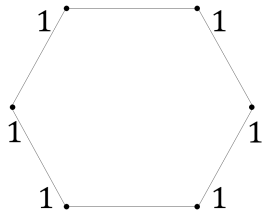


Figure 2 PNE in cycles.

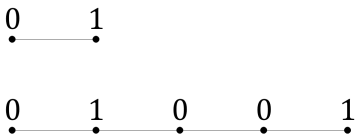


Figure 3 PNE in paths of lengths 2 and 5.

In contrast to the graphs discussed so far, there are graphs in which a pure Nash equilibrium doesn't exist for the 0-Or-2-Neighbors pattern. An example of this can be seen in a graph composed of four triangles, connected as a chain where each two neighboring triangles have a single overlapping vertex, as demonstrated in Figure 4. One may verify that no PNE exists in this graph. This specific structure will also be of use to us during our proof[6].

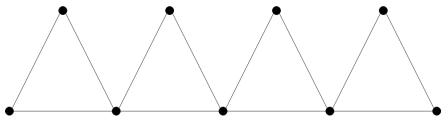


Figure 4 No PNE exists in this graph.

Having provided some intuition regarding the problem, we move on to prove Theorem 3.2. The reduction is from ONE-IN-THREE 3SAT, which is a well known NP-complete problem (see [8]). In ONE-IN-THREE 3SAT, the input is a CNF formula with 3 literals in each clause, and the goal is to determine whether there exists a boolean assignment to the variables such that in each clause exactly one of the literals is assigned True. We begin by introducing our Clause Gadget, which is a main component of the proof. Given a CNF formula, for each of its clauses we construct a 21-nodes Clause Gadget, in which three of the nodes, denoted l_1, l_2, l_3 (also referred to as the *literal nodes*) represent the three literals of the matching clause. The purpose of this gadget is to enforce the property that in any NTPNE, exactly one literal node in the gadget will be assigned 1, which easily translates to the key property of a satisfying assignment in the ONE-IN-THREE 3SAT problem. The three literal

[6] The Negation Gadget defined throughout the proof of Theorem 3.2 is constructed similarly to the graph described here.

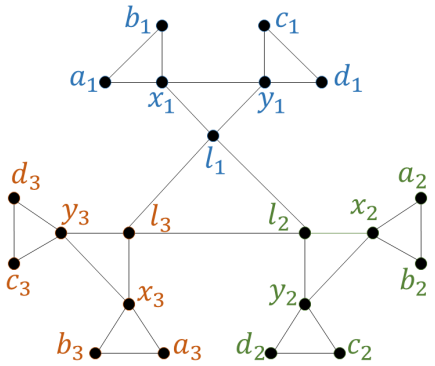


Figure 5 Clause Gadget.

nodes are connected to one another, forming a triangle. Additionally, for each $i \in \{1, 2, 3\}$, l_i is connected to two other nodes x_i, y_i, which are also connected to one another, forming another triangle. Lastly, x_i and y_i each form yet another triangle, along with nodes a_i, b_i and c_i, d_i respectively. We refer to $x_i, y_i, a_i, b_i, c_i, d_i$ as the *sub-gadget* of l_i. We note that out of the nodes of the Clause Gadget, only the literal nodes may be connected to other nodes outside of their gadget, a property on which we rely throughout the proof. The structure of the Clause Gadget is demonstrated in Figure 5, where each sub-gadget is colored differently.

The next four lemmas lead us to the conclusion that the gadget indeed has the desired property mentioned above.

▶ **Lemma 3.3.** *In any NTPNE in a graph G which includes a Clause Gadget cg, if a literal node l_i of cg is assigned 1 then so are its two neighbors from its respective sub-gadget, x_i, y_i. Furthermore, there exists an assignment to the sub-gadget of l_i such that all its nodes play best response.*

Proof. Divide into cases.

Case 1: If $x_i = y_i = 0$, then if $a_i \neq b_i$ (meaning only one of them is assigned 1) then x_i would have two productive neighbors and would not be playing its best response. However, if $a_i = b_i$ then a_i and b_i would not be playing their best response, and we reach a contradiction.

Case 2: If $x_i = 1, y_i = 0$ (the case where $x_i = 0, y_i = 1$ is, of course, symmetric) then x_i must have exactly one more productive neighbor (either a_i or b_i) in order to be playing best response. But then that node would not be playing best response, in contradiction.

Case 3: We are left with the option where $x_i = y_i = 1$, where it is easy to verify that all nodes of the sub-gadget of l_i are playing their best response if we set $a_i = b_i = c_i = d_i = 0$. ◀

▶ **Lemma 3.4.** *In any NTPNE in a graph G which includes a Clause Gadget cg, if one of the literal nodes l_i of cg is assigned 1 then the other two literal nodes of cg must be assigned 0.*

Proof. Since $l_i = 1$, from Lemma 3.3 we have that $x_i = y_i = 1$. Therefore, l_i has two productive neighbors and cannot have any more, and so we have that the other two literal nodes must play 0. ◀

▶ **Lemma 3.5.** *In any graph G which includes a Clause Gadget cg, if exactly one of the literal nodes of cg is assigned 1 then there exists an assignment to the other nodes of cg such that they all (excluding the literal nodes) play best response. In addition, in any such assignment, if the literal nodes have no productive neighbors outside cg, then they also play best response.*

Proof. W.l.o.g. assume that $l_1 = 1, l_2 = l_3 = 0$. Let us observe several details that must hold in such an assignment. Focusing first on the sub-gadget of l_1, according to Lemma 3.3 there exists an assignment the nodes of this sub-gadget such that they all play best response. Furthermore, Lemma 3.3 tells us that $x_1 = y_1 = 1$ (and so l_1 has two productive neighbors within cg). We move on to the sub-gadget of l_2. If $x_2 \neq y_2$ then l_2 would have 2 productive neighbors and would not be playing its best response. If $x_2 = y_2 = 1$ then there is no assignment to a_2, b_2 s.t. a_2, b_2, x_2 all play their best response. Therefore $x_2 = y_2 = 0$. We are left only with the option of setting $a_2 \neq b_2$ and $c_2 \neq d_2$ (for instance $a_2 = c_2 = 1, b_2 = d_2 = 0$). The sub-gadget of l_3 is symmetric to that of l_2. One may verify that in this assignment all nodes of cg excluding the literal nodes indeed play best response, and that if the literal nodes have no productive neighbors outside cg then they also play best response. ◀

▶ **Lemma 3.6.** *In any graph G which includes a Clause Gadget cg, if all three of the literal nodes of cg are assigned 0, and the literal nodes do not have any productive neighbors outside of cg, then the assignment is not a PNE.*

Proof. Assume by way of contradiction that there exists a PNE where $l_1 = l_2 = l_3 = 0$, and all three of them have no productive neighbors outside cg. It must be that the other two neighbors of l_1, x_1, y_1, are assigned with different values (otherwise l_1 is not playing its best response). W.l.o.g. assume $x_1 = 1, y_1 = 0$. Now, if the remaining neighbors of y_1 (c_1 and d_1) are both assigned with 0 or both assigned with 1, then they themselves would not be playing their best response. On the other hand, if we assign them with different values then y_1 would not be playing its best response, and so we have reached a contradiction. ◀

So far, we have seen that in any PNE which includes a Clause Gadget, it must be that exactly one of the literal nodes of that gadget is assigned with 1, as long as the literal nodes don't have productive neighbors outside of their Clause Gadget. As we introduce the external nodes that will be connected to the literal nodes, we will show that indeed they all must be assigned 0 in any PNE, and thus a literal node cannot have any productive neighbor outside of its Clause Gadget, which will finalize the property we were looking to achieve with the Clause Gadget.

Our next goal is to make sure the translation between solutions from one domain to the other is always valid. Specifically, we wish to ensure that in any PNE in our constructed graph, if any two literal nodes represent the same variable in the CNF formula then they will be assigned the same value, and if they represent a variable and its negation then they will be assigned opposite values. We begin with the latter, introducing our Negation Gadget. The goal of the Negation Gadget is to force opposite assignments to two chosen nodes, in any Nash equilibrium. The Negation Gadget is composed of 9 nodes: five 'bottom' nodes b_1, b_2, b_3, b_4, b_5, and four 'top' nodes t_1, t_2, t_3, t_4, and for each $i \leq 4$ we create the edges $\{b_i, b_{i+1}\}$, $\{t_i, b_i\}$ and $\{t_i, b_{i+1}\}$. It can intuitively be described as four triangles that are connected as a chain. Say we have two nodes u, v which we want to force to have opposite assignments, we simply connect them both to node t_2 of a Negation Gadget, as demonstrated in Figure 6.

▶ **Lemma 3.7.** *In any Nash equilibrium in a graph G which includes two nodes u, v connected through a Negation Gadget ng, u and v must have different assignments. Moreover, the node t_2 of ng, to which u and v are connected, must be assigned 0. In addition, if indeed $u \neq v$ and $t_2 = 0$, there exists an assignment to the the nodes of ng such that they all play best response.*

Proof. We first show that u and v must have different assignments, dividing into two cases.

Case 1: Assume by way of contradiction that $u = v = 0$. We divide into two sub-cases, where in the first one $t_2 = 0$; in this case, exactly one of the two remaining neighbors of t_2 must be assigned 1 in order for t_2 itself to be playing best response. If $b_2 = 0$ then $b_3 = 1$ and so, looking at t_1, b_1 (the remaining neighbors of b_2), we see that any assignment to them results either in b_2 not playing best response, or in t_1 not playing best response, in contradiction. If, however, $b_3 = 0$, then $b_2 = 1$, and so, symmetrically, looking at t_3, b_4 (the remaining neighbors of b_3) we see that any assignment to them results either in b_3 not playing best response, or in t_3 not playing best response, in contradiction. In the second sub-case, where $t_2 = 1$, we have that its two remaining neighbors must be assigned the same value in order for t_2 itself to be playing best response. If $b_2 = b_3 = 0$ then again there is no assignment to b_1, t_1 s.t. all of b_1, t_1, b_2 play best response, and if $b_2 = b_3 = 1$ then one may verify that there is no assignment to t_3, t_4, b_4, b_5 s.t. all of t_3, t_4, b_3, b_4, b_5 play best response, and so we reach a contradiction.

Case 2: Assume $u = v = 1$. Then we again divide into sub-cases according to t_2's assignment. If $t_2 = 0$, it must have at least one more productive neighbor in order to play best response. The assignments where $b_2 = b_3 = 1$ or $b_2 = 0, b_3 = 1$ are easily disqualified, seeing as there is no assignment to t_1, b_1 s.t. t_1, b_1, b_2 all play best response. If $b_2 = 1, b_3 = 0$ then it must hold that $t_3 = b_4$ in order for b_3 to play best response, but this would mean that t_3 is not playing best response, in contradiction. If $t_2 = 1$, then its two remaining neighbors b_2, b_3 must be set to 0 in order for it to play best response, and then there is no assignment to b_1, t_1 s.t. all of t_1, b_1, b_2 play best response, in contradiction.

And so it cannot be that $u = v$. We move on to show that t_2 must play 0. Assume by way of contradiction that $t_2 = 1$. Then, seeing as exactly one of u, v is productive, t_2 must have exactly one more productive neighbor in order to play best response. If $b_2 = 1, b_3 = 0$ we reach a contradiction as there is no assignment to t_1, b_1 s.t. t_1, b_1, b_2 all play best response. If $b_2 = 0, b_3 = 1$ we reach a contradiction as there is no assignment to t_3, t_4, b_4, b_5 s.t. all of t_3, t_4, b_3, b_4, b_5 play best response. Lastly, one may verify that in the assignment where $t_1 = b_4 = 1, b_1 = b_2 = b_3 = b_5 = t_2 = t_3 = t_4 = 0$ all nodes of the gadget play best response. ◀

Now, for each variable that appears in the CNF formula, we choose one instance of it and one instance of its negation[7] and connect the literal nodes representing these instances via a Negation Gadget, thus ensuring they are assigned opposite values in any PNE, according to Lemma 3.7. We note that this is not the only place where we use this gadget, as we will see shortly.

We move on to introduce our Copy Gadget, which we will use to force literal nodes which represent the same variable to have the same assignment in any PNE. The Copy Gadget is composed of two negation gadgets ng_1, ng_2, and two additional nodes x, y which have an edge between them. Say we have two nodes u, v which we want to force to have the same assignment in any PNE, then we simply connect u and x to ng_1, and we connect v and x to ng_2. The gadget is demonstrated in Figure 7.

▶ **Lemma 3.8.** *In any Nash equilibrium in a graph G which includes two nodes u, v connected through a Copy Gadget cpg, u, v must have the same assignment, and must have no productive neighbors from cpg. In addition, if $u = v$ then there exists an assignment to the nodes of cpg s.t. all of them play best response.*

[7] We will soon ensure that instances of the same variable would get the same assignment in any PNE, and thus it is sufficient to negate the assignments of only one instance of a variable and its negation.

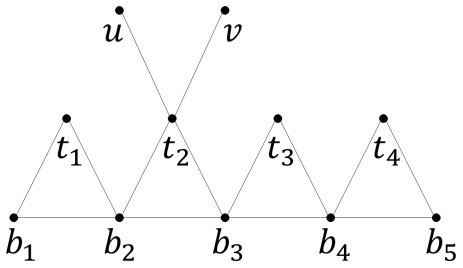


Figure 6 *Negation Gadget connecting u and v.*

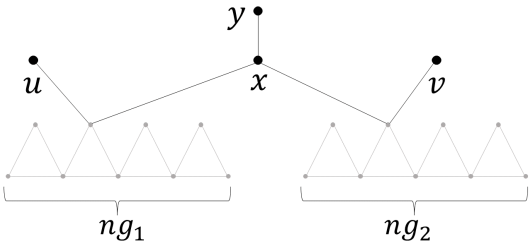


Figure 7 *Copy Gadget connecting u and v.*

Proof. We first show that u and v must have the same assignment. This follows directly from the fact that x is connected to both u and v via a Negation Gadget. Therefore, from Lemma 3.7 we have that $x \neq u$ and $x \neq v$, and so $u = v$. Lemma 3.7 also tells us that the Negation Gadget cannot add productive neighbors to the nodes that are connected to it in any PNE, and therefore u and v have no productive neighbors from *cpg*. Lastly, we show that there exists an assignment to the nodes of *cpg* s.t. they all play best response. From Lemma 3.7 x cannot have any productive neighbors from ng_1 or ng_2. Therefore, if $u = v = 0$ then we can assign $x = 1, y = 0$, and if $u = v = 1$ then we can assign $x = 0, y = 1$. In both cases, we assign ng_1 and ng_2 as suggested in the proof of Lemma 3.7. One may verify that in this assignment indeed all nodes of *cpg* play best response. ◀

Now, for each variable in the CNF formula, we connect all the literal nodes representing its different instances via a chain of copy gadgets, thus (transitively) ensuring they are all assigned the same value in any PNE, according to Lemma 3.8.

Given these lemmas and the graph we constructed, we can now prove Theorem 3.2.

Proof. (Theorem 3.2) Given a ONE-IN-THREE 3SAT instance, we construct a graph G as described previously[8]: For each clause we create a Clause Gadget, we connect all literal nodes representing the same variable through a chain of Copy Gadgets, and for each variable we choose one instance of it and one instance of its negation, and connect the literal nodes representing those instances with a Negation Gadget. If there exists a satisfying assignment to the 3SAT problem, we can set all literal nodes according to the assignment of their matching variable, and set all other nodes as described throughout Lemmas 3.5, 3.7 and 3.8, and according to those lemmas, we get a pure Nash equilibrium. On the opposite direction, if there exists a non-trivial pure Nash equilibrium, then by Lemmas 3.4 and 3.6 in each Clause Gadget exactly one literal node is assigned 1, and by Lemmas 3.7 and 3.8 we have that literal nodes have the same assignment if they represent the same variable, and opposite ones if they represent a variable and its negation. Thus we can translate the NTPNE into a satisfying ONE-IN-THREE 3SAT assignment, assigning 'True' to variables whose literal nodes are set to 1, and 'False' otherwise. ◀

[8] Note that we do not need to explicitly represent the conjunction between the clauses: It is given to us implicitly by the fact that each Clause Gadget must independently satisfy the 1-In-3 property in any Nash equilibrium.

We now wish to expand this result to two slightly more general classes of patterns. Firstly, we notice that the graph constructed throughout the proof of Theorem 3.2 is bounded[9] by a maximum degree of 6. Therefore, the proof is indifferent to entries of the pattern from index 7 onward, which means it holds for any pattern that agrees with the first 7 entries of the 0-Or-2-Neighbors pattern.

▶ **Corollary 3.9.** *Let T be a BRP such that:*

- $T[0] = T[2] = 1$
- $\forall k \in \{1, 3, 4, 5, 6\}\ \ T[k] = 0$

Then NTPNE(T) is NP-complete.

Secondly, according to Theorem 7 in [3], adding 1,0 at the beginning of a hard pattern that begins with 1 yields yet another hard pattern. Using this theorem recursively on the patterns of Corollary 3.9, we have that the equilibrium decision problem is hard for any pattern of the form:

$$T = [\underbrace{1, 0, 1, 0, 1, 0, ..., 1, 0}_{finite\ number\ of\ '1,0'}, 0, 0, 0, ?, ?, ...].$$

▶ **Corollary 3.10.** *Fix $m \geq 1$, and let T be a BRP such that:*

- $\forall 0 \leq k \leq m$
  1. $T[2k] = 1$
  2. $T[2k + 1] = 0$
- $T[2m + 2] = T[2m + 3] = T[2m + 4] = 0$

Then NTPNE(T) is NP-complete.

We will see later on that this result will also be of use during the proof of Theorem 5.1.

There is one very similar class of patterns on which the proofs throughout the paper rely. This is the class of all finite patterns that start with a finite number of 1,0, followed by 1,1, i.e. all patterns of the form:

$$T = [\underbrace{1, 0, 1, 0, 1, 0, ..., 1, 0}_{finite\ number\ of\ '1,0'}, 1, 1, ?, ?, ..., 0, 0, ...].$$

The complexity of those patterns was already discussed and solved in Section 5.4 of [3], but was not formalized and so we state it here in the following lemma.

▶ **Lemma 3.11.** *Fix $m \geq 2$, and let T be a BRP s.t.:*

- *T is finite*
- *$\forall k \in \mathbb{N}$ s.t. $2k \leq m$ $T[2k] = 1$*
- *$T[1] = 0$*
- *$\exists 1 \leq n$ where $2n + 1 \leq m + 1$, s.t. $T[2n + 1] = 1$*

Then NTPNE(T) is NP-complete under Turing reduction.

Proof. The proof follows directly from Theorems 6 and 7 from [3]. ◀

[9] A literal node is connected to 4 nodes within its clause gadget, and possibly 2 nodes from copy gadgets or 1 node from a negation gadget and 1 node from a copy gadget (assuming we connect the negation gadgets at the end of their respective Copy-Gadget-chains).

4 Hardness of Semi-Sharp Patterns

In this section we show hardness of semi-sharp Best-Response Patterns, beginning with a specific sub-class of those patterns in Section 4.1, and expanding to all other semi-sharp patterns in Section 4.2. We remind the reader that semi-sharp patterns are patterns that begin with 1,0,0.

4.1 Semi-Sharp Patterns with Isolated Odd 1

In this section we show that any finite, semi-sharp pattern such that there exists some "isolated" 1 (namely it has a zero right before and after it) at an odd index, presents a hard equilibrium decision problem. Those patterns can be summarized by the following form:

$$T = [1, 0, 0, ?, ?, ..., 0, \underbrace{1}_{odd\ index}, 0, ?, ?, ..., 0, 0, 0, ...]$$

▶ **Theorem 4.1.** *Let T be a BRP which satisfies the following conditions:*

- *T is finite*
- *T is semi-sharp*
- *$\exists m \geq 1$ s.t.:*
  1. *$T[2m] = T[2m+2] = 0$*
  2. *$T[2m+1] = 1$*

Then NTPNE(T) is NP-complete under Turing reduction.

Before we proceed to the proof, we introduce two gadgets and prove two lemmas regarding their functionality.

Force-1-Gadget. The first gadget is denoted the Force-1-Gadget, and it will appear in several parts of the graph we construct for the reduction. The goal of this gadget is to enable us to force any node to be assigned 1 in any Nash equilibrium in a PGG defined by T. This gadget is composed primarily of a triangle x, y, z, where the triangle's nodes have also several 'Antenna' nodes, which are connected only to their respective node from the triangle. Specifically, x will have $2m+1$ Antenna nodes, and y and z will each have $2m$ Antenna nodes. Say we have some node u, whose assignment we wish to force to be 1, then we simply connect u to one of the Antenna nodes of x, denoted a. The gadget is demonstrated in Figure 8.

Add-1-Gadget. our second gadget of this proof is denoted the Add-1-Gadget, and its goal is to enable us to assure the existence of (at least) a single productive neighbor to any node in a Nash equilibrium of a PGG defined by T. Say we have a node v, to which we wish to add a single productive neighbor, in any equilibrium. We construct the Add-1-Gadget as follows. We create $m+1$ nodes denoted $x_1, ..., x_{m+1}$, $m+1$ nodes denoted $y_1, ...y_{m+1}$, and an additional 'bridge' node, denoted b. We connect x_1 and y_1 to all of the other x_i and y_i nodes. For all $i, j \geq 2$ s.t. $i \neq j$, we create the edges $\{x_i, x_j\}, \{y_i, y_j\}, \{x_i, y_j\}$ (the x_i, y_i nodes almost form a clique, except that for each $i \geq 2$ we omit the edge $\{x_i, y_i\}$). Additionally, for all $i \geq 2$ the bridge node b is connected to x_i and to y_i. To b we attach a Force-1-Gadget, and we also connect b to v. The gadget is demonstrated in Figure 9.

The following lemmas formalize the functionality of the two gadgets, beginning with the Force-1-Gadget in Lemma 4.2.

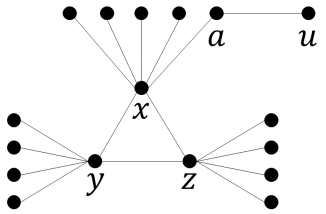


Figure 8 *Force-1-Gadget with $m = 2$, attached to u.*

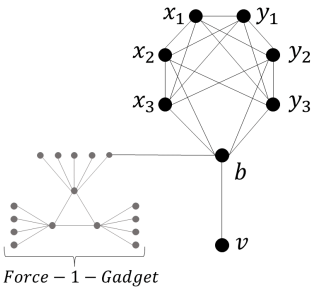


Figure 9 *Add-1-Gadget with $m = 2$, attached to v.*

▶ **Lemma 4.2.** *In any PNE in a graph G corresponding to the BRP T (from Theorem 4.1), where G has a node u that is connected to a Force-1-Gadget fg as described, u must be assigned 1, and its neighbor from fg, a, must be assigned 0.*[10] *Furthermore, if $u = 1$ there exists an assignment to the nodes of fg such that they each play their best response.*

Proof. First we show that u must be assigned 1. Assume by way of contradiction that $u = 0$. Divide into the following two cases. If $x = 1$, then all of its Antenna nodes must be assigned 0 (according to T). Additionally, y and z must also be assigned 0, as otherwise x wouldn't be playing best response, since T is semi-sharp. Therefore, the best response of all of the Antenna nodes of y and z is to play 1, which leaves y and z with $2m+1$ productive neighbors each, and so they are not playing best response, in contradiction. If $x = 0$, then all of its Antenna nodes must play 1. Therefore, x must have at least one other productive neighbor, as otherwise it would have $2m + 1$ productive neighbors and wouldn't be playing best response; w.l.o.g. assume $y = 1$. Then all of y's Antenna nodes must play 0. Therefore, z must play 0, as otherwise y wouldn't be playing best response. This means the best response for z's Antenna nodes is to play 1, which leaves z with $2m + 1$ productive neighbors, and so it isn't playing best response, in contradiction. We move on to showing that a must play 0. This follows directly from the fact that $u = 1$. Since a only has one other neighbor (x), regardless of its strategy the best response for a, according to T, would be playing 0. It is left to show that when $u = 1$ and $a = 0$, there exists an assignment to the nodes of fg s.t. they all play best response. One may verify that when we set $x = y = z = 0$ and set all the Antenna nodes in fg (except for a) to 1, then all nodes of fg play best response (specifically, x, y, z would each have exactly $2m$ productive neighbors, which, by definition of T, means they are playing best response). ◀

We move on to proving the following Lemma, which formalizes the functionality of the Add-1-Gadget.

▶ **Lemma 4.3.** *In any graph G corresponding to the BRP T (from Theorem 4.1), where G has a node v that is connected to an Add-1-Gadget ag as described, there always exists an assignment to the nodes of ag such that they all play best response, regardless of v's strategy. In addition, the bridge node b of ag must be assigned 1 in such an assignment.*

[10] The property that $a = 0$ allows us to use the Force-1-Gadget without risking potentially adding productive neighbors to the respective node.

Proof. The claim that b must play 1 follows directly from the fact that it has a Force-1-Gadget attached to it, i.e. from Lemma 4.2. Additionally, all the nodes of the Force-1-Gadget attached to b can be assigned as suggested in Lemma 4.2. It is left to show a possible assignment to the rest of the nodes of ag. We divide into cases. If $v = 0$, then we set $x_1 = 1$ and all other x_i, y_i nodes we set to 0. If $v = 1$, then we set $x_i = y_i = 1$ for all $1 \leq i \leq m + 1$. One may verify that given these assignments all nodes of ag play their best response. ◀

Given these two gadgets, we are almost ready to prove Theorem 4.1. We now introduce a useful definition, and then proceed to the proof of the theorem.

▶ **Definition 4.4.** *Let T and T' be two BRPs. We say that T' is shifted left by t from T if*

$$\forall k \geq 0 \;\; T'[k] = T[k + t].$$

Proof. (Theorem 4.1) Denote by T' the pattern which is shifted left by 1 from T, i.e.:

$$\forall k \geq 0 \;\; T'[k] = T[k + 1].$$

Notice that T' is non-monotonic, finite, and begins with 0, and therefore NTPNE(T') is NP-complete according to Theorem 4 in [3], which allows us to construct a Turing reduction from it. The technique of the reduction is very similar to those of the proofs of Theorems 5 and 6 in [3]. Given any graph $G = (V, E)$, where $V = v_1, ..., v_n$, we construct n graphs $G_1, ..., G_n$, where for each $1 \leq i \leq n$ the graph G_i is defined as follows. The graph G_i contains the original input graph G, and in addition, we connect a unique Add-1-Gadget to each of the original nodes, and a Force-1-Gadget only to node v_i. If there exists some non-trivial PNE in the PGG defined on G by T', let v_j be some node who plays 1. Then the same NTPNE is also an NTPNE in the PGG defined by T on G_j, when we assign the nodes of the additional gadget as suggested in Lemmas 4.2 and 4.3. To see why, notice that T' is shifted left by 1 from T, and the Add-1-Gadgets ensure that all nodes have exactly one additional productive neighbor than they had in G.

In the other direction, if there exists an NTPNE in a PGG defined by T on one of the graphs G_j, then by the same logic this is also a PNE in the game defined by T' on G (ignoring the assignments of the added nodes). Moreover, the Force-1-Gadget ensures this assignment is non-trivial even after removing the added nodes, since v_j must play 1 in this assignment. ◀

4.2 All Semi-Sharp Patterns

In this section we show that any finite, non-monotone, semi-sharp pattern presents a hard equilibrium problem.

▶ **Theorem 4.5.** *Let T_1 be a finite, non-monotone, semi-sharp BRP. Then NTPNE(T_1) is NP-complete under Turing reduction.*

Before proceeding to the proof, we wish to introduce the following definition and prove two lemmas related to it.

▶ **Definition 4.6.** *Let T and T' be two BRPs such that $\forall k \in \mathbb{N}$ it holds that $T[k] = T'[2k]$. Then we say that T' is a double-pattern of T, and T is the half-pattern of T'. Notice that a pattern has a unique half-pattern, whereas, since the definition does not restrict T' in the odd indices, any pattern has infinite double-patterns.*

The first lemma is very simple and intuitive, stating that the largest index with value 1 in a half pattern is strictly smaller than the largest index with value 1 in its original pattern. This is true since for any index i s.t. the value of the half pattern is 1 in that index, the original pattern has a value of 1 in index $2i$.

▶ **Lemma 4.7.** *Let T and T' be two finite BRPs such that T is the half-pattern of T'. Denote by i the largest index s.t. $T[i] = 1$ and denote by j the largest index s.t. $T'[j] = 1$. Then if $j > 0$ we have that $i < j$.*

Proof. The proof is trivially given by the definition of a half pattern, since $T'[2i] = T[i]$. ◀

The next lemma is less trivial, stating the relation between hardness of a pattern and its double-pattern.

▶ **Lemma 4.8.** *Let T be a BRP such that NTPNE(T) is NP-complete, and let T' be a double-pattern of T. Then NTPNE(T') is NP-complete.*

Proof. We use a specific case of the same reduction that was used to prove Theorem 4 in [3]. Given a graph $G_1 = (V_1, E_1)$ as input, where $V_1 = v_1^1, ..., v_n^1$, we create another replica of it $G_2 = (V_2, E_2)$, where $V_2 = v_1^2, ..., v_n^2$. For each node (from both graphs), we add edges connecting it to all replicas of its neighbors from the opposite graph. That is, the following group of edges is added to the graph:

$$E = \{\{v_i^1, v_j^2\} | \{v_i^1, v_j^1\} \in E_1\}.$$

A demonstration of the reduction can be seen in Figure 10.

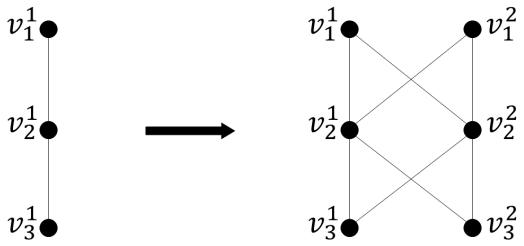


Figure 10 Example of the reduction of Lemma 4.8's proof.

Denote by P the PGG defined on G_1 by T, and by P' the PGG defined by T' on $G' = (V', E')$ where $V' = V_1 \cup V_2$, $E' = E \cup E_1 \cup E_2$. We show that there exists an NTPNE in P iff there exists one in P'. If there exists an NTPNE in P, we simply give the nodes of G_2 the same assignment as those of G_1. Since T' is a double pattern of T, any node $v' \in V'$ must play best response, having exactly twice as many supporting neighbors than it had (or its replica had) in P. In addition, this assignment is clearly non-trivial as the nodes of G_1 have the same (non-trivial) assignment they had in P.

In the opposite direction, if there exists an NTPNE in P', notice that for all $1 \leq i \leq n$ it must be that v_i^1 and v_i^2 have identical assignments, since they both share exactly the same neighbors, and thus have identical best responses. Therefore, any node $v' \in V$ must have an even number of productive neighbors, half of which are in V_1 and the other half in V_2 (as for each productive neighbor from V_1 there is a respective productive neighbor from V_2). We then simply ignore G_2, and leave the assignment of G_1 as it is, and each node shall now have exactly half as many productive neighbors as it had in the original assignment. Since T is a half pattern of T', we get a PNE in P. Furthermore, the symmetry between matching nodes from G_1 and G_2 ensures that at least one node from G_1 was originally assigned 1, and so the constructed PNE in P' is also non-trivial. ◀

Given Lemmas 4.7 and 4.8, we are now able to prove Theorem 4.5. The intuitive idea of the proof is that we halve the pattern T_1 (i.e. find its half-pattern) repeatedly, until eventually we reach some pattern for which we already know the equilibrium problem is hard, which, as we will see, must happen at some point. Then, by applying Lemma 4.8 recursively, we have that T_1 is hard.

Proof of Theorem 4.5. From Lemma 4.7 we have that if we halve a pattern beginning with 1 enough times, we will eventually reach the Best-Shot pattern: $T_{Best\text{-}Shot}[0] = 1$ and $\forall k \geq 1 \;\; T_{Best\text{-}Shot}[k] = 0$. Divide into two cases.

In the first case assume that $\forall k \in \mathbb{N}$ it holds that $T_1[2^k] = 0$. In this case, we know that no matter how many times we halve T_1 into patterns $T_2, T_3, ...$, the value in index 1 of all these half-patterns will always be 0, i.e. $T_i[1] = 0$ for all i. Assume that we halve T_1 repeatedly into patterns $T_2, T_3, ..., T_m$ (where T_i is the half pattern of T_{i-1}) such that T_m is the first time that we reach the Best-Shot pattern. Observe T_{m-1}. For any *even* index $k \neq 0$ it must hold that $T_{m-1}[k] = 0$, otherwise T_m would not be the Best-Shot pattern. Additionally, there must exist at least one *odd* index j s.t. $T_{m-1}[j] = 1$, since T_m is the *first* time we reach the Best-Shot pattern. For these two reasons, we have that T_{m-1} satisfies the conditions of Theorem 4.1 and therefore NTPNE(T_{m-1}) is NP-complete under Turing reduction. From Lemma 4.8 (used inductively), we have that $\forall 1 \leq i \leq m-2$ NTPNE(T_i) is also NP-complete under Turing reduction, and specifically NTPNE(T_1).

In the second case, assume that there exists some $k \in \mathbb{N}$ s.t. $T_1[2^k] = 1$. In that case, after at most k halvings, we reach some pattern for which the value of index 1 is 1. Assume that we halve T_1 repeatedly into patterns $T_2, T_3, ..., T_n$ (where T_i is the half pattern of T_{i-1}) such that T_n is the first time that we reach a pattern for which index 1 is 1, i.e. $\forall 1 \leq i \leq n-1 \;\; T_i[1] = 0$ and $T_n[1] = 1$. Notice that, additionally, by definition of a half-pattern for each i it holds that $T_i[0] = 1$ (since $T_1[0] = 1$). If T_n is non-monotone, then by Theorem 5 in [3] we have that NTPNE(T_n) is NP-complete under Turing reduction, and from Lemma 4.8 (used inductively), we have that $\forall 1 \leq i \leq n-1$ NTPNE(T_i) is also NP-complete under Turing reduction, and specifically NTPNE(T_1). Otherwise (i.e. T_n is monotone), denote by l the largest index s.t. $T_n[l] = 1$, and observe T_{n-1}. By definition of double-patterns, we have that:

$$\forall j \in \mathbb{N} \;\; T_{n-1}[2j] = \begin{cases} 1 & \text{if } j \leq l \\ 0 & \text{otherwise} \end{cases}$$

i.e. the value in the even indices up to $2l$ is 1, and afterwards 0. Since T_n is defined to be the first halving of T_1 s.t. its value in index 1 is 1, we have that $T_{n-1}[1] = 0$. However, since the definition of a double-pattern does not restrict its values in odd indices, there might be odd indices (strictly larger than 1) for which the value of T_{n-1} is 1. Divide into 3 sub-cases:

Sub-case 1: If there exists some $z \leq l$ s.t. $T_{n-1}[2z+1] = 1$, then by Lemma 3.11, we have that NTPNE(T_{n-1}) is NP-complete under Turing reduction.

Sub-case 2: Otherwise, if there exists some $z > l$ s.t. $T_{n-1}[2z+1] = 1$, then observe the pattern T'_{n-1}, which we define as the pattern shifted left by $2l$ from T_{n-1} i.e.:

$$\forall j \in \mathbb{N} \;\; T'_{n-1}[j] = T_{n-1}[j + 2l]$$

Notice that this pattern satisfies the conditions of Theorem 4.1, and therefore NTPNE(T'_{n-1}) is NP-complete under Turing reduction. Then, by applying Theorem 7 from [3] l times, we have that NTPNE(T_{n-1}) is also NP-complete under Turing reduction.

Sub-case 3: Otherwise (i.e. there is no odd index whatsoever in which the value of T_{n-1} is 1), then by Corollary 3.10 we have that NTPNE(T_{n-1}) is NP-complete under Turing reduction.

And so, in either case we have that NTPNE(T_{n-1}) is NP-complete under Turing reduction, and therefore from Lemma 4.8 (used inductively), we have that $\forall 1 \leq i \leq n-1$ NTPNE(T_i) is also NP-complete under Turing reduction, and specifically NTPNE(T_1). ◀

5 Hardness of All Spiked Patterns

There are several finite, spiked patterns that we have not yet proved hardness for, and we now have enough tools to close the remaining gaps. We remind the reader that spiked patterns are patterns that begin with 1,0,1. The following theorem formalizes the result of this section, and completes the characterization of all finite patterns.

▶ **Theorem 5.1.** *Let T be a finite, spiked BRP. Then NTPNE(T) is NP-complete under Turing reduction.*

The intuitive idea of the proof is as follows. If the pattern simply alternates between 1 and 0 a finite amount of times (and at least twice, since the pattern is spiked), followed by infinite 0's, i.e. the pattern is of the form

$$T = [\underbrace{1,0,1,0,1,0,...,1,0}_{finite\ number\ of\ '1{,}0'},0,0,0,...]$$

then the problem[11] is already shown to be hard by Corollary 3.10. Otherwise, we wish to look at the first "disturbance" where this pattern stops alternating from 1 to 0 regularly. Either the first "disturbance" is a 1 at an odd index, i.e. the pattern is of the form

$$T = [\underbrace{1,0,1,0,1,0,...,1,0}_{finite\ number\ of\ '1{,}0'},1,1,?,?,...]$$

or the first "disturbance" is a 0 at an even index, i.e. the pattern is of the form

$$T = [\underbrace{1,0,1,0,1,0,...,1,0}_{finite\ number\ of\ '1{,}0'},0,?,?,...,1,?,?,...]$$

(in the latter option, after the first "disturbance" there must be some other index with value 1, since otherwise the pattern fits the form of Corollary 3.10). The first option was solved in Lemma 3.11, and the second option can be solved using our previous results, as we shall now formalize in the proof.

Proof of Theorem 4.5. If T satisfies the conditions of Corollary 3.10 or Lemma 3.11 then NTPNE(T) is NP-complete under Turing reduction according to them. Otherwise, let k be the smallest integer such that $T[2k] = 0$. Denote by T' the pattern which is shifted left by $2k-2$ from T, i.e.:

$$\forall j \geq 0 \;\; T'[j] = T[j + 2k - 2]$$

[11] In fact, Corollary 3.10 gives a more general result, but we currently only need the special case where the pattern ends with infinite 0's.

Notice that from definition of k (being the first even index such that $T[2k] = 0$) we have that for all $j < k$ it holds that $T[2j] = 1$. Moreover, since T does not satisfy the conditions of Lemma 3.11 it must hold for all $j \leq k$ that $T[2j-1] = 0$, i.e. the value of T in the odd indices until $2k$ is 0 (since otherwise T would start with a finite number of 1,0, followed by two consecutive 1's, and would satisfy the conditions of Lemma 3.11). Thus, we have that

$$\forall j < 2k \;\; T[j] = \begin{cases} 1 \text{ if } j \text{ is even} \\ 0 \text{ if } j \text{ is odd} \end{cases} \tag{1}$$

In particular, we have that $T[2k-2] = 1,\ T[2k-1] = 0$, which implies that $T'[0] = 1,\ T'[1] = 0$; as $T[2k] = 0$ we have that $T'[2] = 0$, and thus we conclude that T' is semi-sharp. In addition, since T does not satisfy the conditions of Corollary 3.10, there must be some other index $x > 2k$ such that $T[x] = 1$, and therefore we have that T' is non-monotone. Therefore, by Theorems 4.1 and 4.5, we have that NTPNE(T') is NP-complete under Turing reduction. We now wish to use this in order to prove that NTPNE(T) is also hard.

From Equation 1, we can apply Theorem 7 of [3] $(k-1)$ times (we remind the reader that this theorem states that prefixing a hard pattern with $1, 0$ maintains its hardness), and we have that NTPNE(T) is NP-complete under Turing reduction. ◀

References

1 Yann Bramoullé and Rachel Kranton. Public goods in networks. *Journal of Economic Theory*, 135(1):478–494, 2007. doi:10.1016/j.jet.2006.06.006.

2 Mustapha Chellali, Odile Favaron, Adriana Hansberg, and Lutz Volkmann. k-domination and k-independence in graphs: A survey. *Graphs and Combinatorics*, 28(1):1–55, 2012. doi:10.1007/s00373-011-1040-3.

3 Matan Gilboa and Noam Nisan. Complexity of public goods games on graphs. In Panagiotis Kanellopoulos, Maria Kyropoulou, and Alexandros A. Voudouris, editors, *Algorithmic Game Theory - 15th International Symposium, SAGT 2022, Colchester, UK, September 12-15, 2022, Proceedings*, volume 13584 of *Lecture Notes in Computer Science*, pages 151–168. Springer, 2022. doi:10.1007/978-3-031-15714-1_9.

4 David Kempe, Sixie Yu, and Yevgeniy Vorobeychik. Inducing equilibria in networked public goods games through network structure modification. In Amal El Fallah Seghrouchni, Gita Sukthankar, Bo An, and Neil Yorke-Smith, editors, *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*, pages 611–619. International Foundation for Autonomous Agents and Multiagent Systems, 2020. doi:10.5555/3398761.3398835.

5 Max Klimm and Maximilian J. Stahlberg. Complexity of equilibria in binary public goods games on undirected graphs. In Kevin Leyton-Brown, Jason D. Hartline, and Larry Samuelson, editors, *Proceedings of the 24th ACM Conference on Economics and Computation, EC 2023, London, United Kingdom, July 9-12, 2023*, pages 938–955. ACM, 2023. doi:10.1145/3580507.3597780.

6 Arnab Maiti and Palash Dey. On parameterized complexity of binary networked public goods game. In Piotr Faliszewski, Viviana Mascardi, Catherine Pelachaud, and Matthew E. Taylor, editors, *proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2022)*, pages 871–879, Auckland New Zealand, 2022. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS). doi:10.5555/3535850.3535948.

7 Christos H. Papadimitriou and Binghui Peng. Public goods games in directed networks. In Péter Biró, Shuchi Chawla, and Federico Echenique, editors, *EC '21: The 22nd ACM Conference on Economics and Computation, Budapest, Hungary, July 18-23, 2021*, pages 745–762, Budapest Hungary, 2021. ACM. doi:10.1145/3465456.3467616.

8 Thomas J. Schaefer. The complexity of satisfiability problems. In Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho, editors, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 216–226, San Diego California USA, 1978. ACM. doi:10.1145/800133.804350.

9 Yongjie Yang and Jianxin Wang. A refined study of the complexity of binary networked public goods games. *CoRR*, abs/2012.02916, 2020. doi:10.48550/arXiv.2012.02916.

10 Sixie Yu, Kai Zhou, P. Jeffrey Brantingham, and Yevgeniy Vorobeychik. Computing equilibria in binary networked public goods games. *CoRR*, abs/1911.05788, 2019. doi:10.48550/arXiv.1911.05788.

11 Sixie Yu, Kai Zhou, P. Jeffrey Brantingham, and Yevgeniy Vorobeychik. Computing equilibria in binary networked public goods games. In *The 34th AAAI Conference on Artificial Intelligence, AAAI 2020, The 32nd Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The 10th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, volume 34(2), pages 2310–2317. AAAI Press, 2020. doi:10.1609/aaai.v34i02.5609.

Linear Relaxed Locally Decodable and Correctable Codes Do Not Need Adaptivity and Two-Sided Error

Guy Goldberg
Weizmann Institute of Science, Rehovot, Israel

Abstract

Relaxed locally decodable codes (RLDCs) are error-correcting codes in which individual bits of the message can be recovered by querying only a few bits from a noisy codeword. For uncorrupted codewords, and for every bit, the decoder must decode the bit correctly with high probability. However, for a noisy codeword, a relaxed local decoder is allowed to output a "rejection" symbol, indicating that the decoding failed.

We study the power of adaptivity and two-sided error for RLDCs. Our main result is that if the underlying code is linear, *adaptivity and two-sided error do not give any power to relaxed local decoding.* We construct a reduction from adaptive, two-sided error relaxed local decoders to non-adaptive, one-sided error ones. That is, the reduction produces a relaxed local decoder that never errs or rejects if its input is a valid codeword and makes queries based on its internal randomness (and the requested index to decode), independently of the input.

The reduction essentially maintains the query complexity, requiring at most one additional query. For any input, the decoder's error probability increases at most two-fold. Furthermore, assuming the underlying code is in systematic form, where the original message is embedded as the first bits of its encoding, the reduction also conserves both the code itself and its rate and distance properties

We base the reduction on our new notion of *additive* promise problems. A promise problem is additive if the sum of any two YES-instances is a YES-instance and the sum of any NO-instance and a YES-instance is a NO-instance. This novel framework captures both linear RLDCs and property testing (of linear properties), despite their significant differences.

We prove that in general, algorithms for *any* additive promise problem do not gain power from adaptivity or two-sided error, and obtain the result for RLDCs as a special case. The result also holds for relaxed locally *correctable* codes (RLCCs), where a *codeword* bit should be recovered.

As an application, we improve the best known lower bound for linear adaptive RLDCs. Specifically, we prove that such codes require block length of $n \geq k^{1+\Omega(1/q^2)}$, where k denotes the message length and q denotes the number of queries.

2012 ACM Subject Classification Theory of computation → Error-correcting codes

Keywords and phrases Locally decodable codes, Relaxed locally correctable codes, Relaxed locally decodable codes

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.74

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://eccc.weizmann.ac.il/report/2023/067/

Funding *Guy Goldberg*: Research supported by the European Research Council (ERC) CoG (grant No. 772839) and the Israel Science Foundation (grant No. 2073/21).

Acknowledgements We would like to thank Irit Dinur for her guidance and encouragement. We would also like to thank Oded Goldreich for insightful discussions, and to Yotam Dikstein for his helpful comments.

1 Introduction

Suppose you receive a binary string (a "message"), and would like to know the value of the message at some index i. How many queries do you need to make? The answer is obviously 1, as you can query index i and get the value. But what if some of the message bits are corrupted, because they were, say, transmitted over a noisy channel? The bit at the needed index i might have been corrupted.

Error-correcting codes might help. Such codes allow encoding the message with extra redundancy as a codeword, and the original message can be recovered even if some bits of the codeword were corrupted. However, one needs to read the entire codeword to recover the original message. As our goal was to read only one bit from the message, this solution seems inefficient.

Locally decodable codes (LDCs), introduced by Katz and Trevisan [17], are aimed at solving this problem. These codes are equipped with a local decoding algorithm ("decoder") that recovers each message bit by querying a few bits from a codeword, instead of reading all of it. Two main measures of efficiency for LDCs are the query complexity of the decoder (which we want to be as small as possible) and the rate of the code (which we want to be high). A similar notion, originated in works on program checking by Blum and Kannan [5] and Lipton [21], is of locally *correctable* codes (LCCs). These are error-correcting codes that admit a local algorithm (now called "corrector") that not only recovers each message bit, but is also required to correct any bit from the *codeword*.

LDCs and LCCs have profoundly impacted theoretical computer science and found numerous applications. Despite the extensive research, current constructions require adding a large amount of redundancy. Motivated by this, Ben-Sasson et. al. [3] introduced Relaxed Locally Decodable Codes (RLDCs). For uncorrupted codewords, and for every bit, a relaxed local decoder still must correctly decode the bit with high probability. However, for a noisy codeword, it is now allowed to output a "rejection" symbol, indicating that the decoding failed. This relaxation allows constructing codes with a dramatically better tradeoff between query complexity and rate. Such codes have found various applications, notably in the construction of proof systems (e.g., [22]) and within the area of property testing (e.g., [11]).

More formally, a relaxed local decoder of radius $\rho > 0$ for a code C with soundness error $\epsilon_{\text{soundness}}$ and completeness error $\epsilon_{\text{completeness}}$ is a procedure that gets oracle access to $w \in \{0,1\}^n$, that is ρ-close[1] to some codeword $c = C(x)$ and an index i, and satisfies the following two requirements:

1. (completeness) If w is a valid codeword (that is, $w = c$) then for every i the relaxed local decoder outputs x_i with probability at least $1 - \epsilon_{\text{completeness}}$.
2. (relaxed local decoding) Otherwise, with probability at least $1 - \epsilon_{\text{soundness}}$, the relaxed local decoder outputs x_i or a "reject" symbol $\perp$, indicating the decoding failed.

We call n the *block length* of the code, and the length of x the *message length*.

Gur, Ramnarayan, and Rothblum [14] considered an analogous relaxation for local *correction*, where the corrector either recovers the desired codeword bit, or rejects in case it detects a corruption. They named such codes Relaxed Locally *Correctable* Codes, or RLCCs (see Definition 7).

In this work, we study the power of adaptivity and two-sided error of linear RLDCs and RLCCs. We say that a local algorithm is *adaptive* if it is allowed to choose its queries according to answers of previous queries, and that it is *non-adaptive* otherwise (i.e, if it

[1] Two strings are ρ-close to each other if the normalized Hamming distance between them is at most ρ.

determines its queries based only on its internal coin tosses). Though adaptive algorithms are syntactically stronger than non-adaptive ones, all known constructions of RLDCs and RLCCs are non-adaptive (e.g. [3, 15, 2, 7, 20, 8]). This raises the question of whether it is possible to cleverly utilize adaptivity in order to make improved constructions.

Adaptivity also plays a central role in the study of lower bounds. A common strategy for establishing these bounds is to first address the easier case of non-adaptive RLDCs. Then, the lower bound for adaptive RLDCs is derived by using a generic reduction from adaptive decoders to non-adaptive ones. Alas, known reductions cause an exponential blow-up in the query complexity (or a similar blow-up in the soundness error), resulting in worse lower bounds for adaptive RLDCs [13, 10].

A main question in the study of probabilistic algorithms concerns the strength of two-sided error algorithms vs. one-sided ones. We say that an algorithm has a one-sided error if it never errs on "YES" instances, and a two-sided error if it is allowed to err on "both sides". A major open problem in computational complexity is whether $\mathcal{BPP} = \mathcal{RP}$, which asks whether, in general, an algorithm allowed two-sided error possesses more computational power than one restricted to have a one-sided error. In the standard definition of RLDCs, the decoder can err with a small probability, even when its input is a valid codeword. That is, it is allowed to have two-sided error. In this work, we ask the equivalent $\mathcal{BPP} = \mathcal{RP}$ question for RLCCs: Are one-sided error relaxed local decoders weaker than two-sided error ones? Can we transform any two-sided error decoder, eliminate its errors on valid codewords, to become a one-sided error decoder?

1.1 Our results

Our main result is that for linear codes, two-sided error and adaptivity do not give any strength to RLDCs and RLCCs.

We show a reduction that starts with a relaxed local decoder (resp., corrector) that might query adaptively and err on valid codewords, and ends with a relaxed local decoder (resp., corrector) for the same code, that is non-adaptive and never errs or rejects on valid codewords. The reduction adds at most one additional query. The new soundness error is the sum of the completeness error and soundness error of the original algorithm. Hence the gap between completeness and soundness stays the same. In particular, the error probability, which is the probability to err on *any* specific input (i.e, $\max(\epsilon_{\text{completeness}}, \epsilon_{\text{soundness}})$) is at most doubled.

▶ **Theorem 1.** *Let C be a* linear *systematic*[2] *code.*

If C has a relaxed local decoder of radius ρ with completeness error $\epsilon_{completeness}$, soundness error $\epsilon_{soundness}$ and query complexity q, then it has a one-sided error, non-adaptive *decoder of radius ρ, with soundness error $\epsilon_{completeness} + \epsilon_{soundness}$ and query complexity $q + 1$.*

▶ **Theorem 2.** *Let C be a* linear *code.*

If C has a relaxed local corrector of radius ρ with completeness error $\epsilon_{completeness}$, soundness error $\epsilon_{soundness}$ and query complexity q, then it has a one-sided error, non-adaptive *corrector of radius ρ, with soundness error $\epsilon_{completeness} + \epsilon_{soundness}$ and query complexity $q + 1$.*

[2] A code is called *systematic* if the entire original message is embedded as the first bits of its encoding.

Improved lower bound

Building upon the work of Gur and Lachish [13], Goldreich [10] established lower bounds on the achievable rate by an RLDC with a constant query complexity q. Specifically, the lower bounds are $n \geq k^{1+\Omega(1/q^2)}$ for the non-adaptive case and $n \geq k^{1+\Omega(1/q^3)}$ for adaptive RLDCs. By utilizing Theorem 1, we can remove the adaptivity restriction from the tighter lower bound (for linear codes), leading to the following result:

▶ **Theorem 3.** *Let C be a linear RLDC with message length k, block length n, constant query complexity q and constant correction radius and error probability.*

Then $n \geq k^{1+\Omega(1/q^2)}$.

This represents an improvement over the current state-of-the-art lower bound by Dall'Agnol, Gur and Lachish [9], which is $n \geq k^{1+\Omega(\frac{1}{q^2 \log^2 q})}$ (although it holds for non-linear codes as well).

Linearity

Our reduction only works for linear codes. Nevertheless, linear codes are an important type of error-correcting codes and have been extensively studied. Virtually all known RLDCs and RLCCs (and their non-relaxed counterparts) constructions are of linear codes.[3]

Non-systematic codes

In Theorem 1, it is assumed that the code is in a systematic form. This is a technical detail, not an inherent limitation. Any RLDC can be transformed to be systematic by adding the message bits to the beginning of its encoding. This transformation results in, at most, a doubling of the block length of the code and a corresponding reduction in the decoding radius by the same factor. Consequently, if the code's rate and decoding radius were initially constant, they remain unchanged after this transformation. In addition, this transformation enables the elimination of the systematic requirement in Theorem 3.

It is worth noting that any linear code can be made systematic through a basis change without altering its block length. While such a transformation does not affect the set of valid codewords, it does alter the encoding function. For RLDCs, this implies that the code's decoder may no longer be valid after the transformation.

Known reductions

We note two well-known immediate reductions from adaptive to non-adaptive local algorithms. These reductions date back to [17], which stated them for non-relaxed LDCs, but they also apply to relaxed ones.

The first reduction is to replace each of the q adaptive queries with multiple non-adaptive ones. We replace the i-th query with 2^{i-1} queries, one for each possible result of the previous queries. This reduction yields an exponential blowup in the query complexity. The second reduction is to "guess" the result of the first $q-1$ queries, and query a set of indices based on that guess. This reduction exponentially decreases the algorithm's soundness.

[3] A remarkable exception is multiplicity codes, which are not linear. Fortunately, their codewords constitute a subgroup (the sum of two codewords is a codeword), so our framework still covers them. See Definition 4.

In light of the above, even a reduction with a polynomial blowup in the query complexity would have been an exciting result.

Our contribution

Ben-Sasson, Harsh and, Raskhodnikov [4] showed that for testing *linear* properties, adaptivity and two-sided error do not help. Our work is based on [4], and extends their reductions to the setting of RLDCs and RLCCs.

One technical difficulty in extending their work is that decoders and correctors get an index as an input, in addition to the noisy codeword. This difficulty turns out to be minor. The major difficulty is regarding the actual task at hand. First, testers work under the promise that the input either has the property, or is *far* from any element having it. In contrast, decoders and correctors work under the promise that the input is *close* to the code (or in it). Furthermore, the "output type" is different between those algorithms. On the one hand, a tester has a binary verdict - it accepts or rejects the input. On the other hand, the output of decoders and correctors is not binary - it is a symbol of the message, or a rejection symbol.

To overcome those differences, we introduce the abstract notion of *additive promise problem* (Definition 4). We show that testing linear properties and relaxed decoding/correction of linear codes satisfy this notion.

We then present a reduction that applies to *any* promise problem satisfying this notion. The reduction is a generalization of the one in [4]. The generalization is done by decoupling the logic of the reduction from the testing context. We prove the main results by combining the interpretation of relaxed decoding/correction as promise problems with the generic reduction. Arguably, the generalized reduction also provides a cleaner and more straightforward presentation of [4].

We consider the main contribution of this work to be conceptual. By formalizing the notion of additive promise problem, we demonstrate what is the minimal set of requirements necessary for the techniques of [4] to be applicable. We believe the new notion is natural and intuitive, and hope it will find more applications in the future.

1.2 Motivation

The primary motivation for this work is to enhance our understanding of RLDCs and RLCCs. Additionally, we outline specific motivations as follows.

Applications for lower bounds

Our results show that, for linear codes, any lower bound for non-adaptive, one-sided error RLDCs (resp., RLCCs) can be translated to a lower bound for adaptive, two-sided error RLDCs (resp., RLCCs). Applying this outcome to the work of [10] yields Theorem 3, which is the current best lower bound for linear, adaptive, two-sided error RLDCs. However, this bound is not known to be tight; specifically, the best RLCC construction with constant query complexity q, of Asadi and Shinkar [2], achieves block length $n = k^{1+O(1/q)}$. As our reduction is applicable to any RLCC, it holds potential for accommodating future improvements in lower bounds

In addition, in the case of linear codes, any lower bound applicable to RLDCs also serves as a lower bound for RLCCs, but the reverse is not true. This arises from the fact that every linear code can be represented systematically, with the initial bits of the codeword corresponding to the original message. Hence, for such codes, the corrector can be used as a

decoder, so RLCCs are stronger objects than RLDCs. Our reduction operates independently on each type, implying its potential application for improved lower bounds on RLCCs, which may surpass those for RLDCs.

We note that the work of [9] extends the result of [13] to adaptive, two-sided error RLDCs, for all codes, including non-linear ones. However, this extension does not rely on a generic reduction and is notably highly involved. In contrast, our reduction is arguably much simpler, offering an alternative, simpler proof for linear codes.

Constructions

Virtually all known constructions of RLDCs and RLCCs are non-adaptive, linear, and have one-sided error. Our results show that this should not be a surprise. It is impossible to use adaptivity or two-sided error to improve constructions of linear RLDCs / RLCCs.

Definitions

In some works (e.g, [6, 15, 2, 7]), the definition of RLDC requires it to have one-sided error (i.e., it is not an additional property that a decoder might have). Other works (e.g., [3, 9]) use the same definition we gave above. Our result settles this nuance in the definitions - both are equivalent (for linear codes).

1.3 Technical overview

We next give a high-level sketch of the reduction.

Promise problems

We prove Theorem 1 and Theorem 2 by proving a general result on a family of promise problems. First, a promise problem is a pair of disjoint sets, Y (the YES-instances) and N (the NO-instances). A randomized algorithm for a promise problem gets as input $x \in Y \cup N$ (we sometimes call $Y \cup N$ "the promise") and outputs YES or NO. If $x \in Y$ then the algorithm must output YES with high probability. Similarly, if $x \in N$ it must output NO with high probability.

The main new idea we introduce in this work is of *additive* promise problems.

▶ **Definition 4.** *A promise problem* $(Y, N) \subseteq \{0,1\}^n$ *is* additive *if it satisfies the following conditions:*

1. *(YES-instances are a linear subspace*[4]*) For every* $x, y \in Y$, $x + y \in Y$
2. *(NO-instances are a collection of cosets) For every* $x \in N, y \in Y$, $x + y \in N$

This definition can be generalized to any abelian group instead of $\{0,1\}^n$. For simplicity, in this work we focus on $\{0,1\}^n$.

Testing linear properties

As a demonstration for the new definition, we next show that property testing, when the tested property $\Pi \subseteq \{0,1\}^n$ is a linear subspace, is an additive promise problem. The YES-instances in this case are the elements of the tested property. That is, $Y = \Pi$. The NO-instances are the elements ϵ-far from every YES-instances. Namely,

[4] Strictly speaking, the requirement is that the YES-instances are a *subgroup*. For $\{0,1\}^n$ these requirements are equivalent.

$$N = \{x \in \{0,1\}^n \mid \forall y \in Y,\ \mathrm{dist}(x,y) > \epsilon\}$$

where $\mathrm{dist}(x,y)$ is the relative Hamming distance between x and y, (i.e., $\mathrm{dist}(x,y) = \frac{|\{x_i \neq y_i | i \in [n]\}|}{n}$).

The first item of Definition 4 follows from the assumptions that the property Π is linear. For the second item, let $x \in N, y \in Y$. We need to show that $x + y \in N$. I.e, that $x + y$ is ϵ-far from every $y' \in Y$. Indeed, for every $y' \in Y$ we have

$$\mathrm{dist}(x + y, y') = \mathrm{dist}(x, y' - y) > \epsilon.$$

The equality holds because in general, $\mathrm{dist}(a,b) = \mathrm{dist}(a + c, b + c)$ for every a, b, c. The inequality holds because, since y and y' are in the linear space Y then $y' - y \in Y$, and since $x \in N$ it is ϵ-far from every element in Y.

In Section 3, we show how to interpret relaxed decoding and correction of linear codes as promise problems. This step might result in performing one additional query. We use similar arguments as in the proof above to show that the resulting promise problems are additive.

The reduction

Next, we construct a reduction from adaptive, two-sided error local algorithms to non-adaptive, one-sided error ones that works for *any* additive promise problem.

▶ **Theorem 5.** *Let $(Y, N) \subseteq \{0,1\}^n$ be an additive promise problem. If (Y, N) has an adaptive algorithm A with completeness error ϵ_Y, soundness error ϵ_N and query complexity q, it has a non-adaptive, one-sided error algorithm A' with soundness error $\epsilon_Y + \epsilon_N$ and query complexity q.*

By applying Theorem 5 to relaxed decoding we prove Theorem 1, and by applying it to relaxed correction we prove Theorem 2.

The reduction works in two steps. The first step ensures that the algorithm never errs on YES-instances. We start with an adaptive, two-sided error arbitrary algorithm, and transform it to have one-sided error (and it remains adaptive). This step does not increase the query complexity. If the original algorithm errs on YES-instances with probability at most ϵ_Y and on NO-instances with probability at most ϵ_N, then the transformed algorithm errs on NO-instances with probability at most $\epsilon_Y + \epsilon_N$.

The second step handles adaptivity. We start with an adaptive, one-sided error algorithm, and transform it to a non-adaptive algorithm (that still has one-sided error). This step maintains the query complexity and the soundness error.

We next describe the two reductions.

Two-sided to one-sided error

Every randomized algorithm A can be described as a distribution over a set of deterministic decision trees. Each leaf of each decision tree is labeled with YES or NO, which is the output of the algorithm when that tree is chosen. The first step of the reduction is to *relabel* the leaves of all trees, in the following way: If there is an input $x \in Y$ that "leads" to this leaf, then it is relabeled to YES. This step is necessary to get a one-sided error algorithm. However, this transformation may not maintain the algorithm's soundness. In Section 4.1, we explain the issue in detail.

The solution is to modify the algorithm. Instead of using the (relabeled) decision trees of A with the given input x, choose a random YES-instance y, and use the tree as if the input was $x + y$. Since (Y, N) is an additive promise problem, if $x \in Y$ then $x + y \in Y$ for any (randomly chosen) y, and the original algorithm's completeness can be used. Similarly, if $x \in N$ then $x + y \in N$ for any y, and the soundness of the original algorithm can be used. In Lemma 16, we prove that with this modification the transformation maintains the sum of soundness and completeness error.

Adaptivity

Consider an adaptive, one-sided error algorithm A. Without loss of generality, the only freedom A has is in choosing its queries. Once it queried an input x, it must output YES if there exists a YES-instance consistent with the queries. Otherwise, when no YES-instance is consistent, then x cannot be a YES-instance and w.l.o.g A outputs NO.

The new non-adaptive algorithm A' works as follows: On input x, choose a random YES-instance y. Query x on all indices A would have queried y, and output YES if the partial view of x is consistent with some YES-instance (which might be different than y).

The new algorithm is non-adaptive since now it determines its queries independently of its input. Its query complexity is maintained, and it has one-sided error (as it always outputs YES for YES-instances). In Lemma 20, we show that its soundness error is also maintained. This is done by relating the probability A' outputs YES on some specific x to the *average* probability A outputs YES for a random element of the set $x + Y$.

1.4 Related work

Error correcting codes date back to the seminal works of Shannon [23] and Hamming [16]. LDCs, LCCs and their relaxed counterparts have attracted significant attention in recent years. See the works of Yekhanin [26] and Kopparty and Saraf [19] and references within for comprehensive surveys of LDCs, LCCs and their applications.

RLDCs and RLCCs constructions

The constructions of RLDCs and RLCCs can be separated into two regimes of parameters: constant query complexity, and constant rate. In the constant rate regime, the state-of-the-art code is the construction of Cohen and Yankovitz [8]. This construction is of a linear RLCC with rate arbitrarily close to 1, and query complexity $q = (\log n)^{2+o(1)}$. This construction builds upon the result of Kumar and Mon [20], which shows a similar code but with query complexity $q = (\log n)^{O(1)}$.

In the constant query regime, the original work of [3] achieves RLDC with constant query complexity $O(q)$ and block length $n = O(k^{1+1/\sqrt{q}})$. The work of [15] introduced the notion of RLCCs, and constructed such a code with constant query complexity, but with worse block length. Chiesa, Gur, and Shinkar [6] constructed an improved RLCC, matching the block length of [3].

The current state-of-the-art construction is of Asadi and Shinkar [2], which builds upon [15] and [6]. Their construction is of RLCC and RLDC with constant query complexity $O(q)$ and block length $n = O(k^{1+1/q})$. This is the first construction of RLDC, in the constant query complexity regime, with better block length than of [3].

We remark that all the above constructions are linear, non-adaptive, and have one-sided error.

Lower bounds

In recent decades, extensive research has been conducted on lower bounds for (non-relaxed) LDCs in various regimes [17, 18, 24, 25, 1]. Gur and Lachish [13] presented the first lower bound for relaxed LDCs. Such lower bounds are arguably harder to obtain, as RLDCs are weaker objects than LDCs. Specifically, they showed that any non-adaptive RLDC requires a block length of $n \geq k^{1+\Omega(\frac{1}{q^2 \log^2 q})}$. For the adaptive case, they established a lower bound of $n \geq k^{1+\Omega(\frac{1}{2^{2q} \log^2 q})}$, by using the known reduction mentioned above, that causes an exponential blowup in the query complexity.

The result of [13] was extended to additional settings, such as proofs of proximity, property testing and to adaptive settings by Dall'Agnol, Gur and Lachish [9]. Their result extends the lower bound of $n \geq k^{1+\Omega(\frac{1}{q^2 \log^2 q})}$ to *adaptive* RLDCs.

Goldreich [10] surveyed and simplified the work of [13], without employing the new techniques of [9]). He established an improved bound of $n \geq k^{1+\Omega(1/q^2)}$ for the non-adaptive case, and a bound of $n \geq k^{1+\Omega(1/q^3)}$ for the adaptive case (which is weaker than the one presented in [9]).

Locally testable codes and RLDCs

Our work follows the theme of extending ideas from the area of *LTCs* (locally testable codes [12]) to RLDCs. Refer to the full paper for an analysis comparing RLDCs and LTCs, including details about their similarities, distinctions, and relevance to this study.

1.5 Open problems

We conclude this section with a few questions we leave for future research.

Non-linear codes

Our result holds only for linear codes. A natural open problem is to extend the result to non-linear codes, or to show that for non-linear codes, adaptivity and / or two-sided error *do* give more power. Our reduction heavily relies on linearity, so we believe other techniques than the ones used in this paper will be needed.

Adaptivity of LDCs and LCCs

The power of adaptivity for (non-relaxed) LDCs and LCCs is an interesting open problem. As we discuss in Section 3, our framework of additive problems does not seem to cover them. This leaves the question open, even for linear codes. To the best of our knowledge, there are currently no known reductions from adaptive to non-adaptive LDCs, besides the ones described above. Even a reduction that polynomially increases the query complexity, or slightly increases the code's block length, would be an interesting result.

A main open problem is to separate the power of LDCs and that of relaxed LDCs. Constructing an adaptive (linear) LDC might aid in this effort, showing that LDCs and relaxed LDC differ regarding adaptivity.

Efficient reduction

The reduction we propose could potentially incur an exponential cost in terms of time complexity. Addressing the challenge of making the reduction efficient, (i.e., ensuring that the transformed algorithm runs in polynomial time), remains an open problem. See Remark 19 for details.

2 Definitions and preliminaries

Refer to the full paper for basic notations used herein.

2.1 Error correcting codes

Throughout, an error correcting code C with *message length* k and *block length* n is a function $C : \{0,1\}^k \to \{0,1\}^n$. For simplicity, we consider only binary alphabet in this work. We identify a code with its image, i.e, $C \subseteq \{0,1\}^n$.

A code C is *linear* if it is a linear function (or, equivalently, if C as a set is closed under addition).

▶ **Definition 6.** *Let $C : \{0,1\}^k \to \{0,1\}^n$ be an error correcting code. A relaxed* decoder *of radius $\rho > 0$ for C is a randomized procedure A, that gets as inputs oracle access to $x \in \{0,1\}^n$, and explicit input $i \in [k]$, outputs an element of $\{0,1,\perp\}$, and satisfies the following two requirements:*

1. *(completeness) If $x = C(y)$ for some $y \in \{0,1\}^k$ then $A^x(i) = y_i$ with probability at least $1 - \epsilon_{completeness}$.*
2. *(relaxed local decoding) If there exists $y \in \{0,1\}^k$ such that $dist(x, C(y)) < \rho$, then $A^x(i) \in \{y_i, \perp\}$ with probability at least $1 - \epsilon_{soundness}$.*

The probabilities are over the internal randomness of A.

▶ **Definition 7.** *Let $C \subseteq \{0,1\}^n$ be an error correcting code. A relaxed* corrector *of radius $\rho > 0$ for C is a randomized procedure A, that gets as inputs oracle access to $x \in \{0,1\}^n$, and explicit input $i \in [n]$, outputs an element of $\{0,1,\perp\}$, and satisfies the following two requirements:*

1. *(completeness) If $x \in C$ then $A^x(i) = x_i$ with probability at least $1 - \epsilon_{completeness}$.*
2. *(relaxed local correction) If there exists $c \in C$ such that $dist(x, c) < \rho$, then $A^x(i) \in \{c_i, \perp\}$ with probability at least $1 - \epsilon_{soundness}$.*

The probabilities are over the internal randomness of A.

In both definitions we call $\epsilon_{\text{completeness}}$ the *completeness error*, and $\epsilon_{\text{soundness}}$ the *soundness error*.

In what follows we use the term *local algorithm* to refer to an algorithm which is a relaxed local decoder or a relaxed local corrector. A local algorithm has *one-sided error* if its completeness error is 0 (i.e., if it never errs on valid codewords). We say that a local algorithm has *query complexity* $q = q(n)$ if, on input i, and with oracle access to any $x \in \{0,1\}^n$, the corrector makes at most $q(n)$ queries. We say that a local algorithm is *non-adaptive* if it determines all its queries based on its explicit input (namely, the index to decode / correct) and internal coin tosses, independently of the specific x to which it is given oracle access. Otherwise, we say that it is *adaptive*.

For an RLDC, we view its decoder A of a code as a set of k decoders $A_1, \ldots, A_k$, where $A_i(x) = A^x(i)$. We call A_i the *decoders* of C. Similarly, we view the corrector of a RLCC as a set of n correctors, $A_1, \ldots, A_n$, and call them the *correctors* of C. The benefit of this view is that each A_i is now an algorithm that gets a single (implicit) input $x \in \{0,1\}^n$.

2.2 Promise problems

▶ **Definition 8.** *A* Promise Problem *is couple $(Y, N) \subseteq \{0,1\}^n$ such that $Y \cap N = \emptyset$. We call Y the* YES-instances *of the problem, and N the* NO-instances *of the problem.*

▶ **Definition 9.** *An* algorithm for a promise problem $(Y, N) \subseteq \{0,1\}^n$ *with* completeness error $\epsilon_Y > 0$ *and* soundness error $\epsilon_N > 0$ *is a randomized procedure that gets oracle access to an input* $x \in \{0,1\}^n$*, outputs YES or NO, and satisfies the following conditions:*

1. *(completeness) If* $x \in Y$ *then it outputs YES with probability at least* $1 - \epsilon_Y$*.*
2. *(soundness) If* $x \in N$ *then its outputs NO with probability at least* $1 - \epsilon_N$*.*

We define query complexity and adaptivity of promise problem algorithms as they are defined in Definition 7. We say that a promise problem algorithm has *one-sided* error if its completeness error is 0 (i.e., if it never errs on YES-instances).

The main new definition of this work is of additive promise problems. See Definition 4.

3 Relaxed decoding and correction as additive promise problems

In this section, we show how to interpret relaxed decoding and relaxed correction of linear codes as promise problems. We do that in Section 3.1. We then show, in Section 3.2, that the resulting promise problems are additive.

3.1 Interpretation as promise problems

We start by showing how to interpret relaxed correction as a promise problem. There are three possible values for the output of a relaxed local corrector: $0, 1$ or $\perp$.[5] In contrast, an algorithm for a promise problem has only two possible outputs: YES and NO. We must specify how to translate a correction problem (with multiple possible output values) into a yes/no question. The following observation enables us to do that.

▷ Claim 10. If a code has a corrector A, then it has a corrector A' such that for every $x \in \{0,1\}^n$, the output of A' for index i is x_i or $\perp$. A' has the same completeness and soundness errors as A, and it might make one additional query.

Proof. The new corrector A'_i works according to the following rule:

$$A'_i(x) = \begin{cases} A_i(x), & \text{if } A_i(x) \in \{x_i, \perp\} \\ \perp, & \text{otherwise.} \end{cases}$$

From the construction, the output of $A'_i(x)$ is x_i or $\perp$ for every input x (and never $1 - x_i$). A' makes at most one additional query compared to A in order to retrieve the value x_i.

We next show that A'_i satisfies the required completeness and soundness (Definition 7). For completeness, if $x \in C$ then $A_i(x) = x_i$ with probability $1 - \epsilon_{\text{completeness}}$, and hence $A'_i(x) = x_i$ with the same probability.

For soundness, assume there exists $c \in C$ such that $\text{dist}(x, c) < \rho$. We need to show $A'_i(x) \in \{c_i, \perp\}$ with high probability. Consider the case $c_i = x_i$. From the soundness of A_i, $A_i(x) \in \{x_i, \perp\}$ with probability at least $1 - \epsilon_{\text{soundness}}$, and hence $A'_i(x) \in \{x_i, \perp\} = \{c_i, \perp\}$ with the same probability.

Otherwise, $c_i \neq x_i$. With probability at least $1 - \epsilon_{\text{soundness}}$, the output of A_i is c_i or $\perp$. From the construction, whenever the output of A_i is c_i or $\perp$, the output of A'_i is $\perp$. Hence, with the same probability, $A'_i(x) = \perp \in \{c_i, \perp\}$. ◁

By using Claim 10, we can replace item 2 of Definition 7, and assume the corrector always outputs x_i or $\perp$:

[5] For a larger alphabet, the number of possible outputs of a corrector is the size of the alphabet +1.

▶ **Definition 11** (alternative definition of RLCCs). *A relaxed corrector of radius $\rho > 0$ for C is a randomized procedure A, that gets as inputs oracle access to $x \in \{0,1\}^n$, and explicit input $i \in [n]$, outputs x_i or $\perp$, and satisfies the following:*

1. *(completeness) If $x \in C$ then $A^x(i) = x_i$ with probability at least $1 - \epsilon_{completeness}$.*
2. *(soundness) If there exists $c \in C$ such that $dist(x, c) < \rho$ and $x_i \neq c_i$, then $A_i(x) = \perp$ with probability at least $1 - \epsilon_{soundness}$.*

Definition 11 allows us to treat a corrector as having a "binary" output; On input x, the output of A_i is either $x_i \in \{0,1\}$ (signifying "accept") or $\perp$ (signifying "reject"). We can now phrase relaxed correction as a promise problem, as follows[6]:

▶ **Definition 12.** *Let $C \subset \{0,1\}^n$ be an error correcting code, let $\rho > 0$ and let $i \in [n]$. The* promise problem of relaxed correction *of C at index i with correction radius ρ is defined by:*

1. *(YES-instances are the codewords)*
   $Y = C$
2. *(NO-instances are the inputs a corrector rejects)*
   $N = \{x \in \{0,1\}^n \mid \exists c \in C \text{ with } dist(x, c) < \rho \text{ and } x_i \neq c_i\}$.

The promise problem of relaxed correction is equivalent to relaxed correction of codes (of Definition 11) in the following sense:

▷ Claim 13. A corrector for index i of the code C can be translated to an algorithm for the promise problem of relaxed correction of C at index i (Definition 12) by identifying the outputs $0, 1$ as "YES" and the output $\perp$ as "NO". The parameters $\epsilon_{\text{completeness}}, \epsilon_{\text{soundness}}, \rho$ remain the same.

Relaxed decoding

From the hypothesis for RLDCs, the code at hand is in systematic form. That is, we assume that the first k bits of each codeword are the message encoded in it. Hence, a decoder is simply a corrector that needs to "correct" only the first k bits of the input. The observation above (Claim 10) holds for relaxed local decoders as well. Hence, we can assume w.l.o.g that the output of the decoder, for input x and index $i \in [k]$, is either x_i or $\perp$. This allows us to handle RLDCs in a manner similar to RLCCs. For further details, please refer to the full paper."

3.2 Additive promise problems

In this section, we show that the promise problems formulated above for relaxed decoding and relaxed correction, have the special property of being *additive* (Definition 4).

▷ Claim 14. The relaxed correction and relaxed decoding promise problems for linear codes is additive.

We prove the claim for relaxed correction. The proof for relaxed decoding is the same, with the restriction that $i \in [k]$ (instead of in $[n]$).

[6] Another possible formulation is $Y = \{(x, b) \mid x \in C \text{ and } b = x_i\}$ and $N = \{(x, b) \mid \exists c \in C \text{ with } \text{dist}(x, c) < \rho \text{ and } c_i \neq b\}$. This formulation preserves better the decoding "flavor" of the problem. We use the formulation of Definition 11, as it emphasizes the similarity to testing, with $\perp$ corresponding to "reject" and any other output corresponding to "accept".

Proof. Let C be a *linear* error correcting code, let $\rho > 0$ and let $i \in [n]$. Let Y, N be as in Definition 12.

From the linearity assumption $Y = C$ is a linear subspace of $\{0,1\}^n$, and the first item of Definition 4 holds.

To show the second item, let $x \in N, y \in Y$. We need to show that $x + y \in N$. Since $x \in N$, there exists a codeword $c \in C$ such that $\text{dist}(x, c) < \rho$ and $x_i \neq c_i$. Define $c' = c + y$. c' is a codeword in C, since it is a sum of two codewords, and C is a linear code. We get $\text{dist}(x + y, c') = \text{dist}(x + y, c + y) = \text{dist}(x, c) < \rho$, and $(x + y)_i = x_i + y_i \neq c_i + y_i = c'_i$. Hence $x + y \in N$. ◁

Non-relaxed LDCs

We remark that *non-relaxed* decoding / correction (for a specific index) does not seem to fit into our new framework. Refer to the full paper for an in-depth discussion on this topic.

4 The reduction

In this section, we prove our main results, Theorem 1 and Theorem 2.

We prove these theorems by constructing the following, more general reduction:

▶ **Theorem 15** (Restatement of Theorem 5). *Let $(Y, N) \subseteq \{0,1\}^n$ be an additive promise problem. If (Y, N) has an adaptive algorithm A with completeness error ϵ_Y, soundness error ϵ_N and query complexity q, it has a* one-sided error, non-adaptive algorithm *A' with soundness error $\epsilon_Y + \epsilon_N$ and query complexity q.*

Theorem 1 and Theorem 2 are direct corollary of Theorem 15 applied to the relaxed decoding and correcting promise problem.

The proof of Theorem 15 has two steps. The first is a reduction from two-sided error to one-sided error algorithms. We show this reduction in Section 4.1. The second step is a reduction from one-sided, adaptive algorithms to one-sided, non-adaptive algorithms. We show this reduction in Section 4.2.

4.1 From two-sided to one-sided error

In this section, we show a reduction from two-sided error algorithms to one-sided error ones for additive promise problems. The reduction does not change the query complexity of the algorithm, and maintains the sum of the completeness and soundness errors.

▶ **Lemma 16.** *Let $(Y, N) \subseteq \{0,1\}^n$ be an additive promise problem. If (Y, N) has an (adaptive) algorithm A with completeness error ϵ_Y, soundness error ϵ_N and query complexity q, it has an (adaptive) one-sided error algorithm A' with soundness error $\epsilon_Y + \epsilon_N$ and query complexity q.*

Randomized algorithms as distributions over decision trees

Consider some randomized algorithm A for a promise problem. A can be described as a distribution D_A over a set of deterministic decision trees $\Upsilon_A = \{\Gamma_1, \Gamma_2, \ldots\}$. We denote by $\Gamma \sim D_A$ a tree chosen randomly from Υ_A according to the distribution D_A. Each leaf ℓ of each tree corresponds to a set of indices $I = (i_1, \ldots, i_t) \in [n]^t$ that are queried along the path leading to ℓ, and the corresponding values $\sigma = (\sigma_1, \ldots, \sigma_t) \in \{0,1\}^t$ at these indices. We identify each leaf with the corresponding indices and values and write $\ell = (I, \sigma)$.

For an input x, we denote by $\Gamma(x)$ the *leaf* of Γ at the end of the path corresponding to querying x. That is, $\Gamma(x) = \ell$ if $x|_I = \sigma$, where $\ell = (I, \sigma)$, and $x|_I$ is the restriction of x to indices I. Each leaf is labeled with YES or NO.

We can now describe the operation of A on input x as follows: It chooses a $\Gamma \sim D_A$ and outputs the label of $\Gamma(x)$.

Relabeling decision trees

To convert an algorithm A to have one-sided error, we first go over all of $\Gamma \in \Upsilon_A$ and relabel them, so each Γ will have one-sided error. The relabeling works as follows: For every leaf ℓ of Γ, if there exists a YES-instance y such that $\Gamma(y) = \ell$, relabel ℓ with YES. We denote the relabeled tree by Γ_c and call it the *one-sided error relabeling* of Γ.

This step is necessary to get a one-sided error algorithm; as long as there exists $y \in Y$ and Γ such that the label of $\Gamma(y)$ is NO, there is some probability that A outputs NO for a YES-instance. The issue, however, is that this transformation may not maintain the algorithm's soundness. There might be NO-instances that the new algorithm (wrongly) accepts with a high probability.

The soundness issue with relabeling

Pretend the transformed algorithm A' worked as follows. For an input x, choose a random decision tree $\Gamma \sim D_A$, relabel the tree to get Γ_c, and output the label of $\Gamma_c(x)$ (instead of using $\Gamma(x)$ as the original algorithm did). This new algorithm has one-sided error. In fact, we can transform *any* algorithm this way to have one-sided error, so we should not expect it to maintain soundness.

To see that the soundness is not necessarily maintained, consider some leaf ℓ of a tree Γ that was relabeled from NO to YES. Let $x \in \{0,1\}^n$ such that $\Gamma(x) = \ell$. If $x \in Y$, the relabeling was beneficial. Before the relabeling, the algorithm returned a wrong output for x (whenever it used the tree Γ), and now it returns the correct output. However, what if x is a NO-instance? In this case, after the relabeling we return the *wrong* output for x each time Γ is used. The tree Γ is sampled according to the distribution D_A, which is arbitrary. If D_A gives much weight to Γ (say, it is chosen with probability $\frac{1}{2}$), then the new algorithm returns the wrong output for x with high probability.[7] This implies that A' does not have the required soundness property, as it should hold for *every* NO-instance.

The actual reduction

The solution is to modify the algorithm in the following way. Instead of deterministically returning the label of $\Gamma_c(x)$ (after Γ was chosen at random), the transformed algorithm outputs the label of $\Gamma_c(x + y)$ for a random $y \in Y$. We give a formal description of the transformation in Algorithm 1. Now, even if $\Gamma_c(x)$ was relabeled to YES, we output its label with a small probability. In the proof of Lemma 16, we show that the soundness error of the transformed algorithm is at most $\epsilon_Y + \epsilon_N$.[8]

[7] Even if there is no one heavy tree, relabels of many leaves in different trees might have the same effect if their total weight is high.

[8] Recall that ϵ_N is error probability of the original algorithm for NO-instances, and ϵ_Y is its error probability for YES-instances.

Algorithm 1 One-sided error local algorithm, A'.

Input: Oracle access to $x \in \{0,1\}^n$.
Output: YES or NO.

1. Choose $\Gamma \sim D_A$.
2. Choose $y \in Y$ uniformly at random.
3. Output the label of $\Gamma_c(x+y)$.

Query complexity

The relabeling does not change the query complexity (i.e., depth) of the decision trees. Hence, the query complexity of A' is the same as that of A.

One-sided error

Let $x \in Y$. We argue that A' always outputs YES for x. From the definition of A', its output for x is the label of $\Gamma_c(x+y)$ for some $y \in Y$. Since (Y,N) is an additive problem and $x \in Y$, we get that $x+y \in Y$. From the relabeling scheme for Γ_c, the label of $\Gamma_c(y')$ is YES for every $y' \in Y$, and in particular for $y' = x+y$. Hence, the output of A' for x is YES.

Before arguing about the soundness of the transformed algorithm, we need the following preparations.

Probability of hitting a specific leaf

We stated above that for an input x, the modified algorithm uses the label of a specific leaf ℓ with a small probability. We next calculate this probability (conditioning on first choosing the tree Γ_c of ℓ). There are $|Y|$ possible options for y. The leaf ℓ is used if the chosen y satisfies $\Gamma_c(x+y) = \ell$. Hence, we choose the label of ℓ with probability $\frac{|(x+Y)\cap\Gamma^{-1}(\ell)|}{|Y|}$, where $x+Y = \{x+y \mid y \in Y\} \subseteq \{0,1\}^n$, and $\Gamma^{-1}(\ell) = \{z \mid \Gamma(z) = \ell\} \subseteq \{0,1\}^n$. We argue that this probability is either 0 (when there is no $y \in Y$ such that $\Gamma(x+y) = \ell$ so $(x+Y)\cap\Gamma^{-1}(\ell)$ is empty), or is equal to a quantity not depending on x.

▶ **Lemma 17.** *Let $Y \subseteq \{0,1\}^n$ be a subspace of $\{0,1\}^n$, and fix a decision tree Γ and a leaf $\ell = (I,\sigma)$. Define $U = \{u \in Y \mid u|_I = 0\}$. Then for every $x \in \{0,1\}^n$, if there exists $y \in Y$ such that $\Gamma(x+y) = \ell$, then $|(x+Y)\cap\Gamma^{-1}(\ell)| = |U|$.*

Proof. First, notice that U is not empty since the all-zeros string is in Y (as Y is a linear subspace). Furthermore, U is a subspace of Y, since if $u,u' \in U$ then $(u+u')|_I = u|_I + u'|_I = 0$ and $u+u' \in U$. We argue that $(x+Y)\cap\Gamma^{-1}(\ell)$ is a coset of U, hence having the same size as U. Namely, we claim:

$$(x+Y)\cap\Gamma^{-1}(\ell) = x+y+U$$

where y is an element of Y such that $\Gamma(x+y) = \ell$.

We begin by proving the inclusion $(x+Y)\cap\Gamma^{-1}(\ell) \subseteq x+y+U$. Let $x+y' \in (x+Y)\cap\Gamma^{-1}(\ell)$. That is, $(x+y')|_I = \sigma$. Define $u = y'-y$. Since $(x+y)|_I = (x+y')|_I = \sigma$, we have $u|_I = ((x+y')-(x+y))|_I = 0$ and $u \in U$. Therefore $x+y' = x+y+u \in x+y+U$.

To prove that $(x+Y)\cap\Gamma^{-1}(\ell) \supseteq x+y+U$, let $u \in U$. Since $y \in Y$ and $u \in U \subseteq Y$, and Y is closed under addition, we have $y+u \in Y$ and $x+y+u \in x+Y$. Next, $(x+y+u)|_I = (x+y)_I + u|_I = (x+y)_I = \sigma$ and hence $x+y+u \in \Gamma^{-1}(\ell)$. ◀

We are now ready to prove Lemma 16.

Proof of Lemma 16. We proved above that the reduction maintains the query complexity of the algorithm, and that the transformed algorithm has one-sided error. We are left with establishing an upper bound for its soundness error.

Let $x \in N$. We need to show that A' outputs NO for x with probability at least $1-\epsilon_{\mathrm{Y}}-\epsilon_{\mathrm{N}}$. That is, we need to prove $\Pr[A'(x) = \text{ YES }] < \epsilon_{\mathrm{Y}} + \epsilon_{\mathrm{N}}$.

The probability for (wrongly) outputting YES for x may increase due to the transformation. Nevertheless, it does not increase too much. We argue the transformation does not decrease the gap between the expected probability of returning YES for a random element of Y and the expected probability of outputting YES for a random element of $x + Y$. That is, we claim:

▷ Claim 18. For every $x \in N$:

$$\begin{aligned}\mathop{\mathbb{E}}_{y\in Y}[\Pr[A(y) = \mathrm{YES}]] - \mathop{\mathbb{E}}_{y\in Y}[\Pr[A(x+y) = \mathrm{YES}]] \\ \leq \quad \mathop{\mathbb{E}}_{y\in Y}[\Pr[A'(y) = \mathrm{YES}]] - \mathop{\mathbb{E}}_{y\in Y}[\Pr[A'(x+y) = \mathrm{YES}]]\end{aligned}$$

where the probabilities are over the internal randomness of A and A'.

Proof of Claim 18. To prove this claim, it is enough to show that relabeling one leaf ℓ of one decision tree does not decrease the gap. Then we obtain the claim by relabeling one leaf at a time, to get A' from A.

Assume ℓ was relabeled from NO to YES. Let $G := Y \cap \Gamma^{-1}(\ell)$ be the strings $y \in Y$ such that $\Gamma(y) = \ell$ (these are the "Good" strings, which are now labeled YES and are in Y). The set G is not empty since we relabel ℓ only if there exists $y \in Y$ such that $\Gamma(y) = \ell$, i.e. $y \in G$. Let $B := (x + Y) \cap \Gamma^{-1}(\ell)$ be the strings $x + y \in x + Y$ such that $\Gamma(x + y) = \ell$ (these are the "Bad" strings, which are now labeled YES but in $x + Y \subseteq N$).

Every string in $G \cup B$ was rejected before the relabeling but is now accepted. The algorithm's behavior on the other elements in Y and $x+Y$ is unaltered. Hence, $\mathop{\mathbb{E}}_{y\in Y}[\Pr[A(y) = \mathrm{YES}]]$ increases by $D(\Gamma) \cdot \frac{|G|}{|Y|}$ when ℓ is relabeled (recall that $D(\Gamma)$ is the probability the algorithm chooses Γ). Similarly, $\mathop{\mathbb{E}}_{y\in Y}[\Pr[A(x + y) = \mathrm{YES}]]$ increases by $D(\Gamma) \cdot \frac{|B|}{|Y|}$. It suffices to show that $|G| \geq |B|$. Intuitively, this means any "harm" done to x by the relabeling (an element of B that increases the probability to wrongly output YES) is "compensated" by the relabeling (by an element of G improving the algorithm's completeness).

If B is empty, we are done. Otherwise, due to Lemma 17, $|B| = |U|$ and $|G| = |U|$, and we conclude that $|G| = |B|$. ◁

From the claim, the soundness error for x gets worse by an amount bounded by the completeness improvement. Since the completeness error reduces from ϵ_{Y} to 0, the soundness error for x increases by at most ϵ_{Y}.

More formally, from Claim 18, the prefect completeness of A' and the completeness error of A:

$$\begin{aligned}&\mathop{\mathbb{E}}_{y\in Y}[\Pr[A'(x+y) = \mathrm{YES}]] \\ &\quad\leq \quad \mathop{\mathbb{E}}_{y\in Y}[\Pr[A'(y) = \mathrm{YES}]] - \mathop{\mathbb{E}}_{y\in Y}[\Pr[A(y) = \mathrm{YES}]] + \mathop{\mathbb{E}}_{y\in Y}[\Pr[A(x+y) = \mathrm{YES}]] \qquad (1) \\ &\quad\leq \quad 1 - (1-\epsilon_{\mathrm{Y}}) + \mathop{\mathbb{E}}_{y\in Y}[\Pr[A(x+y) = \mathrm{YES}]] = \mathop{\mathbb{E}}_{y\in Y}[\Pr[A(x+y) = \mathrm{YES}]] + \epsilon_{\mathrm{Y}}.\end{aligned}$$

Now, $x + y \in N$ for every $y \in Y$, since $x \in N$ and (Y, N) is additive. Hence, applying the soundness of A to *every* $x + y$, we have $\mathop{\mathbb{E}}_{y \in Y}[\Pr[A(x + y) = \text{ YES }]] < \epsilon_{\mathrm{N}}$.

In addition, from the definition of A', $\Pr[A'(x + y) = \text{YES}] = \Pr[A'(x) = \text{YES}]$ for every $y \in Y$. Hence, from equation 1 we get that $\Pr[A'(x) = \text{YES}] < \epsilon_{\mathrm{Y}} + \epsilon_{\mathrm{N}}$. ◀

▶ Remark 19. As described, the reduction might be expansive in the terms of time complexity. Sampling a YES instance uniformly at random and relabeling the trees can be computationally intensive tasks.

On the positive side, sampling a YES instance can be implemented in polynomial time. The set of YES instances Y forms a subspace, allowing for efficient sampling by taking a random linear combination of elements in a basis of Y. In the context of codes, this is equivalent to encoding a random message using a generating matrix of the code, where the columns of the matrix constitute a basis for the code.

Unfortunately, the relabeling process, even for a single leaf, appears to be computationally hard. Brute-force relabeling involves iterating over a potentially exponentially large set of candidates (all elements of $\{0,1\}^n$ consistent with the leaf) and checking their membership in Y. Checking membership in Y can be preformed efficiently by considering the dual space of Y. However, we do not see a way to eliminate the iteration over an exponential set of candidates, and leave this question for further research.

4.2 From adaptive to non-adaptive algorithms

In this section, we show a reduction from one-sided error adaptive to (one-sided error) non-adaptive algorithms for additive promise problems. The reduction maintains the algorithm's query complexity and its soundness error.

▶ **Lemma 20.** *Let $(Y, N) \subseteq \{0,1\}^n$ be an additive promise problem. If (Y, N) has an* one-sided error, adaptive algorithm *A with soundness error ϵ_N and query complexity q, it has an one-sided error* non-adaptive algorithm *A' with soundness error ϵ_N and query complexity q.*

Proof. Let $D = D_A$ be a distribution over decision trees $\Upsilon = \Upsilon_A$ corresponding to A.

We first observe that we can assume w.l.o.g that the label of each leaf $\ell = (I, \sigma)$ is YES if and only if $\exists y \in Y$ such that $y|_I = \sigma$. Since A never errs on YES-instances, if there exists $y \in Y$ such that $y|_I = \sigma$, then ℓ must be labeled YES. On the other hand, we can assume that if no such $y \in Y$ exists, then ℓ is labeled NO. Otherwise, ℓ can be relabeled from YES to NO while only improving the algorithm's soundness and maintaining its one-sided error.

Description of A'

The new non-adaptive algorithm works as follows. On input x, choose a random $y \in Y$. Query x on all indices A would have queried y, and output YES if the partial view of x is consistent with some $y' \in Y$. We give a formal description of the new algorithm in Algorithm 2.

Analysis

The algorithm A' is non-adaptive, as its queries depend only on its internal randomness (the choice of Γ and y). It has the same query complexity as A, since it uses the same decision trees. The algorithm has one-sided error since if $x \in Y$, we can take $y' = x$ at the last step of the algorithm, and $x|_I = y'|_I$ for every I.

Algorithm 2 Non-adaptive local algorithm, A'.

Input: Oracle access to $x \in \{0,1\}^n$.
Output: YES or NO.

1. Choose $\Gamma \sim D$.
2. Choose $y \in Y$ uniformly at random.
3. Let $\ell = (I, \sigma) = \Gamma(y)$.
4. Query x on the indices I.
5. Output "YES" if $\exists y' \in Y$ such that $x|_I = y'|_I$, and "NO" otherwise.

We are left with proving that the transformation does not decrease the soundness error. Towards this end, we relate the acceptance probability of A' to the *average* acceptance probability of A.

▷ Claim 21. For every $x \in \{0,1\}^n$:

$$\Pr[A'(x) = \text{YES}] = \mathop{\mathbb{E}}_{y \in Y}[\Pr[A(x+y) = \text{YES}]]$$

where the probabilities are over the internal randomness of A and A'.

This claim shows that soundness is maintained. If $x \in N$, then since (Y, N) is additive, $x + y \in N$ for every $y \in Y$. Hence, by using the soundness of A for *every* $x + y$, we get that

$$\Pr[A'(x) = \text{YES}] = \mathop{\mathbb{E}}_{y \in Y}[\Pr[A(x+y) = \text{YES}]] < \epsilon_{\text{N}}$$ ◀

Proof of Claim 21. We begin by calculating $\mathop{\mathbb{E}}_{y \in Y}[\Pr[A(x+y) = \text{YES}]]$. Identifying the output YES with 1, we can take expectation instead of probability. We denote by $\Gamma(Y)$ the set of leaves in Γ labeled YES[9] , and get:

$$\begin{aligned}\mathop{\mathbb{E}}_{y \in Y}[\Pr[A(x+y) = \text{YES}]] &= \mathop{\mathbb{E}}_{y \in Y} \mathop{\mathbb{E}}_{\Gamma \sim D}[A(x+y)] = \mathop{\mathbb{E}}_{\Gamma \sim D} \mathop{\mathbb{E}}_{y \in Y}[A(x+y)] \\ &= \mathop{\mathbb{E}}_{\Gamma \sim D}\left[\frac{1}{|Y|} \cdot \sum_{\ell \in \Gamma(Y)} |(x+Y) \cap \Gamma^{-1}(\ell)|\right] \end{aligned} \tag{2}$$

where in the last equality, we take into account all $y \in Y$ by iterating over each leaf $\ell \in \Gamma(Y)$ (for other leaves the algorithm's output is 0) and counting the number of y values for which $\Gamma(x+y) = \ell$ (i.e., that lead the algorithm to output the label of ℓ).

On the other hand, for any $I \subseteq \{0,1\}^n$ and $x \in \{0,1\}^n$ define:

$$H_I(x) = \begin{cases} 1, & \text{if } \exists y \in Y \text{ such that } x|_I = y|_I \\ 0, & \text{otherwise.} \end{cases}$$

With this notation, the last step of A' can be described as "output $H_I(x)$".

[9] This definition is equivalent to $\Gamma(Y) = \{\Gamma(y) \mid y \in Y\}$ since, as discussed above, a leaf ℓ is labeled YES if and only if there exists $y \in Y$ such that $\Gamma(y) = \ell$.

We get:

$$\Pr[A'(x) = \text{YES}] = \mathbb{E}[A'(x)] = \mathop{\mathbb{E}}_{\Gamma\sim D} \mathop{\mathbb{E}}_{\substack{y\in Y \\ (I,\sigma)\leftarrow\Gamma(y)}} [H_I(x)]$$

$$= \mathop{\mathbb{E}}_{\Gamma\sim D}\left[\frac{1}{|Y|}\cdot \sum_{\ell=(I,\sigma)\in\Gamma(Y)} |Y\cap\Gamma^{-1}(\ell)|\cdot H_I(x)\right] \tag{3}$$

Here it is sufficient to iterate over the leaves in $\Gamma(Y)$: the algorithm A never errs on YES instances, so if ℓ is labeled NO, there cannot be $y \in Y$ such that $\Gamma(y) = \ell$.

From equations 2 and 3 it is enough to show that for every $\ell = (I, \sigma) \in \Gamma(Y)$:

$$|(x + Y) \cap \Gamma^{-1}(\ell)| = |Y \cap \Gamma^{-1}(\ell)| \cdot H_I(x)$$

Since (I, σ) is labeled YES, and as discussed above, there exists $y' \in Y$ such that $y'|_I = \sigma$ and $Y \cap \Gamma^{-1}(\ell)$ is not empty.

Consider the case $H_I(x) = 0$. We claim $(x + Y) \cap \Gamma^{-1}(\ell)$ is empty and hence the equality holds. Assume towards contradiction that this set is not empty. Then there exists $(x + y) \in (x + Y)$ such that $(x + y)|_I = \sigma$. Hence $(x + y)|_I = y'|_I$ and $x|_I = (y' - y)|_I$, which implies $H_I(x) = 1$ (since $y' - y \in Y$).

Next, consider the case $H_I(x) = 1$. We argue that $(x + Y) \cap \Gamma^{-1}(\ell)$ is not empty. $H_I(x) = 1$ implies there exists $y \in Y$ such that $x|_I = y|_I$, and $(x - y)|_I = 0$. Now $(x-y+y')|_I = (x-y)|_I + y'|_I = \sigma$, and hence $x-y+y' \in (x+Y)\cap\Gamma^{-1}(\ell)$ (as $-y+y' \in Y$). Since $(x + Y) \cap \Gamma^{-1}(\ell)$ is not empty, from Lemma 17 we get that $|(x + Y) \cap \Gamma^{-1}(\ell)| = |U|$. The set $Y \cap \Gamma^{-1}(\ell)$ is also not empty, and again from Lemma 17 $|Y \cap \Gamma^{-1}(\ell)| = |U|$. We conclude that $|(x + Y) \cap \Gamma^{-1}(\ell)| = |Y \cap \Gamma^{-1}(\ell)|$. ◁

▶ Remark 22. As in the previous reduction, the adaptive to non-adaptive reduction might be expansive in the terms of time complexity. Similar to before, sampling a YES instance uniformly at random might appear computationally intensive, although it can be implemented in polynomial time without much difficulty.

Unfortunately, determining whether there exists a YES instance consistent with a partial view of the input (step 5 in Algorithm 2) is computationally hard. We currently do not see a feasible way to perform this task efficiently, for reasons similar to those outlined in Remark 19. We leave addressing the efficiency of this reduction as an open problem.

References

1. Omar Alrabiah, Venkatesan Guruswami, Pravesh Kothari, and Peter Manohar. A near-cubic lower bound for 3-query locally decodable codes from semirandom CSP refutation. *Electron. Colloquium Comput. Complex.*, TR22-101, 2022.
2. Vahid R. Asadi and Igor Shinkar. Relaxed locally correctable codes with improved parameters. In *ICALP*, volume 198 of *LIPIcs*, pages 18:1–18:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
3. Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust pcps of proximity, shorter pcps, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006. doi:10.1137/S0097539705446810.
4. Eli Ben-Sasson, Prahladh Harsha, and Sofya Raskhodnikova. Some 3cnf properties are hard to test. In *STOC*, pages 345–354. ACM, 2003.
5. Manuel Blum and Sampath Kannan. Designing programs that check their work. *J. ACM*, 42(1):269–291, 1995.

6 Alessandro Chiesa, Tom Gur, and Igor Shinkar. Relaxed locally correctable codes with nearly-linear block length and constant query complexity. In *SODA*, pages 1395–1411. SIAM, 2020.

7 Gil Cohen and Tal Yankovitz. Relaxed locally decodable and correctable codes: Beyond tensoring. *Electron. Colloquium Comput. Complex.*, TR22-045, 2022. `arXiv:TR22-045`.

8 Gil Cohen and Tal Yankovitz. Asymptotically-good RLCCs with $(\log(n))^{2+o(1)}$ queries. *Electron. Colloquium Comput. Complex.*, TR23-110, 2023.

9 Marcel de Sena Dall'Agnol, Tom Gur, and Oded Lachish. A structural theorem for local algorithms with applications to coding, testing, and privacy. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1651–1665. SIAM, 2021. `doi:10.1137/1.9781611976465.100`.

10 Oded Goldreich. On the lower bound on the length of relaxed locally decodable codes. *Electron. Colloquium Comput. Complex.*, TR23-064, 2023.

11 Oded Goldreich, Tom Gur, and Ilan Komargodski. Strong locally testable codes with relaxed local decoders. In *CCC*, volume 33 of *LIPIcs*, pages 1–41. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.

12 Oded Goldreich and Madhu Sudan. Locally testable codes and pcps of almost-linear length. *J. ACM*, 53(4):558–655, 2006.

13 Tom Gur and Oded Lachish. On the power of relaxed local decoding algorithms. In *SODA*, pages 1377–1394. SIAM, 2020.

14 Tom Gur, Govind Ramnarayan, and Ron Rothblum. Relaxed locally correctable codes. *Electron. Colloquium Comput. Complex.*, TR17-143, 2017.

15 Tom Gur, Govind Ramnarayan, and Ron Rothblum. Relaxed locally correctable codes. *Theory Comput.*, 16:1–68, 2020. `doi:10.4086/toc.2020.v016a018`.

16 Richard Wesley Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29:147–160, 1950.

17 Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 80–86. ACM, 2000. `doi:10.1145/335305.335315`.

18 Iordanis Kerenidis and Ronald de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. In *STOC*, pages 106–115. ACM, 2003.

19 Swastik Kopparty and Shubhangi Saraf. Local testing and decoding of high-rate error-correcting codes. *Electron. Colloquium Comput. Complex.*, TR17-126, 2017.

20 Vinayak M. Kumar and Geoffrey Mon. Relaxed local correctability from local testing. *CoRR*, abs/2306.17035, 2023.

21 Richard J. Lipton. Efficient checking of computations. In *STACS*, volume 415 of *Lecture Notes in Computer Science*, pages 207–215. Springer, 1990.

22 Noga Ron-Zewi and Ron D. Rothblum. Local proofs approaching the witness length. *IACR Cryptol. ePrint Arch.*, page 1062, 2019.

23 C.E. Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949. `doi:10.1109/JRPROC.1949.232969`.

24 David P. Woodruff. New lower bounds for general locally decodable codes. *Electron. Colloquium Comput. Complex.*, TR07-006, 2007.

25 David P. Woodruff. A quadratic lower bound for three-query linear locally decodable codes over any field. *J. Comput. Sci. Technol.*, 27(4):678–686, 2012.

26 Sergey Yekhanin. Locally decodable codes. *Found. Trends Theor. Comput. Sci.*, 6(3):139–255, 2012.

Sharp Noisy Binary Search with Monotonic Probabilities

Lucas Gretta
University of California, Berkeley, CA, USA

Eric Price
University of Texas at Austin, TX, USA

Abstract

We revisit the noisy binary search model of [10], in which we have n coins with unknown probabilities p_i that we can flip. The coins are sorted by increasing p_i, and we would like to find where the probability crosses (to within ε) of a target value τ. This generalized the fixed-noise model of [2], in which $p_i = \frac{1}{2} \pm \varepsilon$, to a setting where coins near the target may be indistinguishable from it. It was shown in [10] that $\Theta(\frac{1}{\varepsilon^2} \log n)$ samples are necessary and sufficient for this task.

We produce a *practical* algorithm by solving two theoretical challenges: high-probability behavior and sharp constants. We give an algorithm that succeeds with probability $1 - \delta$ from

$$\frac{1}{C_{\tau,\varepsilon}} \cdot \left(\log_2 n + O(\log^{2/3} n \log^{1/3} \frac{1}{\delta} + \log \frac{1}{\delta}) \right)$$

samples, where $C_{\tau,\varepsilon}$ is the optimal such constant achievable. For $\delta > n^{-o(1)}$ this is within $1 + o(1)$ of optimal, and for $\delta \ll 1$ it is the first bound within constant factors of optimal.

2012 ACM Subject Classification Theory of computation → Randomness, geometry and discrete structures; Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Lower bounds and information complexity

Keywords and phrases fine-grained algorithms, randomized/probabilistic methods, sublinear/streaming algorithms, noisy binary search

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.75

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2311.00840`

Funding *Lucas Gretta*: NSF Award CCF-2231095. Work partially done while at UT Austin.
Eric Price: NSF awards CCF-2008868, CCF-1751040 (CAREER), and the NSF AI Institute for Foundations of Machine Learning (IFML).

1 Introduction

Binary search is one of the most fundamental algorithms in computer science, finding an index $i^* \in [n]$ from $\log_2 n$ queries asking if a given index i is larger than i^*. But what if the queries are noisy?

One model for noisy binary search has each query be incorrect independently with exactly the same probability $\frac{1}{2} - \varepsilon$. In this model, which we call FixedNoiseNBS, a line of work [2, 1, 5, 9] has found a sharp bound for the required expected sample complexity, with tight constants. However, in many applications of noisy binary search the error probability is not fixed, but varies with i: comparing i to i^* is much harder when i is close to i^*.

As one example, consider the problem of estimating the sample complexity of an algorithm such as for distribution testing or noisy binary search itself. Proofs in this space are often sloppy with constant factors, so the proven bound is not reflective of the true performance. If so, we would like to empirically estimate the sample complexity i at which the success

Table 1 Comparison of our result to prior algorithms for MonotonicNBS in the regime of $\varepsilon \ll \min(\tau, 1-\tau)$ and $\delta = 1/n^{o(1)}$, ignoring lower order terms. The analysis in [10] is not careful with constants, so we also include our best estimate of the actual constant after tuning constant factors in the algorithms.

Algorithm	Proven query complexity	Actual constant
Binary Search w/ Repetition	$2\frac{\tau(1-\tau)}{\varepsilon^2}\log n \cdot \log\frac{\log_2 n}{\delta}$	
[10] Multiplicative Weights	$4000\frac{\max(\tau,1-\tau)}{\varepsilon^2}\log n \cdot \log\frac{1}{\delta}$	≈ 31
[10] Backtracking	$476909\frac{\max(\tau,1-\tau)}{\varepsilon^2}\log n \cdot \log\frac{1}{\delta}$	≈ 2000
BayesianScreeningSearch	$2\frac{\tau(1-\tau)}{\varepsilon^2}\log n$	

probability p_i is above a given threshold τ (say, 90%). (In some cases we even know the worst-case distribution [6] so the empirical estimate is of the worst-case performance, not just the distributional performance.) We can run the algorithm at a given sample complexity i and check correctness, getting Success with probability p_i. The success probability is monotonic in i, and we would like to estimate the i^* where p_i crosses τ. Finding i^* exactly may be very hard – the success probability at 10000 and 10001 samples are likely to be almost identical – so we would settle for *some* index with $p_i \approx \tau$.

For a non-computer science example, calculating the LD50 for a substance (the dose needed to kill half of the members of a specific population) is a noisy binary search problem with error probability that skyrockets close to the true answer.

Such considerations led to the noisy binary search model of Karp and Kleinberg [10], which we call MonotonicNBS: we have n coins whose unknown probabilities $p_i \in [0,1]$ are sorted in nondecreasing order. We can flip coin i to see heads with probability p_i. The goal is to find any coin i with nonempty $[p_i, p_{i+1}] \cap (\tau-\varepsilon, \tau+\varepsilon)$ (See Figure 1 for a graphical representation). This model subsumes FixedNoiseNBS (where $p_i = \frac{1}{2}-\varepsilon$ for $i \le i^*$ and $\frac{1}{2}+\varepsilon$ otherwise) and of course regular binary search (where $p_i \in \{0,1\}$). Throughout this paper we will suppose that τ is a constant bounded away from $\{0,1\}$, n grows to ∞, and ε and the desired failure probability δ may be constant or may approach 0 as $n \to \infty$.

The naive solution to MonotonicNBS is binary search with repetition: we do regular binary search, but repeat each query enough times to have $\frac{\delta}{\log n}$ failure probability if $p_i \notin [\tau-\varepsilon, \tau+\varepsilon]$. This gives sample complexity $O(\frac{1}{\varepsilon^2}\log n \log\frac{\log n}{\delta})$. In [10] it was shown that this extra $\log\log n$ term is unnecessary, giving two algorithms that each have sample complexity

$$O(\frac{1}{\varepsilon^2}\log n \log\frac{1}{\delta}).$$

In this paper, we show how to improve this bound. We show upper and lower bounds that achieve the tight constant on $\log n$, and reduce the $\log\frac{1}{\delta}$ dependence from multiplicative to additive. Table 1 compares our result to existing methods for MonotonicNBS.

On Studying Constants. When analyzing sublinear algorithms, and trying to remove $\log\log n$ factors in query complexity, constant factors really matter. The proofs in [10] are not careful with constants, but the algorithms themselves inherently lose constants. Our best estimate is that one algorithm "improves" upon naive repetition by a factor of $\frac{\log\log_2 n}{31}$, and the other by $\frac{\log\log_2 n}{2000}$ (where log is the natural log). Neither is an improvement for any n that will ever be practical – the better algorithm is only an improvement for $n > 2^{e^{31}} \approx 10^{10^{11}}$. By studying constants, we are forced to design an algorithm that (as we shall see) gives improvements for practical values of n. We give further discussion of the value of studying constants in Section 2.2.

Connection to Communication. Noisy binary search is intimately connected to the asymmetric binary channel, i.e., the binary channel that can choose between sending 1 with probability $\tau - \varepsilon$ or with probability $\tau + \varepsilon$. If each $p_i \in \{\tau \pm \varepsilon\}$, then noisy binary search needs to reveal the $\log_2 n$-bit i^* through such a channel; queries below i^* are 1 with probability $\tau - \varepsilon$ and those above i^* are 1 with probability $\tau + \varepsilon$. The natural target sample complexity is therefore $\frac{1}{C_{\tau,\varepsilon}} \log_2 n$, where $C_{\tau,\varepsilon}$ is the *information capacity* of the asymmetric binary channel:

$$C_{\tau,\varepsilon} := \max_q H((1-q)(\tau-\varepsilon) + q(\tau+\varepsilon)) - (1-q)H(\tau-\varepsilon) - qH(\tau+\varepsilon) \tag{1}$$

where $H(p)$ is the binary entropy function. For $\tau = \frac{1}{2}$, the maximum is at $q = \frac{1}{2}$ and this is just $C_{\frac{1}{2},\varepsilon} = 1 - H(\frac{1}{2} - \varepsilon)$, the capacity of the binary *symmetric* channel with error probability $\frac{1}{2} - \varepsilon$. For $\tau \neq \frac{1}{2}$, the information obtained from $\tau - \varepsilon$ and $\tau + \varepsilon$ probability coins is not the same, so the capacity is achieved by getting $\tau + \varepsilon$ coins with some probability q different from 1/2; it satisfies $C_{\tau,\varepsilon} \approx \frac{\varepsilon^2}{2\tau(1-\tau)\log 2}$ for fixed τ as $\varepsilon \to 0$.

Our results. Our main result is the following:

▶ **Theorem 1** (Upper bound). *Let $0 < \tau < 1$ be a constant. Consider any parameters $0 < \varepsilon, \delta < 1/2$ with $0 < \varepsilon < \min(\tau, 1-\tau)/2$. On any* MonotonicNBS$(\tau, \varepsilon)$ *input, the algorithm* BayesianScreeningSearch *uses at most*

$$\frac{1}{C_{\tau,\varepsilon}}(\log_2 n + O(\log^{2/3} n \log^{1/3} \frac{1}{\delta} + \log \frac{1}{\delta}))$$

queries and succeeds with probability $1 - \delta$.

Unlike [2, 1, 5, 15, 9], our results apply to MonotonicNBS, not just FixedNoiseNBS, so they do not restrict the value of p_i and handle $\tau \neq \frac{1}{2}$. Unlike [10], we achieve good constant factors, high-probability results, and a better scaling with the target τ. In particular, [10] scales multiplicatively rather than additively with $O(\log \frac{1}{\delta})$; and it uses a reduction that incurs a constant-factor *loss* for targets $\tau \neq \frac{1}{2}$, while Theorem 1 scales with $\Theta(\tau(1-\tau))$ so *improves* for $\tau \neq \frac{1}{2}$.

Using Shannon's strong converse theorem, we show that the dependence on n is tight: for $\varepsilon \gg n^{-1/4}$, any algorithm must sometimes use $(1 - o(1))\frac{1}{C_{\tau,\varepsilon}} \log_2 n$ queries; in fact, it must use this many queries with nearly $1 - \delta$ probability.

▶ **Theorem 2** (Strong converse). *Any* MonotonicNBS(τ, ε) *algorithm that succeeds with $1 - \delta$ probability on inputs with all $p_i \in \{\tau \pm \varepsilon\}$ must have at least a $1 - \delta - O(\frac{1}{\gamma^2 n \varepsilon^4})$ chance of using at least*

$$(1-\gamma)\frac{1}{C_{\tau,\varepsilon}} \log_2 n$$

queries, for any $\gamma > 0$.

For $\tau = \frac{1}{2}$, this is also a lower bound for FixedNoiseNBS. Thus Theorem 2 gives a new *worst-case* lower bound for FixedNoiseNBS, which is a $\frac{1}{1-\delta}$ factor larger than the lower bound for *expected* query complexity achieved in prior work [2, 1, 5, 9].

For $\tau \neq \frac{1}{2}$, our results are the first ones connecting noisy binary search to $C_{\tau,\varepsilon}$, the information capacity of the binary asymmetric channel.

Our results: expected queries. For constant δ, one can get a better bound for the expected number of queries in a simple way: only run the algorithm with probability $1-(1-\frac{1}{\log n})\delta$, and otherwise output the wrong answer from zero queries. This saves essentially a $1-\delta$ factor in queries, which for constant δ is nontrivial:

▶ **Corollary 3** (Upper bound: expected queries)**.** *Under the same conditions as Theorem 1 and for any* MonotonicNBS(τ,ε) *input, algorithm* SillyBayesianScreeningSearch *uses*

$$\frac{1-\delta}{C_{\tau,\varepsilon}}(\log_2 n + O(\log^{2/3} n \log^{1/3} \frac{\log n}{\delta} + \log\frac{1}{\delta}))$$

queries in expectation and succeeds with probability $1-\delta$*.*

This $1-\delta$ savings is essentially the best possible. Our strong converse (Theorem 2) already implies this, if $\varepsilon \gg n^{-1/4}$; but using Fano's inequality, the optimality is true in general:

▶ **Theorem 4** (Weak converse)**.** *Any* MonotonicNBS(τ,ε) *algorithm that succeeds with* $1-\delta$ *probability on inputs with all* $p_i \in \{\tau \pm \varepsilon\}$ *must use*

$$(1-\delta)\frac{\log_2(n-2)-1}{C_{\tau,\varepsilon}}$$

queries in expectation.

Theorem 4 was essentially shown in [2], which proved the $\tau=\frac{1}{2}$ case (by giving hardness for FixedNoiseNBS).

Our results: experiments. In Section A we compare our approach to naive repetition and the [10] algorithms. We find, for $n \geq 10^3$ and $\varepsilon = .1$, that our approach outperforms naive repetition, which outperforms both [10] algorithms. For $n = 10^9$, our approach uses $2.3\times$ fewer samples than naive repetition.

1.1 Algorithm Overview

We now describe our noisy binary search algorithm in the case of $\tau=\frac{1}{2}$ and $\delta > 1/n^{o(1)}$.

Bayesian start. The natural choice for a "hard" instance is when $p_i \in \{\tau \pm \varepsilon\}$ (though we will see that having multiple right answers is also hard in a different way), so the algorithm must find the transition location i^*, and information theoretic arguments show $\frac{1}{C_{\tau,\varepsilon}}\log_2 n$ queries are necessary. To avoid losing a constant factor in sample complexity, the algorithm essentially must spend most of its time running the Bayesian algorithm. This algorithm starts with a uniform prior over which interval crosses τ, makes the maximally informative query, updates its posterior, and repeats. When $\tau=\frac{1}{2}$, the maximally informative query is the median under the posterior, and the Bayesian update is to multiply intervals on one side of the query by $1+2\varepsilon$ and the other side by $1-2\varepsilon$. This algorithm, BayesLearn, is given in Algorithm 1; the algorithm for general τ is given in Section 4.

As a technical side note, the discrete nature of the problem introduces a bit of subtlety. Note that MonotonicNBS flips coins i but returns an *interval* between coins that should be good:

▶ **Definition 5.** *We say that an interval* $[i,i+1]$ *is* (τ,ε)-good *if* $[p_i,p_{i+1}]\cap(\tau-\varepsilon,\tau+\varepsilon)$ *is nonempty.*

Precisely, our version of the Bayesian algorithm is as follows: we start with a uniform prior over intervals. The median of our posterior can be viewed as a fractional coin, and we flip the nearest actual coin but update our posterior as if we flipped the fractional coin. So, for example, suppose the median is 4.7 ($.7 * w(5) + \sum_{i=1}^{4} w(i) = .5$). We flip coin 5, and if it comes out 0, that suggests the true threshold is probably above 5. We then scale up our posterior on all intervals above 5 by $1 + 2\varepsilon$; scale down intervals below 4 by $1 - 2\varepsilon$; and scale the weight on interval $[4, 5]$ by $.3(1 + 2\varepsilon) + .7(1 - 2\varepsilon)$. This new posterior is still a distribution that sums to 1.

Algorithm 1 Bayesian learner in $\tau = \frac{1}{2}$ case. Flips M coins and returns M intervals.

Input A set of n queryable coins, update size ε, number of steps M.
Output A list of M intervals queried.

procedure BayesLearn(coins, ε, M)
  $n \leftarrow |\text{coins}|$
  $w_1 \leftarrow \text{uniform}([n-1])$ ▷ Prior distribution over intervals
  $L \leftarrow \{\}$
  for $i \in [M]$ **do**
    $j_i \leftarrow$ median interval of w_i
    $x_i \leftarrow$ either j_i or $j_i + 1$, whichever is closer to the median
    append j_i to L
    $y_i \leftarrow$ flip coin x_i ▷ 1 with probability p_{x_i}
    $$w_{i+1}(x) \leftarrow \begin{cases} w_i(x) \cdot (1 - 2\varepsilon(-1)^{y_i}) & \text{if } x < j_i \\ w_i(x) \cdot (1 + 2\varepsilon(-1)^{y_i}) & \text{if } x > j_i \\ \text{remainder so } w_{i+1} \text{ sums to 1} & \text{if } x = j_i \end{cases}$$
  return L

Using the result. After running the Bayesian algorithm for most of our query budget, we need to output an answer. The question becomes: how can we take the transcript of the Bayesian algorithm and extract a useful worst-case frequentist guarantee? We need the algorithm to work for *all* monotonic p, which can have values very different than $\tau \pm \varepsilon$.

In the prior work achieving tight constants for FixedNoiseNBS [2, 5], because the p_i are guaranteed to be $\frac{1}{2} \pm \varepsilon$, the analysis can show that the weight of the single "good" interval grows in expectation at each step. By a Hoeffding bound, after the desired number of iterations the "good" interval has more weight than every other interval combined, so it can be easily selected. But that property is not true for the more general p_i of MonotonicNBS: if many p_i are $\frac{1}{2} \pm 0.6\varepsilon$, the Bayesian algorithm will wander somewhat too slowly through these samples without growing any single interval by the desired amount.

However, in such cases the Bayesian algorithm is spending a lot of time among good intervals. This holds in general. Our key lemma shows that, if we run BayesLearn for $1 + O(\gamma)$ times the information theoretic bound $\frac{1}{C_{\tau,\varepsilon}} \log_2 n$, a γ fraction of the intervals it visits are (τ, ε)-good:

▶ **Lemma 6** (Bayesian performance). *Consider any $0 < \varepsilon, \tau, \delta, \gamma < 1$ with $\gamma \leq \frac{1}{7}$, $\varepsilon < \min(\tau, 1 - \tau)/2$, and let L be the list of intervals returned by BayesLearn, when run for*

$$\frac{1 + O(\gamma)}{C_{\tau,\varepsilon}} \cdot \left(\log_2 n + O(\sqrt{\log n \log \frac{1}{\delta}} + \log \frac{1}{\delta}) \right)$$

iterations on an MonotonicNBS instance. With probability $1 - \delta$, at least a γ fraction of the intervals in L are (τ, ε)-good.

2 Proofs of Statements

By considering the γ-quantiles of the returned list, we reduce n to $\frac{1}{\gamma}$. We can now run a less efficient noisy binary search algorithm on this small subproblem. There are some complications, as the solution to the new noisy binary search could correspond to a larger interval than two adjacent coins. To deal with this, we run BayesLearn with $\varepsilon' = (1-o(1))\varepsilon$, which lets us test our candidate answers.

Technical comparison of techniques. How we leverage the bayesian learner is the main technical difference between our upper bound and that of prior work [10, 2, 5]. As described above, the situation is rather simpler for FixedNoiseNBS. For MonotonicNBS, [10] instead used conservative updates in their multiplicative weights algorithm: rather than the true Bayesian update $1 \pm 2\varepsilon$, it multiplies by about $1 \pm \frac{3}{5}\varepsilon$. This necessarily loses a constant factor, but ensures that either the median interval queried or the last interval queried is good. This property is not true for the true Bayesian algorithm with sharp constant.

2.1 Related Work

The FixedNoiseNBS version of noisy binary search, where $\tau = \frac{1}{2}$ and $p_i \in \{\frac{1}{2} \pm \varepsilon\}$, was posed by Burnashev and Zigangirov [2], who showed how to achieve

$$\frac{1}{C_{\frac{1}{2},\varepsilon}}\left(\log_2 n + \log_2 \frac{1}{\delta} + \log_2 \frac{1+2\varepsilon}{1-2\varepsilon}\right)$$

expected queries (in Russian; see [15] for an English proof). Essentially the same [2] algorithm for FixedNoiseNBS was rediscovered in [1]. Some bugs with the [1] proof were discovered and fixed in [5], as well as an analysis of a variant of the algorithm for *worst-case* sample complexity

$$\frac{1}{C_{\frac{1}{2},\varepsilon}}\left(\log_2 n + O(\sqrt{\log n \log \frac{1}{\delta}}) + O(\log \frac{1}{\delta})\right).$$

For $1 \ll \log \frac{1}{\delta} \ll \log n$, Gu and Xu [9] showed black-box improvements for other δ. If δ is constant, they output $\perp$ with probability $\delta - \frac{1}{\log n}$, and otherwise run the [5] algorithm with $\delta' = \frac{1}{\log n}$. On the other hand, for $\delta = n^{-\Omega(1)}$, repeatedly running [5] with $\delta' = \frac{1}{\log n}$ and checking the result gives improvements:

$$(1+o(1))\left(\frac{1-\delta}{C_{\frac{1}{2},\varepsilon}}\log_2 n + \frac{\log \frac{1}{\delta}}{\varepsilon \log \frac{1+2\varepsilon}{1-2\varepsilon}}\right)$$

For $\varepsilon \ll 1$, this is a factor 2 improvement on the constant factor on $\log \frac{1}{\delta}$. Moreover, [9] shows that this bound is sharp in both n and δ.

Our version of noisy binary search, MonotonicNBS, was first posed by Karp and Kleinberg [10]. They gave two algorithms, based on recursive backtracking and multiplicative weights respectively, that take $O(\frac{1}{\varepsilon^2}\log n)$ queries for constant δ, which they showed is within constant factors of optimal for constant τ, δ. Unfortunately, the constant factors make both algorithms worse than the naive repetition algorithm for any reasonable n (see Table 1 and Section A).

Other models. There are many different variations for noisy binary search (see [12] for a survey of older work on the subject). Emamjomeh-Zadeh, Kempe, and Singhal [7] solve an extension of FixedNoiseNBS from the line to graphs. This result was improved and simplied by Dereniowski, Tiegel, Uznański and Wolleb-Graf [4], which was in later improved and simplified by Dereniowski, Łukasiewicz, and Uznański [5]. Nowak developed a different generalization of FixedNoiseNBS to general hypothesis classes [11]. Waeber, Frazier, and Henderson [14] investigates a continuous variant of FixedNoiseNBS, where the target is a point in the real interval $[0, 1]$, and show that the Bayesian algorithm converges geometrically (the ideal convergence up to constant factors).

To our knowledge, [10] is the only previous work that handles a setting like MonotonicNBS where the "true" coin may be indistinguishable from nearby coins, and the goal is just to find a sufficiently good answer.

Applications. Noisy binary search is also used as a subroutine in other algorithms. For instance in [13] it is used for group testing, and in Crume [3] as a replacement for git-bisect under unreliable tests. Both implementations were based on the multiplicative weights algorithm of Karp and Kleinberg [10].

2.2 Why constants?

There is a tendency in theoretical computer science to regard constant factors as unimportant. But theorists care about constants in many situations, such as approximation ratios or rates of codes, and we believe that the query complexity of sublinear algorithms is another situation where they should be considered.

In general, the arguments for ignoring constants in time complexity hold with much less force for query complexity. The constant for time complexity is highly dependent on the machine architecture, which changes over time (e.g., the relative cost of addition and multiplication). Moreover, these hardware improvements mitigate the cost of poor constants. But the number of queries is a mathematical value, and the cost of queries (which may be, e.g., blood tests or running a giant test suite) does not clearly decrease with time.

The question should be: does theoretical study of constant factors lead to algorithmic insights necessary for more practical algorithms? Our paper shows that it does. By considering constants, we are forced to find a more efficient way of translating the Bayesian algorithm into one with frequentist guarantees (via Lemma 6). The constants lost in the previous attempt at this (in [10]) mean that it is worse than the naive method until $n > 10^{10^{11}}$.

It should not be surprising that a simple method that loses an $O(\log \log n)$ factor can beat an algorithm that loses "only" constants, for all practical values of n. The study of leading constants is a lens by which we found a new algorithm that actually outperforms the naive method for reasonable values of n (namely $n > 1000$).

3 Detailed Proof Sketch for Upper Bound

3.1 Key Lemma on Bayesian Learner

For this proof overview, we focus on the case of $\delta > n^{-o(1)}$ and target $\tau = \frac{1}{2}$, where BayesLearn queries the median of the posterior at each stage, and

$$C_{\tau,\varepsilon} = 1 - H(\frac{1}{2} + \varepsilon) = (\frac{1}{2} + \varepsilon)\log_2(1 + 2\varepsilon) + (\frac{1}{2} - \varepsilon)\log_2(1 - 2\varepsilon) \approx \frac{2\varepsilon^2}{\log 2}.$$

We give an overview of the proof of our key lemma in this case:

▶ **Lemma 6** (Bayesian performance). *Consider any $0 < \varepsilon, \tau, \delta, \gamma < 1$ with $\gamma \le \frac{1}{7}$, $\varepsilon < \min(\tau, 1-\tau)/2$, and let L be the list of intervals returned by* BayesLearn*, when run for*

$$\frac{1+O(\gamma)}{C_{\tau,\varepsilon}} \cdot \left(\log_2 n + O(\sqrt{\log n \log \frac{1}{\delta}} + \log \frac{1}{\delta}) \right)$$

iterations on an MonotonicNBS *instance. With probability $1-\delta$, at least a γ fraction of the intervals in L are (τ,ε)-good.*

Let a be the "best answer", an interval that straddles the bias $\frac{1}{2}$. The algorithm keeps track of a distribution w on $[n-1]$; at each step i, it queries the median of the current distribution w_i, then multiplies the density on one side by $1+2\varepsilon$ and the other by $1-2\varepsilon$ to form w_{i+1}. We analyze the algorithm by looking at $\log_2 w(a)$.

At each step, the interval j we choose is either good (a valid answer) or bad (invalid). If it is *bad*, suppose the sampled coin x has probability $p_x \ge \frac{1}{2} + \varepsilon$. Then x is above a, so $w(a)$ multiplies by $1+2\varepsilon$ with probability p_x, and $1-2\varepsilon$ with probability $1-p_x$. Hence:

$$\mathbb{E}[\log_2 w_{i+1}(a) - \log_2 w_i(a)] = p_x \log_2(1+2\varepsilon) + (1-p_x)\log_2(1-2\varepsilon) \ge C_{\tau,\varepsilon}.$$

The case of $p_x \le \frac{1}{2} - \varepsilon$ is symmetric, giving the same bound. So every bad interval we select increases $\log_2 w(a)$ by $C_{\tau,\varepsilon}$ in expectation.

On the other hand, if the interval we select is *good*, $\log_2 w(a)$ may decrease in expectation. For example, if we query coin a and $\sum_{i=1}^{a-1} w(i) = \frac{1}{2}$, we could have

$$\mathbb{E}[\log_2 w_{i+1}(a) - \log_2 w_i(a)] = \frac{1}{2}\log_2(1-2\varepsilon) + \frac{1}{2}\log_2(1+2\varepsilon) \approx -\frac{2\varepsilon^2}{\log 2} \approx -C_{\tau,\varepsilon}$$

It turns out this is essentially the worst case, and in general the expected decrease in $\log_2 w(a)$ is no more than $5C_{\tau,\varepsilon}$ for any $\varepsilon < \frac{1}{2}\min(\tau, 1-\tau)$. As a result, the potential function

$$\log_2 w_i(a) - \gamma C_{\tau,\varepsilon} \cdot (\#\text{ intervals chosen}) + 6C_{\tau,\varepsilon} \cdot (\#\text{ good intervals chosen})$$

increases by at least $(1-\gamma)C_{\tau,\varepsilon}$ in expectation in each step i, regardless of where the median is in that step. This potential function starts at $-\log_2(n-1)$, so after $M = (1+2\gamma)\frac{1}{C_{\tau,\varepsilon}}\log_2 n$ steps it is at least $\Theta(\gamma)\log_2 n$ in expectation. An Azuma-Hoeffding bound shows that the value concentrates about this expectation, and in particular will be positive with $1-\delta$ probability. If so, since $\log_2 w_i(a) \le 0$ always, we have

$$6 \cdot (\#\text{ good intervals chosen}) - \gamma(\#\text{ intervals chosen}) \ge 0,$$

and hence a $\frac{\gamma}{6}$ fraction of chosen intervals are good.

This proves the key lemma: after $(1+O(\gamma))\frac{1}{C_{\tau,\varepsilon}}\log n$ steps of BayesLearn, a γ fraction of coins flipped are good with decent probability.

Targets $\tau \ne \frac{1}{2}$. When $\tau \ne \frac{1}{2}$, the maximum-information query is no longer the median coin, but a slightly different quantile $\frac{1}{2} \pm O(\frac{\varepsilon}{\tau(1-\tau)})$, and the Bayesian updates use more complicated factors. This choice is still capacity-achieving on bad intervals, i.e., the expected "information gain" is $\mathbb{E}[\log_2 w_i(a) - \log_2 w_{i+1}(a)] \ge C_{\tau,\varepsilon}$, and on good intervals the expected information loss is still at most $5C_{\tau,\varepsilon}$, so the proof structure works unchanged.

3.2 Rest of Upper Bound

Recall that in this overview we assume $\log \frac{1}{\delta} \ll \log n$. By Lemma 6, if we take all $\{\gamma, 2\gamma, \dots, \lfloor \frac{1}{\gamma} \rfloor \gamma\}$ quantiles of the list returned by BayesLearn, run with parameter $\varepsilon' = \varepsilon(1-\alpha)$ (where α is introduced so we can later test the bias of each coin), we get a size-$\frac{1}{\gamma}$ list containing at least one ε'-good interval. This ε' has $C_{\tau,\varepsilon'} = (1 - O(\alpha))C_{\tau,\varepsilon}$. For any γ, we can just flip all of these coins $O(\frac{1}{\alpha^2\varepsilon^2} \log \frac{1}{\gamma\delta})$ times to find an ε-good one. This would give sample complexity

$$\underbrace{(1+O(\gamma))(1+O(\alpha))\frac{1}{C_{\tau,\varepsilon}}\left(\log_2 n + O(\sqrt{\log n \log \frac{1}{\delta}})\right)}_{\text{BayesLearn, Lemma 6}} + \underbrace{O(\frac{1}{\gamma} \cdot \frac{1}{\alpha^2\varepsilon^2} \log \frac{1}{\gamma\delta})}_{\text{Testing quantiles}} \tag{2}$$

which, by setting γ and α to $(\frac{\log \frac{1}{\delta}}{\log n})^{1/4}$, gives sample complexity

$$(1 + O(\frac{\log \frac{1}{\delta}}{\log n})^{1/4})\frac{1}{C_{\tau,\varepsilon}} \log_2 n.$$

This is the desired sharp bound, within $(1 + o(1))$ of optimal. One can do slightly better: the second stage is itself a noisy binary search question on $O(1/\gamma)$ coins, so by applying the algorithm recursively with $\gamma' = O(1)$ we can solve it on the size-$O(1/\gamma)$ list in $O(\frac{1}{(1-\alpha)C_{\tau,\varepsilon}} \log \frac{1}{\gamma\delta} + \frac{1}{\alpha^2\varepsilon^2} \log \frac{1}{\gamma\delta})$ queries. As we recurse on a much smaller list, the samples used are all lower order and we do not need to recurse more than once. However, the answer to the recursive call might not be a valid answer to the original problem. Regardless, one of the endpoints of the return call must be a valid answer, which we can test for. By optimizing the parameters, this improves the sample complexity to

$$(1 + O(\frac{\log \frac{1}{\delta}}{\log n})^{1/3})\frac{1}{C_{\tau,\varepsilon}} \log_2 n,$$

giving Theorem 1.

4 Proof of Lemma 6

4.1 Definitions

Let $\{l, \dots, r\}$ be the set of good intervals. Let a be the maximum $i \in [n-1]$ such that $p_i \le \tau$. We also define the following functions:

$$C_{\tau,\varepsilon} = \max_q H((1-q)(\tau-\varepsilon) + q(\tau+\varepsilon)) - (1-q)H(\tau-\varepsilon) - qH(\tau+\varepsilon) \tag{3}$$

$$W(x) = \sum_{i \in [x]} w(i) \tag{4}$$

$$\Phi(w, L) = \log_2 w(a) + 6C_{\tau,\varepsilon}(|\{x \in L | x \in [l, r]\}| - \gamma|L|) \tag{5}$$

$$q = \arg\max_x H((1-x)(\tau-\varepsilon) + x(\tau+\varepsilon)) - (1-x)H(\tau-\varepsilon) - xH(\tau+\varepsilon) \tag{6}$$

$C_{\tau,\varepsilon}$ is the capacity of a (τ, ε)-BAC. We let q satisfy the equation which expresses the shared information between a sent and received message through a (τ, ε)-BAC. (See 12, 13 for explicit formulas for $C_{\tau,\varepsilon}, q$) If our prior were true – so the coins really were $\tau \pm \varepsilon$ – we would like to flip a $\tau + \varepsilon$ coin with probability q. This is achieved by selecting the q-quantile of our posterior, which is above the true threshold with probability q. If $\tau = \frac{1}{2}$, $q = \frac{1}{2}$ and we query the median; in general, we query the $q = \frac{1}{2} \pm O(\frac{\varepsilon}{\tau(1-\tau)})$ quantile.

Φ is a potential function that we will be analyzing. We also define:

$$d_{0,0} = \frac{1-\tau-\varepsilon}{1-\tau-(2q-1)\varepsilon}, d_{0,1} = \frac{1-\tau+\varepsilon}{1-\tau-(2q-1)\varepsilon}, d_{1,0} = \frac{\tau+\varepsilon}{\tau+(2q-1)\varepsilon}, d_{1,1} = \frac{\tau-\varepsilon}{\tau+(2q-1)\varepsilon} \tag{7}$$

for brevity. In terms of BayesLearn we can think of $d_{x,y}$ as "the multiplicative effect of a flip resulting in x ($1 =$ Heads, $0 =$ Tails) on the density of an interval on side y ($1 =$ Right, $0 =$ Left) of the flipped coin." When $\tau = \frac{1}{2}$, $d_{x,y} = 1 - 2\varepsilon(-1)^{x \oplus y}$.

Algorithm 2 Acts as a Bayesian learner for M iterations, returns a list of all the chosen intervals. Expressions for the $d_{x,y}$ values are given in (7).

procedure getIntervalFromQuantile(w, q)
  $i \leftarrow \min i \in [n]$ s.t. $W(i) \geq q$
procedure roundIntervalToCoin(i, w, q)
  return i **if** $\frac{q-W(i-1)}{w(i)} \leq q$ **else** $i+1$
procedure BayesLearn($\{c_i\}_{i=1}^n, n, \tau, \varepsilon, M$)
  $w_1 \leftarrow$ uniform($[n-1]$)
  Define q as in (13) ▷ The quantile we choose
  $L \leftarrow \{\}$
  for $i \in [M]$ **do**
    $j_i \leftarrow$ getIntervalFromQuantile(w_i, q) ▷ The chosen interval
    $x_i \leftarrow$ roundIntervalToCoin(j_i, w_i, q) ▷ The index of the coin we are going to flip
    append j_i to L
    $y_i \leftarrow$ flip(c_{x_i})
    $$w_{i+1} \leftarrow \begin{cases} w_i(x) d_{y_i,0} & \text{if } x \in \{1, \ldots, j_i - 1\} \\ d_{y_i,0}(q - W_i(j_i - 1)) + d_{y_i,1}(W_i(j_i) - q) & \text{if } x = j_i \\ w(x) d_{y_i,1} & \text{if } x \in \{j_i + 1, \ldots, n-1\} \end{cases}$$
  return L

▶ **Lemma 7.** *In BayesLearn,* $\mathbb{E}[\Phi_{t+1} - \Phi_t | y_t, y_{t-1}, \ldots, y_1] \geq (1 - O(\gamma)) C_{\tau,\varepsilon}$.

Proof. Φ is given by the sum of equations (8) and (9).

$$6C_{\tau,\varepsilon}(|\{j \in L | j \in [l, r]\}| - \gamma |L|) \tag{8}$$

$$\log_2 w(a) \tag{9}$$

Recall that in the tth round, j_t is the interval chosen, and x_t is index of the coin flipped. Let p be the probability c_{x_t} lands heads.

Bad queries. Suppose $j_t \notin [l, r]$. If $j_t > r$, then $p \geq \tau + \varepsilon$ and the expected change in (9) is

$$p \log_2 d_{1,0} + (1-p) \log_2 d_{0,0}$$

The first log is positive and the second log is negative, so this expression is minimized at $p = \tau + \varepsilon$, at which point some computation (Lemma 9) shows that it equals $C_{\tau,\varepsilon}$. Similarly, if $x_t < l$ then $p \leq \tau - \varepsilon$ and the expected change is

$$p \log_2 d_{1,1} + (1-p) \log_2 d_{0,1}$$

which is also at least $C_{\tau,\varepsilon}$ by Lemma 9. As $j_t \notin [l, r]$, the change in (8) is $-\gamma \cdot 6C_{\tau,\varepsilon}$. Therefore in this case the expected change in Φ is at least $(1 - 6\gamma)C_{\tau,\varepsilon}$.

Good queries. Suppose $j_t \in [l, r]$. The change in (8) is now $6C_{\tau,\varepsilon}(1-\gamma)$. But how much can (9) decrease in expectation? Suppose that $j_t \neq a$. Then the expected change is either

$$p \log_2 d_{1,0} + (1-p) \log_2 d_{0,0}$$

with $p \geq \tau$, or

$$p \log_2 d_{1,1} + (1-p) \log_2 d_{0,1}$$

with $p \leq \tau$.

As $d_{1,0} = \frac{\tau+\varepsilon}{\tau+(2q-1)\varepsilon} \geq \frac{1-\tau-\varepsilon}{1-\tau-(2q-1)\varepsilon} = d_{0,0}$ and $d_{1,1} = \frac{\tau-\varepsilon}{\tau+(2q-1)\varepsilon} \leq \frac{1-\tau+\varepsilon}{1-\tau-(2q-1)\varepsilon} = d_{0,1}$, both of these expressions are minimized when $p = \tau$. So the expected change in (9) is lower bounded by:

$$\min\left(\tau \log_2 d_{1,0} + (1-\tau)\log_2 d_{0,0}, \tau \log_2 d_{1,1} + (1-\tau)\log_2 d_{0,1}\right). \tag{10}$$

We note that

$$\begin{aligned}
\tau \log_2 d_{1,0} + (1-\tau)\log_2 d_{0,0} &= (\tau+\varepsilon)\log_2 d_{1,0} + (1-\tau-\varepsilon)\log_2 d_{0,0} - \varepsilon \log_2 d_{1,0} + \varepsilon \log_2 d_{0,0} \\
&= C_{\tau,\varepsilon} - \varepsilon \log_2 d_{1,0} + \varepsilon \log_2 d_{0,0} \\
&\geq C_{\tau,\varepsilon} - 3\varepsilon\left(\frac{\varepsilon}{\tau} + \frac{\varepsilon}{1-\tau}\right) \qquad \text{(Lemma 13)} \\
&= C_{\tau,\varepsilon} - \frac{3\varepsilon^2}{\tau(1-\tau)} \\
&\geq C_{\tau,\varepsilon} - (6\log 2) C_{\tau,\varepsilon} \qquad \text{(Lemma 10)} \\
&\geq -5C_{\tau,\varepsilon}
\end{aligned}$$

a symmetric argument for lower bounding $\tau \log_2 d_{1,1} + (1-\tau)\log_2 d_{0,1}$ holds. Therefore, the change in (9) is lower bounded by $-5C_{\tau,\varepsilon}$.

Now suppose that $j_t = a$. Then for some $k \in [0,1]$ the expected change in (9) is:

$$p \log_2(d_{1,0}k + d_{1,1}(1-k)) + (1-p)\log_2(d_{0,0}k + d_{0,1}(1-k))$$

If $k \leq q$ then we flip a so $p \leq \tau$. $d_{0,0}k + d_{0,1}(1-k) \geq d_{0,0}q + d_{0,1}(1-q) = 1$. Also $d_{1,0}k + d_{1,1}(1-k) \leq d_{1,0}q + d_{1,1}(1-q) = 1$. Therefore, this expression is minimized when $p = \tau$. By symmetry, when $k > q$ this expression is also minimized when $p = \tau$.

So the expected change in (9) is lower bounded by

$$\tau \log_2(d_{1,0}k + d_{1,1}(1-k)) + (1-\tau)\log_2(d_{0,0}k + d_{0,1}(1-k))$$

for some $k \in [0,1]$. Taking the derivative with respect to k, we get

$$\tau \frac{d_{1,0} - d_{1,1}}{d_{1,1} + (d_{1,0} - d_{1,1})k} + (1-\tau)\frac{d_{0,0} - d_{0,1}}{d_{0,1} + (d_{0,0} - d_{0,1})k}$$

As $d_{1,1} < d_{1,0}$ and $d_{0,1} > d_{0,0}$, $\tau \frac{d_{1,0}-d_{1,1}}{d_{1,1}+(d_{1,0}-d_{1,1})k} > 0 > (1-\tau)\frac{d_{0,0}-d_{0,1}}{d_{0,1}+(d_{0,0}-d_{0,1})k}$. We note that as k increases, $\tau \frac{d_{1,0}-d_{1,1}}{d_{1,1}+(d_{1,0}-d_{1,1})k}$ decreases in magnitude, while $(1-\tau)\frac{d_{0,0}-d_{0,1}}{d_{0,1}+(d_{0,0}-d_{0,1})k}$ increases in magnitude. Therefore, the minimum value of the above expression is achieved when $k = 0$ or $k = 1$.

So the expected change in (9) is lower bounded by

$$\min(\tau \log_2 d_{1,0} + (1-\tau)\log_2 d_{0,0}, \tau \log_2 d_{1,1} + (1-\tau)\log_2 d_{0,1})$$

which is the same expression which we lower bounded for the $j_t \neq a$ case. Combining these two cases, when we are querying a good interval, the expected change is lower bounded by $6C_{\tau,\varepsilon}(1-\gamma) - 5C_{\tau,\varepsilon} = (1-6\gamma)C_{\tau,\varepsilon}$. Therefore $\mathbb{E}[\Phi_{t+1} - \Phi_t | y_t, y_{t-1}, \ldots, y_1] \geq (1 - O(\gamma))C_{\tau,\varepsilon}$. ◀

Now that we have the gain in our potential function, we can apply a stochastic domination argument with Freedman's inequality in order to get Lemma 6 (Appendix C).

5 Algorithm and Analysis

Algorithm 3 Noisy Binary Search. It recurses at most once.

Input n queryable coins $\{c_i\}_{i=1}^n$, update size ε, target τ, failure probability δ
Output An interval which is (τ, ε)-good

procedure ReductionToGamma($\{c_i\}_{i=1}^n, n, \tau, \varepsilon, \delta, \gamma$)
  $L \leftarrow$ BayesLearn($\{c_i\}_{i=1}^n, n, \tau, \varepsilon, \frac{1+O(\gamma)}{C_{\tau,\varepsilon}}(\log_2 n + O(\sqrt{\log n \log \frac{1}{\delta}} + \log \frac{1}{\delta}))$)
  $R \leftarrow \{\}$
  for $i \in [\lfloor \frac{|L|}{\lceil \gamma |L| \rceil} \rfloor]$ **do**
    append $L_{\lceil \gamma |L| \rceil i}$ to R
  return Sorted(removeDuplicates(R))
procedure BayesianScreeningSearch($\{c_i\}_{i=1}^n, n, \tau, \varepsilon, \delta, \gamma = \frac{1}{7\log_2(n)}$)
  $\varepsilon' = \varepsilon \cdot \max(1 - \sqrt[3]{\log_n(1/\delta)}, \frac{2}{3})$
  $R \leftarrow$ ReductionToGamma($\{c_i\}_{i=1}^n, n, \tau, \varepsilon', \delta/3, \frac{1}{3\log_2(n)}$)
  if $|R| > 7$ **then**
    $R \leftarrow [1] + R + [n]$ ▷ Pad R with the extremes of the initial problem.
    $i \leftarrow$ BayesianScreeningSearch($\{c_{R_i}\}_{i=1}^{|R|}, |R|, \tau, \varepsilon', \delta/3, \frac{1}{7}$)
    $\hat{p}_{R_i+1} \leftarrow$ estimate p_{R_i+1} up to $\pm \frac{\varepsilon - \varepsilon'}{2}$ error with $\delta/3$ f.p.
    if $\hat{p}_{R_i+1} > \tau - \varepsilon + \frac{\varepsilon - \varepsilon'}{2}$ **then**
      return R_i
    else
      return $R_{i+1} - 1$
  else
    for $x \in R$ **do**
      $\hat{p}_{x+1} \leftarrow$ estimate p_{x+1} up to $\pm \frac{\varepsilon - \varepsilon'}{2}$ error with $\delta/18$ f.p.
      if $\hat{p}_{x+1} > \tau - \varepsilon + \frac{\varepsilon - \varepsilon'}{2}$ **then**
        return x

▶ **Theorem 1** (Upper bound). *Let $0 < \tau < 1$ be a constant. Consider any parameters $0 < \varepsilon, \delta < 1/2$ with $0 < \varepsilon < \min(\tau, 1-\tau)/2$. On any MonotonicNBS$(\tau, \varepsilon)$ input, the algorithm BayesianScreeningSearch uses at most*

$$\frac{1}{C_{\tau,\varepsilon}}(\log_2 n + O(\log^{2/3} n \log^{1/3} \frac{1}{\delta} + \log \frac{1}{\delta}))$$

queries and succeeds with probability $1 - \delta$.

Proof.

Correctness. Suppose that we run BayesianScreeningSearch on a MonotonicNBS instance with parameters $\{c_i\}_{i=1}^n, n, \tau, \varepsilon, \delta$. Also assume that all probabilistic stages succeed, meaning that ReductionToGamma, BayesianScreeningSearch, and our coin bias estimation all succeed. By a union bound, this occurs with probability $\geq 1 - \delta$.

As we pick every $\gamma|L|$th coin from L and L contains at least $\lceil \gamma |L| \rceil$ ε'-good intervals, R contains at least one ε'-good interval. Suppose that $|R| \leq 7$ and that R_i is the first ε'-good interval in R.

Then for all $j \in \{1, \dots, i-1\}$, either R_j is an ε-good interval or it is not. If it is, then we have nothing to worry about outputting it. If it is not, then $p_{R_j+1} \leq \tau - \varepsilon$ (as if $p_{R_j+1} \geq \tau + \varepsilon$ then R_i is not ε-good), so $\hat{p}_{R_j+1} \leq \tau - \varepsilon + \frac{\varepsilon - \varepsilon'}{2}$. So we do not output any not ε-good interval before R_i. Once we reach R_i, $p_{R_i+1} > \tau - \varepsilon'$, so $\hat{p}_{R_i+1} > \tau - \varepsilon' - \frac{\varepsilon - \varepsilon'}{2} = \tau - \varepsilon + \frac{\varepsilon - \varepsilon'}{2}$ and we output R_i.

Now suppose that $|R| > 7$. As we recursively run BayesianScreeningSearch with $\gamma = 1/7$, we note that for the R in the recursive call R', $|R'| = \lfloor \frac{|L|}{\lceil \gamma |L| \rceil} \rfloor \leq \lfloor \frac{1}{\gamma} \rfloor = 7$, so $|R'| \leq 7$. By our work above, this means that the recursive call returns i such that $[p_{R_i}, p_{R_{i+1}}] \cap (\tau - \varepsilon', \tau + \varepsilon') \neq \emptyset$.

Either R_i or $R_{i+1} - 1$ is ε'-good, as if $p_{R_i+1} \leq \tau - \varepsilon'$ and $p_{R_{i+1}-1} \geq \tau + \varepsilon'$ then R must not contain any good intervals. The same logic as for the $|R| \leq 7$ case holds, and we have shown correctness.

Number of samples. Suppose that we run BayesianScreeningSearch with $\gamma = 1/7$. The ReductionToGamma call takes

$$\frac{1 + O(\gamma)}{C_{\tau,\varepsilon'}} \left(\log_2 n + O(\sqrt{\log n \log \frac{1}{\delta}} + \log \frac{1}{\delta}) \right) = \frac{1}{C_{\tau,\varepsilon}} O(\log n + \log \frac{1}{\delta})$$

samples. As we have $\gamma = 1/7$, $|R| \leq 7$ and we go through the second branch. Then the bias estimation takes $O(\frac{\tau(1-\tau)\log \frac{1}{\delta}}{(\varepsilon - (1 - \sqrt[3]{\log_n \frac{1}{\delta}})\varepsilon)^2}) = O(\frac{\tau(1-\tau)\log \frac{1}{\delta}}{(\varepsilon \sqrt[3]{\log_n \frac{1}{\delta}})^2}) = O(\frac{\log^{2/3} n \log^{1/3} \frac{1}{\delta}}{C_{\tau,\varepsilon}})$ samples, for overall $\frac{1}{C_{\tau,\varepsilon}} O(\log n + \log \frac{1}{\delta})$ samples.

Now consider the case $\gamma = \frac{1}{7 \log_2 n}$, and suppose that $1 - \sqrt[3]{\log_n(1/\delta)} \geq 2/3$. When $\gamma = O(1/\log(n)), \varepsilon' = \varepsilon * (1 - \sqrt[3]{\log_n(1/\delta)})$, we have $\frac{1+O(\gamma)}{C_{\tau,\varepsilon'}} = \frac{1+O(\frac{1}{\log n})}{C_{\tau,\varepsilon'}} = \frac{1}{C_{\tau,\varepsilon'}} = \frac{1}{(1 - O(\sqrt[3]{\log_n(1/\delta)}) C_{\tau,\varepsilon}}$, by Lemma 12. So ReductionToGamma takes

$$\frac{1}{(1 - O(\sqrt[3]{\log_n(1/\delta)}) C_{\tau,\varepsilon}} \cdot \left(\log_2 n + O(\sqrt{\log n \log \frac{1}{\delta}} + \log \frac{1}{\delta} \right)$$
$$= \frac{1}{C_{\tau,\varepsilon}} \cdot \left(\log_2 n + O(\log^{2/3} n \log^{1/3} \frac{1}{\delta} + \log \frac{1}{\delta}) \right)$$

samples.

If $|R| \leq 7$ we take the second branch and take $O(\frac{\log^{2/3} n \log^{1/3} \frac{1}{\delta}}{C_{\tau,\varepsilon}})$ more samples, which meets our bound. If $|R| > 7$ we take the first branch and recurse with $\gamma = 1/7$ and $n' = O(\log n)$, for $\frac{1}{C_{\tau,\varepsilon}} O(\log \log n + \log \frac{1}{\delta})$ samples.

As established previously, the bias estimation takes $O(\frac{\log^{2/3} n \log^{1/3} \frac{1}{\delta}}{C_{\tau,\varepsilon}})$ samples. Overall the algorithm takes

$$\frac{1}{C_{\tau,\varepsilon}} (\log_2 n + O(\log^{2/3} n \log^{1/3} \frac{1}{\delta} + \log \frac{1}{\delta}))$$

samples. In the case $1 - \sqrt[3]{\log_n(1/\delta)} < 2/3$, the $\frac{O(\log \frac{1}{\delta})}{C_{\tau,\varepsilon}}$ term dominates the rest, and the bound holds. ◀

Corollary 3 then follows by elementary analysis.

6 Lower Bounds

We achieve our lower bounds by a communication complexity reduction.

▶ **Lemma 8.** *Given any algorithm $\mathcal{A}$ which solves NBS for parameters (τ, ε) with sample budget m and failure probability δ, there exists a protocol that communicates over a discrete memoryless channel with capacity $C_{\tau,\varepsilon}$ with rate $R = \frac{\log_2(n-1)}{m}$ with failure probability δ.*

Now we can use lower bounds from information theory. By applying Shannon's Strong Converse Theorem, we get Theorem 2, and by applying Fano's inequality we get Theorem 4. See Appendix C for proofs.

7 Future Work

One interesting topic of research is *instance-dependent* noisy binary search. If an instance is much nicer than the worst case, say every coin has bias $\frac{1}{2} \pm \alpha$ for $\alpha \gg \varepsilon$, we would hope to get a $O(\frac{\log n}{\alpha^2})$ dependence, which BayesianScreeningSearch does not get. One could use an adaptive coin bias estimator to get some adaptivity, but the constants gotten from this will likely not be good.

Another open problem is attenuating the lower order terms in the upper bound for NBS. For realistic n, lower order terms such as $\sqrt{\log n}$, or even $\log \log n$ are not negligible compared to $\log n$, and influences the practical application of BayesianScreeningSearch, as seen in the experimental results where we spent 28% of our samples on the "lower order" recursive calls.

One conjectural algorithm for noisy binary search would be: run BayesLearn for $(1 + O(\gamma))OPT$ steps, then output the median of the last γOPT intervals chosen. This interpolates between the overall median (which loses a constant factor) and the final interval (which has a large probability of failure), and avoids the inefficiency of recursive calls.

References

1. Michael Ben-Or and Avinatan Hassidim. The bayesian learner is optimal for noisy binary search (and pretty good for quantum as well). In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 221–230, 2008. doi:10.1109/FOCS.2008.58.
2. Marat Valievich Burnashev and Kamil'Shamil'evich Zigangirov. An interval estimation problem for controlled observations. *Problemy Peredachi Informatsii*, 10(3):51–61, 1974.
3. Adam Crume. Robust binary search. https://github.com/adamcrume/robust-binary-search, 2020.
4. Dariusz Dereniowski, Daniel Graf, Stefan Tiegel, and Przemyslaw Uznanski. A framework for searching in graphs in the presence of errors. *CoRR*, abs/1804.02075, 2018. arXiv:1804.02075.
5. Dariusz Dereniowski, Aleksander Lukasiewicz, and Przemyslaw Uznanski. Noisy searching: simple, fast and correct. *CoRR*, abs/2107.05753, 2021. arXiv:2107.05753.
6. Ilias Diakonikolas, Themis Gouleakis, John Peebles, and Eric Price. Sample-optimal identity testing with high probability. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
7. Ehsan Emamjomeh-Zadeh, David Kempe, and Vikrant Singhal. Deterministic and probabilistic binary search in graphs. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 519–532, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2897518.2897656.
8. Robert G Gallager. *Information theory and reliable communication*, volume 2. Springer, 1968.
9. Yuzhou Gu and Yinzhan Xu. Optimal bounds for noisy sorting. *STOC*, 2023. doi:10.48550/arXiv.2302.12440.

10 Richard M. Karp and Robert Kleinberg. Noisy binary search and its applications. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 881–890, USA, 2007. Society for Industrial and Applied Mathematics.

11 Robert Nowak. Noisy generalized binary search. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009. URL: https://proceedings.neurips.cc/paper/2009/file/556f391937dfd4398cbac35e050a2177-Paper.pdf.

12 Andrzej Pelc. Searching games with errors—fifty years of coping with liars. *Theoretical Computer Science*, 270(1):71–109, 2002. doi:10.1016/S0304-3975(01)00303-6.

13 Bernard Teo and Jonathan Scarlett. Noisy adaptive group testing via noisy binary search. *IEEE Trans. Inf. Theory*, 68(5):3340–3353, 2022. doi:10.1109/TIT.2022.3140604.

14 Rolf Waeber, Peter I. Frazier, and Shane G. Henderson. Bisection search with noisy responses. *SIAM Journal on Control and Optimization*, 51(3):2261–2279, 2013. doi:10.1137/120861898.

15 Ziao Wang, Nadim Ghaddar, Banghua Zhu, and Lele Wang. Noisy sorting capacity. In *2022 IEEE International Symposium on Information Theory (ISIT)*, pages 2541–2546. IEEE, 2022.

A Figures and Experiments

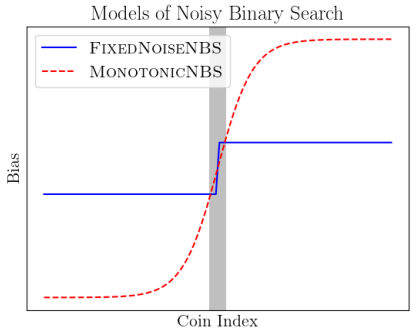


Figure 1 In FixedNoiseNBS, every coin is ε-far from the true i^* that must be found. We consider MonotonicNBS, where many coins may be close to the threshold and the goal is to find some good coin (the gray shaded region).

Applying NBS. To demonstrate the practicality of BayesianScreeningSearch we compare it to standard binary search with repetition (NaiveNBS) and the two algorithms of [10] (KKBacktracking and KKMultiplicativeWeights).

To fairly compare between these algorithms, we can't just use the descriptions given in [10], as the constants used in analysis are not optimized. We leverage BayesianScreeningSearch to address this. We tweak the listed algorithms so they take a sample budget as input which they allocate among all their stages. To estimate how large a budget is needed for algorithm $\mathcal{A}$ to perform well on distribution $\mathcal{D}$, we run BayesianScreeningSearch where when the ith coin is flipped we run $\mathcal{A}$ on some input drawn from $\mathcal{D}$, and return 1 if $\mathcal{A}$ succeeds and 0 if $\mathcal{A}$ fails. By setting $\tau = .8, .9$ and $\varepsilon = .05$, we get upper and lower bounds for how many samples is needed to get $\delta = .15$ failure probability.

Experiments. We compare results on 4 different problem distributions: Standard, Biased, Lopsided, and Wide.

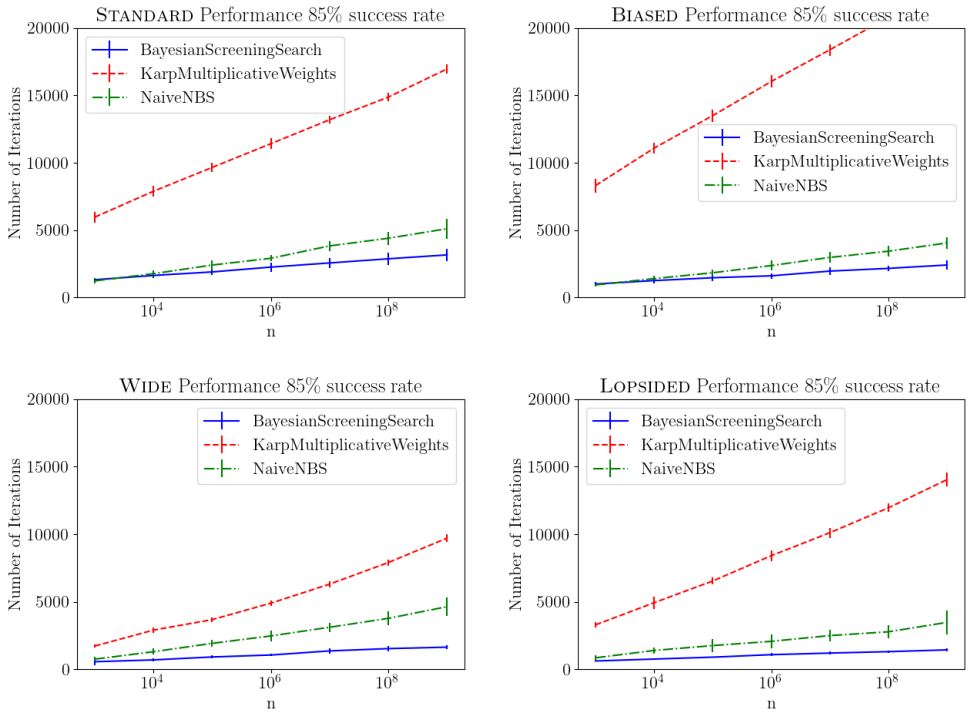


Figure 2 Performance of various MONOTONICNBS algorithms for listed distributions.

- STANDARD. $p_i \in \{\tau - \varepsilon, \tau + \varepsilon\}$, $\tau = \frac{1}{2}, \varepsilon = .1$, the transition interval chosen uniformly at random..
- BIASED. $p_i \in \{\tau - \varepsilon, \tau + \varepsilon\}, \tau = \frac{3}{4}, \varepsilon = .1$, the transition interval chosen uniformly at random.
- LOPSIDED. $p_i \in \{\tau - .6\varepsilon, \tau + \varepsilon\}$, $\tau = \frac{1}{2}, \varepsilon = .1$, the transition interval chosen uniformly at random..
- WIDE. we choose an interval (uniformly at random) of size $10 \log n$ that linearly interpolates between $\tau - \varepsilon$ and $\tau + \varepsilon$, and set the rest to be $p_i \in \{\tau - \varepsilon, \tau + \varepsilon\}, \tau = \frac{1}{2}, \varepsilon = .1$.

Results. We remark that KKBACKTRACKING performed markedly worse than the other algorithms, and so is not included in the figures. For reference, for STANDARD, $N = 1000$ KKBACKTRACKING required $m > 2.9 \times 10^6$ samples, while the other algorithms need $m < 6000$ samples (see Figure 2).

We find that KKMULTIPLICATIVEWEIGHTS is outperformed by NAIVENBS on all of these distributions. In contrast, BAYESIANSCREENINGSEARCH outperforms NAIVENBS for $n > 10^3$.

When $\tau \neq \frac{1}{2}$ the difference between BAYESIANSCREENINGSEARCH, NAIVENBS and the [10] algorithms increases. This is in line with our theory, as the first two perform better when τ is further from $\frac{1}{2}$, while the [10] algorithms reduce to the case $\tau = \frac{1}{2}$, losing a constant factor.

More details on the experiments are in the full version.

B Computations

This section gives the statements of some approximations used in the body of the paper. The proofs are in the full version.

We give explicit formulas for some functions used in this paper

$$z = 2^{\frac{H(\tau-\varepsilon)-H(\tau+\varepsilon)}{2\varepsilon}} \tag{11}$$

$$C_{\tau,\varepsilon} = \log_2(z+1) + \frac{\tau-\epsilon}{2\epsilon}H(\tau+\epsilon) - \frac{\tau+\epsilon}{2\epsilon}H(\tau-\epsilon) \tag{12}$$

$$q = \frac{(1-\tau+\varepsilon) - \frac{1}{1+z}}{2\varepsilon} \tag{13}$$

▶ **Lemma 9.**

$$C_{\tau,\varepsilon} = (\tau+\epsilon)\log_2(\frac{\tau+\epsilon}{\tau+(2q-1)\varepsilon}) + (1-\tau-\epsilon)\log_2(\frac{1-\tau-\epsilon}{1-\tau-(2q-1)\varepsilon})$$

and

$$C_{\tau,\varepsilon} = (\tau-\epsilon)\log_2(\frac{\tau-\epsilon}{\tau+(2q-1)\varepsilon}) + (1-\tau+\epsilon)\log_2(\frac{1-\tau+\epsilon}{1-\tau-(2q-1)\varepsilon})$$

▶ **Lemma 10.** *For* $\varepsilon \le \frac{1}{2}\min(\tau, 1-\tau)$,

$$\frac{1}{2\log 2}\frac{\varepsilon^2}{\tau(1-\tau)} \le C_{\tau,\varepsilon} \le \frac{1}{\log 2}\frac{\varepsilon^2}{\tau(1-\tau)}.$$

▶ **Lemma 11.** *For* $0 < \varepsilon \le \frac{1}{2}\min(\tau, 1-\tau)$,

$$|q - \frac{1}{2}| \le \frac{2\varepsilon}{\tau(1-\tau)}.$$

▶ **Lemma 12.**

$$C_{\tau,(1-o(1))\varepsilon} \ge (1-o(1))C_{\tau,\varepsilon}$$

▶ **Lemma 13.** *For* $\varepsilon < \frac{1}{2}\min(\tau, 1-\tau)$,

- $\log_2 d_{0,0} \ge -3\frac{\varepsilon}{1-\tau}$
- $\log_2 \frac{1}{d_{0,1}} \ge -3\frac{\varepsilon}{1-\tau}$
- $\log_2 \frac{1}{d_{1,0}} \ge -3\frac{\varepsilon}{\tau}$
- $\log_2 d_{1,1} \ge -3\frac{\varepsilon}{\tau}$

C Omitted Proofs

▶ **Lemma 6** (Bayesian performance). *Consider any* $0 < \varepsilon, \tau, \delta, \gamma < 1$ *with* $\gamma \le \frac{1}{7}$, $\varepsilon < \min(\tau, 1-\tau)/2$, *and let* L *be the list of intervals returned by* BayesLearn, *when run for*

$$\frac{1+O(\gamma)}{C_{\tau,\varepsilon}} \cdot \left(\log_2 n + O(\sqrt{\log n \log\frac{1}{\delta}} + \log\frac{1}{\delta})\right)$$

iterations on an MonotonicNBS *instance. With probability* $1-\delta$, *at least a* γ *fraction of the intervals in* L *are* (τ,ε)*-good.*

Proof of Lemma 6. In this proof we omit some of the routine calculations, see the full version for the complete proof.

Recall that Φ is given by the sum of equations (8) and (9).

$$6C_{\tau,\varepsilon}(|\{j \in L | j \in [l,r]\}| - \gamma|L|) \tag{8}$$

$$\log_2 w(a) \tag{9}$$

Reduction to $\Phi > 0$. First note that $\Phi_1 = -\log_2(n-1)$, as initially L is empty so (8) is 0, and we initialize w as uniform so $w(a) = \frac{1}{n-1}$. Next note that if (8) > 0, then

$$\begin{aligned} 6C_{\tau,\varepsilon}(|\{x \in L | x \in [l,r]\}| - \gamma|L|) &> 0 \\ |\{x \in L | x \in [l,r]\}| &> \gamma|L| \end{aligned}$$

So there are strictly more than $\gamma|L|$ good intervals in L. Next note that (9) is ≤ 0 always, so $\Phi > 0 \implies (8) > 0$. So it suffices to show that with δ failure probability $\Phi_{t+1} > 0$, where $t = \frac{1+O(\gamma)}{C_{\tau,\varepsilon}}(\log_2 n + O(\sqrt{\log n \log \frac{1}{\delta}} + \log \frac{1}{\delta}))$

Establishing a submartingale. By a stochastic domination argument, we can consider the worst case where all coins sampled have bias in $[\tau - \varepsilon, \tau + \varepsilon]$. Suppose that we are flipping a coin that has bias $p \leq \tau - \varepsilon$. The potential function consists of two parts, one of which does not depend on the flip, and the other which does. We see that a tails increases the potential function, while a heads decreases it. So $p = \tau - \varepsilon$ is the worst case. The argument for when $p \geq \tau + \varepsilon$ is symmetric. We do not need to worry about how this affects future states as Lemma 7 conditions on the prior flips.

Let σ_i^2 be the variance of the difference random variables $\Phi_{i+1} - \Phi_i$, then we note that σ_i^2 is a Bernoulli random variable with parameter $p \in [\tau - \varepsilon, \tau + \varepsilon]$, that is scaled by at most a $\max(\log_2 d_{1,0} - \log_2 d_{0,0}, \log_2 d_{0,1} - \log_2 d_{1,1}) \lesssim \frac{\varepsilon}{\tau(1-\tau)}$ factor, therefore

$$\sigma_i^2 \lesssim p(1-p)\left(\frac{\varepsilon}{\tau(1-\tau)}\right)^2 \lesssim \tau(1-\tau)\left(\frac{\varepsilon}{\tau(1-\tau)}\right)^2 = \frac{\varepsilon^2}{\tau(1-\tau)} \lesssim C_{\tau,\varepsilon}$$

Where we use the fact that $\varepsilon \leq \frac{\min(\tau, 1-\tau)}{2}$. Therefore $\sigma_i^2 \lesssim C_{\tau,\varepsilon}$.

Freedman's inequality. For brevity let $g = (1 - O(\gamma))C_{\tau,\varepsilon}$, the lower bound given in Lemma 7.

$$\begin{aligned} \Pr[\Phi_{t+1} \leq 0] &= \Pr[\Phi_{t+1} - \Phi_1 \leq -\Phi_1] \\ &\leq \exp\left(-\frac{2(-gt - \Phi_1)^2}{\sum_{i=1}^t \sigma_i^2 + O(\frac{\varepsilon}{\tau(1-\tau)})(gt + \Phi_1)}\right) \quad \text{Freedman's when } gt \geq -\Phi_1 \\ &\leq \exp(-\frac{O(1)}{tC_{\tau,\varepsilon}} \cdot (g^2t^2 + 2gt\Phi_1 + \Phi_1^2)) \end{aligned}$$

Bounding this expression by δ, we get

$$g^2t^2 + (2g\Phi_1 - \log_{O(1)}(1/\delta)C_{\tau,\varepsilon})t + \Phi_1^2 \geq 0 \tag{14}$$

(14) is a quadratic with respect to t, and has a positive leading coefficient. Applying the quadratic formula, if we set

$$t \geq \frac{-\Phi_1}{g} + \frac{C_{\tau,\varepsilon}\log_{O(1)}(1/\delta)}{2g^2} + \frac{\sqrt{(\log_{O(1)}(1/\delta)C_{\tau,\varepsilon})^2 - 4g\Phi_1\log_{O(1)}(1/\delta)C_{\tau,\varepsilon}}}{2g^2} \tag{15}$$

then (14) holds. As $\Phi_1 = -\log_2(n-1), g = (1 - O(\gamma))C_{\tau,\varepsilon}$ we get that (15) is

$$\frac{1}{1-O(\gamma)} \cdot \frac{1}{C_{\tau,\varepsilon}}(\log_2 n + O(\sqrt{\log n \log \frac{1}{\delta}} + \log \frac{1}{\delta}))$$

As $\frac{1}{1-O(\gamma)}$ is $1 + O(\gamma)$, our lemma holds. ◀

Algorithm 4 Noisy Binary Search that gets the optimal expected queries.

1: **procedure** SillyBayesianScreeningSearch($\{c_i\}_{i=1}^n, n, \tau, \varepsilon, \delta$)

2: **return** $\begin{cases} \textsc{Random}([n-1]) & \text{w.p. } \delta - \delta/\log_2 n \\ \textsc{BayesianScreeningSearch}(\{c_i\}_{i=1}^n, n, \tau, \varepsilon, \frac{\delta}{\log(n)}) & \text{otherwise} \end{cases}$

Proof of Corollary 3. The failure probability of SillyBayesianScreeningSearch is $\leq \delta - \delta/\log_2 n + (1-\delta+\delta/\log_2 n)\delta/\log_2 n = \delta - \delta^2/\log_2 n + \delta^2/\log_2^2 n \leq \delta$. We use 0 samples with probability $\delta - \delta/\log_2 n$, and the expression in Theorem 1 with $\delta' = \delta/\log_2 n$, with probability $1 - \delta + \delta/\log_2 n$. ◀

Proof of Lemma 8. Omitted, see the full version. ◀

▶ **Lemma 14** (Shannon's Strong Converse Theorem). *Over any discrete memoryless channel, for $R > C$*

$$P_e \geq 1 - \frac{K_1}{n(R-C)^2} - \exp(-K_2 n(R-C))$$

where P_e is the probability of error, K_1, K_2 are positive constants which depend on the channel, n is the input alphabet size, R is the rate of information, and C is the channel capacity [8].

Proof of Theorem 2. Let $\alpha = \frac{1}{1+\frac{K}{C_{\tau,\varepsilon}\sqrt{\beta(n-1)}}}$, for constant K to be determined later. Suppose that $\mathcal{A}$ uses at most $\alpha\frac{\log_2(n-1)}{C_{\tau,\varepsilon}}$ samples with probability at least $\delta + \beta$. Let $\mathcal{A}'$ be the algorithm that runs $\mathcal{A}$, but outputs a random answer if $\mathcal{A}$ uses more than $\alpha\frac{\log_2(n-1)}{C_{\tau,\varepsilon}}$ samples. $\mathcal{A}'$ fails only whenever $\mathcal{A}$ fails or uses more than $\alpha\frac{\log_2(n-1)}{C_{\tau,\varepsilon}}$ samples, so by a union bound $\mathcal{A}'$ has a failure probability of at most $1-\beta$. By Lemma 8 we can construct a protocol over a discrete memoryless channel with capacity $C_{\tau,\varepsilon}$ that communicates at rate $R = \frac{\log_2(n-1)}{\alpha\frac{\log_2(n-1)}{C_{\tau,\varepsilon}}} = \frac{C_{\tau,\varepsilon}}{\alpha} = \frac{C}{\alpha}$ with a failure probability of at most $1-\beta$.

By Lemma 14 we have that

$$\begin{aligned} 1-\beta &\geq 1 - \frac{K_1}{(n-1)(R-C)^2} - \exp(-K_2(n-1)(R-C)) \\ &= 1 - \frac{K_1}{(n-1)((1/\alpha-1)C)^2} - \exp(-(n-1)K_2((1/\alpha-1)C)) \\ &= 1 - \frac{K_1\beta}{K} - \exp(-\sqrt{\frac{n-1}{\beta}}K_2K) \\ &\geq 1 - \frac{\beta}{2} && \text{sufficiently large } K \text{ and } n \end{aligned}$$

which is a contradiction. Therefore with probability at least $1-\delta-\beta$ $\mathcal{A}$ uses $\frac{1}{1+\frac{K}{\sqrt{\beta(n-1)}(C_{\tau,\varepsilon})}}\frac{1}{C_{\tau,\varepsilon}}\log_2 n$ samples. ◀

Proof of Theorem 4. Omitted, see the full version. ◀

Solution Discovery via Reconfiguration for Problems in P

Mario Grobler
University of Bremen, Germany

Stephanie Maaz
University of Waterloo, Canada

Nicole Megow
University of Bremen, Germany

Amer E. Mouawad
American University of Beirut, Lebanon

Vijayaragunathan Ramamoorthi
University of Bremen, Germany

Daniel Schmand
University of Bremen, Germany

Sebastian Siebertz
University of Bremen, Germany

Abstract

In the recently introduced framework of solution discovery via reconfiguration [Fellows et al., ECAI 2023], we are given an initial configuration of k tokens on a graph and the question is whether we can transform this configuration into a feasible solution (for some problem) via a bounded number b of small modification steps. In this work, we study solution discovery variants of polynomial-time solvable problems, namely Spanning Tree Discovery, Shortest Path Discovery, Matching Discovery, and Vertex/Edge Cut Discovery in the unrestricted token addition/removal model, the token jumping model, and the token sliding model. In the unrestricted token addition/removal model, we show that all four discovery variants remain in P. For the token jumping model we also prove containment in P, except for Vertex/Edge Cut Discovery, for which we prove NP-completeness. Finally, in the token sliding model, almost all considered problems become NP-complete, the exception being Spanning Tree Discovery, which remains polynomial-time solvable. We then study the parameterized complexity of the NP-complete problems and provide a full classification of tractability with respect to the parameters solution size (number of tokens) k and transformation budget (number of steps) b. Along the way, we observe strong connections between the solution discovery variants of our base problems and their (weighted) rainbow variants as well as their red-blue variants with cardinality constraints.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Fixed parameter tractability; Mathematics of computing → Combinatorics

Keywords and phrases solution discovery, reconfiguration, spanning tree, shortest path, matching, cut

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.76

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2311.13478

Funding *Vijayaragunathan Ramamoorthi*: Funded by the "Mind, Media, Machines" high-profile area at the University of Bremen, jointly with *Nicole Megow, Daniel Schmand* and *Sebastian Siebertz.*

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 76; pp. 76:1–76:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In classical optimization problems, we are given a problem instance and the task is to compute an optimal solution. However, in many applications and real-world scenarios we are already provided with a current solution, albeit non-optimal or infeasible for a given instance. Depending on the application, it might be desirable to find an optimal or feasible solution via a bounded number of small modification steps starting from the current solution.

Very prominent examples for such "systems" are typically settings where humans are involved in the system and big changes to the running system are not easily implementable or even accepted. When optimizing public transport lines, shift plans, or when assigning workers to tasks it is clearly desirable to aim for an optimal solution that is as similar as possible to the current state of the system. Fellows et al. [13] recently introduced the *solution discovery via reconfiguration* framework addressing the computational aspects of such problems. In their model, an optimizer is given a problem instance together with a current (possibly) infeasible solution. The aim is to decide whether a feasible solution to the given problem can be constructed by applying only a bounded number of changes to the current state. We extend this line of work and focus on core polynomial-time solvable problems on graphs, namely Spanning Tree, Shortest Path, Matching, and Vertex/Edge Cut. More precisely, for any of the four aforementioned problems, say Π, we consider instances consisting of a graph G, a budget b, and a starting configuration of k tokens, which is not necessarily a feasible solution for Π (and where tokens either occupy vertices or edges of G). The goal is to decide whether one can transform the starting configuration of tokens into a feasible solution for Π by applying at most b "local changes".

The solution discovery framework is inspired by approaches transforming one solution to another such as *local search*, *reoptimization*, and *combinatorial reconfiguration*. *Local search* is an algorithmic paradigm that is based on iterative improvement of solutions in a previously defined neighborhood. In contrast to our setting, *local search* typically improves the current solution in each step, while we allow arbitrary configurations between the starting and ending configurations (our only restriction is that each vertex/edge can be occupied by at most one token). In *reoptimization* the aim is also to compute optimal solutions starting from optimal solutions of "neighboring" instances (distance between instances being usually defined as the number of vertex/edge addition/deletion required to make the two graphs isomorphic). Closely related is the field of *sensitivity analysis*, a very classical area studying how sensitive an optimal solution is (how it reacts) to small changes in the input. In *combinatorial reconfiguration* we are also given a starting solution, but additionally a *target solution*, and very often constraints on the intermediate steps, e. g., that every intermediate step maintains a valid solution. In our setting the target and intermediate steps are not explicitly specified, but we aim for any final configuration satisfying some desired properties.

As an illustrating example application, consider the scenario in which a city experiences severe weather conditions, leading to a rapid increase in river water levels. The city relies on the protection of a dam, but there is a foreseeable risk that the dam may eventually fail. We assume that the dam breaks (continuously) from point A to point B, where the (shortest) distance between A and B in our graph-view of the world is exactly k vertices (including A and B). In anticipation of such emergencies, the city has also placed (at least) k sandbags in fixed locations across the city so that one can move them as fast as possible to avoid greater damage (we here make the simplifying assumption that each unit of the broken dam can be fixed by a single sandbag). When the dam breaks we can easily compute a shortest path between A and B, which also allows us to compute the minimum number of

sandbags required to stop the flowing water. However, computing such a shortest path is not enough in this situation. Instead, we additionally need to account for sandbags having to move to the appropriate locations as quickly as possible. This corresponds exactly to the problem of finding a shortest path (between two fixed vertices) that is quickly reachable from a predefined set of positions (vertices) in the graph. In other words, this motivates the study of discovery variants of the Shortest Path problem.

An alternative perspective at solution discovery problems is as follows. Consider a vertex (resp. edge) selection problem Π on graphs. Assume that each element (vertex or edge) in the solution of size k must be *supported* by one of k support points, which are located at k different elements of the input graph, and which can each support exactly one element of the solution. The cost of supporting an element is measured by the distance to the chosen support point. The problem of deciding whether there exists a solution that can be supported with cost b corresponds to a discovery variant of problem Π.

Before we proceed and state our results we quickly recall the solution discovery framework of Fellows et al. [13] (see preliminaries for formal definitions). Consider an instance of a vertex (resp. edge) selection problem Π on graphs, where some vertices (resp. edges) of the input graph are occupied by (distinct and indistinguishable) tokens. A token may be moved (in a specific way depending on the concrete model) for a cost of 1. In the *unrestricted token addition/removal* model[1], an existing token may be removed, or a new token may be placed on an unoccupied vertex (resp. edge) for a cost of 1. In the *token jumping* model, a token may be moved from one vertex (resp. edge) to an arbitrary unoccupied vertex (resp. edge) for a cost of 1. In the *token sliding* model, a token may be moved to a neighboring unoccupied vertex (resp. edge) for a cost of 1. The goal is to move the tokens such that they form a valid solution for problem Π within the given budget. We remark that these notions of token moves have also been studied in the realm of combinatorial reconfiguration [21, 24].

Fellows et al. [13] considered the solution discovery variants of (computationally hard) fundamental graph problems, namely Vertex Cover, Independent Set, Dominating Set and Vertex Coloring. The complexity of solution discovery for Vertex Coloring in the color flipping model was studied in [16] under the name k-Fix and in the color swapping model in [10] under the name k-Swap. Since these problems, which we call *base problems*, are NP-complete, it is not surprising that their solution discovery variants are also NP-complete in all of the aforementioned models. In this work, we continue the examination of the solution discovery framework and focus on the discovery variants of polynomial-time solvable base problems, namely Spanning Tree Discovery, Shortest Path Discovery, Matching Discovery, and Vertex/Edge Cut Discovery. When a base problem is polynomial-time solvable, one may, given an instance with a partial or infeasible solution, efficiently compute an optimal solution from scratch. However, as previously illustrated, there are situations in which a solution that is close to a currently established configuration is more desirable. As we show in this work, the constraints put on these problems in the solution discovery framework, namely a limited number of changes, can drastically alter their complexities.

We observe strong connections between the solution discovery variants of our base problems and their *weighted rainbow variants* as well as their *red-blue variants with cardinality constraints*. An instance of a weighted rainbow vertex (resp. edge) selection problem consists

[1] We call the model "unrestricted" to differentiate it from the addition/removal model usually considered in reconfiguration problems as the latter imposes a lower or upper bound on the number of tokens in the graph at all times.

of a weighted vertex (resp. edge) colored graph, and the solution of such an instance may not contain two vertices (resp. edges) of the same color, while "collecting" a certain amount of weight. We show that if (the parameterized version of) the weighted rainbow variant of problem Π admits a fixed-parameter tractable (FPT-) algorithm, then this algorithm can be used to design a fixed-parameter tractable algorithm for the solution discovery variant of Π in the token sliding model. Similarly, solving the solution discovery variant of a problem in the token jumping model boils down to solving the red-blue variant of that same problem. An instance of a red-blue vertex (resp. edge) selection problem consists of a graph where every vertex (resp. edge) is either colored red or blue and two integer parameters k and b. The goal is to find a solution of size k that contains at most b blue vertices (resp. edges).

1.1 Our results

We provide a full classification of tractability vs. intractability with respect to the classical as well as the parameterized complexity of the aforementioned solution discovery problems in all three token models (Table 1). Moreover, we prove some results for rainbow problems as well as red-blue problems, which we believe to be of independent interest. Our main results can be summarized as follows:

- Spanning Tree Discovery, Shortest Path Discovery, Matching Discovery, and Vertex/Edge Cut Discovery are polynomial-time solvable in the unrestricted token addition/removal model.
- Spanning Tree Discovery, Shortest Path Discovery and Matching Discovery are polynomial-time solvable, while Vertex/Edge Cut Discovery is NP-complete in the token jumping model.
- Spanning Tree Discovery is polynomial-time solvable, while Shortest Path Discovery, Matching Discovery, and Vertex/Edge Cut Discovery are NP-complete in the token sliding model.

We then consider the parameterized complexity of the NP-complete discovery problems and establish the following connection with their rainbow variants.

- Meta theorem: For an optimization problem Π, if the weighted Rainbow-Π problem (parameterized by solution size k) admits an FPT algorithm, then Π-Discovery (parameterized by solution size k) admits an FPT algorithm in the token sliding model.

FPT algorithms for the Weighted Rainbow Shortest Path problem [1], Weighted Rainbow Matching problem [19], and the Weighted Rainbow Vertex/Edge Cut problem (which we provide in this paper) immediately imply FPT algorithms for the discovery variants parameterized by k (in the token sliding model). We demonstrate the power of our meta theorem by using it to show that Vertex/Edge Cut Discovery is FPT with respect to parameter k in the sliding model. For Shortest Path Discovery and Matching Discovery, we then give more intuitive and direct FPT algorithms, which also achieve better running times. We conclude by studying the parameterized complexity of all hard problems (not covered by the meta-theorem) when parameterized by either k or b, obtaining the following results.

- Shortest Path Discovery, Matching Discovery, and Vertex/Edge Cut Discovery are FPT when parameterized by solution size k in the token sliding model. Furthermore, Vertex/Edge Cut Discovery is FPT when parameterized by k in the token jumping model.

Table 1 Overview of our results.

	Spanning Tree	Shortest Path	Matching	Vertex/Edge Cut
Discovery Add/Rem.	in P	in P	in P	in P
Discovery Jumping	in P	in P	in P	NP-c., FPT[k], W[1]-hard[b]
Discovery Sliding	in P	NP-complete, FPT[k], FPT[b]	NP-c., FPT[k], W[1]-hard[b]	NP-c., FPT[k], W[1]-hard[b]
Rainbow	in P [6]	NP-complete	NP-complete on paths [25]	NP-complete [3], NP-c. on planar
Red-Blue	in P	in P	in P	NP-c., FPT[k], W[1]-hard[b]

- Shortest Path Discovery is FPT, while Matching Discovery and Vertex/Edge Cut Discovery are W[1]-hard when parameterized by the budget b in the token sliding model. Furthermore, Vertex/Edge Cut Discovery is W[1]-hard when parameterized by the budget b in the token jumping model.

1.2 Related work

The solution discovery framework is closely related to the combinatorial reconfiguration framework, introduced by Ito et al. [21] and studied widely since then. In the reconfiguration variant of a problem we are given an initial solution S and a target solution T and the question is whether S can be transformed into T by a sequence of reconfiguration steps (e. g., token additions/removals, token jumps, or token slides) such that each intermediate configuration also constitutes a solution.

The Minimum Spanning Tree Reconfiguration problem by edge exchanges, i.e., token jumps, was first studied by Ito et al. [21]. They showed that the problem is in P by extending the exchange property of matroids to the reconfiguration of weighted matroid bases. Shortest Path Reconfiguration was introduced by Kamiński et al. [23] and shown to be PSPACE-complete by Bonsma [4]. Reconfiguration of perfect matchings was studied by Ito et al. [22]. The Vertex Cut Reconfiguration and Minimum Vertex Cut Reconfiguration problems were studied by Gomes et al. [18, 17]. For further related work on reconfiguration problems we refer the reader to the surveys of van den Heuvel [20], Nishimura [28], and Bousquet et al. [5].

Rainbow spanning trees have been investigated by Broersma and Li [6]. They characterize graphs in which there exists a rainbow spanning tree via matroid intersection. The question to decide whether a graph contains a rainbow path has been studied in the classical and influential work of Alon et al. [1] that introduced the color coding technique. In the related Rainbow s-t-Connectivity problem the question is to decide whether there exists a rainbow path between s and t, that is, a path on which no color repeats [31]. To the best of our knowledge the Rainbow Shortest Path problem has not been studied in the literature. We refer the reader to [7] for more background. The Rainbow Matching problem is NP-complete, even when restricted to properly edge-colored paths [25]. The Rainbow s-t-Cut problem is known to be NP-complete [3] on general graphs. We show that this problem remains NP-complete even if we restrict it to the class of planar graphs.

We furthermore consider red-blue variants of our base problems. Graphs are now vertex (resp. edge) colored with colors red and blue. We are given two integers k and b and the question is whether there exists a solution of size k using at most b blue vertices (resp. edges). To the best of our knowledge these problem variants have not been studied in the literature, however, variants where we have a cardinality constraint on both colors are related, but seem to be more difficult to solve. For example in the Color Constrained Matching problem [29] we are given a 2-edge-colored graph (colors red and blue) and two parameters k and w and we search for a matching of size k with at most w blue and at most w red edges. This problem is known to be at least as hard as the Exact Matching problem [29] (via a logarithmic-space reduction), which was introduced in [30]. Here, where we are given a red-blue edge-colored graph and a parameter b and the question is whether there exists a perfect matching with exactly b blue edges. The complexity of Exact Matching has been open for more than 40 years [26].

2 Preliminaries

We denote the set of non-negative integers by $\mathbb{N}$ and the set of non-negative reals by $\mathbb{R}_+$. For $k \in \mathbb{N}$ we define $[k] = \{1, 2, \ldots, k\}$ with the convention $[0] = \varnothing$.

Graphs. We consider finite and loopless graphs. An undirected simple graph G consists of its vertex set $V(G)$ and edge set $E(G)$, where $E(G)$ is a subset of all two element sets of $V(G)$. Similarly, the edge set $E(G)$ of a directed simple graph is a subset of pairs of its vertices. In a multigraph we allow $E(G)$ to be a multiset. We assume our graphs to be undirected and simple if not stated otherwise. We denote an edge connecting vertices u and v by uv. Observe that $uv = vu$ for every undirected edge $uv \in E(G)$. A sequence $v_1, \ldots, v_q$ of pairwise distinct vertices is a path of length $q-1$ if $v_i v_{i+1} \in E(G)$ for all $1 \leq i < q$. We write P_q for the path of length q. The distance $\mathrm{dist}_G(u, v)$ (or simply $\mathrm{dist}(u, v)$ if G is clear) between two vertices $u, v \in V(G)$ is the length of a shortest path starting in u and ending in v in G. A graph is d-degenerate if it can be reduced to the empty graph by iterative removal of vertices of degree at most d. For example, forests are 1-degenerate. A graph is bipartite if its vertices can be partitioned into two parts A, B such that no edge has both its endpoints in the same part. Equivalently, a graph is bipartite if it does not contain cycles of odd length. For a vertex subset $S \subseteq V(G)$, we denote by $G[S]$ the subgraph of G induced by S, i.e., the graph with vertex set S and edge set $\{uv \in E(G) \mid u, v \in S\}$. Likewise, for an edge subset $M \subseteq E(G)$, we denote by $G[M]$ the graph with edge set M and vertex set $\{u, v \mid uv \in M\}$.

An edge coloring $\varphi : E(G) \to \mathcal{C}$ is a function mapping each edge $e \in E(G)$ to a color $\varphi(e) \in \mathcal{C}$. Similarly, a vertex coloring assigns colors to vertices. An edge-weight function is a function $w : E(G) \to \mathbb{R}_+$, and similarly a vertex-weight function assigns weights to vertices. We denote colored weighted graphs by tuples (G, w, φ). The weight of a set of vertices/edges is the sum of the weights of its elements.

Solution discovery. Let G be a graph. A *configuration* of G is either a subset of its vertices or a subset of its edges. We formalize the notions of token moves. In the *unrestricted token addition/removal* model[2], a configuration C' can be obtained (in one step) from C, written $C \vdash C'$, if $C' = C \cup \{x\}$ for an element $x \notin C$, or if $C' = C \setminus \{x\}$ for an element

[2] Recall that this definition differs from the definition in [13].

$x \in C$. In the *token jumping* model, a configuration C' can be obtained (in one step) from C if $C' = (C \setminus \{y\}) \cup \{x\}$ for elements $y \in C$ and $x \notin C$. In the *token sliding* model, a configuration C' can be obtained (in one step) from C if $C' = (C \setminus \{y\}) \cup \{x\}$ for elements $y \in C$ and $x \notin C$ if x and y are neighbors in G, that is, if $x, y \in V(G)$, then $xy \in E(G)$; and if $x, y \in E(G)$, then $x \cap y \neq \varnothing$. If C' can be obtained from C (in any model), we write $C \vdash C'$. A *discovery sequence* of length ℓ in G is a sequence of configurations $C_0C_1 \dots C_\ell$ of G such that $C_i \vdash C_{i+1}$ for all $0 \leq i < \ell$.

Let Π be a vertex (resp. edge) selection problem, i.e., a problem defined on graphs such that a solution consists of a subset of vertices (resp. edges) satisfying certain requirements. The Π-Discovery problem is defined as follows. We are given a graph G, a subset $S \subseteq V(G)$ (resp. $S \subseteq E(G)$) of size k (which at this point is not necessarily a solution for Π), and a budget b (as a non-negative integer). The goal is to decide whether there exists a discovery sequence $C_0C_1 \dots C_\ell$ in G for some $\ell \leq b$ such that $S = C_0$ and C_ℓ is a solution for Π.

Note that for discovery problems in the token sliding model we can always assume that $b \leq kn$, where n is the number of vertices in the input graph. This follows from the fact that each token will have to traverse a path of length at most n to reach its target position. For discovery problems in the token jumping model we can always assume $b \leq k$, as it is sufficient to move every token at most once. Similarly, for the unrestricted token addition/removal model we can always assume that $b \leq n$ for vertex selection problems and $b \leq m$ for edge selection problems, where m is the number of edges in the input graph. As k is trivially upper-bounded by n for vertex selection problems (resp. m for edge selection problems), all solution discovery variants we consider are in NP and thus proving NP-hardness suffices to prove NP-completeness.

Parameterized complexity. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed finite alphabet. For an instance $(x, \kappa) \in \Sigma^* \times \mathbb{N}$, κ is called the *parameter*. The problem L is called *fixed-parameter tractable*, FPT for short, if there exists an algorithm that on input (x, κ) decides in time $f(\kappa) \cdot |(x, \kappa)|^c$ whether $(x, \kappa) \in L$, for a computable function f and constant c.

The *W-hierarchy* is a collection of parameterized complexity classes $\mathsf{FPT} \subseteq \mathsf{W[1]} \subseteq \mathsf{W[2]} \subseteq \dots$. It is standard to assume that the inclusion $\mathsf{FPT} \subseteq \mathsf{W[1]}$ is strict. Therefore, showing intractability in the parameterized setting is usually accomplished by establishing an FPT-reduction from a W-hard problem.

Let $L, L' \subseteq \Sigma^* \times \mathbb{N}$ be parameterized problems. A *parameterized reduction* from L to L' is an algorithm that, given an instance (x, κ) of L, outputs an instance (x', κ') of L' such that $(x, \kappa) \in L \Leftrightarrow (x', \kappa') \in L'$, $\kappa' \leq g(\kappa)$ for some computable function g, and the running time of the algorithm is bounded by $f(\kappa) \cdot |(x, \kappa)|^c$ for some computable function f and constant c. We refer to the textbooks [9, 12, 14] for extensive background on parameterized complexity.

3 Spanning trees

A *spanning tree* in a connected graph G is a subset of edges $E' \subseteq E(G)$, where $|E'| = |V(G)| - 1$ and $G[E']$ is a tree containing all vertices of G. In the Spanning Tree problem we are given a graph G and the goal is to compute a spanning tree in G.

3.1 Rainbow minimum spanning trees

We reduce the problem of discovering spanning trees in the sliding model to the problem of finding (weighted) rainbow spanning trees. A spanning tree $T \subseteq E(G)$ in a weighted edge-colored multigraph (G, w, φ) is a *rainbow spanning tree* if every edge in T has a distinct color, i.e., $\forall e, e' \in T$ we have $\varphi(e) = \varphi(e')$ if and only if $e = e'$. In the Rainbow Minimum Spanning Tree (Rainbow MST) problem, we are given (G, w, φ) and the goal is to compute a rainbow spanning tree of minimum total weight in G, or report that no such rainbow spanning tree exists. The Rainbow Spanning Tree (Rainbow ST) problem can be defined similarly, i. e., by dropping the weights or assuming that all weights are uniform.

Rainbow spanning trees and their existence have been discussed by Broersma and Li [6]. In our reduction we will construct an instance of Rainbow Minimum Spanning Tree that trivially guarantees the existence of at least one rainbow spanning tree. We show that we can find a Rainbow MST efficiently, even in multigraphs. The proof of the theorem uses arguments similar to those presented in [6] and builds on matroid intersection.

▶ **Theorem 1.** *The* Rainbow Minimum Spanning Tree *problem in weighted edge-colored multigraphs can be solved in polynomial time.*

3.2 Spanning tree discovery

In the Spanning Tree Discovery (STD) problem in the token sliding model, we are given a graph G, an edge subset $S \subseteq E(G)$ with $|S| = |V(G)| - 1$ as a starting configuration, and a non-negative integer b. The goal is to decide whether there is a spanning tree of G that can be discovered (starting from S) using at most b token slides.

▶ **Theorem 2.** *The* Spanning Tree Discovery *problem in the token sliding model can be solved in polynomial time.*

In the full version of the paper we show that the weighted discovery problem, i.e., where we seek a spanning tree of minimum weight, can also be solved in polynomial time.

4 Shortest paths

A *shortest path* between two vertices of a graph G, say s and t, is a path connecting s and t of minimum length. In the Shortest Path problem we are given a graph G and $s, t \in V(G)$ and the goal is to compute a shortest path between s and t in G.

4.1 Rainbow shortest paths

In the Rainbow Shortest Path (Rainbow SP) problem we are given a vertex-colored graph (G, φ) and two vertices $s, t \in V(G)$. A shortest path P from s to t in G is a *rainbow shortest path* if every vertex in P has a distinct color, i. e., $\forall v, v' \in V(P)$, we have $\varphi(v) = \varphi(v')$ if and only if $v = v'$. The Rainbow Shortest Path problem asks for a rainbow shortest path P from s to t in G (one can similarly define the edge-colored variant of the problem).

Rainbow paths have been studied in the literature before, specifically in relation to the rainbow vertex-connection or edge-connection number of a graph. We refer the reader to [7] for more details. In the following theorem, we show that the extra rainbow requirement makes the problem a lot harder than the base problem, even on very restricted families of graphs. To do so, we describe a reduction from the NP-complete [15] Hamiltonian Path problem to the Rainbow Shortest Path problem.

▶ **Theorem 3.** *The* Rainbow Shortest Path *problem is* NP*-complete on the class of* 2*-degenerate bipartite graphs.*

Proof. The fact that the problem is in NP is immediate. We show NP-hardness by a reduction from the Hamiltonian Path problem, which is known to be NP-complete [15]. Given an instance G of Hamiltonian Path, where G is a graph with n vertices denoted by $V(G) = \{v_1, \ldots, v_n\}$, we construct an instance (H, φ, s, t) of Rainbow Shortest Path as follows. See Figure 1 for an illustration.

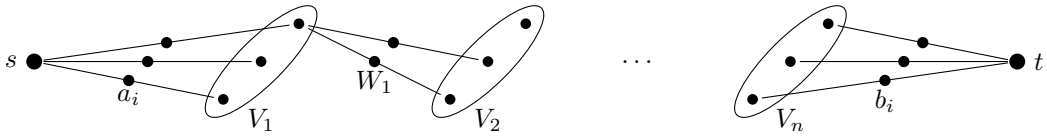


Figure 1 An illustration of the hardness reduction for the Rainbow Shortest Path problem.

First, H consists of two vertices s and t. For every pair $i, j \leq n$ we add a new vertex $v_{i,j}$ in H. We define $V_i = \{v_{i,j} \mid j \leq n\}$. Then, for every $i < n$ and $e \in E(G)$ we add a new vertex $w_{i,e}$ and define $W_i = \{w_{i,e} \mid e \in E(G)\}$. Finally, for every $i \leq n$ we add new vertices a_i and b_i. We define $A = \{a_i \mid i \leq n\}$ and $B = \{b_i \mid i \leq n\}$. We connect the vertices as follows. For every $i \leq n$, we insert the edges $\{s, a_i\}, \{a_i, v_{1,i}\}, \{v_{n,i}, b_i\}$, and $\{b_i, t\}$. Finally, for every $i < n$ and $e = \{v_j, v_\ell\}$ we insert the edges $\{v_{i,j}, w_{i,e}\}$ and $\{w_{i,e}, v_{i+1,\ell}\}$. We assign all vertices in A the same color, all vertices in B the same new color, all vertices $\{v_{i,j} \mid i \leq n\}$ the same new color (this set represents all copies of a vertex $v \in V(G)$), all vertices in W_i the same new color, all vertices in B the same new color, and s and t receive two fresh new colors. This finishes the construction of the instance (H, φ, s, t).

Observe that H is indeed bipartite and 2-degenerate; all vertices of the form $w_{i,j}$ are of degree two. Their removal results in a forest, which is 1-degenerate. Bipartiteness follows from the fact that there are no edges in $G[S]$, $G[T]$, $G[V_i]$, nor $G[W_i]$.

We show that G contains a Hamiltonian path if and only if (H, φ, s, t) is a yes-instance of Rainbow Shortest Path. Observe that every shortest s-t-path in H consists of s, exactly one of the vertices $a_i \in A$, exactly one vertex from every V_i and one from every W_i, exactly one of the vertices $b_i \in B$ and t. Hence, there is no path of length less than $2n + 2$ and every such path of length $2n + 2$ is exactly of this form.

Now assume that G contains a Hamiltonian path $v_{i_1} v_{i_2} \ldots v_{i_n}$. We pick the following vertices in H. We pick a_{i_1} and b_{i_n}, the vertices v_{j,i_j}, and, for every $j < n$, we pick the vertices $w_{j,e}$ where $e = \{v_{i_j}, v_{i_{j+1}}\}$. It is not hard to see that these vertices indeed form a rainbow shortest path from s to t in H.

Conversely, assume that (H, φ, s, t) is a yes-instance of Rainbow Shortest Path. By the above observations, every shortest path must be of the same form (picking vertices from the same sets). As every rainbow shortest path cannot contain two copies of the same vertex from $V(G)$ the claim follows, finishing the proof. ◀

4.2 Shortest path discovery

In the Shortest Path Discovery (SPD) problem in the token sliding model we are given a graph G, two vertices $s, t \in V(G)$, a starting configuration $S \subseteq V(G)$ with $|S| = k$, which is equal to the number of vertices on a shortest path between s and t (including s and t), and a non-negative integer b. The goal is to decide whether we can discover a shortest path between s and t (starting from S) using at most b token slides. We denote an instance of

Shortest Path Discovery by a tuple (G, s, t, S, b). We show that the Shortest Path Discovery problem is NP-complete even when restricted to 2-degenerate bipartite graphs by a minor modification of the reduction from the Hamiltonian Path problem to the Rainbow Shortest Path problem.

▶ **Theorem 4.** *The Shortest Path Discovery problem in the token sliding model is NP-complete on the class of 2-degenerate bipartite graphs.*

In fact, since Hamiltonian Path remains NP-complete on cubic planar graphs, it turns out that the class of all graphs arising in the reduction of Theorem 4 is not only 2-degenerate but also has bounded expansion.

It remains an interesting open problem to determine whether Shortest Path Discovery is polynomial-time solvable on classes of graphs excluding a topological minor, or even on planar graphs. Next, we show that the problem is fixed-parameter tractable with respect to the parameter k as well as the parameter b.

▶ **Theorem 5.** *The Shortest Path Discovery problem in the token sliding model is fixed-parameter tractable with respect to parameter k.*

Proof. Let (G, s, t, S, b) be an instance of SPD. For every $v \in V(G)$ we compute its distance to s and delete v if the distance is larger than k. We enumerate the tokens in S as $s_0, s_1, \ldots, s_{k-1}$ such that token s_i shall slide to a vertex at distance i from s. There are $k!$ such enumerations. Now we orient and assign weights to the edges of G to obtain a weighted directed graph H. If uv is an edge such that $\text{dist}_G(s, u) = i$ and $\text{dist}_G(s, v) = i + 1$, then we orient the edge as (u, v) and assign it weight $\text{dist}_G(s_{i+1}, v)$, that is, the cost of moving token s_{i+1} to vertex v. We delete all edges that did not receive a weight, that is, all edges that connect vertices at the same distance from s.

Since H contains only edges between vertices of distance i to distance $i + 1$ from s, an s-t-path in H corresponds to a shortest s-t-path in G. Furthermore, by definition of the weights, a shortest s-t-path P (with respect to the weights) corresponds exactly to the discovery of a shortest path in G where token s_i slides to the vertex of P which is at distance i from s. We can therefore simply compute a shortest s-t-path in H and verify whether its weight is at most $b - \text{dist}_G(s, s_0)$. Since H is a DAG we can compute such a path in time $\mathcal{O}(n + m)$ and, consequently, the full algorithm runs in time $\mathcal{O}(k!(n + m))$. ◀

▶ **Theorem 6.** *The Shortest Path Discovery problem in the token sliding model is fixed-parameter tractable with respect to parameter b.*

Proof. Let (G, s, t, S, b) be an instance of SPD. Let us call the vertices at distance i from s the vertices at level i. If we can discover a solution with budget b, then it must be the case that all but at most b tokens of S are already in their correct positions. In particular, there can be at most b many levels that do not contain a token from S and at most b many levels containing two or more tokens (and each level can contain at most $b + 1$ many tokens). We call levels not containing exactly one token *very bad* levels. Now, for a level i containing exactly one token on vertex v (not a very bad level), we say that i is a *bad* level whenever one of the following is true:

- $i - 1 \geq 0$, level $i - 1$ contains exactly one token on vertex u, and $uv \notin E(G)$; or
- $i + 1 \leq \text{dist}(s, t) + 1$, level $i + 1$ contains exactly one token on vertex w, and $vw \notin E(G)$.

Note that we can have at most b very bad levels and at most $3b$ bad levels (every 3 bad levels require at least one unit of the budget); otherwise we can reject the instance. Each bad or very bad level contains at most $b+1$ tokens and at most b tokens do not belong to a level between s and t. Hence, the total number of tokens that potentially have to move is so far $5b(b+1) = 5b^2 + 5b$ (located on at most $5b$ levels).

We call a level that is neither bad nor very bad a *good* level. Note that we can group the good levels into at most $5b+1$ groups of consecutive levels. Moreover, the tokens of each group form a path. We denote those paths by $P_1, \ldots, P_q$, for $q \leq 5b+1$. Consider some P_i with at least $2b+1$ many vertices, say $P_i = v_1 \ldots v_\ell$ for $\ell \geq 2b+1$. Let v_x be a vertex of P_i such that $\ell + 1 \leq x \leq \ell - (b+1)$. In other words, there are at least b vertices preceding and at least b vertices succeeding v_x in P_i. In what follows we assume, without loss of generality, that the token on v_x does not move whenever v_x belongs to the final shortest path between s and t. This is a safe assumption because if any other token must pass via v_x to reach its destination we simply swap the roles of both tokens. In other words, getting a token on v_x does not consume any units of the budget. We claim that in a shortest discovery sequence of at most b token slides the token on vertex v_x does not move. This follows from the fact that otherwise it must be the case that either all tokens on v_y, $1 \leq y < x$, or all tokens on v_z, $\ell \geq z > x$, must move. However, as there are at least b tokens to move in either case this contradicts the fact that our discovery sequence slides no more than b tokens. To prove that either every token before or after v_x must move in a shortest discovery sequence we aim towards a contradiction. Assume that the token at level x moves but there are two levels $y < x$ and $z > x$ such that the tokens on level y and z do not move. Since the subpath between levels y and z is already a path connecting v_y and v_z (going through v_x), the movement of the token at level x can be ignored, contradicting our assumption of a shortest discovery sequence. Since at most b tokens can move, and by symmetry, we conclude that at most $2b$ many tokens on the two boundaries of P_i can move. We call the tokens on good levels that are not at distance at most b from some boundary of a P_i *fixed* tokens. Note that the number of tokens that are not fixed is at most $10b^2 + 2b$.

Putting it all together, we conclude that the set of tokens that can potentially move has size at most $15b^2 + 7b$. We can now proceed just as in the proof of Theorem 5 to obtain the required algorithm for parameter b; we can now guess the subset of tokens that move as well as the level each moving token will occupy. ◀

5 Matchings

A *matching* in a graph G is a set of edges $M \subseteq E(G)$ such that each vertex $v \in V(G)$ appears in at most one edge in M. In the Matching problem we are given a graph G and an integer k and the goal is to compute a matching of size k in G.

5.1 Rainbow matchings

We show NP-hardness of the Matching Discovery problem via a reduction from the Rainbow Matching problem. Let (G, φ) be an edge-colored graph. A matching $M \subseteq E(G)$ is said to be a *rainbow matching* if all edges in M have pairwise distinct colors. Formally, the Rainbow Matching problem is defined as follows. Given an edge-colored graph (G, φ) and an integer k, decide whether there exists a rainbow matching of size k in G.

▶ **Theorem 7** ([25]). *The Rainbow Matching problem is NP-complete, even when restricted to properly edge-colored paths.*

5.2 Matching discovery

In the Matching Discovery problem in the token sliding model we are given a graph G, a starting configuration $S \subseteq E(G)$ of size k, and a non-negative integer b. The goal is to decide whether we can discover a k-sized matching M (starting from S) using at most b token slides. Recall that a token on some edge e can only slide (in one step) to some edge e' such that e and e' are adjacent, i.e., share a vertex. We denote an instance of Matching Discovery by a tuple (G, S, b). We first show that the Matching Discovery problem is NP-complete even on restricted graph classes.

▶ **Theorem 8.** *The Matching Discovery problem in the token sliding model is NP-hard on 2-degenerate bipartite graphs.*

Furthermore, we show intractability of the problem when parameterized by the budget. This result is established by a parameterized reduction from the W[1]-hard Multicolored Clique problem and is the technically most demanding contribution of the paper.

▶ **Theorem 9.** *The Matching Discovery problem in the token sliding model is W[1]-hard when parameterized by b, even on the class of 3-degenerate graphs.*

On the positive side, we next show that the Matching Discovery problem is fixed-parameter tractable with respect to parameter $k + b$. Then we show that for any instance of the problem one can bound b by a (quadratic) function of k, which implies fixed-parameter tractability of the problem parameterized by k alone. We start with some relevant definitions.

Let $d(t, w)$ denote the minimum number of slides token t has to slide in order to become incident to vertex w. For an integer $i \in \{1, \ldots, b\}$ and a token $t \in S$, let $Y_t^i = \{w \in V(G) \mid d(t, w) = i - 1\}$ and $Z_t^i = \{w \in V(G) \mid d(t, w) = i\}$. Let G_t^i denote the graph $G[Y_t^i \cup Z_t^i] - E(G[Z_t^i])$.

▶ **Theorem 10.** *The Matching Discovery problem in the token sliding model parameterized by both the number of tokens k and the budget b is fixed-parameter tractable.*

Proof Sketch. Let (G, S, b) be an instance of the Matching Discovery problem, where $|S| = k$. The algorithm proceeds as follows. We first guess a set $S' \subseteq S$ of tokens that will slide form their original positions. If such a guess leaves any overloaded vertices, i.e., vertices incident to more than one token, we ignore the guess and proceed to the next one. Note that this *guessing* procedure requires $\mathcal{O}(2^k)$ time in the worst case. Next, for each $r \in \{0, 1, \ldots, b\}$ and for a fixed S', we guess a partition of r over the tokens in S'. In other words, we try all possible ways of distributing the budget r over the tokens in S'.

Now, for a fixed subset S', a fixed r, and a fixed distribution D of r over the tokens of S', let r_i be the budget allocated to a token $t_i \in S'$ under D. The algorithm computes the sets $Y_{t_i}^{r_i}$ and $Z_{t_i}^{r_i}$ for each $t_i \in S'$. The next step is to produce a set of candidate target edges for each t_i, which we denote by T_i. To do so, we run a maximum matching algorithm in the graph $G_{t_i}^{r_i} - (S \setminus S')$. Roughly speaking, we show that it is enough to retain $\mathcal{O}(k^2)$ edges of the matching in $G_{t_i}^{r_i} - (S \setminus S')$; which also implies a bound on the size of T_i. Once the candidate sets have been constructed, the algorithm exhaustively checks whether it can reach a valid matching in the graph G where each token $t_i \in S'$ moves to some edge in T_i. The algorithm declares a no-instance if no valid matching is found during the exhaustive search; otherwise we have a yes-instance. ◀

Finally, using the notion of augmenting paths, we show that the algorithm of Theorem 10 is also a fixed-parameter tractable algorithm for parameter k alone by proving that k upper bounds the parameter b.

▶ **Lemma 11.** *In any yes-instance (G, S, b) of the* Matching Discovery *problem in the sliding model, b can be upper bounded by $2(k^2 + k)$, where $k = |S|$.*

Combining Theorem 10 with Lemma 11 we get the following result.

▶ **Corollary 12.** *The* Matching Discovery *problem in the token sliding model is fixed-parameter tractable with respect to parameter k.*

6 Vertex/edge cuts

Let G be a graph and s and t vertices of G. A *vertex s-t-cut* is a set of vertices $C \subseteq V(G)$ such that every s-t-path contains a vertex of C. Likewise, an *edge s-t-cut* is a set of edges $C \subseteq E(G)$ such that every s-t-path contains an edge of C. In the Vertex Cut (resp. Edge Cut) problem we are given a graph G, vertices s, t and an integer k and the goal is to compute a vertex s-t-cut (resp. edge s-t-cut) of size k.

6.1 Rainbow vertex/edge cut

Given an edge-colored graph (G, φ) with vertices $s, t \in V(G)$, the Rainbow Vertex Cut problem asks whether there exists a vertex s-t-cut $C \subseteq V(G)$ such that all vertices in C have pairwise different colors. Such a cut is called a rainbow vertex s-t-cut. Likewise, the Rainbow Edge Cut problem asks whether there exists an edge s-t-cut $C \subseteq E(G)$ such that all edges in C have pairwise different colors. The Rainbow Edge Cut problem is known to be NP-complete [3, Theorem 5.5]. We start by showing that the problem remains NP-complete on planar graphs.

▶ **Theorem 13.** *The Rainbow Edge Cut problem is NP-complete on planar graphs.*

Proof. Containment in NP is clear as Rainbow Edge Cut on general graphs is in NP. Hence we focus on the hardness proof. We present a reduction from Rainbow Matching on paths. Let (P, κ) be an instance of Rainbow Matching where P is a path on n vertices denoted by $v_1, \dots v_n$ and the edges are colored with colors from a color set $\mathcal{C}$.

We construct an instance $(G, \psi : E(G) \to \mathcal{C}')$ of Rainbow s-t-Cut as follows. The new color set is $\mathcal{C}' = \mathcal{C} \cup \{\mathsf{black}\} \cup \{c_i \mid i \leq n-2\}$, that is, $\mathcal{C}'$ uses the colors from $\mathcal{C}$ as well as $n-2$ fresh colors and the color black.

Let us describe the construction of G in detail; see Figure 2 for an illustration. In the first step, G consists of κ disjoint copies of P, which we call $P_1, \dots, P_\kappa$. Let the vertices of P_j be called $v_{j,1} \dots v_{j,n}$. Additionally, we add two fresh vertices s and t. For every $j < \kappa$, insert a set L_j of $n-1$ fresh vertices, and call them $u_{j,1}, \dots, u_{j,n-1}$. Hence, $V(G) = \bigcup_{j \leq \kappa} V(P_j) \cup \bigcup_{j < \kappa} V(L_j) \cup \{s, t\}$.

Now we connect s and t with a black edge, which enforces that this black edge must be part of every (rainbow) s-t-cut. Hence, any other black edge may not be part of any rainbow s-t-cut of G. We connect s and t to the vertices in P_j and L_j as follows. For every $j \leq \kappa$ we insert the edges $\{s, v_{j,1}\}$ and $\{v_{j,n}, t\}$, which are colored black. Likewise, for every $j < \kappa$ we insert the edges $\{s, u_{j,1}\}$ and $\{u_{j,n-1}, t\}$, which are also colored black. Finally, for every $j < \kappa$ and $i \leq n-2$, we insert the edge $\{v_{j,i}, u_{j,i}\}$ which is colored black and the edges $\{u_{j,i}, v_{j,i+1}\}$ and $\{u_{j,i}, v_{j+1,i+2}\}$, which are colored c_i. This finishes the construction.

We claim that P has a rainbow matching of size κ if and only if G admits a rainbow s-t-cut. Assume that P has a rainbow matching $M = \{e_1, \dots, e_\kappa\}$ of size κ. Without loss of generality, we assume that the edges in M are ordered with respect to the v_i, i.e., if

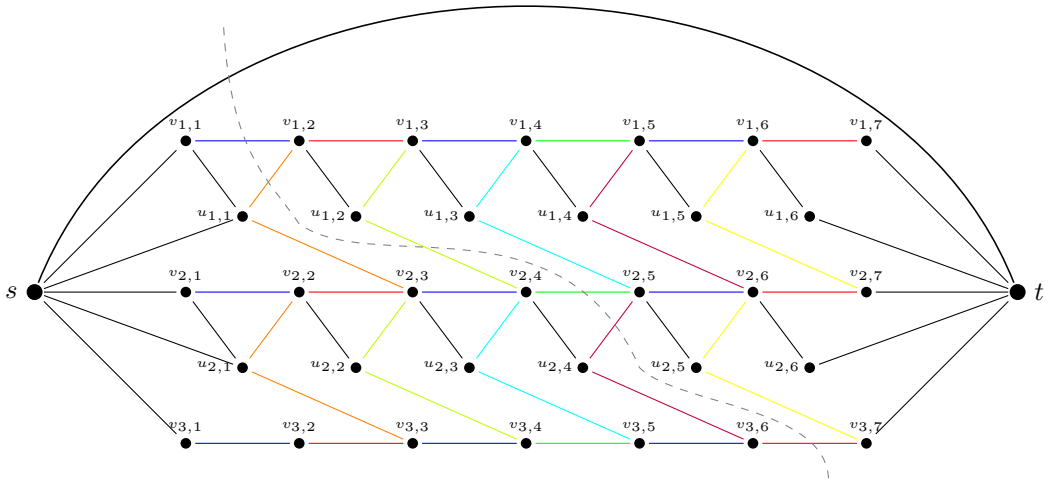


Figure 2 Illustration of the hardness reduction for Rainbow s-t-Cut on planar graphs.

$e_i = \{v_{\ell_i}, v_{\ell_{i+1}}\}$ and $i < j$, then $\ell_i < \ell_j$. We claim that the set C that consists of the black edge $\{s,t\}$, the copy of e_i in P_i, and the obvious edges connecting the P_i and L_j is a rainbow s-t-cut of G. To be precise, we have

$$C = \{s,t\} \cup \{\{v_{i,\ell_i}, v_{i,\ell_i+1}\}, \{u_{i,\ell_i}, v_{i,\ell_i+1}\} \mid i \leq \kappa\} \cup \bigcup_{i \leq \kappa} \{\{u_{i,j}, v_{i,j+2}\} \mid \ell_i < j \leq \ell_{i+1} - 2\}.$$

Observe that C is a cut by construction (see Figure 2), and that no two edges in C have the same color, as M is a rainbow matching, and for every $j < \kappa$ and $i \leq n-2$ at most one of the edges $\{v_{j,i}, u_{j_i}\}$ and $\{u_{j_i}, v_{j+1,i+2}\}$ is contained in C.

Now assume that G admits a rainbow s-t-cut. As s and t are directly connected, every rainbow s-t-cut C for G must contain $\{s,t\}$. Furthermore, by construction C contains exactly one edge from every P_j, say the edge $\{v_{j,\ell_i}, v_{j,\ell_{i+1}}\}$, as no other black edge is part of C. We claim that $M = \{\{v_{\ell_i}, v_{\ell_{i+1}}\} \mid \{v_{j,\ell_i}, v_{j,\ell_{i+1}}\} \in C \text{ for some } j \leq \kappa\}$ is a rainbow matching of P. Obviously, M is rainbow, as C is rainbow. To show that M is indeed a matching, observe that for all $\ell_i \neq \ell_j$ we have $|\ell_i - \ell_j| \geq 2$, that is, M does not contain two (copies of) consecutive edges of P. To see this, assume for the sake of contradiction that there are $i \neq j$ such that $|\ell_i - \ell_j| \leq 1$. Let $j_1 < j_2$ be such that $\{v_{j_1,\ell_i}, v_{j_1,\ell_{i+1}}\}$ and $\{v_{j_2,\ell_j}, v_{j_2,\ell_{j+1}}\}$ are contained in C. By construction, C must also contain $\{u_{j_1,\ell_i}, v_{j_1,\ell_{i+1}}\}$ and can hence not contain $\{u_{j_1,\ell_i}, v_{j_1+1,\ell_i+2}\}$, as they share the same color. This however implies that $\{v_{j_2,\ell_j}, v_{j_2,\ell_{i+1}}\}$ cannot be contained in C, a contradiction. This finishes the proof. ◀

We reuse the ideas of the proof of Theorem 13 combined with standard tricks for edge/vertex cuts, e.g., subdivision of edges, to obtain the following corollary.

▶ **Corollary 14.** *The Rainbow Vertex Cut problem is NP-complete on planar graphs.*

The following theorem is one of two main ingredients required for establishing the fixed-parameter tractability of the discovery variants of cut problems parameterized by k, the other being the aforementioned meta-theorem, which is formally stated in Section 7. In what follows, we consider weighted variants of the rainbow problems, where in addition to finding a rainbow cut of size k it is also required that it has weight at most b.

▶ **Theorem 15.** *The* Weighted Rainbow Vertex/Edge Cut *problem is fixed-parameter tractable when parameterized by k.*

Proof Sketch. We present an FPT algorithm for the Weighted Rainbow Vertex Cut problem. The tractability of Weighted Rainbow Edge Cut follows by considering the vertex cut version in the line graph of the input graph.

We use the *treewidth reduction theorem* [27, Theorem 2.15], which essentially states that for every colored and weighted graph G and integer k we can efficiently compute a graph H of low treewidth that preserves all minimal vertex s-t-cuts of size at most k. Hence, it is sufficient to find a minimum weight rainbow vertex s-t-cut of size at most k in H.

To do so, we apply the optimization version of Courcelle's Theorem as presented in [2, 8]. In our case we conclude a running time of $f(k) \cdot n^2$ where f is some computable function and $n = |V(H)| \leq |V(G)|$. ◀

6.2 Vertex/edge cut discovery

In the Vertex Cut Discovery problem in the token sliding model we are given a graph G, vertices $s, t \in V(G)$, a starting configuration $S \subseteq V(G)$ of size k and a non-negative integer b. The goal is to decide whether we can discover a vertex cut separating s and t in G (starting from S) using at most b token slides. Similarly, in the Edge Cut Discovery problem, we are given a graph G, vertices $s, t \in V(G)$, a starting configuration $S \subseteq E(G)$ of size k and a non-negative integer b. The goal is again to decide whether we can discover an edge cut separating s and t in G (starting from S) using at most b token slides. We denote an instance of Vertex Cut Discovery resp. Edge Cut Discovery by a tuple (G, s, t, S, b). We always use k to denote the size of S.

We prove that Vertex Cut Discovery in the token sliding model is NP-hard, fixed-parameter tractable with respect to parameter k and W[1]-hard with respect to parameter b. We show that these observations translate to Edge Cut Discovery as well. We start by proving hardness via a reduction from the Clique problem, which is NP-hard and its parameterized variant is W[1]-hard with respect to the solution size [11].

▶ **Theorem 16.** *The Vertex Cut Discovery problem in the sliding model is NP-hard and W[1]-hard with respect to parameter b on 2-degenerate bipartite graphs.*

Proof Sketch. We show NP-hardness by a reduction from the Clique problem. The reduction is both a polynomial time reduction as well as a parameterized reduction, showing both claimed results. Let (G, κ) be an instance of the Clique problem.

We construct the following graph H (see Figure 3 for an illustration). We add two vertices s and t in H where s has κ pendent vertices, which we collect in a set named Z. For every vertex $u \in V(G)$, we add a vertex x_u in H, which we connect with s, and for every $e \in E(G)$ we add a vertex y_e in H, which we connect with t. Let X denote the set of the x_u and Y denote the set of the y_e. For every vertex $u \in V(G)$ and for each edge $e \in E(G)$ incident with u, connect x_u and y_e via a 1-subdivided edge (that is, a path with one interval vertex). Furthermore, we add $\binom{\kappa}{2}$ disjoint paths $P_1, \ldots, P_{\binom{\kappa}{2}}$, each with a single internal vertex, connecting s and t. We denote the internal vertex of P_i by p_i. This completes the construction of H. We define the initial configuration S as $Z \cup Y$ and $b = 2\kappa + 2\binom{\kappa}{2}$. ◀

▶ **Corollary 17.** *The Edge Cut Discovery problem in the token sliding model is NP-hard and W[1]-hard with respect to parameter b on 2-degenerate bipartite graphs.*

Finally, combining Theorem 15 and Theorem 19 (Section 7), we get the following result:

▶ **Theorem 18.** *The Vertex/Edge Cut Discovery problem in the token sliding model is fixed-parameter tractable with respect to parameter k.*

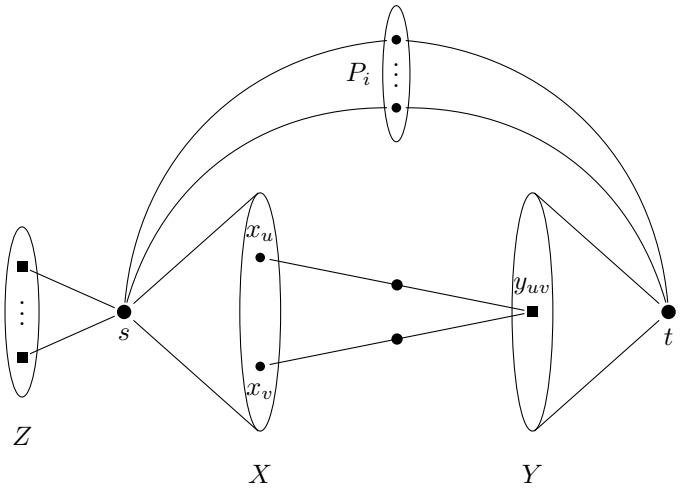


Figure 3 An illustration of the hardness reduction for the Vertex Cut Discovery problem.

7 Tractability via rainbow problems

In this section, we establish an algorithmic meta-theorem showing tractability of discovery problems in the token sliding model (when parameterized by k) via the tractability of (weighted) rainbow variants of optimization problems. We utilize the color-coding technique introduced by Alon et al. [1] along with FPT algorithms for the rainbow problems to design FPT algorithms for the discovery problems parameterized by k.

Let Π be an optimization problem and (G, k) be an instance of Π. In the Rainbow-Π problem we assume that G is either a vertex-colored graph or an edge-colored graph. The parameter k refers to the solution size to be optimized. We consider problems that seek to optimize the selection of edges or vertices, but not both. For instance, the rainbow variants for shortest paths and vertex cuts are vertex selection problems whereas for spanning trees, matchings and edge cuts we have edge selection problems. For a vertex (or edge) selection problem, the Weighted Rainbow-Π problem refers to the weighted variant where the input graph G additionally has weights on the edges (or vertices) and we seek a solution of weight at most b (amongst all solutions satisfying the cardinality constraint k).

▶ **Theorem 19.** *For an optimization problem Π, if the Weighted Rainbow-Π problem parameterized by k admits an FPT algorithm, then the Π-Discovery problem in the token sliding model parameterized by k admits an FPT algorithm.*

Proof Sketch. Without loss of generality, assume that the problem Π is an edge selection problem. Let $(G, S \subseteq E(G), b)$ be an instance of the Π-Discovery problem. Let $\mathcal{C}$ be a palette of k colors, and $\pi : \mathcal{C} \to S$ be a bijection. We color the edges $E(G) \setminus S$ uniformly at random using $\mathcal{C}$, yielding an edge coloring $\varphi : E(G) \to \mathcal{C}$. Now we define a weight function $w : E(G) \to \mathbb{R}_+$ such that for each $e \in E(G)$ we have $w(e) = \text{dist}(e, \pi(\varphi(e)))$. Intuitively, the weight function denotes the cost of moving a token from the initial configuration to an edge with the same color. Observe that the weights of the edges in the initial configuration are zero and they are colored using piece-wise distinct colors. Now we have an edge-colored graph (G, φ). We claim that if (G, φ, k, b) is a yes-instance of the Weighted Rainbow-Π problem, then (G, S, b) is a yes-instance of the Π-Discovery problem in the token sliding model. Intuitively speaking, the color-coding step allows us to transforms the discovery problem into a weighted rainbow problem.

The rest of the proof consists of showing that if (G, S, b) is a yes-instance of the Π-Discovery problem, then the probability that (G, φ, k, b) is a yes-instance of the Weighted Rainbow-Π problem is at least e^{-k}. We then derandomize the algorithm using the technique introduced by Alon et al. [1]. Instead of a random coloring φ, we construct an $(m-k, k)$-perfect hash family $\mathcal{F}$ such that for every $X \subseteq E(G) \setminus S$ with $|X| = k$, there is a function $f \in F$ such that every element of X is mapped to a different element in $\mathcal{C}$ (color palette as a k-sized set). The family $\mathcal{F}$ can be constructed deterministically in time $O(2^{O(k)} \log m)$ [1]. ◀

8 The jumping and addition/removal models

We now turn to the *token jumping* and *unrestricted token addition/removal* models. Recall that in these models we are no longer restricted to sliding tokens along edges.

8.1 Token jumping

We first define the *red-blue* variant of a vertex (resp. edge) selection problem and show that it is always at least as hard as the solution discovery variant in the token jumping model.

Let Π be an arbitrary vertex (resp. edge) selection problem. An instance of the Red-Blue-Π problem consists of a graph G whose vertices (resp. edges) are either colored red or blue, as well as two non-negative integers k and b. If Π is the decision variant of a minimization/maximization problem, the goal is to decide whether there exists a solution $X \subseteq V(G)$ (resp. $X \subseteq E(G)$) of Π of size k such that the number of blue elements in X is at most b. We denote an instance of Red-Blue-Π by (G, k, b).

▶ **Corollary 20.** *Let Π be an arbitrary vertex (resp. edge) selection problem. Then the following results hold in the token jumping model:*

1. *If Red-Blue-Π is in P, then so is Π-Discovery.*
2. *If Red-Blue-Π is in FPT with respect to k or b, then so is Π-Discovery.*
3. *If Π-Discovery is NP-hard, then so is Red-Blue-Π.*
4. *If Π-Discovery is W[1]-hard with respect to k or b, then so is Red-Blue-Π.*

We remark that the other direction does not trivially hold, i. e., we can in general not consider an instance of Red-Blue-Π as an instance of Π-Discovery, as the number of red vertices/edges might exceed the bound k on the solution size. Nevertheless, we show that Red-Blue Spanning Tree, Red-Blue Shortest Path and Red-Blue Matching are in P, implying that their discovery variants in the token jumping model are in P as well, whereas Vertex/Edge Cut Discovery is NP-hard in the token jumping model, implying that Red-Blue Vertex/Edge Cut Discovery is NP-hard as well.

▶ **Proposition 21.** *Red-Blue Spanning Tree, Red-Blue Shortest Path, and Red-Blue Matching can be solved in polynomial time. Hence, Spanning Tree Discovery, Shortest Path Discovery, and Matching Discovery in the token jumping model can be solved in polynomial time.*

▶ **Proposition 22.** *The Vertex/Edge Cut Discovery problem in the token jumping model is NP-complete. Hence, the Red-Blue Vertex/Edge Cut problem is NP-complete.*

▶ **Theorem 23.** *The Weighted Red-Blue Vertex/Edge Cut problem is fixed-parameter tractable when parameterized by k.*

8.2 Unrestricted token addition/removal

Obviously, every spanning tree of a connected graph G has size $|V(G)| - 1$ by definition. Hence, the Spanning Tree Discovery problem in the unrestricted token addition/removal model can easily be reduced to the token jumping model by halving the budget b (rounded down). To see this, note that any solution to the Spanning Tree Discovery problem in the unrestricted token addition/removal model adds and removes the same number of tokens. Furthermore, the order in which we add or remove tokens from the initial configuration S does not matter. Thus, we assume that the first modification step to S is a token removal, followed by a token addition, followed by removals/additions in alternating order. This can easily be simulated by jumps.

Similarly, the length of a shortest s-t-path in a graph G is a fixed number for fixed s and t. Hence, the same argument as above boils the Shortest Path Discovery problem in the unrestricted token addition/removal model down to the Shortest Path Discovery problem in the token jumping model. When it comes to Matching Discovery and Vertex/Edge Cut Discovery, the unrestricted token addition/removal model allows for two natural variations on the problems. On one hand, and in tune with the rest of the paper, if we impose solutions of size exactly $|S|$ then it is not hard to see that the addition/removal model becomes again equivalent to the jumping model.

▶ **Corollary 24.** *The Spanning Tree Discovery, Shortest Path Discovery, Matching Discovery, and Vertex/Edge Cut Discovery problems in the unrestricted token addition/removal model can be solved in polynomial time.*

Alternatively, if we drop the constraints on solution size and allow solutions of any size then the problems become considerably different. In particular, the Matching Discovery problem becomes equivalent to asking for a smallest subset of S whose removal (from S) results in a matching. Similarly, the Vertex/Edge Cut Discovery problem becomes equivalent to the problem of computing a smallest subset of vertices whose addition to S results in a cut between s and t. We show that, even in the relaxed setting, both problems remain polynomial-time solvable.

▶ **Proposition 25.** *The (Relaxed) Matching Discovery problem in the unrestricted token addition/removal model can be solved in polynomial time.*

▶ **Proposition 26.** *The (Relaxed) Vertex/Edge Cut Discovery problem can be solved in the unrestricted token addition/removal model in polynomial time.*

Conclusion and future work

We contribute to the new framework of solution discovery via reconfiguration with complexity theoretic and algorithmic results for solution discovery variants of *polynomial-time solvable* problems. While we can employ the efficient machinery of Weighted Matroid Intersection for the Spanning Tree Discovery problem, all other problems under consideration are shown to be NP-complete, namely, Shortest Path Discovery, Matching Discovery, and Vertex/Edge Cut Discovery. For the latter problems, we provide a full classification of tractability with respect to the parameters solution size k and transformation budget b.

We expect further research on this new model capturing the dynamics of real-world situations as well as constraints on the adaptability of solutions. It seems particularly interesting to investigate directed and/or weighted versions of the studied problems. Notice that all base problems that we studied remain polynomial-time solvable in the presence of

edge weights and for cost functions summing the total weight of the solution (spanning tree, path, matching, cut). Interestingly, our result on the Spanning Tree Discovery problem can be generalized to the weighted setting with a clever adaptation of the reduction. For other problems the hardness results clearly hold, but we leave the existence of FPT algorithms for future work. Another challenge is the design of efficient algorithms that can compute approximate solutions with respect to the solution size/value or with respect to the allowed transformation budget. We note, without proof, that the hardness reduction for the Vertex Cut Discovery problem (Theorem 16) can be adjusted to give a $n^{1-\varepsilon}$-inapproximability of the optimal transformation budget.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

2 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

3 Xuqing Bai, Renying Chang, and Xueliang Li. More on rainbow disconnection in graphs. *Discussiones Mathematicae Graph Theory*, 42:1185–1204, 2020.

4 Paul Bonsma. The complexity of rerouting shortest paths. *Theoretical computer science*, 510:1–12, 2013.

5 Nicolas Bousquet, Amer E. Mouawad, Naomi Nishimura, and Sebastian Siebertz. A survey on the parameterized complexity of the independent set and (connected) dominating set reconfiguration problems. *arXiv preprint*, 2022. arXiv:2204.10526.

6 Hajo Broersma and Xueliang Li. Spanning trees with many or few colors in edge-colored graphs. *Discuss. Math. Graph Theory*, 17(2):259–269, 1997.

7 Lily Chen, Xueliang Li, and Yongtang Shi. The complexity of determining the rainbow vertex-connection of a graph. *Theoretical Computer Science*, 412(35):4531–4535, 2011.

8 Bruno Courcelle and Mohamed Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109(1-2):49–82, 1993.

9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

10 Marzio De Biasi and Juho Lauri. On the complexity of restoring corrupted colorings. *Journal of Combinatorial Optimization*, 37(4):1150–1169, 2019.

11 Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, 141(1-2):109–131, 1995.

12 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

13 Michael R. Fellows, Mario Grobler, Nicole Megow, Amer E. Mouawad, Vijayaragunathan Ramamoorthi, Frances A. Rosamond, Daniel Schmand, and Sebastian Siebertz. On solution discovery via reconfiguration. In *ECAI 2023 - 26th European Conference on Artificial Intelligence*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, pages 700–707. IOS Press, 2023.

14 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

15 Michael R. Garey, David S. Johnson, and R Endre Tarjan. The planar hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing*, 5(4):704–714, 1976.

16 Valentin Garnero, Konstanty Junosza-Szaniawski, Mathieu Liedloff, Pedro Montealegre, and Paweł Rzazewski. Fixing improper colorings of graphs. *Theoretical Computer Science*, 711:66–78, 2018.

17 Guilherme Gomes, Sérgio H. Nogueira, and Vinicius F. dos Santos. Some results on vertex separator reconfiguration. *arXiv preprint*, 2020. arXiv:2004.10873.

18 Guilherme C. M. Gomes, Clément Legrand-Duchesne, Reem Mahmoud, Amer E. Mouawad, Yoshio Okamoto, Vinícius Fernandes dos Santos, and Tom C. van der Zanden. Minimum separator reconfiguration. In *18th International Symposium on Parameterized and Exact Computation, IPEC 2023*, volume 285 of *LIPIcs*, pages 9:1–9:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

19 Sushmita Gupta, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi. Parameterized algorithms and kernels for rainbow matching. *Algorithmica*, 81(4):1684–1698, 2019.

20 Jan van den Heuvel. The complexity of change. *Surveys in Combinatorics*, 409(2013):127–160, 2013.

21 Takehiro Ito, Erik D. Demaine, Nicholas J.A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12-14):1054–1065, 2011.

22 Takehiro Ito, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, and Yoshio Okamoto. Shortest reconfiguration of perfect matchings via alternating cycles. *SIAM Journal on Discrete Mathematics*, 36(2):1102–1123, 2022.

23 Marcin Kamiński, Paul Medvedev, and Martin Milanič. Shortest paths between shortest paths. *Theoretical Computer Science*, 412(39):5205–5210, 2011.

24 Marcin Kaminski, Paul Medvedev, and Martin Milanic. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012.

25 Van Bang Le and Florian Pfender. Complexity results for rainbow matchings. *Theoretical Computer Science*, 524:27–33, 2014. doi:10.1016/J.TCS.2013.12.013.

26 Nicolas El Maalouly and Raphael Steiner. Exact matching in graphs of bounded independence number. In *47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022*, volume 241 of *LIPIcs*, pages 46:1–46:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

27 Dániel Marx, Barry O'Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Transactions on Algorithms (TALG)*, 9(4):1–35, 2013.

28 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018.

29 Christos Nomikos, Aris Pagourtzis, and Stathis Zachos. Randomized and approximation algorithms for blue-red matching. In *32nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2022*, pages 715–725. Springer, 2007.

30 Christos H. Papadimitriou and Mihalis Yannakakis. The complexity of restricted spanning tree problems. *Journal of the ACM (JACM)*, 29(2):285–309, 1982.

31 Kei Uchizawa, Takanori Aoki, Takehiro Ito, Akira Suzuki, and Xiao Zhou. On the rainbow connectivity of graphs: complexity and fpt algorithms. *Algorithmica*, 67:161–179, 2013.

Towards Tight Bounds for the Graph Homomorphism Problem Parameterized by Cutwidth via Asymptotic Matrix Parameters

Carla Groenland
Delft Institute of Applied Mathematics, The Netherlands

Isja Mannens
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Jesper Nederlof
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Marta Piecyk
Warsaw University of Technology, Poland

Paweł Rzążewski
Warsaw University of Technology, Poland
University of Warsaw, Poland

Abstract

A homomorphism from a graph G to a graph H is an edge-preserving mapping from $V(G)$ to $V(H)$. In the graph homomorphism problem, denoted by $\textsc{Hom}(H)$, the graph H is fixed and we need to determine if there exists a homomorphism from an instance graph G to H. We study the complexity of the problem parameterized by the *cutwidth* of G, i.e., we assume that G is given along with a linear ordering $v_1, \ldots, v_n$ of $V(G)$ such that, for each $i \in \{1, \ldots, n-1\}$, the number of edges with one endpoint in $\{v_1, \ldots, v_i\}$ and the other in $\{v_{i+1}, \ldots, v_n\}$ is at most k.

We aim, for each H, for algorithms for $\textsc{Hom}(H)$ running in time $c_H^k n^{\mathcal{O}(1)}$ and matching lower bounds that exclude $c_H^{k \cdot o(1)} n^{\mathcal{O}(1)}$ or $c_H^{k(1-\Omega(1))} n^{\mathcal{O}(1)}$ time algorithms under the (Strong) Exponential Time Hypothesis. In the paper we introduce a new parameter that we call $\mathrm{mimsup}(H)$. Our main contribution is strong evidence of a close connection between c_H and $\mathrm{mimsup}(H)$:

- an information-theoretic argument that the number of states needed in a natural dynamic programming algorithm is at most $\mathrm{mimsup}(H)^k$,
- lower bounds that show that for almost all graphs H indeed we have $c_H \geq \mathrm{mimsup}(H)$, assuming the (Strong) Exponential-Time Hypothesis, and
- an algorithm with running time $\exp(\mathcal{O}(\mathrm{mimsup}(H) \cdot k \log k)) n^{\mathcal{O}(1)}$.

In the last result we do not need to assume that H is a fixed graph. Thus, as a consequence, we obtain that the problem of deciding whether G admits a homomorphism to H is fixed-parameter tractable, when parameterized by cutwidth of G and $\mathrm{mimsup}(H)$.

The parameter $\mathrm{mimsup}(H)$ can be thought of as the p-th root of the maximum induced matching number in the graph obtained by multiplying p copies of H via a certain graph product, where p tends to infinity. It can also be defined as an *asymptotic rank parameter* of the adjacency matrix of H. Such parameters play a central role in, among others, algebraic complexity theory and additive combinatorics. Our results tightly link the parameterized complexity of a problem to such an asymptotic matrix parameter for the first time.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases graph homomorphism, cutwidth, asymptotic matrix parameters

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.77

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2312.03859 [31]

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 77; pp. 77:1–77:21

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding This work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 853234 for CG, IM, JN and grant agreement No. 948057 for PRz). MP was funded by Polish National Science Centre, grant no. 2022/45/N/ST6/00237.

Acknowledgements The authors are grateful to Koblich for enlightening discussions about communication complexity.

1 Introduction

The study of the fine-grained complexity of NP-hard problems parameterized by width parameters has recently received an explosive amount of attention. In this study one aims to determine, for a given computational problem, a function f such that (1) the problem can be solved in $f(k)n^{\mathcal{O}(1)}$ time on given instances formed by an n-vertex graph along with an appropriate decomposition with width k, and (2) any improvement to $f(k)^{o(1)}n^{\mathcal{O}(1)}$ or even $f(k)^{1-\Omega(1)}n^{\mathcal{O}(1)}$ time would violate a standard hypothesis, typically being respectively the Exponential Time Hypothesis (ETH) and the Strong Exponential Time Hypothesis (SETH). Characterizing this complexity tightly often gives a deep insight in the combinatorial structure of the problem at hand, in particular about the relation that indicates when two "subsolutions" (for some definition of "subsolutions") combine into a global solution. An example where such insights had major consequences is Hamiltonian Cycle and the Traveling Salesperson Problem [4, 17, 50].

In contrast to the study of such fine-grained complexity, on the other side of the spectrum, a celebrated *meta-theorem* by Courcelle [13] shows that every graph property definable in the *monadic second-order logic* can be decided in time $f(k) \cdot n$ on n-vertex graphs given with a tree decomposition of width k. While this is extremely general, it is not precise at all in the sense that the functions $f(k)$ given by Courcelle's theorem are typically doubly-exponential or more, while more tailored algorithms with single-exponential functions exist. This begs the question: Could there be such a meta-theorem that gives a more fine-grained upper bound akin to the ones sought after above? Unfortunately, such a fine-grained meta-theorem still seems out of reach, and many recent works apply some highly non-trivial problem-specific insights to actually get the combination of tight algorithms and lower bounds [4, 5, 15, 16, 37, 38, 43, 46, 57, 60].

An intermediate step towards more general results such as Courcelle's theorem is to consider general problems that capture many natural well-studied problems as special cases. Such a step was already taken for certain locally checkable vertex subset problems, which capture natural problems including Independent Set and Dominating Set [25]. A particularly rich and elegant family of such problems can be defined via *graph homomorphisms*. A homomorphism from G to H is a mapping $\varphi : V(G) \to V(H)$ such that for every $uv \in E(G)$ we have $\varphi(u)\varphi(v) \in E(H)$. If H is the complete graph on k vertices, such mappings φ are exactly proper k-colorings, and this is why these mappings are often referred to as a *H-colorings* of G. For a fixed graph H, by Hom(H) we denote the computational problem in which one needs to determine whether there is a homomorphism to H from an input graph G. The complexity dichotomy for Hom(H) was provided by Hell and Nešetřil [34]: Hom(H) is polynomial-time solvable if H is bipartite, and NP-hard otherwise. So the cases relevant to our work are when H is non-bipartite.

There has been impressive work on the complexity of Hom(H) in various settings [7, 9, 11, 12, 14, 19, 20, 29, 58]. From the fine-grained perspective, a lot of attention was put in the parameterization by treewidth of the instance graph [21, 23, 26, 53, 54]. In particular,

for Hom(H) and some of its close relatives we exactly understand the fastest possible (up to the SETH) running time of algorithms parameterized by treewidth. Perhaps even more interestingly, the techniques developed in this line of research led to a deep understanding of combinatorial properties of Hom(H) and its variants, and the results obtained on the way can be used far beyond the bounded-treewidth case.

Typically, the lower bounds for (a variant of) Hom(H) are shown by a reduction from (a variant of) q-Coloring, where the choice of q depends on H. In particular, Marx, Lokshtanov, and Saurabh [44] showed that for any $q \geq 3$, the q-Coloring problem on every instance G cannot be solved in time $(q-\varepsilon)^{\mathrm{tw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$ for any $\varepsilon > 0$, unless the SETH fails (here $\mathrm{tw}(G)$ is the treewidth of G). Similar lower bounds for q-Coloring are also known for other parameters, like *cliquewidth* [40], *feedback vertex set number* [44], *vertex cover number* [35], or *component-order-connectivity* [23]. A common element in all these results is that the constant in the base of the exponential factor in the complexity bound is an increasing function of the number q of colors.

However, it appears that this is not the case for all natural width parameters. For a linear ordering $v_1, v_2, \ldots, v_n$ of vertices of a graph G, its *width* is the maximum number of edges between the sets $\{v_1, \ldots, v_i\}$ and $\{v_{i+1}, \ldots, v_n\}$, over all $i \in \{1, \ldots, n-1\}$. The *cutwidth* of G, denoted by $\mathrm{ctw}(G)$, is the minimum width of a linear ordering of $V(G)$.

In stark contrast to the results listed above, Jansen and Nederlof [36] showed that for every q, the q-Coloring problem on instances G given with a linear ordering of width k can be solved in randomized[1] time $2^k \cdot |V(G)|^{\mathcal{O}(1)}$. In particular, the base of the exponential factor does not depend on q.

This phenomenon appears to be very fragile, e.g., it no longer occurs for the *counting* variant of q-Coloring [32]. In the context of the discussion above, it is very natural to ask about the situation for Hom(H), i.e., whether the natural dynamic programming approach that works in time $|V(H)|^k \cdot |V(G)|^{\mathcal{O}(1)}$ can be improved. In particular, whether there exists an absolute constant c_H, such that for every graph H, the Hom(H) problem on n-vertex instances of cutwidth k can be solved in time $c_H^k \cdot n^{\mathcal{O}(1)}$. This question was answered in the negative by Piecyk and Rzążewski [56], who showed that the base c_H of the exponential factor in the complexity bound (seen as a function of H) grows to infinity even if we restrict ourselves to cycles. More specifically, they show that c_H is lower-bounded by the number of edges in a maximum *induced matching*[2] in H, multiplied by 2.

Note that a maximum induced matching of a clique has only one edge, so this lower bound matches the running time of the randomized algorithm for q-Coloring [36].

However, Piecyk and Rzążewski [56] failed to provide an algorithm matching this lower bound, even functionally, i.e., an algorithm with running time $f(p,k) \cdot n^{\mathcal{O}(1)}$, where f is any function of the size p of a maximum matching in H and the cutwidth k of the instance. Thus, while the size of a maximum induced matching in H certainly plays some important role in the complexity of Hom(H) parameterized by the cutwidth, it is far from clear whether it indeed determines the base of the exponential factor. The discussion above leads to the following.

▶ **Open Problem 1.** *Describe, for any fixed non-bipartite graph H, a constant c_H such that:*

1. *There is an algorithm that, for all $k, n \in \mathbb{N}$, given an n-vertex graph G with linear ordering of width k, solves Hom(H) in time $c_H^k \cdot n^{\mathcal{O}(1)}$, and*
2. *Assuming the SETH, for any $\varepsilon > 0$, there is no algorithm that, for all $k, n \in \mathbb{N}$, given an n-vertex graph G with linear ordering of width k, solves Hom(H) in time $(c_H - \varepsilon)^k \cdot n^{\mathcal{O}(1)}$.*

[1] A slightly slower deterministic algorithm was also given.

[2] An *induced matching* of a graph is a set M of edges such that the graph induced by the endpoints of M is a matching.

Recall that when H is bipartite, $\textsc{Hom}(H)$ is already known to be solvable in polynomial time. Therefore, we restrict ourselves to non-bipartite graphs.

Moreover, for each graph H we have $c_H \leq |V(H)|$, as a straightforward dynamic programming algorithm works in time $|V(H)|^k \cdot n^{\mathcal{O}(1)}$.

Our contribution. We make significant progress towards Open Problem 1. In particular, for each non-bipartite graph H we define a constant c_H which we conjecture to have the desired properties. We prove, for almost all graphs, that $\textsc{Hom}(H)$ in n-vertex instances given with a linear ordering of width k cannot be solved in time $(c_H - \varepsilon)^k \cdot n^{\mathcal{O}(1)}$ for any $\varepsilon > 0$, assuming the SETH. Moreover, we give a dynamic programming approach of which we show the table sizes can be compressed to $c_H^k \cdot n^{\mathcal{O}(1)}$ (see the paragraph on *representative sets* below for more details). This can be interpreted as an upper bound, for each $i \in \{1, \ldots, n-1\}$ on the amount of *information* of the graph $G[\{v_1, \ldots, v_i\}]$ needed to decide $\textsc{Hom}(H)$ based on *only* $G - G[\{v_1, \ldots, v_i\}]$. Unfortunately, this is an existential result and we do not yet know how to efficiently perform this compression. We give partial progress towards such a computation, yielding an algorithm with running time $\exp(2c_H k \log k) n^{\mathcal{O}(1)}$.

For a 0/1-matrix A, we define $\operatorname{mim}(A)$ as the largest r for which A has an $r \times r$ permutation submatrix.[3] The aforementioned work [56] shows that c_H needs to be at least $\operatorname{mim}(A_H)$, if A_H is the adjacency matrix of H. However, one of our main insights is that, as the cutwidth k increases, the accurate parameter for measuring the aforementioned amount of needed information on $G[\{v_1, \ldots, v_i\}]$ is actually $\operatorname{mim}(A_H^{\otimes k})$, where $A_H^{\otimes k}$ denotes the result of taking the Kronecker product of k copies of A_H. Specifically, we introduce a new asymptotic rank parameter, called *mimsup*, defined by

$$\operatorname{mimsup}(A) = \limsup_{k \to \infty} \operatorname{mim}(A^{\otimes k})^{1/k}.$$

For a graph H, we define $\operatorname{mimsup}(H)$ to be $\operatorname{mimsup}(A_H)$, where A_H is the adjacency matrix of H. We remark that $\operatorname{mimsup}(H)$ can be also defined in a purely graph-theoretic way, in terms of the size of a maximum matching in a certain graph product. See Section 2 for more thorough definitions and details. We prove the results above for c_H equal to $\operatorname{mimsup}(H)$ (modulo some standard preprocessing of H).

The parameter becomes especially clean and elegant if H is a *projective core*; such graphs play a prominent role in the study of graph homomorphisms [29, 42, 54]. Their definition is somewhat complicated, we refer the interested reader to the full version of the paper (in Appendix). Let us just mention that this class captures *almost all graphs* [33, 45]. Formally, let p_n denote the probability that an n-vertex graph, chosen uniformly at random, is a non-bipartite projective core. Then p_n tends to 1 as n tends to infinity. Furthermore, up to some conjectures from algebraic graph theory from the early 2000s [41, 42], every graph H that cannot be simplified by the above-mentioned preprocessing is actually a projective core. We refer the interested reader to [54] for more information.

Going back to our setting, if H is a non-bipartite projective core, then we simply have $c_H = \operatorname{mimsup}(H)$. The first evidence that c_H is indeed the "right" choice of the parameter is the following lower bound[4].

[3] It is easily seen that $\operatorname{mim}(A)$ equals the maximum size of an induced matching in the bipartite graph that has A as biadjacency matrix. If A is symmetric, it is the adjacency matrix of a graph H and $\operatorname{mim}(A)$ equals twice the maximum size of an induced matching in H.

[4] Full proofs of statements marked with (♠) can be found in the full version of the paper [31]

▶ **Theorem 1** (♠).
1. *There exists $\delta > 0$, such that for every non-bipartite projective core H, there is no algorithm solving every instance G of $\textsc{Hom}(H)$ in time* $\operatorname{mimsup}(H)^{\delta \cdot \operatorname{ctw}(G)} \cdot n^{\mathcal{O}(1)}$*, unless the ETH fails.*
2. *Let H be a connected non-bipartite projective core. There is no algorithm solving every instance G of $\textsc{Hom}(H)$ in time* $(\operatorname{mimsup}(H) - \varepsilon)^{\operatorname{ctw}(G)} \cdot |V(G)|^{\mathcal{O}(1)}$ *for any $\varepsilon > 0$, unless the SETH fails.*

We next elaborate on the mentioned dynamic programming approach along with the table size compression via which we aim to match these lower bounds.

Representative Sets. A crucial technique in our arguments is that of *representative sets*. This is a method that allows us to considerably speed up dynamic programming algorithms by sparsifying the associated tables. Specifically, dynamic programming algorithms generally define a space of possible partial solutions $\mathcal{S}$, and a dynamic programming table stores a subset $\mathcal{A}$ of partial solutions that are valid in the given instance. A binary *compatibility matrix* M with rows and columns indexed by $\mathcal{S}$ indicates whether two partial solutions combine into a global solution. Generally speaking, a *representative set* of a set $\mathcal{A} \subseteq \mathcal{S}$ is a subset $\mathcal{A}'$ such that for each $j \in \mathcal{S}$ we have that there exists $i \in \mathcal{A}$ with $M[i,j] = 1$ if and only if there exists $i' \in \mathcal{A}'$ with $M[i',j] = 1$; see Section 3.1 for details more specific to the setting of our paper.

The power of representative sets lies in that (i) by definition, in any dynamic programming algorithm we can replace the set $\mathcal{A}$ with the smaller set $\mathcal{A}'$ without missing solutions, and (ii) for many matrices M, surprisingly small representative sets are guaranteed to exist. This underlies, for example, fast algorithms for the k-Path problem [48] or connectivity problems parameterized by treewidth [4,27]. However, a serious bottleneck in these algorithm is the *computation* of such representative sets: It withholds us, for example, for getting faster algorithms for connectivity problems such as Traveling Salesperson (both parameterized by treewidth [4,17] and the classic parameterization by the number of cities [50, Theorem 3]), and polynomial kernelization algorithms for Odd Cycle Transversal [39].

This already led some researchers to design faster algorithms for finding representative sets in special settings. A natural setting that comes up, for example for connectivity problems parameterized by treewidth, is to find representative sets for sets of partial solutions with a certain *product structure*. In [28], the authors show that representative sets for such families can be found faster than known for general families.

In this paper, the computation of representative sets is also a major bottleneck; in fact, modulo the standard conjectures discussed above, it is the only issue that withholds us from solving Open Problem 1 completely. Specifically, we show:

▶ **Theorem 2** (Informal statement of Theorem 6). *In the context of the natural dynamic programming algorithm for $\textsc{Hom}(H)$ parameterized by cutwidth k, there exist representative sets of size at most* $\operatorname{mimsup}(H)^k$.

Thus, by the definition of representative sets, any algorithm that computes these representative sets fast enough would imply a $\operatorname{mimsup}(H)^{\operatorname{ctw}(G)} n^{\mathcal{O}(1)}$ time algorithm for $\textsc{Hom}(H)$ and thus solve Open Problem 1. We view this as strong evidence that our lower bounds cannot be improved. Indeed, state-of-the-art hardness reduction techniques (like [44]) for problems parameterized by width parameters encode assignments to decision variables as states of dynamic programming tables and gradually check constraints on global consistency of these assignments throughout the graph. Our proof of existence of small representative

0	**0**	0	**0**	**1**	1	0	1	1
0	**0**	0	**1**	**0**	1	1	0	1
0	0	0	1	1	0	1	1	0
0	**1**	1	**0**	**0**	0	0	1	1
1	**0**	1	**0**	**0**	0	1	0	1
1	1	0	0	0	0	1	1	0
0	1	1	0	1	1	0	0	0
1	0	1	1	0	1	0	0	0
1	1	0	1	1	0	0	0	0

Figure 1 Illustration of a maximum induced matching (or equivalently, induced permutation submatrix) of size $\mathrm{mim}(A_H^{\otimes 2})$ shown in red, where $A_H = \left(\begin{smallmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{smallmatrix}\right)$ and $H = K_3$. More generally, the proof from [36] for determining the chromatic number of a graph shows that whenever H is a complete graph, $\mathrm{mim}(A_H^{\otimes k}) = 2^k$ and thus $\mathrm{mimsup}(H) = 2$.

sets means the number of assignments that need to be considered in order to find a global solution is also small, which means that this kind of approach to design lower bounds hits a natural barrier at our lower bound.

Coping Algorithmically with the Mimsup Parameter. Matrix or graph parameters that are defined in terms of large powers are sometimes called *asymptotic rank parameters*, and they are notoriously hard to compute. For example, the value of the asymptotic rank of the matrix multiplication tensor [8] (also known as ω) or the Shannon capacity of the cycle on 7 vertices [30] remain elusive. Unfortunately, $\mathrm{mimsup}(H)$ seems no exception. Similarly to the Shannon capacity [1, Question 6], it is even not clear whether its computation is decidable. For $\mathrm{mimsup}(H)$, even for simple graphs such as $H = K_q$, determining its value is non-trivial as well. As an illustration we depict the maximum induced matching in the second Kronecker power of the adjacency matrix of K_3 in Figure 1. One of the main insights of the $2^{\mathrm{ctw}(G)}n^{\mathcal{O}(1)}$ time algorithm for the chromatic number from [36] shows that $\mathrm{mimsup}(K_q)$ in fact equals 2.

Even when the existence of small representative sets is guaranteed because $\mathrm{mimsup}(H)$ is small, it is still challenging to find them quickly. Since mimsup is about products of graphs, one may expect that this product structure can be used algorithmically. Indeed, product structure has been exploited to compute representative sets in previous work [28], but there the family that needs to be represented has some (Cartesian) product structure. In our setting, this is not guaranteed and it is much less obvious how to proceed.

Nevertheless we show that, with some loss in precision (and hence, running time), we can work with graphs with small mimsup, by approximating it with another (easier to compute) value that we call the maximum *half induced matching* number $\mathrm{him}(H)$. For a matrix A, we define $\mathrm{him}(A)$ as the largest r for which A has an $r \times r$ triangular submatrix with ones on its diagonal.[5] For a graph H, we define $\mathrm{him}(H)$ to be $\mathrm{him}(A_H)$, where A_H is the adjacency matrix of H. We show that $\mathrm{him}(A_H)$ approximates $\mathrm{mimsup}(A_H)$ in the following sense:

▶ **Theorem 3.** *For every non-bipartite graph H with adjacency matrix A_H and $k \in \mathbb{N}$,*

$$\mathrm{him}(A_H) \leq \mathrm{mimsup}(A_H) = \limsup_{k\to\infty} \mathrm{mim}(A_H^{\otimes k})^{1/k} \quad \text{and} \quad \mathrm{mim}(A_H^{\otimes k}) \leq k^{(\mathrm{him}(A_H)+1)k}.$$

[5] A submatrix of a matrix A is any matrix that can be obtained from A by removing and reordering any of its rows and columns.

The parameter $\mathrm{him}(A_H)$ is easily computable in time $2^{\mathcal{O}(|V(H)|)}$. While the lower bound on $\mathrm{mimsup}(A_H^{\otimes k})$ is relatively easy, the upper bound uses an argument similar to the "neighborhood chasing" argument for the upper bounds on multi-colored Ramsey numbers [22]. This argument can in fact be made algorithmic in the sense that it can be used to compute representative sets for $\textsc{Hom}(H)$ of size at most $\mathcal{O}(k^{\mathrm{him}(H)k})$ in time $\mathcal{O}(k^{2\mathrm{him}(H)k})$. Combining this result with the dynamic programming algorithm for $\textsc{Hom}(H)$ for graphs of small cutwidth yields the following.

▶ **Theorem 4.** *For any graphs G and H, where G is given with a linear ordering of width k, in time $\mathcal{O}(k^{2k\cdot\mathrm{mimsup}(H)} \cdot |V(H)|^4|V(G)|)$ one can decide whether G admits a homomorphism to H.*

Let us compare the running time in Theorem 4 with the naive approach; recall that its complexity is $|V(H)|^k \cdot |V(G)|^{\mathcal{O}(1)}$. If we treat H as a constant and k as a parameter, then the latter one is faster. However, we emphasize here that in Theorem 4 we do not assume that H is a constant, so these two algorithms are incomparable. In particular, Theorem 4 shows that the homomorphism problem, where the input consists of both G and H, is *fixed-parameter-tractable* when parameterized by the cutwidth of G and $\mathrm{mimsup}(H)$.

It should be noted that a similar notion of half-induced matching of a compatibility matrix was already introduced in previous work in the context of representative sets of the AntiFactor problem [47] parameterized by treewidth and list size. However, in that setting, the authors were only able to provide a lower bound for their problem, and they did not manage to make the connection with half-induced matchings algorithmic. Additionally, their compatibility matrix has a very specific structure: it is indexed with integers and the value of an entry only depends on the sum of the values associated with the row and column.

Comparison of Mimsup With Other Rank Parameters. One of our main conceptual contributions is the introduction of the mimsup parameter in the context of parameterized algorithms. It is actually the first asymptotic rank parameter shown to be relevant in this context. Our mimsup parameter is very similar to the *asymptotic induced matching number* studied by Arunachalam et al. [2] which was introduced for k-partite, k-uniform hypergraphs (and so, in the graph setting, only for bipartite graphs). Various asymptotic variants of rank parameters have been studied for tensors. For example, this has been done for rank parameters such as subrank, tensor rank and slice rank. However, for matrices (2-tensors) these are equal to the "standard" rank for matrices and so have no interesting asymptotic aspects.

That being said, it is only natural to compare the mimsup parameter with different related rank parameters from the literature. We will discuss this now, and provide proofs that formally support this discussion in the Appendix. The approach from [36] naturally extends to solve $\textsc{Hom}(H)$ quickly for all graphs H where the so-called *support-rank* [18, 49] of the adjacency matrix of H is small. The following sequence of inequalities holds for every matrix A:

$$\mathrm{mim}(A) \leq \mathrm{him}(A) \leq \mathrm{mimsup}(A) \leq \text{support-rank}(A) \leq \mathrm{rank}(A).$$

We believe all of the inequalities can be strict. When A is the $(r \times r)$-matrix with ones on and above the diagonal and zeros below the diagonal, then $\mathrm{mim}(A) = 1 < r = \mathrm{him}(A)$ for $r \geq 2$. This means that mim is not functionally equivalent to him nor mimsup. We use random matrices to show that him and mimsup may take very different values (see Theorem 13 for a formal statement). In the Appendix we find a connection of support-rank to a parameter called *graph dimension* (or *Prague dimension*) [51, 52]. We also show there

that him is not functionally equivalent to support-rank. This shows that our algorithm from Theorem 4 can be significantly faster than the discussed natural generalization of [36]. We leave it as an interesting open problem if mimsup is functionally equivalent to him or support-rank.

The aforementioned (well studied) Shannon capacity has a definition that is very similar to the mimsup parameter: It is defined in terms of the maximum size of an independent set (also called the independence number) in an appropriate graph product, and the size of a maximum induced matching of a graph equals the independence number of the square of its line graph. Unfortunately, because the definitions of mimsup and Shannon capacity use different graph products, the relation between the two is somewhat loose; see Appendix for details. Nevertheless, based on their similarity, one may expect that Shannon capacity shares some of its peculiarities with mimsup, such as an unpredictable behaviour of the value in graph powers [1].

2 Preliminaries

For an integer n, by $[n]$ we denote the set $\{1, 2, \ldots, n\}$ and for integers a, b we write $[a, b] = \{a, a+1, \ldots, b\}$. For a set X, by 2^X we denote the family of all subsets of X. For $i, s \in \mathbb{N}$, the multinomial coefficient

$$\binom{is}{s, \ldots, s} = \frac{(is)!}{s!, \ldots, s!} = i^{(1+o(1))is}$$

is the number of partitions of $[as]$ into a parts of size s.

For a graph G and $V' \subseteq V(G)$ (resp. $E' \subseteq E(G)$), by $G[V']$ (resp. $G[E']$) we denote the subgraph of G induced by V' (resp. E'). We say two graph parameters p and q are functionally equivalent if there are functions $f, g : \mathbb{R} \to \mathbb{R}$ such that $p(G) \leq f(q(G))$ and $q(G) \leq g(p(G))$ for all graphs G.

Homomorphisms. For graphs G and H, a *homomorphism* from G to H is a mapping $\varphi : V(G) \to V(H)$ such that for every $uv \in E(G)$ we have $\varphi(u)\varphi(v) \in E(H)$. If φ is a homomorphism from G to H, we denote it by writing $\varphi : G \to H$. If G admits a homomorphism to H, we denote is shortly by $G \to H$.

In the Hom problem we are given a pair (G, H) of graphs, and we ask whether $G \to H$. In the Hom(H) the graph H is considered to be fixed and we ask whether a graph G given as an input admits a homomorpshism to H.

We will always assume that H is a connected graph. Indeed, each component of G must map to a single component of H, so the problem can be solved component-wise.

Cutwidth. Let G be a graph and consider a linear ordering $\sigma = (v_1, \ldots, v_n)$ of its vertices. For $i \in [n-1]$, the *i-th cut* is the partition of $V(G)$ into sets $\{v_1, \ldots, v_i\}$ and $\{v_{i+1}, \ldots, v_n\}$. The *width* of such a cut is the number of edges with one endvertex in $\{v_1, \ldots, v_i\}$ and the other in $\{v_{i+1}, \ldots, v_n\}$. The *width* of σ is the maximum width of a cut of σ. Finally, the *cutwidth* of G, denoted by $\mathrm{ctw}(G)$, is the minimum width of a linear ordering of the vertices of G.

Associated bipartite graphs. In order to define the main parameters of our paper, we will use a notion of *associated bipartite graphs*. For a graph G, the graph G^* is defined as follows.

$$V(G^*) = \{u', u'' \mid u \in V(G)\},$$
$$E(G^*) = \{u'w'', u''w' \mid uw \in E(G)\}.$$

Induced matchings and half-induced matchings. A set $M \subseteq E$ of edges of a graph $H = (V, E)$ forms an *induced matching* if the edges in M are vertex disjoint and no edge in E is incident with two edges from M. We may also view this as two sequences of distinct vertices $v_1, \ldots, v_m$ and $u_1, \ldots, u_m$ where $v_i u_j \in E$ if and only if $i = j$. For a bipartite graph H, by $\mathrm{mim}(H)$ we denote the size of a maximum induced matching in H. For non-bipartite H, we define $\mathrm{mim}(H) := \mathrm{mim}(H^*)$.

A *half-induced matching* of a graph H consists of two sequences $v_1, \ldots, v_m$ and $u_1, \ldots, u_m$ of distinct vertices where $v_i u_i \in E$ for $i \in [m]$ and $u_i v_j \notin E$ if $1 \leq i < j \leq m$. For a bipartite graph H, we denote the size of the largest half-induced matching in H by $\mathrm{him}(H)$. We extend the definition to graphs H that are non-bipartite via $\mathrm{him}(H) = \mathrm{him}(H^*)$. This notion has been studied under the name *constrained matching* (a subset with a unique matching, see e.g. [10, 55, 59]), but we decided to use the name which appeared more recently in a similar setting to ours [47], since the word "constrained matching" has also been used for various other purposes in the algorithmic community.

Mim and him for matrices. Let $A \in \{0,1\}^{n \times n}$ be a matrix. Given a sequence $r \in [n]^a$ of distinct row indices and $c \in [n]^b$ of distinct columns indices, for some integers $a, b \in [n]$, we write $A[r, c]$ for the $a \times b$ matrix with entries $A[r, c]_{i,j} = A_{r_i, c_j}$ for $i \in [a]$ and $j \in [b]$. We refer to any matrix which arises in such a manner as a *submatrix after permutation* of A.

We write $\mathrm{mim}(A)$ for the maximum r for which A has the $r \times r$ identity matrix as submatrix after permutation (equivalently, the largest permutation submatrix). We write $\mathrm{him}(A)$ for the largest r for which A has an $r \times r$ triangular matrix with 1's on the diagonal as submatrix after permutation. We will also refer to such a submatrix as *half induced matching*. (A matrix is called *triangular* if either all entries below the diagonal, or all entries above the diagonal are 0.)

For bipartite graphs $H = (U, V, E)$, the *bi-adjacency matrix* B is indexed by rows from U and columns from V where $B[u, v] = 1$ if $uv \in E$ and $B[u, v] = 0$ otherwise. For a bipartite graph H, there is a one-to-one correspondence between induced matchings of H of size m and $m \times m$ permutation submatrices of the bi-adjancency matrix of H. In particular, for a bi-adjacency matrix B of H, $\mathrm{mim}(B) = \mathrm{mim}(H)$. Similarly, $\mathrm{him}(B) = \mathrm{him}(H)$.

For a non-bipartite graph G, if A_G is its adjacency matrix, then A_G is also the bi-adjacency matrix of G^*. This means that for non-bipartite H with adjacency matrix A_H, $\mathrm{mim}(H) = \mathrm{mim}(A_H)$ and $\mathrm{him}(H) = \mathrm{him}(A_H)$.

Mimsup. For a matrix A, we define

$$\mathrm{mimsup}(A) = \limsup_{k \to \infty} \mathrm{mim}(A^{\otimes k})^{1/k}.$$

Here $\otimes$ denotes the Kronecker product of the matrix. Given an $n \times m$ matrix $A = (a_{i,j})_{i \in [n], j \in [m]}$ and a matrix B, the Kronecker product is given by

$$A \otimes B = \begin{pmatrix} a_{1,1}B & a_{1,2}B & \ldots & a_{1,m}B \\ a_{2,1}B & a_{2,2}B & \ldots & a_{2,m}B \\ & \ldots & & \\ a_{n,1}B & a_{n,2}B & \ldots & a_{n,m}B \end{pmatrix}.$$

Since $\operatorname{mim}(A \otimes B) \geq \operatorname{mim}(A)\operatorname{mim}(B)$, Fekete's lemma [24] applies to show that (♠)

$$\limsup_{k\to\infty} \operatorname{mim}(A^{\otimes k})^{1/k} = \lim_{k\to\infty} \operatorname{mim}(A^{\otimes k})^{1/k} = \sup_{k\in\mathbb{N}} \operatorname{mim}(A^{\otimes k})^{1/k}.$$

For a non-bipartite graph H, with adjacency matrix A, we set

$$\operatorname{mimsup}(H) = \operatorname{mimsup}(A).$$

When H is bipartite with bi-adjancency matrix[6] B, $\operatorname{mimsup}(H) = \operatorname{mimsup}(B)$. The parameters can also be defined in purely graph theoretical terms, as we now explain.

For a bipartite graph H with bipartition classes X, Y, and for $k \in \mathbb{N}$, we define $H^{\otimes k}$ to be the graph on vertex set $X^k \cup Y^k$ where there is an edge $(x_1, \ldots, x_k)(y_1, \ldots, y_k)$ in $H^{\otimes k}$ if and only if $x_i y_i \in E(H)$ for every $i \in [k]$. With this definition of graph product $\otimes$, we define

$$\operatorname{mimsup}(H) = \begin{cases} \limsup_{k\to\infty} \operatorname{mim}(H^{\otimes k})^{1/k} & \text{if } H \text{ is bipartite,} \\ \operatorname{mimsup}(H^*) & \text{otherwise.} \end{cases}$$

The following property of mimsup is straightforward.

▶ **Observation 5.** *If H is an induced subgraph of G, then* $\operatorname{mimsup}(H) \leq \operatorname{mimsup}(G)$.

For bipartite graphs, mimsup coincides with the parameter asymptotic induced matching number studied by [2]. Although asymptotic rank parameters (e.g. asymptotic subrank, asymptotic tensor rank and asymptotic slice rank) have been widely studied for tensors, the "non-asymptotic" parameters are usually equal to the matrix rank for matrices, which has no interesting asymptotic behaviour since $\operatorname{rank}(A^{\otimes n}) = \operatorname{rank}(A)^n$. In particular, the *subrank* in some sense looks for the largest "identity subtensor", similar to our mim, but since it allows row operations to be applied (instead of merely permutations), this notion is the same as the usual rank for matrices and the same holds for the asymptotic subrank.

3 Solving Hom with representative sets

In this section we discuss how we can use representative sets to create fast algorithms for Hom. We start by giving a definition of a representative set in our setting. Intuitively we want a representative set $\mathcal{A}'$ of $\mathcal{A}$ to carry all the important information from $\mathcal{A}$, while being smaller in size. In practice, being able to find small representative sets corresponds to having to compute less entries in a dynamic programming algorithm. So this gives the following natural extremal problem: how small of a representative set are we always guaranteed to find, that is, *what is the largest size of a set which has no smaller representative set?* After giving the definition, we explain why mimsup exactly determines the answer to this question in our setting.

Finally, we give a general framework for solving Hom instances; it consists of an algorithm that takes as input some reduction algorithm R that produces small representative sets and uses it to solve Hom on input graphs G and H. In Section 4 we give examples of such reduction algorithms.

[6] Note that mimsup is invariant under row and column permutations. This means that the choice of bi-adjacency matrix does not affect the mimsup and thus mimsup on bipartite graphs is well-defined.

3.1 Definition of Representative Set

Given a 0/1 matrix M, with rows indexed by a set $\mathcal{R}$ and $\mathcal{A} \subseteq \mathcal{R}$, we are interested in knowing whether for a column c, there is a row $r \in \mathcal{A}$ with $M[r,c] = 1$. In our case,

- each row represents a coloring of the left-hand side of the cut;
- all the colorings that can be extended to the left-hand side of the (input) graph are contained in $\mathcal{A}$;
- each column represents a coloring of the right-hand side of the cut;
- $M[r,c] = 1$ if and only if the colorings represented by row r and column c are compatible.

This makes the following definition very natural. We say that a subset $\mathcal{A}' \subseteq \mathcal{A}$ *M-represents* $\mathcal{A}$, if for any column j we have that if there is a row index $i \in \mathcal{A}$ such that $M[i,j] = 1$, then there is also $i' \in \mathcal{A}'$ such that $M[i',j] = 1$. Intuitively, this means that we do not "lose any solutions" by restricting to $\mathcal{A}'$.

We will also refer to $\mathcal{A}'$ as an *M-representative set* of $\mathcal{A}$. We may omit M if it is clear from context.

We remark that representing is transitive: if $\mathcal{A}''$ represents $\mathcal{A}'$ and $\mathcal{A}'$ represents $\mathcal{A}$, then $\mathcal{A}''$ represents $\mathcal{A}$.

Suppose we aim to solve Hom for input graphs G and H, where H is non-bipartite. We will be interested in representative sets with respect to $M = A_H^{\otimes k}$ for integers k, where A_H is the adjacency matrix of H. We assume that G is given with a linear order $v_1, \ldots, v_n$ of width at most w. For an integer $i \in [n]$, then $G[\{v_1, \ldots, v_i\}]$ is the "left-hand side of the graph" with respect to the ith cut and

$$X_i := \{v \in \{v_1, \ldots, v_i\} \mid \exists v' \in \{v_{i+1}, \ldots, v_n\}, vv' \in E(G)\}.$$

is "the left-hand side of the i-th cut." Suppose there are k edges crossing the ith cut: $\{a_1, b_1\}, \ldots, \{a_k, b_k\} \in E(G)$ with $a_1, \ldots, a_k \in \{v_1, \ldots, v_i\}$ and $b_1, \ldots, b_k \in \{v_{i+1}, \ldots, v_n\}$. Let $L_i = (a_1, \ldots, a_k)$ and $R_i = (b_1, \ldots, b_k)$. Note that $\{a_1, \ldots, a_k\} = X_i$ but some elements may be repeated. A row r (seen as "index") of the matrix $M = A_H^{\otimes k}$ is a k-tuple $(r_1, \ldots, r_k) \in V(H)^k$, which corresponds to a coloring $X_i \to V(H)$ if $r_j = r_{j'}$ whenever $a_j = a_{j'}$. If similarly $c \in V(H)^k$ represents a coloring of the "right-hand side of the cut", then $M[r,c] = 1$ if and only if $r_j c_j \in E(H)$ for all $j \in [k]$, i.e. the colorings are compatible. So indeed we capture the properties informally claimed above.

Since in our setting M will be the adjacency matrix of some graph H, we may refer to H-representative sets rather than A_H-representative sets.

3.2 Connection to Mimsup

When applied to the adjacency matrix A_H of a non-bipartite graph H, the following result shows that $\text{mimsup}(H)^k$ approximates how large a set $\mathcal{A} \subseteq V(H)^k$ without smaller $A_H^{\otimes k}$-representative set can be. This easily follows from the definitions but is still an important conceptual contribution.

▶ **Theorem 6.** *Let $M \in \{0,1\}^{h \times h}$ be a matrix with rows indexed by $\mathcal{R}$.*

- *For each integer $k \in \mathbb{N}$, for any $\mathcal{A} \subseteq \mathcal{R}^k$, there is a subset $\mathcal{A}' \subseteq \mathcal{A}$ of size $\text{mimsup}(M)^k$ that $M^{\otimes k}$-represents $\mathcal{A}$.*
- *Conversely, for each $\varepsilon > 0$, for each sufficiently large k, there is a $\mathcal{A} \subseteq \mathcal{R}^k$, for which no $\mathcal{A}' \subseteq \mathcal{A}$ of size at most $(\text{mimsup}(M) - \varepsilon)^k$ can $M^{\otimes k}$-represent $\mathcal{A}$.*

Proof. Let $\mathcal{A}' \subseteq \mathcal{A}$ be of minimal size among the subsets that M-represent $\mathcal{A}$. Then no proper subset of it M-represents $\mathcal{A}$. This means that for each $a \in \mathcal{A}'$ it cannot be removed from $\mathcal{A}'$ to get a set that M-represents $\mathcal{A}$. Thus, for each $a \in \mathcal{A}'$ there is some $\mu(a) \in V(H)^k$

such that $M[a, \mu(a)] = 1$, but for every $a' \in \mathcal{A}' \setminus \{a\}$ we have that $M[a', \mu(a)] = 0$. This gives a permutation $(|\mathcal{A}'| \times |\mathcal{A}'|)$-submatrix of M. This shows that $|\mathcal{A}'| \leq \mathrm{mim}(M^{\otimes k})$. By definition of mimsup, $\mathrm{mim}(M^{\otimes k}) \leq \mathrm{mimsup}(M)^k$.

Conversely, by definition of limit, for each $\varepsilon > 0$ there is a k_0 such that $\mathrm{mim}(M^{\otimes k}) \geq (\mathrm{mimsup}(M) - \varepsilon)^k$ for all $k \geq k_0$. Let $k \geq k_0$. Let $\mathcal{A} \subseteq \mathcal{R}$ be the rows of a largest induced permutation submatrix of $M^{\otimes k}$. Then $|\mathcal{A}| = \mathrm{mim}(M^{\otimes k}) \geq (\mathrm{mimsup}(M) - \varepsilon)^k$ and none of the strict subsets of $\mathcal{A}$ can $M^{\otimes k}$-represent it. ◀

3.3 Exploiting Representative Sets in Dynamic Programming

The main idea behind the use of representative sets in an algorithmic setting is as follows. We solve the problem with a standard dynamic programming approach, where the cells are indexed by the elements of the set $\mathcal{A}$. A representative set then forms a small subset of these indices, which still carries enough information to solve the problem. By regularly applying the reduction algorithm, we can effectively run our dynamic programming algorithm on only a small subset of the cells in the table. We formalize this in the following theorem. Let us emphasize that H is not assumed to be fixed here but rather given as an input.

▶ **Theorem 7.** *Let H be a non-bipartite graph on h vertices. Let R be a reduction algorithm that, given an integer $k \geq 2$ and a subset $\mathcal{A} \subseteq V(H)^k$, outputs a set $\mathcal{A}'$ of size $\mathsf{size}(H, k)$ that $A_H^{\otimes k}$-represents $\mathcal{A}$, running in time $\mathsf{time}(|\mathcal{A}|, H, k)$. Then there exists an algorithm that, given a linear ordering of an n-vertex graph G of width w, decides whether $G \to H$ in time*

$$\mathcal{O}\Big(\big(\mathsf{size}(H, w) \cdot h + \mathsf{time}\left(\mathsf{size}(H, w) \cdot h, H, w\right)\big)n\Big).$$

Proof. Let $v_1, \ldots, v_n$ be a linear ordering of G of width k. For $i \in [n]$, by E_i we denote the set of edges that cross the i-th cut, i.e., those with one endpoint in $\{v_1, \ldots, v_i\}$ and the other in $\{v_{i+1}, \ldots, v_n\}$. For $i \in [n]$, let X_i be the set that contains all vertices from $\{v_1, \ldots, v_i\}$ incident to an edge from E_i, i.e.,

$$X_i := \{v \in \{v_1, \ldots, v_i\} \mid \exists v' \in \{v_{i+1}, \ldots, v_n\}, vv' \in E(G)\}.$$

Note that we have $|X_i| \leq |E_i| \leq w$ and $X_1 = \{v_1\}$ (since G is connected). For a mapping $c : X_i \to V(H)$, we define the table entry $T_i[c]$ as true if there exists a homomorphism $\varphi : G[\{v_1, \ldots, v_i\}] \to V(H)$, such that for all $v \in X_i$ we have $\varphi(v) = c(v)$. (In other words, the keys are given by the H-colorings of X_i and the table entry is true if there is an extension of the coloring of X_i to the left-hand side of the graph.)

This table can be easily computed in time $h^{w+1} \cdot n^{\mathcal{O}(1)}$ by the following naive dynamic programming procedure. We initiate every entry $T_i[c]$ to be false and every entry $T_1[c]$ to be true. Then, for every $i \in [2, n]$, every mapping $c' : X_{i-1} \to V(H)$, such that $T_{i-1}[c']$ is true, and every $u \in V(H)$, we check whether $c : X_{i-1} \cup \{v_i\} \to V(H)$ defined as

$$c(v) = \begin{cases} u & \text{if } v = v_i, \\ c'(v) & v \in X_{i-1}. \end{cases} \tag{1}$$

is a homomorphism from $G[X_{i-1} \cup \{v_i\}]$ to H. If so, we set $T_i[c|_{X_i}]$ to true.

We first outline why this correctly computes the table entries (that is, that at the end $T_i[c]$ is true if and only if c extends to a coloring of $G[\{v_1, \ldots, v_i\}]$) and then explain how to improve on this naive algorithm. We prove the claim by induction on i. For $i = 1$, the coloring only assigns a color to v_1 and does not need to be extended (and is automatically proper). Now suppose that the claim has been shown for $i = 1, \ldots j$ and let $\alpha : X_{j+1} \to V(H)$

be a coloring. If this extends to a coloring ϕ of $G[\{v_1, \ldots, v_{j+1}\}]$, then $T_j[c']$ is true for $c' = \phi|_{X_j}$ (by the induction hypothesis) and we could obtain α as the restriction from c from (1) with $u = \phi(v_{j+1})$ and $i = j + 1$. So $T_{j+1}[\alpha]$ is true. Vice versa, if $T_{j+1}[\alpha]$ has been set to true, then there is a $c' : X_j \to V(H)$ and $u \in V(H)$ such that c (again defined as in (1)) is a homomorphism $G[X_j \cup \{v_{j+1}\}] \to H$ which restricts to α on X_{j+1}. By the induction hypothesis, there exists a proper coloring ϕ' that extends c' to $G[\{v_1, \ldots, v_j\}]$ and we extend this to a coloring ϕ of $G[\{v_1, \ldots, v_{j+1}\}]$ by setting $\phi(v_{j+1}) = u$. Then ϕ still restricts to α and all of the edge constraints have been verified by c and/or ϕ'. In particular, $G \to H$ if and only if $T_n[\emptyset]$ is true, where $\emptyset$ denotes the empty mapping ($X_n = \emptyset$).

We will speed up this naive version of the dynamic program by computing a representative table T' as follows. We first set $T'_1 = T_1$. For $i = 1, 2, \ldots, n-1$ we proceed as follows. Let $k = |E_i| \leq w$ and $M = A_H^{\otimes k}$. Let $\{a_1, b_1\}, \ldots, \{a_k, b_k\} \in E_i$ be an enumeration of the edges, with $a_j \in \{v_1, \ldots, v_i\}$ for all $j \in [k]$. For each $c : X_i \to V(H)$ such that $T'_i[c]$ is set to true, we put the k-tuple $(c(a_1), \ldots, c(a_k))$ in $\mathcal{A}_i$. When $k \geq 2$, we apply the reduction algorithm R to $\mathcal{A}_i$, resulting in a set $\mathcal{A}'_i$ of size at most $\mathsf{size}(H, k)$ that $A_H^{\otimes k}$-represents $\mathcal{A}_i$. When $k = 1$, we set $\mathcal{A}'_i = \mathcal{A}_i$. We then compute the next table entries similarly as in the previous approach. Each element of $\mathcal{A}'_i$ corresponds to a coloring $c' : X_i \to V(H)$. For $u \in V(H)$, we check whether $c : X_i \cup \{v_{i+1}\} \to V(H)$ with $c(v_{i+1}) = u$ and $c|_{X_i} = c'$ is a homomorphism from $G[X_i \cup \{v_{i+1}\}]$ to H. If so, we set $T'_{i+1}[c|_{X_{i+1}}]$ to true. We repeat this for all pairs (c', u).

The procedure above is repeated for $i = 1, \ldots, n-1$, after which we return $T'_n[\emptyset]$.

When $|\mathcal{A}'_i| \leq \mathsf{size}(H, k)$, we find that $|\mathcal{A}_{i+1}| \leq \mathsf{size}(H, k)h$ (for $k = |E_i| \leq w$ and $\mathsf{size}(H, k) = h$ for $k = 1$). We may assume size is a non-decreasing function on each coordinate. So the total running time is as claimed:

$$\mathcal{O}\Big(\big(\mathsf{size}(H, w) \cdot h + \mathsf{time}\left(\mathsf{size}(H, w) \cdot h, H, w\right)\big)n\Big).$$

The fact that the dynamic programming steps preserve representation follows from transitivity of representation, but let us spell out the details.

Let Y_{i+1} be the set of endpoints on the right-hand side of the $(i+1)$th cut and enumerate the edges in E_{i+1} as $\{x_1, y_1\}, \ldots, \{x_k, y_k\}$, with $x_j \in X_{i+1}$ and $y_j \in Y_{i+1}$. We will show that for every $i \in [n-1]$, if $\mathcal{A}'_i$ represents the set $\mathrm{True}_i := \{(c(x_1), \ldots, c(x_k)) \mid T_i[c] = \mathrm{True}\}$, then $\mathcal{A}_{i+1}$ represents the set True_{i+1}. The same then holds for $\mathcal{A}'_{i+1}$ since the reduction algorithm is assumed to work correctly.

We started with setting $T'_1 = T_1$, so $\mathcal{A}'_1$ indeed represents True_1.

Let us first unravel the definitions to see what we need to show. Let $i \in [n-1]$ and suppose that $c : G[X_{i+1}] \to H$ extends to a homomorphism $\phi : G[\{v_1, \ldots, v_{i+1}\}] \to H$ (i.e. $T_i[c] = \mathrm{True}$). For the definition of represents, we will then assume there is a homomorphism $d : G[Y_{i+1}] \to H$ for which $c \cup d$ respects all edges from the $(i+1)$th cut (those in E_{i+1}), i.e. this corresponds to a "one-entry in the compatibility matrix". What needs to be shown is that this "one-entry" can also be generated via a coloring coming from $\mathcal{A}_{i+1}$, that is, there is $\alpha : G[X_{i+1}] \to H$, such that $(\alpha(x_1), \ldots, \alpha(x_k)) \in \mathcal{A}_{i+1}$ and $(\alpha \cup d)|_{G[E_{i+1}]}$ is a homomorphism.

By assumption, $\phi \cup d$ respects all the edges with at least one endpoint in $\{v_1, \ldots, v_{i+1}\}$, and in particular those with one endpoint in $\{v_1, \ldots, v_i\}$. Since $\mathcal{A}'_i$ is a representative set of True_i, there must be $c' : G[X_i] \to H$ such that $(c'(x'_1), \ldots, c'(x'_{k'})) \in \mathcal{A}'_i$, for $\{x'_1, \ldots, x'_{k'}\} = X_i$, and where $c' \cup \phi|_{\{v_{i+1}\}} \cup d$ respects all the edges with at least one endpoint in $\{v_1, \ldots, v_i\}$. We set $\alpha = (c' \cup \phi|_{\{v_{i+1}\}})|_{X_{i+1}}$. Then $(\alpha(x_1), \ldots, \alpha(x_k)) \in \mathcal{A}_{i+1}$, by definition of how we obtain $\mathcal{A}_{i+1}$ from $\mathcal{A}'_i$. Moreover, $\alpha \cup d$ is a homomorphism $G[E_{i+1}] \to H$, as desired. ◀

4 Computing representative sets via half-induced matchings

In this section we present one of our main technical contributions, i.e., an algorithm to compute H-representative sets whose size is bounded in terms of $\mathrm{him}(H)$. Actually, our approach is rather general since it finds representative sets non-trivially fast for any large Kronecker power of a matrix with small him parameter.

We will show how to find a representative set that has one fewer element, by finding some element that can be safely removed. We then use this intermediate result to find our final reduction algorithm, which will result in the following lemma.

▶ **Lemma 8.** *Let $\ell \geq 1$ and $k \geq 2$ be integers. Let $A \in \{0,1\}^{h \times h}$ be a matrix with $\mathrm{him}(A) < \ell$, and let $\mathcal{A} \subseteq [h]^k$. Then we can compute $\mathcal{A}' \subseteq \mathcal{A}$ that $A^{\otimes k}$-represents $\mathcal{A}$ with $|\mathcal{A}'| \leq k^{k\ell}$ in time $\mathcal{O}(|\mathcal{A}|^2 h^2 k^2)$.*

We will combine the reduction algorithm from Lemma 8 with Theorem 7 to find the following result.

▶ **Theorem 9.** *The Hom problem on an instance (G, H), where G is given with a linear ordering of width k, can be solved in time $\mathcal{O}(k^{2k\cdot(\mathrm{him}(H)+1)} \cdot |V(H)|^4 |V(G)|)$.*

We emphasize that the algorithm does not need to know the value of $\mathrm{him}(H)$.

Proof. Let $h = |V(H)|$ and let A_H be the adjacency matrix of H. Recall that $\mathrm{him}(A_H) = \mathrm{him}(H)$ is always an integer. By Lemma 8 we have a reduction algorithm R that returns a representative set of size $\mathsf{size}(H,k) \leq k^{k\cdot(\mathrm{him}(H)+1)}$ in time $\mathsf{time}(|\mathcal{A}|, H, k) = \mathcal{O}(|\mathcal{A}|^2 h^2 k^2)$. Then

$$\mathsf{time}(\mathsf{size}(H,k)\cdot h, H, k) = \mathcal{O}\left(k^2 \cdot k^{2k\cdot(\mathrm{him}(H)+1)} \cdot h^4\right).$$

By Theorem 7 we find an algorithm that decides $\textsc{Hom}(H)$ in time

$$\mathcal{O}\left((\mathsf{size}(H,k)\cdot h + \mathsf{time}(\mathsf{size}(H,k)\cdot h, H, k))|V(G)|\right) = \mathcal{O}(k^{2k\cdot(\mathrm{him}(H)+1)} h^4 \cdot |V(G)|).$$

This completes the proof. ◀

Since $\mathrm{him}(H) \leq \mathrm{mimsup}(H)$ (see Lemma 11), we obtain Theorem 4 as a corollary from Theorem 9. Lemma 8 and Lemma 11 also imply Theorem 3.

In order to prove the lemma, we will perform a recursive algorithm for which we want to no longer treat all the coordinates symmetrically. We therefore define

$$g_k(\ell_1, \ldots, \ell_k) = \binom{\sum_i \ell_i}{\ell_1, \ldots, \ell_k}.$$

When $\ell_1 = \cdots = \ell_k = \ell$, we have $g_k(\ell, \ldots, \ell) = \binom{k\ell}{\ell,\ldots,\ell} \leq k^{k\ell}$ (= the number of partitions of $k\ell$ into k parts, which no longer need to have the same size). The lemma will follow easily from the following more complicated statement.

▶ **Lemma 10.** *Let $k \geq 2, \ell_1, \ldots, \ell_k \geq 1$ be integers. Let $A \in \{0,1\}^{h\times h}$ be a matrix and let $\mathcal{A} \subseteq [h]^k$ with $|\mathcal{A}| \geq g_k(\ell_1, \ldots, \ell_k)$. Suppose that for every $i \in [k]$, for the set of rows $\mathcal{R}_i = \{r_i \mid r \in \mathcal{A}\}$, we have $\mathrm{him}(A[\mathcal{R}_i, \cdot]) < \ell_i$. Then there exists $v \in \mathcal{A}$ such that $\mathcal{A} \setminus \{v\}$ $A^{\otimes k}$-represents $\mathcal{A}$. Moreover, v can be found in time $\mathcal{O}(\sum_{i=1}^k \ell_i \cdot |\mathcal{A}| hk)$.*

Proof. Note that $|\mathcal{A}| \geq g_k(\ell_1, \dots, \ell_k) \geq 1$ for $\ell_1, \dots, \ell_k, k \geq 1$ so $\mathcal{A}$ is non-empty.

For $i \in [k]$ and $u \in [h]$, let

$$\mathcal{A}^i_u = \{v = (v_1, \dots, v_k) \in \mathcal{A} \mid A[v_i, u] = 0\}$$

be the set of rows which cannot "represent" u in the ith coordinate. We choose $v \in \mathcal{A}$ (arbitrarily). We then iterate over $u \in [h]$ and $i \in [k]$ to find if there is (u, i) for which

- $A[v_i, u] = 1$, and
- $|\mathcal{A}^i_u| \geq g_k(\ell_1, \dots, \ell_i - 1, \dots, \ell_k)$.

This step can be performed in time $\mathcal{O}(|\mathcal{A}|hk)$.

If we cannot find such (u, i) pair for v, then we return v as the row to be removed from $\mathcal{A}$ (and the algorithm terminates).

Otherwise, we did find (u, i). If $\ell_i = 1$, then since $\mathrm{him}(A[\mathcal{R}_i, \cdot]) < \ell_i$, we know $A[\mathcal{R}_i, \cdot]$ has all zero-entries and so $A[v_i, u] = 1$ would not have been possible. This means that $\ell_i \geq 2$. We apply the same process after updating $\ell_i \leftarrow \ell_i - 1$ and $\mathcal{A} \leftarrow \mathcal{A}^i_u$. Note that $v \notin \mathcal{A}^i_u$ and $\ell_i - 1 \geq 1$. We will show that

- when v is returned, indeed $\mathcal{A} \setminus \{v\}$ $A^{\otimes k}$-represents $\mathcal{A}$, and
- when we recursively apply the algorithm, the conditions of the lemma are again satisfied, for which it remains to show that $\mathrm{him}(A[\mathcal{R}'_i, \cdot]) < \ell_i - 1$ for $\mathcal{R}'_i = \{r_i \mid r \in \mathcal{A}^i_u\}$.

Since we reduce $\sum_{i=1}^k \ell_i$ by one in each recursive call, the algorithm will terminate. Moreover, the number of recursive calls is at most $\sum_{i=1}^k \ell_i$. This shows that assuming the claims above, the time complexity is as stated.

Correctness. We first show the first claim: if the algorithm outputs v, indeed it can be removed. Note that if for some subset $\mathcal{A}' \subseteq \mathcal{A}$, it is the case that $\mathcal{A}' \setminus \{v\}$ represents $\mathcal{A}'$, then

$$\mathcal{A}' \setminus \{v\} \cup (\mathcal{A} \setminus \mathcal{A}') = \mathcal{A} \setminus \{v\}$$

will also represent $\mathcal{A}$. This means we only have to check the claims in the "base case". Suppose towards a contradiction that we wrongly outputted $v \in \mathcal{A}$, so

- there exists $u \in [h]^k$ such that $A^{\otimes k}[v, u] = 1$ yet $A^{\otimes k}[v', u] = 0$ for all $v' \in \mathcal{A} \setminus \{v\}$ (since we "wrongly" outputted v, there needs to be a reason why we could not remove it),
- for this u, for all $i \in [k]$, $|\mathcal{A}^i_{u_i}| < g_k(\ell_1, \dots, \ell_i - 1, \dots, \ell_k)$ (else the algorithm would have "recursed" instead of outputting v).

The fact that $A^{\otimes k}[v', u] = 0$ in the first condition, means that each $v' \in \mathcal{A} \setminus \{v\}$ is an element of $\mathcal{A}^i_{u_i}$ for some $i \in [k]$. In particular,

$$|\mathcal{A} \setminus \{v\}| \leq \sum_{i=1}^k |\mathcal{A}^i_{u_i}| \leq \sum_{i=1}^k g_k(\ell_1, \dots, \ell_i - 1, \dots, \ell_k) - k = g_k(\ell_1, \dots, \ell_k) - k,$$

which contradicts the assumptions of the lemma since $k \geq 2$.

We now prove the second claim: the conditions of the lemma are satisfied when we "recurse". By assumption, $\ell_i \geq 1$ for all i and the new $\mathcal{A}$ is sufficiently large. Moreover, him can also decrease when taking submatrices, so indeed we only need to show that $\mathrm{him}(A[\mathcal{R}'_i, \cdot]) < \ell_i - 1$ for $\mathcal{R}'_i = \{r_i \mid r \in \mathcal{A}^i_{u_i}\}$. If there is a half-induced matching of size $\ell_i - 1$, induced on rows $w_1, \dots, w_{\ell_i - 1} \in \mathcal{R}'_i$ and columns $z_1, \dots, z_{\ell_i - 1}$, then there is a half-induced matching of size ℓ_i in $A[\mathcal{R}_i, \cdot]$ by considering rows $w_1, \dots, w_{\ell_i - 1}, v_i \in \mathcal{R}_i$ and columns $z_1, \dots, z_{\ell - 1}, u$. But by assumption this does not exist, so indeed $\mathrm{him}(A[\mathcal{R}'_i, \cdot]) < \ell_i - 1$. ◀

Proof of Lemma 8. Suppose that $|\mathcal{A}| \geq g_k(\ell, \ldots, \ell)$. For $i \in [k]$, set $\mathcal{R}_i = \{r_i \mid r \in \mathcal{A}\}$. Then $\operatorname{him}(A[\mathcal{R}_i, \cdot]) < \ell$ for each i. By Lemma 10 we can find a row v in $\mathcal{A}$ such that $\mathcal{A} \setminus \{v\}$ $A^{\otimes k}$-represents $\mathcal{A}$ in time $\mathcal{O}(\ell k \cdot |\mathcal{A}|hk) = \mathcal{O}(|\mathcal{A}|h^2k^2)$, where we use that $\ell \leq h$.

We repeat this at most $|\mathcal{A}| - g_k(\ell, \ldots, \ell)$ times until we find the desired representative set in time $\mathcal{O}(|\mathcal{A}|^2h^2k^2)$. ◀

5 Comparing him and mimsup

In this section, we discuss the relation between the considered parameters.

▶ **Lemma 11.** *Let A be a matrix. Then* $\operatorname{mimsup}(A) \geq \operatorname{him}(A) \geq \operatorname{mim}(A)$.

Proof. The second inequality follows directly since each induced matching is a half-induced matching. We prove the first inequality.

Let $R = \{a_1, \ldots, a_i\}$ and $C = \{b_1, \ldots, b_i\}$ be the rows and columns respectively of a maximum half-induced matching in A. We may assume that these are ordered such that $A[a_j, b_j] = 1$ for all $j \in [i]$ and $A[a_k, b_j] = 0$ for all $k < j$. For integers $s \geq 1$, we consider the submatrix of $A^{\otimes is}$ induced on the rows consisting of "balanced" sequences, and similarly for the columns

$$\{(r_1, \ldots, r_{is}) \in R^{is} \mid |\{\ell : r_\ell = a_j\}| = s, \text{ for every } j \in [i]\},$$
$$\{(c_1, \ldots, c_{is}) \in C^{is} \mid |\{\ell : c_\ell = b_j\}| = s, \text{ for every } j \in [i]\}.$$

We claim this forms an induced matching of size $\binom{is}{s,\ldots,s}$. By Stirling approximation, $\binom{is}{s,\ldots,s} = i^{(1+o(1))is}$ and so the claim implies that $\operatorname{mimsup}(A) \geq i = \operatorname{him}(A)$. Since the size is clear from the definition, it only remains to check that it indeed forms an induced matching. To show this, we explain why the row of

$$r = (a_1, \ldots, a_1, a_2, \ldots, a_2, \ldots, a_i, \ldots, a_i)$$

has a single one entry in column

$$c = (b_1, \ldots, b_1, b_2, \ldots, b_2, \ldots, b_i, \ldots, b_i).$$

The other cases follow by symmetry. It is clear that $A^{\otimes is}[r, c] = 1$. Any column c' with $A^{\otimes is}[r, c'] = 1$ must have $A[a_1, c'_j] = 1$ for all $j \in [s]$. But $A[a_1, b_j] = 0$ when $j > 1$, so this implies that $\{j : c'_j = b_1\} = [s]$. Similarly, we require $A[a_2, c'_j] = 1$ for all $j \in [s+1, 2s]$, but c' has already "used" all its b_1's and so $\{j \mid c'_j = b_2\}$ needs to be $[s+1, 2s]$. Continuing inductively, we find that $\{j \mid c'_j = b_k\} = [(k-1)s + 1, ks]$ for all $k \in [i]$, that is, $c' = c$. ◀

We are now ready to prove previously claimed bounds.

▶ **Theorem 3.** *For every non-bipartite graph H with adjacency matrix A_H and $k \in \mathbb{N}$,*

$$\operatorname{him}(A_H) \leq \operatorname{mimsup}(A_H) = \limsup_{k\to\infty} \operatorname{mim}(A_H^{\otimes k})^{1/k} \quad \text{and} \quad \operatorname{mim}(A_H^{\otimes k}) \leq k^{(\operatorname{him}(A_H)+1)k}.$$

Proof. Lemma 11 shows the first inequality. The proof of Lemma 10 implies that for any matrix A, $\operatorname{mim}(A^{\otimes k}) \leq k^{\operatorname{him}(A)k}$, since no smaller representative set can be found when the rows induce an induced matching (with some set of columns). ◀

The following observation implies that sequences such as $\operatorname{mim}(A^{\otimes 2^k})^{1/2^k}$ are non-decreasing.

▶ **Lemma 12 (♠).** *Given two matrices A and B,* $\operatorname{mim}(A \otimes B) \geq \operatorname{mim}(A)\operatorname{mim}(B)$. *In particular,* $\operatorname{mimsup}(A^{\otimes k}) = \operatorname{mimsup}(A)^k$.

The results above also imply that

$$\operatorname{mimsup}(A) = \limsup_{k\to\infty} \operatorname{him}(A^{\otimes k})^{1/k}.$$

At first glance, it may be natural to conjecture that in fact $\operatorname{mimsup}(A) = \operatorname{him}(A)$ for all matrices A. This is however not true, as the following result shows.

▶ **Theorem 13 (♠).** *For all sufficiently large integers h, there is a symmetric $(2h \times 2h)$ matrix A with* $\operatorname{him}(A) \leq 10\log_2 h$ *and* $\operatorname{mimsup}(A) \geq \sqrt{h}$.

6 Conclusion

An obvious open problem is to fully resolve Open Problem 1. As discussed, to achieve this goal we only lack a fast algorithm that computes representative sets for partial solutions to $\textsc{Hom}(H)$. A far more ambitious (and probably currently out of reach) goal is to provide a (more) fine-grained version of the Courcelle's theorem for deciding any graph property definable in the monadic second-order logic. While being homomorphic to a given graph H is of course only a very special sort of such a property, we find our progress on Open Problem 1 encouraging in this respect and hope that eventually similar connections between the complexity of more general computational problems and asymptotic rank parameters can be made as well. In particular, we believe that mimsup (or a similar parameter that tracks the asymptotic behavior under appropriate products) is likely to determine the limit of dynamic programming approaches in other settings as well, especially those determined by various graph width parameters.

Another suggested direction of research is purely combinatorial/algebraic: we expect that mimsup is an interesting parameter for further study in its own right. We suggest following questions. (i) What type of values can $\operatorname{mimsup}(H)$ take given a matrix H? Can it take non-integer values? Similar questions have recently been investigated for asymptotic tensor parameters, see e.g. [3,6]. (ii) What is the value of mimsup for a $n \times n$ random matrix, where each entry of the matrix gets sampled independently to be 1 with probability p and to be 0 with probability $1-p$? (iii) We showed that him and the support rank are not functionally equivalent. Is mimsup functionally equivalent to either him or the support rank?

References

1 Noga Alon and Eyal Lubetzky. The shannon capacity of a graph and the independence numbers of its powers. *IEEE Trans. Inf. Theory*, 52(5):2172–2176, 2006. doi:10.1109/TIT.2006.872856.

2 Srinivasan Arunachalam, Péter Vrana, and Jeroen Zuiddam. The asymptotic induced matching number of hypergraphs: Balanced binary strings. *The Electronic Journal of Combinatorics*, 27(3), 2020. doi:10.37236/9019.

3 Andreas Blatter, Jan Draisma, and Filip Rupniewski. Countably many asymptotic tensor ranks. *arXiv*, 2022. arXiv:2212.12219.

4 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.

5 Glencora Borradaile and Hung Le. Optimal dynamic program for r-domination problems over tree decompositions. In Jiong Guo and Danny Hermelin, editors, *IPEC 2016*, volume 63 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.8.

6 Jop Briët, Matthias Christandl, Itai Leigh, Amir Shpilka, and Jeroen Zuiddam. Discreteness of asymptotic tensor ranks. *arXiv*, 2023. arXiv:2306.01718.

7 Andrei A. Bulatov and Amirhossein Kazeminia. Complexity classification of counting graph homomorphisms modulo a prime number. In Stefano Leonardi and Anupam Gupta, editors, *STOC 2022*, pages 1024–1037. ACM, 2022. doi:10.1145/3519935.3520075.

8 Peter Bürgisser, Michael Clausen, and Amin Shokrollahi. Algebraic complexity theory. In *Grundlehren der mathematischen Wissenschaften*, 1997.

9 Jin-Yi Cai and Ashwin Maran. The complexity of counting planar graph homomorphisms of domain size 3. In Barna Saha and Rocco A. Servedio, editors, *STOC 2023*, pages 1285–1297. ACM, 2023. doi:10.1145/3564246.3585173.

10 Airlie Chapman and Mehran Mesbahi. On strong structural controllability of networked systems: A constrained matching approach. In *2013 American Control Conference*, pages 6126–6131, 2013. doi:10.1109/ACC.2013.6580798.

11 Prasad Chaugule, Nutan Limaye, and Aditya Varre. Variants of homomorphism polynomials complete for algebraic complexity classes. *ACM Trans. Comput. Theory*, 13(4):21:1–21:26, 2021. doi:10.1145/3470858.

12 Rajesh Chitnis, László Egri, and Dániel Marx. List h-coloring a graph by removing few vertices. *Algorithmica*, 78(1):110–146, 2017. doi:10.1007/s00453-016-0139-6.

13 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

14 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *STOC 2017*, pages 210–223. ACM, 2017. doi:10.1145/3055399.3055502.

15 Radu Curticapean, Nathan Lindzey, and Jesper Nederlof. A tight lower bound for counting Hamiltonian cycles via matrix rank. In Artur Czumaj, editor, *SODA 2018*, pages 1080–1099. SIAM, 2018. doi:10.1137/1.9781611975031.70.

16 Radu Curticapean and Dániel Marx. Tight conditional lower bounds for counting perfect matchings on graphs of bounded treewidth, cliquewidth, and genus. In Robert Krauthgamer, editor, *SODA 2016*, pages 1650–1669. SIAM, 2016. doi:10.1137/1.9781611974331.ch113.

17 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3):12:1–12:46, 2018. doi:10.1145/3148227.

18 Ronald de Wolf. Nondeterministic quantum query and communication complexities. *SIAM Journal on Computing*, 32(3):681–699, 2003. doi:10.1137/S0097539702407345.

19 MARTIN E. DYER and CATHERINE S. GREENHILL. The complexity of counting graph homomorphisms. *RANDOM STRUCT. ALGORITHMS*, 17(3-4):260–289, 2000.

20 László Egri, Andrei A. Krokhin, Benoît Larose, and Pascal Tesson. The complexity of the list homomorphism problem for graphs. In Jean-Yves Marion and Thomas Schwentick, editors, *STACS 2010*, volume 5 of *LIPIcs*, pages 335–346. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPIcs.STACS.2010.2467.

21 László Egri, Dániel Marx, and Paweł Rzążewski. Finding list homomorphisms from bounded-treewidth graphs to reflexive graphs: a complete complexity characterization. In Rolf Niedermeier and Brigitte Vallée, editors, *STACS 2018*, volume 96 of *LIPIcs*, pages 27:1–27:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.27.

22 Paul Erdős and George Szekeres. A combinatorial problem in geometry. *Compositio mathematica*, 2:463–470, 1935.

23 Baris Can Esmer, Jacob Focke, Dániel Marx, and Paweł Rzążewski. List homomorphisms by deleting edges and vertices: tight complexity bounds for bounded-treewidth graphs. *CoRR*, abs/2210.10677, 2022. doi:10.48550/arXiv.2210.10677.

24 M. Fekete. Über die Verteilung der Wurzeln bei gewissen algebraischen Gleichungen mit ganzzahligen Koeffizienten. *Mathematische Zeitschrift*, 17(1):228–249, December 1923. doi:10.1007/bf01504345.

25 Jacob Focke, Dániel Marx, Fionn Mc Inerney, Daniel Neuen, Govind S. Sankar, Philipp Schepper, and Philip Wellnitz. Tight complexity bounds for counting generalized dominating sets in bounded-treewidth graphs. In Nikhil Bansal and Viswanath Nagarajan, editors, *SODA 2023*, pages 3664–3683. SIAM, 2023. doi:10.1137/1.9781611977554.ch140.

26 Jacob Focke, Dániel Marx, and Paweł Rzążewski. Counting list homomorphisms from graphs of bounded treewidth: tight complexity bounds. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *SODA 2022*, pages 431–458. SIAM, 2022. doi:10.1137/1.9781611977073.22.

27 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016. doi:10.1145/2886094.

28 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Representative families of product families. *ACM Trans. Algorithms*, 13(3):36:1–36:29, 2017. doi:10.1145/3039243.

29 Robert Ganian, Thekla Hamm, Viktoriia Korchemna, Karolina Okrasa, and Kirill Simonov. The fine-grained complexity of graph homomorphism parameterized by clique-width. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *ICALP 2022*, volume 229 of *LIPIcs*, pages 66:1–66:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.66.

30 Chris Godsil. Problems in algebraic combinatorics. *Electr. J. Comb.*, 2, January 1995. doi:10.37236/1224.

31 Carla Groenland, Isja Mannens, Jesper Nederlof, Marta Piecyk, and Pawel Rzazewski. Towards tight bounds for the graph homomorphism problem parameterized by cutwidth via asymptotic rank parameters. *CoRR*, abs/2312.03859, 2023. doi:10.48550/arXiv.2312.03859.

32 Carla Groenland, Isja Mannens, Jesper Nederlof, and Krisztina Szilágyi. Tight bounds for counting colorings and connected edge sets parameterized by cutwidth. In Petra Berenbrink and Benjamin Monmege, editors, *STACS 2022*, volume 219 of *LIPIcs*, pages 36:1–36:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.36.

33 Pavol Hell and Jaroslav Nešetřil. The core of a graph. *Discrete Mathematics*, 109(1-3):117–126, 1992. doi:10.1016/0012-365X(92)90282-K.

34 Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990. doi:10.1016/0095-8956(90)90132-J.

35 Lars Jaffke and Bart M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. *Discret. Appl. Math.*, 327:33–46, 2023. doi:10.1016/j.dam.2022.11.011.

36 Bart M. P. Jansen and Jesper Nederlof. Computing the chromatic number using graph decompositions via matrix rank. *Theor. Comput. Sci.*, 795:520–539, 2019. doi:10.1016/j.tcs.2019.08.006.

37 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discret. Appl. Math.*, 264:90–117, 2019. doi:10.1016/j.dam.2018.11.002.

38 Tomasz Kociumaka and Marcin Pilipczuk. Deleting vertices to graphs of bounded genus. *Algorithmica*, 81(9):3655–3691, 2019. doi:10.1007/s00453-019-00592-7.

39 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *J. ACM*, 67(3):16:1–16:50, 2020. doi:10.1145/3390887.

40 Michael Lampis. Finer tight bounds for coloring on clique-width. *SIAM J. Discret. Math.*, 34(3):1538–1558, 2020. doi:10.1137/19M1280326.

41 Benoit Larose. Families of strongly projective graphs. *Discussiones Mathematicae Graph Theory*, 22:271–292, 2002.

42 Benoit Larose and Claude Tardif. Strongly rigid graphs and projectivity. *Multiple-Valued Logic*, 7:339–361, 2001.

43 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/92.

44 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

45 Tomasz Łuczak and Jaroslav Nešetřil. Note on projective graphs. *Journal of Graph Theory*, 47(2):81–86, 2004.

46 Dániel Marx, Govind S. Sankar, and Philipp Schepper. Degrees and gaps: Tight complexity results of general factor problems parameterized by treewidth and cutwidth. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *ICALP 2021*, volume 198 of *LIPIcs*, pages 95:1–95:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.95.

47 Dániel Marx, Govind S. Sankar, and Philipp Schepper. Anti-Factor Is FPT Parameterized by Treewidth and List Size (But Counting Is Hard). In Holger Dell and Jesper Nederlof, editors, *IPEC 2022*, volume 249 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:23, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2022.22.

48 Burkhard Monien. The complexity of determining paths of length k. In Manfred Nagl and Jürgen Perl, editors, *WG '83*, pages 241–251. Universitätsverlag Rudolf Trauner, Linz, 1983.

49 Jesper Nederlof. Algorithms for np-hard problems via rank-related parameters of matrices. In Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen, editors, *Treewidth, Kernels, and Algorithms - Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *Lecture Notes in Computer Science*, pages 145–164. Springer, 2020. doi:10.1007/978-3-030-42071-0_11.

50 Jesper Nederlof. Bipartite TSP in $O(1.9999^n)$ time, assuming quadratic time matrix multiplication. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *STOC 2020*, pages 40–53. ACM, 2020. doi:10.1145/3357713.3384264.

51 Jaroslav Nešetřil and Aleš Pultr. A Dushnik - Miller type dimension of graphs and its complexity. In Marek Karpiński, editor, *Fundamentals of Computation Theory*, pages 482–493, Berlin, Heidelberg, 1977. Springer Berlin Heidelberg.

52 Jaroslav Nešetřil and Vojtéch Rödl. A simple proof of the Galvin-Ramsey property of the class of all finite graphs and a dimension of a graph. *Discrete Mathematics*, 23(1):49–55, 1978. doi:10.1016/0012-365X(78)90186-3.

53 Karolina Okrasa, Marta Piecyk, and Paweł Rzążewski. Full complexity classification of the list homomorphism problem for bounded-treewidth graphs. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *ESA 2020*, volume 173 of *LIPIcs*, pages 74:1–74:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.74.

54 Karolina Okrasa and Paweł Rzążewski. Fine-grained complexity of the graph homomorphism problem for bounded-treewidth graphs. *SIAM J. Comput.*, 50(2):487–508, 2021. doi:10.1137/20M1320146.

55 D.D. Olesky, Michael Tsatsomeros, and P. van den Driessche. Qualitative controllability and uncontrollability by a single entry. *Linear Algebra and its Applications*, 187:183–194, 1993. doi:10.1016/0024-3795(93)90134-A.

56 Marta Piecyk and Paweł Rzążewski. Fine-grained complexity of the list homomorphism problem: Feedback vertex set and cutwidth. In Markus Bläser and Benjamin Monmege, editors, *STACS 2021*, volume 187 of *LIPIcs*, pages 56:1–56:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.56.

57 Michał Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: A logical approach. In *MFCS 2011*, volume 6907, pages 520–531. Springer, 2011.

58 Marc Roth and Philip Wellnitz. Counting and finding homomorphisms is universal for parameterized complexity theory. In Shuchi Chawla, editor, *SODA 2020*, pages 2161–2180. SIAM, 2020. doi:10.1137/1.9781611975994.133.

59 Maguy Trefois and Jean-Charles Delvenne. Zero forcing number, constrained matchings and strong structural controllability. *Linear Algebra and its Applications*, 484:199–218, 2015.

60 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In Amos Fiat and Peter Sanders, editors, *ESA 2009*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0_51.

Isomorphism for Tournaments of Small Twin Width

Martin Grohe
RWTH Aachen University, Germany

Daniel Neuen
University of Regensburg, Germany

Abstract

We prove that isomorphism of tournaments of twin width at most k can be decided in time $k^{O(\log k)}n^{O(1)}$. This implies that the isomorphism problem for classes of tournaments of bounded or moderately growing twin width is in polynomial time. By comparison, there are classes of undirected graphs of bounded twin width that are isomorphism complete, that is, the isomorphism problem for the classes is as hard as the general graph isomorphism problem. Twin width is a graph parameter that has been introduced only recently (Bonnet et al., FOCS 2020), but has received a lot of attention in structural graph theory since then. On directed graphs, it is functionally smaller than clique width. We prove that on tournaments (but not on general directed graphs) it is also functionally smaller than directed tree width (and thus, the same also holds for cut width and directed path width). Hence, our result implies that tournament isomorphism testing is also fixed-parameter tractable when parameterized by any of these parameters.

Our isomorphism algorithm heavily employs group-theoretic techniques. This seems to be necessary: as a second main result, we show that the combinatorial Weisfeiler-Leman algorithm does not decide isomorphism of tournaments of twin width at most 35 if its dimension is $o(n)$. (Throughout this abstract, n is the order of the input graphs.)

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Mathematics of computing → Graph algorithms; Theory of computation → Finite Model Theory

Keywords and phrases tournament isomorphism, twin width, fixed-parameter tractability, Weisfeiler-Leman algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.78

Category Track A: Algorithms, Complexity and Games

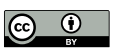
Related Version *Full Version*: `https://arxiv.org/abs/2312.02048`

Funding *Martin Grohe*: The research is funded by the European Union (ERC, SymSim, 101054974). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Acknowledgements We thank the anonymous reviewers of an earlier version of this paper for helpful feedback which, in particular, resulted an improved bound in Theorem 1.2.

1 Introduction

The tournament isomorphism problem (TI) was recognized as a particularly interesting special case of the graph isomorphism problem (GI) early-on. Already in 1983, Babai and Luks [3] proved that TI is solvable in time $n^{O(\log n)}$; it took 33 more years for Babai [2] to prove that the general GI is in quasi-polynomial time. An important fact that makes TI more accessible than GI is that tournaments always have solvable automorphism groups. This is a consequence of the observation that the automorphism groups of tournaments have odd order and the famous Feit-Thompson Theorem [18] stating that all groups of odd order are solvable. However, even Babai's powerful new machinery did not help us to




improve the upper bound for TI, as one might have hoped. But TI is not only special from a group-theoretic perspective. Another remarkable result, due to Schweitzer [42], states that TI reduces to the problem of deciding whether a tournament has a nontrivial automorphism; the so-called *rigidity problem*. It is an open question whether the same holds for general graphs.

While there is an extensive literature on GI restricted to classes of graphs (see [25] for a recent survey), remarkably little is known for restrictions of TI. Ponomarenko [40] proved that TI is in polynomial time for tournaments whose automorphism group contains a regular cyclic subgroup, and recently Arvind, Ponomarenko, and Ryabov [1] proved that TI is in polynomial time for edge-colored tournaments where at least one edge color induces a (strongly) connected spanning subgraph of bounded degree (even fixed-parameter tractable when parameterized by the out-degree). While both of these results are very interesting from a technical perspective, they consider classes of tournaments that would hardly be called natural from a graph-theoretic point of view. Natural graph parameters that have played a central role in the structural theory of tournaments developed by Chudnovsky, Seymour and others [13, 14, 15, 16, 19] are *cut width* and *path width*. The more recent theory of structural sparsity [20, 21, 36, 37] highlights *clique width* and *twin width*. Here twin width is the key parameter. Not only is it functionally smaller than the other parameters, which means that if cut width, path width, or clique width is bounded, then twin width is bounded as well, it is also known [23] that a class of tournaments has bounded twin width if and only if it has a property known as *monadic dependence (NIP)*. Dependence is a key property studied in classical model theory. A class of graphs is monadically dependent if and only if all set systems definable in this class by a first-order transduction have bounded VC dimension. This property seems to characterize precisely the graph classes that are regarded as structurally sparse. Since twin width of graphs and binary relational structures has been introduced in [11], it received a lot of attention in algorithmic structural graph theory [5, 6, 7, 8, 9, 10, 21, 22, 23, 29, 44]. (We defer the somewhat unwieldy definition of twin width to Section 2.3.) Our main result states that tournament isomorphism is fixed-parameter tractable when parameterized by twin width.

▶ **Theorem 1.1.** *The isomorphism problem for tournaments of twin width at most k can be solved in time $k^{O(\log k)} \cdot n^{O(1)}$.*

Interestingly, isomorphism testing for undirected graphs of bounded twin width is as hard as the general GI. This follows easily from the fact that an $\Omega(\log n)$-subdivision of every graph with n vertices has bounded twin width [5]. Once more, this demonstrates the special role of the tournament isomorphism problem, though here the reason is not group-theoretic, but purely combinatorial.

Note that the dependence on the twin width k of the algorithm in Theorem 1.1 is subexponential, so our result implies that TI is in polynomial time even for tournaments of twin width $2^{O(\sqrt{\log n})}$. Since twin width is functionally smaller than clique width, our result implies that TI is also fixed-parameter tractable when parameterized by clique width. Additionally, we prove that the twin width of a tournament is functionally smaller than its *directed tree width*, a graph parameter originally introduced in [31]. Since the directed tree width of every directed graph is smaller than its cut width or directed path width, the same also holds for these two parameters. Hence, TI is fixed-parameter tractable also when parameterized by directed tree width, directed path width or cut width. To the best of our knowledge, this was not known for any of these parameters. The fact that twin width is functionally smaller than directed tree width, directed path width and cut width on tournaments is interesting in its own right, because this result does not extend to general directed graphs (for any of the three parameters).

Our proof of Theorem 1.1 heavily relies on group-theoretic techniques. In a nutshell, we show that bounded twin width allows us to cover a tournament by a sequence of directed graphs that have a property resembling bounded degree sufficiently closely to apply a group-theoretic machinery going back to Luks [33] and developed to great depth since then (see, e.g., [2, 3, 27, 35, 38]). Specifically, we generalize arguments that have been introduced by Arvind et al. [1] for TI on edge-colored tournaments where at least one edge color induces a spanning subgraph of bounded out-degree.

Yet one may wonder if this heavy machinery is even needed to prove our theorem, in particular in view of the fact that on many natural graph classes, including, for example, undirected graphs of bounded clique width [26], the purely combinatorial Weisfeiler-Leman algorithm is sufficient to decide isomorphism (see, e.g., [24, 32]). We prove that this is not the case for tournaments of bounded twin width.

▶ **Theorem 1.2.** *For every $k \geq 2$ there are non-isomorphic tournaments T_k and T'_k of order $|V(T_k)| = |V(T'_k)| = O(k)$ and twin width at most 35 that are not distinguished by the k-dimensional Weisfeiler-Leman algorithm.*

We remark that it was known before that the Weisfeiler-Leman algorithm fails to decide tournament isomorphism. Indeed, Dawar and Kopczynski (unpublished) proved that for every $k \geq 2$ there are non-isomorphic tournaments U_k and U'_k of order $|V(U_k)| = |V(U'_k)| = O(k)$ that are not distinguished by the k-dimensional Weisfeiler-Leman algorithm. Theorem 1.2 strengthens this result by constructing tournaments where additionally the twin width is bounded by a fixed constant.

The paper is organized as follows. After introducing the necessary preliminaries in Section 2, Theorem 1.1 is proved in Sections 3 and 4. First, we give our main combinatorial arguments in Section 3. After that, the mainly group-theoretic isomorphism algorithm of Theorem 1.1 is presented in Section 4. Theorem 1.2 is proved in Section 5. Finally, in Section 6 we compare twin width to other width measures for directed graphs. All omitted proofs can be found in the full version.

2 Preliminaries

2.1 Graphs

Graphs in this paper are usually directed. We often emphasize this by calling them "digraphs". However, when we make general statements about graphs, this refers to directed graphs and includes undirected graphs a special case (directed graphs with a symmetric edge relation). We denote the vertex set of a graph G by $V(G)$ and the edge relation by $E(G)$. The vertex set $V(G)$ is always finite and non-empty. The edge relation is always anti-reflexive, that is, graphs are loop-free, and there are no parallel edges. For a digraph G and a vertex $v \in V(G)$, we denote the set of out-neighbors and in-neighbors of v by $N_+(v)$ and $N_-(v)$, respectively. Also, the *out-degree* and *in-degree* of v are denoted by $\deg_+(v) := |N_+(v)|$ and $\deg_-(v) := |N_-(v)|$, respectively. Furthermore, $E_+(v)$ and $E_-(v)$ denote the set of outgoing and incoming edges into v, respectively. For $X \subseteq V(G)$, we write $G[X]$ to denote the subgraph of G induced on X. For two sets $X, Y \subseteq V(G)$ we write $E_G(X,Y) := \{(v,w) \in E(G) \mid v \in X, w \in Y\}$ to denote the set of directed edges from X to Y.

Let G be an undirected graph. A directed graph $\vec{G}$ is an *orientation* of G if, for every undirected edge $\{v,w\} \in E(G)$, exactly one of (v,w) and (w,v) is an edge of $\vec{G}$, and there are no other edges present in $\vec{G}$. A *tournament* is an orientation of a complete graph.

A tournament T is *regular* if $\deg_+(v) = \deg_+(w)$ for all $v, w \in V(T)$. In this case, $\deg_+(v) = \deg_-(v) = \frac{|V(G)|-1}{2}$ for all $v \in V(T)$. This implies that every regular tournament has an odd number of vertices.

Let G_1, G_2 be two graphs. An *isomorphism* from G_1 to G_2 is a bijection $\varphi\colon V(G_1) \to V(G_2)$ such that $(v,w) \in E(G_1)$ if and only if $(\varphi(v), \varphi(w)) \in E(G_2)$ for all $v, w \in V(G_1)$. We write $\varphi\colon G_1) \cong G_2$ to denote that φ is an isomorphism from G_1 to G_2. Also, $\mathrm{Iso}(G_1, G_2)$ denotes the set of all isomorphisms from G_1 to G_2. The graphs G_1 and G_2 are *isomorphic* if $\mathrm{Iso}(G_1, G_2) \neq \emptyset$. The *automorphism group* of G_1 is $\mathrm{Aut}(G_1) \coloneqq \mathrm{Iso}(G_1, G_1)$.

An *arc coloring* of a digraph G is a mapping $\lambda\colon (E(G) \cup \{(v,v) \mid v \in V(G)\}) \to C$ for some set C of "colors". An *arc-colored graph* is a triple $G = (V, E, \lambda)$, where (V, E) is a graph an λ an arc coloring of (V, E). Isomorphisms between arc-colored graphs are required to preserve the coloring.

2.2 Partitions and Colorings

Let S be a finite set. A *partition* of S is a set $\mathcal{P} \subseteq 2^S$ whose elements we refer to as *parts*, such that any two parts are mutually disjoint, and the union of all parts is S. A partition $\mathcal{P}$ *refines* another partition $\mathcal{Q}$, denoted by $\mathcal{P} \preceq \mathcal{Q}$, if for all $P \in \mathcal{P}$ there is some $Q \in \mathcal{Q}$ such that $P \subseteq Q$. We say a partition $\mathcal{P}$ is *trivial* if $|\mathcal{P}| = 1$, which means that the only part is S, and it is *discrete* if $|P| = 1$ for all $P \in \mathcal{P}$.

Every mapping $\chi\colon S \to C$, for some set C, induces a partition $\mathcal{P}_\chi$ of S into the sets $\chi^{-1}(c)$ for all c in the range of χ. In this context, we think of χ as a "coloring" of S, the elements $c \in C$ as "colors", and the parts $\chi^{-1}(c)$ of the partition as "color classes". If $\chi'\colon S \to C'$ is another coloring, then we say that χ *refines* χ', denoted by $\chi \preceq \chi'$, if $\mathcal{P}_\chi \preceq \mathcal{P}_{\chi'}$. The colorings are *equivalent* (we write $\chi \equiv \chi'$) if $\chi \preceq \chi'$ and $\chi \preceq \chi'$, i.e., $\mathcal{P}_\chi = \mathcal{P}_{\chi'}$.

2.3 Twin Width

Twin width [11] is defined for binary relational structures, which in this paper are mostly directed graphs. We need one distinguished binary relation symbol E_{red} that plays a special role in the definition of twin width. Following [11], we refer to elements of E_{red} as *red edges*. For every structure A, we assume the relation $E_{\mathsf{red}}(A)$ to be symmetric and anti-reflexive, that is, the edge relation of an undirected graph, and we refer to the maximum degree of this graph as the *red degree* of A. If $E_{\mathsf{red}}(A)$ is not explicitly defined, we assume $E_{\mathsf{red}}(A) = \emptyset$ (and the red degree of A is 0).

Let $A = (V(A), R_1(A), \ldots, R_k(A))$ be a binary relational structure, where $V(A)$ is a non-empty finite vertex set and $R_i(A) \subseteq (V(A))^2$ are binary relations on $V(A)$ (possibly, $R_i = E_{\mathsf{red}}$ for some $i \in [k]$). We call a pair (X, Y) of disjoint subsets of $V(A)$ *homogeneous* if for all $x, x' \in X$, and all $y, y' \in Y$ it holds that

- **(i)** $(x,y) \in R_i(A) \Leftrightarrow (x',y') \in R_i(A)$ and $(y,x) \in R_i(A) \Leftrightarrow (y',x') \in R_i(A)$ for all $i \in [k]$, and
- **(ii)** $(x,y) \notin E_{\mathsf{red}}(A)$ and $(y,x) \notin E_{\mathsf{red}}(A)$.

For a partition $\mathcal{P}$ of $V(A)$, we define $A/\mathcal{P}$ to be the structure with vertex set $V(A/\mathcal{P}) \coloneqq \mathcal{P}$ and relations

$$R_i(A/\mathcal{P}) \coloneqq \left\{(X,Y) \in \mathcal{P}^2 \;\middle|\; (X,Y) \text{ is homogeneous and } X \times Y \subseteq R_i(A)\right\}$$

for all $R_i \neq E_{\mathsf{red}}$, and

$$E_{\mathsf{red}}(A/\mathcal{P}) \coloneqq \left\{(X,Y) \in \mathcal{P}^2 \;\middle|\; (X,Y) \text{ is not homogeneous and } X \neq Y\right\}.$$

A *contraction sequence for* A is a sequence of partitions $\mathcal{P}_1, \dots, \mathcal{P}_n$ of $V(A)$ such that $\mathcal{P}_1 = \{\{v\} \mid v \in V(A)\}$ is the discrete partition, $\mathcal{P}_n = \{V(A)\}$ is the trivial partition, and for every $i \in [n-1]$ the partition $\mathcal{P}_{i+1}$ is obtained from $\mathcal{P}_i$ by merging two parts, i.e., there are distinct $P, P' \in \mathcal{P}_i$ such that $\mathcal{P}_{i+1} = \{P \cup P'\} \cup (\mathcal{P}_i \setminus \{P, P'\})$. The *width* of a contraction sequence $\mathcal{P}_1, \dots, \mathcal{P}_n$ of A is the minimum k such that for every $i \in [n]$ the structure $A/\mathcal{P}_i$ has red degree at most k. The *twin width* of A, denoted by $\operatorname{tww}(A)$, is the minimum $k \geq 0$ such that A has a contraction sequence of width k.

Note that red edges are introduced as we contract parts of the partitions. However, the structure A we start with may already have red edges, which then have direct impact on its twin width. In particular, the twin width of a graph G may be smaller than the twin width of the structure G^{red} obtained from G by coloring all edges red. This fact is used later.

We also remark that for our isomorphism algorithms, we never have to compute a contraction sequence of minimum width or the twin width.

We state two simple lemmas on basic properties of twin width.

▶ **Lemma 2.1** ([11]). *Let A be a binary relational structure and $X \subseteq V(A)$. Then* $\operatorname{tww}(A[X]) \leq \operatorname{tww}(A)$.

▶ **Lemma 2.2.** *Let A be a structure over the vocabulary τ. Then there is a linear order $<$ on $V(A)$ such that* $\operatorname{tww}(A, <) = \operatorname{tww}(A)$.

2.4 Weisfeiler-Leman

In this section, we describe the k-dimensional Weisfeiler-Leman algorithm (k-WL). The algorithm has been originally introduced in its 2-dimensional form by Weisfeiler and Leman [46] (see also [45]). The k-dimensional version, coloring k-tuples, was introduced later by Babai and Mathon (see [12]).

Fix $k \geq 2$, and let G be a graph. For $i \geq 0$, we describe the coloring $\chi^{k,G}_{(i)}$ of $(V(G))^k$ computed in the i-th iteration of k-WL. For $i = 0$, each tuple is colored with the isomorphism type of the underlying ordered induced subgraph. So if H is another graph and $\bar{v} = (v_1, \dots, v_k) \in (V(G))^k$, $\bar{w} = (w_1, \dots, w_k) \in (V(H))^k$, then $\chi^{k,G}_{(0)}(\bar{v}) = \chi^{k,H}_{(0)}(\bar{w})$ if and only if, for all $i, j \in [k]$, it holds that $v_i = v_j \Leftrightarrow w_i = w_j$ and $(v_i, v_j) \in E(G) \Leftrightarrow (w_i, w_j) \in E(H)$. If G and H are arc-colored, then the colors are also taken into account.

Now let $i \geq 0$. For $\bar{v} = (v_1, \dots, v_k)$ we define

$$\chi^{k,G}_{(i+1)}(\bar{v}) \coloneqq \left(\chi^{k,G}_{(i)}(\bar{v}), \mathcal{M}_i(\bar{v})\right)$$

where

$$\mathcal{M}_i(\bar{v}) \coloneqq \left\{\!\!\left\{ \left(\chi^{k,G}_{(i)}(\bar{v}[w/1]), \dots, \chi^{k,G}_{(i)}(\bar{v}[w/k])\right) \;\middle|\; w \in V(G) \right\}\!\!\right\}$$

and $\bar{v}[w/i] \coloneqq (v_1, \dots, v_{i-1}, w, v_{i+1}, \dots, v_k)$ is the tuple obtained from $\bar{v}$ by replacing the i-th entry by w (and $\{\!\{\dots\}\!\}$ denotes a multiset).

Clearly, $\chi^{k,G}_{(i+1)} \preceq \chi^{k,G}_{(i)}$ for all $i \geq 0$. So there is a unique minimal $i_\infty \geq 0$ such that $\chi^{k,G}_{(i_\infty+1)} \equiv \chi^{k,G}_{(i_\infty)}$ and we write $\chi^{k,G} \coloneqq \chi^{k,G}_{(i_\infty)}$ to denote the corresponding coloring.

The k-dimensional Weisfeiler-Leman algorithm takes as input a (possibly colored) graph G and outputs (a coloring that is equivalent to) $\chi^{k,G}$. This can be done in time $O(k^2 n^{k+1} \log n)$ [30].

Let H be a second graph. The k-dimensional Weisfeiler-Leman algorithm *distinguishes* G and H if there is a color $c \in C$ such that

$$\left|\left\{\bar{v} \in (V(G))^k \;\middle|\; \chi^{k,G}(\bar{v}) = c\right\}\right| \neq \left|\left\{\bar{w} \in (V(H))^k \;\middle|\; \chi^{k,H}(\bar{w}) = c\right\}\right|.$$

We write $G \simeq_k H$ to denote that k-WL does not distinguish between G and H.

A graph G is *k-WL-homogeneous* if for all $v, w \in V(G)$ it holds that $\chi^{k,G}(v, \dots, v) = \chi^{k,G}(w, \dots, w)$.

2.5 Group Theory

For a general background on group theory we refer to [41], whereas background on permutation groups can be found in [17]. Also, basics facts on algorithms for permutation groups are given in [43].

Basics for Permutation Groups. A *permutation group* acting on a set Ω is a subgroup $\Gamma \leq \mathrm{Sym}(\Omega)$ of the symmetric group. The size of the permutation domain Ω is called the *degree* of Γ. If $\Omega = [n] \coloneqq \{1, \dots, n\}$, then we also write S_n instead of $\mathrm{Sym}(\Omega)$. For $A \subseteq \Omega$ and $\gamma \in \Gamma$ let $\gamma(A) \coloneqq \{\gamma(\alpha) \mid \alpha \in A\}$. The set A is Γ-*invariant* if $\gamma(A) = A$ for all $\gamma \in \Gamma$.

Let $\theta \colon \Omega \to \Omega'$ be a bijection. We write $\Gamma\theta \coloneqq \{\gamma\theta \mid \gamma \in \Gamma\}$ for the set of bijections from Ω to Ω' obtained from concatenating a permutation from Γ and θ. Note that $(\gamma\theta)(\alpha) = \theta(\gamma(\alpha))$ for all $\alpha \in \Omega$.

A set $S \subseteq \Gamma$ is a *generating set* for Γ if for every $\gamma \in \Gamma$ there are $\delta_1, \dots, \delta_k \in S$ such that $\gamma = \delta_1 \dots \delta_k$. In order to perform computational tasks for permutation groups efficiently the groups are represented by generating sets of small size (i.e., polynomial in the size of the permutation domain). Indeed, most algorithms are based on so-called strong generating sets, which can be chosen of size quadratic in the size of the permutation domain of the group and can be computed in polynomial time given an arbitrary generating set (see, e.g., [43]).

Group-Theoretic Methods for Isomorphism Testing. In this work, we shall be interested in a particular subclass of permutation groups. Let Γ be a group and let $\gamma, \delta \in \Gamma$. The *commutator* of γ and δ is $[\gamma, \delta] \coloneqq \gamma^{-1}\delta^{-1}\gamma\delta$. The *commutator subgroup* $[\Gamma, \Gamma]$ of Γ is the unique subgroup of Γ generated by all commutators $[\gamma, \delta]$ for $\gamma, \delta \in \Gamma$. Note that $[\Gamma, \Gamma]$ is a normal subgroup of Γ. The *derived series of* Γ is the sequence of subgroups $\Gamma^{(0)} \trianglerighteq \Gamma^{(1)} \trianglerighteq \Gamma^{(2)} \trianglerighteq \dots$ where $\Gamma^{(0)} \coloneqq \Gamma$ and $\Gamma^{(i+1)} \coloneqq [\Gamma^{(i)}, \Gamma^{(i)}]$ for all $i \geq 0$. A group Γ is *solvable* if there is some $i \geq 0$ such that $\Gamma^{(i)}$ is the trivial group (i.e., it only contains the identity element). The next theorem follows from the Feit-Thompson Theorem stating that every group of odd order is solvable.

▶ **Theorem 2.3.** *Let T be a tournament. Then* $\mathrm{Aut}(T)$ *is solvable.*

Next, we state several basic group-theoretic algorithms for isomorphism testing.

▶ **Theorem 2.4** ([3, Theorem 4.1])**.** *There is an algorithm that, given two tournaments T_1 and T_2, computes* $\mathrm{Iso}(T_1, T_2)$ *in time $n^{O(\log n)}$.*

Note that $\mathrm{Iso}(T_1, T_2)$ may be of size exponential in the number of vertices of T_1 and T_2. However, if T_1 and T_2 are isomorphic (i.e., $\mathrm{Iso}(T_1, T_2) \neq \emptyset$), we have $\mathrm{Iso}(T_1, T_2) = \mathrm{Aut}(T_1)\varphi$ where $\varphi \in \mathrm{Iso}(T_1, T_2)$ is an arbitrary isomorphism from T_1 to T_2. Hence, the set $\mathrm{Iso}(T_1, T_2)$ can be represented by a generating set for $\mathrm{Aut}(T_1)$ of size polynomial in $|V(T_1)|$ and a single element $\varphi \in \mathrm{Iso}(T_1, T_2)$. Let us stress at this point that all isomorphism sets computed by the various algorithms discussed in this work are represented in this way.

Let G_1 and G_2 be two (colored) directed graphs. Also let $\Gamma \leq \mathrm{Sym}(V(G_1))$ be a permutation group and let $\theta \colon V(G_1) \to V(G_2)$ be a bijection. We define

$$\mathrm{Iso}_{\Gamma\theta}(G_1, G_2) \coloneqq \mathrm{Iso}(G_1, G_2) \cap \Gamma\theta = \{\varphi \in \Gamma\theta \mid \varphi \colon G_1 \cong G_2\}$$

and $\mathrm{Aut}_\Gamma(G_1) \coloneqq \mathrm{Iso}_\Gamma(G_1, G_1)$. Note that $\mathrm{Aut}_\Gamma(G_1) \leq \Gamma$ and, if $\mathrm{Iso}_{\Gamma\theta}(G_1, G_2) \neq \emptyset$, then $\mathrm{Iso}_{\Gamma\theta}(G_1, G_2) = \mathrm{Aut}_\Gamma(G_1)\varphi$ where $\varphi \in \mathrm{Iso}(G_1, G_2)$ is an arbitrary isomorphism from G_1 to G_2.

▶ **Theorem 2.5** ([3, Corollary 3.6]). *Let $G_1 = (V_1, E_1, \lambda_1)$ and $G_1 = (V_2, E_2, \lambda_2)$ be two arc-colored directed graphs. Also let $\Gamma \leq \mathrm{Sym}(V_1)$ be a solvable group and $\theta\colon V_1 \to V_2$ a bijection. Then $\mathrm{Iso}_{\Gamma\theta}(G_1, G_2)$ can be computed in polynomial time.*

3 Small Degree Partition Sequences

In the following, we design an isomorphism test for tournaments of twin width k which runs in time $k^{O(\log k)}n^{O(1)}$. On a high level, the algorithm essentially proceeds in three phases. First, we use well-established group-theoretic methods going back to [3, 33] to reduce to the case where both input tournaments are 2-WL-homogeneous (without increasing the twin width). In the second step, we identify a substructure of an input tournament T (that is 2-WL-homogeneous) that has some kind of bounded-degree property. More concretely, we apply the 2-dimensional Weisfeiler-Leman algorithm and compute a sequence of colors $c_1, \dots, c_\ell$ in the image of the 2-WL coloring $\chi^{2,T}$ so that the subgraph induced by all edges with a color from $c_1, \dots, c_\ell$ has a certain type of bounded-degree property. After that, we rely on the computed bounded-degree structure to determine isomorphisms based on the group-theoretic graph isomorphism machinery. Similar tools have also been used in [1] to solve isomorphism of k-spanning tournaments. However, as we shall see below, the bounded-degree property guaranteed by the second step is weaker than the notion of k-spanning tournaments, which requires us to further extend the methods from [1].

In this section, we implement the second phase and prove the key combinatorial lemma (Lemma 3.6) underlying our isomorphism algorithm. Our arguments rely on the notion of *mixed neighbors* for a pair of vertices. For a pair $v, w \in V(T)$ of vertices we let

$$M(v, w) \coloneqq \Big(N_-(v) \cap N_+(w)\Big) \cup \Big(N_+(v) \cap N_-(w)\Big). \tag{1}$$

We call the elements of $M(v, w)$ the *mixed neighbors* of (v, w), and we call $\mathrm{md}(v, w) \coloneqq |M(v, w)|$ the *mixed degree* of (v, w). The following simple observation links the mixed degree to twin width.

▶ **Observation 3.1.** *There is an edge $(v, w) \in E(T)$ such that $\mathrm{md}(v, w) \leq \mathrm{tww}(T)$.*

Proof. Let $k \coloneqq \mathrm{tww}(T)$ and let $\mathcal{P}_1, \dots, \mathcal{P}_n$ be a contraction sequence of T of width k. Let $\{v, w\}$ be the unique 2-element part in $\mathcal{P}_2$. Then $\mathrm{md}(v, w) = \mathrm{md}(w, v) \leq k$, and either $(v, w) \in E(T)$ or $(w, v) \in E(T)$. ◀

In the following, let G_T be the directed graph with vertex set $V(G_T) \coloneqq V(T)$ and edge set $E(G_T) \coloneqq \{(v, w) \in E(T) \mid \mathrm{md}(v, w) \leq \mathrm{tww}(T)\}$. The next lemma implies that G_T has maximum out-degree at most $2 \cdot \mathrm{tww}(T) + 1$.

▶ **Lemma 3.2.** *Suppose $k \geq 1$. Let T be a tournament and let $v \in V(T)$. Also let*

$$W \coloneqq \{w \in N_+(v) \mid \mathrm{md}(v, w) \leq k\}.$$

Then $|W| \leq 2k + 1$.

Proof. Let $\ell := |W|$. The induced subtournament $T[W]$ has a vertex w of in-degree at least $(\ell-1)/2$. Since $\mathrm{md}(v,w) \leq k$ and $(v,w') \in E(T)$ for all $w' \in W$, we have

$$|\{w' \in W \mid (w',w) \in E(T)\}| \leq k.$$

Thus $\frac{\ell-1}{2} \leq k$, which implies that $|W| = \ell \leq 2k+1$. ◀

So G_T is a subgraph of T of maximum out-degree $d := 2 \cdot \mathrm{tww}(T) + 1$. We remark that similar arguments also show that G_T has maximum in-degree at most d (technically, this property is not required by our algorithm, but it is helpful for the following explanations). Now, first suppose that G_T is strongly connected. Then the edges of G_T define a (strongly) connected spanning subgraph of maximum degree $2d$ (in-degree plus out-degree). In this situation, we can directly use the algorithm from [1] to test isomorphism in time $d^{O(\log d)} n^{O(1)}$.

So suppose G_T is not strongly connected. If T is 2-WL-homogeneous, then G_T is also not weakly connected (i.e., the undirected version of G_T is not connected). In this case, the basic idea is to identify further edges to be added to decrease the number of connected components while keeping some kind of bounded-degree property.

In the following, let $\mathcal{Q}$ be a partition of $V(T)$ that is non-trivial, that is, has at least two parts. The reader is encouraged to think of $\mathcal{Q}$ as the partition into the (weakly) connected components of G_T, but the following results hold for any non-trivial partition $\mathcal{Q}$. We call an edge $(v,v') \in E(T)$ *cross-cluster with respect to* $\mathcal{Q}$ if it connects distinct $Q, Q' \in \mathcal{Q}$. For a cross-cluster edge (v,v') with $Q \ni v, Q' \ni v'$, we let

$$\mathcal{M}_{\mathcal{Q}}(v,v') := \left\{Q'' \in \mathcal{Q} \setminus \{Q,Q'\} \;\middle|\; Q'' \cap M(v,v') \neq \emptyset\right\}$$

and $\mathrm{md}_{\mathcal{Q}}(v,v') := |\mathcal{M}_{\mathcal{Q}}(v,v')|$.

The next two lemmas generalize Observation 3.1 and Lemma 3.2.

▶ **Lemma 3.3.** *Let T be a tournament and suppose $\mathcal{Q}$ is a non-trivial partition of $V(T)$. Then there is a cross-cluster edge $(v,w) \in E(T)$ such that $\mathrm{md}_{\mathcal{Q}}(v,w) \leq \mathrm{tww}(T)$.*

Proof. Let $k := \mathrm{tww}(T)$ and let $\mathcal{P}_1, \ldots, \mathcal{P}_n$ be a contraction sequence of T of width k. Note that $\mathcal{P}_1$ refines $\mathcal{Q}$ and $\mathcal{P}_n$ does not refine $\mathcal{Q}$, because $\mathcal{Q}$ is nontrivial. Let $i \geq 1$ be minimal such that $\mathcal{P}_{i+1}$ does not refine $\mathcal{Q}$.

Let $P, P' \in \mathcal{P}_i$ denote the parts merged in the step from $\mathcal{P}_i$ to $\mathcal{P}_{i+1}$. Since $\mathcal{P}_i$ refines $\mathcal{Q}$, there are $Q, Q' \in \mathcal{Q}$ such that $P \subseteq Q$ and $P' \subseteq Q'$. Moreover, $Q \neq Q'$, because $\mathcal{P}_{i+1}$ does not refine $\mathcal{Q}$. We pick arbitrary elements $v \in P$ and $w \in P'$ such that $(v,w) \in E(T)$ (if $(w,v) \in E(T)$, we swap the roles of P and P'). Then (v,w) is a cross-cluster edge with respect to $\mathcal{Q}$.

Let $P_1, \ldots, P_{k'}$ be a list of all $P'' \in \mathcal{P}_{i+1} \setminus \{P \cup P'\}$ such that the pair $(P \cup P', P'')$ is not homogeneous. Then $k' \leq k$ by the definition of twin width. Since $\mathcal{P}_i \setminus \{P,P'\} = \mathcal{P}_{i+1} \setminus \{P \cup P'\}$ and $\mathcal{P}_i$ refines $\mathcal{Q}$, there are $Q_1, \ldots, Q_{k'} \in \mathcal{Q}$ such that $P_i \subseteq Q_i$ for all $i \in [k']$.

Now let $Q'' \in \mathcal{Q} \setminus \{Q, Q', Q_1, \ldots, Q_{k'}\}$. Suppose for contradiction that $Q'' \cap M(v,w) \neq \emptyset$, and pick an element $w' \in Q'' \cap M(v,w)$. Then there is a $P'' \in \mathcal{P}_i \setminus \{P, P', P_1, \ldots, P_{k'}\} = \mathcal{P}_{i+1} \setminus \{P \cup P', P_1, \ldots, P_{k'}\}$ such that $w' \in P''$. But then the pair $(P \cup P', P'')$ is not homogeneous, which is a contradiction. So $Q'' \cap M(v,w) = \emptyset$. This implies that $\mathcal{M}_{\mathcal{Q}}(v,w) \subseteq \{Q_1, \ldots, Q_{k'}\}$. In particular, $\mathrm{md}_{\mathcal{Q}}(v,w) \leq k' \leq k$. ◀

▶ **Lemma 3.4.** *Suppose $k \geq 1$. Let T be a tournament and let $\mathcal{Q}$ be a non-trivial partition of $V(T)$. Also let $Q \in \mathcal{Q}$ and $v \in Q$. Let*

$$\mathcal{W} := \{Q' \in \mathcal{Q} \setminus \{Q\} \mid \exists w \in Q' \colon (v,w) \in E(T) \wedge \mathrm{md}_{\mathcal{Q}}(v,w) \leq k\}.$$

Then $|\mathcal{W}| \leq 2k+1$.

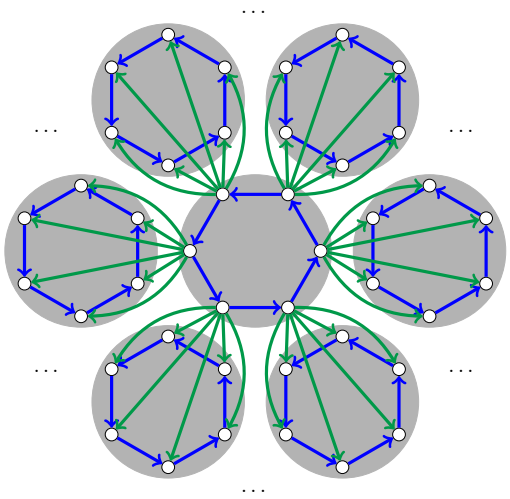


Figure 1 The figure shows part of a tournament T. The colors c_1 and c_2 are shown in blue and green, respectively. Also, the parts of the partition $\mathcal{Q}_1$ are highlighted in gray. Note that only green edges, which are outgoing from the middle part, are shown.

The proof is very similar to the proof of Lemma 3.2.

Proof. Let $\ell := |\mathcal{W}|$ and suppose $\mathcal{W} = \{Q_1, \dots, Q_\ell\}$. For every $i \in [\ell]$ pick an element $w_i \in Q_i$ such that $(v, w_i) \in E(T)$ and $\mathrm{md}_{\mathcal{Q}}(v, w_i) \leq k$. We define $W := \{w_1, \dots, w_\ell\}$. Then there is some $w \in W$ such that

$$|\{w' \in W \mid (w', w) \in E(T)\}| \geq \frac{\ell - 1}{2},$$

because the induced subtournament $T[W]$ has a vertex of in-degree at least $(\ell - 1)/2$.

Since $\mathrm{md}_{\mathcal{Q}}(v, w) \leq k$ and $(v, w') \in E(T)$ for all $w' \in W$, it follows that

$$|\{w' \in W \mid (w', w) \in E(T)\}| \leq k.$$

Thus $\frac{\ell-1}{2} \leq k$, which implies that $|\mathcal{W}| = \ell \leq 2k + 1$. ◀

Now, suppose we color all edges (v, w) of T with $\mathrm{md}(v, w) \leq \mathrm{tww}(T)$ using the color $c_1 = \mathsf{blue}$ (see Figure 1). Let $\mathcal{Q}_1$ be the partition into the (weakly) connected components of the graph induced by the blue edges and suppose that $\mathcal{Q}_1$ is non-trivial. We can compute isomorphisms between the different parts of $\mathcal{Q}_1$ using the algorithm from [1]. Next, let us color all cross-cluster edges $(v, w) \in E(T)$ with $\mathrm{md}_{\mathcal{Q}}(v, w) \leq \mathrm{tww}(T)$ using the color $c_2 = \mathsf{green}$. Then every vertex has outgoing green edges to at most $2\,\mathrm{tww}(T) + 1$ other parts of $\mathcal{Q}_1$ (see Lemma 3.4). However, since a vertex may have an unbounded number of green neighbors in a single part, the out-degree of the graph induced by the green edges may be unbounded. So it is not possible to use the algorithm from [1] as a black-box on the components induced by blue and green edges. Luckily, the methods used in [1] can be extended to work even in this more general setting (see Section 4). So if the graph induced by the blue and green edges is connected, then we are again done. Otherwise, we let $\mathcal{Q}_2$ denote the partition into (weakly) connected components of the graph induced by the blue and green edges. Now, we can continue in the same fashion identifying colors $c_3, c_4, \dots$ and corresponding partitions $\mathcal{Q}_3, \mathcal{Q}_4, \dots$ until the graph induced by all edges of colors $c_1, \dots, c_\ell$ is eventually connected.

Below, we provide a lemma that computes the corresponding sequence of partitions and edge colors using 2-WL. To state the lemma in its cleanest form, we restrict our attention to tournaments that are 2-WL-homogeneous. Recall that a tournament T is 2-WL-homogeneous if for all $v, w \in V(T)$ is holds that $\chi^{2,T}(v,v) = \chi^{2,T}(w,w)$.

We also require another piece of notation. For a directed graph G and a set of colors $C \subseteq \{\chi^{2,G}(v,w) \mid (v,w) \in E(G)\}$ we write $G[C]$ for the directed graph with vertex set $V(G[C]) \coloneqq V(G)$ and edge set

$$E(G[C]) \coloneqq \{(v,w) \in E(G) \mid \chi^{2,G} \in C\}.$$

To prove Lemma 3.6 we need the following lemma about the connected components of the graphs $G[C]$.

▶ **Lemma 3.5.** *Let G be a 2-WL-homogeneous graph, and let C a set of colors in the range of $\chi^{2,G}$. Then the weakly connected components of $G[C]$ equal the strongly connected components of $G[C]$.*

▶ **Lemma 3.6.** *Let T be a 2-WL-homogeneous tournament of twin width $\mathrm{tww}(T) \leq k$. Then there is a sequence of partitions $\{\{v\} \mid v \in V(T)\} = \mathcal{Q}_0, \dots, \mathcal{Q}_\ell = \{V(T)\}$ of $V(T)$ where $\mathcal{Q}_{i-1}$ refines $\mathcal{Q}_i$ for all i, and a sequence of colors $c_1, \dots, c_\ell$ in the range of $\chi^{2,T}$ such that*

(1) *$\mathcal{Q}_i$ is the partition into the strongly connected components of $T[\{c_1, \dots, c_i\}]$ for every $i \in [\ell]$, and*

(2) *for every $i \in [\ell]$ and every $v \in V(T)$ it holds that*

$$\left|\left\{Q \in \mathcal{Q}_{i-1} \;\middle|\; \exists w \in Q \colon \chi^{2,T}(v,w) = c_i\right\}\right| \leq 2k+1.$$

Moreover, there is a polynomial-time algorithm that, given a tournament T and an integer $k \geq 1$, computes the desired sequences $\mathcal{Q}_0, \dots, \mathcal{Q}_\ell$ and $c_1, \dots, c_\ell$ or concludes that $\mathrm{tww}(T) > k$.

Proof. We set $\mathcal{Q}_0 = \{\{v\} \mid v \in V(T)\}$ and inductively define a sequence of partitions and colors as follows. Let $i \geq 0$ and suppose we already defined partitions $\mathcal{Q}_0 \prec \cdots \prec \mathcal{Q}_i$ and colors $c_1, \dots, c_i$. If $\mathcal{Q}_i = \{V(T)\}$, we set $\ell \coloneqq i$ and complete both sequences. Otherwise, there is a cross-cluster edge (v_{i+1}, w_{i+1}) with respect to $\mathcal{Q}_i$ such that $\mathrm{md}_{\mathcal{Q}_i}(v_{i+1}, w_{i+1}) \leq k$ by Lemma 3.3. We set $c_{i+1} \coloneqq \chi^{2,T}(v_{i+1}, w_{i+1})$ and define $\mathcal{Q}_{i+1}$ to be the set of weakly connected components of $T[c_1, \dots, c_{i+1}]$. By Lemma 3.5, these are also the strongly connected components.

First observe that $\mathcal{Q}_i \prec \mathcal{Q}_{i+1}$ since (v_{i+1}, w_{i+1}) is a cross-cluster edge with respect to $\mathcal{Q}_i$, and v_{i+1}, w_{i+1} are contained in the same part of $\mathcal{Q}_{i+1}$ by Lemma 3.5. Also, Property 1 is satisfied by definition. For Property 2 note that every edge $(v,w) \in E(T)$ such that $\chi^{2,G}(v,w) = c_{i+1}$ is a cross-cluster edge with respect to $\mathcal{Q}_i$. So Property 2 follows directly from Lemma 3.4.

Finally, it is clear from the description above that $\mathcal{Q}_0 \prec \cdots \prec \mathcal{Q}_\ell$ and $c_1, \dots, c_\ell$ can be computed in polynomial time, or we conclude that $\mathrm{tww}(T) > k$. ◀

4 The Isomorphism Algorithm

Based on the structural insights summarized in Lemma 3.6, we now design an isomorphism test for tournaments of small twin width.

The strategy of our algorithm is the following. We are given two tournaments T_1 and T_2, and we want to compute $\mathrm{Iso}(T_1, T_2)$. First, we reduce to the case where both T_1 and T_2 are 2-WL-homogeneous.

Towards this end, we start by applying 2-WL and, for $j = 1, 2$, compute the coloring χ^{2,T_j}. If 2-WL distinguishes the two tournaments, we can immediately conclude that they are non-isomorphic and return $\mathrm{Iso}(T_1, T_2) = \emptyset$.

So suppose that 2-WL does not distinguish the tournaments. Then T_1 is 2-WL-homogeneous if and only if T_2 is 2-WL-homogeneous.

If the T_j are not 2-WL-homogeneous, we rely on the following standard argument. Let $c_1, \dots, c_p$ be the vertex colors. For $i \in [p]$ and $j = 1, 2$, let $P_{j,i}$ be the set of all $v \in V(T_j)$ such that $\chi^{2,T_j}(v, v) = c_i$. We recursively compute the sets $\Lambda_i := \mathrm{Iso}(T_1[P_{1,i}], T_2[P_{2,i}])$ for all $i \in [p]$. Note that this is possible since $\mathrm{tww}(T_1[P_{1,i}]) \leq \mathrm{tww}(T_1)$ and $\mathrm{tww}(T_2[P_{2,i}]) \leq \mathrm{tww}(T_2)$ for all $i \in [p]$ by Lemma 2.1. If there is some $i \in [p]$ such that $\Lambda_i = \emptyset$, then T_1 and T_2 are non-isomorphic, and we return $\mathrm{Iso}(T_1, T_2) = \emptyset$. Otherwise, the set Λ_i is a coset of $\Gamma_i := \mathrm{Aut}(T_1[P_{1,i}])$ for all $i \in [p]$, i.e., $\Lambda_i = \Gamma_i\theta_i$ for some bijection $\theta_i \colon P_{1,i} \to P_{2,i}$. As the automorphism group of a tournament, Γ_i is solvable (see Theorem 2.3). Moreover, since the color classes $P_{1,i}$ are invariant under automorphisms of T_1, the automorphism group $\Gamma := \mathrm{Aut}(T_1)$ is a subgroup of the direct product $\prod_i \Gamma_i$, which is also a solvable group. Also, $\mathrm{Iso}(T_1, T_2) \subseteq \Gamma\theta$ where $\theta \colon V(T_1) \to V(T_2)$ is the unique bijection defined via $\theta(v) := \theta_i(v)$ for all $v \in V(T_1)$, where $i \in [p]$ is the unique index such that $v \in P_{1,i}$. So $\mathrm{Iso}(T_1, T_2) = \mathrm{Iso}_{\Gamma\theta}(T_1, T_2)$ can be computed in polynomial time using Theorem 2.5.

So we may assume that T_1 and T_2 are 2-WL-homogeneous. In this case, we apply Lemma 3.6 and obtain colors $c_1, \dots, c_\ell$ and, for $j = 1, 2$, a partition sequence $\{\{v\} \mid v \in V(T_j)\} = \mathcal{Q}_{j,0}, \dots, \mathcal{Q}_{j,\ell} = \{V(T_j)\}$ where $\mathcal{Q}_{j,i-1}$ refines $\mathcal{Q}_{j,i}$ for all $i \in [\ell]$.

Now, we iteratively compute for $i = 0, \dots, \ell$ the sets $\mathrm{Iso}(T_j[Q], T_{j'}[Q'])$ for all $j, j' \in \{1, 2\}$ and all $Q \in \mathcal{Q}_{j,i}$ and $Q' \in \mathcal{Q}_{j',i}$. For $i = 0$ this is trivial since all parts have size 1. So suppose $i > 0$ and consider some elements $j, j' \in \{1, 2\}$ and $Q \in \mathcal{Q}_{j,i}$, $Q' \in \mathcal{Q}_{j',i}$. For simplicity, let us assume that $j = 1$ and $j' = 2$. Our goal is to compute $\mathrm{Iso}(T_1[Q], T_2[Q'])$. To do so, we exploit that we already computed all isomorphisms between all pairs of subgraphs of $T_1[Q]$ and $T_2[Q']$ induced by sets $R \in \mathcal{Q}_{1,i-1} \cup \mathcal{Q}_{2,i-1}$ and for which $R \subseteq Q$ or $R \subseteq Q'$. The next lemma describes the key subroutine of the main algorithm which achieves this goal. Note that, on the last level ℓ, we compute the set $\mathrm{Iso}(T_1, T_2)$ since $\mathcal{Q}_{j,\ell} = \{V(T_j)\}$ for both $j \in \{1, 2\}$.

To state the lemma, we need additional terminology. Let $T = (V, E, \lambda)$ be an arc-colored tournament. A partition $\mathcal{Q}$ of V is *λ-definable* if there is a set of colors $C \subseteq \{\lambda(v, w) \mid v = w \vee (v, w) \in E\}$ such that

$$v \sim_{\mathcal{Q}} w \quad \iff \quad \lambda(v, w) \in C$$

for all $v, w \in V$ such that $v = w$ or $(v, w) \in E$. We also say that *$\mathcal{Q}$ is λ-defined by C*. If $\mathcal{Q}$ is λ-defined by C then we can partition the colors in the range of λ into the colors in C, which we call *intra-cluster* colors, and the remaining colors, which we call *cross-cluster* colors. Note that if a color c is intra-cluster, then for all $(v, w) \in E$ with $\lambda(v, w) = c$ it holds that $v, w \in Q$ for some $Q \in \mathcal{Q}$, and if c is cross-cluster, then for all $(v, w) \in E$ with $\lambda(v, w) = c$ it holds that (v, w) is a cross-cluster edge, that is, $v \in Q$ and $w \in Q'$ for distinct $Q, Q' \in \mathcal{Q}$.

▶ **Lemma 4.1.** *There is an algorithm that, given*

(A) *an integer $d \geq 1$;*
(B) *two arc-colored tournaments $T_1 = (V_1, E_1, \lambda_1)$ and $T_2 = (V_2, E_2, \lambda_2)$;*
(C) *a set of colors C and for $j = 1, 2$ a partition $\mathcal{Q}_j$ of V_j that is λ_j-defined by C;*
(D) *a color c^* that is cross-cluster with respect to $\mathcal{Q}_j$ for $j = 1, 2$ and*

- *for every $v \in V_j$ it holds that*

$$\left|\left\{Q \in \mathcal{Q}_j \;\middle|\; \exists w \in Q \colon (v, w) \in E_j \wedge \lambda_j(v, w) = c^*\right\}\right| \leq d;$$

- *for*

$$F_j \coloneqq \left\{(Q,Q') \in \mathcal{Q}_j^2 \;\middle|\; Q \neq Q', \exists w \in Q, w' \in Q' \colon (w,w') \in E_j \wedge \lambda_j(w,w') = c^*\right\}$$

the directed graph $G_j = (\mathcal{Q}_j, F_j)$ *is strongly connected;*

(E) $\mathrm{Iso}\left(T_j[Q], T_{j'}[Q']\right)$ *for every* $j, j' \in \{1,2\}$ *and every* $Q \in \mathcal{Q}_j$, $Q' \in \mathcal{Q}_{j'}$,

computes $\mathrm{Iso}(T_1, T_2)$ *in time* $d^{O(\log d)} \cdot n^{O(1)}$.

The proof builds on the algorithmic ideas presented in [1]. Since it is quite lengthy and technical, we only present a rough idea of the proof; the details can be found in the full version.

Proof Idea. The algorithm fixes an arbitrary vertex $r_1 \in V_1$ and for every $r_2 \in V_2$ computes the set $\mathrm{Iso}((T_1, r_1), (T_2, r_2))$ of all isomorphisms $\varphi \in \mathrm{Iso}(T_1, T_2)$ such that $\varphi(r_1) = r_2$. Observe that $\mathrm{Iso}(T_1, T_2)$ is the union over all these sets.

The central idea is to iteratively compute larger and larger sets $W_{1,i} \subseteq V_1$ and $W_{2,i} \subseteq V_2$ such that

(I.1) there is some $\mathcal{W}_{j,i} \subseteq \mathcal{Q}_j$ such that $W_{j,i} = \bigcup_{Q \in \mathcal{W}_{j,i}} Q$, and

(I.2) $\varphi(W_{1,i}) = W_{2,i}$ for every $\varphi \in \mathrm{Iso}((T_1, r_1), (T_2, r_2))$

and compute the set $\mathrm{Iso}((T_1[W_{1,i}], r_1), (T_2[W_{2,i}], r_2))$. Initially, we set $W_{1,0} \coloneqq R_1$ and $W_{2,0} \coloneqq R_2$ where R_1 and R_2 are the unique parts of $\mathcal{Q}_1$ and $\mathcal{Q}_2$ containing r_1 and r_2, respectively. Note that the set $\mathrm{Iso}((T_1[W_{1,0}], r_1), (T_2[W_{2,0}], r_2))$ can easily be computed from Item E.

Now suppose we already computed $\mathrm{Iso}((T_1[W_{1,i}], r_1), (T_2[W_{2,i}], r_2))$ for some $W_{1,i} \subseteq V_1$ and $W_{2,i} \subseteq V_2$ satisfying (I.1) and (I.2). Consider a vertex $u \in W_{1,i}$ that has an outgoing edge of color c^* to a vertex outside of $W_{1,i}$. Note that, if $W_{1,i} \neq V_1$, such a vertex exists by the second part of Item D. We call such a vertex a *boundary vertex*. Let $U_{1,i}$ denote the set of boundary vertices. For a boundary vertex $u \in U_{1,i}$ let $\mathcal{L}^u_{1,i+1}$ denote the set of all parts $Q \in \mathcal{Q}_1$ that are outside of $W_{1,i}$ and contain a vertex $v \in Q$ such that $(u,v) \in E_1$ and $\lambda_1(u,v) = c^*$. Also, we define $L^u_{1,i+1} \coloneqq \bigcup_{Q \in \mathcal{L}^u_{1,i+1}} Q$. A visualization is given in Figure 2. Note that $|\mathcal{L}^u_{1,i+1}| \leq d$ by the first part of Item D.

For every boundary vertex $u \in U_{1,i}$ we construct an isomorphism-invariant tournament $\widetilde{T}^u_{1,i+1}$ with vertex set $\mathcal{L}^u_{1,i+1}$. Roughly speaking, for two distinct $Q, Q' \in \mathcal{L}^u_{1,i+1}$, in order to decide whether to add (Q,Q') or (Q',Q) to the edge set of $\widetilde{T}^u_{1,i+1}$, we take a majority vote on the edges between Q and Q' in T_1 (if there is a tie, we need to invoke further rules; see the full version for details).

Since $|\mathcal{L}^u_{1,i+1}| \leq d$, the automorphism group of $\widetilde{T}^u_{1,i+1}$ can be computed in time $d^{O(\log d)}$ by Theorem 2.4. Also, for each $Q \in \mathcal{L}^u_{1,i+1}$, the automorphism group of $T[Q]$ is given as part of the input (see Item E). By taking a wreath product, we obtain a solvable permutation group $\Gamma \leq \mathrm{Sym}(L^u_{1,i+1})$ such that $\mathrm{Aut}(T_1[L^u_{1,i+1}]) \leq \Gamma$. Using Theorem 2.5, this allows us to compute $\mathrm{Aut}(T_1[L^u_{1,i+1}])$ and, more generally, compute the isomorphism sets between the corresponding subgraphs for different boundary vertices $u, u' \in V_1 \cup V_2$.

Now, we extend $W_{1,i}$ by all sets $L^u_{1,i+1}$, $u \in U_{1,i}$, to obtain the next layer $W_{1,i+1}$ (actually, for technical reasons, the final proof proceeds in a slightly different manner); the set $W_{2,i+1}$ is defined analogously. To compute the desired isomorphism set, we proceed as follows. For simplicity, first suppose that all sets $L^u_{1,i+1}$, $u \in U_{1,i}$, are pairwise disjoint. Then, we can again take a wreath product to obtain a solvable permutation group $\Delta \leq \mathrm{Sym}(W_{1,i+1} \setminus W_{1,i})$ such that $\mathrm{Aut}(T_1[W_{1,i+1}], r_1) \leq \mathrm{Aut}(T_1[W_{1,i}], r_1) \times \Delta$. Using Theorem 2.5, this allows us to compute $\mathrm{Aut}(T_1[W_{1,i+1}], r_1)$. Similarly, we can also compute the set $\mathrm{Iso}((T_1[W_{1,i+1}], r_1), (T_2[W_{2,i+1}], r_2))$.

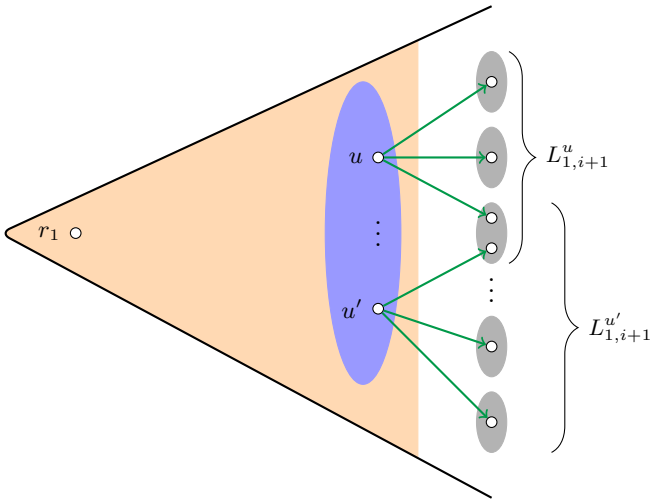


Figure 2 The figure shows the sets $W_{1,i}$ (orange), $U_{1,i}$ (blue) and $L^u_{1,i+1}$ computed in the proof sketch of Lemme 4.1. The color c^* is shown in green and gray regions depict parts of the partition $\mathcal{Q}_1$.

To cover the case that not all sets $L^u_{1,i+1}$, $u \in U_{1,i}$, are pairwise disjoint, we use the following trick. For each vertex $v \in W_{1,i+1} \setminus W_{1,i}$ let $\mu(v)$ denote the number of boundary vertices $u \in U_{1,i}$ such that $v \in L^u_{1,i+1}$. We replace each $v \in W_{1,i+1} \setminus W_{1,i}$ with $\mu(v)$ copies of the vertex; each copy is associated with one of the corresponding boundary vertices. Now, we can proceed as in the previous case to compute the set of isomorphisms. Afterwards, we rely on group-theoretic algorithms from [35] to "merge" the different copies of the same vertex again to finally obtain the group $\mathrm{Iso}((T_1[W_{1,i+1}], r_1), (T_2[W_{2,i+1}], r_2))$.

Note that the second part of Item D guarantees that we can continue this process until eventually $W_{1,i} = V_1$ and $W_{2,i} = V_2$, at which point we have computed the desired set $\mathrm{Iso}((T_1, r_1), (T_2, r_2))$. ◀

Building on the subroutine from Lemma 4.1, we can now design an isomorphism test for tournaments of bounded twin width following the outline given above.

▶ **Theorem 4.2.** *There is an algorithm that, given two tournaments T_1 and T_2 and an integer $k \geq 1$, either concludes that $\mathrm{tww}(T_1) > k$ or computes $\mathrm{Iso}(T_1, T_2)$ in time $k^{O(\log k)} \cdot n^{O(1)}$.*

5 The WL-Dimension of Tournaments of Bounded Twin Width

In this section, we prove Theorem 1.2. To prove that the WL algorithm on its own is unable to determine isomorphisms between tournaments of bounded twin width, we adapt a construction of Cai, Fürer and Immerman [12]. Towards this end, we first describe a construction of directed graphs with large WL dimension, and then argue how to translate those graphs into tournaments while preserving their WL dimension.

In the following, let G be a connected, 3-regular (undirected) graph. Let G^{red} denote the structure obtained from G be replacing every edge with a red edge and let $t := \mathrm{tww}(G^{\mathsf{red}})$.

Let us remark at this point that the Cai-Fürer-Immerman construction [12] replaces each vertex in G with a certain gadget, and those gadgets are connected along the edges of G. In order to bound the twin width of the resulting graph, we start with a graph G for which the

twin width of G^{red} is bounded.[1] This is because the connections between gadgets are not homogeneous, so when contracting all gadgets in a contraction sequence, we obtain precisely the graph G^{red} which allows to "complete" the contraction sequence using that G^{red} has bounded twin width.

Now, let us formally describe the construction of tournaments with large WL dimension. By Lemma 2.2 there is some linear order $<$ on $V(G)$ such that $\mathrm{tww}(G^{\mathsf{red}}, <) = t$. Also, let $\vec{G}$ be an arbitrary orientation of G.

Recall that for every $v \in V(G)$ we denote by $E_+(v)$ the set of outgoing edges and $E_-(v)$ the set of incoming edges in $\vec{G}$. Also, we write $E(v)$ to denote the set of incident (undirected) edges in G. For $a \in \mathbb{Z}_3$ we define

$$M_a(v) \coloneqq \left\{ f \colon E(v) \to \mathbb{Z}_3 \;\middle|\; \sum_{(v,w)\in E_+(v)} f(\{v,w\}) - \sum_{(w,v)\in E_-(v)} f(\{v,w\}) = a \pmod 3 \right\}$$

We also define $F_a(v)$ to contain all pairs $(f,g) \in (M_a(v))^2$ such that, for the minimal element $w \in N_G(v)$ (with respect to $<$) such that $f(vw) \neq g(vw)$, it holds that

$$f(vw) + 1 = g(vw) \pmod 3.$$

Observe that, for every distinct $f, g \in M_a(v)$, either $(f,g) \in F_a(v)$ or $(g,f) \in F_a(v)$.

Let $\alpha \colon V(G) \to \mathbb{Z}_3$ be a function. We define the graph $\mathsf{CFI}_3(\vec{G}, <, \alpha)$ with vertex set

$$V(\mathsf{CFI}_3(\vec{G}, <, \alpha)) \coloneqq \bigcup_{v \in V(G)} \{v\} \times M_{\alpha(v)}(v)$$

and edge set

$$\begin{aligned} E(\mathsf{CFI}_3(\vec{G}, <, \alpha)) \coloneqq \; & \left\{ \{(v,f)(w,g)\} \;\middle|\; vw \in E(G) \wedge f(vw) = g(vw) \right\} \\ & \cup \left\{ ((v,f)(v,g)) \;\middle|\; (f,g) \in F_{\alpha(v)}(v) \right\}. \end{aligned}$$

Observe that $\mathsf{CFI}_3(\vec{G}, <, \alpha)$ is a *mixed graph*, i.e., it contains both directed and undirected edges. Also, we color the vertices of $\mathsf{CFI}_3(\vec{G}, <, \alpha)$ using the coloring $\lambda \colon V(\mathsf{CFI}_3(\vec{G}, <, \alpha)) \to C$ defined via $\lambda(v,f) \coloneqq v$ for all $(v,f) \in V(\mathsf{CFI}_3(\vec{G}, \alpha))$, i.e., each set $M_{\alpha(v)}(v)$ forms a color class under λ.

Now fix an arbitrary vertex $u_0 \in V(G)$. For every $i \in \mathbb{Z}_3$ we define the mapping $\alpha_i \colon V(G) \to \mathbb{Z}_3$ via $\alpha_i(u_0) \coloneqq i$ and $\alpha_i(w) \coloneqq 0$ for all $w \in V(G) \setminus \{u_0\}$.

▶ **Lemma 5.1.** $\mathsf{CFI}_3(\vec{G}, <, \alpha_0) \ncong \mathsf{CFI}_3(\vec{G}, <, \alpha_1)$.

Now, we analyse the WL algorithm on the graphs $\mathsf{CFI}_3(\vec{G}, <, \alpha)$ for different functions $\alpha \colon V(G) \to \mathbb{Z}_3$. We write $\mathrm{tw}(G)$ to denote the tree width of G.

▶ **Lemma 5.2.** *Let k be an integer such that $\mathrm{tw}(G) \geq k+1$. Also let $\alpha, \beta \colon V(G) \to \mathbb{Z}_3$ be two functions. Then $\mathsf{CFI}_3(\vec{G}, <, \alpha) \simeq_k \mathsf{CFI}_3(\vec{G}, <, \beta)$.*

Together, Lemmas 5.1 and 5.2 provide pairs of non-isomorphic graphs that are not distinguished by k-WL assuming $\mathrm{tw}(G) > k$. Next, we argue how to turn these graphs into tournaments.

[1] We remark that, since G has maximum degree 3, the twin width of G^{red} is actually bounded in the twin width of G. However, we feel it is more convenient to directly bound the twin width of G^{red}.

We define the tournament $T = T(\vec{G}, <, \alpha)$ with vertex set

$$V(T) \coloneqq V(\mathsf{CFI}_3(\vec{G}, <, \alpha)) = \bigcup_{v \in V(G)} \{v\} \times M_{\alpha(v)}(v)$$

and edge set

$$\begin{aligned} E(T) \coloneqq & \left\{((v,f)(v,g)) \;\middle|\; (f,g) \in F_{\alpha(v)}(v)\right\} \\ & \cup \left\{((v,f)(w,g)) \;\middle|\; v < w \wedge \{(v,f)(w,g)\} \notin E(\mathsf{CFI}_3(\vec{G}, <, \alpha))\right\} \\ & \cup \left\{((w,g)(v,f)) \;\middle|\; v < w \wedge \{(v,f)(w,g)\} \in E(\mathsf{CFI}_3(\vec{G}, <, \alpha))\right\}. \end{aligned}$$

It can be shown that the relevant properties are preserved by this translation.

▶ **Lemma 5.3.** *Let $\alpha, \beta \colon V(G) \to \mathbb{Z}_3$ be two functions. Then $T(\vec{G}, <, \alpha) \cong T(\vec{G}, <, \beta)$ if and only if $\mathsf{CFI}_3(\vec{G}, <, \alpha) \cong \mathsf{CFI}_3(\vec{G}, <, \beta)$.*

▶ **Lemma 5.4.** *Let $k \geq 2$ be an integer such that $\mathrm{tw}(G) \geq k+1$. Also let $\alpha, \beta \colon V(G) \to \mathbb{Z}_3$ be two functions. Then $T(\vec{G}, <, \alpha) \simeq_k T(\vec{G}, <, \beta)$.*

To prove Theorem 1.2, we also need to bound the twin width of the resulting graph. Recall that $t \coloneqq \mathrm{tww}(G^{\mathsf{red}})$ where G^{red} denotes the version of G where every edge is red.

▶ **Lemma 5.5.** *For every function $\alpha \colon V(G) \to \mathbb{Z}_3$ it holds that $\mathrm{tww}(T(\vec{G}, <, \alpha)) \leq \max(35, t)$.*

Proof. Throughout the proof, we define $M(v) \coloneqq M_{\alpha(v)}(v)$ for every $v \in V(G)$. Since G is 3-regular, we have that $|M(v)| = 9$ for every $v \in V(G)$. So $|V(T(\vec{G}, <, \alpha))| = 9 \cdot |V(G)|$. Also note that $(M(v), M(w))$ is homogeneous for all distinct $v, w \in V(G)$ such that $\{v, w\} \notin E(G)$.

We construct a partial contraction sequence as follows. Let $n \coloneqq |V(G)|$. We define $\mathcal{P}_1, \ldots, \mathcal{P}_{8n+1}$ arbitrarily such that $\mathcal{P}_{8n+1} = \{M(v) \mid v \in V(G)\}$. Since G is 3-regular and $|M(v)| = 9$ for every $v \in V(G)$, we conclude that $(T(\vec{G}, <, \alpha))/\mathcal{P}_i$ has red degree at most $4 \cdot 9 - 1 = 35$ for every $i \in [8n+1]$. Now observe that

$$\mathrm{tww}((T(\vec{G}, <, \alpha))/\mathcal{P}_{8n+1}) \leq \mathrm{tww}(G^{\mathsf{red}}, <) = t.$$

It follows that $\mathrm{tww}(T(\vec{G}, <, \alpha)) \leq \max(35, t)$ as desired. ◀

With Lemma 5.5 in hand, we apply the construction $T(\vec{G}, <, \alpha)$ to a 3-regular base graph G which has tree width linear in the number of vertices, but the twin width of G^{red} is bounded. The existence of such graphs has already been observed in [5]. More precisely, the following theorem follows from combining the arguments from [5, Lemma 5.1] and the results from [4, 34].

▶ **Theorem 5.6.** *There is a family of 3-regular graphs $(G_n)_{n \geq 1}$ such that $|V(G_n)| = O(n)$, $\mathrm{tww}(G_n^{\mathsf{red}}) \leq 6$ (where G_n^{red} denotes the version of G_n where all edges are turned into red edges), and $\mathrm{tw}(G_n) \geq n$ for every $n \geq 1$.*

Now, we are ready to give a proof for Theorem 1.2.

▶ **Theorem 5.7.** *For every $k \geq 2$ there are non-isomorphic tournaments T_k and T_k' such that*
(1) *$|V(T_k)| = |V(T_k')| = O(k)$,*
(2) *$\mathrm{tww}(T_k) \leq 35$ and $\mathrm{tww}(T_k') \leq 35$, and*
(3) *$T_k \simeq_k T_k'$.*

Proof. Let $k \geq 2$. Let G_{k+1} be the 3-regular graph obtained from Theorem 5.6. Note that $\mathrm{tw}(G_{k+1}) \geq k+1$.

We also fix an arbitrary orientation $\vec{G}_{k+1}$ of G_{k+1} and let G^{red}_{k+1} denote the version of G_{k+1} where every edge is replaced by a red edge. We have $t := \mathrm{tww}(G^{\mathsf{red}}_{k+1}) \leq 6$ by Theorem 5.6. By Lemma 2.2 there is some linear order $<$ on $V(G_{k+1})$ such that $\mathrm{tww}(G^{\mathsf{red}}_{k+1}, <) \leq t \leq 6$.

Now fix an arbitrary $u_0 \in V(G_{k+1})$. For $p \in \mathbb{Z}_3$ we define the mapping $\alpha_p \colon V(G_{k+1}) \to \mathbb{Z}_3$ via $\alpha_p(u_0) = p$ and $\alpha_p(w) = 0$ for all $w \in V(G_{k+1}) \setminus \{u_0\}$. We define $T_k := T(\vec{G}_{k+1}, <, \alpha_0)$ and $T'_k := T(\vec{G}_{k+1}, <, \alpha_1)$. We have $|V(T_k)| = |V(T'_k)| = 9 \cdot |V(G_{k+1})| = O(k)$. Also, $\mathrm{tww}(T_k) \leq 35$ and $\mathrm{tww}(T'_k) \leq 35$ by Lemma 5.5. Finally, $T_k \ncong T'_k$ by Lemmas 5.1 and 5.3, and $T_k \simeq_k T'_k$ by Lemma 5.4. ◀

6 Twin Width is Smaller Than Other Widths

In this section, we compare twin width with other natural width parameters of tournaments. If f, g are mappings from (directed) graphs to the natural numbers, we say that f is *functionally smaller* than g on a class $\mathcal{C}$ of graphs if for every k there is a k' such that for all graphs $G \in \mathcal{C}$, if $g(G) \leq k$ then $f(G) \leq k'$. We write $f \preceq_{\mathcal{C}} g$ to denote that f is functionally smaller than g on $\mathcal{C}$. We omit the subscript $_{\mathcal{C}}$ if $\mathcal{C}$ is the class of all digraphs.

Natural width measures for directed graphs are *cut width*, *directed path width*, *directed tree width*, and *clique width*. On the class of tournaments twin width turns out to be functionally smaller than all of these. For clique width, it has already been shown in [11] that twin width is functionally smaller than clique width on undirected graphs; the proof easily extends to arbitrary binary relational structures and hence to tournaments.

We start by giving definitions for the other width measures. Let G be a digraph. For a linear order $\leq$ on $V(G)$ and a vertex $v \in V(G)$, we let $S_{\leq}(v) := \{w \in V(G) \mid w \leq v\}$ be the set of all vertices smaller than or equal to v in $\leq$. Let $s_{\leq}(v) := |E_G(S_{\leq}(v), V(G) \setminus S_{\leq}(v))|$ be the number of edges from $S_{\leq}(v)$ to its complements. The *width* of $\leq$ is $\max_{v \in V(G)} s_{\leq}(v)$, and the *cut width* $\mathrm{ctw}(G)$ is the minimum over the width of all linear orders of $V(G)$.

A *directed path decomposition* of a digraph G is a mapping $\beta \colon [p] \to 2^{V(G)}$, for some $p \in \mathbb{N}$, such that for every vertex $v \in V(G)$ there are $\ell, r \in [p]$ such that $v \in \beta(t) \iff \ell \leq t \leq r$, and for all edges $(v, w) \in E(G)$ there are $\ell, r \in [p]$ with $\ell \leq r$ such that $v \in \beta(r)$ and $w \in \beta(\ell)$. The sets $\beta(t)$, $t \in [p]$, are the *bags* of the decomposition. The *width* of the decomposition is $\max_{t \in [p]} |\beta(t)| - 1$, and the *directed path width* $\mathrm{dpw}(G)$ is the minimum width of a directed path decomposition of G.

A digraph R is a *rooted directed tree* if there is a vertex $r_0 \in V(R)$ such that for every $t \in V(R)$ there is a unique directed walk from r_0 to t. Note that every rooted directed tree can be obtained from an undirected tree by selecting a root r_0 and directing all edges away from the root. For $t \in V(R)$ we denote by R_t the unique induced subgraph of R rooted at t.

Let G be a digraph. A *directed tree decomposition* of G is a triple (R, β, γ) where R is a rooted directed tree, $\beta \colon V(R) \to 2^{V(G)}$ and $\gamma \colon E(R) \to 2^{V(G)}$ such that

(D.1) $\{\beta(t) \mid t \in V(R)\}$ is a partition of $V(G)$, and

(D.2) for every $(s, t) \in E(R)$ the set $\gamma(s, t)$ is a hitting set for all directed walks that start and end in $\beta(R_t) := \bigcup_{t' \in V(R_t)} \beta(t')$ and contain a vertex outside of $\beta(R_t)$.

For $t \in V(R)$ we define $\Gamma(t) := \beta(t) \cup \bigcup_{(s,s') \in E(t)} \gamma(s, s')$ where $E(t)$ denotes the set of edges incident to t. The *width* of (R, β, γ) is defined as

$$\mathrm{width}(R, \beta, \gamma) := \max_{t \in V(R)} |\Gamma(t)| - 1.$$

The *directed tree width* $\mathrm{dtw}(G)$ is the minimum width of a directed tree decomposition of G.

Let us first recall the following well-known inequalities.

▶ **Proposition 6.1.** *For all digraphs G, it holds that* $\mathrm{dtw}(G) \leq \mathrm{dpw}(G) \leq \mathrm{ctw}(G)$.

The proposition implies that dtw $\precsim$ dpw $\precsim$ ctw. It can be shown that tww $\not\precsim$ ctw and hence tww $\not\precsim$ dpw and tww $\not\precsim$ dtw on the class of all digraphs. In contrast, it turns out that on the class of tournaments, twin width is functionally smaller than directed tree width. Actually, this even holds for the larger class of semi-complete graphs. A digraph G is *semi-complete* if for all distinct $v, w \in V(G)$ at least one of the pairs $(v, w), (w, v)$ is an edge. Note that every tournament is semi-complete.

▶ **Theorem 6.2** ([28, Proposition 5])**.** *Let G be a semi-complete graph. Then*

$$\mathrm{dpw}(G) \leq 4(\mathrm{dtw}(G) + 2)^2 + 7(\mathrm{dtw}(G) + 2) - 1.$$

▶ **Theorem 6.3.** *Let G be a semi-complete graph. Then* $\mathrm{tww}(G) \leq \mathrm{dpw}(G)$.

In combination, we get that $\mathrm{tww}(G) \leq 4(\mathrm{dtw}(G) + 2)^2 + 7(\mathrm{dtw}(G) + 2) - 1$ for every semi-complete graph G. In particular

$$\text{tww} \precsim_{\mathcal{S}} \text{dtw}, \tag{2}$$

where $\mathcal{S}$ denotes the class of all semi-complete digraphs. This inequality is strict even on tournaments, that is, dtw $\not\precsim_{\mathcal{T}}$ tww where $\mathcal{T}$ denotes the class of all tournaments.

7 Conclusion

We prove that the isomorphism problem for classes of tournaments of bounded (or slowly growing) twin width is in polynomial time. Many algorithmic problems that can be solved efficiently on (classes of) tournaments can also be solved efficiently on (corresponding classes of) semi-complete graphs, that is, directed graphs where for every pair (v, w) of vertices at least one of the pairs $(v, w), (w, v)$ is an edge (see, e.g., [39]). Contrary to this, we remark that isomorphism of semi-complete graphs of bounded twin width is GI-complete: we can reduce isomorphism of oriented graphs to isomorphism of semi-complete graphs by replacing each non-edge by a bidirectional edge. This reduction preserves twin with.

Classes of tournaments of bounded twin width are precisely the classes that are considered to be structurally sparse. Formally, these are the classes that are monadically dependent, which means that all set systems definable over the tournaments in such a class have bounded VC dimension. The most natural set systems definable within a tournament are those consisting of the in-neighbors of the vertices and of the out-neighbors of the vertices. Bounded twin width implies that the VC dimension of these two set systems is bounded, but the converse does not hold. It is easy to see that the VC dimensions of the in-neighbors and out-neighbors systems as well as the set system consisting of the mixed neighbors of all edges are within a linear factor of one another. As a natural next step, we may ask if isomorphism of tournaments where the VC-dimension of these systems is bounded is in polynomial time.

References

1 Vikraman Arvind, Ilia N. Ponomarenko, and Grigory Ryabov. Isomorphism testing of k-spanning tournaments is fixed parameter tractable. *CoRR*, abs/2201.12312, 2022. `arXiv:2201.12312`.

2 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697. ACM, 2016. doi:10.1145/2897518.2897542.

3 László Babai and Eugene M. Luks. Canonical labeling of graphs. In David S. Johnson, Ronald Fagin, Michael L. Fredman, David Harel, Richard M. Karp, Nancy A. Lynch, Christos H. Papadimitriou, Ronald L. Rivest, Walter L. Ruzzo, and Joel I. Seiferas, editors, *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 171–183. ACM, 1983. doi:10.1145/800061.808746.

4 Yonatan Bilu and Nathan Linial. Lifts, discrepancy and nearly optimal spectral gap. *Comb.*, 26(5):495–519, 2006. doi:10.1007/s00493-006-0029-7.

5 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1977–1996. SIAM, 2021. doi:10.1137/1.9781611976465.118.

6 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: max independent set, min dominating set, and coloring. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 35:1–35:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.35.

7 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Torunczyk. Twin-width IV: ordered graphs and matrices. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 924–937. ACM, 2022. doi:10.1145/3519935.3520037.

8 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, and Stéphan Thomassé. Twin-width V: linear minors, modular counting, and matrix multiplication. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 15:1–15:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.STACS.2023.15.

9 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, and Stéphan Thomassé. Twin-width VI: the lens of contraction sequences. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 1036–1056. SIAM, 2022. doi:10.1137/1.9781611977073.45.

10 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, Stéphan Thomassé, and Rémi Watrigant. Twin-width and polynomial kernels. *Algorithmica*, 84(11):3300–3337, 2022. doi:10.1007/s00453-022-00965-5.

11 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.

12 Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Comb.*, 12(4):389–410, 1992. doi:10.1007/BF01305232.

13 Maria Chudnovsky, Alexandra Ovetsky Fradkin, and Paul D. Seymour. Tournament immersion and cutwidth. *J. Comb. Theory, Ser. B*, 102(1):93–101, 2012. doi:10.1016/j.jctb.2011.05.001.

14 Maria Chudnovsky, Ringi Kim, Chun-Hung Liu, Paul D. Seymour, and Stéphan Thomassé. Domination in tournaments. *J. Comb. Theory, Ser. B*, 130:98–113, 2018. doi:10.1016/j.jctb.2017.10.001.

15 Maria Chudnovsky, Alex Scott, and Paul D. Seymour. Disjoint paths in unions of tournaments. *J. Comb. Theory, Ser. B*, 135:238–255, 2019. doi:10.1016/j.jctb.2018.08.007.

16 Maria Chudnovsky and Paul D. Seymour. A well-quasi-order for tournaments. *J. Comb. Theory, Ser. B*, 101(1):47–53, 2011. doi:10.1016/j.jctb.2010.10.003.

17 John D. Dixon and Brian Mortimer. *Permutation Groups*, volume 163 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1996. doi:10.1007/978-1-4612-0731-3.

18 Walter Feit and John G. Thompson. Solvability of groups of odd order. *Pacific J. Math.*, 13:775–1029, 1963.

19 Fedor V. Fomin and Michal Pilipczuk. On width measures and topological problems on semi-complete digraphs. *J. Comb. Theory, Ser. B*, 138:78–165, 2019. doi:10.1016/j.jctb.2019.01.006.

20 Jakub Gajarský, Stephan Kreutzer, Jaroslav Nesetril, Patrice Ossona de Mendez, Michal Pilipczuk, Sebastian Siebertz, and Szymon Torunczyk. First-order interpretations of bounded expansion classes. *ACM Trans. Comput. Log.*, 21(4):29:1–29:41, 2020. doi:10.1145/3382093.

21 Jakub Gajarský, Michal Pilipczuk, and Szymon Torunczyk. Stable graphs of bounded twin-width. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 39:1–39:12. ACM, 2022. doi:10.1145/3531130.3533356.

22 Robert Ganian, Filip Pokrývka, André Schidler, Kirill Simonov, and Stefan Szeider. Weighted model counting with twin-width. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 15:1–15:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SAT.2022.15.

23 Colin Geniet and Stéphan Thomassé. First order logic and twin-width in tournaments. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 53:1–53:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.53.

24 Martin Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*, volume 47 of *Lecture Notes in Logic*. Cambridge University Press, 2017. doi:10.1017/9781139028868.

25 Martin Grohe and Daniel Neuen. Recent advances on the graph isomorphism problem. In Konrad K. Dabrowski, Maximilien Gadouleau, Nicholas Georgiou, Matthew Johnson, George B. Mertzios, and Daniël Paulusma, editors, *Surveys in Combinatorics, 2021: Invited lectures from the 28th British Combinatorial Conference, Durham, UK, July 5-9, 2021*, pages 187–234. Cambridge University Press, 2021. doi:10.1017/9781009036214.006.

26 Martin Grohe and Daniel Neuen. Canonisation and definability for graphs of bounded rank width. *ACM Trans. Comput. Log.*, 24(1):6:1–6:31, 2023. doi:10.1145/3568025.

27 Martin Grohe, Daniel Neuen, and Pascal Schweitzer. A faster isomorphism test for graphs of small degree. *SIAM J. Comput.*, 52(6):S18–1, 2023. doi:10.1137/19m1245293.

28 Frank Gurski, Dominique Komander, Carolin Rehs, and Sebastian Wiederrecht. Directed width parameters on semicomplete digraphs. In Ding-Zhu Du, Donglei Du, Chenchen Wu, and Dachuan Xu, editors, *Combinatorial Optimization and Applications - 15th International Conference, COCOA 2021, Tianjin, China, December 17-19, 2021, Proceedings*, volume 13135 of *Lecture Notes in Computer Science*, pages 615–628. Springer, 2021. doi:10.1007/978-3-030-92681-6_48.

29 Petr Hlinený and Jan Jedelský. Twin-width of planar graphs is at most 8, and at most 6 when bipartite planar. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 75:1–75:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.75.

30 Neil Immerman and Eric Lander. Describing graphs: A first-order approach to graph canonization. In Alan L. Selman, editor, *Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday, July 5, 1988*, pages 59–81. Springer New York, New York, NY, 1990. doi:10.1007/978-1-4612-4478-3_5.

31 Thor Johnson, Neil Robertson, Paul D. Seymour, and Robin Thomas. Directed tree-width. *J. Comb. Theory, Ser. B*, 82(1):138–154, 2001. doi:10.1006/jctb.2000.2031.

32 Sandra Kiefer. The Weisfeiler-Leman algorithm: an exploration of its power. *ACM SIGLOG News*, 7(3):5–27, 2020. doi:10.1145/3436980.3436982.

33 Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Syst. Sci.*, 25(1):42–65, 1982. doi:10.1016/0022-0000(82)90009-5.

34 Adam W. Marcus, Daniel A. Spielman, and Nikhil Srivastava. Interlacing families I: Bipartite Ramanujan graphs of all degrees. *Ann. of Math. (2)*, 182(1):307–325, 2015. doi:10.4007/annals.2015.182.1.7.

35 Gary L. Miller. Isomorphism of graphs which are pairwise k-separable. *Inf. Control.*, 56(1/2):21–33, 1983. doi:10.1016/S0019-9958(83)80048-5.

36 Jaroslav Nesetril and Patrice Ossona de Mendez. Structural sparsity. *Russian Math. Surveys*, 71(1):79–107, 2016. doi:10.4213/rm9688.

37 Jaroslav Nesetril, Patrice Ossona de Mendez, and Sebastian Siebertz. Structural properties of the first-order transduction quasiorder. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, pages 31:1–31:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.CSL.2022.31.

38 Daniel Neuen. Hypergraph isomorphism for groups with restricted composition factors. *ACM Trans. Algorithms*, 18(3):27:1–27:50, 2022. doi:10.1145/3527667.

39 Michal Pilipczuk. *Tournaments and Optimality: New Results in Parameterized Complexity*. PhD thesis, University of Bergen, 2013.

40 Ilia N. Ponomarenko. Polynomial time algorithms for recognizing and isomorphism testing of cyclic tournaments. *Acta Appl. Math.*, 29(1-2):139–160, 1992. doi:10.1007/BF00053383.

41 Joseph J. Rotman. *An Introduction to the Theory of Groups*, volume 148 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, fourth edition, 1995. doi:10.1007/978-1-4612-4176-8.

42 Pascal Schweitzer. A polynomial-time randomized reduction from tournament isomorphism to tournament asymmetry. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 66:1–66:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.66.

43 Ákos Seress. *Permutation Group Algorithms*, volume 152 of *Cambridge Tracts in Mathematics*. Cambridge University Press, Cambridge, 2003. doi:10.1017/CBO9780511546549.

44 Stéphan Thomassé. A brief tour in twin-width (invited talk). In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 6:1–6:5. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.6.

45 Boris Weisfeiler. *On Construction and Identification of Graphs*, volume 558 of *Lecture Notes in Mathematics*. Springer-Verlag, 1976.

46 Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series 2*, 1968. English translation by Grigory Ryabov available at https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf.

From Trees to Polynomials and Back Again: New Capacity Bounds with Applications to TSP

Leonid Gurvits
City College New York, NY, USA

Nathan Klein
Institute for Advanced Study, Princeton, NJ, USA

Jonathan Leake
University of Waterloo, Canada

Abstract

We give simply exponential lower bounds on the probabilities of a given strongly Rayleigh distribution, depending only on its expectation. This resolves a weak version of a problem left open by Karlin-Klein-Oveis Gharan in their recent breakthrough work on metric TSP, and this resolution leads to a minor improvement of their approximation factor for metric TSP. Our results also allow for a more streamlined analysis of the algorithm.

To achieve these new bounds, we build upon the work of Gurvits-Leake on the use of the productization technique for bounding the capacity of a real stable polynomial. This technique allows one to reduce certain inequalities for real stable polynomials to products of affine linear forms, which have an underlying matrix structure. In this paper, we push this technique further by characterizing the worst-case polynomials via bipartitioned forests. This rigid combinatorial structure yields a clean induction argument, which implies our stronger bounds.

In general, we believe the results of this paper will lead to further improvement and simplification of the analysis of various combinatorial and probabilistic bounds and algorithms.

2012 ACM Subject Classification Mathematics of computing → Approximation algorithms; Mathematics of computing → Probability and statistics; Theory of computation → Routing and network design problems

Keywords and phrases traveling salesman problem, strongly Rayleigh distributions, polynomial capacity, probability lower bounds, combinatorial lower bounds

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.79

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2311.09072`

Funding *Nathan Klein*: This author is supported by the National Science Foundation under Grant No. DMS-1926686.
Jonathan Leake: This author acknowledges the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), [funding reference number RGPIN-2023-03726]. Cette recherche a été partiellement financée par le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG), [numéro de référence RGPIN-2023-03726].

Acknowledgements The third author would like to thank Vijay Bhattiprolu for many helpful and interesting discussions.

1 Introduction

The theory of real stable and log-concave polynomials has seen numerous applications in combinatorics and theoretical computer science (TCS). This includes bounds and approximation algorithms for various combinatorial quantities [18, 11, 7, 28, 5, 8, 2, 15], proofs of long-standing log-concavity and sampling conjectures related to matroids [1, 3, 4, 14], proofs

of the Kadison-Singer conjecture and generalizations [26, 6, 13], an improved approximation factor for the traveling salesperson problem (TSP) [27, 24, 25], and many more. The power of these polynomial classes comes from two features: (1) their *robustness*, shown in the fact that many natural operations preserve these log-concavity properties, and (2) their *convex analytic properties*, which can be used to prove bounds and other analytic statements. The typical way these polynomials are utilized is by encoding combinatorial objects as real stable and log-concave polynomials, which essentially allows these operations and convexity properties to automatically transfer to the combinatorial objects. This idea, while simple, has led to important breakthroughs in combinatorics, TCS, and beyond.

For example, [18] utilized real stable polynomials to give a new proof of the Van der Waerden conjecture on the permanent of a doubly stochastic matrix (originally due to [16, 17]). This proof led to a vast generalization of Van der Waerden bound, including an improved Schrijver's bound for regular bipartite graphs [18], an analogous bound for mixed discriminants [19], and an analogous bound for mixed volumes that led to the development of strongly log-concave polynomials [21, 20]. One reason the original bound was historically so difficult to prove is a lack of a usable inductive structure coming from the matrices themselves. One of the key insights of the new proof was to use the simple inductive structure of real stable polynomials given by partial derivatives. By encoding the matrices as polynomials, the correct induction becomes straightforward, and the bound follows from a simple calculus argument.

More recently, the approximation factor improvement for the metric traveling salesperson problem (TSP) crucially utilized real stable polynomials [24, 25]. The idea is to encode certain discrete probability distributions related to spanning trees as real stable polynomials. The coefficients of these polynomials give probabilities of certain graph-theoretic events (e.g., the number of edges in a given spanning tree incident on a particular vertex), and analytic properties of real stable polynomials allow one to lower bound these probabilities. This in turn implies bounds on the expected cost of the output of a randomized algorithm for metric TSP.

In this paper, we improve upon the polynomial capacity bounds of [22], and our applications touch on the two problems discussed above. Specifically, we give:

1. robust coefficient lower bounds for all (not necessarily homogeneous) real stable polynomials,
2. simply exponential lower bounds on probabilities of strongly Rayleigh distributions (solving a weak version of an open problem of [24]), and
3. a further improvement to the approximation factor for metric TSP (predicted by [22]).

Interestingly, our approach goes in the opposite direction to that discussed above. Our technical results answer questions regarding real stable polynomials, but to prove these results we use various graph and matrix structures inherent to the polynomials. In [22], this was seen in the "productization" technique: bounds on real stable polynomials were achieved by showing that the worst-case bounds come from polynomials associated to certain matrices. In this paper, we push this idea further by showing that these worst-case matrices are bipartite adjacency matrices of forests. This very rigid structure enables a clean induction argument, which implies stronger polynomial capacity bounds. These new bounds lead to the applications discussed above, with the strongest bounds implying the metric TSP improvement.

2 Main Results

We first state here our main technical results; see Section 3 for any undefined notation.

Our first main result is a non-homogeneous version of Theorem 2.1 of [22] which implies robust coefficient lower bounds for all real stable polynomials as a direct corollary. Crucially, these bounds do not depend on the total degree of the polynomial. This was one of the main barriers to applying the results of [22] to metric TSP.

▶ **Theorem 1** (Main non-homogeneous capacity bound). *Let $p \in \mathbb{R}_{\geq 0}[x_1, \ldots, x_n]$ be a real stable polynomial in n variables, and fix any $\boldsymbol{\kappa} \in \mathbb{Z}^n$ with non-negative entries. If $p(\mathbf{1}) = 1$ and $\|\boldsymbol{\kappa} - \nabla p(\mathbf{1})\|_1 < 1$, then*

$$\inf_{x_1,\ldots,x_n>0} \frac{p(\boldsymbol{x})}{x_1^{\kappa_1} \cdots x_n^{\kappa_n}} \geq (1 - \|\boldsymbol{\kappa} - \nabla p(\mathbf{1})\|_1)^n .$$

This bound is tight for any fixed $\boldsymbol{\kappa}$ with strictly positive entries.

▶ **Corollary 2** (Main non-homogeneous coefficient bound). *Let $p \in \mathbb{R}_{\geq 0}[x_1, \ldots, x_n]$ be a real stable polynomial in n variables, and fix any $\boldsymbol{\kappa} \in \mathbb{Z}^n$ with non-negative entries. If $p(\mathbf{1}) = 1$ and $\|\boldsymbol{\kappa} - \nabla p(\mathbf{1})\|_1 < 1$, then*

$$p_{\boldsymbol{\kappa}} \geq \left(\prod_{i=1}^{n} \frac{\kappa_i^{\kappa_i} e^{-\kappa_i}}{\kappa_i!} \right) (1 - \|\boldsymbol{\kappa} - \nabla p(\mathbf{1})\|_1)^n,$$

where $p_{\boldsymbol{\kappa}}$ is the coefficient of $\boldsymbol{x}^{\boldsymbol{\kappa}}$ in p. The dependence on $(1 - \|\boldsymbol{\kappa} - \nabla p(\mathbf{1})\|_1)$ is tight for any fixed $\boldsymbol{\kappa}$ with strictly positive entries.

The above results[1] are robust (i.e., resilient to ℓ^1 perturbations) versions of the results utilized to bound various combinatorial and probabilistic quantities, as discussed above. That said, they are still not quite strong enough to imply an improvement to the metric TSP approximation factor. To obtain this improvement, we resolve a weak version of an open problem from [24], which we discuss below. Stronger versions of Theorem 1 and Corollary 2 which imply this result can be found in Section 5.

2.1 Application: Minimum Permanent

Before discussing the application to TSP, we first describe a different application of our bounds as a sort of prelude. It is at this point well-known that the permanent of any $n \times n$ doubly stochastic matrix is at least $\frac{n!}{n^n}$, and that $\frac{1}{n}\mathbf{1} \cdot \mathbf{1}^\top$ is the unique minimizer of the permanent over all doubly stochastic matrices. On the other hand, a similar tight lower bound with explicit minimizer is not known for sets of matrices with different row and column sums. The following then slightly extends what is known in the doubly stochastic case. See Section 6 for further details.

Given $\boldsymbol{c} \in \mathbb{R}^n_{\geq 0}$, let $\mathrm{Mat}_n(\boldsymbol{c})$ denote the set of $n \times n$ matrices with non-negative entries, row sums equal to $\mathbf{1}$, and columns sums equal to $\boldsymbol{c}$.

▶ **Theorem 3.** *For all $n \geq 1$, there exists $\epsilon_n > 0$ such that if $\|\boldsymbol{c} - \mathbf{1}\|_1 < \epsilon_n$ then $\frac{1}{n}\mathbf{1} \cdot \boldsymbol{c}^\top$ is the unique minimizer of the permanent over $\mathrm{Mat}_n(\boldsymbol{c})$. Specifically, this holds if $\frac{c_1 c_2 \cdots c_n}{L(\boldsymbol{c})} < \frac{(n-2)^{n-2} n^{n-1}}{(n-1)^{2n-3}}$, where $L(\boldsymbol{c})$ is any lower bound on the capacity.*

[1] Note that these bounds already appear in the arXiv version of [22], but not in the STOC version.

The proof utilizes Theorem 1 or results of [22] to guarantee all minimizers of the permanent lie in the relative interior of $\mathrm{Mat}_n(\boldsymbol{c})$. The symmetry and multilinearity of the permanent then imply the minimizer must be the rank-one matrix of $\mathrm{Mat}_n(\boldsymbol{c})$ described above. See Section Section 6 for the explicit value of ϵ_n and the details of the proof. It should be noted that in proving uniqueness, we were able to avoid usage of the conditions for equality in the Alexandrov-Fenchel inequalities.

On the other hand, when $\boldsymbol{c}$ is far from $\mathbf{1}$, the above result can be far from correct. Recall from [22] that $\mathrm{per}(M) > 0$ for all $M \in \mathrm{Mat}_n(\boldsymbol{c})$ if and only if $\|\boldsymbol{c} - \mathbf{1}\|_1 < 2$.

▶ **Proposition 4.** *For all n large enough, there exists $\boldsymbol{c}$ such that $\|\boldsymbol{c} - \mathbf{1}\|_1 < 2$ and a sparse matrix $M \in \mathrm{Mat}_n(\boldsymbol{c})$ (with linearly many non-zero entries) which has smaller permanent than that of $\frac{1}{n}\mathbf{1} \cdot \boldsymbol{c}^\top$.*

These two results suggest the complexity of the minimizer of the permanent, given the column sums $\boldsymbol{c}$. The coefficient bound given above in Corollary 2 is then a lower bound which generalizes that of the permanent to coefficients of real stable polynomials. (Consider the coefficient $p_{\mathbf{1}}$ of $p(\boldsymbol{x}) = \prod_{i=1}^n \sum_{j=1}^n m_{ij} x_j$.) Thus Corollary 2 can be seen as a sort of "smoothing" of the complexities that can occur for minima of the permanent and its generalizations.

Further, Theorem 3 can generalized to a permanent-like function on rectangular matrices using Theorem 1, and even beyond that to the mixed discriminant. (For the mixed discriminant, the row sum condition becomes a trace condition, and the column sum condition becomes an eigenvalue condition.) On the other hand, we can generalize the statement of Theorem 3 to coefficients of real stable polynomials in general, but we do not yet know how to prove it.

2.2 Application: Metric TSP

We first recall an important probabilistic bound from [24] used in the analysis of their metric TSP approximation algorithm (which is a slight modification of the max entropy algorithm from [27] first studied by [9]). In what follows, we let $A_S := \sum_{i \in S} A_i$ and $\kappa_S := \sum_{i \in S} \kappa_i$. See Section 3 for any undefined notation.

▶ **Theorem 5** (Prop. 5.1 of [24])**.** *Let μ be a strongly Rayleigh distribution on $[m]$, let $A_1, \ldots, A_n$ be random variables counting the number of elements contained in disjoint subsets of $[m]$, and fix $\boldsymbol{\kappa} \in \mathbb{Z}^n$ with non-negative entries. Suppose for all $S \subseteq [n]$ we have*

$$\mathbb{P}_\mu\left[A_S \geq \kappa_S\right] \geq \epsilon \qquad \text{and} \qquad \mathbb{P}_\mu\left[A_S \leq \kappa_S\right] \geq \epsilon.$$

Then we have

$$\mathbb{P}_\mu\left[A_1 = \kappa_1, \ldots, A_n = \kappa_n\right] \geq f(\epsilon) \cdot \mathbb{P}_\mu\left[A_{[n]} = \kappa_{[n]}\right],$$

where $f(\epsilon) \geq \epsilon^{2^n} \prod_{i=2}^n \frac{1}{\max\{\kappa_i, \kappa_{[i-1]}\}+1}$.

In [24], the authors note two things about this bound. First, they note that to apply the bound it is sufficient to have

$$\left|\mathbb{E}_\mu\left[A_S\right] - \kappa_S\right| < 1 \qquad \forall S \subseteq [n], \tag{1}$$

since this implies a lower bound on $\mathbb{P}_\mu\left[A_S = \kappa_S\right]$ for all $S \subseteq [n]$ for strongly Rayleigh distributions. Second, they note that the bound on $f(\epsilon)$ is doubly exponential in n, but they expect the true dependency to only be simply exponential. They leave it as an open problem to determine a tight lower bound on $f(\epsilon)$.

In this paper, we further improve the metric TSP approximation factor by resolving a weak version of this open problem: we give a simply exponential lower bound which depends tightly on ϵ, under the stronger condition of (1). Concretely, we prove the following.

▶ **Theorem 6** (Improved probability lower bound). *Let μ be a strongly Rayleigh distribution on $[m]$, let $A_1, \ldots, A_n$ be random variables counting the number of elements contained in disjoint subsets of $[m]$, and fix $\boldsymbol{\kappa} \in \mathbb{Z}^n$ with non-negative entries. Suppose for all $S \subseteq [n]$ we have*

$$\left|\mathbb{E}_\mu [A_S] - \kappa_S\right| \leq 1 - \epsilon.$$

Then we have

$$\mathbb{P}_\mu [A_1 = \kappa_1, \ldots, A_n = \kappa_n] \geq \epsilon^n \prod_{\kappa_i > 0} \frac{1}{e\sqrt{\kappa_i}}.$$

The dependence on ϵ is tight for any fixed $\boldsymbol{\kappa}$ with strictly positive entries.

In fact, we prove stronger versions of this result which more directly depend on the specific values of $(\mathbb{E}_\mu [A_S] - \kappa_S)$ for all $S \subseteq [n]$; see Theorem 9 and Corollary 10. These results are analogous to our main coefficient bound Corollary 2 because the coefficients and the gradient of probability generating polynomials can be interpreted as the probabilities and the expectation of the associated distribution. That said, our stronger probabilistic results require a more delicate analysis of the expectations (gradient) beyond what is required for Corollary 2. In particular, note that the conditions on the expectations in Theorem 6 are more general than a bound on the ℓ^1 norm of $(\mathbb{E}_\mu [A_i] - \kappa_i)_{i=1}^n$. See Section 4 for further details.

Using Theorem 6, we improve the metric TSP approximation factor for the algorithm given in [24].

▶ **Theorem 7.** *There exists a randomized algorithm for metric TSP with approximation factor $\frac{3}{2} - \epsilon$ for some $\epsilon > 10^{-34}$.*

This is about a 100 times improvement over the result of [23]. Thus our improvement in terms of the approximation factor itself may be smaller than anticipated, given that we were able to improve the probability bound in Theorem 6 from doubly exponential to simply exponential. The reason for this is that while Theorem 5 was useful in [24] to quickly determine which events occurred with constant probability (and indeed provided a single unifying explanation for why one should expect many of their probabilistic bounds to hold), it gave such small guarantees that [24] resorted to ad hoc arguments instead to give their final probabilistic bounds.

We show that Theorem 6 alone can be used to give bounds that are comparable to the ad hod methods of [24] (and, in several important cases, much better) whenever the bounds came purely from information on the expectations as in (1). Thus, we believe our main contribution to work on metric TSP is a version of Theorem 5 that is "reasonable" to use, allowing one to show a similar approximation factor but with a more streamlined proof.

Unfortunately, not all of the bounds in [24] follow from expectation information, and two of them become bottlenecks for improving the approximation factor after applying Theorem 6 to the other statements. Thus, to demonstrate Theorem 7 we need to sharpen these bounds using other techniques. For one of these lemmas we show that the existing proof in [24] was far from tight, and in the other we refine their proof. In particular, using Theorem 6, we show we can reduce this second lemma to a special case that is possible to analyze more carefully. See Section 4 for further details.

3 Technical Overview

In this section we discuss the proof strategy of our main capacity and coefficient bounds, Theorem 1 and Corollary 2, and their stronger forms.

Notation

Given a vector $\boldsymbol{z} \in \mathbb{R}^E$ and a subset S of E, let $\boldsymbol{z}^S := \prod_{e \in S} z_e$. Let $\mu : \{0,1\}^E \to \mathbb{R}$ be a probability distribution over subsets of E. The generating polynomial $g_\mu \in \mathbb{R}_{\geq 0}[\{z_e\}_{e \in E}]$ of μ is defined as

$$g_\mu(z) := \sum_{S \subseteq E} \mu(S) \cdot \boldsymbol{z}^S.$$

The distribution μ is **strongly Rayleigh** if g_μ is real stable, where a polynomial $p \in \mathbb{R}[z_1, \ldots, z_n]$ is **real stable** if $p(\boldsymbol{z}) \neq 0$ whenever $\Im(z_i) > 0$ for all $i \in [n]$ (i.e., when all inputs are in the complex upper half-plane). See [12] for much more on strongly Rayleigh measures. Further, given a polynomial $p \in \mathbb{R}_{\geq 0}[x_1, \ldots, x_n]$ and $\boldsymbol{\kappa} \in \mathbb{Z}^n_{\geq 0}$, the **capacity** of p is defined as

$$\mathrm{Cap}_{\boldsymbol{\kappa}}(p) := \inf_{\boldsymbol{x} > 0} \frac{p(\boldsymbol{x})}{\boldsymbol{x}^{\boldsymbol{\kappa}}}.$$

Finally, we let $p_{\boldsymbol{\kappa}}$ denote the coefficient of $\boldsymbol{x}^{\boldsymbol{\kappa}}$ in p.

Conceptual strategy

We first give an overarching view of the strategy used to prove our main results, as well as the key similarities and differences compared to that of [22]. The general idea for proving our bounds is to find a simple and sparse underlying structure for the worst-case inputs. The space of all real stable polynomials can be complicated, but we show that the worst-case polynomials for our bounds are far simpler: they are "sparse" products of affine linear forms. More concretely, we reduce the space of input polynomials (and the corresponding combinatorial structures) as follows:

$$\begin{array}{ccccc} \text{real stable polynomials} & \Longrightarrow & \text{products of linears} & \Longrightarrow & \text{sparse products of linears} \\ \text{matroids} & \Longrightarrow & \text{matrices} & \Longrightarrow & \text{forests} \end{array}$$

The first reduction step uses the idea of productization which was the key idea from [22]. This allows for one to utilize the matrix structure inherent to products of affine linear forms.

The second reduction step is then new to this paper. We first show that we may restrict to the extreme points of the set of matrices corresponding to products of affine linear forms, and then we show that these extreme matrices are supported on the edges of forests. This implies a significant decrease in density of the matrices: general graphs can have quadratically many edges, whereas forests can only have linearly many. This allows for an intricate but clean induction on the leaf vertices of these forests, which yields the strongest bounds of this paper. Additionally, it is this step that allows for bounds which do not depend on the total degree of the polynomial, and this was a crucial barrier to applying the bounds of [22] to metric TSP.

3.1 Conceptual Strategy, in More Detail

We now go through the steps of the conceptual strategy described above in more detail. Let us first restate our main capacity and coefficient bounds.

▶ **Theorem 8** (= Theorem 1 and Corollary 2)**.** *Let $p \in \mathbb{R}_{\geq 0}[x_1, \ldots, x_n]$ be a real stable polynomial in n variables, and fix any $\boldsymbol{\kappa} \in \mathbb{Z}^n$ with non-negative entries. If $p(\mathbf{1}) = 1$ and $\|\boldsymbol{\kappa} - \nabla p(\mathbf{1})\|_1 < 1$, then*

$$\mathrm{Cap}_{\boldsymbol{\kappa}}(p) \geq (1 - \|\boldsymbol{\kappa} - \nabla p(\mathbf{1})\|_1)^n$$

and

$$p_{\boldsymbol{\kappa}} \geq \left(\prod_{i=1}^{n} \frac{\kappa_i^{\kappa_i} e^{-\kappa_i}}{\kappa_i!} \right) (1 - \|\boldsymbol{\kappa} - \nabla p(\mathbf{1})\|_1)^n ,$$

*where $\mathrm{Cap}_{\boldsymbol{\kappa}}(p) := \inf_{\boldsymbol{x} > 0} \frac{p(\boldsymbol{x})}{\boldsymbol{x}^{\boldsymbol{\kappa}}}$ is the **capacity** of p and $p_{\boldsymbol{\kappa}}$ is the coefficient of $\boldsymbol{x}^{\boldsymbol{\kappa}}$ in p. The dependence on $(1 - \|\boldsymbol{\kappa} - \nabla p(\mathbf{1})\|_1)$ in these bounds is tight for any fixed $\boldsymbol{\kappa}$ with strictly positive entries.*

Analogous bounds required for the metric TSP application then follow from interpreting desired quantities as the coefficients and gradient of certain real stable polynomials. Specifically, the strongly Rayleigh probabilities we wish to lower bound are the coefficients of the corresponding real stable generating polynomial, and the expectations of the associated random variables are given by the gradient of that polynomial. We leave further details to Section 4.

We now discuss the proof of Theorem 8. First note that the coefficient bound follows from the capacity bound. This immediately follows from Corollary 3.6 of [20], which implies

$$p_{\boldsymbol{\kappa}} \geq \left(\prod_{i=1}^{n} \frac{\kappa_i^{\kappa_i} e^{-\kappa_i}}{\kappa_i!} \right) \mathrm{Cap}_{\boldsymbol{\kappa}}(p). \tag{2}$$

Thus what remains to be proven is the capacity bound

$$\mathrm{Cap}_{\boldsymbol{\kappa}}(p) \geq (1 - \|\boldsymbol{\kappa} - \nabla p(\mathbf{1})\|_1)^n ,$$

which is precisely the bound of Theorem 1, as well as its tightness, which follows from considering a particular example of p (see Lemma 28).

The remainder of the proof then has four main steps. We also note here that these proof steps actually imply stronger bounds than Theorem 1, see Corollary 27 for the formal statement. These stronger bounds are required for the metric TSP application.

Step 1: Reduce to products of affine linear forms via productization

We first generalize the productization technique of [22] to non-homogeneous real stable polynomials. The upshot of this technique is that it implies it is sufficient to prove Theorem 1 for products of affine linear forms with non-negative coefficients (see Corollary 18). Such polynomials correspond to $d \times (n+1)$ $\mathbb{R}_{\geq 0}$-valued matrices with row sums $\mathbf{1}$ and column sums $\boldsymbol{\alpha}$ equal the entries of the gradient of the polynomial, via

$$\phi : A \mapsto \prod_{i=1}^{d} \left(a_{i,n+1} + \sum_{j=1}^{n} a_{i,j} x_j \right) .$$

This gives far more structure to work with, beyond that of real stable polynomials in general. This part is a straightforward generalization of the analogous result of [22].

Step 2: Reduce to extreme points

The set of $\mathbb{R}_{\geq 0}$-valued matrices with row sums $\mathbf{1}$ and column sums $\boldsymbol{\alpha}$ forms a convex polytope $P^d_{\boldsymbol{\alpha}}$, and thus the polynomials we now must consider correspond to the points of this polytope via the map ϕ defined above. Inspired by a result of Barvinok (see Lemma 19), we next show that the function

$$A \mapsto \mathrm{Cap}_{\boldsymbol{\kappa}}(\phi(A))$$

is log-concave on the above described polytope. Since we want to minimize the capacity, this implies we may further restrict to the polynomials associated to the extreme points of the polytope.

Step 3: Extreme points correspond to bipartitioned forests

Any $\mathbb{R}_{\geq 0}$-valued matrix A can be interpreted as the weighted bipartite adjacency matrix of a bipartite graph, where the left vertices correspond to the rows of A and the right vertices correspond to the columns of A. A matrix $A \in P^d_{\boldsymbol{\alpha}}$ being extreme implies the associated bipartite graph has no cycles. This implies the associated bipartite graph is a forest (see Lemma 20). The sparsity properties of such matrices implies a simple structure for the associated polynomials, which is particularly amenable to an intricate but clean induction.[2]

Step 4: Induction on leaf vertices of the bipartitioned forests

Leaf vertices in the forest corresponding to a given matrix $A \in P^d_{\boldsymbol{\alpha}}$ indicate rows or columns of the matrix A which have exactly one non-zero entry. If a row of A has exactly one non-zero entry, the induction proceeds in a straightforward fashion, by simply removing the corresponding row of A and recalculating the column sums (see the $d \geq n+1$ case of the proof of Theorem 25).

If a column of A (say column i) has exactly one non-zero entry, then the induction is more complicated. We prove lemmas showing how much the capacity can change after applying the partial derivative ∂_{x_i} (when $\kappa_i \geq \alpha_i$, see Lemma 24) or setting x_i to 0 (when $\kappa_i < \alpha_i$, see Lemma 23). Since column i has only one entry, applying ∂_{x_i} corresponds to removing column i and the row of A which contains the non-zero entry, and setting x_i to 0 corresponds to removing column i. After renormalizing the row sums and recalculating the column sums, the proof again proceeds by induction. (See the proof of Theorem 25 to see the above arguments presented formally.) Example 29 and Example 30 show that the distinction between the $\kappa_i \geq \alpha_i$ and $\kappa_i < \alpha_i$ cases is not an artifact of the proof.

Some comments on tightness of the bounds

The main coefficient bound of Corollary 2 is proven via two different bounds, as discussed at the beginning of this section. That is, one first bounds the coefficient in terms of the capacity (2) via Corollary 3.6 of [20], and then one bounds the capacity via the steps outlined above. Thus while the capacity bound (Theorem 1) is tight for $\boldsymbol{\kappa} > \mathbf{0}$, the coefficient bound

[2] Note that it is already mentioned in [22] that the supports of the extreme points correspond to forests, but the application of this observation in [22] is somewhat "naive" and kind of brute-force: it was mainly used to describe an (inefficient) algorithm to compute the capacity lower bound for products of linear forms, which is not related to main lower bound in this paper. Additionally, its use in [22] is quite conceptually far from how it could actually be used to improve the TSP approximation factor.

(Corollary 2) may not be. We note that the coefficient bound is likely close to tight in the case that $(1 - \|\boldsymbol{\kappa} - \nabla p(\mathbf{1})\|_1)$ is close to 1, but it seems this tightness deteriorates as $(1 - \|\boldsymbol{\kappa} - \nabla p(\mathbf{1})\|_1)$ gets close to 0. That said, the dependence on $(1 - \|\boldsymbol{\kappa} - \nabla p(\mathbf{1})\|_1)$ in Theorem 1 and Corollary 2 is tight for $\boldsymbol{\kappa} > \mathbf{0}$ by Lemma 28, though there is still potential for improvement in the more refined capacity bounds; see Section 5.5.

That said, tight coefficient lower bounds in the univariate case can be achieved by directly applying a bound of Hoeffding, and these lower bounds resemble the coefficient lower bound of Corollary 2. Thus one can view Corollary 2 as a step towards a multivariate generalization of Hoeffding's theorem. It is an interesting question whether or not the techniques used here can be extended to a full multivariate generalization of Hoeffding's theorem.

3.2 Example: The Univariate Case

In this section, we demonstrate the proof of Theorem 1 in the univariate case. This will serve as a sort of proof of concept for the more general proof.

Here we consider real stable non-homogeneous polynomials $p \in \mathbb{R}_{\geq 0}[x_1]$ such that $p(1) = 1$ and $\nabla p(1) = \alpha_1$, and we define $\epsilon := 1 - |\alpha_1 - \kappa_1| > 0$. By Step 1 above, we may assume that p is of the form

$$p(x_1) = \prod_{i=1}^{d}(a_{i,1}x_1 + a_{1,2}),$$

where A is an $\mathbb{R}_{\geq 0}$-valued $d \times 2$ matrix with row sums 1 and column sums $(\alpha_1, d - \alpha_1)$. By Steps 2-3, we may further assume A is the weighted bipartite adjacency matrix of a forest. If every row of A contains exactly one non-zero entry then $p(x_1) = x_1^{k_1}$, and the result is trivial in this case. Otherwise, $d - 1$ rows of M have exactly one non-zero entry (see Lemma 21). Thus for some $k \leq d - 1$ we have

$$p(x_1) = x_1^k(ax_1 + b),$$

where $a + b = 1$ and $a + k = \alpha_1$. Since $\kappa_1 \in \mathbb{Z}_{\geq 0}$ and

$$\epsilon = 1 - |\alpha_1 - \kappa_1| = 1 - |a + k - \kappa_1|,$$

we have that k is equal to either κ_1 or $\kappa_1 - 1$. If $k = \kappa_1$, then

$$\mathrm{Cap}_{\kappa_1}(p) = \inf_{x_1 > 0} \frac{x_1^k(ax_1 + b)}{x_1^k} = b \qquad \text{and} \qquad \epsilon = 1 - |a + k - \kappa_1| = 1 - a = b.$$

If $k = \kappa_1 - 1$, then

$$\mathrm{Cap}_{\kappa_1}(p) = \inf_{x_1 > 0} \frac{x_1^k(ax_1 + b)}{x_1^{k+1}} = a \qquad \text{and} \qquad \epsilon = 1 - |a + k - \kappa_1| = 1 - (1 - a) = a.$$

Therefore in both cases we have

$$\mathrm{Cap}_{\kappa_1}(p) = \epsilon = (1 - |\alpha_1 - \kappa_1|),$$

which proves Theorem 1 in the univariate case and demonstrates the tight dependence on $(1 - \|\boldsymbol{\kappa} - \nabla p(\mathbf{1})\|_1)$ in this case.

4 Proof of Application: Metric TSP

The following is the strongest probability lower bound, which we will use for the metric TSP application. In what follows, we let $A_S := \sum_{i \in S} A_i$ and $\kappa_S := \sum_{i \in S} \kappa_i$. Note that if

$$\left|\mathbb{E}_\mu [A_S] - \kappa_S\right| \leq 1 - \epsilon,$$

then Theorem 6 immediately follows from Theorem 9, Lemma 28, and a standard computation.

▶ **Theorem 9** (Strongest form of the probability bound). *Let μ be a strongly Rayleigh distribution on $[m]$, let $A_1, \ldots, A_n$ be random variables counting the number of elements contained in some associated disjoint subsets of $[m]$, and fix $\boldsymbol{\kappa} \in \mathbb{Z}^n_{\geq 0}$. Suppose for all $S \subseteq [n]$ we have $|\mathbb{E}_\mu [A_S] - \kappa_S| < 1$. Define $\boldsymbol{\epsilon}, \boldsymbol{\delta} \in \mathbb{R}^n_{>0}$ via*

$$\delta_k := 1 + \min_{S \in \binom{[n]}{k}} \left(\mathbb{E}_\mu [A_S] - \kappa_S\right) \qquad \text{and} \qquad \epsilon_k := 1 - \max_{S \in \binom{[n]}{k}} \left(\mathbb{E}_\mu [A_S] - \kappa_S\right).$$

Then we have

$$\mathbb{P}_\mu [A_1 = \kappa_1, \ldots, A_n = \kappa_n] \geq \prod_{i=1}^n \frac{\kappa_i^{\kappa_i} e^{-\kappa_i}}{\kappa_i!} \cdot \min_{0 \leq \ell \leq n} \prod_{k=1}^{\ell} \epsilon_k \prod_{k=1}^{n-\ell} \delta_k.$$

Proof. Let q be the probability generating polynomial of μ, and let p be the polynomial obtained by setting the variables of q associated to A_i to x_i for all $i \in [n]$, and setting all other variables to equal 1. Since μ is strongly Rayleigh, p is real stable. Further, $p(\mathbf{1}) = 1$, $\nabla p(\mathbf{1}) = (\mathbb{E}_\mu [A_i])_{i=1}^n$, and $\mathbb{P}_\mu [A_1 = \kappa_1, \ldots, A_n = \kappa_n]$ is the $\boldsymbol{x}^{\boldsymbol{\kappa}}$ coefficient of p. Thus Gurvits' capacity inequality (2) and Corollary 27 imply the desired result. ◀

We also give a slightly weaker bound which is a bit easier to use in practice. Note that Theorem 6 also follows from Corollary 10.

▶ **Corollary 10.** *Let μ be a strongly Rayleigh distribution on $[m]$, let $A_1, \ldots, A_n$ be random variables counting the number of elements contained in some associated disjoint subsets of $[m]$, and fix $\boldsymbol{\kappa} \in \mathbb{Z}^n_{\geq 0}$. Suppose for all $S \subseteq [n]$ we have $|\mathbb{E}_\mu [A_S] - \kappa_S| < 1$. Define $\boldsymbol{\epsilon} \in \mathbb{R}^n_{>0}$ via*

$$\epsilon_k := 1 - \max_{\substack{S \subseteq [n] \\ |S| \leq k}} \left|\mathbb{E}_\mu [A_S] - \kappa_S\right|$$

Then we have

$$\mathbb{P}_\mu [A_1 = \kappa_1, \ldots, A_n = \kappa_n] \geq \prod_{i=1}^n \frac{\kappa_i^{\kappa_i} e^{-\kappa_i}}{\kappa_i!} \cdot \prod_{k=1}^n \epsilon_k.$$

Proof. Follows from Theorem 9; see the proof of Corollary 26 for more details. ◀

The remainder of this section is devoted to demonstrating how one can use the above results to improve the approximation factor for metric TSP.

4.1 A Simple Application

We recall some definitions from [24]. There, we have a graph $G = (V, E)$ and a strongly Rayleigh (SR) distribution $\mu : 2^E \to \mathbb{R}_{\geq 0}$ supported on spanning trees of G. We let $x_e = \mathbb{P}_{T\sim\mu}[e \in T]$ and for a set of edges F let $x(F) = \sum_{e\in F} x_e$. Furthermore we let $\delta(S) = \{e = \{u, v\} \mid |e \cap S| = 1\}$. The guarantee on x is that $x \in P_{\text{Sub}}$,[3] where

$$P_{\text{Sub}} := \begin{cases} x(\delta(S)) \geq 2 & \forall\, S \subsetneq V \\ x(\delta(v)) = 2 & \forall v \in V \\ x_{\{u,v\}} \geq 0 & \forall\, u, v \in V \end{cases} \tag{3}$$

In the algorithm they analyze, one first samples a spanning tree T from μ and then add the minimum cost matching on the odd vertices of T. Their work involves analyzing the expected cost of this matching over the randomness of the sampled tree T. Unsurprisingly, the *parity* of vertices is therefore very important, as it determines which vertices are involved in the matching.

In this section, to give some sense of the utility of Theorem 9, we show an application in a simplified setting. Namely, we show that for any two vertices u, v, except in the special case that $x_{\{u,v\}} \approx \frac{1}{2}$, we have $\mathbb{P}_{T\sim\mu}[|\delta(u) \cap T| = |\delta(v) \cap T| = 2] \geq \Omega(1)$, i.e. for any two vertices that do not share an edge of value $\frac{1}{2}$ there is a constant probability that they have even parity simultaneously. This is helpful because (under some conditions on the point $x \in P_{\text{Sub}}$) the event that u, v are even simultaneously indicates one can strictly decrease the cost of the matching proportional to the cost of the edge e.

We now prove $\mathbb{P}_{T\sim\mu}[|\delta(u) \cap T| = |\delta(v) \cap T| = 2] \geq \Omega(1)$ whenever $x_{\{u,v\}} \not\approx \frac{1}{2}$. To do this, we split into two cases: when $x_{\{u,v\}} \geq \frac{1}{2} + \epsilon$ and when $x_{\{u,v\}} \leq \frac{1}{2} - \epsilon$.

▶ **Lemma 11.** *Let u, v be two vertices such that $x_{\{u,v\}} \geq \frac{1}{2} + \epsilon$ for some $\epsilon > 0$. Then,*

$$\mathbb{P}_{T\sim\mu}[|\delta(u) \cap T| = |\delta(v) \cap T| = 2] \geq \frac{\epsilon}{2e^3}.$$

Proof. Let $e = (u, v)$. For $T \sim \mu$, let $A_1 = \mathbb{I}[e \in T]$, $A_2 = |(\delta(u) \setminus \{e\}) \cap T|$, $A_3 = |(\delta(v) \setminus \{e\}) \cap T|$. We are now interested in the event $A_i = \kappa_i, \forall i$ for the vector $\kappa = (1, 1, 1)$ as this implies $|\delta(u) \cap T| = |\delta(v) \cap T| = 2$. We have $\mathbb{E}[A_1] = x_e$, $\mathbb{E}[A_2] = \mathbb{E}[A_3] = 2 - x_e$. Therefore, to apply Theorem 9 we can set:

$$\delta_1 = x_e, \delta_2 = 1, \delta_3 = 2 - x_e, \qquad \epsilon_1 = x_e, \epsilon_2 = 2x_e - 1, \epsilon_3 = x_e$$

In this case the worst case is using all of the ϵ terms in the bound, giving $e^{-3}x_e^2(2x_e - 1) \geq \frac{\epsilon}{2e^3}$ as desired. ◀

▶ **Lemma 12.** *Let u, v be two vertices such that $x_{\{u,v\}} \leq \frac{1}{2} - \epsilon$ for some $\epsilon > 0$. Then,*

$$\mathbb{P}_{T\sim\mu}[|\delta(u) \cap T| = |\delta(v) \cap T| = 2] \geq \frac{2\epsilon}{e^4}.$$

Proof. As above, $e = (u, v)$, and for $T \sim \mu$, let $A_1 = \mathbb{I}[e \in T]$, $A_2 = |(\delta(u) \setminus \{e\}) \cap T|$, $A_3 = |(\delta(v) \setminus \{e\}) \cap T|$. We are now interested in the event $A_i = \kappa_i, \forall i$ for the vector $\kappa = (0, 2, 2)$ as this implies $|\delta(u) \cap T| = |\delta(v) \cap T| = 2$. We have $\mathbb{E}[A_1] = x_e$, $\mathbb{E}[A_2] = \mathbb{E}[A_3] = 2 - x_e$. Therefore, to apply Theorem 9 we can set:

$$\delta_1 = 1 - x_e, \delta_2 = 1 - 2x_e, \delta_3 = 1 - x_e, \qquad \epsilon_1 = 1 - x_e, \epsilon_2 = 1, \epsilon_3 = 1 + x_e$$

In this case the worst case is using all of the δ terms in the bound, giving $4e^{-4}(1 - x_e)^2(1 - 2x_e) \geq \frac{2\epsilon}{e^4}$. ◀

[3] Technically, a spanning tree plus an edge is sampled, as otherwise one cannot exactly have $x \in P_{\text{Sub}}$, but we ignore that here.

We leave further background about TSP and the proofs of our improved probabilistic statements to the full version of the paper, as understanding their importance requires some knowledge of the (highly technical) proof in [24]. However, in the next section we summarize the new probabilistic bounds we get and their consequences.

4.2 Summary of Probabilistic Bounds and New Approximation Factor

The current bound on the performance of the max entropy algorithm is $\frac{3}{2} - 4.11 \cdot 10^{-36}$. This is primarily governed by a constant p that is determined by the minimum probability over a number of events. In [24], p was equal to $2 \cdot 10^{-10}$, and these events are described by the following statements in [24]:

1. Corollary 5.9, which gives a bound of $1.5 \cdot 10^{-9}$. We do not modify this bound, although we note that Lemma 5.7 can be slightly improved which would lead to a small improvement here.
2. Lemma 5.21, which gives a bound of $0.005\epsilon_{1/2}^2 = 2 \cdot 10^{-10}$. We improve this to $0.039\epsilon_{1/2}^2 = 1.56 \cdot 10^{-9}$.
3. Lemma 5.22, which gives a bound of $0.006\epsilon_{1/2}^2 = 2.4 \cdot 10^{-10}$. We improve this to $0.038\epsilon_{1/2}^2 = 1.52 \cdot 10^{-9}$.
4. Lemma 5.23, which gives a bound of $0.005 \cdot \epsilon_{1/2} = 2 \cdot 10^{-10}$. We observe here that arguments already in [24] can be used to give $0.0498\epsilon_{1/2}^2 \geq 1.9 \cdot 10^{-9}$.
5. Lemma 5.24, which gives a bound of $0.02\epsilon_{1/2}^2 = 8 \cdot 10^{-10}$. We improve this to $0.0485\epsilon_{1/2}^2 \geq 1.9 \cdot 10^{-9}$.
6. Lemma 5.27, which gives a bound of 0.01. This lemma actually uses that the threshold p is *small*, and therefore this bound decreases slightly upon raising p. However, as it is quite far from being the bottleneck in these bounds, we omit the proof that the probability remains above $1.5 \cdot 10^{-9}$.

Therefore, we may increase p to the minimum of all these probabilities, $1.5 \cdot 10^{-9}$. Using statements from [24, 23], the following then holds:

▶ **Lemma 13.** *Let p be a lower bound on the probabilities guaranteed by (1) - (6) for $\epsilon_{1/2} \leq 0.0002$, and suppose $p \leq 10^{-4}$. Then given $x \in P_{\text{Sub}}$, the max entropy algorithm returns a solution of expected cost at most $(\frac{3}{2} - 9.7p^2 \cdot 10^{-17}) \cdot c(x)$.*

As we improve the bounds on p to $1.5 \cdot 10^{-9}$, an immediate corollary is the following:

▶ **Corollary 14.** *The max entropy algorithm is a $\frac{3}{2} - 2.18 \cdot 10^{-34}$ approximation algorithm for metric TSP.*

Using [25] this guarantee can be made deterministic, as we do not require any modifications to the algorithm.

In the full version of the paper, we also observe a lower bound on Lemma 5.21 for strongly Rayleigh distributions of $\Omega(\epsilon_{1/2}^2)$. The fact that $\epsilon_{1/2} \leq 0.0002$ is used in many places in [24] and thus decreasing it may require more effort. Thus without modifying other parts of the argument, it may not be possible to improve the bound below $1.5 \cdot 10^{-31}$.

In the rest of the paper we prove our main capacity bound.

5 Proofs of the Main Capacity Bounds

In this section we prove the strongest forms of the main capacity bounds, which give Theorem 1 and Corollary 2 as corollaries. See the full version of the paper for any missing proofs. For this section, we utilize the following notation.

▶ **Definition 15.** *For $n, d \in \mathbb{N}$ and $\boldsymbol{\alpha} \in \mathbb{R}^n_{\geq 0}$, we define the following:*

1. $\mathrm{NHMat}^d_n(\boldsymbol{\alpha})$ *is the set of all $\mathbb{R}_{\geq 0}$-valued $d \times (n+1)$ matrices with row sums all equal to 1 and column sums equal to $\alpha_1, \ldots, \alpha_n, d - \|\boldsymbol{\alpha}\|_1$.*
2. $\mathrm{NHProd}^d_n(\boldsymbol{\alpha})$ *is the set of all polynomials of the form*

$$p(x) = \prod_{i=1}^{d} \left(a_{i,n+1} + \sum_{j=1}^{n} a_{i,j} x_j \right),$$

where $A \in \mathrm{NHMat}^d_n(\boldsymbol{\alpha})$. In this case, we call p the polynomial associated to A. *Note that $p(\mathbf{1}) = 1$ and $\nabla p(\mathbf{1}) = \boldsymbol{\alpha}$ for all such polynomials.*
3. $\mathrm{NHStab}^d_n(\boldsymbol{\alpha})$ *is the set of all real stable polynomials in $\mathbb{R}_{\geq 0}[x_1, \ldots, x_n]$ of degree at most d for which $p(\mathbf{1}) = 1$ and $\nabla p(\mathbf{1}) = \boldsymbol{\alpha}$. (Recall that a polynomial is stable if it is never zero when all inputs are in the open complex upper half-plane.)*

We also define the following for $n \in \mathbb{N}$, $\boldsymbol{\alpha} \in \mathbb{R}^n_{\geq 0}$, and $\boldsymbol{\kappa} \in \mathbb{Z}^n_{\geq 0}$:

$$L^{\mathrm{NHProd}}_n(\boldsymbol{\alpha}; \boldsymbol{\kappa}) := \min_{d \in \mathbb{N}} \min_{p \in \mathrm{NHProd}^d_n(\boldsymbol{\alpha})} \mathrm{Cap}_{\boldsymbol{\kappa}}(p).$$

We now follow the steps of the proof given in Section 3.

5.1 Productization for Non-homogeneous Stable Polynomials

We first show how we can reduce the problem of bounding the capacity to products of affine linear forms. We recall the main productization result from [22], which gives the non-homogeneous productization result as an immediate corollary.

▶ **Theorem 16** (Thm. 6.2, [22])**.** *Fix $n, d \in \mathbb{N}$, $\boldsymbol{u}, \boldsymbol{\alpha} \in \mathbb{R}^n_{\geq 0}$, and $p \in \mathbb{R}_{\geq 0}[x_1, \ldots, x_n]$ of homogeneous degree d, such that $p(\mathbf{1}) = 1$ and $\nabla p(\mathbf{1}) = \boldsymbol{\alpha}$. There exists an $\mathbb{R}_{\geq 0}$-valued $d \times n$ matrix A such that the rows sums of A are all equal to $\mathbf{1}$, the column sums of A are given by $\boldsymbol{\alpha}$, and $p(\boldsymbol{u}) = \prod_{i=1}^{d} (A\boldsymbol{u})_i$.*

▶ **Corollary 17.** *Fix $n, d \in \mathbb{N}$, $\boldsymbol{u}, \boldsymbol{\alpha} \in \mathbb{R}^n_{\geq 0}$, and $p \in \mathrm{NHStab}^d_n(\boldsymbol{\alpha})$. There exists $f \in \mathrm{NHProd}^d_n(\boldsymbol{\alpha})$ such that $p(\boldsymbol{u}) = f(\boldsymbol{u})$.*

Proof. Let $q(\boldsymbol{x}) = x_{n+1}^d \cdot p\left(\frac{x_1}{x_{n+1}}, \ldots, \frac{x_n}{x_{n+1}}\right)$ be the homogenization of p, and define $\boldsymbol{\beta} := \nabla q(\mathbf{1})$. So $q \in \mathbb{R}_{\geq 0}[x_1, \ldots, x_{n+1}]$ of homogeneous degree d such that $q(\mathbf{1}) = 1$ and $\nabla q(\mathbf{1}) = \boldsymbol{\beta} = (\alpha_1, \ldots, \alpha_n, d - \|\boldsymbol{\alpha}\|_1)$. Define $u_{n+1} := 1$, apply Theorem 16 to q and $\boldsymbol{u}$, and dehomogenize to obtain the desired result. ◀

We now use this result to reduce the problem of bounding the capacity to products of affine linear forms.

▶ **Corollary 18.** *For $p \in \mathrm{NHStab}^d_n(\boldsymbol{\alpha})$, we have*

$$\mathrm{Cap}_{\kappa}(p) \geq L^{\mathrm{NHProd}}_n(\boldsymbol{\alpha}; \boldsymbol{\kappa}).$$

Proof. For any $\boldsymbol{x} \in \mathbb{R}^n_{\geq 0}$, let $f \in \mathrm{NHProd}^d_n(\boldsymbol{\alpha})$ be such that $p(\boldsymbol{x}) = f(\boldsymbol{x})$ according to Corollary 17. With this, we have

$$\mathrm{Cap}_{\boldsymbol{\kappa}}(p) = \inf_{\boldsymbol{x} > 0} \frac{p(\boldsymbol{x})}{\boldsymbol{x}^{\boldsymbol{\kappa}}} \geq \inf_{\boldsymbol{x} > 0} \min_{d \in \mathbb{N}} \min_{f \in \mathrm{NHProd}^d_n(\boldsymbol{\alpha})} \frac{f(\boldsymbol{x})}{\boldsymbol{x}^{\boldsymbol{\kappa}}} = L^{\mathrm{NHProd}}_n(\boldsymbol{\alpha}; \boldsymbol{\kappa}).$$ ◀

5.2 The Extreme Points of $\mathbf{NHMat}_{\boldsymbol{n}}^{\boldsymbol{d}}(\boldsymbol{\alpha})$

The next result implies we can reduce to the extreme points of $\mathrm{NHMat}_n^d(\boldsymbol{\alpha})$ to lower bound $L_n^{\mathrm{NHProd}}(\boldsymbol{\alpha};\boldsymbol{\kappa})$.

▶ **Lemma 19** (See Thm. 3.1 of [10]). *Given $\boldsymbol{\kappa} \in \mathbb{Z}_{\geq 0}^n$ and $\boldsymbol{\alpha} \in \mathbb{R}_{\geq 0}^n$, let $\phi : \mathrm{NHMat}_n^d(\boldsymbol{\alpha}) \to \mathbb{R}_{\geq 0}$ be the function which maps M to $\mathrm{Cap}_{\boldsymbol{\kappa}}(p)$ where p is the polynomial associated to M. Then ϕ is log-concave on $\mathrm{NHMat}_n^d(\boldsymbol{\alpha})$.*

We next describe the extreme points of $\mathrm{NHMat}_n^d(\boldsymbol{\alpha})$ via bipartitioned forests.

▶ **Lemma 20.** *Any extreme point of $\mathrm{NHMat}_n^d(\boldsymbol{\alpha})$ has support given by a bipartite forest on d left vertices and $n+1$ right vertices.*

Proof. Let M be an extreme point of $\mathrm{NHMat}_n^d(\boldsymbol{\alpha})$, and suppose its bipartite support graph G does not give a forest. Then G must contain an even simple cycle. Group the edges of this cycle into two groups such that the odd edges make up one group, and the even edges make up the other (with any starting point). Add $\epsilon > 0$ to all matrix entries corresponding to even edges and subtract ϵ to all matrix entries corresponding to odd edges, to construct $M_+ \in \mathrm{NHMat}_n^d(\boldsymbol{\alpha})$. Do the same thing, but reverse the signs, to construct $M_- \in \mathrm{NHMat}_n^d(\boldsymbol{\alpha})$. Thus $M = \frac{M_+ + M_-}{2}$, contradicting the fact that M is an extreme point. ◀

In what follows, we will also need the following basic graph theoretic result.

▶ **Lemma 21.** *Let G be a bipartite forest on m left vertices and n right vertices such that G has no vertices of degree 0. Then G has at least $m - n + 1$ left leaves.*

5.3 Capacity Bounds via Induction

Here we complete the proof of Theorem 1. We first give a simple lemma, which bears some resemblance to the probabilistic union bound.

▶ **Lemma 22.** *Given $\boldsymbol{c} \in \mathbb{R}_{\geq 0}^d$ such that $c_i < 1$ for all $i \in [d]$, we have $1 - \sum_{i=1}^d c_i \leq \prod_{i=1}^d (1 - c_i)$.*

The next lemma handles the case from Step 4 in Section 3 of setting some variable equal to 0. To see how these next two lemmas actually are actually used, see Theorem 25 below.

▶ **Lemma 23.** *For $n \geq 1$, let $p \in \mathrm{NHProd}_n^d(\boldsymbol{\alpha})$ be the polynomial associated to a $d \times (n+1)$ matrix M, and suppose $\boldsymbol{\kappa} \in \mathbb{Z}_{\geq 0}^n$ such that $\kappa_n = 0$ and $\alpha_n - \kappa_n \leq 1 - \epsilon$ for some $\epsilon > 0$. Then there exists $q \in \mathrm{NHProd}_{n-1}^d(\boldsymbol{\beta})$ such that*

$$\mathrm{Cap}_{\boldsymbol{\kappa}}(p) \geq \epsilon \cdot \mathrm{Cap}_{\boldsymbol{\gamma}}(q),$$

where $\boldsymbol{\gamma} = (\kappa_1, \ldots, \kappa_{n-1}) \in \mathbb{Z}_{\geq 0}^{n-1}$ and $\boldsymbol{\beta} \in \mathbb{R}_{\geq 0}^{n-1}$ is such that for all $S \subseteq [n-1]$ we have

$$\sum_{j \in S} (\alpha_j - \kappa_j) \leq \sum_{j \in S} (\beta_j - \gamma_j) \leq (\alpha_n - \kappa_n) + \sum_{j \in S} (\alpha_j - \kappa_j).$$

The next lemma handles the case from Step 4 in Section 3 of taking the partial derivative with respect to some variable.

▶ **Lemma 24.** *For $n \geq 1$, let $p \in \mathrm{NHProd}_n^d(\boldsymbol{\alpha})$ be the polynomial associated to a $d \times (n+1)$ matrix M such that column n has exactly one non-zero entry, and suppose $\boldsymbol{\kappa} \in \mathbb{Z}_{\geq 0}^n$ is such that $\kappa_n = 1$ and $\kappa_n - \alpha_n \leq 1 - \epsilon$ for some $\epsilon > 0$. Then there exists $q \in \mathrm{NHProd}_{n-1}^{d-1}(\boldsymbol{\beta})$ such that*

$$\mathrm{Cap}_{\boldsymbol{\kappa}}(p) \geq \epsilon \cdot \mathrm{Cap}_{\boldsymbol{\gamma}}(q),$$

where $\boldsymbol{\gamma} = (\kappa_1, \ldots, \kappa_{n-1}) \in \mathbb{Z}_{\geq 0}^{n-1}$ and $\boldsymbol{\beta} \in \mathbb{R}_{\geq 0}^{n-1}$ is such that for all $S \subseteq [n-1]$ we have

$$\sum_{j \in S} (\kappa_j - \alpha_j) \leq \sum_{j \in S} (\gamma_j - \beta_j) \leq (\kappa_n - \alpha_n) + \sum_{j \in S} (\kappa_j - \alpha_j).$$

We now combine the lemmas above to prove our strongest capacity lower bound for polynomials in $\mathrm{NHProd}_n^d(\boldsymbol{\alpha})$.

▶ **Theorem 25.** *Fix any $\boldsymbol{\kappa} \in \mathbb{Z}_{\geq 0}^n$, $\boldsymbol{\alpha} \in \mathbb{R}_{\geq 0}^n$, and $\boldsymbol{\epsilon}, \boldsymbol{\delta} \in \mathbb{R}_{>0}^n$ such that*

$$\max_{S \in \binom{[n]}{k}} \sum_{j \in S} (\alpha_j - \kappa_j) \leq 1 - \epsilon_k \qquad \text{and} \qquad \max_{S \in \binom{[n]}{k}} \sum_{j \in S} (\kappa_j - \alpha_j) \leq 1 - \delta_k$$

for all $k \in [n]$. Then

$$\mathrm{Cap}_{\boldsymbol{\kappa}}(p) \geq \min_{0 \leq \ell \leq n} \prod_{k=1}^{\ell} \epsilon_k \prod_{k=1}^{n-\ell} \delta_k$$

for every $p \in \mathrm{NHProd}_n^d(\boldsymbol{\alpha})$.

The next result gives the bound which we will use in Section 4 to prove Theorem 6, the simply exponential improvement to the probability bound used for the metric TSP application.

▶ **Corollary 26.** *Fix any $\boldsymbol{\kappa} \in \mathbb{Z}_{\geq 0}^n$ and $\boldsymbol{\alpha} \in \mathbb{R}_{\geq 0}^n$ such that $\boldsymbol{\epsilon} \in \mathbb{R}_{>0}^n$ can be defined via*

$$\epsilon_k := 1 - \max_{\substack{S \subseteq [n] \\ |S| \leq k}} \left| \sum_{j \in S} (\kappa_j - \alpha_j) \right|$$

for all $k \in [n]$. Then $\mathrm{Cap}_{\boldsymbol{\kappa}}(p) \geq \prod_{k=1}^n \epsilon_k$ for every $p \in \mathrm{NHProd}_n^d(\boldsymbol{\alpha})$.

By Corollary 18, the above results hold for all $p \in \mathrm{NHStab}_n^d(\boldsymbol{\alpha})$, and we state this formally now.

▶ **Corollary 27.** *Theorem 25 and Corollary 26 hold for all $p \in \mathrm{NHStab}_n^d(\boldsymbol{\alpha})$.*

5.4 Proving Theorem 1 and Theorem 2

We now complete the proof of Theorem 1, and thus also of Corollary 2. Fix $p \in \mathrm{NHStab}_n^d(\boldsymbol{\alpha})$. Thus

$$1 - \max_{\substack{S \subseteq [n] \\ |S| \leq k}} \left| \sum_{j \in S} (\kappa_j - \alpha_j) \right| \geq 1 - \|\boldsymbol{\kappa} - \boldsymbol{\alpha}\|_1$$

and Corollary 27 (via Corollary 26) imply

$$\mathrm{Cap}_{\boldsymbol{\kappa}}(p) \geq (1 - \|\boldsymbol{\kappa} - \boldsymbol{\alpha}\|_1)^n,$$

which completes the proof. The tightness claim follows from Lemma 28.

5.5 Examples and Tightness

The δ parameters in Theorem 25 are tight, and this is shown in Example 29. However, the ϵ parameters are not, as shown in Example 30. Example 29 also demonstrates tightness of the dependence on the error parameter for some of our results, and we state this formally now.

▶ **Lemma 28.** *The dependence on* $(1 - \|\boldsymbol{\kappa} - \nabla p(\mathbf{1})\|_1)$ *in Theorem 1 and Corollary 2 and the dependence on* ϵ *in Theorem 6 are all tight for any fixed* $\boldsymbol{\kappa} > \mathbf{0}$.

Proof. Define $\alpha_1 := \kappa_1 - (1-\epsilon) > 0$ and $\alpha_i := \kappa_i \geq 1$ for all $i \geq 2$. Let p be the polynomial described by Example 29, given explicitly by

$$p(\boldsymbol{x}) = \left(\prod_{i=1}^{n} x_i^{\kappa_i - 1}\right) \cdot \left(\prod_{i=1}^{n-1} (\epsilon x_i + (1-\epsilon)x_{i+1})\right) \cdot (\epsilon x_n + (1-\epsilon)).$$

Then $\boldsymbol{\kappa}$ is a vertex of of the Newton polytope of p. Thus

$$p_{\boldsymbol{\kappa}} = \mathrm{Cap}_{\boldsymbol{\kappa}}(p) = \prod_{k=1}^{n} \left(1 - \sum_{j=1}^{k} (\kappa_j - \alpha_j)\right) = \epsilon^n = (1 - \|\boldsymbol{\kappa} - \nabla p(\mathbf{1})\|_1)^n.$$

To see that p can be the probability generating polynomial for some random variables associated to a strongly Rayleigh distribution (as in Theorem 6), note that the polarization of p is real stable and gives the probability generating polynomial for such a strongly Rayleigh distribution. ◀

▶ **Example 29.** Fix any $\boldsymbol{\kappa} \in \mathbb{Z}^n_{\geq 0}$ and $\boldsymbol{\alpha} \in \mathbb{R}^n_{\geq 0}$ such that $\kappa_j - \alpha_j \geq 0$ for all $j \in [n]$ and $\|\boldsymbol{\kappa} - \boldsymbol{\alpha}\|_1 < 1$. Thus $\kappa_j \geq 1$ for all $j \in [n]$. For $d = \|\boldsymbol{\kappa}\|_1$, consider the matrix $M = \left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right]$, where A is the $(\|\boldsymbol{\kappa}\| - n) \times (n+1)$ matrix given by

$$A = \begin{bmatrix} \mathbf{1}_{\kappa_1 - 1} \boldsymbol{e}_1^\top \\ \mathbf{1}_{\kappa_2 - 1} \boldsymbol{e}_2^\top \\ \vdots \\ \mathbf{1}_{\kappa_n - 1} \boldsymbol{e}_n^\top \end{bmatrix},$$

and B is the $n \times (n+1)$ matrix for which

$$b_{kk} = 1 - \sum_{j=1}^{k} (\kappa_j - \alpha_j), \qquad b_{k,k+1} = \sum_{j=1}^{k} (\kappa_j - \alpha_j),$$

for all $k \in [n]$ and $b_{ij} = 0$ otherwise. Note that every row sum of A is equal to 1, and the row sums of B are given by

$$\sum_{j=1}^{n} b_{kj} = 1 - \sum_{j=1}^{k} (\kappa_j - \alpha_j) + \sum_{j=1}^{k} (\kappa_j - \alpha_j) = 1$$

for all $k \in [n]$. The column sums of M are then given by

$$\sum_{i=1}^{d} m_{ik} = (\kappa_k - 1) + 1 - \sum_{j=1}^{k} (\kappa_j - \alpha_j) + \sum_{j=1}^{k-1} (\kappa_j - \alpha_j) = \kappa_k - (\kappa_k - \alpha_k) = \alpha_k$$

for all $k \in [n]$. Thus $M \in \mathrm{NHMat}_n^d(\boldsymbol{\alpha})$. Let p be the polynomial associated to M, and let q be the polynomial associated to B. We then have $\mathrm{Cap}_{\boldsymbol{\kappa}}(p) = \mathrm{Cap}_{\mathbf{1}}(q)$. Note that $\mathbf{1}$ is a vertex of the Newton polytope of q, and thus

$$\mathrm{Cap}_{\boldsymbol{\kappa}}(p) = \mathrm{Cap}_{\mathbf{1}}(q) = \prod_{k=1}^{n} b_{kk} = \prod_{k=1}^{n} \left(1 - \sum_{j=1}^{k} (\kappa_j - \alpha_j)\right).$$

By possibly permuting the variables to put $\kappa_j - \alpha_j$ in non-increasing order, this is precisely the lower bound guaranteed by Theorem 25.

▶ **Example 30.** Consider the case that $\boldsymbol{\kappa} = \mathbf{0}$ and $\|\boldsymbol{\alpha}\|_1 < 1$. Given any d, let M be any extreme point of $\mathrm{NHMat}_n^d(\boldsymbol{\alpha})$. Then the column sum of column $n+1$ of M is equal to $d - \sum_{j=1}^{n} \alpha_j > d - 1$. Since every row sum equals 1, all entries of column $n+1$ of M are strictly positive. Since M is an extreme point, this further implies that each column of M has at most 1 positive entry except column $n+1$. Letting p be the polynomial associated to M, there exists a partition $S_1 \sqcup \cdots \sqcup S_k = [n]$ such that

$$p(\boldsymbol{x}) = \prod_{i=1}^{k} \left(\left(\sum_{j \in S_i} \alpha_j x_j \right) + \left(1 - \sum_{j \in S_i} \alpha_j \right) \right).$$

Thus by Lemma 22,

$$\mathrm{Cap}_{\mathbf{0}}(p) = p_{\mathbf{0}} = \prod_{i=1}^{k} \left(1 - \sum_{j \in S_i} \alpha_j \right) \geq 1 - \sum_{j=1}^{n} \alpha_j.$$

By Lemma 19, this gives a lower bound on the capacity of every $p \in \mathrm{NHProd}_n^d(\boldsymbol{\alpha})$. However, this lower bound is strictly better than the one guaranteed by Theorem 25.

As a note, this can be partially remedied by removing all $\kappa_j = 0$ columns at the same time (i.e., adjusting Lemma 23 to remove many columns at once). However, it is currently unclear how to inductively do this correctly.

6 Uniqueness of Permanent Minimizers

In this section, let $\mathrm{Mat}_n(\boldsymbol{c})$ be the set of $n \times n$ matrices with non-negative entries and rows sums $\mathbf{1}$ and column sums $\boldsymbol{c} > 0$, let $p_M(\boldsymbol{x}) := \prod_{i=1}^{n} \sum_{j=1}^{n} m_{ij} x_j$ be the real stable polynomial associated to a given $M \in \mathrm{Mat}_n(\boldsymbol{c})$, and let $L(\boldsymbol{c})$ be any lower bound on $\mathrm{Cap}_{\mathbf{1}}(p_M)$ over all $M \in \mathrm{Mat}_n(\boldsymbol{c})$ (e.g., as given by Theorem 1 above or the results of [22]).

We now prove Theorem 3 via Theorem 32 below. We note that the argument in the proof of Theorem 32 given below can be made into a general statement about minimizers of quadratic forms. And further, the same argument given here applies to mixed discriminants, as mentioned in Section 2.

▶ **Lemma 31.** *If* $\frac{(n-2)^{n-2} n^{n-1}}{(n-1)^{2n-3}} > \frac{c_1 c_2 \cdots c_n}{L(\boldsymbol{c})}$, *then all minimizers of the permanent over over* $\mathrm{Mat}_n(\boldsymbol{c})$ *have all strictly positive entries.*

Proof. Note that the rank-one matrix $\frac{1}{n} \mathbf{1} \cdot \boldsymbol{c}^{\top}$ has all strictly positive entries and permanent equal to $\frac{n!}{n^n} \prod_{i=1}^{n} c_i$. Thus, to obtain a contradiction, let us assume that there exists $M \in \mathrm{Mat}_n(\boldsymbol{c})$ with at least one zero entry such that $\mathrm{per}(M) \leq \frac{n!}{n^n} \prod_{i=1}^{n} c_i$. By the main result of [18], we have

$$\frac{n!}{n^n} \prod_{i=1}^{n} c_i \geq \mathrm{per}(M) \geq \left(\frac{n-2}{n-1} \right)^{n-2} \frac{(n-1)!}{(n-1)^{n-1}} \cdot L(\boldsymbol{c}),$$

which after rearranging implies

$$\frac{(n-2)^{n-2}n^{n-1}}{(n-1)^{2n-3}} > \left(\frac{n-2}{n-1}\right)^{n-2} \frac{n^{n-1}}{(n-1)^{n-1}},$$

which is a contradiction. ◀

▶ **Theorem 32.** *If* $\frac{(n-2)^{n-2}n^{n-1}}{(n-1)^{2n-3}} > \frac{c_1 c_2 \cdots c_n}{L(\boldsymbol{c})}$, *then the rank-one matrix* $\frac{1}{n}\mathbf{1} \cdot \boldsymbol{c}^\top$ *is the unique minimizer of the permanent over* $\mathrm{Mat}_n(\boldsymbol{c})$.

Proof. Let M be a minimizer of the permanent over $\mathrm{Mat}_n(\boldsymbol{c})$. Thus every entry of M is positive by Lemma 31. We will show that every pair of rows of M must be equal, which immediately implies $M = \frac{1}{n}\mathbf{1} \cdot \boldsymbol{c}^\top$.

Let $P \subset \mathbb{R}^{2\times n}$ be the convex polytope given by the first two rows of all matrices in $\mathrm{Mat}_n(\boldsymbol{c})$ with rows 3 through n equal to those of M. Thus by positivity, $(\boldsymbol{m}_1, \boldsymbol{m}_2)$ is in the relative interior of P where $\boldsymbol{m}_i$ is the i^{th} row of M. Let $f(\boldsymbol{u}, \boldsymbol{v})$ be defined as the permanent of the matrix M with the first two rows replaced by $(\boldsymbol{u}, \boldsymbol{v})$. Since M minimizes the permanent and $(\boldsymbol{m}_1, \boldsymbol{m}_2)$ is contained in the relative interior of P, the necessary conditions on the minimum given by Lagrange multipliers implies

$$\nabla f(\boldsymbol{m}_1, \boldsymbol{m}_2) = \big(\mathrm{per}(M_{(i,j)})\big)_{i\in[2],j\in[n]} = (a_i + b_j)_{i\in[2],j\in[n]}$$

for some a_i, b_j, where $M_{(i,j)}$ is the matrix M with row i and column j deleted. That is, the gradient of f points in a direction orthogonal to P at $(\boldsymbol{m}_1, \boldsymbol{m}_2)$.

By row symmetry of the permanent, we also have that $\nabla f(\boldsymbol{m}_2, \boldsymbol{m}_1) = (a'_i + b_j)_{i\in[2],j\in[n]}$, where $a'_1 = a_2$ and $a'_2 = a_1$. By row multilinearity of the permanent, we thus have

$$\nabla f(t \cdot \boldsymbol{m}_1 + (1-t) \cdot \boldsymbol{m}_2, (1-t) \cdot \boldsymbol{m}_1 + t \cdot \boldsymbol{m}_2) = (t \cdot a_i + (1-t) \cdot a'_i + b_j)_{i\in[2],j\in[n]}$$

for all $t \in \mathbb{R}$. If $\boldsymbol{m}_1 \neq \boldsymbol{m}_2$ then the gradient of f is orthogonal to P at all points on a line in P through M, and thus the permanent is minimized at all these points. At least one such point (on the boundary of P) has a zero entry, which contradicts Lemma 31. Therefore it must be that $\boldsymbol{m}_1 = \boldsymbol{m}_2$. Applying this argument to every pair of rows of M implies the desired result. ◀

We now give the example which proves Proposition 4.

▶ **Example 33.** Fix $t > 0$ and $n \in \mathbb{N}$, and define $\epsilon := \frac{1}{n^{1+t}}$. Further define $\boldsymbol{\alpha} \in \mathbb{R}^n_{>0}$ and $\boldsymbol{c} \in \mathbb{R}^{n+1}_{>0}$ via

$$\boldsymbol{\alpha} := \big(1-\epsilon, 1-\epsilon, \ldots, 1-\epsilon\big) \in \mathbb{R}^n_{>0} \quad \text{and} \quad \boldsymbol{c} := \Big(1+\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_n, \sum_{j=1}^{n}(1-\alpha_j)\Big) \in \mathbb{R}^{n+1}_{>0}.$$

Note that $\|\boldsymbol{c} - \mathbf{1}\|_1 = 1 + (n-2)\epsilon + 1 - n\epsilon = 2(1-\epsilon) < 2$. We first have

$$\mathrm{per}\left(\frac{1}{n+1}\mathbf{1} \cdot \boldsymbol{c}^\top\right) = \frac{(n+1)!}{(n+1)^{n+1}}(2-\epsilon)(1-\epsilon)^{n-1}n\epsilon = \frac{n!}{n^{(1+t)n}} \cdot \frac{2-n^{-1-t}}{1-n^{-1-t}} \cdot \frac{(n^{1+t}-1)^n}{n^t(n+1)^n}.$$

Now consider the $(n+1) \times (n+1)$ matrix with diagonal entries

$$1, \quad \sum_{j=1}^{1}(1-\alpha_j), \quad \sum_{j=1}^{2}(1-\alpha_j), \quad \cdots, \quad \sum_{j=1}^{n}(1-\alpha_j),$$

subdiagonal entries to obtain row sums $\mathbf{1}$ and column sums $\boldsymbol{c}$, and zero entries elsewhere. Note that $M \in \mathrm{Mat}_{n+1}(\boldsymbol{c})$. The matrix M is upper-triangular, and thus we have

$$\mathrm{per}(M) = \prod_{k=1}^{n} \sum_{j=1}^{k} (1 - \alpha_j) = \prod_{k=1}^{n} (k\epsilon) = n! \cdot \epsilon^n = \frac{n!}{n^{(1+t)n}}.$$

We then further have

$$\frac{2 - n^{-1-t}}{1 - n^{-1-t}} \cdot \frac{(n^{1+t} - 1)^n}{n^t (n+1)^n} \approx 2n^{(n-1)t} \cdot \frac{(n - n^{-t})^n}{(n+1)^n} \geq 2n^{(n-1)t} \cdot \frac{(n-1)^n}{(n+1)^n} > 1,$$

which implies $\mathrm{per}(M) < \mathrm{per}\left(\frac{1}{n+1}\mathbf{1} \cdot \boldsymbol{c}^\top\right)$ for large enough n.

References

1. Kareem Adiprasito, June Huh, and Eric Katz. Hodge theory for combinatorial geometries. *Annals of Mathematics*, 188(2):381–452, 2018.
2. Yeganeh Alimohammadi, Nima Anari, Kirankumar Shiragur, and Thuy-Duong Vuong. Fractionally log-concave and sector-stable polynomials: counting planar matchings and more. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 433–446, 2021.
3. Nima Anari, Kuikui Liu, Shayan Oveis Gharan, and Cynthia Vinzant. Log-concave polynomials iii: Mason's ultra-log-concavity conjecture for independent sets of matroids. *arXiv preprint*, 2018. `arXiv:1811.01600`.
4. Nima Anari, Kuikui Liu, Shayan Oveis Gharan, and Cynthia Vinzant. Log-concave polynomials ii: High-dimensional walks and an fpras for counting bases of a matroid. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1–12, 2019.
5. Nima Anari, Tung Mai, Shayan Oveis Gharan, and Vijay V Vazirani. Nash social welfare for indivisible items under separable, piecewise-linear concave utilities. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2274–2290. SIAM, 2018.
6. Nima Anari and Shayan Oveis Gharan. The kadison-singer problem for strongly rayleigh measures and applications to asymmetric tsp. *arXiv preprint*, 2014. `arXiv:1412.1143`.
7. Nima Anari and Shayan Oveis Gharan. A generalization of permanent inequalities and applications in counting and optimization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 384–396, 2017.
8. Nima Anari, Shayan Oveis Gharan, and Cynthia Vinzant. Log-concave polynomials, entropy, and a deterministic approximation algorithm for counting bases of matroids. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 35–46. IEEE, 2018.
9. Arash Asadpour, Michel X Goemans, Aleksander Madry, Shayan Oveis Gharan, and Amin Saberi. An o (log n/log log n)-approximation algorithm for the asymmetric traveling salesman problem. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete algorithms*, pages 379–389, 2010.
10. Alexander Barvinok. Enumerating contingency tables via random permanents. *Combinatorics, Probability and Computing*, 17(1):1–19, 2008.
11. Alexander Barvinok. Computing the permanent of (some) complex matrices. *Foundations of Computational Mathematics*, 16:329–342, 2016.
12. Julius Borcea, Petter Brändén, and Thomas Liggett. Negative dependence and the geometry of polynomials. *Journal of the American Mathematical Society*, 22(2):521–567, 2009.
13. Petter Brändén. Hyperbolic polynomials and the kadison-singer problem. *arXiv preprint*, 2018. `arXiv:1809.03255`.

14 Petter Brändén and June Huh. Lorentzian polynomials. *Annals of Mathematics*, 192(3):821–891, 2020.

15 Petter Brändén, Jonathan Leake, and Igor Pak. Lower bounds for contingency tables via lorentzian polynomials. *Israel Journal of Mathematics*, 253(1):43–90, 2023.

16 Gregory P Egorychev. The solution of van der waerden's problem for permanents. *Advances in Mathematics*, 42(3):299–305, 1981.

17 Dmitry I Falikman. Proof of the van der waerden conjecture regarding the permanent of a doubly stochastic matrix. *Mathematical notes of the Academy of Sciences of the USSR*, 29:475–479, 1981.

18 Leonid Gurvits. Hyperbolic polynomials approach to van der waerden/schrijver-valiant like conjectures: sharper bounds, simpler proofs and algorithmic applications. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 417–426, 2006.

19 Leonid Gurvits. The van der waerden conjecture for mixed discriminants. *Advances in Mathematics*, 200(2):435–454, 2006.

20 Leonid Gurvits. On multivariate newton-like inequalities. In *Advances in Combinatorial Mathematics: Proceedings of the Waterloo Workshop in Computer Algebra 2008*, pages 61–78. Springer, 2009.

21 Leonid Gurvits. A polynomial-time algorithm to approximate the mixed volume within a simply exponential factor. *Discrete & Computational Geometry*, 41:533–555, 2009.

22 Leonid Gurvits and Jonathan Leake. Capacity lower bounds via productization. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 847–858, 2021.

23 Anna Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved bound on the integrality gap of the subtour lp for tsp. In *FOCS*, pages 844–855. IEEE Computer Society, 2022.

24 Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric tsp. In *STOC*. ACM, 2021.

25 Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A deterministic better-than-3/2 approximation algorithm for metric tsp. In Alberto Del Pia and Volker Kaibel, editors, *Integer Programming and Combinatorial Optimization*, pages 261–274, Cham, 2023. Springer International Publishing.

26 Adam W Marcus, Daniel A Spielman, and Nikhil Srivastava. Interlacing families ii: Mixed characteristic polynomials and the kadison—singer problem. *Annals of Mathematics*, pages 327–350, 2015.

27 Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 550–559. IEEE, 2011.

28 Damian Straszak and Nisheeth K Vishnoi. Real stable polynomials and matroids: Optimization and counting. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 370–383, 2017.

Distributed Fast Crash-Tolerant Consensus with Nearly-Linear Quantum Communication

Mohammad T. HajiAghayi
University of Maryland, College Park, MD, USA

Dariusz R. Kowalski
School of Computer and Cyber Sciences, Augusta University, GA, USA

Jan Olkowski
University of Maryland, College Park, MD, USA

Abstract

Fault-tolerant *Consensus* is about reaching agreement on some of the input values in a limited time by non-faulty autonomous processes, despite of failures of processes or communication medium. This problem is particularly challenging and costly against an adaptive adversary with full information. Bar-Joseph and Ben-Or (PODC'98) were the first who proved an absolute lower bound $\Omega(\sqrt{n/\log n})$ on expected time complexity of Consensus in any classical (i.e., randomized or deterministic) message-passing network with n processes succeeding with probability 1 against such a strong adaptive adversary crashing processes.

Seminal work of Ben-Or and Hassidim (STOC'05) broke the $\Omega(\sqrt{n/\log n})$ barrier for consensus in the classical (deterministic and randomized) networks by enhancing the model with quantum channels. In such networks, quantum communication between every pair of processes participating in the protocol is also allowed. They showed an (expected) constant-time quantum algorithm for a linear number of crashes $t < n/3$.

In this paper, we improve upon that seminal work by reducing the number of quantum and communication bits to an arbitrarily small polynomial, and even more, to a polylogarithmic number – though, the latter in the cost of a slightly larger polylogarithmic time (still exponentially smaller than the time lower bound $\Omega(\sqrt{n/\log n})$ for the classical computation models).

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Distributed algorithms

Keywords and phrases distributed algorithms, quantum algorithms, adaptive adversary, crash failures, Consensus, quantum common coin, approximate counting

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.80

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2305.10618`

Funding *Mohammad T. HajiAghayi*: Partially supported by DARPA QuICC, ONR MURI 2024 award on Algorithms, Learning, and Game Theory, Army-Research Laboratory (ARL) grant W911NF2410052, NSF AF:Small grants 2218678, 2114269, 2347322.
Jan Olkowski: Partially supported by DARPA QuICC, ONR MURI 2024 award on Algorithms, Learning, and Game Theory, Army-Research Laboratory (ARL) grant W911NF2410052, NSF AF:Small grants 2218678, 2114269, 2347322.

1 Introduction

Consensus is about making a common decision among the processes' initial input values in a limited time by every non-faulty process, despite the faulty behaviour of some of the players. Since its introduction by Pease, Shostak and Lamport [31] (JACM'80), who ruled out trivial solutions (such as always deciding on the same bit), fault-tolerant Consensus has constantly

been among foundation problems in distributed computing. This problem has been studied in synchronous/asynchronous and deterministic/randomized computation models and under various fault-tolerant or adversarial models: Fail-Stop (crashes) and Byzantine, Static and Adaptive, Computationally Bounded and Unbounded Adversaries - just to name a few (see Section 2.1 for related work).

While the landscape of Consensus problem under the classic model of computation is well-developed, much less is known if we allow for quantum computation. The seminal work of Ben-Or and Hassidim [9] (albeit a short 5-pager in STOC'05) broke the $\Omega(\sqrt{n/\log n})$ rounds time barrier for classic computing by employing quantum computing to further use the power of randomization in distributed computing. They showed an (expected) constant-time quantum algorithm for a linear number of crashes $t < n/3$, however, the algorithm is inefficient in terms of communication bits and, even more importantly, in terms of the number of quantum bits (qubits), as it uses $\Omega(n)$ of them per process. Since then no algorithm has managed to reduce the quantum resources needed to solve Consensus. Because generating, maintaining and sending quantum bits is extremely costly (today's quantum devices use less than 500 qubits), thus the main question of our paper emerges naturally:

Could the number of qubits be substantially reduced without harming the time complexity?

1.1 Distributed setting

We consider a quantum synchronous message-passing model (c.f., [6]), consisting of n synchronous processes (also called players), each with common clock (a clock tick is called a round or a step) and unique id from the known set $\mathcal{P} = [n] = \{1, \ldots, n\}$.

Between any pair of processes we assume the existence of a quantum channel being able to transmit reliable[1] messages caring quantum bits, qubits. For the sake of completeness, we also augment the model by classic point-to-point channels between any pair of processes. In each round, a process can send (personalized) quantum and classic messages to any selected subset of other processes. After *multicasting messages*, in the same round a process *receives messages* that were just sent to it by other processes, and performs *local computation*, which involves both quantum and classic bits.[2]

Processes are prone to *crash failures*, also called *fail-stops*. A crashed process permanently stops any activity, including sending and receiving messages.

We model crashes as incurred by a full-information *adversary* (the same as in [7, 9]) that knows the algorithm, the exact pure quantum state (see Section 3) and the classic state of the system at any point of an execution, and has an unbounded computational power. The adversary decides which processes to fail and when. The adversary is also *adaptive* – it can make a decision on-line based on its current full-knowledge of the system. However, the adversary does not know the future computation, which means that it does not know future random bits drawn by processes.

As for the quantum part, the adversary can apply no quantum operation to the system, but it is aware of all quantum and classic changes of state that the network undergoes. If a process is crashed by the adversary, we assume that its quantum bits are not destroyed (in particular, entangled qubits in other processes do not collapse but maintain their entanglement), however they cannot be used in further computation.

[1] Messages are not lost nor corrupted while in transit.
[2] Local computation also decides what messages to send in the next round and to whom.

Failures are *not clean* – when a process crashes when attempting to multicast a message, then some of the recipients may receive the message and some may not; this aspect is controlled by the adversary. A *t-adversary* is additionally restricted by the the number of crashed processes being *smaller than* t; if $t = n$ then the n-adversary is also called an unbounded adversary (note that even in such case, at least one process must survive for Consensus to make sense). Throughout the paper, we will be calling the adversary described above "adaptive", for short.

Consensus problem. each process p has its own initial value $input_p$ and has to output a (common) decision value, so that the following conditions hold: *validity* – decision must be on an input value of some process; *agreement* – no two processes decide on different values; and *termination* – each process eventually decides on some value, unless it is faulty. All those three requirements must hold with probability 1. We focus on *binary Consensus*, in which initial values are in $\{0, 1\}$.

Correctness and complexity – in terms of time (the number of rounds needed for all processes to terminate) and the number of quantum bits (qubits) and communication bits – are analyzed and maximized (worst-case) against an adaptive adversary.

We say that a random event occurs *with high probability* (*whp* for short), if its probability could be made $1 - O(n^{-c})$ for any sufficiently large positive constant c by linear scaling of parameters.

2 Our Results

In this work, we focus on improving quantum bits' and communication complexities (without harming time complexity) of quantum algorithms solving Consensus problem with probability 1 against an adaptive full-information adversary capable of causing processes' crashes. We observe that the maximum, per process, number of communication bits in Consensus problem is $\Omega(n)$, therefore one can only hope to improve amortized communication complexity (per process), see the full version of the paper.[3]

Our first main result is a quantum algorithm that solves Consensus in expected constant number of rounds and amortized number of qubits and classical communication bits per process being an arbitrarily low polynomial. This directly improves, by a polynomial factor, on the result of Ben-Or and Hassidim [9], which required $\Theta(n)$ qubits and communication bits, amortized per process. The detailed description of the algorithm and the proof of its correctness is presented in Section 4.

▶ **Theorem 1.** *For any $\epsilon > 0$, there is an algorithm solving consensus against an adaptive $n/3$-adversary in expected $O(1)$ rounds while using $O(n^\epsilon)$ qubits and communication bits (amortized) per process, w.h.p.*

To achieve this result, we give improved protocols for several existing tools that have been historically used in consensus algorithms. Combining all our advancements together provides a tighter approach to the general problem of consensus.

Our first technique is a new quantum implementation of a weak global coin.

▶ **Definition 2** ([9]). *Let $\mathcal{C}$ be a protocol for n players (with no input), where each player i outputs a (classical) bit $v_i \in \{0, 1\}$. We say that the protocol $\mathcal{C}$ is a t-resilient weak global coin protocol (or computes a weak global coin, for short) with fairness $\rho > 0$, if for any adaptive t-adversary and any value $b \in \{0, 1\}$, with probability at least ρ, $v_i = b$ for all good players i.*

[3] All omitted proofs and materials can be found in the full version of the paper, under the link given in "Related Version" on page 1.

We show a $\frac{1}{3}$-resilient weak global coin that works in constant, $O\left(\frac{1}{\epsilon}\right)$, time, uses an arbitrarily small polynomial number of bits $O(n^\epsilon)$, amortized per process (for any chosen constant $\epsilon > 0$), and results in all non-faulty processes returning the same value with a non-zero constant probability, c.f., Theorem 12. The main idea behind our improvement is to couple a quantum protocol for selecting a leader in the system, proposed in [9], with generalized sparse patterns of fault-tolerant communication induced on random graphs, which in a classical version (i.e., non-quantum) was firstly used in [23].

Our second technical result is a deterministic algorithm for counting, which returns, in constant time, a count of the number of input 1 (or 0, resp.) among the processes. Specifically, our algorithm solves the following problem.

▶ **Definition 3** (Fuzzy Counting [23]). *A Fuzzy Counting is a distributed problem in which every active process is required a number between the initial and the final number of active processes. The notion of "being active" usually depends on the goal of the counting, e.g., all non-faulty processes, non-faulty processes with initial value* 1*, etc.*

The authors of [23] provided an algorithm to this problem that works in $O(\log^3 n)$ rounds and uses $O(\log^7 n)$ communication bits per process. By employing random graphs of asymptotically larger degree than those in [23], we generalize this result by showing a solution that works in constant time. As consequence of this improvement a tradeoff between the number of rounds and the communication complexity arises. Our protocol is faster than the one in [23], but on the other hand it uses a polynomial, yet of an arbitrarily small degree, number of communication bits, amortized per process. On the way to achieve this result, we also propose a new solution to Gossip – a distributed problem in which every non-faulty process, despite the presences of the adversary, have to collect inputs of every other non-faulty processes.[4] The novelty in this approach is that our solution, by exploiting similar properties of random graphs to the approach used for generating a weak global coin, we can solve Gossip in constant time $O\left(\frac{1}{\epsilon}\right)$ while using only $O(n^\epsilon)$ bits per processes, for any constant $1 > \epsilon > 0$. The formal statement of the Gossip algorithm is provided in Theorem 13 and the analysis of our implementation of a weak global coin can be found in the Section 5 and the analysis of our implementation of Fuzzy Counting in Section 6.

Although constant-time algorithms cannot low (sub-polynomial) amortized communication complexity, we show that our main algorithm could be re-instantiated in such a way that it uses only a *polylogarithmic* number of qubits and classical communication bits to solve consensus in just a *polylogarithmically* larger number of rounds (see Section 4, and technical counterpart Theorem 7 for more details).

▶ **Theorem 4.** *There is an algorithm solving consensus against an unbounded adaptive adversary in polylogarithmic number of rounds, in expectation, while using a polylogarithmic number of qubits and communication bits (amortized) per process, whp.*

We believe that the newly developed techniques could be also applied to other types of failures, after failure-specific modifications. For example, although message omission failures require linear amortized communication per process (c.f., [22]), one could still use a small polynomial or even a polylogarithmic number of qubits (together with a linear number of classic communication bits) per process, if qubits are handled according to our techniques while some additional classic communication bits are introduced to handle message omissions. We leave details to follow-up work.

[4] The main difference between Gossip and Fuzzy Counting is that the latter allows for aggregating information, i.e. counting the number of 1's in the system vs collecting a set of identifiers of these active processes that start with 1.

2.1 Previous and Related Work

Consensus in the classic (non-quantum) model. Bar-Joseph and Ben-Or [7] (see also their extended version [8]) proved a lower bound $O(\sqrt{\frac{n}{\log n}})$ on expected time complexity of consensus against an adaptive adversary. They also complemented it by time-optimal randomized algorithm. Their algorithm uses expected $O(\frac{n^{3/2}}{\log n})$ number of communications bits, amortized per process, which has been recently improved by Hajiaghayi et al [23] to $O(\sqrt{n})$ (while maintaining the almost-optimal round complexity $O(\sqrt{n}\log n)$).

Fisher and Lynch [19] proved a lower bound $f+1$ on *deterministic* consensus with f crashes (that actually occurred, i.e., $f < t$), thus separating deterministic solutions from randomized. Regarding communication complexity, Amdur, Weber and Hadzilacos [3] showed that the amortized number of messages per process is at least constant, even in some failure-free execution. Dwork, Halpern and Waarts [18] found a solution with $O(\log n)$ messages per process, but requiring an exponential time, and later Galil, Mayer and Yung [20] developed a message-optimal algorithm working in super-linear $O(n^{1+\varepsilon})$ time, for any $0 < \varepsilon < 1$ and any $f < n$. They also improved the communication further to a constant number of communication bits per process, but the resulting algorithm was exponential in the number of rounds. Chlebus and Kowalski [11] showed that consensus can be solved in $O(f+1)$ time and with $O(\log^2 f)$ messages if only the number $n-f$ of non-faulty processors satisfies $n - f = \Omega(n)$. It was later improved in [12] to $O(f+1)$ time and $O(\text{polylog } n)$ number of communication bits. All the abovementioned communication complexities are amortized per process.

Quantum consensus. To the best of our knowledge, almost all previous papers on quantum consensus concentrated on assuring feasibility of the problem against strong Byzantine adversaries, c.f., [14, 24, 26], or on optimizing time complexity, including the work of Ben-Or and Hassidim [9] achieving constant time against an adaptive adversary.

In recent years, the primary application of (classical) consensus has been in the synchronization of distributed blockchains. A blockchain algorithm based on asymmetric quantum encryption was proposed in [34]. Blockchain algorithms often use leader election as an important subroutine. Quantum algorithms for leader election in anonymous networks, a problem intrinsically unsolved in the classical setting, has been considered in [32, 29]. [36] implemented the first algorithm for the anonymous leader election problem on a quantum computer. A more thorough study of challenges arising from practical implementations of quantum distributed networks is a pending, but unresolved issue [17]. The dynamics of quantum consensus and gossip over a contiguous space was studied in [28].

Sparse quantum communication has been considered by Chlebus, Kowalski and Strojnowski in [13], in the context of solving some version of consensus, but their protocols work correctly only with some probability smaller than 1 and for a specific number of failures corresponding to the probability of success. Another difference is that they used quantum operations to encode the classical inputs in quantum registers and used it to directly solve consensus. In this paper, we show another, more-efficient approach, in which we first create a quantum, weak global coin and later employ this tool to the state-of-the-art framework of solving consensus based on the common coin. Other distributed computing problems, not necessarily fault-prone, were also analyzed in quantum models, c.f., [10, 33, 35]. Finally, the readers interested in more broad treatment of quantum consensus are referenced to a recent survey [27].

More efficient classic randomized solutions against *weak adversaries*. Whenever weaker oblivious adversaries are considered, randomness itself proved to be enough in reducing time complexity to a constant. Chor, Merritt and Shmoys [15] developed constant-time algorithms for consensus against an *oblivious adversary* – that is, the adversary who knows the algorithm but has to decide which process fails and when before the execution starts. Their solution, however, requires a linear number of communication bits per process. Gilbert and Kowalski [21] gave a randomized consensus algorithm that achieves optimal communication complexity of $O(1)$ amortized communication bits per process while terminating in $O(\log n)$ time w.h.p., as long as the number of crashes $f < n/2$.

Classic consensus in more demanding models. Dolev and Reischuk [16] and Hadzilacos and Halpern [22] proved the $\Omega(f+1)$ lower bound on the amortized message complexity per process of deterministic consensus for *omission or (authenticated) Byzantine failures*. However, under some limitation on the adversary and requiring termination only whp, the sublinear expected communication complexity $O(\sqrt{n}\ \text{polylog}\ n)$ per process can be achieved even in case of Byzantine failures, as proved by King and Saia [25]. Such limitations are apparently necessary to achieve subquadratic time complexity for Byzantine failures, c.f., Abraham et al. [1]. *Asynchrony* also implies large communication – Aspnes [4] proved a lower bound $\Omega(n/\log^2 n)$ on communication complexity per process. The complexity bounds in this setting have been later improved, see e.g., [5, 2].

3 Technical Preliminaries

Quantum model of computation. We provide a short review of those parts of the quantum model of computing that are relevant to our results. The reader can find a comprehensive introduction to quantum computing in e.g. [30]. We use only the pure state of qubits. A pure state of a single qubit is a vector in a 2-dimensional Hilbert space $\mathcal{H}$. A pure quantum state of d qubits, denoted $|x\rangle$, is a vector of 2^d-dimensional Hilbert space $\mathcal{H}^{\otimes d} = \mathcal{H} \otimes \ldots \otimes \mathcal{H}$. In our paper, we use only the standard computational basis of the Hilbert space, which consists of vectors $\{|b_1 \ldots b_d\rangle : b_1 \ldots, b_d \in \{0,1\}^d\}$, to describe the system. Therefore, any state $|x\rangle$ can be expressed as $|x\rangle = \sum_{i=0}^{2^d-1} \alpha_i |i\rangle$, with the condition that $\sum_i |\alpha_i|^2 = 1$, since quantum states can be only normalized vectors.

Transitions, or equivalently – changes of states of a quantum system, are given by unitary transformations on the Hilbert space of d qubits. These unitary transformations are called *quantum gates*. These operations are exhaustive in the sense that any quantum computation can be expressed as a unitary operator on some Hilbert space. There are small-size sets of quantum gates working on two-dimensional space that are universal – any unitary transformation on a 2^d-dimensional quantum space can be approximated by a finite collection of these universal gates. In our applications, any quantum algorithm computation run by a process requires a polynomial (in n) number of universal gates.

Finally, an important part of quantum computation is also a quantum measurement. Measurements are performed with respect to a basis of the Hilbert space – in our case, this is always the computational basis. A complete measurement in the computational basis executed on a state $|x\rangle = \sum_{i=0}^{2^d-1} \alpha_i |i\rangle$ leaves the state in one of the basis vectors, $|i\rangle$, for $i \in \{0,1\}^d$, with probability α_i^2. The outcome of the measurement is a classic register of d bits, informing to which vector the state has been transformed. It is also possible to measure only some qubits of the system, which is called a partial measurement. If A describes the subset of qubits that we want to measure and B is the remaining part of the system, then the partial

measurement is defined by the set of projectors $\{\Pi_i = |i\rangle_A \langle i|_A \otimes I_B \mid \text{for } i \in \{0,1\}^d\}$.[5] In the former, a subscript refers to the part of the system on which the object exists, I denotes the identity function, while $\langle i|$ is a functional of the dual space to the original Hilbert space (its matrix representation is the conjugate transpose of the matrix representation of $|i\rangle$). If before the measurement the system was in a state $|x\rangle_{AB}$ then, after the measurement, it is in one of the states $\{\Pi_i |x\rangle_{AB} \mid \text{for } i \in \{0,1\}^d\}$, where state $\Pi_i |x\rangle_{AB}$ is achieved with probability $\langle x|_{AB} \Pi_i |x\rangle_{AB}$.[6] We would like to note that, similarly to all other quantum operations, measurements are local in our model. The result of a measurement is visible only to the process that performed this measurement; however, any quantum operation in some process may affect quantum bits stored in some other process.

Graph notations. Let $G = (V, E)$ denote an undirected graph. Let $W \subseteq V$ be a set of nodes of G. We say that an edge (v, w) of G is *internal for* W if v and w are both in W. We say that an edge (v, w) of G *connects the sets* W_1 *and* W_2, or *is between* W_1 *and* W_2, for any disjoint subsets W_1 and W_2 of V, if one of its ends is in W_1 and the other in W_2. The *subgraph of* G *induced by* W, denoted $G|_W$, is the subgraph of G containing the nodes in W and all the edges internal for W in G. A node adjacent to a node v is a *neighbor of* v and the set of all the neighbors of a node v is the *neighborhood of* v. $N^i_G(W)$ denotes the set of all the nodes in V that are of distance at most i from some node in W in graph G. In particular, the (direct) neighborhood of v is denoted $N_G(v) = N^1_G(v)$.

The following combinatorial properties are of utter importance in the analysis of our algorithms. Graph G is said to be ℓ-*expanding*, or to be an ℓ-*expander*, if any two subsets of ℓ nodes each are connected by an edge. Graph G is said to be (ℓ, α, β)-*edge-dense* if, for any set $X \subseteq V$ of *at least* ℓ nodes, there are at least $\alpha|X|$ edges internal for X, and for any set $Y \subseteq V$ of *at most* ℓ nodes, there are at most $\beta|Y|$ edges internal for Y. Graph G is said to be $(\ell, \varepsilon, \delta)$-*compact* if, for any set $B \subseteq V$ of at least ℓ nodes, there is a subset $C \subseteq B$ of at least $\varepsilon\ell$ nodes such that each node's degree in $G|_C$ is at least δ. We call any such set C a *survival set for* B.

4 Consensus Algorithm

The very high-level description of our consensus algorithm CheapQuantumConsensus is as follows. Each process starts by setting its preferred value (the value which it would like to decide for now) to the input bit. Then, the processes repeatedly use the counting procedure FastCounting, specified in Section 6, to compute the number of preferred 0's and 1's stored by processes that have not crashed yet, see line 3. Due to the definition of Fuzzy Counting, the outcomes of FastCounting can be slightly different across processes, but by no more than the number of crashes. Depending on the outcome, each process may change its preferred value to the dominating one (among the received preferred values), decide if the domination is substantial, or run the quantum common coin procedure, if the number of preferred 0's and 1's are very close to each other – see lines 14-17 in the pseudocode of CheapQuantumConsensus in Figure 1; all lines involving communication are underlined. Repeating the entire process (i.e., counting) for a constant number of rounds and then calculating the weak global coin and updating the decision either deterministically or based on the global coin, guarantees to produce one preferred value in all correct processes with probability 1, provided strong enough protocol for computing the global coin.

[5] We follow the standard notation in quantum computing and skip writing normalizing factors.
[6] $\Pi_i |x\rangle_{AB}$ and $\langle x|_{AB} \Pi_i |x\rangle_{AB}$ are simply linear operations on matrices and vectors.

In general, this type of framework is well-known for solving consensus and was proposed first by Bar-Joseph and Ben-Or [8] in the context of classical randomized computation against an adaptive adversary. However, for classical randomized computation the limitation of this approach is in the fact that any weak global coin obtained so far can tolerate at most $O(\sqrt{n})$ crashes to guarantee constant fairness of the coin. In contrast, by employing quantum communication we propose a new quantum protocol that computes a weak global coin with constant fairness even when a linear number of processes crash.

Besides this improvement, we also propose two new techniques for sparse classical and quantum communication, employed in lines 3 and 18 of the pseudocode in Figure 1: fast and communication-efficient counting and, as mentioned before, fast and quantum-communication-efficient weak global coin, respectively. Both these techniques use parameters x, d, α, which, roughly speaking, correspond to the density of random communication graphs used in these algorithms. The detailed performance formulas of these algorithms, with respect to those parameters, are stated in Theorems 12 and 14.

In the heart of these two techniques lies a crucial observation: consensus (as well as common coin and counting) could be achieved quickly even if many processes do not directly exchange messages, but use some carefully selected sparse set of communication links instead. This way, instead of creating qubits for each pair of processes, we could do it only per some pairs corresponding to some communication links to be used. This set of links, modeled as an evolving communication graph, needs to be maintained adaptively and locally by processes throughout the execution – otherwise, an adaptive adversary would learn it and design crashes to separate processes and prevent consensus.

Algorithm's description. Each process p stores its current choice in b_p (which is initialized to p's input). The value b_p at the end of the algorithm indicates p's decision. Now, processes use $O(1)$ (in expectation) *phases* to update their values b_p such that eventually every process keeps the same decision. To do so, in a round r every process p calculates the number of processes whose current choice is 1 and the number of processes whose current choice is 0, denoted O_p^r and Z_p^r respectively. Based on these numbers, process p: either sets b_p to 1, if the number O_p^r is large enough; or it sets b_p to 0 if the number Z_p^r is large; or it replaces b_p with a random bit if the numbers of zeros and ones are close to each other. If for generating the random bit, in line 18, processes use a quantum implementation of a weak global coin (implemented with CheapQuantumCoin algorithm, specified in Section 5), they will all have the same value b_p with constant probability unless more than third of alive processes crash. Assuming the presence of the adaptive adversary, this could not be achieved quickly if using classical communication only. Once it happens with the help of the quantum weak global coin, the conditional statements in lines 14-17, run in the next iteration of the "while" loop, guarantee that once the majority of processes have the same value b_p, the system converges to this value in at most 2 phases. Since the probability of this event is constant (guaranteed by the quantum weak global coin combined), the expected number of phases before the consensus algorithm terminates is constant. That reasoning holds, assuming that at most 1/3 fraction of processes crashed (we will generalize it to any $t \leq n$ at the end of this section).

As mentioned earlier, the major improvement in the above protocol comes from using novel techniques for counting and weak global coin. For the former, we use the FastCounting algorithm (Theorem 14), which, with the choice of parameters given in line 3, works in $O\Big(\big(\frac{1}{\epsilon}\big)^4\Big)$ rounds and uses $O\big(n^{1+3\epsilon}\log^2 n\big)$ (classic) communication bits in total. Similarly, the CheapQuantumCoin algorithm (Theorem 12), executed in line 18, terminates in $O\Big(\big(\frac{1}{\epsilon}\big)^3\Big)$ rounds and uses $O\Big(n^{1+2\epsilon}\log^2 n\Big)$ both quantum and classical bits; we need to divide the communication formulas by n to obtain the complexity amortized per process.

Algorithm 1 CheapQuantumConsensus for process p.

input: $\mathcal{P}$, p, $input_p$

1 $b_p \leftarrow input_p$; $r \leftarrow 1$; `decided` $\leftarrow FALSE$;
2 **while** $TRUE$ **do**
3 participate in FastCounting$(\mathcal{P}, p, b_p)$ (run with parameters $x = n^\epsilon, d = \log n, \alpha = n^\epsilon$) that counts the processes who have $b_p = 1$ and the processes who have $b_p = 0$; let O_p^r, Z_p^r be the numbers of ones and zeros (resp.) returned by FastCounting;
4 $N_p^r \leftarrow Z_p^r + O_p^r$;
5 **if** $(N_p^r < \sqrt{n/\log n})$ **then**
6 1) send b_p to all processes; 2) receive all messages sent to p in round $r+1$;
7 3) implement a deterministic protocol for $\sqrt{n/\log n}$ rounds;
8 **end**
9 **if** *decided* $= TRUE$ **then**
10 `diff` $\leftarrow N_p^{r-3}$ - N_p^r;
11 **if** *(diff* $\leq N_p^{r-2}/10$*)* **then** STOP;
12 **else** `decided` $\leftarrow FALSE$;
13 **end**
14 **if** $O_p^r > (7N_p^r - 1)/10$ **then** $b_p \leftarrow 1$, `decided` $\leftarrow TRUE$;
15 **else if** $O_p^r > (6N_p^r - 1)/10$ **then** $b_p \leftarrow 1$;
16 **else if** $O_p^r < (4N_p^r - 1)/10$ **then** $b_p \leftarrow 0$, `decided` $\leftarrow TRUE$;
17 **else if** $O_p^r < (5N_p^r - 1)/10$ **then** $b_p \leftarrow 0$;
18 **else** set b_p to the output of CheapQuantumCoin$(\mathcal{P}, p)$ executed with parameters $d = \log n$, $\alpha = n^\epsilon$;
19 $r \leftarrow r + 1$;
20 **end**
21 **return** b_p ; `/* p outputs final decision */`

Algorithm's analysis. To analyze the CheapQuantumConsensus algorithm we first recall a combinations of lemmas from [8].

▶ **Lemma 5** (Lemmas 4.1, 4.2, 4.3 in [8]). *If all processes have the same value at the beginning of an iteration of the main while loop, then the algorithm returns the decision after at most two iterations.*

▶ **Theorem 6.** *For any $\epsilon > 0$, the CheapQuantumConsensus algorithm solves Consensus against $n/3$-adversary in $O\left(\left(\frac{1}{\epsilon}\right)^4\right)$ rounds in expectation while using $O(n^{3\epsilon})$ qubits and communication bits per process (amortized), whp.*

Proof. First, we argue for correctness. Compared to the protocol of Bar-Joseph and Ben-Or [8], which works for an arbitrary number of failures $t \leq n$, we changed the method of deriving random coin, c.f., line 18. Observe that even if CheapQuantumCoin fails to meet conditions of t-resilient coin flip, it always outputs some bit in every non-faulty process. Thus, regardless of the number of crashes the output could be an output of local coin flips with a non-zero probability. Since Bar-Josephs's and Ben-Or's algorithm works with probability 1 (see Theorem 3 in [8]), thus CheapQuantumConsensus also achieves correctness with probability 1.

Next, we estimate the expected number of phases (i.e. the number of iterations of the main while loop). We consider only *good* phases, i.e. phases in which the adversary crashed at most $\frac{1}{10}$ fraction of processes that were correct at the beginning of this iteration. Note, that there can be at most $\frac{1}{3}/\frac{1}{10} < 4$ "bad" phases. Let x be the number of non-faulty processes at the beginning of some good phase. We consider the following cases:
Case *a)* There exists a process that in this iteration executes line 15 or line 14. In this case, all other processes have to execute line 14, 15 or line 18, since the number of ones counted in different processes may differ by $\frac{x}{10}$ at most. All processes that in this iteration execute CheapQuantumCoin will set b_p to 1 with probability $\frac{1}{4}$ at least. What follows, in the next iteration all processes will start with b_p set to 1, and by Lemma 5 the algorithms will decide within two next phases.
Case *b)* There exists a process that in this phase executes line 16 or line 17. Similarly to the previous case, we observe that all other processes have to execute line 17 or line 18, since, again, the number of ones counted in different processes may differ by $\frac{x}{10}$ at most. Therefore the same arguments apply but know the final decision will be 0.
Case *c)* None of processes executes one of lines 14, 15, 16, or 17. Thus, all processes participated in CheapQuantumCoin in line 18. By Theorem 12, with probability at least $\frac{1}{4}$, all processes will set value b_p to the same value. Thus, again by applying Lemma 5, we get that the algorithms will decide within the two next phases.

We showed that if a good phase happens, then the algorithm terminates within 2 next iterations with probability at least $\frac{1}{4}$. Since there can be at most 4 bad iterations, thus we can calculate the expected number of iterations as follows: $\mathbb{E}(\#\text{iterations}) = \sum_{i=4}^{\infty} i \left(\frac{1}{4}\right)^i = O(1)$.

Executing a single phase takes $O\left(\left(\frac{1}{\epsilon}\right)^4\right)$ rounds, which is the round complexity of the FastCounting algorithm and an upper bound of the time complexity of the CheapQuantumCoin algorithm, therefore the algorithm terminates in $O\left(\left(\frac{1}{\epsilon}\right)^4\right)$ rounds in expectation. Similarly, by taking the upper bounds on the communication complexity of the algorithms FastCounting and CheapQuantumCoin we get that the expected number of amortized communication bits used by the algorithm is $O(n^{3\epsilon})$. ◀

Handling arbitrary number of crashes. Consider $O(\log n)$ repetitions of the main loop (phases) of the CheapQuantumConsensus algorithm. If during these phases, the processes with value $b_p = 1$ become a large majority (at least $\frac{6}{10}$ fraction of alive processes), then, as discussed before, every process will decide within the next two rounds. The same holds if processes with value $b_p = 0$ start to overpopulate by a ratio of $\frac{6}{10}$ all non-faulty processes. On the other hand, if the cardinalities of the two groups with different values b_p are close to each other, then the processes execute the CheapQuantumCoin algorithm. It outputs a random bit (the same in every participating process), under the assumption that at least a $\frac{2}{3}$ fraction of processes that started this phase as non-faulty have not crashed during this phase. However, in these $O(\log n)$ phases there must be at least one phase in which the property of a $\frac{2}{3}$ fraction of processes survive holds. In what follows, we argue that if the adversary can crash arbitrarily many processes, but smaller than n, then the expected number of phases should still be $O(\log n)$. Now, to obtain the algorithm stated in Theorem 4, we make two more adjustments to the original CheapQuantumConsensus algorithm. In lines 3 and 18, processes execute the algorithms FastCounting and CheapQuantumCoin, respectively, with parameters x, d, α set as follows: $x = 2, d = \log n, \alpha = \log n$. This corresponds to the use of a sparse graph for communication (of degree roughly $O(\log n)$). In consequence, the time complexity of the FastCounting algorithm increases to $O(\log^4 n)$, but the communication complexity decreases to $O(\log^8 n)$ amortized per process. The details of the implementation

of the FastCounting algorithm are presented in Section 6, and performance follows from putting the aforementioned parameters to the formulas in Theorem 14. Similarly, the time complexity of the CheapQuantumCoin algorithm increases to $O(\log^3 n)$, but the communication complexity (both quantum and classical) decreases to $O(\log^7 n)$ amortized per process. Formally:

▶ **Theorem 7.** *The modified version of the CheapQuantumConsensus algorithm solves Consensus against any adversary in $O(\log^5 n)$ rounds in expectation while using $O(\log^8 n)$ qubits and communication bits per process (amortized), whp.*

Proof. For correctness, we argue exactly the same as in the proof of the previous Theorem. We also define good and bad phases, with only this difference that now the number of bad phases is at most $O(\log n)$ since the adversary has the full power of crashing an arbitrary number of processes. This being said we get that, by the very same reasoning, the expected number of phases is

$$\mathbb{E}(ITE) = \sum_{i=\log n}^{\infty} i \left(\frac{1}{4}\right)^i = \Theta(\log n) \ .$$

By examining the time and bits complexity of the algorithms FastCounting and CheapQuantumCoin (c.f. Theorems 14, 12) with parameters $x = 2, d = \log n, \alpha = \log n$, we get a single phase lasts $O(\log^4 n)$ rounds and contributes $O(n \log^7 n)$ bits to the total communication complexity. The latter, after dividing by n, gives the sought complexity amortized per process. Thus, the theorem follows. ◀

5 Qubit-and-Communication Efficient Quantum Common Coin

In this section, we design a new t-resilient weak global coin, for $t < n$, with the help of quantum communication and computation.

On a high level, our protocol CheapQuantumCoin chooses a leader process uniformly at random and all other processes agree on the random bit proposed by the leader. Quantum phenomena are used to hide the random choices of the leader and its output from the adaptive adversary when processes communicate with each other. The idea was first proposed in [9], yet there are key differences between that work and our algorithm. Instead of all-to-all communication, which required large number of qubits, we use a sequence of random graphs of node degrees $d, d\alpha^1, \ldots, d\alpha^k$, respectively, where $d, \alpha \in \Omega(\log n)$ and $k = \lceil \log n / \log \alpha \rceil$ are some integer parameters. The vertices of these graphs correspond to processes and edges correspond to communication links – each process communicates with neighbors in one of the graphs at a time. If the graph is chosen properly (i.e., so that there is no situation in which too many processes use denser graphs), it reduces the communication complexity but simultaneously imposes a new challenge. Mainly, the communication procedure has to now assure the delivery of quantum bits between every two non-faulty processes regardless of the pattern of crashes. For instance, if only one random graph of degree d was used then the adversary could easily isolate any vertex using only $O(d)$ crashes (i.e., by crashing all its neighbors). Hence, strictly speaking, assuring such delivery is not possible while using a sparse communication graph as relays, but we show that a certain majority could still come up with a common coin value based only on their exchanges with neighbors in the communication graphs; they could later propagate their common value to other processes by adaptively controlling their (increasing) set of neighbors, taken from subsequent communication graphs of increasing density. A thorough analysis shows that in this way it is possible to achieve the same quantum properties that are guaranteed by Ben-Or's and Hassidim's global coin [9], and at the same time reducing the quantum communication by a polynomial factor.

Algorithm's description. We now describe the CheapQuantumCoin algorithm. Its pseudocode is presented in Figure 2. It takes as input: a process name p and two integers, d and α. The two latter parameters are used to determine communication patterns between non-faulty processes, and their choice determines the complexity of the algorithm.

As mentioned before, processes use the quantum equivalent of a procedure in which processes draw random names from the range $[1, \ldots, n^3]$ and decide on a random bit proposed by the process with the largest name.[7] We view the quantum part of the algorithm as a quantum circuit on the joint space of all qubits ever used by different processes. Due to the distributed nature of the system, not all quantum operations are allowed by the quantum circuit. That is, (i) any process can perform arbitrary unitary gates on its part of the system, (ii) unitary gates on qubits of different processes might be performed but must be preceded by quantum communication that sends qubits involved in the operation to a single process. The communication can be either direct or via relays. We use the following unitary gates to simulate the classical algorithm of the leader election in the quantum distributed setting: `Hadamard` gate, `CNOT` and `Pairwise_CNOT` gates, and `F_CNOT` and `Controlled_Swap` gates. (Although they are all standard quantum gates, a thorough description of these gates can be found in the full version.)

Let us now explain how all the gates listed above work together. As mentioned in the description of the Hadamard gate, after line 2 ends, the main registers of the system are in the state of being a uniform superposition of all vectors of the computational basis. Starting from this point, the composition of all gates applied to different registers along the execution can be viewed as a single unitary gate on the entire system, consisting of the qubits that any process ever created. Note that the unitary transformation might be different depending on the failure in communication, i.e., a failure in the delivery of some block of qubits between two processes may result in abandoning gates involving these qubits, but for a moment let us assume that the links are reliable. Since the unitary transformation is linear, it is enough to consider how it affects the vectors of the computational basis. However, all the gates described above behave on the computational basis as their classical equivalents. More precisely, let $|x\rangle$ be a vector from the computational basis spanning the whole circuit. Let p be the process whose main register has the largest[8] value in the state $|x\rangle$. From the point of view of this register, the following happens in the algorithm. In each round, p creates an entangled state on $6 \log n + 2$ qubits (see point (2)) that has the same qubits on its new block of $3 \log n + 1$ qubits as it has on the main register. Then, it propagates the new block to its neighbors (lines 9- 12). The neighbors compare the content of received qubits and exchange them with their main register if their content is smaller (gates `F_CNOT` and `Controlled_Swap` in lines 16- 17). This operation is then repeated $(k+2)^2(\gamma_\alpha + 1)$ times on the set of links defined by some random evolving graphs, see the later paragraph about adaptive communication pattern. In the end, the processes who, either directly or via relays, received the content of the largest main register, have the same value in their main register. Therefore, the result of the measurement in line 27 must be the same in all these processes.

Assume now that we are able to achieve bidirectional quantum communication between any pair of processes of an τ fraction of the entire system, regardless of the (dynamic) actions of the adversary, for some constant $0 \leq \tau \leq 1$. From the above, the algorithm transforms any vector whose largest main register is one of the registers of the τ fraction of the processes to a

[7] Note that the latter procedure cannot be used against an adaptive adversary, as it could crash such a leader.

[8] The probability of a tie is polynomially small.

vector such that the processes from the τ fraction have the same values in the main registers. The initial state of the system is a uniform superposition which is a linear combination of all vectors from the computational basis. From this perspective, the normalized sum of coefficients of these vectors of the superposition that have the property that the largest register is one of the registers of the τ fraction is $\tau - o(1)$.[9] Thus, we get that the probability of measuring the same values in the processes of the τ fraction is at least $\tau - o(1)$ and we can claim the following lemma.

▶ **Lemma 8.** *Let A be a set of correct processes such that any pair of them was connected by quantum communication either directly or by relays. Then the probability that all processes from A output the same bit from the algorithm* CheapQuantumConsensus *is at least* $\frac{|A|}{n} - o(1)$.

Adaptive communication pattern. As explained, we not only need that communication should be efficient in terms of the number of qubits and classical bits, but also it should be such that any two correct processes of a large fraction of the entire system are connected by a short path of the correct process so that quantum registers can be relayed. Let d, α be two integer parameters. We define $k = \left\lceil \frac{\log(n/d)}{\log \alpha} \right\rceil$, $\gamma_\alpha = \frac{\log n}{\log \alpha}$, and $\delta_\alpha = \frac{2}{3}\alpha$. Initially, each process p draws independently $k+1$ sets $\mathcal{N}_p(d), \mathcal{N}_p(d\alpha^1), \ldots, \mathcal{N}_p(d\alpha^k)$, where a set $\mathcal{N}_p(d\alpha^i)$, for $0 \leq i \leq k$, includes each process from $\mathcal{P}$ with probability $\frac{d\alpha^i}{n}$.

Communication is structured into $(k+2)^2$ *epochs*, see line 4. Each epoch consists of $2(\gamma_\alpha + 1)$ communication rounds, also called *testing* rounds. They are scheduled in $\gamma_\alpha + 1$ iterations within the loop "for" in line 7, each iteration containing two communication rounds (underlined in the pseudocode): sending/receiving inquiries in line 9 and sending/receiving responses in line 12. In the testing rounds of the first epoch, a process p sends inquiries to processes in set $\mathcal{N}_p(d)$. The inquired processes respond by sending in the next round (line 12) their current classical state and specially prepared, in line 6, quantum register. However, if in a result of crashes p starts receiving less than δ_α responses per round, it switches its communication neighborhood from $\mathcal{N}_p(d)$ to the next, larger set, $\mathcal{N}_p(d \cdot \alpha)$. A similar adaptation to a crash pattern is continued in the remaining epochs.

Process p stores the cardinally of the set being inquired in an epoch in the variable $\texttt{degree}_p$ (initialized to d in line 2). For the purpose of testing rounds, p copies the value $\texttt{degree}_p$ to a variable $\texttt{adaptive\_degree}_p$ (line 6). In every testing round, p adapts its variable $\texttt{adaptive\_degree}_p$ to the largest value $x \leq \texttt{adaptive\_degree}_p$ such that it received at least δ_a responses from processes that have their variable `adaptive_degree` at least x (loop "while" in line 19). If p had to decrease the value $\texttt{adaptive\_degree}_p$ in testing rounds of an epoch, it then *increases* the main variable $\texttt{degree}_p$ by the factor α before the next epoch, see line 24. The latter operation formally encodes the intuition that the process p expected to have δ_α non-faulty neighbors with their values of `degree` at least as big as its own, but due to crashes it did not happen; Therefore, p increases the number of inquired processes, by adopting the next, larger neighborhood set $\mathcal{N}_p(\cdot)$, randomly selected, in order to increase the chance of communication with the majority of non-faulty processes in the next epoch. On the other hand, the adaptive procedure of reducing `adaptive_degree` in testing rounds of a single epoch helps neighbors of p to estimate correctly the size of the

[9] The $o(1)$ part contributes to the normalized sum of these vectors that correspond to having more than one largest register.

Algorithm 2 CheapQuantumCoin for process p.

input: p, two parameters: d, α

For $0 \leq i \leq \left\lceil \frac{\log(n/d)}{\log \alpha} \right\rceil$: $\mathcal{N}_p(d\alpha^i) \leftarrow$ a set of processes such that each process is chosen independently with probability $\frac{d\alpha^i}{n}$;

$\texttt{degree}_p \leftarrow d$, $\gamma_\alpha \leftarrow \frac{\log n}{\log \alpha}$, $\delta_\alpha \leftarrow \frac{2}{3}\log n$;

$|Leader\rangle_p |Coin\rangle_p \leftarrow H^{\otimes 3\log n+1} |00\ldots0\rangle$ (a gate on $3\log n+1$ qubits) ;

for $i = 1$ *to* $(k+2)^2$; `/* iter. of epochs */`

do

  $\texttt{adaptive\_degree} \leftarrow \texttt{degree}_p$;

  for $j = 1$ *to* $\gamma_\alpha + 1$; `/* iter. of testing rounds */`

  do

    send to each process in $\mathcal{N}_p(\texttt{degree}_p)$: an inquire bit 1 ;

    $\mathcal{I} \leftarrow$ the set of processes who sent an inquire bit to p ;

    $\forall_{q\in\mathcal{I}} : |B\rangle_q \leftarrow \texttt{Pairwise\_CNOT}\left(|LeaderCoin\rangle_p, |0\ldots0\rangle\right)$;

    send to each process $q \in \mathcal{I}$: a quantum message containing first $3\log n+1$ bits of $|B_q\rangle$, and a classical message containing $\texttt{adaptive\_degree}_p$;

    $\mathcal{R} \leftarrow$ the set of processes who responded to p's inquires ;

    foreach $q \in \mathcal{R}$ **do**

      $|CLeader\rangle_q |CCoin\rangle_q \leftarrow$ received quantum bits from q, $|S\rangle \leftarrow |0\rangle$;

      $\texttt{F\_CNOT}_{|Leader\rangle_p > |CLeader\rangle_q}\left(|Leader\rangle_p, |CLeader\rangle_q, |S\rangle\right)$;

      $\texttt{Controlled\_Swap}\left(|LeaderCoin\rangle_p, |CLeaderCCoin\rangle_q, |S\rangle\right)$;

    end

    while $|\{q \in \mathcal{R} : \textit{adaptive\_degree}_q \geq \textit{adaptive\_degree}_p\}| < \delta_\alpha$ *and* $\textit{adaptive\_degree}_p \geq d$; `/* adapting #neighbors during testing */`

    do

      $\texttt{adaptive\_degree}_p \leftarrow \frac{1}{\alpha}\texttt{adaptive\_degree}_p$;

    end

  end

  if $\textit{adaptive\_degree}_p < \textit{degree}_p$ **then**

    $\texttt{degree}_p \leftarrow \min\{\texttt{degree}_p \cdot \alpha, d\alpha^k\}$; `/* neighborhood for next epoch grows */`

end

$b_p \leftarrow$ be the last bit of the result of measuring $|Leader\rangle_p |Coin\rangle_p$ in the computational basis;

return b_p ; `/* p outputs random bit */`

neighborhood that process p is using in the current testing round, which might be much smaller than the value $\texttt{degree}_p$ from the beginning of the epoch. This, in turn, calibrate the value of $\texttt{adaptive\_degree}$ of the neighbors of p, and this calibration can propagate to other processes of distance up to γ_α from p in the next iterations of testing rounds.

Analysis. Let us define graphs $\mathcal{G}(d\alpha^i)$, for $0 \leq i \leq k$, as the union of random sets $\cup_{p\in\mathcal{P}}\mathcal{N}_p(d\alpha^i)$. The probability distribution of the graph $\mathcal{G}(d\alpha^i)$ is the same as the random graph $G(n, y)$ for $y = \frac{d\alpha^i}{n}$. Chlebus, Kowalski and Strojnowski [12] showed in their Theorem 2, applied for $k = \frac{64n}{d\alpha^{i-1}}$, that the graph $\mathcal{G}(d\alpha^i)$ has the following properties, whp:

- **(i)** it is $(\frac{n}{d\alpha^{i-1}})$-expanding, which follows from $(\frac{n}{d\alpha^{i-1}}, \frac{2}{3}\frac{n}{d\alpha^{i-2}}, \frac{4}{3}\frac{n}{d\alpha^{i-2}})$-edge-expanding property,
- **(ii)** it is $(\frac{n}{d\alpha^{i-1}}, \frac{1}{3}\alpha, \frac{2}{3}\alpha)$-edge-dense,
- **(iii)** it is $(16\frac{n}{d\alpha^{i-1}}, 3/4, \frac{2}{3}\alpha)$-compact,
- **(iv)** the degree of each node is at most $\frac{21}{20}d\alpha^i$.

Since the variable $\texttt{degree}_p$ takes values in the set $\{d, d\alpha^1, \ldots, d\alpha^k\}$, the pigeonhole principle assures that eventually p will participate in an epoch in which $\texttt{degree}_p$ has not been increased. In the most severe scenario, p will use the set $\mathcal{N}_p(d\alpha^k)$, which consists of all other processes – because it contains each process, as a neighbor of p, with probability 1. The $(\gamma_\alpha, \delta_\alpha)$-dense-neighborhood property of random graphs composed from neighborhoods $\mathcal{N}(\texttt{degree}_p)$ implies that p will then contact a majority of other non-faulty processes via at most $\gamma_\alpha + 1$ intermediate processes (during testing rounds). Formally, the following holds:

▶ **Lemma 9.** *If a process p does not change its variable $\texttt{degree}_p$ at the end of an epoch i, then at the beginning of epoch i there exists a $(\gamma_\alpha, \delta_\alpha)$-dense-neighborhood of p in the graph $\mathcal{G}(\texttt{degree}_p)$ consisting of non-faulty processes with variable $\texttt{degree}$ being at least $\texttt{degree}_p$ in the epoch i, whp.*

On the other hand, $(16n/d\alpha^{i-1}, 3/4, 2\alpha/3)$-compactness of the (random) graph composed of processes that have the variable $\texttt{degree}$ at least $d\alpha^i$, guarantees that the total number of processes that use sets $\mathcal{N}(d\alpha^i)$ during the epoch i is at most $\frac{n}{\alpha^{i-2}}$, which amortizes communication complexity.

▶ **Lemma 10.** *For any integer i, such that $0 \leq i \leq k$, at the beginning of each epoch there is at most $\frac{16n}{d\alpha^{i-2}}$ processes with the variable $\texttt{degree}$ greater than $d\alpha^i$, whp.*

In the above proof of Lemma 10, we use the fact that a suitable set C of processes that have been correct throughout the whole epoch exists. We may choose this set and argue about it after the epoch, as the communication pattern in the algorithm is deterministic. Hence, in any round of the epoch, processes in C have at least as many non-faulty neighbors in the communication graph as they have neighbors from set C. We use this number of neighbors in C as a *lower bound* to argue about the behavior of variables $\texttt{degree}$; therefore, our arguments do not depend on the behavior of processes outside of C and whether/when some of them fail during the epoch.

▶ **Lemma 11.** *Any two non-faulty processes p and q were connected by a quantum path of communication during the run of the algorithm.*

The above lemmas yield the following result.

▶ **Theorem 12.** *For two integer parameters $d, \alpha \in \Omega(\log n)$, the algorithm* QuantumCoinFlip *generates a weak global coin, provided that at most $\frac{1}{3}$-fraction of initially non-faulty processes have crashed. It terminates in $O\left(\left(\frac{\log n}{\log \alpha}\right)^3\right)$ rounds and with high probability uses $O\left(\left(\frac{\log n}{\log \alpha}\right)^4 d\alpha^2 \log n\right)$ both quantum and classical communication bits (amortized) per process.*

Proof of Theorem 12. Let $\mathcal{H} \subseteq \mathcal{P}$ be the set of initially non-faulty processes. Assume that at least $\frac{2}{3}|\mathcal{H}|$ of them remains non-faulty during the execution of the algorithm. By Lemma 11, any pair of processes from $\mathcal{H}$ is connected by quantum communication, therefore applying Lemma 8 to this set, give that there is at least $\frac{2}{3} - o(1)$ (which is greater than $\frac{1}{2}$ for sufficiently large n) probability that all non-faulty processes return the same output bit. Since 0 and 1 are equally likely, thus the probabilistic guarantee on the weak global coin follows.

The number of rounds is deterministic and upper bounded by $O(k^2 \cdot \gamma_\alpha) = O\left(\left(\frac{\log(n/d)}{\log \alpha}\right)^2 \frac{\log n}{\log \alpha}\right) = O\left(\left(\frac{\log n}{\log \alpha}\right)^3\right)$. To bound the communication complexity, assume that each graph $\mathcal{G}(d\alpha^i)$, for $0 \leq i \leq k$, satisfies the desired expanding properties listed in the description of the algorithm. This, by the union bound argument, holds whp. By Lemma 10

at the beginning of each epoch there are at most $\frac{16n}{d\alpha^{i-2}}$ processes that inquire more than $d\alpha^i$ other processes in testing rounds of this epoch, for each $0 \leq i \leq k$. Since each message uses at most $O(\log n)$ bits and qubits, thus a single testing round in an epoch adds at most $\sum_{i=0}^{k} \frac{16n}{d\alpha^{i-2}} \cdot d\alpha^i \leq 16kd\alpha^2 \cdot n \log n$ qubits and bits to communication complexity. Since there is exactly $O\left(\left(\frac{\log n}{\log \alpha}\right)^3\right)$ testing rounds, thus the $O\left(\left(\frac{\log n}{\log \alpha}\right)^4 d\alpha^2 \cdot n \log n\right)$ upper bound on the total communication complexity of the algorithm follows. Dividing the latter by n we receive amortized formula per process. ◀

6 Constant-Time Communication-Efficient Fuzzy Counting

Although generating a weak global coin in a constant number of rounds against an adaptive adversary requires quantum communication (due to the lower bound by Ben-Or and Bar-Joseph [7]), the CheapQuantumCoin algorithm, even without quantum communication, achieves few other goals. As discussed in the previous section, its random communication pattern guarantees, whp, that any additional classical message, also called a rumor, of a non-faulty process can be conveyed to any other non-faulty process if added to every classical message sent/received in line 12. Even more, assume that there is a set of x messages/rumors $M = \{m_1, \ldots, m_x\}$, distributed as inputs among some subset of processes (one message from M per process). If processes always convey all the known rumors from set M when using classical communication (avoiding repetition), then they solve a Gossip problem, whp, i.e., every rumor m_i given to a non-faulty process, for $1 \leq i \leq x$, is known at the end of the protocol to every other non-faulty process. Observe that processes resign from the quantum content of communication for this purpose, and instead of $\log n$ bits (or qubits) per message, they use $|M|$ bits, where $|M|$ denotes the number of bits needed to encode all rumors from M. Finally, if processes work in a model where the names of other processes are commonly known, they can withdraw from using random communication. Instead, they can use a deterministic family of graphs $\mathcal{G}(d\alpha^i)$, for the same choice of parameters d and α. The proof of existence of such graphs, using the probabilistic argument, was described in [12] (Theorem 2). In such case, the set $\mathcal{N}_p(d\alpha^i)$ is the neighborhood of process p in the deterministic graph $\mathcal{G}(d\alpha^i)$. (Processes compute the same copies of graphs $\mathcal{G}$ locally in the beginning of the algorithm.) The above augmentation of the algorithm, together with the proof of Theorem 12, from which we take the upper bound on the number of messages send and the upper bound on the number of rounds, gives:

▶ **Theorem 13.** *For integer parameters $d, \alpha \in \Omega(\log n)$, there is a deterministic algorithm that solves the gossip problem in $O\left(\left(\frac{\log n}{\log \alpha}\right)^3\right)$ rounds using $O\left(\left(\frac{\log n}{\log \alpha}\right)^4 d\alpha^2 \cdot (|M| + \log n)\right)$ communication bits per process (amortized), where $|M|$ is the number of bits needed to encode all rumors in M.*

Generalized Fuzzy Counting. In this part, we refine the state-of-art method of solving the Fuzzy Counting problem (c.f. Definition 3), even deterministically, and propose a new recursive algorithm with any branching factor x, called FastCounting. We prove the following:

▶ **Theorem 14.** *For any $2 \leq x \leq n$ and $\delta, \alpha \in \Omega(n)$, the FastCounting deterministic algorithm solves the Fuzzy Counting problem in $O\left(\frac{\log n}{\log x}\left(\frac{\log n}{\log \alpha}\right)^3\right)$ rounds, using $O\left(\frac{\log n}{\log x}\left(\frac{\log n}{\log \alpha}\right)^4 d\alpha^2 \cdot x \log n\right)$ communication bits (amortized) per process.*

Obviously, the constant-time is achieved in Theorem 14 when $x, \alpha = n^{\epsilon}$, for a constant $\epsilon \in (0,1)$. In this case, the number of rounds is $O\left(\left(\frac{1}{\epsilon}\right)^4\right)$, while the communication complexity is $O(n^{3\epsilon}\log^2 n)$ (amortized) per process. In our approach, we generalize the method of Hajiaghayi et al. [23] to denser communication graphs of certain properties, which allows us to achieve constant running time. The constant running time is the key feature of the algorithm, since it is used in the implementation of (expected) constant-round quantum consensus protocol CheapQuantumConsensus. The main difference between ours and the state-of-art approach is a different Gossip protocol used in the divide-and-conquer method.

The FastCounting algorithm is recurrent. It takes as an input the following values: $\mathcal{P}$ is the set of processes on which the algorithm is executed; p is the name of a process which executes the algorithm; $a_p \in \{0,1\}$ denotes if p is active ($a_p = 1$) or not; and parameters x, d, α, where x is the branching factor and d, α steer the density of the communication graph in the execution. Knowing the set $\mathcal{P}$ of n processes, FastCounting splits the set into x disjoint groups of processes, each of size between $\lfloor \frac{n}{x} \rfloor$ and $\lceil \frac{n}{x} \rceil$. Name these groups $\mathcal{P}_1, \ldots, \mathcal{P}_x$. The groups are known to each participating process. The algorithm then makes x parallel recursive calls on each of these groups. As a result, a process p from a group $\mathcal{P}_i$, for $1 \leq i \leq x$, gets the number of the active processes in its group $\mathcal{P}_i$. In the merging step, all processes execute Gossip algorithm, proposed in Theorem 13, with parameters d, α, where the input rumors are the numbers calculated in the recursive calls. To keep the communication small, when processes learn new rumors they always store at most one rumor corresponding to each of the x groups. This way, the number of bits needed to encode all rumors is $O(x \log n)$. Let $r_1, \ldots, r_\ell$ be the rumors learned by process p from the execution of the Gossip algorithm. The final output of p is the sum $\sum_{i=1}^{\ell} r_i$. The pseudocode of the algorithm can be found in the full version.

7 Future Work

We believe that the outcome of our work effectively establishes the foundation for the efficient integration of classical fault-tolerant distributed and quantum computing. We show that in the basic model of failures, crash failures, a fundamental problem of reaching consensus fast is feasible with only polylog n qubit communication. An immediate open question arises whether such performance improvement is possible in the case of more severe failures, such as omissions, authenticated Byzantine, or even full Byzantine. Another interesting idea is to approach the communication bound from the lower bound side. We believe that investigating even conditional lower bounds on the minimal quantum communication needed to solve various distributed problems, foremost the Consensus problem, could lead to many breakthroughs in both distributed and quantum realms.

References

1 I. Abraham, T. -H. Hubert Chan, D. Dolev, K. Nayak, R. Pass, L. Ren, and E. Shi. Communication complexity of byzantine agreement, revisited. In *Proceedings of the acm Symposium on Principles of Distributed Computing (podc)* and 2019, editors, *Distributed Computing*, pages 317–326, 2018.

2 Dan Alistarh, James Aspnes, Valerie King, and Jared Saia. Communication-efficient randomized consensus. *Distributed Computing*, 31:489–501, 2018.

3 S. Amdur, S. Weber, and V. Hadzilacos. On the message complexity of binary agreement under crash failures. *Distributed Computing*, 5:175–186, 1992.

4 J. Aspnes. Lower bounds for distributed coin-flipping and randomized consensus. *J. ACM*, 45(3):415–450, 1998.

5 H. Attiya and K. Censor. Tight bounds for asynchronous randomized consensus. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 11–13, California, USA, June, 2007. San Diego.

6 H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. Wiley, 2nd edition, 2004.

7 Z. Bar-Joseph and M. Ben-Or. A tight lower bound for randomized synchronous consensus. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 193–199, 1998.

8 Z. Bar-Joseph and M. Ben-Or. A tight lower bound for randomized synchronous consensus, 2002. doi:10.1145/277697.277733.

9 M. Ben-Or and A. Hassidim. Fast quantum byzantine agreement. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, 2005.

10 A. Broadbent and A. Tapp. Can quantum mechanics help distributed computing? *SIGACT News*, 39(3):67–76, 2008.

11 B. S. Chlebus and D. R. Kowalski. Robust gossiping with an application to consensus. *Journal of Computer and System Sciences*, 72(2006):1262, 2009.

12 B. S. Chlebus, D. R. Kowalski, and M. Strojnowski. Fast scalable deterministic consensus for crash failures. In *Proceedings of the 28th ACM symposium on Principles of distributed computing*, pages 111–120, 2009.

13 B. S. Chlebus, D. R. Kowalski, and M. Strojnowski. Scalable quantum consensus for crash failures. In *Proceedings of the 24th International Symposium on Distributed Computing (DISC)*, 2010.

14 V. Cholvi. Quantum byzantine agreement for any number of dishonest parties. *Quantum Inf. Process.*, 21:1–11, 2022.

15 B. Chor, M. Merritt, and D. B. Shmoys. Simple constant-time consensus protocols in realistic failure models. *J. ACM*, 36(3):591–614, 1989.

16 D. Dolev and R. Reischuk. Bounds on information exchange for byzantine agreement. *Journal of the ACM*, 32:191–204, 1985.

17 D. Dong and I. R. Petersen. Quantum estimation, control and learning: Opportunities and challenges. *Annual Reviews in Control*, 54:243–251, 2022.

18 C. Dwork, J. Halpern, and O. Waarts. Performing work efficiently in the presence of faults. *SIAM Journal on Computing*, 27:1457–1491, 1998.

19 M. Fisher and N. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14:183–186, 1982.

20 Z. Galil, A. Mayer, and M. Yung. Resolving message complexity of byzantine agreement and beyond. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 724–733, 1995.

21 S. Gilbert and D. R. Kowalski. Distributed agreement with optimal communication complexity. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 965–977, 2010.

22 V. Hadzilacos and J. Y. Halpern. Message-optimal protocols for byzantine agreement. *Mathematical Systems Theory*, 26:41–102, 1993.

23 M. Hajiaghayi, D. R. Kowalski, and J. Olkowski. Improved communication complexity of fault-tolerant consensus. In *Proceedings of the 54th Annual ACM Symposium on Theory of Computing (STOC)*, 2022.

24 L. K. Helm, quantum distributed consensus. In *Proceedings of the 27th ACM symposium on Principles of distributed computing (PODC)*, 2008.

25 Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: Scalable byzantine agreement with an adaptive adversaryo. *, J. ACM*, 18(24):18–1, 2011.

26 Q. Luo, K. Feng, and M. Zheng. Quantum multi-valued byzantine agreement based on d-dimensional entangled states. *International Journal of Theoretical Physics*, 58:4025–4032, 2019.

27 M. Marcozzi and L. Mostarda. Quantum consensus: an overview. *arXiv preprint*, 2021. `arXiv:2101.04192`.

28 L. Mazzarella, A. Sarlette, and F. Ticozzi. Consensus for quantum networks: Symmetry from gossip interactions. *IEEE Transactions on Automatic Control*, 60(1):158–172, 2014.

29 C. Mochon. Quantum weak coin flipping with arbitrarily small bias. *arXiv preprint*, 2007. `arXiv:0711.4114`.

30 M. A. Nielsen and I. Chuang. Quantum computation and quantum information. *Phys. Today*, 54(2):60, 2001.

31 M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27:228–234, 1980.

32 S. Tani, H. Kobayashi, and K. Matsumoto. Exact quantum algorithms for the leader election problem. *ACM Transactions on Computation Theory*, 4:1–24, 2012.

33 J. van Apeldoorn and T. de Vos. A framework for distributed quantum queries in the congest model. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, 2022.

34 W. Wang, Y. Yu, and L. Du. Quantum blockchain based on asymmetric quantum encryption and a stake vote consensus algorithm. *Scientific Reports*, 12(1):1–12, 2022.

35 X. Wu and P. Yao. Quantum complexity of weighted diameter and radius in congest networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, 2022.

36 L. Zhang and S. Fulton. Discussion of quantum consensus algorithms. *arXiv preprint*, 2022. `arXiv:2206.04710`.

Oracle-Augmented Prophet Inequalities

Sariel Har-Peled
Department of Computer Science, University of Illinois, Urbana, IL, USA

Elfarouk Harb
Department of Computer Science, University of Illinois, Urbana, IL, USA

Vasilis Livanos
Department of Industrial Engineering, University of Chile, Santiago, Chile

Abstract

In the classical prophet inequality setting, a gambler is given a sequence of n random variables $X_1, \ldots, X_n$, taken from known distributions, observes their values in adversarial order and selects one of them, immediately after it is being observed, aiming to select a value that is as high as possible. The classical *prophet inequality* shows a strategy that guarantees a value at least half of the value of an omniscience prophet that always picks the maximum, and this ratio is optimal.

Here, we generalize the prophet inequality, allowing the gambler some additional information about the future that is otherwise privy only to the prophet. Specifically, at any point in the process, the gambler is allowed to query an oracle $\mathcal{O}$. The oracle responds with a single bit answer: YES if the current realization is greater than the remaining realizations, and NO otherwise. We show that the oracle model with m oracle calls is equivalent to the Top-1-of-$(m+1)$ model when the objective is maximizing the probability of selecting the maximum. This equivalence fails to hold when the objective is maximizing the competitive ratio, but we still show that any algorithm for the oracle model implies an equivalent competitive ratio for the Top-1-of-$(m+1)$ model.

We resolve the oracle model for any m, giving tight lower and upper bound on the best possible competitive ratio compared to an almighty adversary. As a consequence, we provide new results as well as improvements on known results for the Top-1-of-m model.

2012 ACM Subject Classification Theory of computation → Algorithmic mechanism design

Keywords and phrases prophet inequalities, predictions, top-1-of-k model

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.81

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2404.11853` [13]

Funding *Sariel Har-Peled*: Work on this paper was partially supported by NSF AF award CCF-2317241.
Elfarouk Harb: Work on this paper was partially supported by NSF award CCF-1910149.
Vasilis Livanos: Work on this paper was partially supported by NSF award CCF-1750436.

1 Introduction

The field of optimal stopping theory concerns optimization settings where one makes decisions in a sequential manner, given imperfect information about the future, in a bid to maximize a reward or minimize a cost. A canonical setting in this area is the *prophet inequality* [18, 19]. In these settings, a gambler is presented with rewards $X_1, \ldots, X_n$, one after the other, drawn independently from known distributions. Upon seeing a reward X_i, the gambler must immediately make an irrevocable decision to either accept X_i, in which case the process ends, or to reject X_i and continue, losing the option to select X_i in the future. The goal of the gambler is to maximize the selected reward comparing against a *prophet* who knows all realizations in advance and selects the maximum realized reward. Throughout, we assume, without loss of generality, that $X_1, \ldots, X_n$ are continuous random variables.

The performance of the gambler can be measured in terms of several objectives. A common metric used in the literature is the *competitive ratio*, which is also known as the *Ratio of Expectations (RoE)* (see Definition 1.1). Another common distinction is between the case in which the given distributions are different and the case in which they are identical. For the former, Krengel et al. [18, 19] showed an optimal strategy that is $1/2$-competitive. In this setting, the optimal competitive ratio can be achieved by simple, single-threshold algorithms [21, 17]. For IID and non-IID random variables, Hill and Kertz [15] initially gave a $(1 - 1/e)$-competitive algorithm. This was improved to ≈ 0.738 [1] and later ≈ 0.745 [7], which is tight, due to a matching upper bound [15, 16].

Another relevant metric, introduced by Gilbert and Mosteller [12] for IID random variables, is that of maximizing the *Probability of selecting the Maximum realization* ($\mathbb{P}_{\max}$) - see Definition 1.2. For this objective and IID random variables, Gilbert and Mosteller [12] gave an algorithm that achieves a probability of ≈ 0.58, which is the best possible. Later, Esfandiari, Hajiaghayi, Lucier and Mitzenmacher [9] studied the same objective for general random variables, obtaining a tight probability equal to $1/e$ when the random variables arrive in adversarial order and 0.517 when the random variables arrive in random order. The latter case was recently improved to the tight ≈ 0.58 by Nuti [20], showing that the IID setting is not easier than the non-IID setting with random order. In this paper, we introduce a new model as a means to study variations of both the IID and the general settings, for both the RoE and $\mathbb{P}_{\max}$ objectives.

A setting that is related to ours is the Top-1-of-m model, formally introduced by Assaf and Samuel-Cahn [3] for IID random variables, although it had been studied initially by Gilbert and Mosteller [12]. In this setting, the algorithm is allowed to select $m \geq 1$ values, but the value it gets judged by is the maximum selected value. Gilbert and Mosteller [12] gave numerical approximations of the $\mathbb{P}_{\max}$ objective for $2 \leq m \leq 10$, using a simple, single-threshold algorithm. Later, Assaf and Samuel-Cahn [3] studied the RoE objective for general distributions and gave an elegant and simple $(1 - 1/m+1)$-competitive algorithm. This was improved [2] by bounding the competitive ratio of the optimal algorithm by a recursive differential equation. They gave numerical approximations for $2 \leq m \leq 5$, but studying the asymptotic nature of the constants for large m turned out to be difficult. Ezra et al. [11] later revisited the problem and gave a new algorithm for large m that is $1 - \mathcal{O}\big(e^{-m/6}\big)$-competitive for the same problem. This improves the error term from [2] from linear in m to exponential in m. Harb [14] recently improved this into a $1 - e^{-mW_0\left(\frac{m\sqrt{m!}}{m}\right)}$-competitive algorithm, where W_0 is the Lambert-W function[1], and improved the lower bound for $m = 2$ separately. However, the asymptotic nature of this function is difficult to analyze.

Model

We introduce a new model that generalizes the standard prophet inequality setting, and analyze it as a means to obtain new results and improvements in the Top-1-of-m model. Our model allows the algorithm some information about the future that is otherwise privy only to the prophet. Specifically, at any point in the process, upon seeing a reward X_i, the algorithm is allowed to query an oracle $\mathcal{O}$. The oracle $\mathcal{O}$ responds with a single bit answer: YES if the current realization is larger than the remaining realizations, i.e., $X_i > \max_{j=i+1}^{n} X_j$ and NO otherwise. In other words, the oracle $\mathcal{O}$ informs the algorithm it should select X_i, or reject it, because there is a reward coming up that is at least as good. Clearly, with no

[1] The Lambert-W function is $W_0(x)$ defined as the solution y to the equation $ye^y = x$.

queries available, one recovers the classical prophet inequality setting, whereas with $n-1$ queries, the strategy of using a query on every X_i, for $i = 1, \ldots, n-1$, leads to the algorithm selecting the highest realization always. Thus, this model interpolates nicely between the two extremes of full or no information about the future.

In this paper, we consider the following different settings.

▶ **Definition 1.1** (Competitive Ratio). *The competitive ratio or* Ratio of Expectations *is denoted by* RoE*. Specifically, for an instance $\mathcal{I}$ of a prophet inequality setting, we denote by* RoE$(x, \mathcal{I})$ *the competitive ratio of an optimal algorithm for $\mathcal{I}$. An algorithm ALG is* α-competitive*, for $\alpha \in [0,1]$, if $\mathbb{E}[ALG] \geq \alpha \cdot \mathbb{E}[\max_i X_i]$, and α is called the* competitive ratio*.*

▶ **Definition 1.2** (Probability of Selecting the Maximum). *The* Probability of selecting the Maximum *realization is denoted by $\mathbb{P}_{\max}$. An algorithm ALG achieves a $\mathbb{P}_{\max}$ of α if it returns a value v such that $\mathbb{P}[v = Z] \geq \alpha$, where $Z = \max\{X_1, \ldots, X_n\}$. In some works (for example [12]), the notation PbM has also been used.*

▶ **Definition 1.3** (IID Setting). *We use the term* IID *to refer to the setting where $X_1, \ldots, X_n$ are independent and identically distributed random variables. We use* non-IID *to refer to the more general setting where $X_1, \ldots, X_n$ are independent, but not necessarily identical.*

▶ **Definition 1.4** ($\textsc{Proph}_m$). *We use $\textsc{Proph}_m$ to refer to the* Top-1-of-m *model, in which the algorithm can choose up to m values, and its payoff is the maximum of the chosen values. We use $\mathcal{O}_m$ refers to our oracle model where the algorithm has access to m oracle calls, and can only select one value.*

Note that the model $\textsc{Proph}_{m+1}$ is comparable to $\mathcal{O}_m$, since in the former the algorithm can choose $m+1$ values, where as the later can ask the oracle m times and then choose an item. To help distinguish between the different settings, we denote each model as $\mathcal{M}(x, y, z)$, where

- x is either $\textsc{Proph}_m$ or $\mathcal{O}_m$ with $m \in \mathbb{N}$,
- y is either IID or non-IID, and
- z is either $\mathbb{P}_{\max}$ or RoE.

Our Contributions

In this paper, we study the oracle model for independent random variables following identical or general distributions with the $\mathbb{P}_{\max}$ and RoE objectives and make the following contributions:

(I) We establish an *equivalence* between the oracle model and the Top-1-of-m model for the $\mathbb{P}_{\max}$ objective.

(II) We show that this equivalence fails to hold for the RoE objective. However, we show that guarantees for RoE in the oracle model translate to guarantees in the Top-1-of-m model, thus further motivating our study of the oracle model.

(III) We resolve the oracle model $\mathcal{M}(\mathcal{O}_m, \text{non-IID}, \text{RoE})$ by presenting a single-threshold algorithm. Our algorithm achieves a competitive ratio of $1 - e^{-\xi_m} = 1 - \mathcal{O}e^{-m/e}$ for general m, where ξ_m is the unique *positive* solution[2] to the equation $1 - e^{-\xi_m} =$

[2] In Section 3, we prove that there is indeed a unique positive solution.

$\frac{\Gamma(m+1,\xi_m)}{m!}$[3]. Furthermore, we show that this lower bound is optimal by showing a construction that yields an equal upper bound. Since we showed that lower bound guarantees for $\mathcal{M}(\mathcal{O}_m, \textsf{non-IID}, \textsf{RoE})$ also hold for the $\mathcal{M}(\textsc{Proph}_{m+1}, \textsf{non-IID}, \textsf{RoE})$ setting, this strictly improves the current state of the art bounds of [14], even though the guarantees are obtained in the weaker oracle model.

(IV) We give a single-threshold algorithm for the $\mathcal{M}(\mathcal{O}_m, \textsf{IID}, \mathbb{P}_{\max})$ model that achieves a $1 - \mathcal{O}(m^{-m/5})$ probability of selecting the maximum, as well as providing an upper bound that is asymptotically (almost) tight. To the best of our knowledge, this is the first result for the $\mathbb{P}_{\max}$ objective and general m in the well studied Top-1-of-m model. Our algorithm achieves a probability of ≈ 0.797 even with $m = 1$ calls to the oracle, a significant improvement on the ≈ 0.58 achieved without oracle calls [12].

As discussed earlier, the main motivation behind our oracle model comes from our first two results which relate it to the Top-1-of-m model.

Equivalence of Models for $\mathbb{P}_{\max}$

▶ **Theorem 1.5.** *The $\mathcal{M}(\mathcal{O}_m, y, \mathbb{P}_{\max})$ model is equivalent to the $\mathcal{M}(\textsc{Proph}_{m+1}, y, \mathbb{P}_{\max})$ model, where $y =$ IID or non-IID. In other words, for every prophet inequality instance, the probability achieved by the best-possible algorithm in the $\mathcal{M}(\mathcal{O}_m, y, \mathbb{P}_{\max})$ model is the same as the one achieved by the best-possible algorithm in the $\mathcal{M}(\textsc{Proph}_{m+1}, y, \mathbb{P}_{\max})$ model.*

In Section 2 and Theorem 1.5, we establish the equivalence between the $\mathcal{M}(\mathcal{O}_m, y, \mathbb{P}_{\max})$ and $\mathcal{M}(\textsc{Proph}_{m+1}, y, \mathbb{P}_{\max})$ models, for $y =$ IID or non-IID. In other words, the best algorithms in these models achieve the same probability of selecting the maximum.

This result might not seem that surprising due to the apparent similarity of the two models. However, thinking about the Top-1-of-m setting from the viewpoint of oracle calls allows for a different perspective that we exploit in our analysis. Furthermore, such intuition can sometimes be wrong, as our next result shows.

Difference of Models for RoE

▶ **Theorem 1.6.** *For every $m \geq 1$, and for* all *input instances $\mathcal{J}$ (of IID or non-IID variables), we have $\textsf{RoE}(\mathcal{O}_m, \mathcal{J}) \leq \textsf{RoE}(\textsc{Proph}_{m+1}, \mathcal{J})$, Furthermore, for every $m \geq 1$, there* exists *an input instance $\mathcal{I}$ with $m + 2$ non-IID random variables, such that $\textsf{RoE}(\mathcal{O}_m, \mathcal{I}) \leq (1 - 1/2^{m+1})\textsf{RoE}(\textsc{Proph}_2, \mathcal{I})$.*

Perhaps more surprisingly, our oracle model and the Top-1-of-m model stop being equivalent when one considers the RoE objective, with the oracle model being strictly weaker.

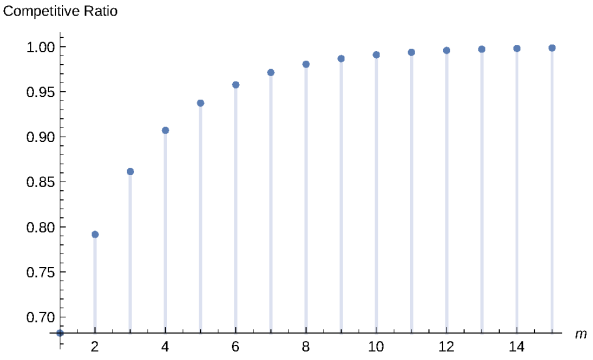
In Section 2, we show Theorem 1.6, which gives a prophet inequality instance, and an algorithm $\mathcal{A}$ for $\mathcal{M}(\textsc{Proph}_{m+1}, \textsf{non-IID}, \textsf{RoE})$, such that no algorithm for $\mathcal{M}(\mathcal{O}_m, \textsf{non-IID}, \textsf{RoE})$ can achieve the same competitive ratio as that of $\mathcal{A}$. Furthermore, we show that any algorithm for $\mathcal{M}(\mathcal{O}_m, y, \textsf{RoE})$ can be modified to an algorithm for $\mathcal{M}(\textsc{Proph}_{m+1}, y, \textsf{RoE})$ that achieves an equal or greater competitive ratio.

Bounding the Performance of the Oracle Model

▶ **Theorem 1.7.** *For every $m \geq 1$, let $\alpha_m = 1 - e^{-\xi_m}$, where ξ_m is the unique positive solution to the equation $1 - e^{-\xi_m} = \frac{\Gamma(m+1,\xi_m)}{m!}$. For any finite sequence $\mathbf{X}$ of non-IID variables, one can compute a value τ, such that the single-threshold algorithm (with initial threshold τ) has competitive ratio $\geq \alpha_m$.*

---

[3] $\Gamma(n, x) = \int_x^\infty t^{n-1}e^{-t}\,dt$ denotes the upper incomplete gamma function.



m	$1-e^{-\xi_m}$	m	$1-e^{-\xi_m}$
1	0.682	9	0.986
2	0.791	10	0.990
3	0.861	11	0.993
4	0.907	12	0.995
5	0.937	13	0.997
6	0.957	14	0.998
7	0.971	15	0.998
8	0.980		

Figure 1.1 The value of $1-e^{-\xi_m}$, for $m=1,\ldots,15$.

After establishing the relationship between our oracle model and the Top-1-of-m model, we turn our attention to upper and lower bounds for the oracle model. First, for the non-IID setting and the RoE objective, we present a simple, single-threshold algorithm achieving a competitive ratio that approaches 1 exponentially fast with respect to m. Even though our algorithm is for the oracle model, for which weaker guarantees are expected due to Theorem 1.6, it improves upon the best-known guarantee for the Top-1-of-m setting, due to Harb [14]. Our algorithm relies on two techniques; sharding and Poissonization, introduced by [14] for the analysis of threshold-based algorithms for prophet inequalities. As an added benefit, the algorithm's analysis is easy to understand.

Specifically, in Section 3, Theorem 1.7, we show that there is a constant ξ_m, such that for the oracle model $\mathcal{M}(\mathcal{O}_m, \text{non-IID}, \text{RoE})$, there exists an algorithm with competitive ratio at least $1-e^{-\xi_m}$. As $m\to+\infty$, this behaves as $1-e^{-m/e+o(m)}$. The competitive ratio plot for $m=1,\ldots 15$ is shown in Figure 1.1.

Matching Upper Bound

▶ **Theorem 1.8.** *For any $m\geq 1$ and $\delta>0$, there exists an input instance $\mathcal{I}$ such that for any algorithm, we have $\text{RoE}(\mathcal{A})\leq 1-e^{-\xi_m}+\delta$.*

In addition, we provide a construction for every m that gives a matching upper bound to the competitive ratio, thus resolving the problem for the case of general distributions and the RoE objective. The construction we have is perhaps of independent interest in the design of counterexamples for other settings, as it combines and generalizes standard counterexamples of prophet inequalities.

In Section 3 and Theorem 1.8, we show that for any $\delta>0$, there exists an instance of $\mathcal{M}(\mathcal{O}_m, \text{non-IID}, z)$, where $z=\text{RoE}$ or $\mathbb{P}_{\max}$, in which no single-threshold algorithm can achieve a $\left(1-e^{-\xi_m}+\delta\right)$-competitive ratio or select the maximum realization with probability at least $\left(1-e^{-\xi_m}+\delta\right)$.

Intuitively, the above follows since an algorithm for the oracle model performs poorly when, every time it uses an oracle call and gets a YES answer, the next value it sees that is at least the queried value is roughly equal, and thus the oracle call was used without any real gain. The idea behind the worst-case for this setting is to have what is essentially a Poisson random variable with rate ξ_m, providing the algorithm with several non-zero values, each roughly the same. By carefully selecting ξ_m in order to equate the probability of having no non-zero values and the probability of having more than m non-zero values, we are forcing the algorithm to use a query for every non-zero realization, thus rendering the oracle calls as useless as possible.

Model	Lower Bound		Upper Bound	
	Prev. Best	Current Best	Prev.	Current Best
RoE, General Setting	$1-\mathcal{O}\left(e^{-m/6}\right)$ [11]	$1-e^{-m/e+o(m)}$	-	$1-e^{-m/e+o(m)}$ single-threshold
$\mathbb{P}_{\max}$, IID Setting	≈ 0.58 [12]	≈ 0.797 $(m=1)$ $1-\mathcal{O}\left(m^{-m/5}\right)$	-	$1-\mathcal{O}\left(m^{-m}\right)$

Figure 1.2 State of the art.

The IID Setting

▶ **Theorem 1.9** (see [13] for proof). *For sufficiently large m, n, there exists an algorithm for the $\mathcal{M}(\mathcal{O}_m, \mathit{IID}, \mathbb{P}_{\max})$ model that selects the maximum realization with probability at least $1-\mathcal{O}\left(m^{-m/5}\right)$.*

Next, we turn our attention to the IID setting with m oracles calls and the $\mathbb{P}_{\max}$ objective. We present a simple, single-threshold algorithm that selects the maximum realization with probability that approaches 1 in a super-exponential fashion. As a warm-up, we first present the analysis for $m=1$ before generalizing it to all m.

Specifically, in Section 4, Theorem 1.9, we show that for $\mathcal{M}(\mathcal{O}_m, \mathsf{IID}, \mathbb{P}_{\max})$, one can select the maximum realization with probability at least $1-\mathcal{O}\left(m^{-m/5}\right)$.

▶ **Theorem 1.10** (see [13] for proof). *There exists an instance of $\mathcal{M}(\mathcal{O}_m, \mathit{IID}, \mathbb{P}_{\max})$ for which no algorithm can select the maximum realization with probability greater than $1-\mathcal{O}(m^{-m})$.*

We also present, in Section 4, Theorem 1.10, an upper bound on the probability of success that is asymptotically tight, up to small multiplicative constants in the exponent. Because of Theorem 1.5, both upper and lower bounds on the probability of success carry over in the Top-1-of-m settings as well. Figure 1.2 contains a summary of our results for the oracle model in the different settings.

1.1 Additional related work

We have already mentioned the works of Gilbert and Mosteller [12], Esfandiari, Hajiaghayi, Lucier and Mitzenmacher [9] and Nuti [20] for the $\mathbb{P}_{\max}$ objective. Related work includes the study of order-aware algorithms by Ezra, Feldman et al. [10], algorithms with fairness guarantees by Correa et al. [6] and algorithms with a-priori information of some of the values by Correa et al. [4]. In addition to these, Esfandiari et al. [9] study a related but distinct variant to ours. They relax the objective to allow the return of one out of the top k realizations, and show exponential upper and lower bounds. Their model, however, is incomparable to ours.

Organization

In Section 2 we relate our model to Top-1-of-m model of Assaf and Samuel-Cahn [3] and prove the reductions. In Section 3 we present our tight algorithm for the non-IID setting. Section 4 contains our algorithms and upper bounds for the IID setting. Due to space constraints, some of the proofs as well as background information on concentration inequalities that we use for our results can be found in the full version [13].

2 Reductions

To motivate our oracle model, we start by establishing an equivalence between $\mathcal{M}(\mathcal{O}_m, y, \mathbb{P}_{\max})$ and $\mathcal{M}(\textsc{Proph}_{m+1}, y, \mathbb{P}_{\max})$, for both the $y = \mathsf{IID}$ and $y = \mathsf{non\text{-}IID}$ case (see Theorem 1.5 below). We also show that, perhaps surprisingly, this equivalence does not hold for the RoE objective; lower bound guarantees for $\mathcal{M}(\mathcal{O}_m, y, \mathsf{RoE})$ translate to guarantees for $\mathcal{M}(\textsc{Proph}_{m+1}, y, \mathsf{RoE})$ (Theorem 1.6), but not the converse. Later, we use this result to improve the best-known lower bound guarantees on $\mathcal{M}(\textsc{Proph}_{m+1}, y, \mathsf{RoE})$.

2.1 The $\mathbb{P}_{\max}$ objective

▶ **Lemma 2.1.** *Fix an instance of the prophet problem. Let $\mathcal{A}$ be an algorithm for this instance in $\mathcal{M}(\mathcal{O}_m, y, \mathbb{P}_{\max})$, where $y = \mathsf{IID}$ or $\mathsf{non\text{-}IID}$. Then, there exists an algorithm $\mathcal{B}$ for tor this instance in $\mathcal{M}(\textsc{Proph}_{m+1}, y, \mathbb{P}_{\max})$, with black-box access to $\mathcal{A}$, such that $\mathbb{P}_{\max}(\mathcal{B}) \geq \mathbb{P}_{\max}(\mathcal{A})$.*

Proof. The idea is for $\mathcal{B}$ to simulate $\mathcal{A}$'s behavior by selecting each realization that $\mathcal{A}$ decides to query. Initially, $\mathcal{B}$ starts with an empty set S of selected values. Whenever $\mathcal{B}$ is presented with a realization X_i, it feeds it to $\mathcal{A}$. If $\mathcal{A}$ decides to select X_i or expend a query for X_i, regardless of the outcome of the query, $\mathcal{B}$ always selects X_i into S, otherwise $\mathcal{B}$ decides not to select X_i. By induction, S contains exactly all the realizations that were queried by $\mathcal{A}$ as well as at most one more realization that might have been selected by $\mathcal{A}$ if it run out of queries. Therefore, $|S| \leq m + 1$.

Observe that $\mathcal{A}$ succeeds if and only if it selects the maximum, and it only selects a realization X_i if (i) it chose to expend a query on X_i, or (ii) when it observed X_i it run out of queries. In both cases, by the description of $\mathcal{B}$, we know that $X_i \in S$, and thus the probability that $\mathcal{B}$ succeeds is at least $\mathbb{P}_{\max}(\mathcal{A})$. ◀

▶ **Lemma 2.2** (see [13] for proof)**.** *Fix an input instance of the prophet problem. Fix an algorithm $\mathcal{B}$ for $\mathcal{M}(\textsc{Proph}_{m+1}, y, \mathbb{P}_{\max})$, where $y = \mathsf{IID}$ or $\mathsf{non\text{-}IID}$. Then, there exists an algorithm $\mathcal{A}$ for $\mathcal{M}(\mathcal{O}_m, y, \mathbb{P}_{\max})$, with black-box access to $\mathcal{B}$, such that such that $\mathbb{P}_{\max}(\mathcal{A}) \geq \mathbb{P}_{\max}(\mathcal{B})$.*

Combining the above two lemmas, we get the following result.

▶ **Theorem 1.5.** *The $\mathcal{M}(\mathcal{O}_m, y, \mathbb{P}_{\max})$ model is equivalent to the $\mathcal{M}(\textsc{Proph}_{m+1}, y, \mathbb{P}_{\max})$ model, where $y = \mathsf{IID}$ or $\mathsf{non\text{-}IID}$. In other words, for every prophet inequality instance, the probability achieved by the best-possible algorithm in the $\mathcal{M}(\mathcal{O}_m, y, \mathbb{P}_{\max})$ model is the same as the one achieved by the best-possible algorithm in the $\mathcal{M}(\textsc{Proph}_{m+1}, y, \mathbb{P}_{\max})$ model.*

2.2 For the RoE objective $\mathcal{O}_m \leq \textsc{Proph}_{m+1}$

We demonstrate that the $\textsc{Proph}_m$ model strictly surpasses the $\mathcal{O}_m$ for $\mathsf{non\text{-}IID}$ random variables.

▶ **Definition 2.3.** *For two integers $i \leq j$, let $[\![i : j]\!] = \{i, i+1, \ldots, j\}$.*

▶ **Lemma 2.4.** *For $m = 1$, there exists an input instance $\mathcal{I}$ with 3 $\mathsf{non\text{-}IID}$ random variables, such that $\mathsf{RoE}(\mathcal{O}_1, \mathcal{I}) \leq \frac{3}{4}\mathsf{RoE}(\textsc{Proph}_2, \mathcal{I})$.*

Proof. For a fixed $\varepsilon > 0$, consider the input instance $\mathcal{I}$ of three independent random variables X_1, X_2, X_3, where

$$X_1 = 1 \text{ w.p. } 1, \qquad X_2 = \begin{cases} 1+\varepsilon & \text{w.p. } \frac{1}{2}-\varepsilon \\ 0 & \text{otherwise} \end{cases}, \qquad \text{and} \qquad X_3 = \begin{cases} \frac{1}{\varepsilon} & \text{w.p. } \varepsilon \\ 0 & \text{otherwise} \end{cases}.$$

We have that

$$\mathbb{E}\Big[\max\{X_1, X_2, X_3\}\Big] = \frac{1}{\varepsilon}\varepsilon + (1+\varepsilon)(1-\varepsilon)\left(\frac{1}{2}-\varepsilon\right) + 1(1-\varepsilon)\left(\frac{1}{2}+\varepsilon\right) = 2 - O(\varepsilon).$$

For small ε, an algorithm $\mathcal{B}$ that is optimal for the $\textsc{Proph}_2$ model in this instance is to select X_1, ignore X_2 and then select X_3 if it is non-zero. This yields

$$\mathbb{E}[\mathcal{B}] = 1(1-\varepsilon) + 1/\varepsilon \cdot \varepsilon = 2 - \varepsilon.$$

However, the optimal $\mathcal{A}$ for the oracle model queries $\mathcal{O}$ at X_1. With probability $(1-\varepsilon)(1/2+\varepsilon)$, it stops and select X_1, getting a value of 1. Otherwise, it continues, with no oracle calls left. It ignores X_2 and select X_3. Thus,

$$\mathbb{E}[\mathcal{A}] = 1\left(\frac{1}{2}+\varepsilon\right)(1-\varepsilon) + \frac{1}{\varepsilon}\varepsilon = \frac{3}{2} + \frac{\varepsilon}{2} - \varepsilon^2.$$

The competitive ratios of $\mathcal{A}$ is $\mathsf{RoE}(\mathcal{O}_1, \mathcal{I}) = \dfrac{\frac{3}{2}+\frac{\varepsilon}{2}-\varepsilon^2}{2-O(\varepsilon)} = \dfrac{3}{4} + O(\varepsilon) \to \dfrac{3}{4}$, as $\varepsilon \to 0$, whereas the competitive ratio of $\mathcal{B}$, as $\varepsilon \to 0$, is

$$\mathsf{RoE}(\textsc{Proph}_2, \mathcal{I}) = \frac{2-\varepsilon}{2+O(\varepsilon)} = 1 - O(\varepsilon) \to 1. \qquad \blacktriangleleft$$

The above example, appropriately generalized for $m > 1$ by having random variables

$$X_1 = 1 \quad \text{w.p. } 1, \quad X_i = \begin{cases} 1+(i-1)\varepsilon & \text{w.p. } \frac{1}{2}-\varepsilon \\ 0 & \text{w.p. } \frac{1}{2}+\varepsilon \end{cases}, \quad \text{for } i = 2, \ldots, m+1, \text{ and}$$

$$X_{m+2} = \begin{cases} \frac{1}{\varepsilon} & \text{w.p. } \varepsilon \\ 0 & \text{w.p. } 1-\varepsilon \end{cases},$$

shows that the gap between $\mathsf{RoE}(\mathcal{O}_m, \mathcal{I}))$ and $\mathsf{RoE}(\textsc{Proph}_{m+1}, \mathcal{I}))$ is at most $1 - 1/2^{m+1}$ for general m. The analysis of this example for general m is similar to the $m = 1$ case. We do not present it here as, even though this example is very simple, this gap is not the tightest possible. For a tighter gap between the competitive ratio of the two models, see the example in the proof of Theorem 1.8.

▶ **Lemma 2.5.** *For any input instance $\mathcal{I}$, we have $\mathsf{RoE}(\textsc{Proph}_{m+1}, \mathcal{I}) \geq \mathsf{RoE}(\mathcal{O}_m, \mathcal{I})$, for IID or non-IID variables.*

Proof. Let $\mathcal{A}$ be the algorithm in $\mathcal{M}(\mathcal{O}_m, \mathsf{RoE}, \mathcal{I})$ realizing the maximum RoE for $\mathcal{I}$. We construct an algorithm $\mathcal{B} \in \mathcal{M}(\mathcal{O}_m, \mathsf{RoE}, \mathcal{I})$.

The algorithm $\mathcal{B}$ simulates $\mathcal{A}$'s behavior by selecting each realization that $\mathcal{A}$ decides to query. Initially, $\mathcal{B}$ starts with an empty set S. Whenever $\mathcal{B}$ is presented with a realization X_i, it feeds it to $\mathcal{A}$. If $\mathcal{A}$ decides to return X_i, or performs an oracle query for X_i, the algorithm $\mathcal{B}$ adds X_i to S.

Observe that the algorithm $\mathcal{A}$ stops as soon as an oracle query returns NO. Thus, the simulation $\mathcal{B}$ of $\mathcal{A}$, assumes the oracle always answers YES (i.e., a larger value is coming up in the future). (i.e., the simulation replaces a call to the oracle by a function that always returns YES), as this enables it (potentially) to save more values into the available slots, thus increasing its RoE.

The set S contains exactly all the realizations that were queried by $\mathcal{A}$, as well as at most one additional realization returned by $\mathcal{A}$. Therefore, $|S| \leq m+1$.

Every possible sequence of realizations $\mathcal{A}$ queried (or selected to return) are in S. Therefore, if $V_{\mathcal{A}}$ is the value returned by $\mathcal{A}$ and $V_{\mathcal{B}}$ is the value returned by $\mathcal{B}$, we have $V_{\mathcal{B}} \geq V_{\mathcal{A}}$, which readily implies that $\mathsf{RoE}(\mathcal{B}) \geq \mathsf{RoE}(\mathcal{A})$. ◀

▶ **Theorem 1.6.** *For every $m \geq 1$, and for* all *input instances $\mathcal{J}$ (of IID or non-IID variables), we have $\mathsf{RoE}(\mathcal{O}_m, \mathcal{J}) \leq \mathsf{RoE}(\textsc{Proph}_{m+1}, \mathcal{J})$, Furthermore, for every $m \geq 1$, there* exists *an input instance $\mathcal{I}$ with $m+2$ non-IID random variables, such that $\mathsf{RoE}(\mathcal{O}_m, \mathcal{I}) \leq (1 - 1/2^{m+1})\mathsf{RoE}(\textsc{Proph}_2, \mathcal{I})$.*

3 The non-IID settings

By Theorem 1.6, any guarantees we provide for the oracle model with the RoE objective can be directly translated to guarantees for the Top-1-of-m model, improving upon the previous work on this model [3, 2, 11, 14]. We provide a simple, single-threshold algorithm that resolves the RoE objective in the oracle model.

3.1 The exponent sequence

▶ **Definition 3.1** (Exponent Sequence). *For every $m \geq 1$, let ξ_m denote the unique positive solution to the following equation:*

$$1 - e^{-\xi_m} = \frac{\Gamma(m+1, \xi_m)}{m!},$$

where $\Gamma(m+1, x) = \int_{t=x}^{\infty} t^m e^{-t}\, dt$ denotes the upper incomplete gamma *function. We call the sequence $\{\xi_m\}_{m \in \mathbb{N}}$ the* exponent sequence.

We show below that the optimal competitive ratio of $\mathcal{M}(\mathcal{O}_m, \mathsf{non\text{-}IID}, \mathsf{RoE})$ is exactly $1 - e^{-\xi_m}$. It is known that, for $x \geq 0$ and an integer $m+1 > 0$, we have

$$\Gamma(m+1, x) = m!\, e^{-x} \sum_{k=0}^{m} \frac{x^k}{k!} \leq m! e^{-x} e^{x} \leq m!. \tag{3.1}$$

As such, the above equation on the value of ξ_m, becomes

$$1 - e^{-\xi_m} = e^{-\xi_m} \sum_{k=0}^{m} \frac{(\xi_m)^k}{k!} \qquad \Longleftrightarrow \qquad \sum_{k=m+1}^{\infty} \frac{(\xi_m)^k}{k!} = 1.$$

This readily implies that the exponent sequence is monotonically increasing, and $m/e^2 \leq \xi_m \leq m$.

▶ **Definition 3.2.** *Let $\mathsf{q}_{k+1}(x) = \frac{\Gamma(k+1,x)}{k!} = e^{-x} \sum_{j=0}^{k} \frac{x^j}{j!}$. This implies $\mathsf{q}_{m+1}(\xi_m) = 1 - e^{-\xi_m}$.*

▶ **Lemma 3.3.** *$\mathsf{q}'_{m+1}(x) = -e^{-x} \frac{x^m}{m!}$.*

Proof. As $(e^{-x})' = -e^{-x}$, we have $\mathsf{q}'_{m+1}(x) = -e^{-x} + \sum_{j=1}^{m}\left(e^{-x}\frac{x^{j-1}}{(j-1)!} - e^{-x}\frac{x^j}{j!}\right) = -e^{-x} + e^{-x} - e^{-x}\frac{x^m}{m!} = -e^{-x}\frac{x^m}{m!}$. ◀

▶ **Lemma 3.4** (see [13] for proof). *For all $m \geq 1$, we have $(m!)^{1/m} < \xi_m < ((m+1)!)^{1/m+1}$.*

▶ Remark 3.5. Setting $\nu(x) = \nu(m,x) = \frac{\Gamma(m+1,x)}{m!}$, and arguing as in Lemma 3.4, we have $\nu'(x) < 0$, which readily implies that $\nu(x)$ is monotonically decreasing.

Stirling's formula applied to Lemma 3.4 readily implies the following.

▶ **Lemma 3.6.** *We have* $\lim_{m\to\infty}\frac{\xi_m}{m} = \frac{1}{e}$.

▶ **Lemma 3.7** (see [13] for proof). *For all $k, m \geq 0$ integers, we have*

$$f(k,m) = \sum_{j=1}^{k}\frac{\xi_m^j}{j!} - \sum_{j=m+1}^{m+k}\frac{\xi_m^j}{j!} \geq 0.$$

3.2 Background: Sharding, poissonization, and stochastic dominance

For a sequence of random variables $\mathbf{X} = X_1, \ldots, X_n$, let $|\alpha \leq \mathbf{X} \leq \beta| = |\{i \mid \alpha \leq X_i \leq \beta\}|$ denote the number of realizations in this sequence falling in the interval $[\alpha, \beta]$.

3.2.1 Sharding

For the lower bound, we use poissonization and sharding [14]. Given random variables $X_1, \ldots, X_n$ with cdfs $F_1, \ldots, F_n$, instead of sampling X_i from F_i, we instead replace it with a sequence of K independent random variables $\mathbf{H}_i = Y_{i,1}, \ldots, Y_{i,K}$, such that $\max_j Y_{i,j}$ has the same distribution as X_i. Specifically, the cdf of $Y_{i,j}$, for all j, is $F_i^{1/K}$. Thus, the distribution of $\max\{Y_{i,1}, \ldots, Y_{i,K}\}$ is the same as X_i. This creates a new sequence of Kn samples $\mathbf{S} = \mathbf{H}_1 \cdot \mathbf{H}_2 \cdot \cdots \cdot \mathbf{H}_n$, where $\cdot$ is the concatenation operator. Observe that for any $\alpha \geq 0$ and integer t, we have

$$\mathbb{P}[|\mathbf{X} \geq \alpha| > t] < \mathbb{P}[|\mathbf{S} \geq \alpha| > t].$$

Intuitively, this implies that, for threshold algorithms, an instance consisting of $\mathbf{S}$ instead of $\mathbf{X}$ can only generate worse results. We emphasize that this sharding technique is done only for analysis purposes.

3.2.2 Poissonization

▶ **Definition 3.8** (Poisson Distribution). *A random variable X has Poisson distribution with rate λ, denoted by $X \sim \mathrm{Pois}(\lambda)$, if $\mathbb{P}[X = i] = \lambda^k e^{-\lambda}/k!$. Conveniently, $\mathbb{E}[X] = \mathbb{V}[X] = \lambda$.*

The purpose of the sharding is to be able to bound quantities of the form $\mathbb{P}[|\beta \leq \mathbf{S} \leq \tau| = t]$. As K grows, the underlying random variable $|\beta \leq \mathbf{S} \leq \tau|$ has a binomial distribution that converges to a Poisson distribution.

▶ **Observation 3.9.** *For $c \in (0,1]$, we have, using L'Hôpital's rule, that* $\lim_{x\to\infty} x(1-c^{1/x}) = \lim_{x\to\infty}\frac{1-\exp(\log(c)/x)}{1/x} = \lim_{x\to\infty}\frac{\log(c)\exp(\log(c)/x)/x^2}{-1/x^2} = -\log c$, *where* $\log = \log_e$.

Let τ be a threshold such that $\sum_{i=1}^{n}\sum_{j=1}^{K}\mathbb{P}[Y_{i,j}\geq\tau]=c$ for some constant c to be determined shortly. We can rewrite this into the following.

$$\sum_{i=1}^{n}K\Big(1-\mathbb{P}[X_i\leq\tau]^{1/K}\Big)=c. \tag{3.2}$$

The limit of Eq. (3.2), as $K\to+\infty$, is $\sum_{i=1}^{n}-\log\mathbb{P}[X_i\leq\tau]=c$, by Observation 3.9. Equivalently, for $Z=\max\{X_1,\ldots,X_n\}$, we have

$$e^{-c}=\exp\Big(\sum_{i=1}^{n}\log\mathbb{P}[X_i\leq\tau]\Big)=\prod_{i=1}^{n}\mathbb{P}[X_i\leq\tau]=\mathbb{P}[X_1,\ldots,X_n\leq\tau]=\mathbb{P}[Z\leq\tau].$$

In particular, the distribution of the number of indices j, such that $Y_{i,j}\geq\tau$ can be well approximated with a Poisson distribution. Specifically, let $V_{i,j}=1\iff Y_{i,j}\geq\tau$, and consider the sum $V_i=\sum_{j=1}^{K}V_{i,j}$. The variable $V_i\sim\mathrm{bin}(K,\psi_i)$, where $\psi_i=1-\mathbb{P}[X_i\leq\tau]^{1/K}$.

Let $\lambda_i=\psi_iK$, and consider the random variable $U_i\sim\mathrm{Pois}(\lambda_i)$ (i.e., U_i has a Poisson distribution with rate λ_i). Intuitively, V_i and U_i have similar distributions. Formally, Le Cam theorem implies that for any set $T\subseteq\{0,1,\ldots,K\}$, we have $|\mathbb{P}[V_i\in T]-\mathbb{P}[U_i\in T]|\leq 2K\psi_i^2=2\lambda_i^2/K\leq 2c^2/K$, by Eq. (3.2). The later quantity goes to zero as K increases.

Thus, we get a variable U_i with a Poisson distribution for each shard sequence $\mathbf{H}_i$, with rate λ_i, where U_i models the number of times we encounter in $\mathbf{H}_i$ values larger than τ. Thus, $U_\tau=\sum_i U_i$ models the total number of times in the splintered sequence $\mathbf{S}$ that values encountered are larger than τ. The variable U_τ has a Poisson distribution with rate $\lambda_\tau=\sum_{i=1}^{n}\lambda_i$.

3.2.3 The distribution in a range

Repeating the same process with a bigger threshold $\beta>\tau$, would yield a similar Poisson random variable U_β with a *lower* rate λ_β. The quantity $\Delta=U_\tau-U_\beta$ is the number of values in $\mathbf{S}$ in the range $[\tau,\beta]$. Furthermore, Δ has a Poisson distribution with rate $\lambda_\tau-\lambda_\beta$. Specifically, $\mathbb{P}[|\beta\leq\mathbf{S}\leq\tau|=t]=\mathbb{P}[\Delta=t]$.

The key to our analysis is that the variables Δ and U_β are independent (in the limit as K increases).

3.2.4 Stochastic dominance

A standard observation is that for a non-negative random variable X, we have that $\mathbb{E}[X]=\int_{x=0}^{\infty}\mathbb{P}[X\geq x]\,\mathrm{d}x$. Thus, for $Z=\max\{X_1,\ldots,X_n\}$, and for an algorithm $\mathcal{A}$, if one can guarantee that there is $c\in[0,1]$, such that for all $\nu\geq 0$, $\mathbb{P}[\mathcal{A}\geq\nu]\geq c\,\mathbb{P}[Z\geq\nu]$, then

$$\mathbb{E}[\mathcal{A}]=\int_0^\infty\mathbb{P}[\mathcal{A}\geq x]\,\mathrm{d}x\geq c\int_0^\infty\mathbb{P}[Z\geq x]\,\mathrm{d}x\geq c\,\mathbb{E}[Z]\,,$$

and thus c is a lower bound on the competitive ratio of $\mathcal{A}$. This argument is used in several results on prophet inequalities and is often referred to as *majorizing* $\mathcal{A}$ with Z.

3.3 An optimal single-threshold algorithm

Here, we describe a single-threshold algorithm that achieves the optimal competitive ratio in the oracle model.

▶ **Definition 3.10** (Single-Threshold Algorithm). *A single threshold algorithm for $\mathcal{O}_m$ sets a threshold τ, and starts observing the sequence. Whenever it encounters a realization $> \tau$, the algorithm stops and queries the oracle whether all the values remaining in the suffix of the sequence are of value $\leq \tau$. If the oracle returns YES, the algorithm accepts the current value and stops. Otherwise, it raises its threshold to $\tau = X_i$ and continues. If the oracle runs out of oracle calls, it selects the first value encountered after the last oracle call that is bigger than τ (which exists, since all oracle calls returned NO).*

While technically, the querying threshold of the algorithm might change during its execution, we call the algorithm a single-threshold algorithm since it uses a single-threshold to decide whether to query the oracle or not, and this threshold *does not change with i*, unlike for example the optimal DP for the IID prophet inequality or the prophet secretary model. Our oracle model is quite different than most other prophet inequality models in the sense that the algorithm has some knowledge of the (true) future. Of course, any algorithm that knows that the maximum of $X_{i+1}, \ldots, X_n$ is larger than X_i would be wasting queries if it expended them on some $X_j < X_i$ for $j > i$, and thus the spirit of it being a single-threshold algorithm to decide whether to query the oracle or not remains.

▶ **Theorem 1.7.** *For every $m \geq 1$, let $\alpha_m = 1 - e^{-\xi_m}$, where ξ_m is the unique positive solution to the equation $1 - e^{-\xi_m} = \frac{\Gamma(m+1,\xi_m)}{m!}$. For any finite sequence $\mathbf{X}$ of non-IID variables, one can compute a value τ, such that the single-threshold algorithm (with initial threshold τ) has competitive ratio $\geq \alpha_m$.*

Proof. Let $\mathbf{X} = X_1, \ldots, X_n$, and $Z = \max_i X_i$. The threshold τ is the $e^{-\xi_m}$ quantile of the maximum, i.e. $\mathbb{P}[Z \leq \tau] = e^{-\xi_m}$. We use $\mathcal{A}(\mathbf{X})$ to denote the result of running the algorithm on $\mathbf{X}$.

As suggested in Section 3.2.1 (for the analysis), we imagine running the algorithm on the splintered sequence $\mathbf{S}$. Somewhat counterintuitively, imagine first generating $\mathbf{S}$, and computing $X_i = \max_j Y_{i,j}$, see Section 3.2.1. Thus, $\max \mathbf{S} = \max \mathbf{X}$. For the sequence $\mathbf{S}$, let $\mathbf{S}_{\geq\tau}$ denote the subsequence of elements of $\mathbf{S}$ that their values are above τ. Observe that $\mathbf{X}_{\geq\tau}$ is a subsequence of $\mathbf{S}_{\geq\tau}$. Thus, we analyze the algorithm performance on $\mathbf{S}$.

Let $\beta \in [0, \tau]$. The probability the algorithm selects a value above β is equal to the probability it selects any value. Thus,

$$\mathbb{P}[\mathcal{A}(\mathbf{X}) \geq \beta] = \mathbb{P}[\mathcal{A} \geq \tau] = \mathbb{P}[Z \geq \tau] = 1 - e^{-\xi_m} \geq \left(1 - e^{-\xi_m}\right) \mathbb{P}[Z \geq \beta]. \tag{3.3}$$

For $\beta \in [\tau, +\infty)$, let $\mathbb{P}[Z \leq \beta] = e^{-q} > e^{-\xi_m}$, implying $\mathbb{P}[Z \geq \beta] = 1 - e^{-q}$. By sharding and Poissonization, the number of shards in the range $[\tau, \beta]$ (resp. $\geq \beta$) is a Poisson random variable Δ (resp. U_β) with rate $\xi_m - q$ (resp. q), see Section 3.2.3. Critically, U_β and Δ are independent. Consider the event of there being at most m values in the range $[\tau, \beta]$, and there being at least one value in $[\beta, +\infty)$. The value $\mathcal{A}(\mathbf{X}) \geq \beta$ in that case. Hence, by the independence of Δ and U_β, we have

$$\frac{\mathbb{P}[\mathcal{A}(\mathbf{X}) \geq \beta]}{\mathbb{P}[Z \geq \beta]} \geq \frac{\mathbb{P}[(U_\beta \geq 1) \cap (0 \leq \Delta \leq m)]}{\mathbb{P}[Z \geq \beta]} = \frac{\mathbb{P}[U_\beta \geq 1]}{\mathbb{P}[Z \geq \beta]} \mathbb{P}[0 \leq \Delta \leq m] = \mathbb{P}[0 \leq \Delta \leq m].$$

Now, we have

$$\mathbb{P}[0 \leq \Delta \leq m] = \sum_{i=0}^{m} e^{-(\xi_m - q)} \frac{(\xi_m - q)^i}{i!} = \frac{\Gamma(m+1, \xi_m - q)}{m!} \geq \frac{\Gamma(m+1, \xi_m)}{m!} = 1 - e^{-\xi_m}.$$

by Eq. (3.1), Remark 3.5 and Definition 3.1.

The above implies that, for any $\beta \geq 0$, we have $\mathbb{P}[\mathcal{A}(\mathbf{X}) \geq \beta] \geq (1 - e^{-\xi_m}) \mathbb{P}[Z \geq \beta]$, Namely, $\mathsf{RoE}(\mathcal{A}) \geq 1 - e^{-\xi_m}$. ◀

3.4 A matching upper bound for single-threshold algorithms

To this end, we present an input sequence for which no algorithm can do better for the oracle that answers if $X_i > \max_{j=i+1}^{n} X_j$. Our upper bound is with respect to the strongest possible form of adversary, the almighty adversary, who knows from the beginning all possible realizations as well as the outcome of any random coins tossed by the algorithm.

Input instance

The input instance $\mathcal{I}$ consists of $n+2$ random variables, for sufficiently large n. Each of these random variables can have only two values – either zero or some positive value. Specifically, for $\varepsilon > 0$ sufficiently small, let

$$X_1 = 1, \quad X_i = \begin{cases} 1 + \varepsilon(i-1) & \text{w.p. } \frac{\xi_m}{n} \\ 0 & \text{otherwise} \end{cases}, \quad \text{for} \quad i \in [\![2 : n+1]\!], \quad \text{and}$$

$$X_{n+2} = \begin{cases} \frac{1}{\varepsilon} & \text{w.p. } \varepsilon \\ 0 & \text{otherwise} \end{cases}.$$

By Lemma 3.6, we have $\xi_m \approx m/e$ and as such, the expected number of non-zero entries in this sequence is (roughly) $m/e + 1$.

▶ **Lemma 3.11.** *For $Z = \max_i X_i$, we have $\mathbb{E}[Z] = 2$ as $\varepsilon \to 0$.*

Proof. Let $Z' = \max_{i \in [\![n+1]\!]} X_i$. Observe that $Z' = 1$. As such, for $Z = \max(Z', X_{n+2})$, we have $\mathbb{E}[Z] = \mathbb{E}[\max_i X_i] = (1/\varepsilon)\varepsilon + (1-\varepsilon)\,\mathbb{E}[Z'] \xrightarrow[\varepsilon \to 0]{} 2$. ◀

Next, we will need the following result on the approximation of a binomial distribution by a Poisson distribution, known as Le Cam's theorem ([5, 8]).

▶ **Theorem 3.12** (Le Cam's theorem). *Let $X_1, \ldots, X_n$ be independent Bernoulli random variables, with $p_i = \mathbb{P}[X_i = 1]$, for $i \in [\![n]\!]$. Let $S = \sum_i X_i$ and $\lambda = \sum_i p_i$. Then S has a Poisson binomial distribution with expectation λ. Furthermore, let $Y \sim \operatorname{Pois} \lambda$. Then we have*

$$\sum_{i=0}^{n} |\mathbb{P}[S = i] - \mathbb{P}[Y = i]| = \sum_{i=0}^{n} \left| \mathbb{P}[S = i] - e^{-\lambda} \frac{\lambda^i}{i!} \right| \leq 2 \sum_{i=1}^{n} p_i^2.$$

▶ **Observation 3.13.** *Let $\widehat{X}_i$ be an indicator variable for the event that $X_i = 1$. For sufficiently large n, $\nabla = \sum_{i=2}^{n+1} \widehat{X}_i$ has a binomial distribution that can be well approximated by a Poisson distribution (Theorem 3.12) with rate ξ_m. That is, $\lim_{n \to \infty} \mathbb{P}[\nabla = k] = e^{-\xi_m} \dfrac{(\xi_m)^k}{k!}$.*

Observe that $\lim_{n \to \infty} \mathbb{P}[\nabla \leq k] = \sum_{i=0}^{k} e^{-\xi_m} \frac{(\xi_m)^i}{i!} = \mathsf{q}_{k+1}(\xi_m)$. In the analysis to follow, we assume $n \to \infty$.

▶ **Theorem 1.8.** *For any $m \geq 1$ and $\delta > 0$, there exists an input instance $\mathcal{I}$ such that for any algorithm, we have $\mathsf{RoE}(\mathcal{A}) \leq 1 - e^{-\xi_m} + \delta$.*

Proof. First, we discuss the strategy that the adversary adopts: the adversary first observes all values. Suppose k nonzero values show up from $X_2, ..., X_n$ at indices $U = \{i_1, ..., i_k\}$, and all other $n-k$ values from $X_2, \ldots, X_{n+1}$ at indices $B = \{\hat{i}_1, \ldots, \hat{i}_{n-k}\}$ are zero. One can easily see that it is optimal for the adversary to provide the random variables in the order $X_{\sigma(1)}, \ldots, X_{\sigma(n+2)}$ where σ is defined as $\sigma(1) = 1$, $\sigma(j) = i_j, j = 2, \ldots, k+1$, $\sigma(j) = \hat{i}_j$ and finally $\sigma(n+2) = n+2$. In other words, the adversary stacks all the k non zero values from $X_2, \ldots, X_{n+1}$ starting from index 2 to index $k+1$.

Now we consider an algorithm for this setting. The algorithm is aware of the adversary's strategy and thus knows that it will observe $X_{\sigma(1)} = X_1$, then a stream of k ones (where k is unknown), then $n-k$ zeros, and finally $X_{\sigma(n+2)} = X_{n+2}$. The algorithm has a simple decision to make in the beginning; it either queries at X_1 and if the answer is NO it continues to $X_{\sigma(2)}, \dots X_{\sigma(n+1)}$ with $m-1$ oracle calls, or it can just proceed to $X_{\sigma(2)}, \dots X_{\sigma(n+1)}$ with m oracle calls. Thus, the only difference in the two cases is that in the former, it has only $m-1$ oracle calls for $X_{\sigma(2)}, \dots, X_{\sigma(n+1)}$ but it gets an expected reward of 1 if $X_{\sigma(2)} = \dots = X_{\sigma(n+2)} = 0$, and in the later case, it has m oracle calls for $X_{\sigma(2)}, \dots, X_{\sigma(n+1)}$, but it gets 0 reward if $X_{\sigma(2)} = \dots = X_{\sigma(n+2)} = 0$.

Let k be the number of non-zeros in $X_2, \dots, X_{n+1}$ (i.e., $X_{\sigma(k+1)}$ is the last 1). When the algorithm observes the stream of approximate ones from $X_{\sigma(2)}, ..., X_{\sigma(n+1)}$, it needs to decide indices $S \subseteq [\![2 : n+1]\!], |S| \leq m$ where it will expend the oracle call. Clearly, it is suboptimal to use the oracle at a 0 value, since regardless, the algorithm will receive a value of 0 in the end if it fails. Consider what happens if the algorithm decides to query at index $i \in [\![2 : n+1]\!]$ with $X_{\sigma(i)} = 1$. If $X_{\sigma(i+1)} = \dots X_{\sigma(n+1)} = 0$, then the algorithm gets on expectation $1/\varepsilon \cdot \varepsilon + (1-\varepsilon) \cdot 1 \xrightarrow[\varepsilon \to 0]{} 2$ reward on expectation. However, if $X_{\sigma(i+1)} > 0$, then the oracle will return NO because $X_{\sigma(i)} \not> \max(X_{\sigma(i+1)}, \dots, X_{\sigma(n+2)})$. On the other hand, if the algorithm does not query at index $k+1$ (i.e., $(k+1) \notin S$), then the algorithm gets on expectation $\mathbb{E}\big[X_{\sigma(n+2)}\big] = \mathbb{E}[X_{n+2}] = 1/\varepsilon \cdot \varepsilon = 1$.

Hence, the crucial observation is that an algorithm starting at $X_{\sigma(2)}$ that uses its query calls at indices $S \subseteq [\![2 : n+1]\!]$ gets on expectation 2 if and only if $(k+1) \in S$, and 1 otherwise. Thus, for algorithm $\mathcal{A}_1$ that skips $X_{\sigma(1)}$ and uses its oracles at indices $S, |S| = m$, it satisfies

$$\begin{aligned}
\mathbb{E}[\mathcal{A}_1] &= 2 \cdot \sum_{i \geq 0, (i+1) \in S} e^{-\xi_m} \frac{\xi_m^i}{i!} + 1 \cdot \sum_{i \geq 0, (i+1) \notin S} e^{-\xi_m} \frac{\xi_m^i}{i!} \\
&= \sum_{i \geq 0} e^{-\xi_m} \frac{\xi_m^i}{i!} + \sum_{i \geq 0, (i+1) \in S} e^{-\xi_m} \frac{\xi_m^i}{i!} \\
&= 1 + \sum_{(i+1) \in S} e^{-\xi_m} \frac{\xi_m^i}{i!}
\end{aligned}$$

On the other hand, for algorithm $\mathcal{A}_2$ that uses its oracle at $X_{\sigma(1)}$ and uses its remaining oracles at indices $S' \in [\![2 : n+1]\!], |S'| = m-1$, it gets an extra benefit of getting a reward with expected value 2 (as $\varepsilon \to 0$) if $X_{\sigma(2)} = \dots = X_{\sigma(n+1)} = 0$. Hence, it satisfies

$$\begin{aligned}
\mathbb{E}[\mathcal{A}_2] &= \left(e^{-\xi_m} \cdot 2\right) + \left(2 \cdot \sum_{i \geq 0, (i+1) \in S'} e^{-\xi_m} \frac{\xi_m^i}{i!}\right) + \left(1 \cdot \sum_{i \geq 1, (i+1) \notin S'} e^{-\xi_m} \frac{\xi_m^i}{i!}\right) \\
&= \left(\sum_{i \geq 0} e^{-\xi_m} \frac{\xi_m^i}{i!}\right) + e^{-\xi_m} + \sum_{(i+1) \in S'} e^{-\xi_m} \frac{\xi_m^i}{i!} \\
&= 1 + e^{-\xi_m} + \sum_{(i+1) \in S'} e^{-\xi_m} \frac{\xi_m^i}{i!}.
\end{aligned}$$

First, we show that the expression $\sum_{(i+1) \in S} e^{-\xi_m} \frac{\xi_m^i}{i!}$ subject to $S \subseteq [\![2 : n+1]\!], |S| = m$ is maximized for $S^* = [\![2 : m+1]\!]$. Note that it is easy to verify that for a Poisson distribution with rate λ, its probability mass function $e^{-\lambda}\lambda^i/i!$ is increasing for $i < \lambda$, and decreasing

after $i > \lambda$. Hence, the optimal $S^* = [\![k : k+m-1]\!]$ for some $k \geq 2$ that "covers" the rate ξ_m (this is the region with the most mass for a Poisson distribution). The optimal choice of k is $k = 2$ because

$$\sum_{i=1}^{m} e^{-\xi_m} \frac{\xi_m^i}{i!} - \sum_{i=k-1}^{k+m-2} e^{-\xi_m} \frac{\xi_m^i}{i!} = \sum_{i=1}^{k-2} e^{-\xi_m} \frac{\xi_m^i}{i!} - \sum_{i=m+1}^{m+k-2} e^{-\xi_m} \frac{\xi_m^i}{i!} \geq 0,$$

where the last inequality holds by Lemma 3.7. Similarly, $k = 2$ is optimal for when $|S| = m-1$. Hence, we get the inequalities

$$\mathbb{E}[\mathcal{A}_1] \leq 1 + \sum_{i=1}^{m} e^{-\xi_m} \frac{\xi_m^i}{i!} = 1 + \mathsf{q}_{m+1}(\xi_m) - e^{-\xi_m},$$

$$\mathbb{E}[\mathcal{A}_2] \leq 1 + e^{-\xi_m} + \sum_{i=1}^{m-1} e^{-\xi_m} \frac{\xi_m^i}{i!} = 1 + \mathsf{q}_m(\xi_m).$$

Thus, we have

$$\max(\mathbb{E}[\mathcal{A}_1], \mathbb{E}[\mathcal{A}_2]) \leq 1 - e^{-\xi_m} + \mathsf{q}_m(\xi_m) + e^{-\xi_m} \max\left\{1, \frac{\xi_m^m}{m!}\right\}$$

But recall from Lemma 3.4 that $\xi_m^m \geq m!$, thus

$$\begin{aligned}\max(\mathbb{E}[\mathcal{A}_1], \mathbb{E}[\mathcal{A}_2]) &\leq 1 - e^{-\xi_m} + \mathsf{q}_m(\xi_m) + e^{-\xi_m} \cdot \frac{\xi_m^m}{m!} \\ &= 1 - e^{-\xi_m} + \mathsf{q}_{m+1}(\xi_m) \\ &= 2\big(1 - e^{-\xi_m}\big).\end{aligned}$$

Therefore, the competitive ratio of every algorithm is

$$\mathsf{RoE} \leq \frac{2\big(1 - e^{-\xi_m}\big)}{2} = 1 - e^{-\xi_m}.$$ ◀

4 The IID settings

Motivated by the early work of [12] for the Top-1-of-m model, in this section we study the IID setting and the $\mathbb{P}_{\max}$ objective. As a warm-up, we take a look at the IID setting with the $\mathbb{P}_{\max}$ objective and the case of $m = 1$, providing a simple, single-threshold algorithm.

4.1 A single-threshold algorithm for $m = 1$

Our single-threshold algorithm $\mathcal{A}_p$ for $\mathcal{M}(\mathcal{O}_1, \mathsf{IID}, \mathbb{P}_{\max})$ selects a threshold τ equal to the pth quantile of the given distribution $\mathcal{D}$, for some $p \in [0, 1]$. In other words, τ is set such that $p = \mathbb{P}[X_i \geq \tau]$. The first time the algorithm observes a realization above τ, it queries the oracle to see whether the realization should be selected or not. If it continues, it simply accepts the first value encountered above the observed realization on which it queried $\mathcal{O}$.

▶ **Lemma 4.1.** *There exists $p \in [0, 1]$ such that $\mathcal{A}_p$ selects the maximum realization with probability at least 0.797 in the $\mathcal{M}(\mathcal{O}_1, \mathsf{IID}, \mathbb{P}_{\max})$ model for large n.*

Proof. Let Y be the total number of realizations above τ, and $i_1 < i_2 < \cdots < i_Y$ be the indices of the random variables above τ, i.e. $X_{i_t} > \tau$, for $t = 1, \ldots, Y$. Furthermore, let r_t be the rank of X_{i_t} in $\mathcal{X} = \{X_{i_1}, \ldots, X_{i_Y}\}$, i.e. the number k such that X_{i_t} is the kth largest number in $\mathcal{X}$, and Z be the maximum realization of $X_1, \ldots, X_n$.

X_{i_1} is the first realization we observe above τ. Notice that if $r_1 = 1$ or $r_1 = 2$ then the algorithm always selects the maximum realization Z. In other words, given that $Y = 1$ or $Y = 2$, the algorithm selects Z with probability 1. Consider the case $Y > 2$. Again, if $r_1 \leq 2$, the algorithm selects Z with probability 1. Otherwise, if $r_1 > 2$, the algorithm returns Z if and only if for all realizations above τ that appear after X_{i_1} and are also larger than X_{i_1}, the first to encounter is Z. In other words, for the algorithm to succeed in this case, it must be that among the $r_1 - 1$ values of rank smaller than r_1, the first one in the arrival order is the element of rank 1. Since the random variables are IID, the probability of this event is exactly $1/r_1-1$.

Let j be the first index such that $X_{i_j} > X_{i_1}$, and $\alpha(Y) = \mathbb{P}[\mathcal{A} \text{ selects } Z \mid Y]$. Conditioned on $Y \geq 3$, the probability that the algorithm selects Z is

$$\begin{aligned}
\alpha(Y \mid Y \geq 3) &= \mathbb{P}[r_1 = 1] + \mathbb{P}[r_1 = 2] + \sum_{t=3}^{Y} \mathbb{P}[r_1 = t]\,\mathbb{P}[r_j = 1 \mid r_1 = t] \\
&= \frac{2}{Y} + \sum_{t=3}^{Y} \frac{\mathbb{P}[r_z = 1 \mid r_1 = t]}{Y} = \frac{1}{Y}\left(2 + \sum_{t=3}^{Y} \mathbb{P}[r_z = 1 \mid r_1 = t]\right) \\
&= \frac{1}{Y}\left(2 + \sum_{t=3}^{Y} \frac{1}{t-1}\right) = \frac{1}{Y}\left(1 + \sum_{t=1}^{Y-1} \frac{1}{t}\right) \\
&= \frac{1}{Y}(1 + H_{Y-1}),
\end{aligned}$$

where H_n denotes the nth harmonic number. Recall also that $\alpha(Y \mid Y = 1) = \alpha(Y \mid Y = 2) = 1$.

Next, we estimate $\mathbb{P}[Y = i]$, by approximating Y with a Poisson distribution via Le Cam's theorem. Let $\delta_i = \left|\binom{n}{i}p^i(1-p)^{n-i} - e^{-np}\frac{(np)^i}{i!}\right|$. The idea is to set p such that $np = q$, where $q \geq 1$ is a fixed constant. We know that $\mathbb{P}[Y = i] = \binom{n}{i}p^i(1-p)^{n-i}$, and thus, by Theorem 3.12, we have

$$\sum_{i=0}^{\infty}\delta_i = \sum_{i=0}^{\infty}\left|\mathbb{P}[Y = i] - e^{-np}\frac{(np)^i}{i!}\right| = \sum_{i=0}^{\infty}\left|\mathbb{P}[Y = i] - e^{-q}\frac{(q)^i}{i!}\right| \leq \frac{2qp}{\max\{1,q\}} \leq 2p = \frac{2q}{n}.$$

Overall, the probability that $\mathcal{A}$ selects Z is

$$\begin{aligned}
\alpha(Y) &= \sum_{i=0}^{n}\mathbb{P}[Y = i]\cdot\alpha(Y \mid Y = i) \\
&= \mathbb{P}[Y = 1] + \sum_{i=2}^{n}\mathbb{P}[Y = i]\cdot\alpha(Y \mid Y = i) \\
&\geq np(1-p)^{(n-1)} + \sum_{i=2}^{n}\left(e^{-q}\frac{q^i}{i!} - \delta_i\right)\cdot\alpha(Y \mid Y = i),
\end{aligned}$$

where the last inequality follows by the definition of δ_i. Thus,

$$\begin{aligned}
\alpha(Y) &= q(1-q/n)^{(n-1)} + \sum_{i=2}^{n} e^{-q}\frac{q^i}{i!}\cdot\alpha(Y \mid Y = i) - \sum_{i=2}^{n}\delta_i\cdot\alpha(Y \mid Y = i) \\
&\geq q(1-q/n)^{(n-1)} + \sum_{i=2}^{n} e^{-q}\frac{q^i}{i!}\frac{1 + H_{i-1}}{i} - \sum_{i=2}^{n}\delta_i \\
&\geq q(1-q/n)^{(n-1)} + e^{-q}\sum_{i=2}^{n}\frac{q^i(1 + H_{i-1})}{i!\cdot i} - \frac{2q}{n}. \qquad (4.1)
\end{aligned}$$

◀

It is easy to see that simply setting $q = 2$, which corresponds to $p = 2/n$ and τ being the $2/n$th quantile of $\mathcal{D}$, yields $\alpha(Y) > 0.5801$ for all $n \geq 20$. Thus, our simple single-threshold algorithm, augmented with a single oracle call, beats, even for small n, the optimal algorithm for the IID prophet inequality which uses different thresholds per distribution and achieves a probability of success approximately 0.5801 [12].

Since the worst-case probability of ≈ 0.5801 by [12] is achieved for $n \to \infty$, one might be interested in the asymptotic behavior of the probability of our algorithm, $\alpha(Y)$, for large n. It is not too difficult to see after some calculations that, as $n \to \infty$, Eq. (4.1) is maximized for $q \approx 2.435$, yielding $\alpha(Y) \approx 0.798$.

4.2 A single-threshold algorithm for general m

As we saw in the previous section, even for a simple, single-threshold algorithm, the analysis of the winning probability gets tedious quickly. In this section, we generalize our single-threshold algorithm to the case of general m, and use the fact that the maximum of a uniformly random permutation of n values changes $\mathcal{O}(\log n)$ times with high probability to obtain a guarantee on the winning probability that is super-exponential with respect to m.

As before, our algorithm selects a threshold τ such that $p = \mathbb{P}[X \geq \tau]$ and every time the algorithm observes a realization above τ, it uses an oracle query and asks $\mathcal{O}$ if the realization should be selected or not. If not, then it updates the threshold to the new higher value. If the algorithm runs out of oracle calls, then it selects the first element above the current threshold τ that is encounters, if any. In other words, the algorithm uses the oracle calls greedily for all realizations above τ.

▶ **Theorem 1.9** (see [13] for proof). *For sufficiently large m, n, there exists an algorithm for the $\mathcal{M}(\mathcal{O}_m, \mathit{IID}, \mathbb{P}_{\max})$ model that selects the maximum realization with probability at least $1 - \mathcal{O}\big(m^{-m/5}\big)$.*

4.3 An (almost) tight upper bound

Given that we have a simple, single-threshold algorithm for the $\mathcal{M}(\mathcal{O}_m, \mathsf{IID}, \mathbb{P}_{\max})$ setting, a reasonable question to ask is how far it is from being optimal. As we show in this section, the algorithm is asymptotically almost optimal.

▶ **Theorem 1.10** (see [13] for proof). *There exists an instance of $\mathcal{M}(\mathcal{O}_m, \mathit{IID}, \mathbb{P}_{\max})$ for which no algorithm can select the maximum realization with probability greater than $1 - \mathcal{O}(m^{-m})$.*

5 Conclusion

In this work, we improved on the known results for the Top-1-of-m model, for both the RoE and $\mathbb{P}_{\max}$ objectives, via the lens of a simple prophet inequality augmented with oracle calls. All our results hold with respect to the strongest possible adversary, the almighty adversary. A weaker, offline, adversary is forced to select the order of distributions upfront, given only access to the same information as the algorithm. For such an adversary, one can do very slightly better than Theorem 1.8 – see the full version [13] for more details.

Our oracle choice was motivated by our efforts to reformulate the Top-1-of-m model in order to improve upon the current known bounds. We mention a few other oracle models that are interesting and could potentially be useful in analyzing other prophet inequality settings: (i) the oracle can predict a *range* for the maximum value, but formalizing this in a more general setting turns out to be difficult without assuming something about the

support of each random variable, (*ii*) the algorithm can ask the oracle if there is a value that is greater than $c \cdot X_i$, for some constant c. This latter oracle is more powerful, as it doesn't exhibit the same limitations that our oracle model has in the example of Theorem 1.8. We leave exploring more complex oracle models for future work.

Finally, there are subtle differences between an oracle that answers queries of the form $X_i > \max\{X_{i+1}, \ldots, X_n\}$ and one that answers queries of the form $X_i \geq \max\{X_{i+1}, \ldots, X_n\}$; in particular, the $>$ oracle is weaker than the $\geq$ oracle. We refer the reader to the full version [13] for a discussion on this topic.

References

1 Melika Abolhassani, Soheil Ehsani, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Robert D. Kleinberg, and Brendan Lucier. Beating 1-1/e for ordered prophets. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 61–71. ACM, 2017. doi:10.1145/3055399.3055479.

2 David Assaf, Larry Goldstein, and Ester Samuel-Cahn. Ratio prophet inequalities when the mortal has several choices. *The Annals of Applied Probability*, 12(3):972–984, 2002.

3 David Assaf and Ester Samuel-Cahn. Simple ratio prophet inequalities for a mortal with multiple choices. *Journal of Applied Probability*, 37(4):1084–1091, 2000. URL: http://www.jstor.org/stable/3215496.

4 Siddhartha Banerjee, Vincent Cohen-Addad, Anupam Gupta, and Zhouzi Li. Graph searching with predictions, 2022. doi:10.48550/arXiv.2212.14220.

5 Lucien Le Cam. An approximation theorem for the poisson binomial distribution. *Pacific Journal of Mathematics*, 10:1181–1197, 1960.

6 Jose Correa, Andres Cristi, Paul Duetting, and Ashkan Norouzi-Fard. Fairness and bias in online selection. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2112–2121. PMLR, July 2021. URL: https://proceedings.mlr.press/v139/correa21a.html.

7 José R. Correa, Patricio Foncea, Ruben Hoeksma, Tim Oosterwijk, and Tjark Vredeveld. Posted price mechanisms and optimal threshold strategies for random arrivals. *Math. Oper. Res.*, 46(4):1452–1478, 2021. doi:10.1287/moor.2020.1105.

8 Frank Den Hollander. Probability theory: The coupling method, 2012. Notes available online: https://mathematicaster.org/teaching/lcs22/hollander_coupling.pdf. URL: https://mathematicaster.org/teaching/lcs22/hollander_coupling.pdf.

9 Hossein Esfandiari, MohammadTaghi Hajiaghayi, Vahid Liaghat, and Morteza Monemizadeh. Prophet secretary. *SIAM Journal on Discrete Mathematics*, 31(3):1685–1701, 2017. doi:10.1137/15M1029394.

10 Tomer Ezra, Michal Feldman, Nick Gravin, and Zhihao Gavin Tang. "who is next in line?" on the significance of knowing the arrival order in bayesian online settings. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3759–3776. SIAM, 2023. doi:10.1137/1.9781611977554.ch145.

11 Tomer Ezra, Michal Feldman, and Ilan Nehama. Prophets and secretaries with overbooking. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, EC '18, pages 319–320. Association for Computing Machinery, 2018. doi:10.1145/3219166.3219211.

12 John P. Gilbert and Frederick Mosteller. Recognizing the maximum of a sequence. *Journal of the American Statistical Association*, 61(313):35–73, 1966. URL: http://www.jstor.org/stable/2283044.

13 Sariel Har-Peled, Elfarouk Harb, and Vasilis Livanos. Oracle-Augmented Prophet Inequalities. *arXiv e-prints*, April 2024. doi:10.48550/arXiv.2404.11853.

14 Elfarouk Harb. New prophet inequalities via poissonization and sharding, 2024. arXiv:2307.00971.

15 T. P. Hill and Robert P. Kertz. Comparisons of stop rule and supremum expectations of i.i.d. random variables. *Ann. Probab.*, 10(2):336–345, May 1982. doi:10.1214/aop/1176993861.

16 Robert P Kertz. Stop rule and supremum expectations of i.i.d. random variables: A complete comparison by conjugate duality. *Journal of Multivariate Analysis*, 19(1):88–112, 1986. doi:10.1016/0047-259X(86)90095-3.

17 Robert Kleinberg and S. Matthew Weinberg. Matroid prophet inequalities and applications to multi-dimensional mechanism design. *Games Econ. Behav.*, 113:97–115, 2019. doi:10.1016/j.geb.2014.11.002.

18 Ulrich Krengel and Louis Sucheston. Semiamarts and finite values. *Bull. Amer. Math. Soc.*, 83(4):745–747, July 1977. URL: https://projecteuclid.org:443/euclid.bams/1183538915.

19 Ulrich Krengel and Louis Sucheston. On semiamarts, amarts, and processes with finite value. *Probability on Banach spaces*, 4:197–266, 1978.

20 Pranav Nuti. The secretary problem with distributions. In *Integer Programming and Combinatorial Optimization: 23rd International Conference, IPCO 2022, Eindhoven, The Netherlands, June 27-29, 2022, Proceedings*, pages 429–439. Springer-Verlag, 2022. doi:10.1007/978-3-031-06901-7_32.

21 Ester Samuel-Cahn. Comparison of threshold stop rules and maximum for independent nonnegative random variables. *The Annals of Probability*, 12(4):1213–1216, 1984. URL: http://www.jstor.org/stable/2243359.

Refuting Approaches to the Log-Rank Conjecture for XOR Functions

Hamed Hatami
School of Computer Science, McGill University, Montreal, Canada

Kaave Hosseini
Department of Computer Science, University of Rochester, NY, USA

Shachar Lovett
Department of Computer Science and Engineering,
University of California at San Diego, La Jolla, CA, USA

Anthony Ostuni
Department of Computer Science and Engineering,
University of California at San Diego, La Jolla, CA, USA

Abstract

The log-rank conjecture, a longstanding problem in communication complexity, has persistently eluded resolution for decades. Consequently, some recent efforts have focused on potential approaches for establishing the conjecture in the special case of XOR functions, where the communication matrix is lifted from a boolean function, and the rank of the matrix equals the Fourier sparsity of the function, which is the number of its nonzero Fourier coefficients.

In this note, we refute two conjectures. The first has origins in Montanaro and Osborne (arXiv'09) and is considered in Tsang, Wong, Xie, and Zhang (FOCS'13), and the second is due to Mande and Sanyal (FSTTCS'20). These conjectures were proposed in order to improve the best-known bound of Lovett (STOC'14) regarding the log-rank conjecture in the special case of XOR functions. Both conjectures speculate that the set of nonzero Fourier coefficients of the boolean function has some strong additive structure. We refute these conjectures by constructing two specific boolean functions tailored to each.

2012 ACM Subject Classification Theory of computation → Communication complexity

Keywords and phrases Communication complexity, log-rank conjecture, XOR functions, additive structure

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.82

Category Track A: Algorithms, Complexity and Games

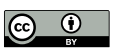
Related Version *Full Version*: `https://arxiv.org/abs/2312.09400`

Funding *Shachar Lovett*: Research supported by NSF awards 1953928 and 2006443, and a Simons investigator award.
Anthony Ostuni: Research supported by NSF award 2006443.

Acknowledgements This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing. A.O. thanks Daniel M. Kane for a number of helpful discussions. We also thank anonymous reviewers for useful comments on earlier versions of this manuscript.

1 Introduction

The study of communication complexity seeks to determine the inherent amount of communication between multiple parties required to complete a computational task. Arguably, the most outstanding conjecture in the field is the *log-rank conjecture* of Lovász and Saks [4]. They suggest that the (deterministic) communication complexity of a two-party boolean function is upper bounded by the matrix rank over $\mathbb{R}$. More precisely,



▶ **Conjecture 1** (Log-rank conjecture [4]). *Let $f : X \times Y \to \{-1, 1\}$ be an arbitrary two-party boolean function. Then,*

$$\mathrm{CC}(f) \leq \mathrm{polylog}(\mathrm{rank}(f)),$$

where $\mathrm{CC}(f)$ is the communication complexity of f and $\mathrm{rank}(f)$ is the rank over $\mathbb{R}$ of the corresponding boolean matrix.

It is well-known that $\log(\mathrm{rank}(f)) \leq \mathrm{CC}(f)$ [7], so a positive resolution to Conjecture 1 would imply that the communication complexity of two-party boolean functions is determined by rank, up to polynomial factors.

To date, the best known bound is still exponentially far from that in Conjecture 1. Concretely, Lovett [5] showed that $\mathrm{CC}(f) \leq O(\sqrt{\mathrm{rank}(f)} \log \mathrm{rank}(f))$. Very recently, Sudakov and Tomon posted a preprint improving the bound to $O(\sqrt{\mathrm{rank}(f)})$ [13]. In hopes of gaining further insight, many researchers have considered the special case of *XOR functions*, where $f_\oplus(x, y) \coloneqq f(x + y)$ for a boolean function $f : \mathbb{F}_2^n \to \{-1, 1\}$ [8, 15, 14, 12, 2, 6].

The XOR setting has several convenient properties. For example, the eigenvalues of $f_\oplus$ correspond to the Fourier coefficients of f. Thus, $\mathrm{rank}(f_\oplus) = |\mathrm{supp}(\widehat{f})|$, the number of nonzero coefficients in f's Fourier expansion (also known as the *Fourier sparsity*). Additionally, Hatami, Hosseini, and Lovett [2] proved a polynomial equivalence between $\mathrm{CC}(f_\oplus)$ and the *parity decision tree* complexity of f, denoted $\mathrm{PDT}(f)$. Parity decision trees are defined similarly to standard decision trees, with the extra power that each node can query an arbitrary parity of input bits. These facts together imply that the log-rank conjecture for XOR functions can be restated as follows:

▶ **Conjecture 2** (XOR log-rank conjecture). *Let $f : \mathbb{F}_2^n \to \{-1, 1\}$. Then,*

$$\mathrm{PDT}(f) \leq \mathrm{polylog}(|\mathrm{supp}(\widehat{f})|).$$

The best known bound, due to [14, 12], is $\mathrm{PDT}(f) \leq O(\sqrt{|\mathrm{supp}(\widehat{f})|})$, a mere log-factor improvement on the general case bound by Lovett [5], and matched by the recent bound of Sudakov and Tomon [13].

1.1 Folding

Folding is a fundamental concept in the analysis of the additive structure of a function's Fourier support. Let

$$\mathcal{S} = \mathrm{supp}(\widehat{f}) = \{\gamma \in \mathbb{F}_2^n : \widehat{f}(\gamma) \neq 0\} \quad \text{and} \quad \mathcal{S} + \gamma = \{s + \gamma : s \in \mathcal{S}\}.$$

If $(s_1, s_2), (s_3, s_4) \in \binom{\mathcal{S}}{2}$ satisfy $s_1 + s_2 = s_3 + s_4 = \gamma$, we say the pairs (s_1, s_2) and (s_3, s_4) *fold* in the direction γ.

Analyzing folding directions is useful in constructing efficient PDTs in the context of Conjecture 2. In particular, when a function f is restricted according to the result of some parity query γ, all pairs of elements in $\mathcal{S}$ that fold in the direction γ collapse to a single term in the restricted function $f|_\gamma$'s Fourier support. Iterating this process until the restricted function is constant yields a PDT whose depth depends on the number of iterations performed and, thus, on the size of the folding directions queried. Indeed, this is the general strategy used to prove the aforementioned closest result to Conjecture 2 [14, 12].

1.2 Refuting a greedy approach

An approach dating back to [8] seeks to prove Conjecture 2 through the existence of a single large folding direction. They conjectured that there always exists γ_1, γ_2 such that $|(\mathcal{S}+\gamma_1)\cap(\mathcal{S}+\gamma_2)| \geq |\mathcal{S}|/K$ for some constant $K > 1$. This yields the following $O(\log|\mathcal{S}|)$-rounds greedy approach: query $\gamma_1+\gamma_2$ and consider the function restricted to the query response. This restriction decreases the Fourier sparsity by a constant factor, so the function must become constant in $O(\log|\mathcal{S}|)$ rounds. This implies the strong upper bound of

$$\mathrm{PDT}(f) \leq O\left(\log|\mathcal{S}|\right).$$

However, O'Donnell, Wright, Zhao, Sun, and Tan [10] constructed a function with communication complexity $\Omega(\log(|\mathcal{S}|)^{\log_3(6)})$; hence one can not take K to be a constant. Yet to prove the log-rank conjecture, it suffices to take $K = O(\mathrm{polylog}(|\mathcal{S}|))$, and this choice of K remained plausible up to date. Such an approach is mentioned in both [14] and [6], and a similar approach was used to verify the log-rank conjecture for many cases of functions lifted with AND (rather than XOR) gadgets [3]. We strongly refute this conjecture.

▶ **Theorem 3** (Informal version of Theorem 8). *For infinitely many n, there is a function $f:\mathbb{F}_2^n \to \{-1,1\}$ such that for $\mathcal{S} = \mathrm{supp}(\widehat{f})$, it holds*

$$|(\mathcal{S}+\gamma_1)\cap(\mathcal{S}+\gamma_2)| \leq O\left(|\mathcal{S}|^{5/6}\right)$$

for all distinct $\gamma_1, \gamma_2 \in \mathbb{F}_2^n$.

▶ Remark 4. Observe that this theorem implies the greedy method cannot obtain a bound better than $\mathrm{PDT}(f) = \widetilde{O}(|\mathcal{S}|^{1/6})$. In fact, a more careful analysis can rule out bounds better than $\mathrm{PDT}(f) = \widetilde{O}(|\mathcal{S}|^{1/5})$ (see Remark 15).

The functions used in Theorem 3 are a variant of the addressing function using disjoint (affine) subspaces. While we believe the specific construction is novel, the concept of using functions defined with disjoint subspaces has previously appeared in the literature in this context. Most notably, Chattopadhyay, Garg, and Sherif used XOR functions based on this idea in the pursuit of stronger counterexamples to a more general version of the log-rank conjecture [1].

1.3 Refuting a randomized approach

Rather than simply looking for a large folding direction, a recent work of Mande and Sanyal [6] attempts to address Conjecture 2 through a deeper understanding of the additive structure of the spectrum of boolean functions. They proposed the following conjecture on the number of nontrivial folding directions, and showed it would yield a polynomial improvement to the state-of-the-art upper bound for the XOR log-rank conjecture via a randomized approach.

▶ **Conjecture 5** ([6]). *There are constants $\alpha, \beta \in (0,1)$ such that for every boolean function $f:\mathbb{F}_2^n \to \{-1,1\}$, for $\mathcal{S} = \mathrm{supp}(\widehat{f})$, it holds*

$$\Pr_{\gamma_1,\gamma_2\in\mathcal{S}}\left[|(\mathcal{S}+\gamma_1)\cap(\mathcal{S}+\gamma_2)| > |\mathcal{S}|^{\beta}\right] \geq \alpha.$$

In fact, Mande and Sanyal conjectured that one can take $\beta = \frac{1}{2} - o(1)$. The conjecture might seem plausible given the numerous results on the additive structure of the spectrum of boolean functions. However, we strongly refute it, as well:

▶ **Theorem 6** (Informal version of Theorem 16). *For infinitely many n, there is a function $f : \mathbb{F}_2^n \to \{-1, 1\}$ such that for $\mathcal{S} = \mathrm{supp}(\widehat{f})$, it holds*

$$\Pr_{\gamma_1,\gamma_2 \in \mathcal{S}}[|(\mathcal{S} + \gamma_1) \cap (\mathcal{S} + \gamma_2)| > k] = O(1/k) \qquad \forall k \geq 1.$$

Overview

Some preliminary material is reviewed in Section 2. We prove more precise versions of Theorem 3 in Section 3 and Theorem 6 in Section 4. Section 5 contains some final thoughts.

2 Preliminaries

Communication complexity

Let $f : X \times Y \to \{-1, 1\}$ be an arbitrary function. Additionally, assume two parties are given an element $x \in X$ and $y \in Y$, respectively, which the other party cannot see. The (deterministic) *communication complexity of* f, denoted $\mathrm{CC}(f)$, is the minimum number of bits over all assignments (x, y) needed to be exchanged in order to evaluate f, where the parties may decide on a strategy prior to receiving their inputs.

One can view such a function as an $X \times Y$ matrix, where the (x, y) entry takes the value $f(x, y)$. Thus, it is natural to consider the relationship between linear algebraic measures, such as matrix rank, and communication complexity, as in Conjecture 1. For a more thorough treatment of communication complexity, see the excellent book [11].

Decision trees

Decision trees are simple models of computation. The (deterministic) decision tree *depth* of a function $f : \mathbb{F}_2^n \to \{-1, 1\}$ is the maximum over all inputs $x \in \mathbb{F}_2^n$ of the fewest number of input bits one must query to correctly evaluate $f(x)$.

Parity decision trees (PDTs) extend the power of "traditional" decision trees by allowing queries to return the sum modulo two of an arbitrary subset of the bits. They are particularly relevant in the study of communication complexity, since for functions of the form $f_{\oplus}(x, y) = f(x + y)$ for $f : \mathbb{F}_2^n \to \{-1, 1\}$, the parity decision tree depth and communication complexity are equivalent (up to polynomial factors) [2].

Boolean analysis

Every function $f : \mathbb{F}_2^n \to \mathbb{R}$ has a unique *Fourier expansion*

$$f = \sum_{\alpha \in \mathbb{F}_2^n} \widehat{f}(\alpha)\chi_\alpha,$$

where

$$\chi_\alpha(x) = (-1)^{\langle x,\alpha \rangle} \quad \text{and} \quad \widehat{f}(\alpha) = \langle f, \chi_\alpha \rangle = \mathbb{E}_{x \in \mathbb{F}_2^n}[f(x)\chi_\alpha(x)].$$

The set $\mathrm{supp}(\widehat{f}) = \{\alpha \in \mathbb{F}_2^n : \widehat{f}(\alpha) \neq 0\}$ is the *Fourier support*, occasionally denoted $\mathcal{S}$. Its size $|\mathrm{supp}(\widehat{f})|$ is the *Fourier sparsity*. In light of Conjecture 2, we are primarily interested in the relationship between a function's Fourier sparsity and parity decision tree depth.

In general, a vast array of information about a function can be learned from its Fourier expansion, and we direct readers to the standard text [9] for additional background. For our purposes, we will only require the following simple fact. Let $V^\perp = \{w : \langle w, v \rangle = 0 \text{ for all } v \in V\}$ be the orthogonal complement of a subspace V.

▶ **Proposition 7** (See e.g., [9, Proposition 3.12]). *If $A = V + v \subseteq \mathbb{F}_2^n$ is an affine subspace of codimension k, then*

$$\mathbb{1}_A = \sum_{\alpha \in V^\perp} 2^{-k} \chi_\alpha(v) \chi_\alpha.$$

3 One excellent folding direction

A large folding direction implies the existence of a parity query whose answer substantially simplifies the resulting restricted function. This suggests the following greedy approach to resolve the XOR log-rank conjecture: if we can always find distinct γ_1, γ_2 such that $|(\mathcal{S} + \gamma_1) \cap (S + \gamma_2)| \geq \Omega(|\mathcal{S}| \,/\, \mathrm{polylog}(|\mathcal{S}|))$, then querying $\gamma_1 + \gamma_2$ and recursing on the appropriate restriction of f will force f to be constant in $\mathrm{polylog}(|\mathcal{S}|)$ rounds.

We refute this strategy by proving a precise version of Theorem 3.

▶ **Theorem 8.** *For $n = 2^k + 7k$ with $k \in \mathbb{N}^{\geq 3}$, there is a function $f : \mathbb{F}_2^n \to \{-1, 1\}$ such that for $\mathcal{S} = \mathrm{supp}(\widehat{f})$, it holds $|\mathcal{S}| \geq 2^{6k}$, and yet $|(\mathcal{S} + \gamma_1) \cap (\mathcal{S} + \gamma_2)| \leq 2^{5k+4}$ for all distinct $\gamma_1, \gamma_2 \in \mathbb{F}_2^n$.*

To build intuition for our construction, we first consider the standard addressing function.

▶ **Example 9** (Addressing). Define $f : \mathbb{F}_2^{k+2^k} \to \{-1, 1\}$ by

$$f(x, y) = (-1)^{y_x} = \sum_{z \in \mathbb{F}_2^k} \mathbb{1}_z(x) \cdot (-1)^{y_z},$$

where $x \in \mathbb{F}_2^k$ and $y \in \mathbb{F}_2^{2^k}$ (and slightly abusing notation by indexing y with vectors).

A greedy approach is sufficient for a PDT to evaluate this function. Simply query each address bit, then the corresponding addressed bit. Each query eliminates half of the remaining possible address values, so the PDT has depth $k + 1$, while the function's sparsity is exponential in k. To modify the function to prevent this approach, we encode the address using subspaces to obfuscate it while maintaining Fourier sparsity.

▶ **Example 10** (Subspace addressing). Let $A_1, \ldots, A_{2^k} \subset \mathbb{F}_2^{7k}$ be disjoint affine subspaces of dimension $2k$. Define $f : \mathbb{F}_2^{7k+2^k} \to \{-1, 1\}$ by

$$f(x, y) = \begin{cases} (-1)^{y_i} & x \in A_i \\ 1 & x \notin A_1 \cup \cdots \cup A_{2^k} \end{cases},$$

where $x \in \mathbb{F}_2^{7k}$ and $y \in \mathbb{F}_2^{2^k}$.

We choose A_i's randomly and show that the resulting function f has the suitable properties we need with high probability.

▶ **Lemma 11.** *Suppose the random function f is constructed by picking random affine subspaces $A_1, \cdots, A_{2^k} \subset \mathbb{F}_2^{7k}$ as follows: for each $i \in [2^k]$, choose vectors $a_i, v_i^1, \cdots, v_i^{2k} \in \mathbb{F}_2^{7k}$ uniformly and independently, and let $V_i = \langle v_i^1, \cdots, v_i^{2k} \rangle$ and $A_i = V_i + a_i$. Then with probability $1 - 2^{-k+2}$, all of the following hold:*

(a) *$\forall i$, $\dim(V_i) = 2k$.*
(b) *$\forall i \neq j$, $A_i \cap A_j = \emptyset$.*
(c) *$\forall i \neq j$, $V_i \cap V_j = \{0\}$.*
(d) *For all nonzero $v \in \mathbb{F}_2^{7k}$, $|\{i : v \in V_i^\perp\}| \leq 7$.*

Proof. For brevity, let $m = 7k$.

(a) Fix $i \in [2^k]$. The probability that vectors $v_i^1, \cdots, v_i^{2k}$ are linearly independent is at least

$$\frac{2^m-1}{2^m} \cdot \frac{2^m-2}{2^m} \cdot \frac{2^m-2^2}{2^m} \cdot \cdots \cdot \frac{2^m-2^{2k-1}}{2^m} \geq (1-2^{2k-m})^m \geq 1 - m2^{2k-m}.$$

Hence the probability that there is $i \in [2^k]$ for which $v_i^1, \cdots, v_i^{2k}$ are not linearly independent is at most $m2^{3k-m} = 7k2^{-4k} \leq 2^{-k}$.

(b) Fix $i \neq j$. The probability that $A_i \cap A_j \neq \emptyset$ is at most $2^{2k}2^{2k-m} = 2^{4k-m}$. Hence, the probability that there are $i \neq j$ with $A_i \cap A_j \neq \emptyset$ is at most $2^{2k}2^{4k-m} = 2^{-k}$.

(c) Fix $i \neq j$. The probability that $V_i \cap V_j \neq \{0\}$ is at most $(2^{2k}-1)2^{2k-m} \leq 2^{4k-m}$. Hence, the probability that there are $i \neq j$ with $V_i \cap V_j \neq \emptyset$ is at most $2^{2k}2^{4k-m} = 2^{-k}$.

(d) The probability that a fixed nonzero vector $v \in \mathbb{F}_2^{7k}$ is orthogonal to at least t subspaces among $V_1, \cdots, V_{2^k}$ is at most $\binom{2^k}{t}2^{-2tk} \leq 2^{-tk}$. Taking $t = 8$ and union bounding over all $2^{7k}-1$ options for v shows that the probability that there is v for which $|\{i : v \in V_i^\perp\}| \geq 8$ is at most 2^{-k}.

By the union bound, the probability that any of items (a) to (d) are not satisfied is at most $4 \cdot 2^{-k} = 2^{-k+2}$. ◀

We will assume from now on that f is chosen randomly so that Lemma 11 holds, and set $\mathcal{S} = \mathrm{supp}(\widehat{f})$. It remains to prove there is no large folding direction. First, we give a lower bound on the size of Fourier support of f.

▷ Claim 12. $|\mathcal{S}| \geq 2^{6k}$.

Proof. We can express f as

$$\begin{aligned} f(x,y) &= \mathbb{1}_{(A_1 \cup \cdots \cup A_{2^k})^c}(x) + \sum_{i=1}^{2^k} \mathbb{1}_{A_i}(x) \cdot (-1)^{y_i} \\ &= 1 - \sum_{i=1}^{2^k} \mathbb{1}_{A_i}(x) + \sum_{i=1}^{2^k} \mathbb{1}_{A_i}(x) \cdot (-1)^{y_i} \\ &= 1 + \sum_{i=1}^{2^k} \mathbb{1}_{A_i}(x) \cdot ((-1)^{y_i} - 1). \end{aligned}$$

By Proposition 7, the Fourier support of the function $\mathbb{1}_{A_i}(x)$ is $V_i^\perp \subset \mathbb{F}_2^{7k}$, and of $\mathbb{1}_{A_i}(x) \cdot (-1)^{y_i}$ is $V_i^\perp + e_i$, where e_i is the i-th basis vector in the standard basis for $\mathbb{F}_2^{2^k}$ embedded in the space $\mathbb{F}_2^{7k} \times \mathbb{F}_2^{2^k}$. Since the affine subspaces $V_i^\perp + e_i$ are disjoint and also $\left(V_i^\perp + e_i\right) \cap \left(V_i^\perp\right) = \emptyset$ the coefficients of characters in $V_i^\perp + e_i$ will not be canceled. Hence, we get that

$$\bigcup_{i=1}^{2^k} \left(V_i^\perp + e_i\right) \subset \mathcal{S}$$

and so $|\mathcal{S}| \geq 2^k \cdot 2^{7k-2k} = 2^{6k}$. ◁

We also need the following claim.

▷ Claim 13. Suppose $W_1, W_2 \subset \mathbb{F}_2^n$ are two linear subspaces such that $W_1 \cap W_2 = \{0\}$. Then for all $x \in \mathbb{F}_2^n$,

$$|W_1^\perp \cap (W_2^\perp + x)| = 2^{n - \dim W_1 - \dim W_2}.$$

Proof. Suppose $\dim(W_1) = d_1$ and $\dim(W_2) = d_2$. Without loss of generality, assume that $W_1 = \mathbb{F}_2^{d_1} \times 0^{d_2} \times 0^{n-d_1-d_2}$ and $W_2 = 0^{d_1} \times \mathbb{F}_2^{d_2} \times 0^{n-d_1-d_2}$. Note that $W_1^\perp = 0^{d_1} \times \mathbb{F}_2^{d_2} \times \mathbb{F}_2^{n-d_1-d_2}$ and $W_2^\perp = \mathbb{F}_2^{d_1} \times 0^{d_2} \times \mathbb{F}_2^{n-d_1-d_2}$. Pick an arbitrary $x = (x_1, x_2, x_3) \in \mathbb{F}_2^{d_1} \times \mathbb{F}_2^{d_2} \times \mathbb{F}_2^{n-d_1-d_2}$. Then $W_2^\perp + (x_1, x_2, x_3) = \mathbb{F}_2^{d_1} \times \{x_2\} \times \mathbb{F}_2^{n-d_1-d_2}$ and $W_1^\perp \cap (W_2^\perp + x) = 0^{d_1} \times \{x_2\} \times \mathbb{F}_2^{n-d_1-d_2}$ has the claimed size. ◁

Finally, Theorem 8 follows from claim below.

▷ Claim 14. For all distinct $\gamma_1, \gamma_2 \in \mathbb{F}_2^{7k+2^k}$, we have

$$|(\mathcal{S} + \gamma_1) \cap (\mathcal{S} + \gamma_2)| \leq 2^{5k+4}.$$

Proof. First, note that it suffices to prove the claim for all distinct $\gamma_1, \gamma_2 \in \mathcal{S}$, since if $s_1 + \gamma_1 = s_2 + \gamma_2$ for $s_1, s_2 \in \mathcal{S}$, it must be that $\gamma_1 + \gamma_2 = s_1 + s_2 \in \mathcal{S} + \mathcal{S}$. Pick an arbitrary non-zero $\gamma = \gamma_1 + \gamma_2$ for $\gamma_1, \gamma_2 \in \mathcal{S}$. Remember that

$$|(\mathcal{S} + \gamma_1) \cap (\mathcal{S} + \gamma_2)| = |\mathcal{S} \cap (\mathcal{S} + \gamma)| \quad \text{and} \quad \mathcal{S} \subseteq \left(\bigcup_{i=1}^{2^k} V_i^\perp \right) \cup \left(\bigcup_{i=1}^{2^k} (V_i^\perp + e_i) \right).$$

Hence $\mathcal{S} \cap (\mathcal{S} + \gamma) \subseteq A \cup B \cup C$, where

$$A = \bigcup_{i,j} \left(V_i^\perp \cap (V_j^\perp + \gamma) \right)$$
$$B = \bigcup_{i,j} \left(V_i^\perp \cap (V_j^\perp + e_j + \gamma) \right)$$
$$C = \bigcup_{i,j} \left((V_i^\perp + e_i) \cap (V_j^\perp + e_j + \gamma) \right).$$

Let $|\cdot|$ denote the Hamming weight of a vector. Decompose $\gamma = (\gamma_x, \gamma_y)$ where $\gamma_x \in \mathbb{F}_2^{7k}$ and $\gamma_y \in \mathbb{F}_2^{2^k}$. Observe that $|\gamma_y| \leq 2$, since (as noted above) we may assume $\gamma \in \mathcal{S} + \mathcal{S}$ without loss of generality.

Case 1: $|\gamma_y| = 0$.

Note that in this case $B = \emptyset$ and $C = \bigcup_i \left((V_i^\perp + e_i) \cap (V_i^\perp + e_i + \gamma_x) \right)$. Overall, we get

$$\begin{aligned}
|\mathcal{S} \cap (\mathcal{S} + \gamma_x)| &\leq \left| \bigcup_{i,j} \left(V_i^\perp \cap (V_j^\perp + \gamma_x) \right) \right| + \left| \bigcup_i \left((V_i^\perp + e_i) \cap (V_i^\perp + e_i + \gamma_x) \right) \right| \\
&= \left| \bigcup_{i,j} \left(V_i^\perp \cap (V_j^\perp + \gamma_x) \right) \right| + \left| \bigcup_i \left(V_i^\perp \cap (V_i^\perp + \gamma_x) \right) \right| \\
&\leq \sum_{i \neq j} |V_i^\perp \cap (V_j^\perp + \gamma_x)| + 2 \sum_i |V_i^\perp \cap (V_i^\perp + \gamma_x)| \\
&\leq \sum_{i \neq j} |V_i^\perp \cap (V_j^\perp + \gamma_x)| + 2 \cdot 2^{5k} \cdot |\{i : \gamma_x \in V_i^\perp\}|.
\end{aligned}$$

To bound the first term, note that $V_i \cap V_j = \{0\}$ for all $i \neq j$ (by item (c) of Lemma 11). Using Claim 13 we have that $|V_i^\perp \cap (V_j^\perp + \gamma_x)| = 2^{7k - \dim(V_i) - \dim(V_j)} = 2^{7k-2k-2k} = 2^{3k}$. To bound the second term, by item (d) of Lemma 11, we have that $|\{i : \gamma_x \in V_i^\perp\}| \leq 7$. Overall, we get that

$$|\mathcal{S} \cap (\mathcal{S} + \gamma)| \leq 2^{2k} \cdot 2^{3k} + 7 \cdot 2^{5k+1} \leq 2^{5k+4}.$$

Case 2: $|\gamma_y| = 1$. Suppose $\gamma_y = e_i$ for some i.

In this case, $A = C = \emptyset$ and $B = V_i^\perp \cap (V_i^\perp + e_i + \gamma_y)$. Hence,

$$|\mathcal{S} \cap (\mathcal{S} + \gamma)| \leq |V_i^\perp \cap (V_i^\perp + e_i + \gamma_y)| \leq |V_i^\perp| = 2^{5k},$$

Case 3: $|\gamma_y| = 2$. This is similar to Case 2. ◀

▶ Remark 15. We have chosen parameters for simplicity of exhibition; however, by choosing the original disjoint affine subspaces from $\mathbb{F}_2^{(6+\varepsilon)k}$ rather than $\mathbb{F}_2^{7k}$, a similar analysis rules out any bounds stronger than $\mathrm{PDT}(f) = \widetilde{O}(|\mathcal{S}|^{1/5})$ resulting from this greedy method.

4 Many good folding directions

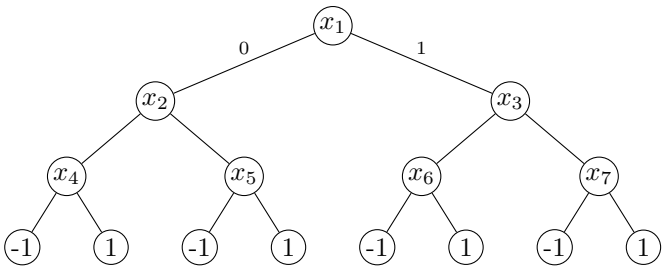
Rather than hoping for one large folding direction, [6] sought many nontrivial ones. In this section, we refute their conjecture (Conjecture 5) with the following quantified version of Theorem 6.

▶ **Theorem 16.** *For $n = 2^d - 1$ with $d \in \mathbb{N}$, there is a function $f : \mathbb{F}_2^n \to \{-1, 1\}$ such that for $\mathcal{S} = \mathrm{supp}(\widehat{f})$, it holds*

$$\Pr_{\gamma_1, \gamma_2 \in \mathcal{S}} \left[|(\mathcal{S} + \gamma_1) \cap (\mathcal{S} + \gamma_2)| \geq 2^{k+2} \right] \leq 2^{-k} + 2^{1-d} \qquad \forall k \geq 1.$$

In our construction, $|\mathcal{S}| = \mathrm{poly}(n)$, which is the primary regime of interest. For larger $\mathcal{S}$, say of size $|\mathcal{S}| = \exp(n^c)$ for some constant $c > 0$, the log-rank conjecture is trivially true, since $n < \mathrm{polylog}(|\mathcal{S}|)$.

Let T be a full binary decision tree of depth d. There are $n = 2^d - 1$ internal nodes indexed by $[2^d - 1]$, where we query (distinct) x_i at node i. Each of the largest depth internal nodes v is adjacent to two leaves: -1 and 1, corresponding to $v = 0$ and $v = 1$, respectively. Let $f : \mathbb{F}_2^n \to \{-1, 1\}$ be the resulting function. For example, the following decision tree corresponds to f for $n = 7$.



As we will soon show, the Fourier support of f corresponds to (subsets of) paths down the tree, where $|(\mathcal{S} + \gamma_1) \cap (\mathcal{S} + \gamma_2)|$ is determined by the lowest common ancestor of the paths of γ_1 and γ_2. Since it is overwhelmingly likely the two paths will quickly diverge, we find $|(\mathcal{S} + \gamma_1) \cap (\mathcal{S} + \gamma_2)|$ is typically small.

Suppose the leaves are indexed by $[2^d]$. Then f can be written as

$$\sum_{i \in [2^d]} \mathrm{sign}(L_i) \cdot \mathbb{1}_{L_i}, \tag{1}$$

where $\mathbb{1}_{L_i}$ denotes the indicator function of the inputs that result in leaf i, and $\mathrm{sign}(L_i) \in \{-1, 1\}$ is the output at leaf i. Let P_i be the ordered set of coordinates that are queried to reach the leaf i. Then for input $x = (x_1, \ldots, x_n) \in \mathbb{F}_2^n$, we can write

$$\mathbb{1}_{L_i}(x) = \prod_{t \in P_i} \left(\frac{1 + (-1)^{a_t + x_t}}{2} \right) = \frac{1}{2^d} \left(\sum_{P \subseteq P_i} (-1)^{\sum_{j \in P} a_j} \cdot (-1)^{\sum_{j \in P} x_j} \right),$$

where $a_t \in \mathbb{F}_2$ is the output of node t on the path P_i.

To find the Fourier support $\mathcal{S} = \mathrm{supp}(\widehat{f})$, it remains to determine which terms "survive" cancellation in Equation (1). Let $\mathcal{N}(i)$ be the index of the internal node adjacent to leaf i. Observe that when $\mathcal{N}(i) = \mathcal{N}(j)$ for $i \neq j$ (so $\mathrm{sign}(L_i) = -\mathrm{sign}(L_j)$),

$$\begin{aligned} 2^d(\mathrm{sign}(L_i) \cdot \mathbb{1}_{L_i}(x) + \mathrm{sign}(L_j) \cdot \mathbb{1}_{L_j}(x)) &= \mathrm{sign}(L_i) \sum_{P \subseteq P_i} (-1)^{\sum_{t \in P} a_t} \cdot (-1)^{\sum_{t \in P} x_t} \\ &\quad - \mathrm{sign}(L_i) \sum_{P \subseteq P_j} (-1)^{\sum_{t \in P} a_t} \cdot (-1)^{\sum_{t \in P} x_t} \\ &= 2 \cdot \mathrm{sign}(L_i) \cdot \sum_{P \subseteq P_i \,:\, \mathcal{N}(i) \in P} (-1)^{\sum_{t \in P} a_t} \cdot (-1)^{\sum_{t \in P} x_t}, \end{aligned}$$

since $x_{\mathcal{N}(i)}$ is the only x value that P_i and P_j disagree on. That is, each term in f's expansion must contain $\mathcal{N}(i)$ for some i. Moreover, once these cancellations are made, $\mathbb{1}_{L_i}$ does not interact with $\mathbb{1}_{L_j}$ for $\mathcal{N}(i) \neq \mathcal{N}(j)$, since no term can contain both $\mathcal{N}(i)$ and $\mathcal{N}(j)$. In summary,

$$\mathcal{S} = \bigcup_{i \in [2^d]} \{s : s \subseteq P_i \text{ and } \mathcal{N}(i) \in s\}.$$

Let $\gamma_1, \gamma_2 \in \mathcal{S}$. By our observation on the structure of $\mathcal{S}$, they have the form $\gamma_1 = \alpha_1 \dot\cup \{\mathcal{N}(i)\}$ and $\gamma_2 = \alpha_2 \dot\cup \{\mathcal{N}(j)\}$ for some $i, j \in [2^d]$. We are interested in the number of pairs $(\beta_1, \beta_2) \in \mathcal{S} \times \mathcal{S}$ such that $\gamma_1 + \gamma_2 = \beta_1 + \beta_2$. It will suffice to focus on the setting $\mathcal{N}(i) \neq \mathcal{N}(j)$ since this occurs with overwhelming probability. In this case, the quantity $|(\mathcal{S} + \gamma_1) \cap (\mathcal{S} + \gamma_2)|$ depends only on the depth of the lowest common ancestor of P_i and P_j.

▷ Claim 17. If $|(\mathcal{S} + \gamma_1) \cap (\mathcal{S} + \gamma_2)| \geq 2^{k+2}$, then the lowest common ancestor of P_i and P_j is at depth at least k.

Proof. We will show the contrapositive. Suppose the lowest common ancestor a of P_i and P_j is at depth $\ell < k$, and suppose $\beta_1, \beta_2 \in \mathcal{S}$ satisfy $\beta_1 + \gamma_1 = \beta_2 + \gamma_2$. Without loss of generality, assume $\mathcal{N}(i) \in \beta_1$ and $\mathcal{N}(j) \in \beta_2$. Then β_1 and β_2 must be a subset of the elements in the paths P_i and P_j, respectively.

First, consider each element $E \in P_i \cap P_j$, which is all those above (and including) a. If $E \in \gamma_1 + \gamma_2$, then $E \in \beta_1 + \beta_2$ only if E is in precisely one of β_1, β_2. Likewise, if $E \notin \gamma_1 + \gamma_2$, then $E \notin \beta_1 + \beta_2$ only if E is in neither or both of β_1, β_2. In either case, we have two options for each E.

Now consider each element $E \in P_i$ below a. By assumption, $E \notin P_j$. Thus, if $E \in \gamma_1 + \gamma_2$, it must be that $E \in \gamma_1$ and $E \notin \gamma_2$. For $\beta_1 + \beta_2$ to contain E, we must likewise have $E \in \beta_1$ and $E \notin \beta_2$. Similarly, if $E \notin \gamma_1 + \gamma_2$, it cannot be in γ_1 or γ_2. Thus, it is not in β_1 or β_2 either. An identical argument for $E \in P_j$ shows that we only have one way to account for elements in the paths P_i or P_j below a.

Doubling to compensate for the cases where $\mathcal{N}(j) \in \beta_1$ and $\mathcal{N}(i) \in \beta_2$, we find the number of options for $(\beta_1, \beta_2) \in \mathcal{S} \times \mathcal{S}$ such that $\beta_1 + \gamma_1 = \beta_2 + \gamma_2$ is at most $2^{\ell+2} < 2^{k+2}$. ◁

Theorem 16 follows quickly from the claim. The probability that P_i and P_j have a common ancestor at depth at least k is at most 2^{-k}, so

$$\begin{aligned}
&\Pr_{\gamma_1,\gamma_2\in\mathcal{S}}\left[|(\mathcal{S}+\gamma_1)\cap(\mathcal{S}+\gamma_2)|\geq 2^{k+2}\right]\\
&\leq \Pr_{\gamma_1,\gamma_2\in\mathcal{S}}\left[|(\mathcal{S}+\gamma_1)\cap(\mathcal{S}+\gamma_2)|\geq 2^{k+2}\,\middle|\,\mathcal{N}(\gamma_1)\neq\mathcal{N}(\gamma_2)\right]+2^{1-d}\\
&\leq 2^{-k}+2^{1-d},
\end{aligned}$$

where we overload notation by letting $\mathcal{N}(\gamma)=\mathcal{N}(i)\in\gamma$.

5 Conclusion

While the provided functions rule out specific approaches, it is worth noting that neither are a counterexample to the log-rank conjecture. The subspace addressing function (Section 3) has a simple PDT: first individually query all $7k$ address bits, then query the bit to the corresponding subspace. Since the Fourier sparsity is at least 2^{6k}, this is certainly affordable. While this example refutes a general greedy approach, such an approach works for the decision tree function (Section 4). Each query of the root variable eliminates half the paths (and thus reduces the sparsity by two), so iterating this process quickly makes the function constant.

References

1 Arkadev Chattopadhyay, Ankit Garg, and Suhail Sherif. Towards stronger counterexamples to the log-approximate-rank conjecture. In Mikolaj Bojanczyk and Chandra Chekuri, editors, *Proceedings of the 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 213 of *LIPIcs*, pages 13:1–13:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSTTCS.2021.13.

2 Hamed Hatami, Kaave Hosseini, and Shachar Lovett. Structure of protocols for XOR functions. *SIAM Journal on Computing*, 47(1):208–217, 2018. doi:10.1137/17M1136869.

3 Alexander Knop, Shachar Lovett, Sam McGuire, and Weiqiang Yuan. Log-rank and lifting for AND-functions. In Samir Khuller and Virginia Vassilevska Williams, editors, *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 197–208. ACM, 2021. doi:10.1145/3406325.3450999.

4 László Lovász and Michael Saks. Communication complexity and combinatorial lattice theory. *Journal of Computer and System Sciences*, 47(2):322–349, 1993. doi:10.1016/0022-0000(93)90035-U.

5 Shachar Lovett. Communication is bounded by root of rank. *Journal of the ACM (JACM)*, 63(1):1:1–1:9, 2016. doi:10.1145/2724704.

6 Nikhil S Mande and Swagato Sanyal. On parity decision trees for Fourier-sparse boolean functions. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 182 of *LIPIcs*, pages 29:1–29:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSTTCS.2020.29.

7 Kurt Mehlhorn and Erik M Schmidt. Las Vegas is better than determinism in VLSI and distributed computing. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 330–337. ACM, 1982. doi:10.1145/800070.802208.

8 Ashley Montanaro and Tobias Osborne. On the communication complexity of XOR functions. *CoRR*, abs/0909.3392, 2009. doi:10.48550/arXiv.0909.3392.

9 Ryan O'Donnell. *Analysis of boolean functions*. Cambridge University Press, 2014. URL: http://www.cambridge.org/de/academic/subjects/computer-science/algorithmics-complexity-computer-algebra-and-computational-g/analysis-boolean-functions.

10 Ryan O'Donnell, John Wright, Yu Zhao, Xiaorui Sun, and Li-Yang Tan. A composition theorem for parity kill number. In *2014 IEEE 29th Conference on Computational Complexity (CCC)*, pages 144–154. IEEE, IEEE Computer Society, 2014. doi:10.1109/CCC.2014.22.

11 Anup Rao and Amir Yehudayoff. *Communication Complexity: and Applications*. Cambridge University Press, 2020.

12 Amir Shpilka, Avishay Tal, and Ben Lee Volk. On the structure of boolean functions with small spectral norm. In Moni Naor, editor, *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 37–48. ACM, 2014. doi:10.1145/2554797.2554803.

13 Benny Sudakov and István Tomon. Matrix discrepancy and the log-rank conjecture. *CoRR*, abs/2311.18524, 2023. doi:10.48550/arXiv.2311.18524.

14 Hing Yin Tsang, Chung Hoi Wong, Ning Xie, and Shengyu Zhang. Fourier sparsity, spectral norm, and the log-rank conjecture. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 658–667. IEEE, IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.76.

15 Zhiqiang Zhang and Yaoyun Shi. On the parity complexity measures of boolean functions. *Theoretical Computer Science*, 411(26-28):2612–2618, 2010. doi:10.1016/j.tcs.2010.03.027.

No Polynomial Kernels for Knapsack

Klaus Heeger
Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Danny Hermelin
Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Matthias Mnich
Institute for Algorithms and Complexity, Hamburg University of Technology, Hamburg, Germany

Dvir Shabtay
Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract

This paper focuses on kernelization algorithms for the fundamental Knapsack problem. A kernelization algorithm (or kernel) is a polynomial-time reduction from a problem onto itself, where the output size is bounded by a function of some problem-specific parameter. Such algorithms provide a theoretical model for data reduction and preprocessing and are central in the area of parameterized complexity. In this way, a kernel for Knapsack for some parameter k reduces any instance of Knapsack to an equivalent instance of size at most $f(k)$ in polynomial time, for some computable function f. When $f(k) = k^{O(1)}$ then we call such a reduction a polynomial kernel.

Our study focuses on two natural parameters for Knapsack: The number $w_\#$ of different item weights, and the number $p_\#$ of different item profits. Our main technical contribution is a proof showing that Knapsack does not admit a polynomial kernel for any of these two parameters under standard complexity-theoretic assumptions. Our proof discovers an elaborate application of the standard kernelization lower bound framework, and develops along the way novel ideas that should be useful for other problems as well. We complement our lower bounds by showing that Knapsack admits a polynomial kernel for the combined parameter $w_\# \cdot p_\#$.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Combinatorial optimization

Keywords and phrases Knapsack, polynomial kernels, compositions, number of different weights, number of different profits

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.83

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2308.12593 [28]

Funding Supported by the ISF, grant No. 1070/20.
Matthias Mnich: Supported by DFG grant MN 59/4-1.

1 Introduction

This paper proves a new complexity-theoretic barrier for the classic Knapsack problem. Namely, we prove that Knapsack has no polynomial kernels when parameterized by either the number $w_\#$ of different weights, or the number $p_\#$ of different profits. Our results hold under the standard complexity-theoretic assumption NP $\not\subseteq$ coNP/poly. We also show that when both $w_\#$ and $p_\#$ are taken as a combined parameter, Knapsack does admit a polynomial kernel. Below we give a brief review of recent algorithmic progress for the

Knapsack problem, as well as a quick survey through kernelization and parameterized complexity. We then describe how our results fit into the current state of the art, and give an overview of the main techniques used for obtaining our new hardness result for Knapsack.

The Knapsack problem. Knapsack (also known as 0/1 Knapsack) is one the most fundamental and well studied problems in combinatorial optimization and theoretical computer science. In its most basic form, it is defined as follows:

Knapsack
Input: A set $\mathcal{X} = \{x_1, \ldots, x_n\}$ of n items, a weight function $w : \mathcal{X} \to \mathbb{N}$, a profit function $p : \mathcal{X} \to \mathbb{N}$, and two integers W and P.
Question: Is there a subset $\mathcal{S} \subseteq \mathcal{X}$ with total weight $w(\mathcal{S}) = \sum_{x \in \mathcal{S}} w(x) \leq W$ and total profit $p(\mathcal{S}) = \sum_{x \in \mathcal{S}} p(x) \geq P$?

Knapsack enjoys a key status in algorithmic design due to numerous reasons. First, it has many natural applications, in various practical areas such as resource allocation and scheduling. Second, it has immense educational value: Karp's NP-hardness proof (from his seminal paper [35]) is the first example of a reduction to a problem involving numbers, while the $O(Wn)$-time (or $O(Pn)$-time) algorithm by Bellman [4] from 1957 is one of the first dynamic programming algorithms, and it is still taught in most undergraduate algorithms courses to this day. And third, Knapsack has deep connections to other areas of computation: For example, one of the earliest cryptosystems by Merkle and Hellman [41] was based on Knapsack, and this was later extended to a host of other *Knapsack-type* cryptosystems [7, 14, 30, 42].

Knapsack is also important since it is both a generalization and a special case of a few other classic problems. For instance, it is a special case of the fundamental scheduling problem of minimizing the weighted number of tardy jobs on a single machine, the so-called $1||\sum w_j U_j$ problem [43, p. 19]. The variant of Knapsack where $W = P$ and $w(x) = p(x)$ for each item $x \in \mathcal{X}$ is precisely the Subset Sum problem:

Subset Sum
Input: A set $\mathcal{A} = \{a_1, \ldots, a_n\}$ of n non-negative integers and a target integer B.
Question: Is there a subset $\mathcal{A}^* \subseteq \mathcal{A}$ with $\sum_{a \in \mathcal{A}^*} a = B$?

Entire books [37, 40] are dedicated to algorithmics for Knapsack. Quite surprisingly, major algorithmic advances are still being discovered in recent years. These typically involve improvements on Bellman's classic $O(\min\{W, P\} \cdot n)$-time algorithm in cases where the maximum weight $w_{\max} = \max_x w(x)$ or the maximum profit $p_{\max} = \max_x p(x)$ (or both) are relatively small. Furthermore, they may result in faster approximation schemes. Currently, the fastest known approximation scheme runs – after a series of improvements in recent years [9, 12, 32] – in $\widetilde{O}(n + (1/\varepsilon)^{2.2})$ time [19]. Pisinger [44] was the first to present such an improvement with his $O(w_{\max} \cdot p_{\max} \cdot n)$-time algorithm. Later followed a series of papers with further improvements [2, 3, 36], culminating in an $\widetilde{O}(\min\{w_{\max}^3, p_{\max}^3\} + n)$-time[1] algorithm by Polak et al. [45], and an $\widetilde{O}(\min\{w_{\max}^{2/3} \cdot p_{\max},\ w_{\max} \cdot p_{\max}^{2/3}\} \cdot n)$-time algorithm by Bringmann and Cassis [10]. Very recently, Chen et al. [13] gave an $O(n + w_{\max}^{2.4})$-time algorithm. Subsequently, Bringmann [8] as well as Jin [33] announced $\widetilde{O}(n + w_{\max}^2)$-time algorithms (and also $\widetilde{O}(n + p_{\max}^2)$-time algorithms) for Knapsack. Cygan et al. [17]

[1] We use $\widetilde{O}(\cdot)$ to suppress polylogarithmic factors.

and independently Künnemann et al. [38] showed that there are no $O((W+n)^{2-\varepsilon})$-time algorithms for Knapsack, for any $\varepsilon > 0$, unless $(\min, +)$-convolutions can be solved in truly subquadratic time. An easy modification of their argument shows also that there are also no $O((P+n)^{2-\varepsilon})$-time algorithms under the same hypothesis. However, interestingly enough, these two lower bounds do not hold simultaneously together as shown by Bringmann and Cassis [9], who showed that Knapsack can be solved in $O((W+P)^{1.5}+n)$ time.

Parameters $w_\#$ and $p_\#$. As discussed above, the time complexity of Knapsack with respect to $w_{\max}$ and $p_{\max}$ is well understood. Thus, it makes sense to consider other natural parameters. Let $w_\# = |\{w(x_i) : i \in \{1,\dots,n\}\}|$ denote the number of *different* weights in a given Knapsack instance, and let $p_\# = |\{p(x_i) : i \in \{1,\dots,n\}|$ denote the number of different profits. These parameters can be expected to be small in several natural applications (for example, the first step of many approximation schemes is to round the profits of the items such that only $f(\varepsilon)$ many different profits remain; see, e.g., the approximation scheme by Deng et al. [19]) and have been initially studied for Subset Sum (i.e., when $w(x_i) = p(x_i)$ for all items x_i) by Fellows et al. [23] in the context of the more general *small number of numbers* paradigm. Clearly, $w_\# \leq w_{\max}$ and $p_\# \leq p_{\max}$, and so designing algorithms which are efficient in terms of $w_\#$ or $p_\#$ is more challenging than for $w_{\max}$ and $p_{\max}$. In particular, since Knapsack is NP-hard and $w_\#, p_\# \leq n$, we cannot expect algorithms with running times of the form $(w_\# \cdot n)^{O(1)}$, $(p_\# \cdot n)^{O(1)}$, or even $(w_\# \cdot p_\# \cdot n)^{O(1)}$.

Since polynomial-time algorithms with respect to $w_\#$ and $p_\#$ are unlikely to exist, it is natural to consider these two parameters in the context of parameterized complexity [16]. In this context, it is not difficult to show that Knapsack can be formulated as an Integer Linear Program (ILP) with either $O(w_\#)$ or $O(p_\#)$ variables [22]. Using one of several solvers for ILP with few variables, such as Lenstra's famous algorithm [39, 34], gives us algorithms with running times of $2^{\widetilde{O}(w_\#)} \cdot |I|$ and $2^{\widetilde{O}(p_\#)} \cdot |I|$ for Knapsack, where $|I|$ is the total encoding length of the input (see Hermelin et al. [29] for algorithms with similar running times for the more general $1||\sum w_j U_j$ problem). Such algorithms are known as *fixed-parameter algorithms* (FPT) in the terminology of parameterized complexity.

Note that both of these algorithms cannot be significantly improved assuming the Exponential Time Hypothesis (ETH). Indeed, it is known that assuming ETH, there are no $2^{o(n)}$-time algorithms for Subset Sum [11, 31]. As both $w_\#$ and $p_\#$ are bounded by n, and Subset Sum is a special case of Knapsack, this implies that, most likely, there are no $2^{o(w_\#)} \cdot n$-time or $2^{o(p_\#)} \cdot n$-time algorithms for Knapsack. Using the Strong Exponential Time Hypothesis (SETH) instead of ETH, Abboud et al. [1] showed a stronger lower bound for Subset Sum, excluding $B^{1-\varepsilon} \cdot 2^{o(n)}$-time algorithms for any $\varepsilon > 0$ (which excludes $(W+P)^{1-\varepsilon} \cdot 2^{o(w_\#+p_\#)}$-time algorithms for Knapsack). Once fixed-parameter algorithms for a particular problem – in this case, Knapsack – have been devised, the next step is to understand the kernelization complexity of the problem at hand.

Kernelization. One of the most fundamental and important techniques in parameterized complexity is kernelization [24]:

▶ **Definition 1.** *A* kernelization *algorithm (or* kernel*) for a parameterized problem* Π *is an algorithm that receives as input an instance* I *of* Π *with parameter* k *and outputs in polynomial time an instance* J *of* Π *with parameter* ℓ *such that*

- (I,k) *is a "yes"-instance for* Π *if and only if* (J,ℓ) *is a "yes"-instance for* Π*, and*
- $|J| + \ell \leq f(k)$ *for some computable function* f*.*

The expression $f(k)$ *is referred to as the* size *of the kernel.*

Thus, a kernel is a self-reduction from a problem onto itself that produces an equivalent instance with size bounded by the input parameter. In this way, kernelization may be thought of as preprocessing that aims to simplify or "kernelize" a problem instance by reducing its size while preserving some solution. Following this line of thought, problems that admit kernels of small sizes may be thought of as problems that allow efficient and effective preprocessing. For this reason, research into kernelization algorithms has seen a significant surge in recent years, and has become one of the central topics in parameterized complexity (see e.g. the monograph by Fomin et al. [24] and the numerous references within).

In the context of Knapsack, a kernelization algorithm for say parameter $w_\#$ transforms any Knapsack instance in polynomial time into an equivalent instance where the total encoding length of item weights and profits are bounded by $f(w_\#)$, for some function f. Observe that the $2^{\widetilde{O}(w_\#)} \cdot n$-time algorithm for Knapsack implies a kernel of size $2^{\widetilde{O}(w_\#)}$ [22]: Indeed, let $|I|$ denote the total encoding length of item weights and profits in a given Knapsack instance I with n items and observe that $n \leq |I|$. The kernelization algorithm can first check whether $|I| = 2^{\widetilde{O}(w_\#)}$. If this is the case, then the instance already has size bounded by $2^{\widetilde{O}(w_\#)}$, and otherwise it is solvable in polynomial time by the $2^{\widetilde{O}(w_\#)} \cdot n = 2^{\widetilde{O}(w_\#)} \cdot |I|$-time algorithm. A similar argument shows that Knapsack has a kernel of size $2^{\widetilde{O}(p_\#)}$.

The obvious question to ask is whether we can obtain smaller kernels with respect to either $w_\#$ or $p_\#$. Here, the gold standard in parameterized complexity are *polynomial kernels*, kernels with size $f(k) = k^{O(1)}$. By now we know of countless fixed-parameter tractable (NP-hard) problems that also admit polynomial kernels [24]; yet, at the same time, for many other problems polynomial kernels were shown to be unlikely to exist. Thus, the central question this paper addresses is:

Does Knapsack admit a polynomial kernel with respect to either $w_\#$ or $p_\#$?

Note that there are results that give encouraging indications to the question above. Etscheid et al. [22] show that Subset Sum admits a polynomial kernel with respect to $a_\#$, the number of different numbers in the instance (i.e, $a_\# = |\{a_i : i \in \{1, \ldots, n\}\}|$). Can this result be generalized to hold also for the Knapsack problem?

1.1 Our results

Our main technical result of the paper is a negative answer to the question above. We use the by now standard framework for excluding polynomial kernels [5] based on the assumption NP $\not\subseteq$ coNP/poly (whose negation implies that the polynomial hierarchy collapses) to show the following:

▶ **Theorem 2.** *Assuming NP $\not\subseteq$ coNP/poly, there is no polynomial kernel for* Knapsack *parameterized by the number $w_\#$ of different weights, nor by the number $p_\#$ of different profits.*

The proof of the theorem above is obtained through a rather involved construction of what is known as a composition algorithm (see Definition 4). This algorithm composes several instances of a specialized variant of Subset Sum into a single Knapsack instance where either $w_\#$ or $p_\#$ is kept relatively small. We give an overview of this algorithm in Section 1.2.

Complementing the negative result of Theorem 2, we show that when both $w_\#$ and $p_\#$ are taken as a combined parameter, the polynomial kernel for Subset Sum [22] can be extended to a polynomial kernel for Knapsack.

▶ **Theorem 3.** Knapsack *parameterized by $w_\# \cdot p_\#$ admits a polynomial kernel.*

Thus, the lower bounds for $w_\#$ and $p_\#$ cannot be combined. This is somewhat reminiscent to the situation mentioned above, where Knapsack is unlikely to have algorithms with subquadratic running times in either $(W+n)$ or $(P+n)$ [17], but admits an algorithm with subquadratic running time in $(W+P+n)$ [9].

1.2 Technical overview

We prove Theorem 2 for the parameter $w_\#$; the statement for parameter $p_\#$ then follows by a reduction from Polak et al. [45, Chapter 4]. In broad terms, our proof follows the standard framework for showing kernelization lower bounds that has been developed over the years [5, 6, 18, 21]. In this framework, to exclude a polynomial kernel for a given problem Π_1 parameterized by some parameter k, one shows what is called an *OR-composition algorithm* (see Definition 4 for a formal definition) from some known NP-hard problem Π_2 to Π_1. This algorithm takes as input t instances of Π_2 of size n each, and converts these instances in polynomial time to an equivalent instance of Π_1 such that $k = (n \lg t)^{O(1)}$. Here, equivalence means that at least one of the t input instances is a "yes"-instance for Π_2 if and only if the output instance of the composition is a "yes"-instance for Π_1. Such a composition algorithm then directly implies that Π_2 has no polynomial kernel with respect to k under the assumption NP $\not\subseteq$ coNP/poly [5, 6].

We show an OR-composition algorithm from a restricted version of Subset Sum which we call Restricted Subset Sum. In this restricted version, any instance of size n is restricted to include only numbers from a known set $\widetilde{\mathcal{A}}_n$ of size $O(n^3)$. This allows us to bound the number of different numbers in any t instances of Restricted Subset Sum of the same size. Our composition algorithm then proceeds as follows: Given any t instances $\mathcal{A}_0, \ldots, \mathcal{A}_{t-1}$ of Restricted Subset Sum, the algorithm converts any integer $a \in \mathcal{A}_i$, $i \in \{0, \ldots, t-1\}$, to a Knapsack item x_a (which we refer to as an *encoding item*) with weight $w(x_a) = a$. We then assign a higher profit to items corresponding to Restricted Subset Sum instances of higher index, meaning that it is always less profitable to choose an item corresponding to some integer $a \in \mathcal{A}_i$ than choosing an item corresponding to some integer $a \in \mathcal{A}_{i_0}$ for $i_0 > i$. This allows us to encode any solution $\mathcal{A}_i^* \subseteq \mathcal{A}_i$ to the i'th Restricted Subset Sum instance by a solution $\mathcal{X}(\mathcal{A}_i^*)$ to our Knapsack instance which includes all items x_a for $a \in \mathcal{A}_i^*$, and all items x_a for $a \in \mathcal{A}_{i_0}$ for $i_0 > i$. Thus, if we knew index i in advance, our composition would be complete.

However, we do not have a priori knowledge of index i. To circumvent this, we use *index items* that encode the selection of $i \in \{0, \ldots, t-1\}$. These are $2 \lg t$ items that encode the selection of t binary values $i(0), \ldots, i(\lg t - 1)$ that correspond to the base-2 representation of i, i.e., $i = \sum_k i(k) \cdot 2^k$. This is somewhat akin to the *"colors and IDs"* technique for composition algorithms which was introduced by Dom et al. [20]. The goal of these index items is to ensure that choosing a solution $\mathcal{X}(\mathcal{A}_i^*)$, for any value of i, allows adding a subset of index items to the knapsack such that any such selection has the same weight and profit. However, just adding the $O(\lg t)$ index items is not sufficient: As the composed instance shall achieve the same profit for any solution $\mathcal{X}(\mathcal{A}_i^*)$, we need to compensate for this difference in profit between $\mathcal{X}(\mathcal{A}_i^*)$ and $\mathcal{A}_{t-1}$. The difference in profit between solution $\mathcal{X}(\mathcal{A}_i^*)$ which includes all items from instances $\mathcal{A}_{i_0}$ for $i_0 > i$, and solution $\mathcal{X}(\mathcal{A}_{t-1}^*)$ which includes only items corresponding to instance $\mathcal{A}_{t-1}$, is quadratic in i. Hence, we cannot compensate for this difference using only the $O(\lg t)$ index items.

We therefore introduce additional $O(\lg^2 t)$ items which we refer to as *quadratization items*. These encode the selection of binary pair values $i(k)i(\ell)$ in $i^2 = \sum_k \sum_\ell i(k)i(\ell) \cdot 2^{k+\ell}$, and allow us to encode the quadratic compensation mentioned above. Unfortunately, this introduces

additional technical difficulties, such as ensuring compatibility between the selection of the index and quadratization items. Meanwhile, we still need to maintain the compatibility between the encoding and index items as well. These difficulties are overcome using various applications of a basic algebraic lemma that we prove in Section 2. In the end, we obtain an instance with $w_\# = O(n^3 \cdot \lg^2 t)$ many different weights, which by previously known results (Theorem 5) implies that a polynomial kernel would imply NP $\not\subseteq$ coNP/poly. The full details of the entire composition are given in Section 3.

In Section 4, we derive a polynomial kernel by first modeling Knapsack by an ILP with $w_\# \cdot p_\#$ many variables, and then reducing the size of the numbers occurring in the ILP by a well-known result by Frank and Tardos [26].

Theorems whose proof is omitted (and can be found in the full version [28]) are marked by $\star$.

2 Preliminaries

We next quickly review the kernelization lower bounds framework, introduced by Bodlaender et al. [5] and further developed by [6, 18, 21], that will be used for proving Theorem 2. At the heart of the framework lies the notion of a composition.

▶ **Definition 4.** *A composition algorithm from a problem Π_1 to a parameterized problem Π_2 is an algorithm that receives as input t instances $I_0, \ldots, I_{t-1}$ of size n of Π_1, and computes in polynomial time an instance (J, k) of Π_2 such that*

- *J is a "yes"-instance of Π_2 if and only if I_i is a "yes"-instance of Π_1 for some $i \in \{0, \ldots, t-1\}$,*
- *and $k \leq (n + \lg t)^{O(1)}$.*

The main connection between composition algorithms and the exclusion of polynomial kernels is given in the theorem below, whose proof relies on a complexity-theoretic lemma by Fortnow and Santhanam [25].

▶ **Theorem 5** ([5, 6])**.** *Let Π_1 be an NP-hard problem, and Π_2 be a parameterized problem with a composition algorithm from Π_1 to Π_2. Then Π_2 does not admit a polynomial kernel, assuming NP $\not\subseteq$ coNP/poly.*

Note that NP $\not\subseteq$ coNP/poly is a widely believed assumption in complexity theory, and if it were false then the polynomial hierarchy would collapse to its third level [47].

As a final point, we will also make use of the following basic algebraic lemma throughout our proof. Below we provide a proof for the sake of completeness.

▶ **Lemma 6.** *Let $b > 1$ and k be positive integers. Then there exists a unique integer solution to the equation*

$$x_0 b^0 + x_1 b^1 + \cdots + x_{k-1} b^{k-1} = \sum_{i=0}^{k-1} b^i$$

constrained by $x_i \in \{0, 1, \ldots, b\}$ for each $i \in \{0, \ldots, k-1\}$. Namely, the solution is

$$x_0 = x_1 = \cdots = x_{k-1} = 1 \ .$$

Proof. We show the statement by induction on k. For $k = 1$, the statement is obvious. So fix $k > 1$. Note that $\sum_{i=0}^{k-1} b^i = b^{k-1} + \sum_{i=0}^{k-2} b^i = b^{k-1} + \frac{b^{k-1}}{b-1} < 2b^{k-1}$, so we have that $x_{k-1} \in \{0, 1\}$. Further, we have $\sum_{i=0}^{k-2} x_i b^i \leq \sum_{i=0}^{k-2} b \cdot b^i = \sum_{i=1}^{k-1} b^i < \sum_{i=0}^{k-1} b^i$, implying that $x_k > 0$. Thus, we have $x_{k-1} = 1$ and $x_i = 1$ for $i \in \{0, \ldots, k-2\}$ follows by induction. ◀

Further, for any function $f : S \to \mathbb{R}$, we define $f(S') := \sum_{s \in S'} f(s)$ for any $S' \subseteq S$.

3 No Polynomial Kernel for Parameter $w_{\#}$

In the following, we present the proof of Theorem 2 for parameter $w_{\#}$, the total number of different weights in a Knapsack instance. As mentioned above, in our proof we construct a composition from a restricted version of Subset Sum to Knapsack parameterized by $w_{\#}$. The proof is divided into four parts: In the first part, we introduce the Restricted Subset Sum problem, and prove that it is NP-hard. In the second part we begin to describe our composition by showing how to encode t instances $\mathcal{A}_0, \ldots, \mathcal{A}_{t-1}$ of Restricted Subset Sum into a single set $\mathcal{X}$ of *encoding items*. In the third part we describe the set $\mathcal{Y}$ of *quadratization items*, and the set $\mathcal{Z}$ of *index items*, which together form the instance selection gadget of the composition. The final part is devoted to finishing details, and to proving the correctness of the composition.

3.1 Restricted Subset Sum

We start with a restricted version of Subset Sum which allows us to bound the number of different numbers appearing in any set of t Subset Sum instances. Namely, in our restricted version of Subset Sum, any instance of size n contains numbers from a restricted set of $O(n^3)$ numbers. Apart from this, its useful properties are that the target value only depends on the number of input numbers, and that any solution must have the same cardinality. For this, we define the set of possible numbers which may be contained in a Restricted Subset Sum instance of size n as

$$\widetilde{\mathcal{A}}_n := \left\{(3n+1)^{j_1} + (3n+1)^{j_2} + (3n+1)^{j_3} \;:\; j_1, j_2, j_3 \in \{1, \ldots, 3n\}\right\} .$$

Note that $\widetilde{\mathcal{A}}_n$ contains $27n^3 = O(n^3)$ integers. Furthermore, we also define a global target for all instances of size n by $B_n := \sum_{j=1}^{3n}(3n+1)^j$.

Restricted Subset Sum
Input: A set $\mathcal{A} = \{a_1, \ldots, a_{3n}\}$ of $3n$ integers from $\widetilde{\mathcal{A}}_n$ with $\sum_{j=1}^{3n} a_j = 3B_n$.
Question: Is there a subset $\mathcal{A}^* \subseteq \mathcal{A}$ with $|\mathcal{A}^*| = n$ such that $\sum_{a \in \mathcal{A}^*} a = B_n$?

▶ **Lemma 7.** Restricted Subset Sum *is NP-complete.*

Proof. We present a reduction from a variant of Restricted Exact Cover by 3-Sets where each element appears in exactly three sets. Recall that in Restricted Exact Cover by 3-Sets, the input consists of a set $\mathcal{T}$ consisting of 3-element subsets of $\{1, \ldots, 3n\}$ such that each $j \in \{1, \ldots, 3n\}$ appears in exactly three sets from $\mathcal{T}$, and the question is whether there exists a subset $\mathcal{T}' \subseteq \mathcal{T}$ such that each element from $\{1, \ldots, 3n\}$ appears in exactly one set from $\mathcal{T}'$. This variant of Restricted Exact Cover by 3-Sets is well-known to be NP-hard [27].

Our reduction is almost identical to the original hardness reduction for Subset Sum by Karp [35], only that we reduce from Restricted Exact Cover by 3-Sets instead of the more general Exact Cover: Let $\mathcal{T}$ be an instance of Restricted Exact Cover by 3-Sets. For each 3-element set $T \in \mathcal{T}$, a number $a_T := \sum_{j \in T}(3n+1)^j$ is added to the Restricted Subset Sum instance. In this way, the constructed instance of Restricted Subset Sum is $\mathcal{A} := \{a_T : T \in \mathcal{T}\}$. Note that $a_T \in \widetilde{\mathcal{A}}_n$ for every $T \in \mathcal{T}$. Further, since each $j \in \{1, \ldots, 3n\}$ appears in exactly three sets from $\mathcal{T}$, we have $\sum_{T \in \mathcal{T}} a_T = 3\sum_{j=1}^{3n}(3n+1)^j = 3B_n$. Below we prove the correctness of this construction.

Suppose there is a solution $\mathcal{T}' \subseteq \mathcal{T}$ for the Restricted Exact Cover by 3-Sets instance. Then $|\mathcal{T}'| = n$ and we have $\sum_{T \in \mathcal{T}} a_T = \sum_{\ell=1}^{3n}(3n+1)^\ell = B_n$. Conversely, suppose there is a solution $\mathcal{A}^* \subseteq \mathcal{A}$ with $|\mathcal{A}^*| = n$ for the Restricted Subset Sum instance. Let

$\mathcal{T}^* := \{T : a_T \in \mathcal{A}^*\}$. Since each term $(3n+1)^j$ appears only $3 < 3n+1$ times, Lemma 6 implies that the only way for some numbers from $\mathcal{A}$ to add up to $B_n = \sum_{j=1}^{3n}(3n+1)^j$ is that each term $(3n+1)^j$ appears in exactly one $a_T \in \mathcal{A}^*$. In other words, for each $j \in \{1,\dots,3n\}$, there is exactly one $T \in \mathcal{T}^*$ with $j \in T$. ◀

3.2 Encoding Gadget

In the following, we show how to encode t instances of Restricted Subset Sum into a single Knapsack instance. Throughout the remainder of the section, we use $\mathcal{A}_0,\dots,\mathcal{A}_{t-1}$ to denote the t input instances of Restricted Subset Sum to our composition, where $|\mathcal{A}_i| = 3n$ for each $i \in \{0,\dots,t-1\}$. By copying instances, we may assume without loss of generality that t is a power of 2, i.e., $t = 2^s$ for some $s \in \mathbb{N}$. Furthermore, we let $\mathcal{A}_i = \{a_1^i,\dots,a_{3n}^i\}$ denote the i'th instance for each $i \in \{0,\dots,t-1\}$. By the definition of Restricted Subset Sum, we have $a_j^i \in \widetilde{\mathcal{A}}_n$ for each each $i \in \{0,\dots,t-1\}$ and $j \in \{1,\dots,3n\}$, and $\sum_j a_j^i = 3B_n$ for each $i \in \{0,\dots,t-1\}$.

Recall that Subset Sum can be seen as a special case of Knapsack where the profit of each item equals its weight. This yields an easy reduction from each single Restricted Subset Sum instance $\mathcal{A}_i$ to Knapsack. We apply this reduction with a slight modification: To capture the condition that each solution of Restricted Subset Sum shall contain exactly n numbers, we add a large number $X = 3tnB_n$ to each number in a Restricted Subset Sum instance, and set $B := B_n + n \cdot X$. We also increase the profit of each item corresponding to the i'th Restricted Subset Sum instance by adding $i \cdot 3B$ to its profit. More precisely, for each $i \in \{0,\dots,t-1\}$ and $j \in \{1,\dots,3n\}$, we construct an *encoding item* x_j^i with

- $w(x_j^i) = X + a_j^i$, and
- $p(x_j^i) = X + a_j^i + i \cdot 3B$.

Intuitvely, adding the constant X to the weights ensures that all encoding items have roughly the same size, so for any weight budget W^*, the set of maximum profit will always have size $\lfloor W^*/X \rfloor$ or $\lfloor W^*/X \rfloor - 1$. Adding the constant X to the profits of x_j^i ensures that for any two sets of encoding items with different cardinalities, the larger one will always have the larger profit. Further, adding $i \cdot 3B$ to the profits will ensure that it will be always more profitable to pick an item $x_{j_1}^{i_1}$ over item $x_{j_0}^{i_0}$ for any $i_1 > i_0$. We use $\mathcal{X} = \{x_j^i : i \in \{0,\dots,t-1\}, j \in \{1,\dots,3n\}\}$ to denote the set of all encoding items.

Let $\mathcal{X}_i$ denote the set of encoding items corresponding to instance $\mathcal{A}_i$ of Restricted Subset Sum, i.e., $\mathcal{X}_i = \{x_j^i : j \in \{1,\dots,3n\}\}$. Note that $w(\mathcal{X}_i) = 3B$ and $p(\mathcal{X}_i) = 3B + 9nB \cdot i$. Now suppose $\mathcal{A}_i$ has a solution $\mathcal{A}_i^* \subset \mathcal{A}_i$. We would like to encode this solution using the following set of encoding items

$$\mathcal{X}(\mathcal{A}_i^*) = \mathcal{X}_i^* \cup \mathcal{X}_{i+1} \cup \mathcal{X}_{i+2} \cup \dots \cup \mathcal{X}_{t-1},$$

where $\mathcal{X}_i^* := \{x_i^j : a_i^j \in \mathcal{A}_i^*\}$ is the set of items corresponding to $\mathcal{A}_i^*$. An easy calculation shows that if the elements of $\mathcal{A}_i^*$ sum up to B (i.e., $\mathcal{A}_i^*$ is indeed a solution), then $w(\mathcal{X}(\mathcal{A}_i^*)) = (3t - 3i - 2) \cdot B$ and $p(\mathcal{X}(\mathcal{A}_i^*)) = (3t - 3i - 2) \cdot B + (\binom{t}{2} - \binom{i+1}{2} + \frac{i}{3}) \cdot 9nB$. The next lemma shows that the converse is also true; namely, that if there is a subset of encoding items with the above weight and profit, then there must be a solution to the i'th Restricted Subset Sum instance.

▶ **Lemma 8** (⋆). *Let $i \in \{0,\dots,t-1\}$. There exists a subset $\mathcal{X}^* \subseteq \mathcal{X}$ of encoding items with total weight*

$$w(\mathcal{X}^*) \leq (3t - 3i - 2) \cdot B$$

and total profit

$$p(\mathcal{X}^*) \geq (3t - 3i - 2) \cdot B \; + \; \left(\binom{t}{2} - \binom{i+1}{2} + \frac{i}{3} \right) \cdot 9nB$$

for our Knapsack *instance if and only if there exist a solution* $\mathcal{A}_i^* \subseteq \mathcal{A}_i$ *to the* i*'th* Restricted Subset Sum *instance.*

Lemma 8 implies that if we knew which Restricted Subset Sum instance $\mathcal{A}_i$ has a solution, then we could easily set the weight and profit of our composed Knapsack instance to encode this solution. However, we do not have prior information about index i. Furthermore, observe that as the value of i increases, both the profit and weight of the required solution decrease. Since we do not know the value of i, it would be beneficial to balance all possible choices of i in terms of weight and profit.

Thus, the remaining construction focuses on ensuring that solutions to the Knapsack instance corresponding to different $\mathcal{A}_i$'s all have the same weight and profit. In particular, the construction guarantees that any choice of i can obtain a profit of $(3t-2) \cdot B + 9 \cdot \binom{t}{2} \cdot nB$, in addition to some large constant. Considering the profit guaranteed by solutions of Lemma 8, we need to compensate for the loss of the quadratic term

$$\left(\binom{i+1}{2} - \frac{i}{3} \right) \cdot 9nB \; . \tag{1}$$

We call the term above the *compensation term of* i. It will play an important role in the remainder of our construction.

3.3 Instance Selection Gadget

We next add additional items to our Knapsack instance that will serve as an instance selection gadget. This gadget selects an instance of Restricted Subset Sum for which presumably there is a solution. The gadget consists of two types of items: The *index items* which encode an index of an Restricted Subset Sum instance $i \in \{0, \ldots, t-1\}$, and *quadratization items* that help to encode the compensation term of i given in Equation 1.

Quadratization items. The main idea behind the quadratization items is as follows: Any integer $i \in \{0, \ldots, t-1\}$ can be written as the sum

$$i = \sum_{k=0}^{\lg t - 1} i(k) \cdot 2^k,$$

for some binary values $i(0), \ldots, i(\lg t - 1) \in \{0, 1\}$. Thus, using these same $\lg t$ binary values, we can write the compensation term of i as

$$\begin{aligned}
\left(\binom{i+1}{2} - \frac{i}{3} \right) \cdot 9nB \quad &= \quad \left(0.5 \cdot i^2 + \frac{1}{6} \cdot i \right) \cdot 9nB \\
&= \quad \left(0.5 \cdot \sum_{k=0}^{\lg t - 1} \sum_{\ell=0}^{\lg t - 1} i(k) \cdot i(\ell) \cdot 2^{k+\ell} + \frac{1}{6} \sum_{k=0}^{\lg t - 1} i(k) \cdot 2^k \right) \cdot 9nB \\
&= \quad \left(9 \cdot \sum_{\substack{i(k)=1, \\ i(\ell)=1, \\ k<\ell}} 2^{k+\ell} \; + \; 4.5 \cdot \sum_{i(k)=1} 2^{k+k} \; + \; 1.5 \cdot \sum_{i(k)=1} 2^k \right) \cdot nB \; .
\end{aligned}$$

(2)

Thus, we construct $3 \cdot \binom{\lg t}{2} + \lg t$ different quadratization items, each modeling the contribution of all possible values of $i(k)$ and $i(\ell)$, $k \leq \ell \in \{0, \ldots, \lg t - 1\}$, in the last equality of Equation 2 above.

Let $Y = t^2 \cdot 3nB$, and observe that Y is larger than the total profit of all encoding items. Furthermore, let $f : \{0, \ldots, \lg t - 1\}^2 \to \{0, \ldots, \lg^2 t - 1\}$ be any bijective function. For each pair of indices k and ℓ with $0 \leq k < \ell \leq \lg t - 1$, we add three quadratization items $y^{1,0}_{k,\ell}$, $y^{0,1}_{k,\ell}$, and $y^{1,1}_{k,\ell}$ with the following weight and profit:

- $w(y^{1,0}_{k,\ell}) = p(y^{1,0}_{k,\ell}) = 3^{f(k,\ell)} \cdot Y$.
- $w(y^{0,1}_{k,\ell}) = p(y^{0,1}_{k,\ell}) = 3^{f(\ell,k)} \cdot Y$.
- $w(y^{1,1}_{k,\ell}) = (3^{f(k,\ell)} + 3^{f(\ell,k)}) \cdot Y$ and $p(y^{1,1}_{k,\ell}) = (3^{f(k,\ell)} + 3^{f(\ell,k)}) \cdot Y + 2^{k+\ell} \cdot 9nB$.

Furthermore, for each $k \in \{0, \ldots, \lg t - 1\}$, we add a single quadratization item $y^{1,1}_{k,k}$ with:

- $w(y^{1,1}_{k,k}) = 3^{f(k,k)} \cdot Y$ and $p(y^{1,1}_{k,k}) = 3^{f(k,k)} \cdot Y + 2^{k+k} \cdot 4.5nB + 2^k \cdot 1.5nB$.

We use $\mathcal{Y} = \{y^{1,0}_{k,\ell}, y^{0,1}_{k,\ell}, y^{1,1}_{k,\ell} : 0 \leq k < \ell \leq \lg t - 1\} \cup \{y^{1,1}_{k,k} : 0 \leq k \leq \lg t - 1\}$ to denote the set of all quadratization items.

The role of the additional terms that depend on Y will become clearer when we introduce the index items. But for now, one can observe that the smaller terms used in the profits of $y^{1,1}_{k,\ell}$ and $y^{1,1}_{k,k}$ allow us to encode the compensation term of i. In particular, an easy calculation using Equation 2 gives us the following useful lemma:

▶ **Lemma 9.** *Let $i \in \{0, \ldots, t-1\}$, and let $i(0), \ldots, i(t-1) \in \{0, 1\}$ be binary values such that $i = \sum_k i(k) \cdot 2^k$. Moreover, let $\mathcal{Y}_i$ denote the set of quadratization items defined by*

$$\mathcal{Y}_i = \{y^{i(k),i(\ell)}_{k,\ell} : 0 \leq k \leq \ell \leq \lg t - 1\},$$

where $y^{i(k),i(\ell)}_{k,\ell}$ is the empty item (i.e., an item with weight and profit 0) if $i(k) = i(\ell) = 0$. Then

$$p(\mathcal{Y}_i) \quad = \quad w(\mathcal{Y}_i) + \left(\binom{i+1}{2} - \frac{i}{3}\right) \cdot 9nB \ .$$

Index items. The index items ensure that only quadratization items that correspond to subsets $\mathcal{Y}_i$ as in Lemma 9 above can be picked into any solution of our Knapsack instance. In particular, the index items will encode the selection of an index $i \in \{0, \ldots, \lg t - 1\}$ that will be compatible with the selection of a subset $\mathcal{Y}_i$ of quadratization items.

Let $Z = \lg^2 t Y^2 \cdot 3^{\lg^2 t}$, and observe that Z is larger than the profit of all encoding and quadratization items in total. For each $k \in \{0, \ldots, \lg t - 1\}$, we construct two *index items* z^0_k and z^1_k corresponding to selecting either $i(k) = 0$ or $i(k) = 1$ in the binary representation $i(0), \ldots, i(\lg t - 1)$ of i. The weight and profit of these two items are defined by:

- $w(z^0_k) = p(z^0_k) = 2^k \cdot Z + \sum_{\ell=0}^{\lg t - 1} 3^{f(k,\ell)} \cdot Y$.
- $w(z^1_k) = p(z^1_k) = 2^k \cdot Z + 2^k \cdot 3B$.

We use $\mathcal{Z} = \{z^0_k, z^1_k : 0 \leq k \leq \lg t - 1\}$ to denote the set of all index items.

Let $i \in \{0, \ldots, t-1\}$, and let $i(0), \ldots, i(t-1) \in \{0, 1\}$ be binary values such that $i = \sum_k i(k) \cdot 2^k$. Observe that the set of index items $\mathcal{Z}_i$ defined by

$$\mathcal{Z}_i = \{z^{i(k)}_k : 0 \leq k \leq \lg t - 1\}$$

naturally corresponds to index i. In the lemma below, we show that due to our selection of the large value Z, any set of items of sufficiently small weight and sufficiently large profit contains a subset of index items that corresponds precisely to some index $i \in \{0, \ldots, t-1\}$. We let w_Z denote the weight function $w_Z(x) = \lfloor w(x)/Z \rfloor$, and p_Z denote the profit function $p_Z(x) = \lfloor p(x)/Z \rfloor$.

▶ **Lemma 10** (⋆). *Let $\mathcal{S} \subseteq \mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z}$ be a set of items with $w_Z(\mathcal{S}) \leq t-1$ and $p_Z(\mathcal{S}) \geq t-1$. Then there exists some $i \in \{0, \ldots, t-1\}$ for which $\mathcal{S} \cap \mathcal{Z} = \mathcal{Z}_i$.*

Now let us address the terms that depend on Y in the profit and weight of the instance selection items. Define T to be the constant $T := \sum_{k=0}^{\lg^2 t-1} 3^k$. Now consider some set $\mathcal{Z}_i$ of index items corresponding to index $i \in \{0, \ldots, t-1\}$. Let w_Y denote the weight function $w_Y(x) = \lfloor (w(x) - Z \cdot w_Z(x))/Y \rfloor$, and similarly define the profit function p_Y as $p_Y(x) = \lfloor (p(x) - Z \cdot p_Z(x))/Y \rfloor$. Then one can observe that, by construction of the weights and profits above, we have that both $w_Y(\mathcal{Z}_i) + w_Y(\mathcal{Y}_i)$ and $p_Y(\mathcal{Z}_i) + p_Y(\mathcal{Y}_i)$ equal T. Moreover, any other set of items with w_Y-weight at most T has lesser profit. This ensures a compatible selection of the index items and the quadratization items, formally proven in the lemma below.

▶ **Lemma 11.** *Let $\mathcal{S} \subseteq \mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z}$ be a set of items with weight $w_Z(\mathcal{S}) \leq (t-1)$, $w_Y(\mathcal{S}) \leq T$, $p_Z(\mathcal{S}) \geq (t-1)$, $p_Y(\mathcal{S}) \geq T$, and there is no set $\mathcal{S}' \subseteq \mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z}$ with $w(\mathcal{S}') \leq w(\mathcal{S})$ and $p(\mathcal{S}') > p(\mathcal{S})$. Then $\mathcal{S} \cap \mathcal{Y} = \mathcal{Y}_i$ and $\mathcal{S} \cap \mathcal{Z} = \mathcal{Z}_i$ for some $i \in \{0, \ldots, t-1\}$.*

Proof. Let $\mathcal{Z}^* = \mathcal{S} \cap \mathcal{Z}$ and $\mathcal{Y}^* = \mathcal{S} \cap \mathcal{Y}$. As $w_Z(\mathcal{S}) \leq (t-1)$ and $p_Z(\mathcal{S}) \geq (t-1)$, by Lemma 10 we have that $\mathcal{Z}^* = \mathcal{Z}_i$ for some $i \in \{0, \ldots, t-1\}$. Thus, to complete the proof, we focus on showing that $\mathcal{Y}^* = \mathcal{Y}_i$. Let $i(0), \ldots, i(\lg t - 1) \in \{0, 1\}$ be such that $i = \sum_k i(k) \cdot 2^k$.

Observe that by the construction of the weights and profits of the index items, we have $w_Y(\mathcal{Z}_i) = p_Y(\mathcal{Z}_i) = \sum_{\alpha(k)=0} \sum_\ell 3^{f(k,\ell)}$. From this, one can see that both $w_Y(\mathcal{Z}_i) + w_Y(\mathcal{Y}^*)$ and $w_Y(\mathcal{Z}_i) + p_Y(\mathcal{Y}^*)$ equal

$$\Big(\sum_{\substack{i(k)=0,\\ 0\leq \ell \leq \lg t-1}} 3^{f(k,\ell)} \Big) + \Big(\sum_{y^{1,0}_{k,\ell} \in \mathcal{Y}_i} 3^{f(k,\ell)} + \sum_{y^{0,1}_{k,\ell} \in \mathcal{Y}_i} 3^{f(\ell,k)} + \sum_{y^{1,1}_{k,\ell} \in \mathcal{Y}_i} (3^{f(k,\ell)} + 3^{f(\ell,k)}) + \sum_{y^{1,1}_{k,\ell} \in \mathcal{Y}_i} 3^{f(k,k)} \Big) =$$
$$\Big(\sum_{\substack{i(k)=0,\\ 0\leq \ell \leq \lg t-1}} 3^{f(k,\ell)} \Big) + \Big(\sum_{\substack{i(k)=1,\\ i(\ell)=0,\\ k<\ell}} 3^{f(k,\ell)} + \sum_{\substack{i(k)=1,\\ i(\ell)=0,\\ k>\ell}} 3^{f(k,\ell)} + \sum_{\substack{i(k)=1,\\ i(\ell)=1,\\ k\neq\ell}} 3^{f(k,\ell)} + \sum_{i(k)=1} 3^{f(k,k)} \Big) \ .$$

Note that as $w_Y(\mathcal{Z}_i) + w_Y(\mathcal{Y}^*) = w_Y(\mathcal{S}) \leq T$, the sum above is bounded from above by T. Moreover, by Lemma 6, the only way this reaches the bound with equality is if we have

$$w_Y(\mathcal{Y}^*) = \sum_{\substack{i(k)=1,\\ i(\ell)=0,\\ k<\ell}} 3^{f(k,\ell)} + \sum_{\substack{i(k)=1,\\ i(\ell)=0,\\ k>\ell}} 3^{f(k,\ell)} + \sum_{\substack{i(k)=1,\\ i(\ell)=1,\\ k\neq\ell}} 3^{f(k,\ell)} + \sum_{i(k)=1} 3^{f(k,k)} = \sum_{\substack{i(k)=0,\\ 0\leq \ell \leq \lg t-1}} 3^{f(k,\ell)} \ ,$$

which then gives us

$$w_Y(\mathcal{Z}_i) + w_Y(\mathcal{Y}^*) = \sum_{\substack{i(k)=0,\\ 0\leq \ell \leq \lg t-1}} 3^{f(k,\ell)} + \sum_{\substack{i(k)=1,\\ 0\leq \ell \leq \lg t-1}} 3^{f(k,\ell)} = \sum_{k=0}^{\lg^2 t-1} 3^k = T \ .$$

(Here, the penultimate equality follows because $f(\cdot,\cdot)$ is bijective.)

Note that by construction and Lemma 6, the only sets of quadratization items $\mathcal{Y}^*$ with $w_Y(\mathcal{Y}^*) = p_Y(\mathcal{Y}^*) = \sum_{\alpha(k)=1} \sum_\ell 3^{f(k,\ell)}$ are either $\mathcal{Y}_i$, or any set of quadratization items obtained from $\mathcal{Y}_i$ by replacing some $y^{1,1}_{k,\ell} \in \mathcal{Y}_i$ with $y^{1,0}_{k,\ell}$ and $y^{0,1}_{k,\ell}$. If $\mathcal{S}$ contained $y^{1,0}_{k,\ell}$ and $y^{0,1}_{\ell,k}$ for some $0 \leq k < \ell \leq \lg t - 1$, then $\mathcal{S}' := (\mathcal{S} \setminus \{y^{1,0}_{k,\ell}, y^{0,1}_{k,\ell}\}) \cup \{y^{1,1}_{k,\ell}\}$ satisfies $w(\mathcal{S}') = w(\mathcal{S})$ and $p(\mathcal{S}') > p(\mathcal{S})$, a contradiction to the definition of $\mathcal{S}$. Thus, we conclude that $\mathcal{Y}^* = \mathcal{Y}_i$ and the lemma is proven. ◀

Table 1 The weights and profits of the items used in the proof of Theorem 2 for parameter $w_{\#}$. The three large constants used in the proof are $X = 3tn \cdot B_n$, $Y = t^2 \cdot 3nB$, and $Z = \lg^2 tY^2 \cdot 3^{\lg^2 t}$.

Item	Weight	Profit	Index Range
z_k^1	$2^k \cdot Z + 2^k \cdot 3B$	$2^k \cdot Z + 2^k \cdot 3B$	$0 \leq k \leq \lg t - 1$
z_k^0	$2^k \cdot Z + \sum_\ell 3^{f(k,\ell)} \cdot Y$	$2^k \cdot Z + \sum_\ell 3^{f(k,\ell)} \cdot Y$	$0 \leq k \leq \lg t - 1$
$y_{k,\ell}^{1,0}$	$3^{f(k,\ell)} \cdot Y$	$3^{f(k,\ell)} \cdot Y$	$0 \leq k < \ell \leq \lg t - 1$
$y_{k,\ell}^{0,1}$	$3^{f(\ell,k)} \cdot Y$	$3^{f(\ell,k)} \cdot Y$	$0 \leq k < \ell \leq \lg t - 1$
$y_{k,\ell}^{1,1}$	$(3^{f(k,\ell)} + 3^{f(\ell,k)}) \cdot Y$	$(3^{f(k,\ell)} + 3^{f(\ell,k)}) \cdot Y + 2^{k+\ell} \cdot 9nB$	$0 \leq k < \ell \leq \lg t - 1$
$y_{k,k}^{1,1}$	$3^{f(k,k)} \cdot Y$	$3^{f(k,k)} \cdot Y + 2^{k+k} \cdot 4.5nB + 2^k \cdot 1.5nB$	$0 \leq k \leq \lg t - 1$
x_j^i	$X + a_j^i$	$X + a_j^i + i \cdot 3B$	$0 \leq i \leq t-1$, $1 \leq j \leq 3n$

3.4 Correctness

Our entire Knapsack instance consists of all items $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z}$. An overview of the weight and profit of each item can be found in Table 1. We set the weight W of the Knapsack instance to

$$W := (t-1) \cdot Z \; + \; T \cdot Y \; + \; (3t-2) \cdot B$$

and the desired profit P to

$$\begin{aligned} P &:= W \; + \; \binom{t}{2} \cdot 9nB \\ &= (t-1) \cdot Z \; + \; T \cdot Y \; + \; (3t-2) \cdot B \; + \; \binom{t}{2} \cdot 9nB \; . \end{aligned}$$

In the next two lemmas below we prove the correctness of our constructed composition.

▶ **Lemma 12.** *If $\mathcal{A}_i$ is a "yes"-instance of* Restricted Subset Sum *for some $i \in \{0, \ldots, t-1\}$ then there exists a subset of items $\mathcal{S} \subseteq \mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z}$ with $w(\mathcal{S}) \leq W$ and $p(\mathcal{S}) \geq P$.*

Proof. Suppose $\mathcal{A}_i$ is a "yes"-instance of Restricted Subset Sum for some $i \in \{0, \ldots, t-1\}$, and let $i(0), \ldots, i(\lg t - 1) \in \{0,1\}$ be such that $i = \sum_k i(k) \cdot 2^k$. Then $w(\mathcal{X}(\mathcal{A}_i)) = (3t-3i-2) \cdot B$ as discussed in Section 3.2. Furthermore, $w(\mathcal{Y}_i) = w_Y(\mathcal{Y}_i) \cdot Y = (T - w_Y(\mathcal{Z}_i)) \cdot Y$ as is shown in the proof of Lemma 11. Finally, we have $w_Z(\mathcal{Z}_i) = (t-1) \cdot Z$, and

$$w(\mathcal{Z}_i) \; = \; w_Z(\mathcal{Z}_i) + w_Y(\mathcal{Z}_i) + \sum_{i(k)=1} 2^k \cdot 3B \; = \; (t-1) \cdot Z + w_Y(\mathcal{Z}_i) \cdot Y + i \cdot 3B \; .$$

So altogether we have

$$\begin{aligned} w(\mathcal{X}(\mathcal{A}_i) \cup \mathcal{Y}_i \cup \mathcal{Z}_i) \; &= \; (3t - 3i - 2) \cdot B \; + \; (T - w_Y(\mathcal{Z}_i)) \cdot Y \\ &\quad + (t-1) \cdot Z \; + \; w_Y(\mathcal{Z}_i) \cdot Y \; + \; i \cdot 3B \\ &= (t-1) \cdot Z \; + \; T \cdot Y \; + \; (3t-2) \cdot B \; = W \; . \end{aligned}$$

Let us next calculate the profit of $\mathcal{X}(\mathcal{A}_i) \cup \mathcal{Y}_i \cup \mathcal{Z}_i$. Recall that $p(\mathcal{X}(\mathcal{A}_i)) = w(\mathcal{X}(\mathcal{A}_i)) + (\binom{t}{2} - \binom{i+1}{2} + \frac{i}{3}) \cdot 9nB$ as discussed in Section 3.2. By Lemma 9 we have $p(\mathcal{Y}_i) = w(\mathcal{Y}_i) + (\binom{i+1}{2} - \frac{i}{3}) \cdot 9nB$, and by construction we have $p(\mathcal{Z}_i) = w(\mathcal{Z}_i)$. Thus, altogether we have

$$\begin{aligned} p(\mathcal{X}(\mathcal{A}_i) \cup \mathcal{Y}_i \cup \mathcal{Z}_i) &= w(\mathcal{X}(\mathcal{A}_i)) + (\binom{t}{2} - \binom{i+1}{2} + \frac{i}{3}) \cdot 9nB \\ &\quad + w(\mathcal{Y}_i) + (\binom{i+1}{2} - \frac{i}{3}) \cdot 9nB \ + \ w(\mathcal{Z}_i) \\ &= w(\mathcal{X}(\mathcal{A}_i) \cup \mathcal{Y}_i \cup \mathcal{Z}_i) + \binom{t}{2} \cdot 9nB = P \ . \end{aligned}$$

Thus the set of items $\mathcal{S} = \mathcal{X}(\mathcal{A}_i) \cup \mathcal{Y}_i \cup \mathcal{Z}_i$ is a solution for our Knapsack instance, and the lemma is proven. ◀

▶ **Lemma 13.** *If there exists a subset of items $\mathcal{S} \subseteq \mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z}$ with $w(\mathcal{S}) \leq W$ and $p(\mathcal{S}) \geq P$ then there is some $i \in \{0, \ldots, t-1\}$ for which $\mathcal{A}_i$ is a "yes"-instance of* Restricted Subset Sum.

Proof. Let $\mathcal{S}$ be a solution with $w(\mathcal{S}) \leq W$ and $p(\mathcal{S}) \geq P$. Let $\mathcal{X}^* = \mathcal{S} \cap \mathcal{X}$, $\mathcal{Y}^* = \mathcal{S} \cap \mathcal{Y}$, and $\mathcal{Z}^* = \mathcal{S} \cap \mathcal{Z}$. Then as $w(\mathcal{S}) \leq W < t \cdot Z$ and $p(\mathcal{S}) \leq P < t \cdot Z$, we have by Lemma 10 that $\mathcal{Z}^* = \mathcal{Z}_i$ for some $i \in \{0, \ldots, t-1\}$. Assume, without loss of generality, that $\mathcal{S}$ is of maximal profit among all solutions with weight at most $w(\mathcal{S})$ (i.e., there is no set $\mathcal{S}'$ with $w(\mathcal{S}') \leq w(\mathcal{S})$ and $p(\mathcal{S}') > p(\mathcal{S})$. Then, by Lemma 11, we have $\mathcal{Y}^* = \mathcal{Y}_i$. As shown in the proof of Lemma 12, we have $w(\mathcal{Z}_i) + w(\mathcal{Y}_i) = (t-1) \cdot Z + T \cdot Y + i \cdot 3B$. Thus,

$$w(\mathcal{X}^*) \leq W - w(\mathcal{Z}_i) - w(\mathcal{Y}_i) = (3t - 3i - 2) \cdot B \ .$$

Moreover, by Lemma 9 we have $p(\mathcal{Y}_i) = w(\mathcal{Y}_i) + (\binom{i+1}{2} - \frac{i}{3}) \cdot 9nB$, and by construction we have $p(\mathcal{Z}_i) = w(\mathcal{Z}_i)$. Thus,

$$\begin{aligned} p(\mathcal{X}^*) &\geq P - p(\mathcal{Z}_i) - p(\mathcal{Y}_i) \\ &= P - w(\mathcal{Z}_i) - w(\mathcal{Y}_i) - \left(\binom{i+1}{2} - \frac{i}{3}\right) \cdot 9nB \\ &= P - (t-1) \cdot Z - T \cdot Y - i \cdot 3B - \left(\binom{i+1}{2} - \frac{i}{3}\right) \cdot 9nB \\ &= (3t - 3i - 2) \cdot B + \left(\binom{t}{2} - \binom{i+1}{2} + \frac{i}{3}\right) \cdot 9nB \ . \end{aligned}$$

It therefore follows by Lemma 8 that instance $\mathcal{A}_i$ is indeed a "yes"-instance of Restricted Subset Sum, and the lemma follows. ◀

Proof of Theorem 2. We presented above an algorithm that composes any t instances $\mathcal{A}_0, \ldots, \mathcal{A}_{t-1}$ of Restricted Subset Sum into a single instance of Knapsack in polynomial-time. By Lemmas 12 and 13, the constructed Knapsack instance is a "yes"-instance if and only if $\mathcal{A}_i$ is "yes"-instance of Restricted Subset Sum for some $i \in \{0, \ldots, t-1\}$. Observe that total number of different weights in our constructed Knapsack instance is

$$w_\# \leq |\widetilde{\mathcal{A}}_n| + |\mathcal{Y}| + |\mathcal{Z}| = O(n^3 + \lg^2 t) \ .$$

Thus our algorithm fulfills all requirements of a composition algorithm, as given in Definition 4. The proof for $w_\#$ then follows by a direct application of Theorem 5. The statement for $p_\#$ follows by applying the reduction from Polak et al. [45, Chapter 4] which reduces an instance with $w_\# = k$ different item weights to an instance with $p_\# = k$ different item profits. ◀

4 Polynomial Kernel for Parameter $w_\# \cdot p_\#$

In this section we present a polynomial kernel for Knapsack parameterized by $w_\# + p_\#$, thereby proving Theorem 3. Our kernel is a direct generalization of the polynomial kernel for Subset Sum parameterized by $a_\#$ of Etscheid et al. [22].

The presented kernel utilizes two classic results: First, we use the fact that integer programming is fixed-parameter tractable with respect to the number of variables (this was first shown by Lenstra [39], and the currently best known running time with respect to the number of variable is due to Reis and Rothvoss [46]):

▶ **Theorem 14** ([46]). Integer Linear Programming *with input size s (i.e., the number of bits needed to encode the instance) and n variables can be solved in $2^{O(n \lg \lg n)} \cdot s^{O(1)}$ time.*

Second, we use the following theorem by Frank and Tardos [26]:

▶ **Theorem 15** ([26]). *There is an algorithm that, given a vector $w \in \mathbb{Q}^r$ and a natural number N, computes in polynomial time a vector $\overline{w} \in \mathbb{Q}^r$ with $\|w\|_\infty \leq 2^{4r^3} \cdot N^{r^2+2r}$ and $\operatorname{sign}(w \cdot b) = \operatorname{sign}(\overline{w} \cdot b)$ for every $b \in \mathbb{Z}^r$ with $\|b\|_1 \leq N$.*

Proof of Theorem 3. Let $w_1, \ldots, w_{w_\#}$ be the different weights and $p_1, \ldots, p_{p_\#}$ be the different profits in a given Knapsack instance with n items. We denote by $n_{i,j}$ for $i \in \{1, \ldots, w_\#\}$ and $j \in \{1, \ldots, p_\#\}$ the number of items with weight w_i and profit p_j. First note that the following is an ILP formulation of Knapsack with $w_\# \cdot p_\#$ many variables $x_{i,j}$, one for each $i \in \{1, \ldots, w_\#\}$ and $j \in \{1, \ldots, p_\#\}$, and two inequalities:

$$\begin{aligned} \sum_{i=1}^{w_\#} \sum_{j=1}^{p_\#} x_{i,j} \cdot w_i &\leq W \\ \sum_{i=1}^{w_\#} \sum_{j=1}^{p_\#} x_{i,j} \cdot p_j &\geq P \\ x_{i,j} &\in \{0, 1, \ldots, n_{i,j}\}. \end{aligned} \tag{3}$$

By Theorem 14, if $w_\# \cdot p_\# \cdot \lg\lg(w_\# \cdot p_\#) \leq \lg n$ (recall that n denotes the total number of items of the Knapsack instance), then the instance can be solved in polynomial time using ILP (3). Thus, devising a polynomial kernel in this case is trivial. So assume that $w_\# \cdot p_\# \cdot \lg(w_\# \cdot p_\#) > \lg n$. To reduce the encoding length of ILP (3), we apply Theorem 15 to the first two inequalities of ILP (3) as follows: let w^* be a $w_\# \cdot p_\#$-dimensional vector whose $(p_\# \cdot (i-1) + j)$'th component equals w_i, for each $i \in \{1, \ldots, w_\#\}$ and $j \in \{1, \ldots, p_\#\}$. We apply Theorem 15 to the $(w_\# \cdot p_\# + 1)$-dimensional vector $w := (w^*, -W)$ and $N := n+1$, resulting in a vector $(\overline{w}^*, -\overline{W})$. Similarly, let p^* be a $w_\# \cdot p_\#$-dimensional vector whose $(p_\# \cdot (i-1) + j)$'th component equals p_j, for $i \in \{1, \ldots, w_\#\}$ and $j \in \{1, \ldots, p_\#\}$. We apply Theorem 15 to the vector $w := (-p^*, P)$ and $N := n+1$, resulting in a vector $(-\overline{p}^*, \overline{P})$. By Theorem 15, ILP (3) is equivalent to the following ILP (4):

$$\begin{aligned} \overline{w}^* \cdot x &\leq \overline{W} \\ \overline{p}^* \cdot x &\geq \overline{P} \\ x_{i,j} &\in \{0, 1, \ldots, n_{i,j}\} \end{aligned} \tag{4}$$

By Theorem 15, each number from ILP (4) has encoding length $O(r^3 + r^2 \lg n)$ for $r = w_\# \cdot p_\#$. Since $\lg n \leq r \cdot \lg r$, it follows that the size of ILP (4) is $O(r^4 \cdot \lg r)$. As Knapsack is NP-complete, and Integer Linear Programming is in NP (see e.g. [15]), we can reduce ILP (4) in polynomial time to an instance of Knapsack. The resulting Knapsack instance is equivalent to the original instance, and has size polynomial in $r = w_\# \cdot p_\#$. ◀

Using the reduction from Unbounded Knapsack (i.e., the variation of Knapsack where a solution is a multiset of items instead of a regular set) to Knapsack (essentially also used e.g. by Etscheid et al. [22, Theorem 12]), one can also make the reduction from ILP (4) to Knapsack explicit, resulting in a kernel of size $\widetilde{O}((w_{\#} \cdot p_{\#})^5)$; see the full version [28] for details.

References

1 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. Seth-based lower bounds for subset sum and bicriteria path. *ACM Trans. Algorithms*, 18(1):6:1–6:22, 2022. doi:10.1145/3450524.

2 Kyriakos Axiotis and Christos Tzamos. Capacitated dynamic programming: Faster knapsack and graph algorithms. In *Proc. of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 19:1–19:13, 2019.

3 MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Saeed Seddighin, and Clifford Stein. Fast algorithms for knapsack via convolution and prediction. In *Proc. of the 50th ACM Symposium on the Theory Of Computing (STOC)*, pages 1269–1282, 2018.

4 Richard E. Bellman. *Dynamic programming*. Princeton University Press, 1957.

5 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009. doi:10.1016/J.JCSS.2009.04.001.

6 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014. doi:10.1137/120880240.

7 Ernest F. Brickell and Andrew M. Odlyzko. Cryptanalysis: A survey of recent results. *Proceedings of the IEEE*, 76(5):578–593, 1988.

8 Karl Bringmann. Knapsack with small items in near-quadratic time. In *Proc. of the 56th ACM Symposium on the Theory of Computing (STOC)*, 2024. To appear.

9 Karl Bringmann and Alejandro Cassis. Faster knapsack algorithms via bounded monotone min-plus-convolution. In *Proc. of the 49th International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 31:1–31:21, 2022.

10 Karl Bringmann and Alejandro Cassis. Faster 0-1-knapsack via near-convex min-plus-convolution. In *Proc. of the 31st Annual European Symposium on Algorithms (ESA)*, pages 24:1–24:16, 2023. doi:10.4230/LIPICS.ESA.2023.24.

11 Harry Buhrman, Bruno Loff, and Leen Torenvliet. Hardness of approximation for knapsack problems. *Theory of Computing Systems*, 56(2):372–393, 2015.

12 Timothy M. Chan. Approximation schemes for 0-1 knapsack. In *1st Symposium on Simplicity in Algorithms (SOSA)*, pages 5:1–5:12, 2018. doi:10.4230/OASICS.SOSA.2018.5.

13 Lin Chen, Jiayi Lian, Yuchen Mao, and Guochuan Zhang. Faster algorithms for bounded knapsack and bounded subset sum via fine-grained proximity results. In *Proc. of the 2024 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4828–4848, 2024. doi:10.1137/1.9781611977912.171.

14 Benny Chor and Ronald R. Rivest. A knapsack-type public key cryptosystem based on arithmetic in finite fields. *IEEE Transactions on Information Theory*, 34(5):901–909, 1988.

15 William J. Cook, A. M. H. Gerards, Alexander Schrijver, and Éva Tardos. Sensitivity theorems in integer linear programming. *Mathematical Programming*, 34(3):251–264, 1986. doi:10.1007/BF01582230.

16 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

17 Marek Cygan, Marcin Mucha, Karol Węgrzycki, and Michał Włodarczyk. On problems equivalent to (min, +)-convolution. *ACM Transactions on Algorithms*, 15(1):14:1–14:25, 2019.

18 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *Journal of the ACM*, 61(4):23:1–23:27, 2014.

19 Mingyang Deng, Ce Jin, and Xiao Mao. Approximating knapsack and partition via dense subset sums. In *Proc. of the 2023 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2961–2979, 2023. doi:10.1137/1.9781611977554.CH113.

20 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization lower bounds through colors and ids. *ACM Transactions on Algorithms*, 11(2):13:1–13:20, 2014.

21 Andrew Drucker. New limits to classical and quantum instance compression. *SIAM Journal on Computing*, 44(5):1443–1479, 2015.

22 Michael Etscheid, Stefan Kratsch, Matthias Mnich, and Heiko Röglin. Polynomial kernels for weighted problems. *Journal of Computer and System Sciences*, 84:1–10, 2017. doi:10.1016/J.JCSS.2016.06.004.

23 Michael R. Fellows, Serge Gaspers, and Frances A. Rosamond. Parameterizing by the number of numbers. *Theory of Computing Systems*, 50(4):675–693, 2012.

24 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.

25 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *Journal of Computer and System Sciences*, 77(1):91–106, 2011.

26 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987. doi:10.1007/BF02579200.

27 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

28 Klaus Heeger, Danny Hermelin, Matthias Mnich, and Dvir Shabtay. No polynomial kernels for knapsack. *CoRR*, 2023. doi:10.48550/arXiv.2308.12593.

29 Danny Hermelin, Shlomo Karhi, Michael L. Pinedo, and Dvir Shabtay. New algorithms for minimizing the weighted number of tardy jobs on a single machine. *Annals of Operations Research*, 298(1):271–287, 2021.

30 Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of Cryptology*, 9(4):199–216, 1996.

31 Klaus Jansen, Felix Land, and Kati Land. Bounding the running time of algorithms for scheduling and packing problems. *SIAM Journal on Discrete Mathematics*, 30(1):343–366, 2016.

32 Ce Jin. An improved FPTAS for 0-1 knapsack. In *Proc. of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 76:1–76:14, 2019. doi:10.4230/LIPICS.ICALP.2019.76.

33 Ce Jin. 0-1 knapsack in nearly quadratic time. In *Proc. of the 56th ACM Symposium on the Theory of Computing (STOC)*, 2024. To appear.

34 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987. doi:10.1287/MOOR.12.3.415.

35 Richard M. Karp. Reducibility among combinatorial problems. In *Proc. of a symposium on the Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

36 Hans Kellerer and Ulrich Pferschy. Improved dynamic programming in connection with an FPTAS for the knapsack problem. *Journal of Combinatorial Optimization*, 8(1):5–11, 2004.

37 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.

38 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In *Proc. of the 44th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 21:1–21:15, 2017. doi:10.4230/LIPICS.ICALP.2017.21.

39 Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983. doi:10.1287/MOOR.8.4.538.

40 Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

41 Ralph Merkle and Martin Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24(5):525–530, 1978.

42 Andrew M. Odlyzko. The rise and fall of knapsack cryptosystems. *Cryptology and Computational Number Theory*, 42:75–88, 1990.

43 Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 2016.

44 David Pisinger. Linear time algorithms for knapsack problems with bounded weights. *Journal of Algorithms*, 32:1–14, 1999.

45 Adam Polak, Lars Rohwedder, and Karol Węgrzycki. Knapsack and subset sum with small items. In *Proc. of the 48th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 106:1–106:19, 2021.

46 Victor Reis and Thomas Rothvoss. The subspace flatness conjecture and faster integer programming. In *Proc. of the 64th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 974–988, 2023. doi:10.1109/FOCS57990.2023.00060.

47 Chee-Keng Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science*, 26:287–300, 1983. doi:10.1016/0304-3975(83)90020-8.

The k-Opt Algorithm for the Traveling Salesman Problem Has Exponential Running Time for $k \geq 5$

Sophia Heimann
Research Institute for Discrete Mathematics, University of Bonn, Germany

Hung P. Hoang
Algorithms and Complexity Group, Faculty of Informatics, TU Wien, Austria

Stefan Hougardy
Research Institute for Discrete Mathematics and Hausdorff Center for Mathematics, University of Bonn, Germany

Abstract

The k-Opt algorithm is a local search algorithm for the Traveling Salesman Problem. Starting with an initial tour, it iteratively replaces at most k edges in the tour with the same number of edges to obtain a better tour. Krentel (FOCS 1989) showed that the Traveling Salesman Problem with the k-Opt neighborhood is complete for the class PLS (polynomial time local search) and that the k-Opt algorithm can have exponential running time for any pivot rule. However, his proof requires $k \gg 1000$ and has a substantial gap. We show the two properties above for a much smaller value of k, addressing an open question by Monien, Dumrauf, and Tscheuschner (ICALP 2010). In particular, we prove the PLS-completeness for $k \geq 17$ and the exponential running time for $k \geq 5$.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Traveling Salesman Problem, k-Opt algorithm, PLS-completeness

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.84

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2402.07061` [7]

Funding *Hung P. Hoang*: Austrian Science Foundation (FWF, project Y1329 START-Programm). *Stefan Hougardy*: Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2047/1 – 390685813.

Acknowledgements This work was initiated at the "Discrete Optimization" trimester program of the Hausdorff Institute of Mathematics, University of Bonn, Germany in 2021. We would like to thank the organizers for providing excellent working conditions and inspiring atmosphere.

1 Introduction

The well-known Traveling Salesman Problem (TSP) consists of finding a spanning cycle of an edge weighted complete graph, such that the total edge weight of the cycle is the smallest possible. A popular heuristic for this problem is a local search algorithm called k-Opt. Starting with an arbitrary tour, it iteratively replaces at most k edges in the tour with the same number of edges, as long as the resulting tour has smaller total edge weight. We define TSP/k-Opt to be the problem of finding a local optimum for a TSP instance with the k-Opt algorithm.

A fundamental question in the area of local search algorithms is to determine the number of iterations a given local search algorithm may need in the worst case. A local search algorithm with a specified pivot rule has the *is-exp* property if there exist problem instances and initial solutions for which the local search algorithm requires an exponential number of iterations. For example, it is well known that the Simplex algorithm for linear programming

has the is-exp property for many different pivot rules [12, 9, 1, 6]. For TSP, Chandra, Karloff, and Tovey [2] showed that TSP/k-Opt has the is-exp property. This even holds for Euclidean TSP with the 2-Opt neighborhood [5].

For the Simplex algorithm it is not known whether there exists a pivot rule that guarantees a polynomial number of iterations. This is one of the most important open problems in the area of linear programming. Contrary to this Krentel [13] proved in 1989 that for sufficiently large values of k, TSP/k-Opt exhibits the *all-exp* property, that is, the k-Opt algorithm requires an exponential number of iterations to find a local optimum, for all possible pivot rules and for infinitely many pairs of a TSP instance and an initial tour. Krentel estimated that his proof yields a value for k between 1,000 and 10,000. By using a straight forward way to implement some missing details in Krentel's proof it was recently shown that his proof yields the value 14,208 for k [8].

Following Krentel's paper there have been claims in other papers [10, 22] through private communication with Krentel that a careful analysis of the original proof can bring down the value to $k = 8$ and conceivably to $k = 6$. However, there has been no available written proof for these claims. In fact, up to date, the 1989 paper of Krentel [13] is the only paper on the topic. Consequently, Monien, Dumrauf, and Tscheuschner [16] posed an open question on the complexity of TSP/k-Opt for $k \ll 1000$.

In this paper, we show that TSP/k-Opt has the all-exp property for much smaller k:

▶ **Theorem 1.** *TSP/k-Opt has the all-exp property for $k \geq 5$.*

Our proof of Theorem 1 is based on a new reduction from the bounded degree Max-Cut problem to TSP (see Section 3) which involves the construction of so called *parity gadgets* (see Section 3.1). With a first such approach we are able to prove the all-exp property of TSP/k-Opt for $k \geq 13$ (see Section 4). To lower the value of k additional ideas are required. First, we exploit the structure of a recent construction of Michel and Scott [15] of a degree-four bounded Max-Cut instance with the all-exp property under the flip neighborhood. Second, we show how to use global properties of our overall reduction to relax some local conditions on our parity gadgets. Combining these two ideas we achieve the value $k \geq 9$ (see Section 5). To arrive at our final result for $k \geq 5$ we have to modify the construction of Michel and Scott [15]. Moreover, we have to combine our parity gadgets with so called *double gadgets* and use a labeling scheme to assign different gadgets at different places in the reduction. These results we present in Section 6.

The second main contribution of our paper is a proof of the following result:

▶ **Theorem 2.** *TSP/k-Opt is PLS-complete for $k \geq 17$.*

The complexity class PLS and the notion of PLS-completeness (for definitions see Section 2) were introduced in 1988 by Johnson, Papdimitriou, and Yannakakis [11] to capture the observation that for many NP-hard problems it is not only difficult to compute a global optimum but even computing a local optimum is also hard. Examples of such problems are the Maximum Satisfiability problem [14], Max-Cut [21], and Set Cover [3]. The PLS-completeness of a problem means that a polynomial time algorithm to find a local optimum for that problem would imply polynomial time algorithms for finding a local optimum for all problems in PLS.

The PLS-completeness of TSP/k-Opt was proved by Krentel [13] for $k \gg 1000$. However, his proof has a substantial gap as he assumes that edges of weight infinity cannot occur in a local optimum. We present in Section 7 the first rigorous proof for the PLS-completeness of TSP/k-Opt and at the same time drastically lower the value of k from Krentel's $k \gg 1000$ [13]

to $k \geq 17$. Our proof uses several of the ideas used in our proof for Theorem 1. But in this case we need to take more care on the order in which the parity gadgets are plugged together in our construction. In addition, we show in Lemma 14 how to assign specific weights to the non-edges in our construction to prove that no local optimum can contain such an edge. We achieve this by defining a weight assignment that exploits the special structure of the TSP instance resulting from our PLS-reduction. This is the first rigorous proof of such a result for the k-Opt algorithm and there seems not to be a generic way to prove it for arbitrary TSP instances (as for example those constructed by Krentel [13]).

2 Preliminaries

2.1 Local search problems and the class PLS

A *local search problem* P is an optimization problem that consists of a set of instances D_P, a finite set of (feasible) solutions $F_P(I)$ for each instance $I \in D_P$, an objective function f_P that assigns an integer value to each instance $I \in D_P$ and solution $s \in F_P(I)$, and a neighborhood $N_P(s, I) \subseteq F_P(I)$ for each solution $s \in F_P(I)$. The size of every solution $s \in F_P(I)$ is bounded by a polynomial in the size of I. The goal is to find a *locally optimal solution* for a given instance I; that is, a solution $s \in F_P(I)$, such that no solution $s' \in N_P(s, I)$ yields a better objective value than $f_P(s, I)$. Formally, this means, for all $s' \in N_P(s, I)$, $f_P(s, I) \leq f_P(s', I)$ if P is a minimization problem, and $f_P(s, I) \geq f_P(s', I)$ if P is a maximization problem.

A *standard local search algorithm* for an instance I proceeds as follows. It starts with some initial solution $s \in F_P(I)$. Then it iteratively visits a neighbor with better objective value, until it reaches a local optimum. If a solution has more than one better neighbor, the algorithm has to choose one by some prespecified rule, often referred as a *pivot rule*.

A local search problem P has the *all-exp* property, if there are infinitely many pairs of an instance I of D_P and an initial solution $s \in F_P(I)$, for which the standard local search algorithm always needs an exponential number of iterations for all possible pivot rules.

A local search problem P is in the class *PLS* [11], if there are three polynomial time algorithms A_P, B_P, C_P such that

- Given an instance $I \in D_P$, A_P returns a solution $s \in F_P(I)$;
- Given an instance $I \in D_P$ and a solution $s \in F_P(I)$, B_P computes the objective value $f_P(s, I)$ of s; and
- Given an instance $I \in D_P$ and a solution $s \in F_P(I)$, C_P returns a neighbor of s with strictly better objective value, if it exists, and "locally optimal", otherwise.

A *PLS-reduction* from a problem $P \in$ PLS to a problem $Q \in$ PLS is a pair of polynomial-time computable functions h and g that satisfy:

1. Given an instance $I \in D_P$, h computes an instance $h(I) \in D_Q$; and
2. Given an instance $I \in D_P$ and a solution $s_q \in F_Q(h(I))$, g returns a solution $s_p \in F_P(I)$ such that if s_q is a local optimum for $h(I)$, then s_p is a local optimum for I.

A problem $Q \in$ PLS is *PLS-complete* [11] if for every problem $P \in$ PLS, there exists a PLS reduction from P to Q.

2.2 TSP/k-Opt

A *spanning cycle*, a *Hamiltonian cycle*, or a *tour* of an undirected graph is a cycle that contains all vertices of the graph.

A TSP instance is a tuple (G, w), where G is a complete undirected graph (V, E), and $w : E \to \mathbb{R}_{\geq 0}$ is a function that assigns a nonnegative weight to each edge of G. The goal is to find a tour of G that minimizes the sum of edge weights in the tour. The definition of the class PLS requires that we have a polynomial time algorithm to find *some* solution. For complete graphs such an algorithm certainly exists. If the graph is not complete then because of the NP-completeness of the Hamiltonian cycle problem we do not know such an algorithm.

A *swap* is a tuple (E_1, E_2) of subsets $E_1, E_2 \subseteq E$, $|E_1| = |E_2|$. We say that it is a swap of $|E_1|$ edges. If $|E_1| \leq k$ for some k, then we call it a k-swap. Performing a swap (E_1, E_2) from a subgraph G' of G refers to the act of removing E_1 from G' and adding E_2 to G'. We also call it swapping E_1 for E_2 in G'. Given a tour τ, a swap (E_1, E_2) is *improving* for τ, if after swapping E_1 for E_2 in τ, we obtain a tour with lower total edge weight.

A *(k-)swap sequence* is a sequence $L = (S_1, \ldots, S_\ell)$, such that each S_i is a (k-)swap. For a tour τ, we denote by τ^L the subgraph obtained from τ by performing $S_1, \ldots, S_\ell$ in their order in L. L is *improving* for a tour τ if each S_i is an improving (k-)swap for $\tau^{(S_1,\ldots,S_{i-1})}$.

The local search problem TSP/k-Opt corresponds to TSP with the k-Opt neighborhood (that is, the neighbors of a tour τ are those that can be obtained from τ by an improving k-swap). The k-Opt algorithm is then the standard local search algorithm for this problem, and an execution of the algorithm corresponds to an improving k-swap sequence.

2.3 Max-Cut/Flip

A Max-Cut instance is a tuple (G, w), where G is an undirected graph (V, E) and $w : E \to \mathbb{R}$ is a function assigning weights to the edges of G. A *cut* (V_1, V_2) of G is a partition of the vertices of G into two disjoint sets V_1 and V_2. The *cut-set* of a cut (V_1, V_2) is the set of edges $xy \in E$ such that $x \in V_1$ and $y \in V_2$. The goal of Max-Cut is to find a cut that maximizes the *value* of the cut, that is the total weight of the edges in the cut-set.

Given a Max-Cut instance and an initial cut, the *flip* of a vertex is a move of that vertex from a set of the cut to the other. The flip of a vertex is *improving*, if it results in an increase in the value of the cut. For a cut σ, its *flip neighborhood* is the set of all cuts obtained from σ by an improving flip. The Max-Cut/Flip problem is the local search problem that corresponds to the Max-Cut problem with the flip neighborhood. We call its standard local search algorithm the *Flip algorithm*. A *flip sequence* is a sequence $(v_1, \ldots, v_\ell)$ of vertices of G. A flip sequence is *improving*, if flipping the vertices in the order in the sequence increases the value of the cut at every step. In other words, an improving flip sequence corresponds to an execution of the Flip algorithm.

Monien and Tscheuschner [17] showed the all-exp property for Max-Cut/Flip even for graphs with bounded degree.

▶ **Theorem 3** ([17, 15]). *Max-Cut/Flip has the all-exp property, even when restricted to instances where all vertices have degree at most four.*

Michel and Scott [15] recently presented an alternative proof for Theorem 3. Interestingly, their construction is highly structured and exhibits a unique property: With a suitable initial cut, there is exactly one (maximal) improving flip sequence, and this sequence has exponential length. We rely on this particular construction and especially the unique property to achieve the low value of k in Theorem 1.

Note that Theorem 3 is tight with respect to the maximum degree, since the Flip algorithm on graphs with maximum degree at most three always terminates after a polynomial number of iterations [20].

3 The main reduction

In this section, we describe the main reduction to TSP/k-Opt from Max-Cut/Flip.

Let (H, w) be a Max-Cut instance. In order to avoid confusion with the vertices and edges in the TSP instance later on, we use *H-vertices* and *H-edges* for the vertices and edges of H. We denote by n and m the number of H-vertices and H-edges, respectively.

We construct from H the corresponding TSP instance as follows. We start with a cycle of $3(n+m)$ edges. We assign $n+m$ edges of this cycle to each of the n H-vertices and the m H-edges, such that any two assigned edges have distance at least two on the cycle.

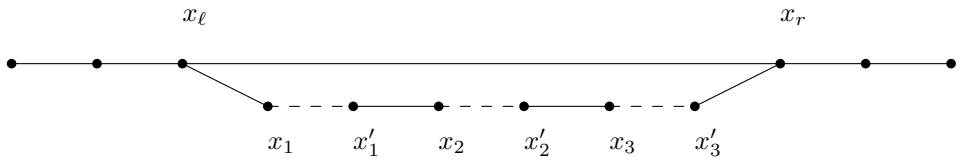


Figure 1 The first-set edge $x_\ell x_r$ and the second-set path $(x_\ell, x_1, x'_1, x_2, x'_2, x_3, x'_3, x_r)$ of an H-vertex x of degree three. The dashed edges are gateways. The other edges of the second-set path are doors.

Next, in the cycle consider an edge that is assigned to an H-vertex x. (Refer to Figure 1 for an illustration of the following concepts.) We label the two incident vertices of this edge x_ℓ and x_r, representing the left and the right vertex of the edge. Let $d(x)$ be the degree of x in H. We add a new path of length $2d(x)+1$ to connect x_ℓ and x_r. We call this new path the *second-set path* of x, while we call the original edge that was assigned to x the *first-set edge* of x. The idea is that the tour can connect x_ℓ and x_r either via the first-set edge or via the second-set path. This simulates whether the H-vertex x is in the first set or second set of the cut for the Max-Cut problem. Let $x_\ell, x_1, x'_1, \ldots, x_{d(x)}, x'_{d(x)}, x_r$ be the labels of the vertices along the second-set path. For $i \in \{1, \ldots, d(x)\}$, we call the edge $x_i x'_i$ a *gateway* of x. The other edges of the second-set path are called the *doors* of x. In other words, we have alternating doors and gateways along the path, with doors at both ends of the path.

For each H-edge xy, we call the edge in the cycle of length $3(n+m)$ assigned to xy the *xy-edge*. We remove a gateway of x, a gateway of y, and the xy-edge, and we connect the six incident vertices of the three removed edges by a *parity gadget*.

The purpose of this parity gadget is to simulate the contribution of the weight of edge xy to the objective of the Max-Cut problem, based on whether x and y are in the same set. We will formally define the parity gadget in Section 3.1.

Finally, for each H-vertex x, we assign an *XOR gadget* to the first-set edge of x and the door of x incident to x_r. The purpose of the XOR gadget is to make sure that we can simulate only one flip in H by a k-swap in the new graph. The formal definition of the XOR gadget and its assignment are discussed in Section 3.2.

Let G be the resulting graph after all the operations above (see Figure 2 for an example). Except for certain edges in the parity gadgets, which we will specify later, the other edges have weight zero, including the edges in the XOR gadgets, the initial cycle, and the doors. As a TSP instance requires a complete graph, we add the remaining edges with weight ∞ to obtain the final graph G_∞. However, if we start with a tour with a finite total weight, the k-Opt algorithm will never visit a tour that uses an edge with weight ∞. Hence, for the remaining of the reduction, we will argue based only on G.

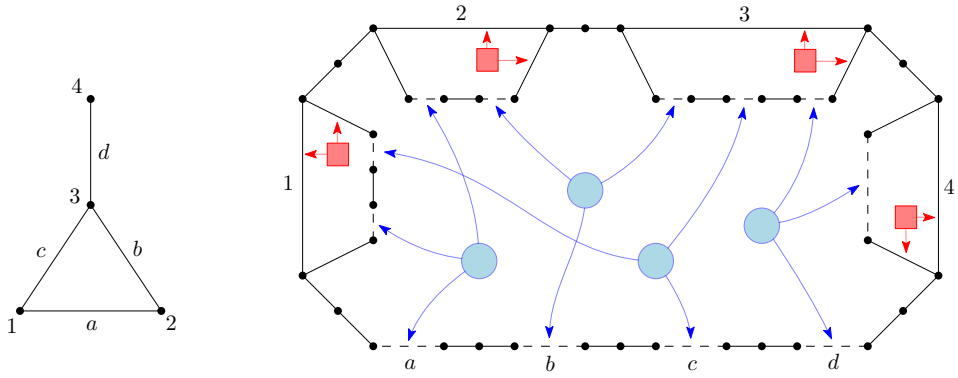


Figure 2 An example of our reduction from a Max-Cut instance (left figure) to a TSP instance (right figure). The parity gadgets are indicated by the blue circles attached to three edges each. The XOR gadgets are indicated by red boxes attached to two edges each.

3.1 Parity gadgets

In this section, we specify the parity gadgets, formally defined as follows.

▶ **Definition 4** (Parity Gadget). *A* parity gadget *is an edge weighted graph containing at least six distinct vertices labeled X, X', Y, Y', Z, Z' that satisfies the following two properties. First there exist at least the following four possibilities to cover the vertices of the parity gadget by vertex disjoint paths with endpoints in the set $\{X, X', Y, Y', Z, Z'\}$:*

(1) *A $\{Z, Z'\}$-path;*

(2) *An $\{X, X'\}$-path and a $\{Z, Z'\}$-path;*

(3) *A $\{Y, Y'\}$-path and a $\{Z, Z'\}$-path; or*

(4) *An $\{X, X'\}$-path, a $\{Y, Y'\}$-path and a $\{Z, Z'\}$-path.*

The four possibilities are called subtour *(1),* subtour *(2),* subtour *(3), and* subtour *(4) (see Figure 3 for an example). We require that in these four cases the cover is unique. A parity gadget may allow more than these four possibilities to cover the vertices by vertex disjoint paths with all endpoints in the set $\{X, X', Y, Y', Z, Z'\}$. Any such cover is called a* subtour *as long as Z and Z' are endpoints of some path(s) in this cover.*

We require in addition if X and X' are in the set of endpoints, then there must exist an $\{X, X'\}$-path in the cover. We require the same for the vertices Y and Y'.

Second, the edges of a parity gadget must allow a partition into three subsets, the same-set edges*, the* different-set edges*, and the remaining edges. The same-set edges have the same weight, which we call the* same-set weight*. Similarly, the different-set edges have the same* different-set weight*. The remaining edges have weight zero.*

We require that subtours (1) and (4) contain exactly one same-set edge and no different-set edge, and that subtours (2) and (3) contain exactly one different-set edge and no same-set edge. See Figure 3 for an example.

As explained before, a parity gadget is used to replace a gateway XX' of an H-vertex x, a gateway YY' of an H-vertex y, and the xy-edge ZZ'. The vertices X, X', Y, Y', Z, and Z' are part of the parity gadget, and the gadget is connected with the rest of G via exactly one incident edge to each of these vertices. We call these six incident edges the *external edges* of the gadget. We define the *internal edges* as the edges within the parity gadget. Further, we say that the gadget is *related* to the H-vertices x and y.

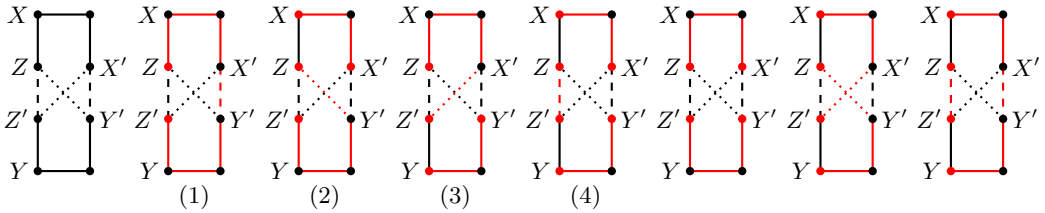


Figure 3 An example of a parity gadget (left figure). The next four figures show the four possibilities, subtour (1)–(4), to cover the vertices of the parity gadget by disjoint paths (red edges and red endpoints). The right three figures show additional subtours. The dashed edges are the same-set edges, the dotted edges are the different-set edges, and the solid edges are the remaining edges.

By construction, the removed edge ZZ' was originally part of a path of length five, say $(Z_1, Z_2, Z, Z', Z_3, Z_4)$. Since Z_2 and Z_3 have degree two in G, any tour of G has to contain Z_2Z and $Z'Z_3$. Therefore, the tour can only contain exactly one internal edge incident to Z and one incident to Z' (which may coincide). This is the reason why in the definition of a parity gadget the set $\{Z, Z'\}$ appears in all four cases.

A subtour containing an $\{X, X'\}$-path (i.e., subtour (2) or subtour (4)) represents that the corresponding H-vertex x is in the second set of the cut; otherwise, x is in the first set. When such a subtour occurs in the gadget, we say the gadget *uses* an $\{X, X'\}$-path. We have a similar representation and notation for the $\{Y, Y'\}$-path.

By definition of a parity gadget the total weight of the tour edges within a parity gadget is the same-set weight, when x and y are in the same set of the cut, and it is the different-set weight, when they are in different sets.

Next, a parity gadget is an (r_x, r_y)-*parity gadget*, if

- We need to remove exactly r_x internal edges and add exactly $r_x - 1$ internal edges to change from subtour (1) to subtour (2) or from subtour (3) to subtour (4);
- We need to remove exactly r_y internal edges and add exactly $r_y - 1$ internal edges to change from subtour (1) to subtour (3) or from subtour (2) to subtour (4);
- In order to change from subtour (1) to subtour (4) or between subtour (2) and subtour (3), we need to remove at least $\max\{r_x, r_y\}$ internal edges and add at least $\max\{r_x, r_y\} - 1$ internal edges.

We call the changes in the first two conditions above and their reverses the *standard subtour changes*.

By definition, a parity gadget may allow more than the four subtours (1)–(4) as possibilities to cover the vertices by disjoint paths with end vertices in the set $\{X, X', Y, Y', Z, Z'\}$. The parity gadget shown in Figure 3 allows for example to cover the vertices by a $\{Z, X'\}$-path and a $\{Z', Y'\}$-path. We say that a parity gadget is a *strict parity gadget*, or simply a *strict gadget*, if subtours (1)–(4) are the only possible subtours for the gadget in G, *after* we equip all gadgets (details are given in Section 3.3) used in the reduction. Thus the property of being strict may depend on the other gadgets used in the reduction.

3.2 XOR gadgets

The remaining gadgets used in the reduction are the XOR gadgets. We generalize these gadgets from the XOR gadget by [18]. See Figure 4(a)–(c) for an illustration for the definition below.

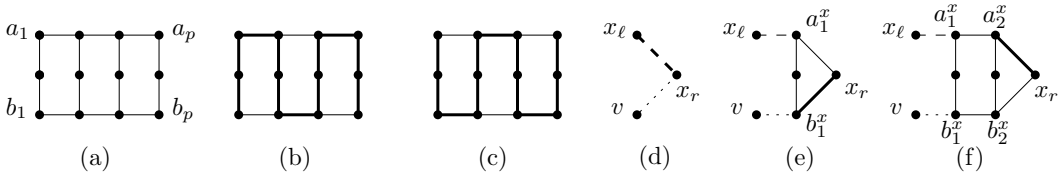


Figure 4 The XOR gadget of order four (a) and its two subtours ((b) and (c)). (d)–(f) are examples of assigning the XOR gadgets of order from zero to two to the H-vertex x, where dashed edges, dotted edges, and bold edges represent the left first-set edge, the door closest to x_r, and the right first-set edge, respectively.

▶ **Definition 5** (XOR Gadget). *Let p be a nonnegative integer. The* XOR gadget *of order p is a graph containing two paths $(a_1, \ldots, a_p)$ and $(b_1, \ldots, b_p)$, and for $i \in \{1, \ldots, p\}$, there is a path of length two with a_i and b_i as endpoints. A* subtour *of the XOR gadget is a spanning path with two endpoints in the set $\{a_1, a_p, b_1, b_p\}$. For convenience, when $p = 0$, both the XOR gadget of order zero and its only subtour are defined to be the empty graph.*

It is easy to see that an XOR gadget has two subtours, except when $p \leq 1$, in which case, it has only one subtour. Further, for $p \geq 2$, changing from one subtour to the other requires a swap of $p - 1$ edges.

We define the *assigning* of the XOR gadget of some order p to an H-vertex x as follows. (See Figure 4(d)–(f) for an illustration.) Recall that $x_\ell x_r$ is the first-set edge of x, and let vx_r be the incident door of x to x_r. We subdivide the two edges above into paths of length $p+1$, $(x_\ell, a_1^x, \ldots, a_p^x, x_r)$ and $(v, b_1^x, \ldots, b_p^x, x_r)$. Then for $i \in \{1, \ldots, p\}$, we connect a_i^x and b_i^x with a path of length two. Note that when $p = 0$, we do nothing. Further note that when we remove the edges incident to x_ℓ, x_r, and v in the construction above, we obtain the XOR gadget of order p as defined in Definition 5.

We call these incident edges to x_ℓ, x_r, and v the *external edges* of the XOR gadget, and we call the other edges in the construction the *internal edges* of the gadget. For convenience, we still refer to the external edge incident to v (i.e., vb_1^x for $p \geq 1$ and vx_r for $p = 0$) as a door of x. Additionally, we call it the *closest door to x_r*. We call the external edge incident to x_ℓ (i.e., $x_\ell a_1^x$ for $p \geq 1$ and $x_\ell x_r$ for $p = 0$) the *left first-set edge* of x. We define the *right first-set edge* of x as $x_r x_\ell$ if $p = 0$, $x_r a_p^x$ if p is positive and even, and $x_r b_p^x$ if p is odd. Lastly, we call the other external edge incident to x_r the *right second-set edge* of x.

We define an *incident edge* of a nonempty subtour of the XOR gadget to be an external edge incident to an endpoint of the subtour. The incident edge of an empty subtour (i.e., when $p = 0$) is defined to be simply an external edge of the XOR gadget.

Based on the definitions above, it is easy to verify the following.

▶ **Observation 6.** *For the XOR gadget assigned to an H-vertex x, one subtour of the gadget is incident to the left and right first-set edges of x, and another subtour of the gadget is incident to the closest door to x_r and the right second-set edge of x. The two subtours above are identical, if the order of the gadget is at most one. Otherwise, they are distinct.*

3.3 Equipping gadgets

Within our reduction we will use different parity gadgets at different places and XOR gadgets of different orders. The exact specification of which parity gadget we use at what place and the XOR gadget of which order is used for which H-vertex will be called the *equipping* of gadgets. We equip the gadgets through a labeling scheme. Specifically, for an H-vertex x

and an incident H-edge z, a labeling L assigns an integer label to the pair (x, z). We say the label is incident to x and to z. Additionally, L also assigns an integer label to each H-vertex. We also say this label is incident to x. We call a labeling L *valid*, if for every H-edge xy, there exists a $(L(x, xy), L(y, xy))$-parity gadget, and for every H-vertex x, $L(x) \geq 0$.

Then we equip the gadgets based on a valid labeling L as follows. For an H-edge xy, we equip the $(L(x, xy), L(y, xy))$-parity gadget to xy. Recall that $w(xy)$ is the weight of xy. If $w(xy) \geq 0$, the same-set weight of the gadget is $w(xy)$, and its different-set weight is zero. Otherwise, the same-set weight of the gadget is zero, and its different-set weight is $-w(xy)$. Finally, for an H-vertex x, we assign the XOR gadget of order $L(x)$ to x, and all edges involved in this assignment have weight zero. We call this a *gadget arrangement* corresponding to L. Note that this construction implies that edge weights in the TSP instance are nonnegative.

For each H-vertex x, we define the *label sum* of x to be the sum of the labels incident to x, i.e., $L(x) + \sum_{y:xy \in E(H)} L(x, xy)$. We say a labeling is an s-labeling, if each H-vertex has label sum s.

3.4 Initial tour

To complete the description of the TSP/k-Opt instance, we specify the initial tour of G. We obtain this tour from the initial cut of H as follows. The tour contains all incident edges of degree-two vertices. For an H-vertex x, if x is in the first set, we include the left and right first-set edges of x in the tour. Further, we use the subtour of the XOR gadget assigned to x, such that this subtour is incident to the left and right first-set edges of x. If x is in the second set, we include all the doors and the right second-set edge of x in the tour. Moreover, we use the subtour of the XOR gadget assigned to x, such that this subtour is incident to the right second-set edge of x and the closest door to x_r.

For an H-edge xy, in the corresponding gadget, we use the subtour (1) if x and y are in the first set, subtour (2) if x is in the second set and y in the first set, subtour (3) if x is in the first set and y in the second set, and subtour (4) if x and y are in the second set.

By the construction of G and the definition of the subtours, it can be verified that we obtain a tour of G.

3.5 Correspondence between flip sequences and swap sequences

Let us assume that we have a valid s-labeling L, and that the parity gadgets in the corresponding gadget arrangement are strict. Then we have the following correspondence (for a proof see [7]).

▶ **Lemma 7.** *Let I be a Max-Cut/Flip instance and σ be an initial cut for I. Suppose for some s, there is a valid s-labeling L for I such that all gadgets in the gadget arrangement in L are strict. Then we can reduce I to a TSP/k-Opt instance I', for $k = s + 1$, and obtain an initial tour τ from σ, such that there is a one-to-one correspondence between improving flip sequences for I and σ and improving k-swap sequences for I' and τ.*

4 All-exp property for $k \geq 13$

In this section, we prove the following statement.

▶ **Lemma 8.** *TSP/k-Opt has the all-exp property for $k \geq 13$.*

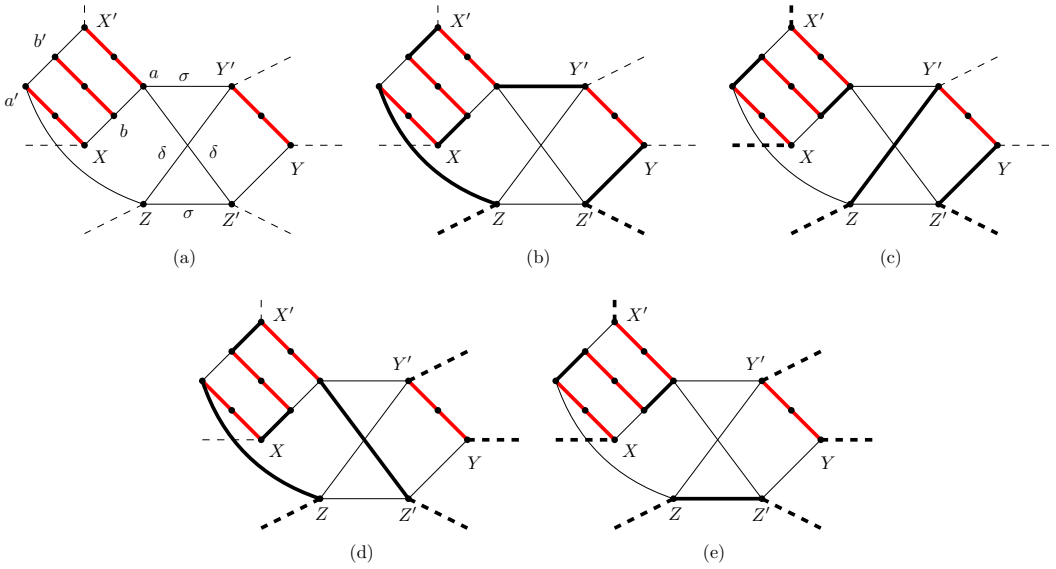


Figure 5 (a) A (4,2)-simple gadget. Bold red edges are incident to degree-two vertices. Dashed edges are external edges. $Y'a$ and ZZ' are same-set edges with same-set weight σ, while $Z'a$ and $Y'Z$ are different-set edges with different-set weight δ. (b)–(e) show the subtours (1)–(4).

We start with the parity gadgets. We call a parity gadget a *simple gadget*, if the parity gadget only allows subtours (1)–(4) (and no other subtours), independent of other gadgets equipped in the graph G. If a simple gadget is an (r_x, r_y)-parity gadget we call it an (r_x, r_y)-simple gadget. Note that a simple gadget is a strict gadget. The next result shows that a $(4, 2)$-simple gadget exists (and by symmetry, we also have a $(2, 4)$-simple gadget). The proof is given in [7].

▶ **Lemma 9.** *The gadget as depicted in Figure 5 is a* $(4, 2)$*-simple gadget.*

We can now prove Lemma 8. We use the reduction from Max-Cut/Flip to TSP/k-Opt as described in Section 3. By Theorem 3 we may assume H to have maximum degree four.

We now construct a valid $(k-1)$-labeling. Firstly, we assign an orientation on the H-edges, such that every degree-four H-vertex has exactly two incoming edges and two outgoing edges. Specifically, we repeat the following procedure: Until all H-edges have an orientation, we take a maximal (possibly closed) walk in the subgraph of unoriented H-edges, and we orient the edges along the walk. It is easy to see that the orientation is as desired.

Next, for every directed H-edge z with head x and tail y, we label (x, z) and (y, z) with four and two, respectively. By Lemma 9, there exists a simple gadget corresponding to these labels. Our construction guarantees that up to this point the label sum of every H-vertex is at most 12. Next, for each H-vertex, we assign a nonnegative label to the H-vertex, such that the label sum at the H-vertex is exactly $k - 1$. This is possible because $k \geq 13$. Hence, we obtain a valid $(k - 1)$-labeling.

Further, since we use only simple gadgets, the labeling above satisfies the condition of Lemma 7. Then by Theorem 3 and Lemma 7, we obtain Lemma 8.

5 All-exp property for $k \geq 9$

In this section, we prove the following statement.

▶ **Lemma 10.** *TSP/k-Opt has the all-exp property for* $k \geq 9$.

We use the reduction in Section 3, with two extra ingredients. Firstly, we look into the construction by Michel and Scott [15]. The graph for the instance is constructed inductively as follows (see Figure 6 for an illustration):

- The base graph F_0 consists of a single edge $v_{0,1}v_{0,8}$ with weight 7.
- The graph F_n contains a path of eight new vertices $v_{n,1}, v_{n,2}, v_{n,3}, v_{n,4}, v_{n,5}, v_{n,6}, v_{n,7}, v_{n,8}$ that appear in the path in that order. The weights of the edges along the path from $v_{n,1}$ to $v_{n,8}$ are $7 \cdot 8^n, 5 \cdot 8^n, 5 \cdot 8^n, 3 \cdot 8^n, 3 \cdot 8^n, 8^n, 8^n$. Next, we connect F_n to F_{n-1} as follows: We add edges connecting $v_{n-1,1}$ to $v_{n,2}, v_{n,4}$, and $v_{n,6}$, with weights $8^n, -8^n$, and 8^n, respectively. Finally, we add edges connecting $v_{n-1,8}$ to $v_{n,3}, v_{n,5}$, and $v_{n,7}$, with weights 1, -1, and 1, respectively.

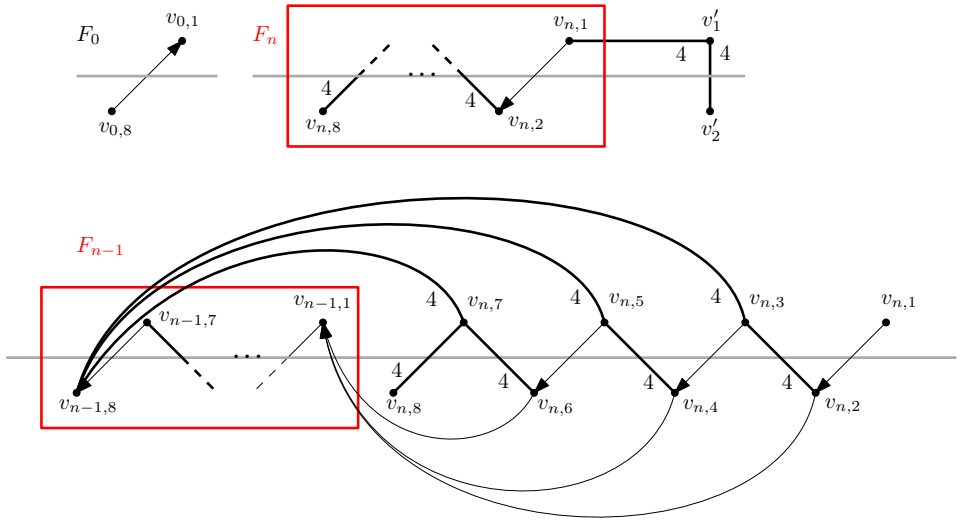


Figure 6 Michel-Scott construction of a Max-Cut instance that has an exponentially long improving flip sequence. Vertices on one side of the horizontal line are in the same set of the initial cut. We show here a valid $(k-1)$-labeling L and its gadget arrangement, for $k \geq 9$. Only labels $L(\cdot, \cdot)$ with value other than two are specified. All labels to vertices have suitable values to ensure the label sum at each vertex is $k-1$. Bold edges indicate simple gadgets, and the directed edges are flexible gadgets. The arrows indicate the direction of the forcing rule (i.e., if the gadget equipped to a directed edge is strict at the tail, then it is forced to be strict at the head).

The final graph H_n consists of all the graphs $F_0, \ldots, F_n$ and the connecting edges, as well as two new vertices v_1' and v_2' and two new edges $v_{n,1}v_1'$ and $v_1'v_2'$ with weights 8^{n+1} and $2 \cdot 8^{n+1}$, respectively. The initial cut of H_n is as follows: One set of the cut contains exactly v_1' and all vertices $v_{i,j}$ such that j is odd. The other set contains the remaining vertices.

Michel and Scott [15] showed that there exists a unique maximal improving flip sequence L_n from the aforementioned cut of H_n. The sequence is described recursively as follows: $L_0 = v_{0,1}v_{0,8}$, and $L_n = v_{n,1}v_{n,2}L_{n-1}v_{n,3}v_{n,4}L_{n-1}v_{n,5}v_{n,6}L_{n-1}v_{n,7}v_{n,8}$.

Secondly, we introduce the *flexible gadget*, a $(2,2)$-parity gadget as depicted in Figure 3. Note that it is not a simple gadget, because besides the subtours (1)–(4), there are other subtours, as shown in Figure 3. However, the following lemma (for a proof see [7]) shows that a suitable equipping of the flexible and simple gadgets can force the flexible gadgets to use only the subtours (1)–(4), and hence, they are strict:

▶ **Lemma 11.** *Suppose in the reduction in Section 3, we only equip simple, flexible, and XOR gadgets. If the H-edges corresponding to the flexible gadgets form a forest in H, then all parity gadgets are strict.*

To prove Lemma 10 we use the reduction from Section 3, with some specialization. Firstly, for the Max-Cut/Flip instance, we use the graph H_n as the graph H, for $n \geq 1$, and we use the corresponding weight and initial cut as described above. Secondly, we use the following labeling L: (See Figure 6 for a depiction of the labeling and gadget assignment.)

- For $i \in \{1, \ldots, n\}$ and $q \in \{3, 5, 7\}$, we have $L(v_{i,q}, v_{i,q}v_{i-1,8}) = 4$;
- For $i \in \{1, \ldots, n\}$ and $q \in \{2, 4, 6\}$, we have $L(v_{i,q}, v_{i,q}v_{i,q+1}) = 4$;
- $L(v_{n,8}, v_{n,8}v_{n,7}) = 4$;
- $L(v'_1, v_{n,1}v'_1) = L(v'_1, v'_1v'_2) = 4$;
- $L(x, z) = 2$, for all other pairs of an H-vertex x and an H-edge z not mentioned above and
- $L(v_{n,8}) = L(v_{n,1}) = k - 5$; $L(v'_2) = k - 3$; for other H-vertex x, we have $L(x) = k - 9$. (As $k \geq 9$, these labels are nonnegative.)

For every H-edge xy, if $L(x, xy) = L(y, xy) = 2$, we equip the flexible gadget to xy. Otherwise, we use the $(4, 2)$-simple gadget as described in Lemma 9 instead.

Observe that the labeling above is a valid $(k-1)$-labeling. Further, consider the subgraph of H containing all H-edges corresponding to the flexible gadgets. This subgraph is a forest with the leaves: $v_{0,8}$, $v_{n,1}$, $v_{n,3}$, $v_{n,5}$, $v_{j,3}$, $v_{j,5}$, $v_{j,7}$, and $v_{j,8}$ for $j \in \{1, \ldots, n-1\}$. By Lemma 11, all parity gadgets are strict.

Then Lemma 10 follows from Theorem 3 and Lemma 7.

6 All-exp property for $k \geq 5$

In this section, we prove Theorem 1 that asserts the all-exp property of TSP/k-Opt for $k \geq 5$. This proof is similar to that of Lemma 10 with a few changes. Firstly, we modify the Michel-Scott construction for the Max-Cut/Flip instance. In particular, we replace certain edges by paths of odd length. Secondly, we introduce a new gadget, called the *double gadget* that simulates two adjacent edges simultaneously. Lastly, we do not insist that all gadgets are strict. However, we argue that with our chosen initial tour, we cannot encounter a subtour other than subtours (1)–(4) in any parity gadget by a k-swap sequence.

6.1 Modified Michel-Scott construction

See Figure 7 for a depiction of the modification explained in this section. Recall the construction by Michel and Scott [15] and the unique maximal improving flip sequence L_n in Section 5. Let p be an odd number that is at least three. We observe that for any consecutive pair (v, v') in the sequence L_n, vv' is an edge in H_n and (v', v) is not a contiguous subsequence of L_n. In that case, for such consecutive pair (v, v'), we orient the edge vv' in H_n from v to v'. Then we obtain a partial orientation $\overrightarrow{H}_n$ of H_n.

For $i \in \{0, \ldots, n\}$ and $q \in \{1, \ldots, 7\}$, note that the vertex $v_{i,q}$ only has one out-neighbor $v_{i',q'}$ in $\overrightarrow{H}_n$, for some i', q'. We replace the edge $v_{i,q}v_{i',q'}$ by a path of length p $(v_{i,q}, u^1_{i,q,q'}, \ldots, u^{p-1}_{i,q,q'}, v_{i',q'})$, with $p-1$ new vertices $u^1_{i,q,q'}, \ldots, u^{p-1}_{i,q,q'}$. The weights of the new edges and which set of the cut the new vertices belong to depend on the sign of the weight of the original edges. In particular, let ω be the original weight of $v_{i,q}v_{i',q'}$ and ε be a very small number (say, 2^{-n}), and define $u^0_{i,q,q'} := v_{i,q}$ and $u^p_{i,q,q'} := v_{i',q'}$. If $\omega > 0$, then we assign the weights for the edges along the path in decreasing order: for $j = 0, \ldots, p-1$, the edge $u^j_{i,q,q'}u^{j+1}_{i,q,q'}$ has weight $\omega - j\varepsilon$. Further, we assign $u^j_{i,q,q'}$ to the same set of the cut as $v_{i,q}$ for j even, and to that as $v_{i',q'}$ for j odd. If $\omega < 0$, we assign the weights for the edges along the path in increasing order: for $j = 0, \ldots, p-1$, the edge $u^j_{i,q,q'}u^{j+1}_{i,q,q'}$ has weight

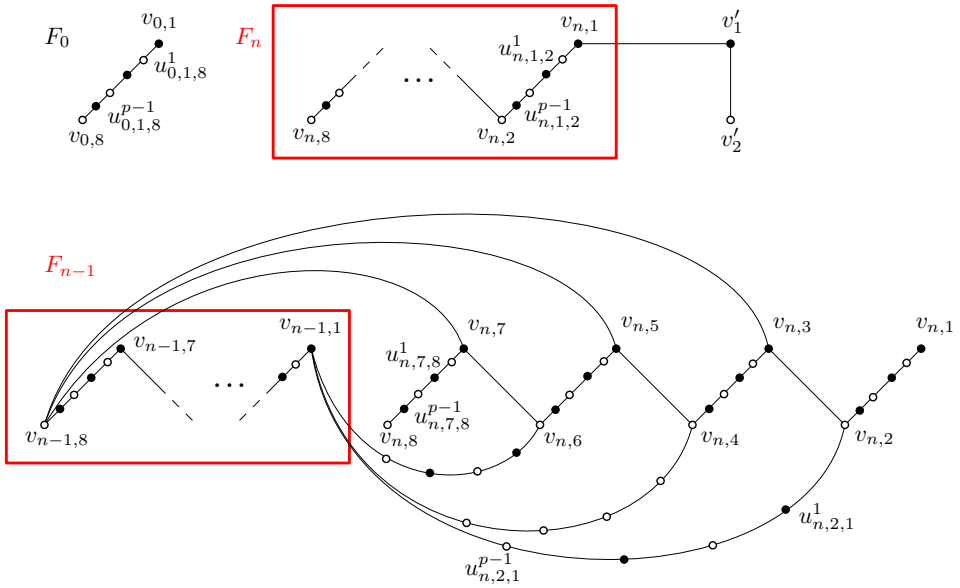


Figure 7 Modified Michel-Scott construction, where we replace certain edges in Figure 6 with paths of length p. Here, we indicate the sets of the initial cut by the colors of the vertices.

$\omega + j\varepsilon$. Further, we assign $u^j_{i,q,q'}$ to the same set of the cut as $v_{i,q}$, for all $j \in [p-1]$. The resulting (undirected) graph, weight, and cut after all replacements are the graph H, the weight w, and the initial cut χ_0 that we will use for the reduction.

We can show (for a proof see [7]) that the uniqueness property carries over to the new instance.

▶ **Lemma 12.** *There is a unique maximal improving flip sequence L'_n from the cut χ_0 of H, and this sequence is exponentially long.*

6.2 Double gadget

Let xy and xt be two H-edges. Denote $z := xy$. A *double gadget* replaces a gateway YY' of y, a gateway TT' of t, the z-edge ZZ', and a subpath (X, X_1, X_2, X') of the second-set path of x, where XX_1 and X_2X' are two gateways of x. Note that the vertices X_1 and X_2 are removed from the graph G, when we equip the double gadget to the pair xy and xt. Further, the xt-edge is not replaced by any gadget. As the third edge in a path of length five in G, the xt-edge is then used in every tour of G. We define the external and internal edges of the double gadget similar to those of parity gadgets.

A double gadget has to guarantee at least eight possible subtours (with subtours defined analogously to subtours defined in Definition 4). A $\{Z, Z'\}$-path is always present in these subtours. The eight subtours corresponds to all possibilities of containing an $\{X, X'\}$-path, a $\{Y, Y'\}$-path, or a $\{T, T'\}$-path.

Let $\sigma(x, y)$ and $\delta(x, y)$ be the same-set and different-set weights for the edge xy, respectively (i.e., they correspond to the weights when x and y are in the same set and in different sets). The numbers $\sigma(x, t)$ and $\delta(x, t)$ are defined analogously. For each of the eight subtours, the total weight of the internal edges in the subtour is the sum of two numbers, a_{xy} and a_{xt}. a_{xy} takes value $\sigma(x, y)$ if an $\{X, X'\}$-path and a $\{Y, Y'\}$-path are both present or both absent; and it takes value $\delta(x, y)$ if exactly one of these paths is present. a_{xt} is defined analogously.

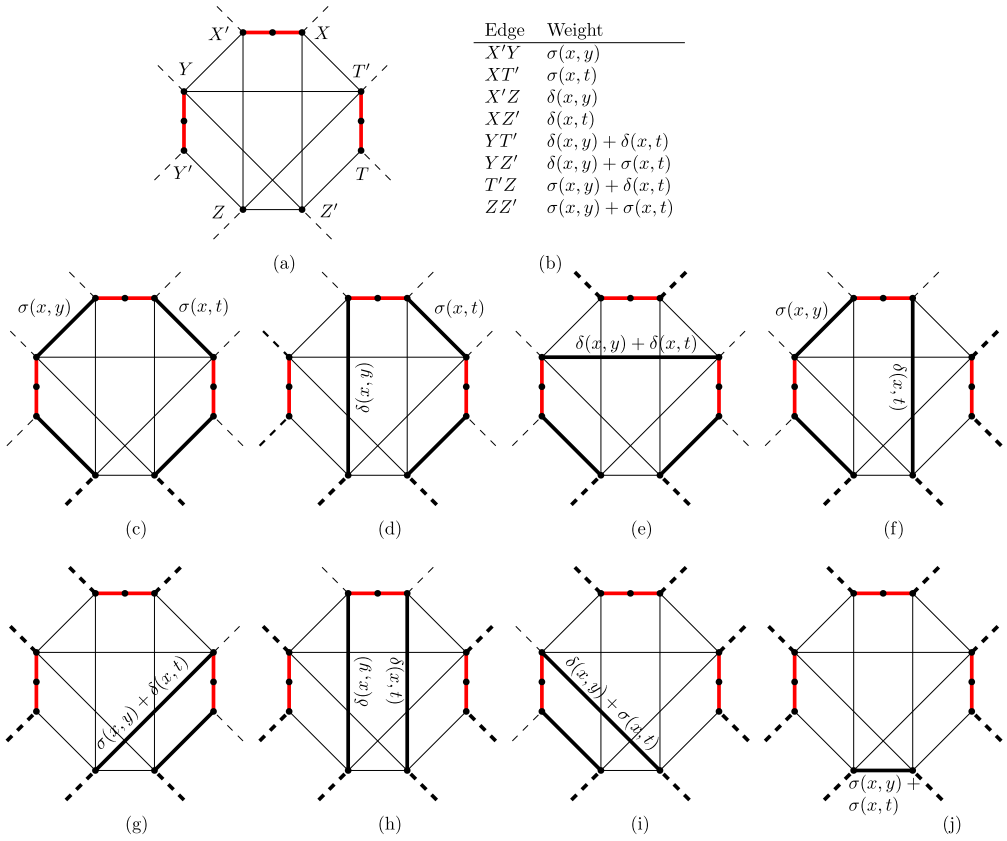


Figure 8 A (2,2,2)-double gadget (a) and the listing of edges with nonzero weights (b). The other panels show the eight subtours when the gadget is locally strict, including the nonzero weights of the edges in the subtours.

We extend the definition of an (r_x, r_y)-parity gadget to an (r_x, r_y, r_t)-double gadget in the obvious way, e.g., r_x is the number of internal edges that have to be removed if in a subtour an $\{X, X'\}$-path is added.

In the following proof, we use the (2,2,2)-double gadget as depicted in Figure 8. One can easily verify that the graph is indeed a (2,2,2)-double gadget. Note that the gadget also allows subtours other than those shown in the figure. However, we will show later that these other subtours do not appear in the improving swap sequence of concern.

6.3 Proof of Theorem 1

We use the reduction described in Section 3, from the Max-Cut instance (H, w) and the initial cut χ_0 indicated in Section 6.1. In the construction of the Max-Cut instance, we use a constant p which is odd and more than $2k$, where we recall that p is the length of the paths that replace certain edges.

We then define a suitable labeling by using the $(2, 2, 2)$-double gadgets, the flexible gadgets, the $(4, 2)$-simple gadgets, and the XOR-gadgets. The crucial idea here is that a double gadget can be used to label two incident edges. As a result, each H-vertex can now be incident to at most three labels, including one label corresponding to the XOR gadget. Accordingly, we can bring the label sum down to as low as four.

Next, we argue that although the flexible gadget and the $(2, 2, 2)$-double gadget allow many subtours, we only encounter a limited subset of these subtours. For this we introduce the notion of local strictness and show that we can get a result similar to Lemma 7. With this we can prove Theorem 1. See [7] for the details of this proof.

7 PLS-Completeness for $k \geq 17$

In this section, we prove the PLS-completeness of TSP/k-Opt for $k \geq 17$. With this result we not only improve the value $k \geq 14,208$ from Krentel [13, 8]. We also present the first rigorous PLS-completeness proof for TSP/k-Opt as Krentel's proof has a substantial gap. He assumes without proof that no edges of infinite weight can appear in a local optimum. The definition of PLS-completeness requires that the function g maps local optima to local optima. Therefore, either one has to show that a local optimum cannot contain edges of infinite weight, or one has to show how to extend the definition of the function g to local optima that contain edges of infinite weight. Both are not done in the paper of Krentel [13], and there is no obvious way how to fill this gap. For our reduction we can prove in Lemma 14 below that local optima cannot contain edges of infinite weight. There seems not to be a generic way to prove such a result for arbitrary TSP instances as for example those constructed by Krentel [13]. A result similar to Lemma 14 was obtained by Papadimitriou [19] for the Lin-Kernighan heuristic.

▶ **Theorem 2.** *TSP/k-Opt is PLS-complete for $k \geq 17$.*

Our proof of Theorem 2 follows closely the proof of Lemma 8. However, there are three key differences. Firstly, while the all-exp property is known to hold for Max-Cut instances with maximum degree four (Theorem 3), the PLS-completeness of Max-Cut/Flip is only known for maximum degree five:

▶ **Theorem 13** ([4]). *Max-Cut/Flip is PLS-complete, even when restricted to graphs of maximum degree five.*

Secondly, we impose certain structure on the graph G in the reduction. Particularly, we specify which gateways a parity gadget can replace.

Lastly, recall that the TSP instance requires a complete graph G_∞, which we obtain from G by adding the missing edges, which we also call the *non-edges*. By choosing suitable weights for the non-edges we will be able to prove in Lemma 14 that no locally optimal tour of G_∞ can contain a non-edge.

Proof of Theorem 2. We use the reduction from Max-Cut to TSP as described in Section 3. By Theorem 13, we can assume in the Max-Cut instance (H, w), that H is a graph of maximum degree five.

We assign an orientation on the H-edges such that each degree-five vertex has in-degree at most three. We can get such an orientation by repeating the following procedure: Until all H-edges have an orientation, we take a maximal (possibly closed) walk in the subgraph of unoriented H-edges, and we orient the edges along the walk. For every directed H-edge z with head x and tail y, we label (x, z) and (y, z) with four and two, respectively. Next, for each H-vertex x, we assign an integer label to x, such that the label sum at x is $k - 1$. This label is nonnegative, as $k \geq 17$. Corresponding to these labels we use the $(4, 2)$-simple gadgets from Lemma 9 and the XOR-gadgets. Hence, this is a valid $(k - 1)$-labeling. We denote it by L.

Next, we specify the gadget arrangement corresponding to L as follows. Recall that n is the number of H-vertices. Let $x^1, \ldots, x^n$ be the H-vertices. For every H-vertex x, we assign the XOR gadget of order $L(x)$ to x. Let ψ be the increasing lexicographical order of the H-edges with respect to the H-vertex indices. That is, for $i < i'$ and $j < j'$, the H-edge $x^i x^{i'}$ precedes the H-edge $x^j x^{j'}$ in the order ψ, if either $i < j$ or $i = j$ and $i' < j'$. Then we equip the parity gadgets according to the labeling L, such that when we go along the second-set path for each H-vertex x from x_ℓ to x_r, the H-edges corresponding to the related gadgets appear in their order in ψ. We equip the gadgets such that for a (4,2)-simple gadget g related to an H-vertex t, either the vertex X or Y in g is adjacent to either t_ℓ or a vertex in a gadget that is also related to t and precedes g in ψ.

Let G and w be the resulting graph and edge weight function. We have the following lemma (for a proof see [7]).

▶ **Lemma 14.** *For $k \geq 3$ there exists a complete graph G_∞ with a corresponding edge weight function w' obtained from G and w by adding the missing edges with suitable weights, such that for the TSP/k-Opt instance consisting of (G_∞, w'), all locally optimal tours only contain edges of G.*

For a given tour τ we can map it to a cut σ as follows: For each H-vertex x we put x into the first set if τ uses the left first-set edge. Otherwise we put x into the second set. Assume we have a tour τ that is a local optimum but the corresponding cut is not a local optimum. By Lemma 14 the tour τ contains only edges from G. By using arguments similar to those used in the proof of Lemma 7 we can conclude that τ is not a local optimum, a contradiction.

Then Theorem 2 follows from Theorem 13. ◀

8 Conclusion

We have shown that for $k \geq 5$ the k-Opt algorithm for TSP has the all-exp property, i.e. it has exponential running time for all possible pivot rules (Theorem 1). Moreover, we proved that TSP/k-Opt is PLS-complete for $k \geq 17$ (Theorem 2). In both cases we drastically lowered the so far best known value for k which was $\gg 1000$. It was mentioned (without explaining the details) in [13] that there is a connection between the PLS-completeness of a problem and the all-exp property. This connection was made precise by Schäffer and Yannakakis [21] who introduced the notion of *tight* PLS-completeness. They proved that the tight PLS-completeness of a problem implies the all-exp property. Our PLS-completeness result for TSP/k-Opt relies on the PLS-completeness of Max-Cut/Flip for graphs with maximum degree five [4]. As for the latter the tight PLS-completeness is not known we do not get tight PLS-completeness for TSP/k-Opt.

We put some effort into getting the constant in Theorem 1 as small as possible. In contrast, the constant in Theorem 2 very likely can be lowered to 15 by using our techniques from Section 6. However, this would require substantially more involved proofs. Finally we would like to mention (as already observed by Krentel [13]) that our results also hold for *metric* TSP as one can add a sufficiently large constant to all edge weights.

References

1 D. Avis and V. Chvátal. Notes on Bland's pivoting rule. In M. L. Balinski and A. J. Hoffman, editors, *Mathematical Programming Study 8, Polyhedral Combinatorics: Dedicated to the memory of D.R. Fulkerson*, pages 24–34. North-Holland Publishing Company, Amsterdam, 1978. doi:10.1007/BFb0121192.

2 Barun Chandra, Howard Karloff, and Craig Tovey. New results on the old k-opt algorithm for the traveling salesman problem. *SIAM Journal on Computing*, 28(6):1998–2029, 1999. doi:10.1137/S0097539793251244.

3 Dominic Dumrauf and Tim Süß. On the complexity of local search for weighted standard set problems. In Fernando Ferreira, Benedikt Löwe, Elvira Mayordomo, and Luís Mendes Gomes, editors, *Programs, Proofs, Processes. 6th Conference on Computability in Europe, CiE 2010*, volume 6158 of *Lecture Notes in Computer Science*, pages 132–140, Berlin, Heidelberg, 2010. Springer-Verlag Berlin Heidelberg. doi:10.1007/978-3-642-13962-8_15.

4 Robert Elsässer and Tobias Tscheuschner. Settling the complexity of local max-cut (almost) completely. In Luca Aceto, Monika Henzinger, and Jiří Sgall, editors, *Automata, languages and programming. 38th International Colloquium, ICALP 2011, Proceedings, Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 171–182. Springer-Verlag Berlin Heidelberg, 2011. doi:10.1007/978-3-642-22006-7_15.

5 Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the TSP. *Algorithmica*, 68:190–264, 2014. doi:10.1007/s00453-013-9801-4.

6 Donald Goldfarb and William Y. Sit. Worst case behavior of the steepest edge simplex method. *Discrete Applied Mathematics*, 1(4):277–285, 1979. doi:10.1016/0166-218X(79)90004-0.

7 Sophia Heimann, Hung P. Hoang, and Stefan Hougardy. The k-opt algorithm for the traveling salesman problem has exponential running time for $k \geq 5$, 2024. doi:10.48550/arXiv.2402.07061.

8 Hung P. Hoang and Stefan Hougardy. On the PLS-completeness of TSP/k-opt. Technical Report No: 231275, Research Institute for Discrete Mathematics, University of Bonn, 2023.

9 R. G. Jeroslow. The simplex algorithm with the pivot rule of maximizing criterion improvement. *Discrete Mathematics*, 4(4):367–377, 1973. doi:10.1016/0012-365X(73)90171-4.

10 David S. Johnson and Lyle A. McGeoch. The traveling salesman problem: a case study. In Emile Aarts and Jan Karel Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. Princeton University Press, Princeton and Oxford, second edition, 2003. doi:10.2307/j.ctv346t9c.13.

11 David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988. doi:10.1016/0022-0000(88)90046-3.

12 Victor Klee and George J. Minty. How good is the simplex algorithm ? In Oved Shisha, editor, *Inequalities—III. Proceedings of the Third Symposium on Inequalities Held at The University of California, Los Angeles. Septemper 1–9, 1969*, pages 159–175, New York and London, 1972. Academic Press.

13 Mark W. Krentel. Structure in locally optimal solutions. In *30th Annual Symposium on Foundations of Computer Science*, pages 216–221. IEEE Computer Society, 1989. doi:10.1109/SFCS.1989.63481.

14 Mark W. Krentel. On finding and verifying locally optimal solutions. *SIAM Journal on Computing*, 19(4):742–749, 1990. doi:10.1137/0219052.

15 Lukas Michel and Alex Scott. Superpolynomial smoothed complexity of 3-flip in local max-cut, 2024. doi:10.48550/arXiv.2310.19594.

16 Burkhard Monien, Dominic Dumrauf, and Tobias Tscheuschner. Local search: simple, successful, but sometimes sluggish. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, languages and programming. 37th International Colloquium, ICALP 2010, Part I*, volume 6198 of *Lecture Notes in Computer Science*, pages 1–17. Springer, Berlin, 2010. doi:10.1007/978-3-642-14165-2_1.

17 Burkhard Monien and Tobias Tscheuschner. On the power of nodes of degree four in the local max-cut problem. In Tiziana Calamoneri and Josep Diaz, editors, *Algorithms and Complexity. 7th International Conference, CIAC 2010*, volume 6078 of *Lecture Notes in Computer Science*, pages 264–275. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-13073-1_24.

18 Christos H. Papadimitriou. The adjacency relation on the traveling salesman polytope is NP-complete. *Mathematical Programming*, 14(1):312–324, 1978. doi:10.1007/BF01588973.

19 Christos H. Papadimitriou. The complexity of the Lin–Kernighan heuristic for the traveling salesman problem. *SIAM Journal on Computing*, 21(3):450–465, 1992. doi:10.1137/0221030.

20 Svatopluk Poljak. Integer linear programs and local search for max-cut. *SIAM Journal on Computing*, 24(4):822–839, 1995. doi:10.1137/S0097539793245350.

21 Alejandro A. Schäffer and Mihalis Yannakakis. Simple local search problems that are hard to solve. *SIAM Journal on Computing*, 20(1):56–87, 1991. doi:10.1137/0220004.

22 Mihalis Yannakakis. Computational complexity. In Emile Aarts and Jan Karel Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 19–55. Princeton University Press, Princeton and Oxford, second edition, 2003. doi:10.2307/j.ctv346t9c.7.

Optimal PSPACE-Hardness of Approximating Set Cover Reconfiguration

Shuichi Hirahara
National Institute of Informatics, Tokyo, Japan

Naoto Ohsaka
CyberAgent, Inc., Tokyo, Japan

Abstract

In the Minmax Set Cover Reconfiguration problem, given a set system $\mathcal{F}$ over a universe $\mathcal{U}$ and its two covers $\mathcal{C}^{\mathsf{start}}$ and $\mathcal{C}^{\mathsf{goal}}$ of size k, we wish to transform $\mathcal{C}^{\mathsf{start}}$ into $\mathcal{C}^{\mathsf{goal}}$ by repeatedly adding or removing a single set of $\mathcal{F}$ while covering the universe $\mathcal{U}$ in any intermediate state. Then, the objective is to minimize the maximum size of any intermediate cover during transformation. We prove that Minmax Set Cover Reconfiguration and Minmax Dominating Set Reconfiguration are PSPACE-hard to approximate within a factor of $2-\frac{1}{\mathrm{polyloglog}\, N}$, where N is the size of the universe and the number of vertices in a graph, respectively, improving upon Ohsaka (SODA 2024) [32] and Karthik C. S. and Manurangsi (2023) [26]. This is the first result that exhibits a sharp threshold for the approximation factor of any reconfiguration problem because both problems admit a 2-factor approximation algorithm as per Ito, Demaine, Harvey, Papadimitriou, Sideri, Uehara, and Uno (Theor. Comput. Sci., 2011) [23]. Our proof is based on a reconfiguration analogue of the FGLSS reduction [12] from *Probabilistically Checkable Reconfiguration Proofs* of Hirahara and Ohsaka (STOC 2024) [19]. We also prove that for any constant $\varepsilon \in (0,1)$, Minmax Hypergraph Vertex Cover Reconfiguration on $\mathrm{poly}(\varepsilon^{-1})$-uniform hypergraphs is PSPACE-hard to approximate within a factor of $2-\varepsilon$.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Interactive proof systems

Keywords and phrases reconfiguration problems, hardness of approximation, probabilistic proof systems, FGLSS reduction

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.85

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2024/039/` [18]

1 Introduction

1.1 Background

In the field of reconfiguration, we study the reachability and connectivity over the space of feasible solutions under an adjacency relation. Given a *source problem* that asks the existence of a feasible solution, its *reconfiguration problem* requires to decide if there exists a *reconfiguration sequence*, namely, a step-by-step transformation between a pair of feasible solutions while always preserving the feasibility of any intermediate solution. One of the reconfiguration problems we study in this paper is Set Cover Reconfiguration [23], whose source problem is Set Cover. In the Set Cover Reconfiguration problem, for a set system $\mathcal{F}$ over a universe $\mathcal{U}$ and its two covers $\mathcal{C}^{\mathsf{start}}$ and $\mathcal{C}^{\mathsf{goal}}$ of size k, we seek a reconfiguration sequence from $\mathcal{C}^{\mathsf{start}}$ to $\mathcal{C}^{\mathsf{goal}}$ consisting only of covers of size at most $k+1$, each of which is obtained from the previous one by adding or removing a single set of $\mathcal{F}$. Countless reconfiguration problems have been defined from a variety of


source problems, including Boolean satisfiability, constraint satisfaction problems, and graph problems. Studying reconfiguration problems may help elucidate the structure of the solution space of combinatorial problems [13].

The computational complexity of reconfiguration problems has the following trend: a reconfiguration problem is likely to be PSPACE-complete if its source problem is intractable (say, NP-complete); e.g., Set Cover [23], 3SAT [13], and Independent Set [16, 17]; a source problem in P frequently induces a reconfiguration problem in P; e.g., Spanning Tree [23] and 2SAT [13]. Some exception are however known; e.g., 3Coloring [7] and Shortest Path [6]. We refer the readers to the surveys by Nishimura [30] and van den Heuvel [35] and the Combinatorial Reconfiguration wiki [20] for more algorithmic and hardness results of reconfiguration problems.

To overcome the computational hardness of a reconfiguration problem, we consider its *optimization version*, which affords to relax the feasibility of intermediate solutions. For example, Minmax Set Cover Reconfiguration [23] is an optimization version of Set Cover Reconfiguration, where we are allowed to use any cover of size greater than $k+1$, but required to minimize the *maximum size of any covers* in the reconfiguration sequence (see Section 4.1 for the formal definition). Solving this problem approximately, we may be able to find a "reasonable" reconfiguration sequence for Set Cover Reconfiguration that consists of covers of size at most, say, 1% larger than $k+1$. Unlike Set Cover, which is NP-hard to approximate within a factor smaller than $\ln n$ [10, 11, 27], Minmax Set Cover Reconfiguration admits a 2-factor approximation algorithm due to Ito, Demaine, Harvey, Papadimitriou, Sideri, Uehara, and Uno [23, Theorem 6]. An immediate question is: *Is this the best possible?*

Here, we summarize known hardness-of-approximation results on Minmax Set Cover Reconfiguration. Ohsaka [32] showed that Minmax Set Cover Reconfiguration is PSPACE-hard to approximate within a factor of 1.0029 assuming the Reconfiguration Inapproximability Hypothesis [31], which was recently proved [19, 26]. Karthik C. S. and Manurangsi [26] proved NP-hardness of the $(2-\varepsilon)$-factor approximation for any constant $\varepsilon \in (0,1)$. Both results are not optimal: Ohsaka's factor 1.0029 is far smaller than 2, while Karthik C. S. and Manurangsi's result is not PSPACE-hardness. This leaves a tantalizing possibility that there may exist a polynomial-length reconfiguration sequence that achieves a 1.0030-factor approximation for Minmax Set Cover Reconfiguration, and hence the approximation problem may be in NP. Note that the PSPACE-hardness result of Ohsaka [32] disproves the existence of a polynomial-length witness (in particular, a polynomial-length reconfiguration sequence) for the 1.0029-factor approximation under the assumption that NP $\neq$ PSPACE.

1.2 Our Results

We present optimal results of PSPACE-hardness of approximation for three reconfiguration problems. Our first result is that Minmax Set Cover Reconfiguration is PSPACE-hard to approximate within a factor smaller than 2, improving upon Ohsaka [32, Corollary 4.2] and Karthik C. S. and Manurangsi [26, Theorem 4]. This is the first result that exhibits a sharp threshold for the approximation factor of any reconfiguration problem: approximating within any factor below 2 is PSPACE-complete and within a 2-factor is in P [23].

▶ **Theorem 1.1** (informal; see Theorem 4.1)**.** *For a set system $\mathcal{F}$ of universe size N and its two covers $\mathcal{C}^{\mathsf{start}}$ and $\mathcal{C}^{\mathsf{goal}}$ of size k, it is* PSPACE*-complete to distinguish between the following cases*:

- (*Completeness*) *There exists a reconfiguration sequence from* $\mathcal{C}^{\mathsf{start}}$ *to* $\mathcal{C}^{\mathsf{goal}}$ *consisting only of covers of size at most* $k+1$.
- (*Soundness*) *Every reconfiguration sequence contains a cover of size greater than* $(2-\varepsilon(N))(k+1)$, *where* $\varepsilon(N) \coloneqq (\mathrm{polyloglog}\, N)^{-1}$.

In particular, Minmax Set Cover Reconfiguration *is* PSPACE*-hard to approximate within a factor of* $2-\varepsilon(N)$.

As a corollary of Theorem 4.1 along with [32], the following PSPACE-hardness of approximation holds for Dominating Set Reconfiguration, which also admits a 2-factor approximation [23] (please refer to [32] for the problem definition).

▶ **Corollary 1.2** (from Theorem 4.1 and [32, Corollary 4.3]). Minmax Dominating Set Reconfiguration *is* PSPACE*-hard to approximate within a factor of* $2-\frac{1}{\mathrm{polyloglog}\, N}$, *where* N *is the number of vertices in a graph.*

Our third result is a similar inapproximability result for Hypergraph Vertex Cover Reconfiguration, which is defined analogously to Set Cover Reconfiguration. Minmax Hypergraph Vertex Cover Reconfiguration is easily shown to be 2-factor approximable [23]; we prove that this is optimal.

▶ **Theorem 1.3** (informal; see Theorem 4.3). *For any constant* $\varepsilon \in (0,1)$, *a* $\mathrm{poly}(\varepsilon^{-1})$*-uniform hypergraph, and its two vertex covers* $\mathcal{C}^{\mathsf{start}}$ *and* $\mathcal{C}^{\mathsf{goal}}$ *of size* k, *it is* PSPACE*-complete to distinguish between the following cases*:

- (*Completeness*) *There exists a reconfiguration sequence from* $\mathcal{C}^{\mathsf{start}}$ *to* $\mathcal{C}^{\mathsf{goal}}$ *consisting only of vertex covers of size at most* $k+1$.
- (*Soundness*) *Every reconfiguration sequence contains a vertex cover of size greater than* $(2-\varepsilon)(k+1)$.

In particular, Minmax Hypergraph Vertex Cover Reconfiguration *on* $\mathrm{poly}(\varepsilon^{-1})$*-uniform hypergraphs is* PSPACE*-hard to approximate within a factor of* $2-\varepsilon$.

We highlight here that the size of hyperedges in a Hypergraph Vertex Cover Reconfiguration instance of Theorem 4.3 depends (polynomially) only on the value of ε^{-1}, whereas the size of subsets in a Set Cover Reconfiguration instance of Theorem 4.1 may depend on the universe size N.

Proofs marked with $*$ are omitted and can be found in the full version of this paper [18].

1.3 Proof Overview

At a high level, our proofs of Theorems 1.1 and 1.3 are given by combining the ideas developed in [19, 31, 32, 26]. Karthik C. S. and Manurangsi [26] proved NP-hardness of the $(2-\varepsilon)$-factor approximation of Minmax Set Cover Reconfiguration as follows.

1. Starting from the PCP theorem for NP [3, 4], they applied the FGLSS reduction [12] to prove NP-hardness of the $\mathcal{O}(\varepsilon^{-1})$-factor approximation of an intermediate problem, which we call Max Partial 2CSP.
2. The $\mathcal{O}(\varepsilon^{-1})$-factor approximation of Max Partial 2CSP is reduced to the $(2-\varepsilon)$-factor approximation of a reconfiguration problem, which we call Label Cover Reconfiguration (Problem 2.3).
3. Label Cover Reconfiguration can be reduced to Minmax Set Cover Reconfiguration via approximation-preserving reductions of Lund and Yannakakis [27] and Ohsaka [32].

Here, Max Partial 2CSP is defined as follows. The input consists of a graph $G = (\mathcal{V}, \mathcal{E})$, a finite alphabet Σ, and constraints $\psi_e \colon \Sigma^2 \to \{0, 1\}$ for each edge $e \in \mathcal{E}$. A *partial assignment* is a function $f \colon \mathcal{V} \to \Sigma \cup \{\bot\}$, where the symbol $\bot$ indicates "unassigned." The task is to maximize the fraction of assigned vertices in a partial assignment f that *satisfies* ψ_e for every $e = (v, w) \in \mathcal{E}$; i.e., $\psi_e(f(v), f(w)) = 1$ if $f(v) \neq \bot$ and $f(w) \neq \bot$.

To improve this NP-hardness result to PSPACE-hardness, we replace the starting point with the PCRP (Probabilistically Checkable Reconfiguration Proof) system of Hirahara and Ohsaka [19], which is a reconfiguration analogue of the PCP theorem. We also replace Max Partial 2CSP with its reconfiguration analogue, which we call Partial 2CSP Reconfiguration (Problem 2.2). The proof of PSPACE-hardness is outlined as follows.

1. Starting from the *PCRP theorem* for PSPACE [19], we apply the FGLSS reduction [12] to prove PSPACE-hardness of Partial 2CSP Reconfiguration (Sections 3.1 and 3.2).
2. We reduce Partial 2CSP Reconfiguration to Label Cover Reconfiguration (Section 3.3).
3. We reduce Label Cover Reconfiguration to Minmax Set Cover Reconfiguration by the reductions of [32, 27] (Section 4.1).

The second and third steps are similar to the previous work [26]. Our main technical contribution lies in the first step, which we explain below.

Roughly speaking, the PCRP theorem [19] shows that any PSPACE computation on inputs of length n can be encoded into a sequence $\pi^{(1)}, \cdots, \pi^{(T)} \in \{0, 1\}^{\mathrm{poly}(n)}$ of exponentially many proofs such that any adjacent pair of proofs $\pi^{(t)}$ and $\pi^{(t+1)}$ differs in at most one bit, and each proof $\pi^{(t)}$ can be probabilistically checked by reading $q(n)$ bits of the proof and using $r(n)$ random bits, where $q(n) = \mathcal{O}(1)$ and $r(n) = \mathcal{O}(\log n)$. The FGLSS reduction [12] transforms such a proof system into a graph $G = (\mathcal{V}, \mathcal{E})$, an alphabet Σ, and constraints $(\psi_e)_{e \in \mathcal{E}}$ such that each vertex $v \in \mathcal{V} \coloneqq \{0, 1\}^{r(n)}$ corresponds to a coin flip sequence of a verifier, each value $\alpha \in \Sigma = \{0, 1\}^{q(n)}$ corresponds to a local view of the verifier, and the constraints ψ_e check the consistency of two local views of the verifier. This reduction works in the case of the PCP theorem and proves NP-hardness of Max Partial 2CSP [26]. However, the reduction does not work in the case of the PCRP theorem: We need to ensure that the reconfiguration sequence of proofs $\pi^{(1)}, \cdots, \pi^{(T)}$ is transformed into a sequence of partial assignments $f^{(1)}, \cdots, f^{(T)}$, each adjacent pair of which differs in at most one vertex. The issue is that changing one bit in the original proof $\pi^{(t)}$ may result in changing the assignments of many vertices in a partial assignment $f^{(t)} \colon \mathcal{V} \to \Sigma \cup \{\bot\}$.

To address this issue, we employ the ideas developed in [31, 32], called the *alphabet squaring trick*, and modify the FGLSS reduction as follows. Given a verifier that reads $q(n)$ bits of a proof, we define the alphabet as $\Sigma = \{0, 1, 01\}^{q(n)}$. Intuitively, the symbol "01" means that we are taking 0 and 1 simultaneously. This enables us to construct a reconfiguration sequence of partial assignments $f^{(1)}, \cdots, f^{(T)}$ from a reconfiguration sequence of proofs $\pi^{(1)}, \cdots, \pi^{(T)}$. Details can be found in Section 3.2.

1.4 Related Work

Ito, Demaine, Harvey, Papadimitriou, Sideri, Uehara, and Uno [23] showed that optimization versions of SAT Reconfiguration and Clique Reconfiguration are NP-hard to approximate, relying on NP-hardness of approximating Max SAT [15] and Max Clique [14], respectively. Note that their NP-hardness results are not optimal since SAT Reconfiguration and Clique Reconfiguration are PSPACE-complete. Toward PSPACE-hardness of approximation for reconfiguration problems, Ohsaka [31] proposed the *Reconfiguration Inapproximability Hypothesis* (RIH), which postulates that a reconfiguration analogue of

Constraint Satisfaction Problem is PSPACE-hard to approximate, and demonstrated PSPACE-hardness of approximation for many popular reconfiguration problems, including those of 3SAT, Independent Set, Vertex Cover, Clique, Dominating Set, and Set Cover. Ohsaka [32] adapted Dinur's gap amplification [9] to demonstrate that under RIH, optimization versions of 2CSP Reconfiguration and Set Cover Reconfiguration are PSPACE-hard to approximate within a factor of 0.9942 and 1.0029, respectively.

Very recently, Hirahara and Ohsaka [19] and Karthik C. S. and Manurangsi [26] announced the proof of RIH independently, implying that the above PSPACE-hardness results hold *unconditionally*. Karthik C. S. and Manurangsi [26] further proved that (optimization versions of) 2CSP Reconfiguration and Set Cover Reconfiguration are NP-hard to approximate within a factor smaller than 2, which is numerically tight because both problems are (nearly) 2-factor approximable. Our result partially resolves an open question of [26, Section 6]: "*Can we prove tight* PSPACE*-hardness of approximation results for GapMaxMin-2-*CSP_q *and Set Cover Reconfiguration?*"

Other reconfiguration problems whose approximability was investigated include those of Set Cover [23], Subset Sum [22], and Submodular Maximization [33]. We note that optimization variants of reconfiguration problems frequently refer to those of *the shortest reconfiguration sequence* [29, 5, 24, 25], which are orthogonal to this study.

2 Preliminaries

2.1 Notations

For a nonnegative integer $n \in \mathbb{N}$, let $[n] \coloneqq \{1, 2, \ldots, n\}$. Unless otherwise specified, the base of logarithms is 2. A *sequence* $\mathscr{S}$ of a finite number of objects $S^{(1)}, \ldots, S^{(T)}$ is denoted by $(S^{(1)}, \ldots, S^{(T)})$, and we write $S^{(t)} \in \mathscr{S}$ to indicate that $S^{(t)}$ appears in $\mathscr{S}$. Let Σ be a finite set called *alphabet*. For a length-n string $\pi \in \Sigma^n$ and a finite sequence of indices $I \subseteq [n]^*$, we use $\pi|_I \coloneqq (\pi_i)_{i \in I}$ to denote the restriction of π to I. The *Hamming distance* between two strings $f, g \in \Sigma^n$, denoted by $\Delta(f, g)$, is defined as the number of positions on which f and g differ; namely,

$$\Delta(f, g) \coloneqq \left| \left\{ i \in [n] \;\middle|\; f_i \neq g_i \right\} \right|. \tag{2.1}$$

2.2 Reconfiguration Problems on Constraint Graphs

Constraint Graphs. In this section, we formulate reconfiguration problems on constraint graphs. The notion of *constraint graph* is defined as follows.

▶ **Definition 2.1.** A *q-ary constraint graph* is defined as a tuple $G = (\mathcal{V}, \mathcal{E}, \Sigma, \Psi)$ such that
- $(\mathcal{V}, \mathcal{E})$ is a q-uniform[1] hypergraph called the *underlying graph*,
- Σ is a finite set called the *alphabet*, and
- $\Psi = (\psi_e)_{e \in \mathcal{E}}$ is a collection of q-ary *constraints*, where each $\psi_e \colon \Sigma^e \to \{0, 1\}$ is a circuit.

A binary constraint graph is simply referred to as a *constraint graph*. ⌟

For an *assignment* $f \colon \mathcal{V} \to \Sigma$, we say that f *satisfies* a hyperedge $e = \{v_1, \ldots, v_q\} \in \mathcal{E}$ (or a constraint ψ_e) if $\psi_e(f(e)) = 1$, where $f(e) \coloneqq (f(v_1), \ldots, f(v_q))$, and f *satisfies* G if it satisfies all the hyperedges of G. In the qCSP Reconfiguration problem, for a q-ary constraint

[1] A hypergraph is said to be *q-uniform* if each of its hyperedges has size exactly q.

graph G and its two satisfying assignments f^{start} and f^{goal}, we are required to decide if there exists a reconfiguration sequence from f^{start} to f^{goal} consisting only of satisfying assignments for G, each adjacent pair of which differs in at most one vertex. qCSP Reconfiguration is PSPACE-complete in general [13, 23]; thus, we formulate its two optimization versions.

Partial 2CSP Reconfiguration. For a constraint graph $G = (\mathcal{V}, \mathcal{E}, \Sigma, \Psi = (\psi_e)_{e\in\mathcal{E}})$, a *partial assignment* is defined as a function $f\colon \mathcal{V} \to \Sigma \cup \{\bot\}$, where the symbol $\bot$ indicates "unassigned." We say that a partial assignment $f\colon \mathcal{V} \to \Sigma \cup \{\bot\}$ *satisfies* edge $e = (v, w) \in \mathcal{E}$ if $\psi_e(f(v), f(w)) = 1$ whenever $f(v) \neq \bot$ and $f(w) \neq \bot$. The *size* of f, denoted by $\|f\|$, is defined as the number of vertices whose value is assigned; namely,

$$\|f\| \coloneqq \left|\left\{v \in \mathcal{V} \;\middle|\; f(v) \neq \bot\right\}\right|. \tag{2.2}$$

For two satisfying partial assignments f^{start} and f^{goal} for G, a *reconfiguration partial assignment sequence from f^{start} to f^{goal}* is a sequence $F = (f^{(1)}, \dots f^{(T)})$ of satisfying partial assignments such that $f^{(1)} = f^{\mathsf{start}}$, $f^{(T)} = f^{\mathsf{goal}}$, and $\Delta(f^{(t)}, f^{(t+1)}) \leqslant 1$ (i.e., $f^{(t)}$ and $f^{(t+1)}$ differ in at most one vertex) for all t. For any reconfiguration partial assignment sequence $F = (f^{(1)}, \dots, f^{(T)})$, we define $\|F\|_{\min}$ as

$$\|F\|_{\min} \coloneqq \min_{1\leqslant t\leqslant T} \|f^{(t)}\|. \tag{2.3}$$

Partial 2CSP Reconfiguration is formally defined as follows:

▶ **Problem 2.2** (Partial 2CSP Reconfiguration)**.** For a constraint graph $G = (\mathcal{V}, \mathcal{E}, \Sigma, \Psi)$ and its two satisfying partial assignments $f^{\mathsf{start}}, f^{\mathsf{goal}}\colon \mathcal{V} \to \Sigma \cup \{\bot\}$, we are required to find a reconfiguration partial assignment sequence F from f^{start} to f^{goal} such that $\|F\|_{\min}$ is maximized. ⌟

Let $\mathsf{MaxPar}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}})$ denote the maximum value of $\frac{\|F\|_{\min}}{|\mathcal{V}|}$ over all possible reconfiguration sequences F from f^{start} to f^{goal}; namely,

$$\mathsf{MaxPar}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) \coloneqq \max_{F=(f^{\mathsf{start}},\dots,f^{\mathsf{goal}})} \frac{\|F\|_{\min}}{|\mathcal{V}|}. \tag{2.4}$$

Note that $0 \leqslant \mathsf{MaxPar}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) \leqslant 1$. For every numbers $0 \leqslant s \leqslant c \leqslant 1$, $\textsc{Gap}_{c,s}$ Partial 2CSP Reconfiguration requests to determine for a constraint graph G and its two satisfying partial assignments f^{start} and f^{goal}, whether $\mathsf{MaxPar}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) \geqslant c$ or $\mathsf{MaxPar}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) < s$. Note that we can assume $\|f^{\mathsf{start}}\| = \|f^{\mathsf{goal}}\| = |\mathcal{V}|$ when $c = 1$.

Label Cover Reconfiguration. For a constraint graph $G = (\mathcal{V}, \mathcal{E}, \Sigma, \Psi = (\psi_e)_{e\in\mathcal{E}})$, a *multi-assignment* is defined as a function $f\colon \mathcal{V} \to 2^{\Sigma}$. We say that a multi-assignment f *satisfies* edge $e = (v, w) \in \mathcal{E}$ if there exists a pair $(\alpha, \beta) \in f(v) \times f(w)$ such that $\psi_e(\alpha, \beta) = 1$. The size of f, denoted by $\|f\|$, is defined as the sum of $|f(v)|$ over all $v \in \mathcal{V}$; namely,

$$\|f\| \coloneqq \sum_{v\in\mathcal{V}} |f(v)|. \tag{2.5}$$

For two satisfying multi-assignments f^{start} and f^{goal} for G, a *reconfiguration multi-assignment sequence from f^{start} to f^{goal}* is a sequence $F = (f^{(1)}, \dots, f^{(T)})$ of satisfying multi-assignments such that $f^{(1)} = f^{\mathsf{start}}$, $f^{(T)} = f^{\mathsf{goal}}$, and

$$\sum_{v\in\mathcal{V}} \left|f^{(t)}(v) \triangle f^{(t+1)}(v)\right| \leqslant 1 \quad \text{for all } t. \tag{2.6}$$

For any reconfiguration multi-assignment sequence $F = (f^{(1)}, \dots, f^{(T)})$, we define $\|F\|_{\max}$ as

$$\|F\|_{\max} \coloneqq \max_{1 \leqslant t \leqslant T} \|f^{(t)}\|. \tag{2.7}$$

Label Cover Reconfiguration is formally defined as follows.[2]

▶ **Problem 2.3** (Label Cover Reconfiguration)**.** For a constraint graph $G = (\mathcal{V}, \mathcal{E}, \Sigma, \Psi)$ and its two satisfying multi-assignments $f^{\mathsf{start}}, f^{\mathsf{goal}} \colon \mathcal{V} \to 2^{\Sigma}$, we are required to find a reconfiguration multi-assignment sequence F from f^{start} to f^{goal} such that $\|F\|_{\max}$ is minimized. ⌟

Let $\mathsf{MinLab}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}})$ denote the minimum value of $\frac{\|F\|_{\max}}{|\mathcal{V}|+1}$ over all possible reconfiguration multi-assignment sequences F from f^{start} to f^{goal}; namely,

$$\mathsf{MinLab}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) \coloneqq \min_{F=(f^{\mathsf{start}}, \dots, f^{\mathsf{goal}})} \frac{\|F\|_{\max}}{|\mathcal{V}|+1}. \tag{2.8}$$

Note that $\mathsf{MinLab}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) \geqslant \frac{|\mathcal{V}|}{|\mathcal{V}|+1}$. For every numbers $1 \leqslant c \leqslant s$, $\mathrm{Gap}_{c,s}$ Label Cover Reconfiguration requests to determine whether $\mathsf{MinLab}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) \leqslant c$ or $\mathsf{MinLab}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) > s$ for a constraint graph G and its two satisfying multi-assignments f^{start} and f^{goal}. Note that we can assume $\frac{\|f^{\mathsf{start}}\|}{|\mathcal{V}|+1} = \frac{\|f^{\mathsf{goal}}\|}{|\mathcal{V}|+1} \leqslant 1$ when $c = 1$.

2.3 Probabilistically Checkable Reconfiguration Proof Systems

First, we formally define the notion of *verifier*.

▶ **Definition 2.4.** A *verifier* with *randomness complexity* $r \colon \mathbb{N} \to \mathbb{N}$ and *query complexity* $q \colon \mathbb{N} \to \mathbb{N}$ is a probabilistic polynomial-time algorithm V that given an input $x \in \{0,1\}^*$, tosses $r = r(|x|)$ random bits R and uses R to generate a sequence of $q = q(|x|)$ queries $I = (i_1, \dots, i_q)$ and a circuit $D \colon \{0,1\}^q \to \{0,1\}$. We write $(I, D) \sim V(x)$ to denote the random variable for a pair of the query sequence and circuit generated by V on input $x \in \{0,1\}^*$ and r random bits, and write $(I, D) = V(x, R)$ when we wish to fix the random bits R. Denote by $V^{\pi}(x) \coloneqq D(\pi|_I)$, where $(I, D) = V(x, R)$ for $R \sim \{0,1\}^r$, the random variable for the output of V on input x given oracle access to a proof $\pi \in \{0,1\}^*$. We say that $V(x)$ *accepts* a proof π if $V^{\pi}(x) = 1$; i.e., $D(\pi|_I) = 1$ for $(I, D) \sim V(x)$. ⌟

We proceed to the definition of *Probabilistically Checkable Reconfiguration Proofs* (PCRPs) due to Hirahara and Ohsaka [19], which offer a PCP-type characterization of PSPACE. For any pair of proofs $\pi^{\mathsf{start}}, \pi^{\mathsf{goal}} \in \{0,1\}^{\ell}$, a *reconfiguration sequence from* π^{start} *to* π^{goal} is a sequence $(\pi^{(1)}, \dots, \pi^{(T)}) \in (\{0,1\}^{\ell})^*$ such that $\pi^{(1)} = \pi^{\mathsf{start}}$, $\pi^{(T)} = \pi^{\mathsf{goal}}$, and $\Delta(\pi^{(t)}, \pi^{(t+1)}) \leqslant 1$ (i.e., $\pi^{(t)}$ and $\pi^{(t+1)}$ differ in at most one bit) for all t.

▶ **Theorem 2.5** (PCRP theorem [19, Theorem 5.1])**.** *For any language L in* PSPACE*, there exists a verifier V with randomness complexity $r(n) = \mathcal{O}(\log n)$ and query complexity $q(n) = \mathcal{O}(1)$, coupled with polynomial-time computable functions $\pi^{\mathsf{start}}, \pi^{\mathsf{goal}} \colon \{0,1\}^* \to \{0,1\}^*$, such that the following hold for any input $x \in \{0,1\}^*$:*

- *(Completeness) If $x \in L$, there exists a reconfiguration sequence $\Pi = (\pi^{(1)}, \dots, \pi^{(T)})$ from $\pi^{\mathsf{start}}(x)$ to $\pi^{\mathsf{goal}}(x)$ over $\{0,1\}^{\mathrm{poly}(|x|)}$ such that $V(x)$ accepts every proof with probability 1; namely,*

$$\forall t \in [T], \quad \mathbb{P}\Big[V(x) \text{ accepts } \pi^{(t)}\Big] = 1. \tag{2.9}$$

[2] This problem can be thought of as a reconfiguration analogue of Min Rep [8].

- (*Soundness*) *If* $x \notin L$*, every reconfiguration sequence* $\Pi = (\pi^{(1)}, \dots, \pi^{(T)})$ *from* $\pi^{\mathsf{start}}(x)$ *to* $\pi^{\mathsf{goal}}(x)$ *over* $\{0,1\}^{\mathrm{poly}(|x|)}$ *includes a proof that is rejected by* $V(x)$ *with probability more than* $\frac{1}{2}$*; namely,*

$$\exists t \in [T], \quad \mathbb{P}\Big[V(x) \text{ accepts } \pi^{(t)}\Big] < \frac{1}{2}. \tag{2.10}$$

We further introduce the notion of regular verifier. We say that a verifier is *regular* if each position in its proof is equally likely to be queried.[3]

▶ **Definition 2.6.** For a verifier V and an input $x \in \{0,1\}^*$, the *degree* of a position i of a proof is defined as the number of times i is queried by $V(x)$ over $r(|x|)$ random bits; namely,

$$\left|\left\{R \in \{0,1\}^{r(|x|)} \;\middle|\; i \in I_R\right\}\right| = \mathop{\mathbb{P}}_{(I,D)\sim V(x)}\Big[i \in I\Big] \cdot 2^{r(|x|)}, \tag{2.11}$$

where r is the randomness complexity of V and I_R is the query sequence generated by $V(x,R)$. A verifier V is said to be Δ-*regular* if the degree of every position is exactly equal to Δ. ⌟

3 Subconstant Error PCRP Systems and FGLSS Reduction

We construct a bounded-degree PCRP verifier with subconstant error using Theorem 2.5 in Section 3.1, and prove PSPACE-hardness of approximation for Partial 2CSP Reconfiguration and Label Cover Reconfiguration by the FGLSS reduction [12] in Sections 3.2 and 3.3, respectively.

3.1 Bounded-degree PCRP Systems with Subconstant Error

Starting from Theorem 2.5, we first obtain a regular PCRP verifier for any PSPACE language, whose proof uses the degree reduction technique due to Ohsaka [31].

▶ **Proposition 3.1** (∗)**.** *For any language* L *in* PSPACE*, there exists a* Δ*-regular PCRP verifier* V *with randomness complexity* $r(n) = \mathcal{O}(\log n)$*, query complexity* $q(n) = \mathcal{O}(1)$*, perfect completeness, and soundness* $1-\varepsilon$*, for some constant* $\Delta \in \mathbb{N}$ *and* $\varepsilon \in (0,1)$*.*

Subsequently, using a randomness-efficient sampler over expander graphs (e.g., [21, Section 3]), we construct a *bounded-degree* PCRP verifier with *subconstant error*.

▶ **Proposition 3.2.** *For any language* L *in* PSPACE *and any function* $\delta\colon \mathbb{N} \to \mathbb{R}$ *with* $\delta(n) = \Omega(n^{-1})$*, there exists a bounded-degree PCRP verifier* V *with randomness complexity* $r(n) = \mathcal{O}(\log \delta(n)^{-1} + \log n)$*, query complexity* $q(n) = \mathcal{O}(\log \delta(n)^{-1})$*, perfect completeness, and soundness* $\delta(n)$*. Moreover, for any input* $x \in \{0,1\}^*$*, the degree of any position is* $\mathrm{poly}(\delta(|x|)^{-1})$*.*

Verifier Description. Our PCRP verifier is described as follows. By Proposition 3.1, let V be a Δ-regular PCRP verifier for a PSPACE-complete language L with randomness complexity $r(n) = \mathcal{O}(\log n)$, query complexity $q(n) = q \in \mathbb{N}$, perfect completeness, and soundness $1-\varepsilon$, where $\Delta \in \mathbb{N}$ and $\varepsilon \in (0,1)$. The proof length, denoted by $\ell(n)$, is polynomially bounded

[3] Note that regular verifiers are sometimes called *smooth* verifiers, e.g., [34]. Since the term "regularity" is compatible with that of (hyper)graphs, we do not use the term "smoothness" but "regularity."

since $\ell(n) \leqslant q(n)2^{r(n)} = \mathrm{poly}(n)$. Hereafter, for any $r(n)$ random bit sequence R, let I_R and D_R respectively denote the query sequence and circuit generated by $V(x, R)$. Given a function $\delta\colon \mathbb{N} \to \mathbb{R}$ with $\delta(n) = \Omega(n^{-1})$, we construct the following verifier $\widetilde{V}$:

Bounded-degree verifier $\widetilde{V}$ with subconstant error.

Input: a Δ-regular verifier V with soundness $1-\varepsilon$, a function $\delta\colon \mathbb{N} \to \mathbb{R}$, and an input $x \in \{0,1\}^n$.
Oracle access: a proof $\pi \in \{0,1\}^{\ell(n)}$.
1: construct a (d, λ)-expander graph X over vertex set $\{0,1\}^{r(n)}$ with $\frac{\lambda}{d} < \frac{\varepsilon}{4}$.
2: let $\rho \coloneqq \left\lceil \frac{2}{\varepsilon} \ln \delta(n)^{-1} \right\rceil = \mathcal{O}(\log \delta(n)^{-1})$.
3: uniformly sample a $(\rho-1)$-length random walk $\mathbf{R} = (R_1, \ldots, R_\rho)$ over X using $r(n) + \rho \cdot \log d$ random bits.
4: **for each** $1 \leqslant k \leqslant \rho$ **do**
5: execute $V(x)$ on R_k to generate a query sequence $I_{R_k} = (i_1, \ldots, i_q)$ and a circuit $D_{R_k}\colon \{0,1\}^q \to \{0,1\}$.
6: **if** $D_{R_k}(\pi|_{I_{R_k}}) = 0$ **then**
7: declare reject.
8: declare accept.

Correctness. The perfect completeness and soundness for a fixed proof $\pi \in \{0,1\}^{\ell(n)}$ are shown below, whose proof relies on the property about random walks over expander graphs due to Alon, Feige, Wigderson, and Zuckerman [2].

▷ Claim 3.3 (∗). If $V(x)$ accepts π with probability 1, then $\widetilde{V}(x)$ accepts π with probability 1. If $V(x)$ accepts π with probability less than $1-\varepsilon$, then $\widetilde{V}(x)$ accepts π with probability less than $\delta(n)$.

We are now ready to prove Proposition 3.2.

Proof of Proposition 3.2. We first show the perfect completeness and soundness. Suppose $x \in L$, then there exists a reconfiguration sequence $\Pi = (\pi^{(1)}, \ldots, \pi^{(T)})$ from $\pi^{\mathsf{start}}(x)$ to $\pi^{\mathsf{goal}}(x)$ such that $\mathbb{P}[V(x) \text{ accepts } \pi^{(t)}] = 1$ for all t. By Claim 3.3, we have that $\mathbb{P}[\widetilde{V}(x) \text{ accepts } \pi^{(t)}] = 1$ for all t. Suppose $x \notin L$, then for every reconfiguration sequence $\Pi = (\pi^{(1)}, \ldots, \pi^{(T)})$ from $\pi^{\mathsf{start}}(x)$ to $\pi^{\mathsf{goal}}(x)$, it holds that $\mathbb{P}[V(x) \text{ accepts } \pi^{(t)}] < 1-\varepsilon$ for some t. By Claim 3.3, we have $\mathbb{P}[\widetilde{V}(x) \text{ accepts } \pi^{(t)}] < \delta(n)$ for such t.

Since $\rho = \mathcal{O}(\log \delta(n)^{-1})$, the randomness complexity of $\widetilde{V}$ is equal to $\widetilde{r}(n) = r(n) + \rho \cdot \log d = \mathcal{O}(\log \delta(n)^{-1} + \log n)$, and the query complexity is $\widetilde{q}(n) = q(n) \cdot \rho = \mathcal{O}(\log \delta(n)^{-1})$. Note that d and λ may depend only on ε, and a (d, λ)-expander graph X over $\{0,1\}^{r(n)}$ can be constructed in polynomial time in $2^{r(n)} = \mathrm{poly}(n)$, e.g., by using an explicit construction of near-Ramanujan graphs [28, 1].

Observe finally that $\widetilde{V}$ queries each position $i \in [\ell(n)]$ of a proof with probability equal to

$$\mathop{\mathbb{P}}_{\mathbf{R}}\left[\bigvee_{1 \leqslant k \leqslant \rho} \left(i \in I_{R_k}\right)\right]. \tag{3.1}$$

Since V is Δ-regular, it holds that

$$\mathop{\mathbb{P}}_{R \sim \{0,1\}^{r(n)}}\left[i \in I_R\right] = \frac{\Delta}{2^{r(n)}}. \tag{3.2}$$

Using the fact that each R_k is uniformly distributed over $\{0,1\}^{r(n)}$, we bound Eq. (3.1) as follows:

$$\mathop{\mathbb{P}}_{\mathbf{R}}\left[\bigvee_{1\leqslant k\leqslant \rho}\left(i\in I_{R_k}\right)\right] \underbrace{\leqslant}_{\text{union bound}} \sum_{k\in[\rho]}\mathop{\mathbb{P}}_{\mathbf{R}}\left[i\in I_{R_k}\right] = \frac{\rho\cdot\Delta}{2^{r(n)}} = \mathcal{O}\left(\frac{\log\delta(n)^{-1}}{2^{r(n)}}\right). \tag{3.3}$$

Consequently, the degree of each position i with respect to $\widetilde{V}$ is at most

$$\begin{aligned}\mathop{\mathbb{P}}_{\mathbf{R}}\left[\bigvee_{1\leqslant k\leqslant \rho}\left(i\in I_{R_k}\right)\right]\cdot 2^{\widetilde{r(n)}} &= \mathcal{O}\left(\frac{\log\delta(n)^{-1}}{2^{r(n)}}\right)\cdot 2^{r(n)+\rho\cdot\log d}\\ &= \mathcal{O}(\log\delta(n)^{-1})\cdot 2^{\mathcal{O}(\log\delta(n)^{-1})}\\ &= \mathrm{poly}(\delta(n)^{-1}),\end{aligned} \tag{3.4}$$

which completes the proof. ◀

3.2 FGLSS Reduction and PSPACE-hardness of Approximation for Partial 2CSP Reconfiguration

We now establish the FGLSS reduction from Proposition 3.2 and show that Partial 2CSP Reconfiguration is PSPACE-hard to approximate within a factor arbitrarily close to 0.

▶ **Theorem 3.4.** *For any function $\varepsilon\colon\mathbb{N}\to\mathbb{R}$ with $\varepsilon(n)=\Omega\left(\frac{1}{\mathrm{polylog}\, n}\right)$, $\mathrm{Gap}_{1,\varepsilon(N)}$ Partial 2CSP Reconfiguration with alphabet size $\mathrm{poly}(\varepsilon(N)^{-1})$ is PSPACE-complete, where N is the number of vertices.*

Reduction. We describe a reduction from a bounded-degree PCRP verifier to Partial 2CSP Reconfiguration. Define $\delta(n) \coloneqq \frac{\varepsilon(\mathrm{poly}(n))}{2}$, whose precise expression is given later. For any PSPACE-complete language L, let V be a bounded-degree PCRP verifier of Proposition 3.2 with randomness complexity $r(n)=\mathcal{O}(\log\delta(n)^{-1}+\log n)$, query complexity $q(n)=\mathcal{O}(\log\delta(n)^{-1})$, perfect completeness, and soundness $\delta(n)$. The proof length $\ell(n)$ is polynomially bounded. Suppose we are given an input $x\in\{0,1\}^n$. Let $\pi^{\mathsf{start}},\pi^{\mathsf{goal}}\in\{0,1\}^{\ell(n)}$ be the two proofs associated with $V(x)$. Because the degree of V is bounded by $\mathrm{poly}(\delta(n)^{-1})$, for some constant $\kappa\in\mathbb{N}$, we have

$$\mathop{\mathbb{P}}_{(I,D)\sim V(x)}\left[i\in I\right] \leqslant \frac{\delta(n)^{-\kappa}}{2^{r(n)}} \text{ for any } i\in[\ell(n)]. \tag{3.5}$$

Hereafter, for any $r(n)$ random bit sequence R, let I_R and D_R denote the query sequence and the circuit generated by $V(x,R)$, respectively. Construct a constraint graph $G=(\mathcal{V},\mathcal{E},\Sigma,\Psi)$ as follows:

$$\mathcal{V} \coloneqq \{0,1\}^{r(n)}, \tag{3.6}$$

$$\mathcal{E} \coloneqq \left\{(R_1,R_2)\in\mathcal{V}\times\mathcal{V}\;\middle|\; I_{R_1}\cap I_{R_2}\neq\emptyset\right\}, \tag{3.7}$$

$$\Sigma \coloneqq \left\{\{0\},\{1\},\{0,1\}\right\}^{q(n)}, \tag{3.8}$$

$$\Psi \coloneqq \{\psi_e\}_{e\in\mathcal{E}}, \tag{3.9}$$

where we define $\psi_{R_1,R_2} : \Sigma \times \Sigma \to \{0,1\}$ for edge $(R_1, R_2) \in \mathcal{E}$ so that $\psi_{R_1,R_2}(f(R_1), f(R_2)) = 1$ for an assignment $f : \mathcal{V} \to \Sigma$ if and only if the following three conditions are satisfied:

$$\forall \alpha \in \prod_{i \in I_R} f(R_1)_i, \quad D_{R_1}(\alpha) = 1, \tag{3.10}$$

$$\forall \beta \in \prod_{i \in I_R} f(R_2)_i, \quad D_{R_2}(\beta) = 1, \tag{3.11}$$

$$\forall i \in I_{R_1} \cap I_{R_2}, \quad f(R_1)_i \subseteq f(R_2)_i \text{ or } f(R_1)_i \supseteq f(R_2)_i. \tag{3.12}$$

Here, for the sake of notation simplicity, we consider $f(R)$ as if it were indexed by I_R (rather than $[q(n)]$); namely, $f(R) \in \{\{0\},\{1\},\{0,1\}\}^{I_R}$. Thus, $f(R)$ for each $R \in \mathcal{V}$ corresponds the local view of $V(x, R)$.

For any proof $\pi \in \{0,1\}^{\ell(n)}$, we associate it with an assignment $f_\pi : \mathcal{V} \to \Sigma$ such that

$$f_\pi(R) \coloneqq \Big(\{\pi_i\}\Big)_{i \in I_R} \text{ for all } R \in \mathcal{V}. \tag{3.13}$$

Note that $f_\pi(R) \in \{\{0\},\{1\}\}^{I_R}$. Constructing two assignments f^{start} from π^{start} and f^{goal} from π^{goal} by Eq. (3.13), we obtain an instance $(G, f^{\mathsf{start}}, f^{\mathsf{goal}})$ of Partial 2CSP Reconfiguration. Observe that f^{start} and f^{goal} satisfy G and $\|f^{\mathsf{start}}\| = \|f^{\mathsf{goal}}\| = |\mathcal{V}|$. Note that $N \coloneqq |\mathcal{V}| \leqslant n^c$ for some constant $c \in \mathbb{N}$. Letting $\delta(n) \coloneqq \frac{\varepsilon(n^c)}{2} = \Omega\left(\frac{1}{\operatorname{polylog} n}\right)$ ensures that the alphabet size is $|\Sigma| = \mathcal{O}(3^{q(n)}) = \operatorname{poly}(\varepsilon(N)^{-1})$. This completes the description of the reduction.

Correctness. We first prove the completeness.

▶ **Lemma 3.5** (Completeness). *If $x \in L$, then* $\mathsf{MaxPar}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) = 1$.

Proof. It is sufficient to consider the case that π^{start} and π^{goal} differ in exactly one position, say, $i^\star \in [\ell(n)]$; namely, $\pi^{\mathsf{start}}_{i^\star} \neq \pi^{\mathsf{goal}}_{i^\star}$ and $\pi^{\mathsf{start}}_i = \pi^{\mathsf{goal}}_i$ for all $i \neq i^\star$. Note that f^{start} and f^{goal} may differ in two or more vertices. Consider a reconfiguration partial assignment sequence F from f^{start} to f^{goal} obtained by the following procedure:

Reconfiguration sequence F from f^{start} to f^{goal}.

1: **for each** $R \in \mathcal{V}$ such that $i^\star \in I_R$ **do**
2: change the $i^{\star\text{th}}$ entry of R's current value from $f^{\mathsf{start}}(R)_{i^\star} = \{\pi^{\mathsf{start}}_{i^\star}\}$ to $\{0,1\}$.
3: **for each** $R \in \mathcal{V}$ such that $i^\star \in I_R$ **do**
4: change the $i^{\star\text{th}}$ entry of R's current value from $\{0,1\}$ to $f^{\mathsf{goal}}(R)_{i^\star} = \{\pi^{\mathsf{goal}}_{i^\star}\}$.

Observe that any partial assignment f° of F satisfies G for the following reasons:

- Since $f^\circ(R)_{i^\star} \subseteq \{0,1\} = \{\pi^{\mathsf{start}}_{i^\star}, \pi^{\mathsf{goal}}_{i^\star}\} = f^{\mathsf{start}}(R)_{i^\star} \cup f^{\mathsf{goal}}(R)_{i^\star}$ when $i^\star \in I_R$, f° satisfies Eqs. (3.10) and (3.11).
- Letting $K \coloneqq \{f^\circ(R)_{i^\star} \mid i^\star \in I_R\}$, we find K to be either $\{\{0\}\}$, $\{\{1\}\}$, $\{\{0,1\}\}$, $\{\{0\},\{0,1\}\}$, or $\{\{1\},\{0,1\}\}$ by construction; i.e., f° satisfies Eq. (3.12).

Since $\|f^\circ\| = |\mathcal{V}|$, it holds that $\mathsf{MaxPar}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) \geqslant \frac{\|F\|_{\max}}{|\mathcal{V}|} = 1$, completing the proof. ◀

▶ **Lemma 3.6** (Soundness). *If $x \notin L$, then*

$$\mathsf{MaxPar}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) < \delta(n) + \frac{q(n) \cdot \delta(n)^{-\kappa}}{2^{r(n)}}. \tag{3.14}$$

The proof of Theorem 3.4 follows from Lemmas 3.5 and 3.6 because for any sufficiently large n such that $\frac{q(n)\cdot\delta(n)^{-\kappa}}{2^{r(n)}} \leqslant \delta(n)$ (note that $\delta(n) = \Omega\left(\frac{1}{\mathrm{polylog}\, n}\right)$), the following hold:

- (Perfect completeness) If $x \in L$, then $\mathsf{MaxPar}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) = 1$;
- (Soundness) If $x \notin L$, then $\mathsf{MaxPar}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) < 2\delta(n) = \varepsilon(N)$.

Proof of Lemma 3.6. We prove the contrapositive. Suppose $\mathsf{MaxPar}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) \geqslant \Gamma$ for some $\Gamma \in (0,1)$, and there is a reconfiguration partial assignment sequence $F = (f^{(1)}, \dots, f^{(T)})$ from f^{start} to f^{goal} such that $\|F\|_{\min} = \mathsf{MaxPar}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}})$. Define then a (not necessarily reconfiguration) sequence $\Pi = (\pi^{(1)}, \dots, \pi^{(T)})$ over $\{0,1\}^{\ell(n)}$ such that each proof $\pi^{(t)}$ is determined based on the *plurality vote* over $f^{(t)}$; namely,

$$\pi_i^{(t)} \coloneqq \operatorname*{argmax}_{b\in\{0,1\}} \left|\left\{R \in \mathcal{V} \;\middle|\; i \in I_R \text{ and } b \in f^{(t)}(R)_i\right\}\right| \text{ for all } i \in [\ell(n)], \tag{3.15}$$

where ties are broken so that 0 is chosen. In particular, $\pi^{(1)} = \pi^{\mathsf{start}}$ and $\pi^{(T)} = \pi^{\mathsf{goal}}$. Observe the following:

▶ **Observation 3.7.** *For any $t \in [T]$ and $R \in \mathcal{V}$, it holds that*

$$f^{(t)}(R) \neq \bot \implies D_R(\pi^{(t)}|_{I_R}) = 1. \tag{3.16}$$

Since $\mathbb{P}_{R\sim\mathcal{V}}[f^{(t)}(R) \neq \bot] = \|f^{(t)}\| \geqslant \Gamma$, by Observation 3.7, we have that for all t,

$$\mathbb{P}\Big[V(x) \text{ accepts } \pi^{(t)}\Big] = \mathop{\mathbb{P}}_{R\sim\{0,1\}^{r(n)}}\Big[D_R(\pi^{(t)}|_{I_R}) = 1\Big] \geqslant \mathop{\mathbb{P}}_{R\sim\mathcal{V}}\Big[f^{(t)}(R) \neq \bot\Big] \geqslant \Gamma. \tag{3.17}$$

Unfortunately, Π is *not* a reconfiguration sequence because $\pi^{(t)}$ and $\pi^{(t+1)}$ may differ in two or more positions. Since $f^{(t)}$ and $f^{(t+1)}$ differ in a single vertex $R \in \mathcal{V}$, we have $\pi_i^{(t)} \neq \pi_i^{(t+1)}$ only if $i \in I_R$, implying $\Delta(\pi^{(t)}, \pi^{(t+1)}) \leqslant |I_R| = q(n)$. Using this fact, we interpolate between $\pi^{(t)}$ and $\pi^{(t+1)}$ to find a valid reconfiguration sequence $\Pi^{(t)}$ such that $V(x)$ accepts every proof of $\Pi^{(t)}$ with probability $\Gamma - o(1)$.

▷ Claim 3.8. There exists a reconfiguration sequence $\Pi^{(t)}$ from $\pi^{(t)}$ to $\pi^{(t+1)}$ such that for every proof π° of $\Pi^{(t)}$,

$$\mathbb{P}\Big[V(x) \text{ accepts } \pi^\circ\Big] \geqslant \Gamma - \frac{q(n)\cdot\delta(n)^{-\kappa}}{2^{r(n)}}. \tag{3.18}$$

Concatenating $\Pi^{(t)}$'s of Claim 3.8 for all t, we obtain a valid reconfiguration sequence Π from π^{start} to π^{goal} such that

$$\min_{1\leqslant t\leqslant T} \mathbb{P}\Big[V(x) \text{ accepts } \pi^{(t)}\Big] \geqslant \Gamma - \frac{q(n)\cdot\delta(n)^{-\kappa}}{2^{r(n)}}. \tag{3.19}$$

Substituting $\delta(n) + \frac{q(n)\cdot\delta(n)^{-\kappa}}{2^{r(n)}}$ for Γ, we have that if $\mathsf{MaxPar}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) \geqslant \delta(n) + \frac{q(n)\cdot\delta(n)^{-\kappa}}{2^{r(n)}}$, then $V(x)$ accepts every proof $\pi^{(t)}$ of Π with probability at least $\delta(n)$; i.e., $x \in L$. This completes the proof of Lemma 3.6. ◀

What remains to be done is to prove Observation 3.7 and Claim 3.8.

Proof of Observation 3.7. Suppose $f^{(t)}(R) \neq \bot$ for some $t \in [T]$ and $R \in \mathcal{V}$. We will show that $\pi_i^{(t)} \in f^{(t)}(R)_i$ for every $i \in I_R$. Define

$$K \coloneqq \left\{f^{(t)}(R')_i \;\middle|\; \exists R' \in \mathcal{V} \text{ s.t. } i \in I_{R'} \text{ and } f^{(t)}(R') \neq \bot\right\}. \tag{3.20}$$

Then, any pair $\alpha, \beta \in K$ must satisfy that $\alpha \subseteq \beta$ or $\alpha \supseteq \beta$ because otherwise, $f^{(t)}$ would violate Eq. (3.12) at edge (R_1, R_2) such that $i \in R_1 \cap R_2$, $f^{(t)}(R_1)_i = \alpha$, and $f^{(t)}(R_2)_i = \beta$, which is a contradiction. For each possible case of K, the result of the plurality vote $\pi_i^{(t)}$ is shown below, implying that $\pi_i^{(t)} \in f^{(t)}(R)_i$.

K	$\{\}$	$\{\{0\}\}$	$\{\{1\}\}$	$\{\{0,1\}\}$	$\{\{0\},\{0,1\}\}$	$\{\{1\},\{0,1\}\}$
$\pi_i^{(t)}$	0	0	1	0	0	1

Since $f^{(t)}(R)$ must satisfy a self-loop $(R, R) \in \mathcal{E}$, by the definition of $\psi_{R,R}$, we have

$$\forall \alpha \in \prod_{i \in I_R} f^{(t)}(R)_i, \quad D_R(\alpha) = 1, \tag{3.21}$$

On the other hand, it holds that

$$\pi^{(t)}|_{I_R} \in \prod_{i \in I_R} f^{(t)}(R)_i, \tag{3.22}$$

implying $D_R(\pi^{(t)}|_{I_R}) = 1$, as desired. ◀

Proof of Claim 3.8. Recall that $\pi^{(t)}$ and $\pi^{(t+1)}$ may differ in at most $q(n)$ positions. Consider any trivial reconfiguration sequence $\Pi^{(t)}$ from $\pi^{(t)}$ to $\pi^{(t+1)}$ by simply changing at most $q(n)$ positions on which $\pi^{(t)}$ and $\pi^{(t+1)}$ differ. By construction, any proof π° of $\Pi^{(t)}$ differs from $\pi^{(t)}$ in at most $q(n)$ positions, say, $I^\circ \in \binom{[\ell(n)]}{\leqslant q(n)}$. Then, we derive the following:

$$\begin{aligned}
&\mathbb{P}\Big[V(x) \text{ accepts } \pi^\circ\Big] \\
&= \mathop{\mathbb{P}}_{(I,D)\sim V(x)}\Big[D(\pi^\circ|_I) = 1\Big] \geqslant \mathop{\mathbb{P}}_{(I,D)\sim V(x)}\Big[D(\pi^\circ|_I) = 1 \text{ and } I \cap I^\circ = \emptyset\Big] \\
&= \mathop{\mathbb{P}}_{(I,D)\sim V(x)}\Big[D(\pi^{(t)}|_I) = 1 \text{ and } I \cap I^\circ = \emptyset\Big] \\
&= \underbrace{\mathop{\mathbb{P}}_{(I,D)\sim V(x)}\Big[D(\pi^{(t)}|_I) = 1\Big]}_{=\mathbb{P}\left[V(x) \text{ accepts } \pi^{(t)}\right] \geqslant \Gamma} - \mathop{\mathbb{P}}_{(I,D)\sim V(x)}\Big[D(\pi^{(t)}|_I) = 1 \text{ and } I \cap I^\circ \neq \emptyset\Big] \\
&\geqslant \Gamma - \mathop{\mathbb{P}}_{(I,D)\sim V(x)}\Big[I \cap I^\circ \neq \emptyset\Big].
\end{aligned} \tag{3.23}$$

Recall that $\mathbb{P}_{(I,D)\sim V(x)}[i \in I] \leqslant \frac{\delta(n)^{-\kappa}}{2^{r(n)}}$ for any $i \in [\ell(n)]$ by assumption. Since $|I^\circ| \leqslant q(n)$, taking a union bound, we have

$$\mathop{\mathbb{P}}_{(I,D)\sim V(x)}\Big[I \cap I^\circ \neq \emptyset\Big] \leqslant \sum_{i \in I^\circ} \mathop{\mathbb{P}}_{(I,D)\sim V(x)}\Big[i \in I\Big] \leqslant \frac{q(n) \cdot \delta(n)^{-\kappa}}{2^{r(n)}}, \tag{3.24}$$

implying that

$$\mathbb{P}\Big[V(x) \text{ accepts } \pi^\circ\Big] \geqslant \Gamma - \frac{q(n) \cdot \delta(n)^{-\kappa}}{2^{r(n)}}. \tag{3.25}$$

This completes the proof. ◁

3.3 Reducing Partial 2CSP Reconfiguration to Label Cover Reconfiguration

Subsequently, we show PSPACE-hardness of approximation for Label Cover Reconfiguration by reducing from Partial 2CSP Reconfiguration, whose proof is similar to [26]. Note that Label Cover Reconfiguration admits a 2-factor approximation, similarly to Minmax Set Cover Reconfiguration (see Section 4.1).

▶ **Theorem 3.9** (∗). *For any function* $\varepsilon\colon \mathbb{N} \to \mathbb{R}$ *with* $\varepsilon(n) = \Omega\left(\frac{1}{\operatorname{polylog} n}\right)$, $\textsc{Gap}_{1,2-\varepsilon(N)}$ Label Cover Reconfiguration *with alphabet size* $\operatorname{poly}(\varepsilon(N)^{-1})$ *is* PSPACE*-complete, where* N *is the number of vertices. In particular,*

- *for any constant* $\varepsilon \in (0,1)$, $\textsc{Gap}_{1,2-\varepsilon}$ Label Cover Reconfiguration *with constant alphabet size is* PSPACE*-complete, and*
- $\textsc{Gap}_{1,2-\frac{1}{\operatorname{polyloglog} N}}$ Label Cover Reconfiguration *with alphabet size* $\operatorname{polyloglog} N$ *is* PSPACE*-complete.*

4 Applications

In this section, we apply Theorem 3.9 to show optimal PSPACE-hardness of approximation results for Minmax Set Cover Reconfiguration (Theorem 4.1) and Minmax Hypergraph Vertex Cover Reconfiguration (Theorem 4.3).

4.1 PSPACE-hardness of Approximation for Set Cover Reconfiguration

We first prove that Minmax Set Cover Reconfiguration is PSPACE-hard to approximate within a factor smaller than 2. Let $\mathcal{U}$ be a finite set called the *universe* and $\mathcal{F} = \{S_1, \ldots, S_m\}$ be a family of m subsets of $\mathcal{U}$. A *cover* for a set system $(\mathcal{U}, \mathcal{F})$ is a subfamily of $\mathcal{F}$ whose union is equal to $\mathcal{U}$. For any pair of covers $\mathcal{C}^{\mathsf{start}}$ and $\mathcal{C}^{\mathsf{goal}}$ for $(\mathcal{U}, \mathcal{F})$, a *reconfiguration sequence from* $\mathcal{C}^{\mathsf{start}}$ *to* $\mathcal{C}^{\mathsf{goal}}$ is a sequence $\mathscr{C} = (\mathcal{C}^{(1)}, \ldots, \mathcal{C}^{(T)})$ of covers such that $\mathcal{C}^{(1)} = \mathcal{C}^{\mathsf{start}}$, $\mathcal{C}^{(T)} = \mathcal{C}^{\mathsf{goal}}$, and $|\mathcal{C}^{(t)} \triangle \mathcal{C}^{(t+1)}| \leqslant 1$ (i.e., $\mathcal{C}^{(t+1)}$ is obtained from $\mathcal{C}^{(t)}$ by adding or removing a single set of $\mathcal{F}$) for all t. In Set Cover Reconfiguration [23], for a set system $(\mathcal{U}, \mathcal{F})$ and its two covers $\mathcal{C}^{\mathsf{start}}$ and $\mathcal{C}^{\mathsf{goal}}$ of size k, we are asked to decide if there is a reconfiguration sequence from $\mathcal{C}^{\mathsf{start}}$ to $\mathcal{C}^{\mathsf{goal}}$ consisting only of covers of size at most $k+1$. Next, we formulate its optimization version. Denote by $\mathsf{opt}(\mathcal{F})$ the size of the minimum cover of $(\mathcal{U}, \mathcal{F})$. For any reconfiguration sequence $\mathscr{C} = (\mathcal{C}^{(1)}, \ldots, \mathcal{C}^{(T)})$, its *cost* is defined as the maximum value of $\frac{|\mathcal{C}^{(t)}|}{\mathsf{opt}(\mathcal{F})+1}$ over all $\mathcal{C}^{(t)}$'s in $\mathscr{C}$; namely,[4]

$$\mathsf{cost}_{\mathcal{F}}(\mathscr{C}) \coloneqq \max_{\mathcal{C}^{(t)} \in \mathscr{C}} \frac{|\mathcal{C}^{(t)}|}{\mathsf{opt}(\mathcal{F}) + 1}, \tag{4.1}$$

In Minmax Set Cover Reconfiguration, we wish to minimize $\mathsf{cost}_{\mathcal{F}}(\mathscr{C})$ subject to $\mathscr{C} = (\mathcal{C}^{\mathsf{start}}, \ldots, \mathcal{C}^{\mathsf{goal}})$. For a pair of covers $\mathcal{C}^{\mathsf{start}}$ and $\mathcal{C}^{\mathsf{goal}}$ for $(\mathcal{U}, \mathcal{F})$, let $\mathsf{cost}_{\mathcal{F}}(\mathcal{C}^{\mathsf{start}} \leftrightsquigarrow \mathcal{C}^{\mathsf{goal}})$ denote the minimum value of $\mathsf{cost}_{\mathcal{F}}(\mathscr{C})$ over all possible reconfiguration sequences $\mathscr{C}$ from $\mathcal{C}^{\mathsf{start}}$ to $\mathcal{C}^{\mathsf{goal}}$; namely,

$$\mathsf{cost}_{\mathcal{F}}(\mathcal{C}^{\mathsf{start}} \leftrightsquigarrow \mathcal{C}^{\mathsf{goal}}) \coloneqq \min_{\mathscr{C} = (\mathcal{C}^{\mathsf{start}}, \ldots, \mathcal{C}^{\mathsf{goal}})} \mathsf{cost}_{\mathcal{F}}(\mathscr{C}). \tag{4.2}$$

For every $1 \leqslant c \leqslant s$, $\textsc{Gap}_{c,s}$ Set Cover Reconfiguration requests to distinguish whether $\mathsf{cost}_{\mathcal{F}}(\mathcal{C}^{\mathsf{start}} \leftrightsquigarrow \mathcal{C}^{\mathsf{goal}}) \leqslant c$ or $\mathsf{cost}_{\mathcal{F}}(\mathcal{C}^{\mathsf{start}} \leftrightsquigarrow \mathcal{C}^{\mathsf{goal}}) > s$.

[4] Here, division by $\mathsf{opt}(\mathcal{F}) + 1$ is derived from the nature that we must first add at least one set whenever $|\mathcal{C}^{\mathsf{start}}| = |\mathcal{C}^{\mathsf{goal}}| = \mathsf{opt}(\mathcal{F})$ and $\mathcal{C}^{\mathsf{start}} \neq \mathcal{C}^{\mathsf{goal}}$.

For the sake of completeness, we here present a 2-factor approximation algorithm for Minmax Set Cover Reconfiguration of [23]:[5]

2-factor approximation for Minmax Set Cover Reconfiguration.

1: ▷ *start from* $\mathcal{C}^{\mathsf{start}}$. ◁
2: insert each set of $\mathcal{C}^{\mathsf{goal}} \setminus \mathcal{C}^{\mathsf{start}}$ into the current cover in any order.
3: discard each set of $\mathcal{C}^{\mathsf{start}} \setminus \mathcal{C}^{\mathsf{goal}}$ from the current cover in any order.
4: ▷ *end with* $\mathcal{C}^{\mathsf{goal}}$. ◁

Our main result is stated below, whose proof uses a gap-preserving reduction from Label Cover Reconfiguration to Minmax Set Cover Reconfiguration [32, 27].

▶ **Theorem 4.1.** $\mathrm{Gap}_{1,2-\frac{1}{\mathrm{polyloglog}\, N}}$ Set Cover Reconfiguration *is* PSPACE*-complete, where* N *is the universe size. In particular,* Minmax Set Cover Reconfiguration *is* PSPACE*-hard to approximate within a factor of* $2 - \frac{1}{\mathrm{polyloglog}\, N}$.

Theorem 4.1 along with [32] implies that Minmax Dominating Set Reconfiguration is PSPACE-hard to approximate within a factor of $2 - \frac{1}{\mathrm{polyloglog}\, N}$, where N is the number of vertices (see Corollary 1.2).

Proof of Theorem 4.1. The reduction from Label Cover Reconfiguration to Minmax Set Cover Reconfiguration is almost the same as that due to Lund and Yannakakis [27] and Ohsaka [32]. Let $(G = (\mathcal{V}, \mathcal{E}, \Sigma, \Psi), f^{\mathsf{start}}, f^{\mathsf{goal}})$ be an instance of Label Cover Reconfiguration with N vertices and alphabet size $|\Sigma| = \mathrm{polyloglog}\, N$, where $\|f^{\mathsf{start}}\| = \|f^{\mathsf{goal}}\| = |\mathcal{V}|$. Define $B \coloneqq \{0,1\}^{\Sigma}$. For each $\alpha \in \Sigma$ and $S \subseteq \Sigma$, we construct $\overline{Q_\alpha} \subset B$ and $Q_S \subset B$ according to [32] in $2^{\mathcal{O}(|\Sigma|)}$ time. Let $\prec$ be an arbitrary order over V. Create an instance of Minmax Set Cover Reconfiguration as follows. For each vertex $v \in \mathcal{V}$ and each value $\alpha \in \Sigma$, we define $S_{v,\alpha} \subset \mathcal{E} \times B$ as

$$S_{v,\alpha} \coloneqq \left(\bigcup_{e=(v,w)\in\mathcal{E}: v\prec w} \{e\} \times \overline{Q_\alpha} \right) \cup \left(\bigcup_{e=(v,w)\in\mathcal{E}: v\succ w} \{e\} \times Q_{\pi_e(\alpha)} \right), \tag{4.3}$$

where $\pi_e(\alpha) \coloneqq \{\beta \in \Sigma \mid \psi_e(\alpha,\beta) = 1\}$. Then, a set system $(\mathcal{U}, \mathcal{F})$ is defined as

$$\mathcal{U} \coloneqq \mathcal{E} \times B \text{ and } \mathcal{F} \coloneqq \left\{ S_{v,\alpha} \;\middle|\; v \in \mathcal{V}, \alpha \in \Sigma \right\}. \tag{4.4}$$

For a satisfying multi-assignment $f\colon \mathcal{V} \to 2^{\Sigma}$ for G with $\|f\| = |\mathcal{V}|$,[6] we associate it with a subfamily $\mathcal{C}_f \subset \mathcal{F}$ such that

$$\mathcal{C}_f \coloneqq \left\{ S_{v,\alpha} \;\middle|\; v \in \mathcal{V}, \alpha \in f(v) \right\}, \tag{4.5}$$

which is a minimum cover for $(\mathcal{U}, \mathcal{F})$ [32]; i.e., $|\mathcal{C}_f| = |\mathcal{V}| = \mathsf{opt}(\mathcal{F})$. Constructing minimum covers $\mathcal{C}^{\mathsf{start}}$ from f^{start} and $\mathcal{C}^{\mathsf{goal}}$ from f^{goal} by Eq. (4.5), we obtain an instance $((\mathcal{U}, \mathcal{F}), \mathcal{C}^{\mathsf{start}}, \mathcal{C}^{\mathsf{goal}})$ of Minmax Set Cover Reconfiguration. This completes the description of the reduction.

[5] Similarly, a 2-factor approximation algorithm can be obtained for Minmax Dominating Set Reconfiguration and Minmax Hypergraph Vertex Cover Reconfiguration.
[6] In other words, each $f(v)$ is a singleton.

Here, we will show that

$$\mathsf{MinLab}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) = \mathsf{cost}_{\mathcal{F}}(\mathcal{C}^{\mathsf{start}} \leftrightsquigarrow \mathcal{C}^{\mathsf{goal}}), \tag{4.6}$$

which implies the completeness and soundness; for this, we use the following lemma [32].

▶ **Lemma 4.2** ([32, Observation 4.4; Claim 4.7]). *Let $f\colon \mathcal{V} \to 2^{\Sigma}$ be a multi-assignment and $\mathcal{C} \subseteq \mathcal{F}$ be a subfamily such that for any $v \in \mathcal{V}$ and $\alpha \in \Sigma$, $\alpha \in f(v)$ if and only if $S_{v,\alpha} \in \mathcal{C}$. Then, f satisfies an edge $e = (v, w) \in \mathcal{E}$ if and only if $\mathcal{C}$ covers $\{e\} \times B$. In particular, f satisfies G if and only if $\mathcal{C}$ covers $\mathcal{E} \times B$. Moreover, it holds that $\|f\| = |\mathcal{C}|$.*

We first show that $\mathsf{MinLab}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) \geqslant \mathsf{cost}_{\mathcal{F}}(\mathcal{C}^{\mathsf{start}} \leftrightsquigarrow \mathcal{C}^{\mathsf{goal}})$. For any reconfiguration multi-assignment sequence $F = (f^{(1)}, \dots, f^{(T)})$ from f^{start} to f^{goal} such that $\|F\|_{\max} = \mathsf{MinLab}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}})$, we can construct a reconfiguration sequence $\mathscr{C} = (\mathcal{C}_{f^{(1)}}, \dots, \mathcal{C}_{f^{(T)}})$ from $\mathcal{C}^{\mathsf{start}}$ to $\mathcal{C}^{\mathsf{goal}}$ by Eq. (4.5). By Lemma 4.2, each $\mathcal{C}_{f^{(t)}}$ covers $\mathcal{U}$; thus, $\mathscr{C}$ is a valid reconfiguration sequence from $\mathcal{C}^{\mathsf{start}}$ to $\mathcal{C}^{\mathsf{goal}}$. Moreover, $\mathsf{cost}_{\mathcal{F}}(\mathcal{C}^{\mathsf{start}} \leftrightsquigarrow \mathcal{C}^{\mathsf{goal}}) \leqslant \mathsf{cost}_{\mathcal{F}}(\mathscr{C}) = \|F\|_{\max} = \mathsf{MinLab}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}})$, as desired. We then show that $\mathsf{MinLab}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) \leqslant \mathsf{cost}_{\mathcal{F}}(\mathcal{C}^{\mathsf{start}} \leftrightsquigarrow \mathcal{C}^{\mathsf{goal}})$. For any reconfiguration sequence $\mathscr{C} = (\mathcal{C}^{(1)}, \dots, \mathcal{C}^{(T)})$ from $\mathcal{C}^{\mathsf{start}}$ to $\mathcal{C}^{\mathsf{goal}}$ such that $\mathsf{cost}_{\mathcal{F}}(\mathscr{C}) = \mathsf{cost}_{\mathcal{F}}(\mathcal{C}^{\mathsf{start}} \leftrightsquigarrow \mathcal{C}^{\mathsf{goal}})$, we can construct a sequence $F = (f^{(1)}, \dots, f^{(t)})$ of multi-assignments such that $f^{(t)}\colon \mathcal{V} \to 2^{\Sigma}$ is defined as follows:

$$f^{(t)}(v) \coloneqq \left\{ \alpha \in \Sigma \;\middle|\; S_{v,\alpha} \in \mathcal{C}^{(t)} \right\} \text{ for all } v \in \mathcal{V}. \tag{4.7}$$

By Lemma 4.2, each $f^{(t)}$ satisfies G; thus, F is a valid reconfiguration multi-assignment sequence from f^{start} to f^{goal}. Moreover, $\mathsf{MinLab}_G(f^{\mathsf{start}} \leftrightsquigarrow f^{\mathsf{goal}}) \leqslant \|F\|_{\max} = \mathsf{cost}_{\mathcal{F}}(\mathscr{C}) = \mathsf{cost}_{\mathcal{F}}(\mathcal{C}^{\mathsf{start}} \leftrightsquigarrow \mathcal{C}^{\mathsf{goal}})$, which completes the proof of Eq. (4.6).

Since $|\Sigma| = \operatorname{polyloglog} N$, the reduction takes polynomial time in N, and it holds that $|\mathcal{U}| = |\mathcal{E} \times B| = \mathcal{O}(N^2 \cdot 2^{\operatorname{polyloglog} N}) = \mathcal{O}(N^3)$; i.e., $N = \Omega(|\mathcal{U}|^{\frac{1}{3}})$. By Theorem 3.9, $\textsc{Gap}_{1,2-\frac{1}{\operatorname{polyloglog} N}}$ Label Cover Reconfiguration with alphabet size $\operatorname{polyloglog} N$ is PSPACE-complete; thus, $\textsc{Gap}_{1,2-\frac{1}{\operatorname{polyloglog}|\mathcal{U}|}}$ Set Cover Reconfiguration is PSPACE-complete as well, accomplishing the proof. ◀

4.2 PSPACE-hardness of Approximation for Hypergraph Vertex Cover Reconfiguration

We conclude this section with a similar inapproximability result for Minmax Hypergraph Vertex Cover Reconfiguration on $\mathcal{O}(1)$-uniform hypergraphs. Minmax Hypergraph Vertex Cover Reconfiguration is defined analogously to Minmax Set Cover Reconfiguration; refer to the full version [18] for the formal definition. Our inapproximability result is shown below, whose proof reuses the reduction of Theorem 4.1.

▶ **Theorem 4.3** (∗). *For any constant $\varepsilon \in (0, 1)$, $\textsc{Gap}_{1,2-\varepsilon}$ Hypergraph Vertex Cover Reconfiguration on $\operatorname{poly}(\varepsilon^{-1})$-uniform hypergraphs is* PSPACE*-complete. In particular,* Minmax Hypergraph Vertex Cover Reconfiguration *on $\operatorname{poly}(\varepsilon^{-1})$-uniform hypergraphs is* PSPACE*-hard to approximate within a factor of $2 - \varepsilon$.*

References

1 Noga Alon. Explicit expanders of every degree and size. *Comb.*, 41(4):447–463, 2021.

2 Noga Alon, Uriel Feige, Avi Wigderson, and David Zuckerman. Derandomized graph products. *Comput. Complex.*, 5:60–75, 1995.

3 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

4 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.

5 Marthe Bonamy, Marc Heinrich, Takehiro Ito, Yusuke Kobayashi, Haruka Mizuta, Moritz Mühlenthaler, Akira Suzuki, and Kunihiro Wasa. Shortest reconfiguration of colorings under Kempe changes. In *STACS*, pages 35:1–35:14, 2020.

6 Paul Bonsma. The complexity of rerouting shortest paths. *Theor. Comput. Sci.*, 510:1–12, 2013.

7 Luis Cereceda, Jan van den Heuvel, and Matthew Johnson. Finding paths between 3-colorings. *J. Graph Theory*, 67(1):69–82, 2011.

8 Moses Charikar, MohammadTaghi Hajiaghayi, and Howard J. Karloff. Improved approximation algorithms for label cover problems. *Algorithmica*, 61(1):190–206, 2011.

9 Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):12, 2007.

10 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *STOC*, pages 624–633, 2014.

11 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

12 Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *J. ACM*, 43(2):268–292, 1996.

13 Parikshit Gopalan, Phokion G. Kolaitis, Elitza Maneva, and Christos H. Papadimitriou. The connectivity of Boolean satisfiability: Computational and structural dichotomies. *SIAM J. Comput.*, 38(6):2330–2355, 2009.

14 Johan Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Math.*, 182:105–142, 1999.

15 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

16 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1-2):72–96, 2005.

17 Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation*. A K Peters, Ltd., 2009.

18 Shuichi Hirahara and Naoto Ohsaka. Optimal PSPACE-hardness of approximating set cover reconfiguration. *Electron. Colloquium Comput. Complex.*, pages TR24–039, 2024.

19 Shuichi Hirahara and Naoto Ohsaka. Probabilistically checkable reconfiguration proofs and inapproximability of reconfiguration problems. In *STOC*, 2024. to appear.

20 Duc A. Hoang. Combinatorial reconfiguration. `https://reconf.wikidot.com/`, 2023.

21 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bull. Am. Math. Soc.*, 43(4):439–561, 2006.

22 Takehiro Ito and Erik D. Demaine. Approximability of the subset sum reconfiguration problem. *J. Comb. Optim.*, 28(3):639–654, 2014.

23 Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theor. Comput. Sci.*, 412(12-14):1054–1065, 2011.

24 Takehiro Ito, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, and Yoshio Okamoto. Shortest reconfiguration of perfect matchings via alternating cycles. *SIAM J. Discret. Math.*, 36(2):1102–1123, 2022.

25 Marcin Kamiński, Paul Medvedev, and Martin Milanič. Shortest paths between shortest paths. *Theor. Comput. Sci.*, 412(39):5205–5210, 2011.

26 Karthik C. S. and Pasin Manurangsi. On inapproximability of reconfiguration problems: PSPACE-hardness and some tight NP-hardness results. *CoRR*, abs/2312.17140, 2023.

27 Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994.

28 Sidhanth Mohanty, Ryan O'Donnell, and Pedro Paredes. Explicit near-Ramanujan graphs of every degree. *SIAM J. Comput.*, 51(3):STOC20–1–STOC20–23, 2021.

29 Amer E. Mouawad, Naomi Nishimura, Vinayak Pathak, and Venkatesh Raman. Shortest reconfiguration paths in the solution space of Boolean formulas. *SIAM J. Discret. Math.*, 31(3):2185–2200, 2017.

30 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018.

31 Naoto Ohsaka. Gap preserving reductions between reconfiguration problems. In *STACS*, pages 49:1–49:18, 2023.

32 Naoto Ohsaka. Gap amplification for reconfiguration problems. In *SODA*, pages 1345–1366, 2024.

33 Naoto Ohsaka and Tatsuya Matsuoka. Reconfiguration problems on submodular functions. In *WSDM*, pages 764–774, 2022.

34 Orr Paradise. Smooth and strong PCPs. *Comput. Complex.*, 30(1):1, 2021.

35 Jan van den Heuvel. The complexity of change. In *Surveys in Combinatorics 2013*, volume 409, pages 127–160. Cambridge University Press, 2013.

Problems on Group-Labeled Matroid Bases

Florian Hörsch
Algorithms and Complexity Group, CISPA, Saarbrücken, Germany

András Imolay
MTA-ELTE Matroid Optimization Research Group, Department of Operations Research, ELTE Eötvös Loránd University, Budapest, Hungary

Ryuhei Mizutani
Department of Mathematical Informatics, Graduate School of Information Science and Technology, The University of Tokyo, Japan

Taihei Oki
Department of Mathematical Informatics, Graduate School of Information Science and Technology, The University of Tokyo, Japan

Tamás Schwarcz
MTA-ELTE Matroid Optimization Research Group, Department of Operations Research, ELTE Eötvös Loránd University, Budapest, Hungary

Abstract

Consider a matroid equipped with a labeling of its ground set to an abelian group. We define the label of a subset of the ground set as the sum of the labels of its elements. We study a collection of problems on finding bases and common bases of matroids with restrictions on their labels. For zero bases and zero common bases, the results are mostly negative. While finding a non-zero basis of a matroid is not difficult, it turns out that the complexity of finding a non-zero common basis depends on the group. Namely, we show that the problem is hard for a fixed group if it contains an element of order two, otherwise it is polynomially solvable.

As a generalization of both zero and non-zero constraints, we further study F-avoiding constraints where we seek a basis or common basis whose label is not in a given set F of forbidden labels. Using algebraic techniques, we give a randomized algorithm for finding an F-avoiding common basis of two matroids represented over the same field for finite groups given as operation tables. The study of F-avoiding bases with groups given as oracles leads to a conjecture stating that whenever an F-avoiding basis exists, an F-avoiding basis can be obtained from an arbitrary basis by exchanging at most $|F|$ elements. We prove the conjecture for the special cases when $|F| \leq 2$ or the group is ordered. By relying on structural observations on matroids representable over fixed, finite fields, we verify a relaxed version of the conjecture for these matroids. As a consequence, we obtain a polynomial-time algorithm in these special cases for finding an F-avoiding basis when $|F|$ is fixed.

2012 ACM Subject Classification Mathematics of computing → Matroids and greedoids

Keywords and phrases matroids, matroid intersection, congruency constraint, exact-weight constraint, additive combinatorics, algebraic algorithm, strongly base orderability

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.86

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2402.16259

Funding This research has been implemented with the support provided by the Lendület Programme of the Hungarian Academy of Sciences – grant number LP2021-1/2021.
András Imolay: Supported by the Rényi Doctoral Fellowship of the Rényi Institute.
Ryuhei Mizutani: Supported by Grant-in-Aid for JSPS Fellows Grant Number JP23KJ0379 and JST SPRING Grant Number JPMJSP2108.
Taihei Oki: Supported by JSPS KAKENHI Grant Number JP22K17853 and JST ERATO Grant Number JPMJER1903.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 86; pp. 86:1–86:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Tamás Schwarcz: Supported by the ÚNKP-23-3 New National Excellence Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund.

Acknowledgements The authors are grateful to the organizers of the 14th Emléktábla Workshop, where the collaboration of the authors started. The authors thank Naonori Kakimura, Kevin Long, and Tomohiko Yokoyama for several discussions during the workshop, and Kristóf Bérczi, András Frank, and András Sebő for pointing out the connections to lattices and dual lattices.

1 Introduction

Several combinatorial optimization problems involve additional constraints, such as parity, congruency, and exact-weight constraints [35, 41, 42, 43, 44]. These constraints are subsumed by *group-label constraints* defined as follows: the ground set E is equipped with a labeling $\psi\colon E \to \Gamma$ to an abelian group Γ and a solution $X \subseteq E$ must ensure that the sum of the labels of its entries is not in a prescribed forbidden set $F \subseteq \Gamma$, i.e., $\psi(X) \coloneqq \sum_{e\in X} \psi(e) \notin F$. We call such a solution *F-avoiding*.

Particularly important special cases of group-label constraints include the *non-zero* ($F = \{0\}$) and *zero* ($F = \Gamma \setminus \{0\}$) constraints, where F-avoiding sets are referred to as *non-zero* and *zero*, respectively. The non-zero constraint has been extensively studied for path problems on graphs as it generalizes constraints on parity and topology. This line of research includes packing non-zero A-paths [12] as well as finding a shortest odd s–t path [46, Section 29.11e], a shortest non-zero s–t path [25], and an F-avoiding s–t path with $|F| \leq 2$ [28]. For these problems, some literature allows Γ to be non-abelian since the order of operations can be naturally defined for paths. Problems related to non-zero perfect bipartite matchings in $\mathbb{Z}_2$ have also been dealt with, see [1, 18, 26]. The zero constraint, or, slightly more generally, the group-label constraint with $|\Gamma \setminus F| = 1$, can encode the congruency and exact-weight constraints by setting Γ to be a cyclic group $\mathbb{Z}_m$ and the integers $\mathbb{Z}$, respectively. Examples of problems whose congruency-constrained versions have been studied include submodular function minimization [42], minimum cut [43], and integer linear programming with totally unimodular coefficients [41]. For the last problem, Nägele, Santiago, and Zenklusen [41] gave a randomized strongly polynomial-time algorithm to test the existence of an F-avoiding feasible solution, where the group is $\mathbb{Z}_m$ with prime m and $|F| \leq 2$, implying the congruency constraint if $m = 3$.

The exact-weight constraint was first considered for the matching problem by Papadimitriou and Yannakakis [44]. Mulmuley, Vazirani, and Vazirani [40] gave a randomized polynomial-time algorithm for solving the problem using an algebraic technique. Derandomizing this algorithm is a major open problem and there is a collection of partial results for it, see e.g. [6, 18, 22, 49, 52]. Other exact problems include arborescences, matchings, cycles [2], and independent sets or bases in a matroid [11, 16, 45].

In this work, we explore group-label constraints for matroid bases and matroid intersection. Throughout the paper, we assume that any group Γ is abelian without mentioning it. In the problems Non-Zero Basis and Zero Basis, we are given a matroid M on a ground set E and a group labeling $\psi\colon E \to \Gamma$, and we are to find a non-zero or zero basis, respectively, or to correctly report that no such basis exists. In F-avoiding Basis, along with the matroid M and labeling ψ, we are also given a forbidden label set $F \subseteq \Gamma$, and we need to find an F-avoiding basis, that is, a basis B with $\psi(B) \notin F$. In Non-Zero Common Basis, Zero Common Basis, and F-avoiding Common Basis, instead of a single matroid, we are given two matroids M_1 and M_2 on a common ground set E and seek a non-zero, zero,

and F-avoiding common basis, respectively. We also tackle the weighted variants of these problems, referred to as Weighted Non-Zero Basis for example, where we are to find a feasible solution minimizing a given weight function $w\colon E \to \mathbb{R}$.

We note that the target label 0 in the non-zero and zero problems can be changed to an arbitrary group element $g \in \Gamma$ by appending a coloop to the ground set with label $-g$. Regarding the input of the group, we consider the following three types: (i) operation and zero-test oracle, (ii) operation table of a finite group, and (iii) a fixed finite group. Unless otherwise stated, we assume that a group is given as the oracles and the matroids are given as independence oracles. In this case, by a polynomial-time algorithm, we mean an algorithm making polynomially many elementary steps, group operations, and independence oracle calls. If the group is finite and is given by its operation table, then the running time of a polynomial-time algorithm can depend polynomially on the group size.

Our research follows the recent initiative by Liu and Xu [35], who addressed Zero Basis[1]. They conjectured that, given the existence of a zero basis, for any non-zero basis B, there is a zero basis B^* such that $|B^* \setminus B| \leq D(\Gamma) - 1$, where $D(\Gamma)$ denotes the Davenport constant of Γ (see Section 6.3 for definition), which is upper-bounded by $|\Gamma|$. Liu and Xu proved the conjecture for cyclic groups $\Gamma = \mathbb{Z}_m$ with the order m being a prime power or the product of two primes, with the aid of an additive combinatorics result by Schrijver and Seymour [47], deriving an FPT algorithm parameterized by $|\Gamma| = m$ for Zero Basis. In Theorem 6.6, we give a counterexample to this conjecture for groups with $\mathbb{Z}_2^d$ with $d \geq 4$.

The non-zero constraint is closely related to *lattices* studied by Lovász [37]. The *lattice* generated by vectors $\{v_1, \ldots, v_n\} \subseteq \mathbb{R}^n$ is the set $\{\sum_{i=1}^n \lambda_i v_i \mid \lambda_1, \ldots, \lambda_n \in \mathbb{Z}\}$. For a set family $\mathcal{F} \subseteq 2^E$, let $\mathrm{lat}(\mathcal{F})$ denote the lattice generated by the characteristic vectors of $\mathcal{F}$. Every lattice has a *lattice basis* $A = \{a_1, \ldots, a_n\} \subseteq \mathbb{Z}^E$, which is a set of linearly independent vectors generating it. Since $\mathcal{F}$ and its lattice basis A generate the same lattice, $\mathcal{F}$ has a non-zero member if and only if A has a non-zero member, i.e., $\psi(a_i) := \sum_{e \in E} a_i(e)\psi(e) \neq 0$ for some i. This implies that if a basis of $\mathrm{lat}(\mathcal{F})$ can be computed in polynomial time, then the existence of a non-zero member of $\mathcal{F}$ can be decided in polynomial time. Such set families $\mathcal{F}$ include matroid bases [45], common bases of a matroid and a partition matroid having two classes [45], and perfect matchings [37, 38].

Below, we summarize our results for each problem.

Non-Zero Basis. The tractability of Non-Zero Basis can be derived from the above lattice argument together with a characterization of base lattices [45]. We observe that for any zero basis B, there exists a non-zero basis B^* such that $|B^* \setminus B| \leq 1$, provided that at least one non-zero basis exists. A weighted variant of this statement is shown in the same way. This result generalizes an algorithm for Weighted Zero Basis with $\Gamma = \mathbb{Z}_2$ by Liu and Xu [35].

Non-Zero Common Basis. We show in Theorems 3.7 and 6.1 that Non-Zero Common Basis is polynomially solvable if and only if Γ does not contain $\mathbb{Z}_2$ as a subgroup. Our hardness proof for $\Gamma = \mathbb{Z}_2$ is based on an information-theoretic argument using sparse paving matroids, which is independent of the $\mathsf{P} \neq \mathsf{NP}$ conjecture. The polynomial-time algorithm for $\Gamma \not\geq \mathbb{Z}_2$ is a modification of the negatively directed cycle elimination algorithm for weighted matroid intersection [9]. In Theorem 3.9, we also give a 2-approximation algorithm for Weighted Non-Zero Common Basis if $\Gamma \not\geq \mathbb{Z}_2$ and the weight function is nonnegative. Finally, in Theorems 3.11 and 3.12, we solve Weighted

[1] Called *Group-Constrained Matroid Base* (GCMB) in [35].

Non-Zero Common Basis for an arbitrary group when both matroids are partition matroids or one of the matroids is a partition matroid defined by a partition having two classes.

F-avoiding Basis and Common Basis. If the group is fixed and finite, (Weighted) F-avoiding Basis reduces to polynomially many instances of (weighted) matroid intersection [35]. On the other hand, it follows from the results of [16] that F-avoiding Basis requires exponentially many independence oracle queries if F is part of the input and the group is finite and is given as an operation table, while the same hardness of F-avoiding Common Basis follows from our Theorem 6.1 even if the group is fixed and finite. Regarding positive results for F-avoiding problems, our contribution is twofold. First, using similar ideas as in [11, 51], in Theorem 4.2 we give a randomized algebraic algorithm for F-avoiding Common Basis in case where the matroids are represented over the same field and the group is finite and is given as an operation table. We observe in Theorem 4.3 that the algorithm can be derandomized in certain cases, including F-avoiding Basis for graphic matroids. Second, we turn to the study of F-avoiding Basis for cases where $|F|$ is fixed and the group is given by an oracle. Motivated by the work of Liu and Xu [35], we propose Conjecture 5.1 stating that if at least one F-avoiding basis exists, then each basis can be turned into an F-avoiding basis by exchanging at most $|F|$ elements. The validity of the conjecture follows from the results of [35] for groups of prime order. We show that the conjecture also holds if Γ is an ordered group (Theorem 5.6) or if $|F| \leq 2$ (Theorem 5.18). By introducing a novel relaxation of strong base orderability, in Theorem 5.11 we show that a relaxation of the conjecture holds for $\mathrm{GF}(q)$-representable matroids for every fixed prime power q. In Theorem 5.17, we prove a somewhat stronger version of this result for graphic matroids. In each of these special cases, we obtain the polynomial solvability of F-avoiding Basis for fixed F.

Zero Basis and Zero Common Basis. The zero constraint for $\Gamma = \mathbb{Z}$ corresponds to the exact-weight constraint, implying that many problems are NP-hard, in particular, Zero Basis is NP-hard even for uniform matroids (Theorem 6.2). It follows from the results of [16] that Zero Basis requires exponentially many independence oracle queries for a finite group given by operation table. We show the same hardness of Zero Common Basis for any fixed nontrivial group (Theorem 6.4). On the other hand, we obtain positive results from the aforementioned results on F-avoiding problems. In particular, Zero Basis is polynomially solvable if the group is fixed and finite [35], Theorem 4.2 implies a randomized polynomial-time algorithm for Zero Common Basis for matroids represented over the same field if Γ is finite and is given as an operation table, and using the results of [35], Theorem 5.11 implies an FPT algorithm for Zero Basis when parameterized by $|\Gamma|$ if the matroids are representable over a fixed, finite field.

Other work related to group-labeled matroids

Bérczi and Schwarcz [5] showed the hardness of partitioning into common bases, see also [4, 24] for later results. A natural relaxation of that problem gives rise to problems related to Non-Zero Common Basis for the group $\mathbb{R}/\mathbb{Z}$. This relation is explained in the full version.

It is straightforward to verify that the family of non-zero subsets of a set satisfies the axiom of *delta-matroids*, which are a generalization of matroids introduced by Bouchet [8]. From this viewpoint, Non-Zero Basis offers a tractable special case of the intersection of a matroid and a delta-matroid. We note that the intersection of a matroid and a delta-matroid is intractable in general, as it encompasses matroid parity. Kim, Lee, and Oum [29] defined a delta-matroid, called a Γ-*graphic delta-matroid*, from a graph equipped with a labeling of

vertices to an abelian group Γ. In the definition, they employ a constraint similar to but different from non-zero. Exploring the relationship between Γ-graphic delta-matroids and our findings is left for future work.

Organization

The rest of this paper is organized as follows. Section 2 provides preliminaries on groups and matroids. Section 3 deals with non-zero problems. Section 4 provides an algebraic algorithm for F-avoiding Common Basis for represented matroids. Section 5 studies F-avoiding Basis if $|F|$ is fixed. Section 6 includes our hardness results for each of the problems. Finally, Section 7 concludes this paper enumerating open problems.

2 Preliminaries

Let $\mathbb{N}$, $\mathbb{Z}_{\geq 0}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}_{\geq 0}$, and $\mathbb{R}$ denote the set of positive integers, nonnegative integers, integers, rationals, nonnegative reals, and reals, respectively. We let $[n] \coloneqq \{1, \dots, n\}$ for $n \in \mathbb{Z}_{\geq 0}$. For a set S, we simply write $S \setminus \{x\}$ as $S - x$ for $x \in S$ and $S \cup \{y\}$ as $S + y$ for $y \notin S$. For a set E and $r \in \mathbb{Z}_{\geq 0}$, we let $\binom{E}{r} \coloneqq \{ S \subseteq E \mid |S| = r \}$.

In this paper, all groups are implicitly assumed to be abelian. We use the additive notation for the operations of groups except in Section 4. For $m \in \mathbb{N}$, let $\mathbb{Z}_m = \{0, \dots, m-1\}$ denote the cyclic group of order m. For groups Γ_1 and Γ_2, we mean by $\Gamma_1 \leq \Gamma_2$ that Γ_1 is a subgroup of Γ_2. A group Γ is said to be *ordered* if Γ is equipped with a total order $\leq$ compatible with the operation of Γ in the sense that $a \leq b$ implies $a + c \leq b + c$ for all $a, b, c \in \Gamma$. A *labeling* is a function $\psi \colon E \to \Gamma$ from a set E to a group Γ, and we let $\psi(S) \coloneqq \sum_{x \in S} \psi(x)$ for $S \subseteq E$. Let $\mathrm{GF}(q)$ denote the finite field of q elements for a prime power q.

We follow [14] for basic terminologies on graphs such as paths and cycles. The vertex and edge sets of a graph G are denoted by $V(G)$ and $E(G)$, respectively. Similarly, $V(D)$ denotes the vertex set of a directed graph D, and $A(D)$ denotes its arc set. Given a weight function $w \colon A(D) \to \mathbb{R}$ and a subgraph C of D, the *weight* of C is defined as $w(C) \coloneqq w(A(C))$. A weight function w is said to be *conservative* if D does not contain a directed cycle of negative weight.

We refer the reader to [20, 46] for basics on matroid optimization. A *matroid* M consists of a finite ground set $E(M)$ and a nonempty set family $\mathcal{B}(M) \subseteq 2^{E(M)}$ such that for any $B_1, B_2 \in \mathcal{B}(M)$ and $x \in B_1 \setminus B_2$, there exists $y \in B_2 \setminus B_1$ such that $B_1 - x + y \in \mathcal{B}(M)$. Elements in $\mathcal{B}(M)$ are called *bases*. The next basis exchange property was proved by Brualdi [10], see also [46, Theorem 39.12].

▶ **Lemma 2.1** (Brualdi [10]). *If B and B' are bases of a matroid M, then there exists a bijection $\phi \colon B \setminus B' \to B' \setminus B$ such that $B - e + \phi(e)$ is a basis for each $e \in B \setminus B'$.*

Following [20], we define a *partition matroid* as a direct sum of uniform matroids and a *unitary partition matroid* as a direct sum of rank-1 uniform matroids. We note that several authors refer to the latter object as partition matroids. Given a matrix A over some field, we denote by $M(A)$ the matroid defined on the column indices of A where a set is a basis of $M(A)$ if the corresponding columns form a basis of the vector space spanned by the columns of A. Given a connected graph G, its *cycle matroid* $M(G)$ is the matroid whose ground set is $E(G)$ and whose bases are the edge sets of the spanning trees of G. If $M = M(A)$ for a matrix A over a field $\mathbb{F}$ or a graph A, we say that M is $\mathbb{F}$-*representable* or *graphic*, respectively.

3 Non-Zero Basis and Common Basis

3.1 Non-Zero Basis

In this section, we consider (Weighted) Non-Zero Basis. The following theorem can be derived from the description of the lattices of matroid bases by Rieder [45]. In what follows we give a direct proof of the result.

▶ **Theorem 3.1.** *Let M be a matroid and $\psi\colon E(M) \to \Gamma$ a group labeling. The following are equivalent:*

(i) *all bases of M have the same label,*
(ii) *M has a basis B such that $\psi(B') = \psi(B)$ holds for each basis B' with $|B \setminus B'| \leq 1$, and*
(iii) *ψ is constant on each component of M.*

Proof. It is clear that (i) implies (ii) and (iii) implies (i). In what follows, we show that (ii) implies (iii). Let B be a basis such that $\psi(B') = \psi(B)$ holds for each basis B' with $|B \setminus B'| \leq 1$. Let G_B denote the bipartite graph with bipartition $(B, E(M) \setminus B)$ and edge set $\{\, xy \mid x \in B, y \in E(M) \setminus B, B - x + y \in \mathcal{B}(M) \,\}$. By the assumption on B, it follows that $\psi(x) = \psi(y)$ for each edge xy of G_B. Then, ψ is constant on each connected component of the graph G_B, and thus (iii) follows by using that the connected components of the graph G_B coincide with the components of the matroid M [32]. ◀

Note that Theorem 3.1(iii) provides a characterization for "NO" instances of Non-Zero Basis, while Theorem 3.1(ii) provides a simple algorithm for this problem. Liu and Xu [35] gave the following simple and constructive algorithm for Weighted Zero Basis with $\Gamma = \mathbb{Z}_2$, for which the zero and non-zero constraints are equivalent, without decomposing the matroid into components. First, find a minimum weight basis $B \in \mathcal{B}(M)$, and if $\psi(B) \neq 0$, then we are done. Otherwise, consider all the bases of the form $B - x + y$ with $x \in B$ and $y \in E(M) \setminus B$. Among these bases, if there is none with a non-zero label, then there does not exist a non-zero basis, otherwise, choose a non-zero basis of minimum weight among the considered ones. Actually, this algorithm works correctly for Weighted Non-Zero Basis for any group, and the proof of [35, Proposition 1] can be modified to show its correctness. In what follows, we give a different proof of this fact.

▶ **Lemma 3.2.** *Let M be a matroid, $\psi\colon E(M) \to \Gamma$ a group labeling, and $w\colon E(M) \to \mathbb{R}$ a weight function. Then, for any minimum-weight basis B, there exists a minimum-weight non-zero basis B^* such that $|B \setminus B^*| \leq 1$, provided that at least one non-zero basis exists.*

Proof. Let B' be a minimum-weight non-zero basis with $|B \setminus B'|$ being minimal. If $B = B'$ then we are done, otherwise $\psi(B) = 0$. According to the symmetric exchange axiom, we can choose $x \in B \setminus B'$ and $y \in B' \setminus B$ such that $B - x + y$ and $B' + x - y$ are both bases. As $0 \neq \psi(B) + \psi(B') = \psi(B - x + y) + \psi(B' + x - y)$, one of $B - x + y$ and $B' + x - y$ must be non-zero. Suppose $\psi(B - x + y) \neq 0$. Since $w(B) + w(B') = w(B - x + y) + w(B' + x - y)$ and $w(B)$ has the minimum weight, we have $w(B - x + y) \leq w(B')$, which implies $w(B - x + y) = w(B')$ by $\psi(B - x + y) \neq 0$. Thus, we can take $B^* = B - x + y$. If $\psi(B' + x - y) \neq 0$, then it can be shown in the same way that $B' + x - y$ is a minimum-weight non-zero basis, contradicting the assumption that B' is a minimum-weight non-zero basis closest to B. ◀

We obtain the following from Lemma 3.2.

▶ **Theorem 3.3.** Weighted Non-Zero Basis *can be solved in polynomial time.*

3.2 Non-Zero Common Basis

3.2.1 Polynomial-time Algorithm with $\mathbb{Z}_2 \not\leq \Gamma$

In this section, we show the polynomial solvability of Non-Zero Common Basis when $\mathbb{Z}_2 \not\leq \Gamma$, that is, Γ does not contain any element of order two. Later, we will show in Theorem 6.1 that the problem is hard if $\mathbb{Z}_2 \leq \Gamma$. Our algorithm is a modification of the weighted matroid intersection algorithm given by Krogdahl [30, 31, 32] and independently by Brezovec, Cornuéjols, and Glover [9].

We will use the following result on directed graphs. While several works concentrated on finding non-zero paths and cycles in group-labeled graphs [25], their setting does not seem to include group-labeled digraphs. Therefore, we give a proof of the next result in the full version. While this result may be of independent interest, it will later be applied as a subroutine.

▶ **Theorem 3.4.** *Let D be a digraph, $\psi\colon A(D) \to \Gamma$ a group labeling, and $w\colon A(D) \to \mathbb{R}$ a conservative weight function. Then, there is a polynomial-time algorithm that returns a non-zero directed cycle in D which is shortest with respect to w or correctly reports that D contains no non-zero directed cycle.*

We note that the problem of finding an odd directed path is NP-hard even in the unweighted case [33]. In contrast to Theorem 3.4, the key distinction here lies between walks and paths: while a walk can include a directed cycle to change a group label, a path cannot. Consequently, a Dijkstra-style algorithm for finding an odd directed path must track not only the last vertex but also all intermediate vertices, leading to an exponential increase in running time.

Let M_1 and M_2 be matroids on a common ground set E and $\psi\colon E \to \Gamma$ a group labeling. Given a common basis B, we define the digraph $D_{M_1,M_2}(B)$ on the vertex set E and the labeling ψ' on its arcs as follows. For each $x \in B$ and $y \in E \setminus B$ such that $B - x + y \in \mathcal{B}(M_1)$, we add an arc xy to $D_{M_1,M_2}(B)$ with label $\psi'(xy) \coloneqq \psi(y)$. Similarly, for each $x \in B$ and $y \in E \setminus B$ such that $B - x + y \in \mathcal{B}(M_2)$, we add an arc yx and with label $\psi'(yx) \coloneqq -\psi(x)$.

▶ **Lemma 3.5.** *Let M_1 and M_2 be matroids on a common ground set E and $\psi\colon E \to \Gamma$ a group labeling. Let B and B' be common bases of M_1 and M_2 such that $\psi(B) = 0$ and $\psi(B') \neq 0$. Then, $D_{M_1,M_2}(B)$ contains a non-zero directed cycle C with $V(C) \subseteq B \triangle B'$.*

Proof. By Lemma 2.1, $D_{M_1,M_2}(B)$ contains a collection P_1 of $|B \setminus B'|$ pairwise vertex-disjoint arcs from $B \setminus B'$ to $B' \setminus B$ and a collection P_2 of $|B \setminus B'|$ pairwise vertex-disjoint arcs from $B' \setminus B$ to $B \setminus B'$. The union of P_1 and P_2 has label $\psi(B' \setminus B) - \psi(B \setminus B') = \psi(B') - \psi(B) = \psi(B') \neq 0$ and consists of pairwise vertex-disjoint directed cycles in $D_{M_1,M_2}(B)$, hence it contains a non-zero directed cycle. ◀

The following result and proof are analogous to that of [9, Theorem 2]. In that result, a weight function is given instead of a labeling, and the constraint "non-zero" is replaced by "negative". In our setting, the proof only works if we assume $\mathbb{Z}_2 \not\leq \Gamma$, as we need to guarantee that if we decompose an arc set having label $2\psi'(C)$ for some non-zero directed cycle C, then at least one member of the decomposition has non-zero label. We give the proof in the full version.

▶ **Lemma 3.6.** *Let Γ be a group such that $\mathbb{Z}_2 \not\leq \Gamma$. Let M_1 and M_2 be matroids on a common ground set E, $\psi\colon E \to \Gamma$ a group labeling, and B a common basis. If C is a non-zero directed cycle of $D_{M_1,M_2}(B)$ whose vertex set is inclusion-wise minimal among non-zero directed cycles, then $B \triangle V(C)$ is a common basis.*

Combining Lemmas 3.5 and 3.6, we obtain the main result of the section.

▶ **Theorem 3.7.** *Let Γ be a group such that $\mathbb{Z}_2 \not\leq \Gamma$. Let M_1 and M_2 be matroids on a common ground set E, $\psi\colon E \to \Gamma$ a group labeling, and B_0 a zero common basis. Then, M_1 and M_2 have a non-zero common basis if and only if $D_{M_1,M_2}(B_0)$ contains a non-zero directed cycle. Moreover,* Non-Zero Common Basis *is polynomially solvable.*

Proof. If there exists a non-zero common basis B^*, then $D_{M_1,M_2}(B_0)$ contains a non-zero directed cycle by Lemma 3.5. Conversely, if $D_{M_1,M_2}(B_0)$ contains a non-zero directed cycle, then let C be a minimum length non-zero directed cycle. Then, Lemma 3.6 implies that $B^* \coloneqq B \triangle V(C)$ is a common basis, and we have $\psi(B^*) = \psi(B_0) + \psi'(C) = \psi'(C) \neq 0$.

This provides the following algorithm for Non-Zero Common Basis. First, we find a common basis B_0. If no common basis exists or B_0 is non-zero, we are done. Otherwise, we find a minimum length non-zero directed cycle C in $D_{M_1,M_2}(B_0)$ using Theorem 3.4. If no non-zero directed cycle exists then we report that there is no non-zero common basis, otherwise we output $B_0 \triangle V(C)$. ◀

We turn to the study of Weighted Non-Zero Common Basis. Given two matroids M_1 and M_2 on a common ground set E, a common basis B, and a weight function $w\colon E \to \mathbb{R}$, we define the weight function w' on the arcs of $D_{M_1,M_2}(B)$ as follows. For each arc xy such that $x \in B$, $y \in E \setminus B$ and $B - x + y \in \mathcal{B}(M_1)$ we define $w'(xy) \coloneqq w(y)$, and for each arc yx such that $x \in B$, $y \in E \setminus B$ and $B - x + y \in \mathcal{B}(M_2)$ we define $w'(yx) \coloneqq -w(x)$. Then, B is a minimum-weight common basis if and only if w' is conservative [31, 21, 9], see also [46, Theorem 41.5]. We observe the following relationship between the weight of a shortest non-zero directed cycle in $D_{M_1,M_2}(B)$ and the weights of non-zero common bases of M_1 and M_2. Its simple proof can be found in the full version.

▶ **Lemma 3.8.** *Let M_1 and M_2 be matroids on a common ground set E, $\psi\colon E \to \Gamma$ a group labeling, and $w\colon E \to \mathbb{R}$ a weight function. Let B_0 be a minimum-weight common basis and assume that $\psi(B_0) = 0$. Let C be a shortest non-zero directed cycle in $D_{M_1,M_2}(B_0)$ with respect to w'. Then, $w(B_0 \triangle V(C)) \leq w(B^*)$ holds for each non-zero common basis B^*.*

We note that Lemma 3.8 generalizes Lemma 3.2, as in the special case $M_1 = M_2$ each arc of $D_{M_1,M_2}(B_0)$ is bidirectional, thus a shortest non-zero directed cycle consists of two vertices.

In Lemma 3.8, the weight of C is measured by w' (which takes negative values on some arcs), so $V(C)$ is not necessarily inclusion-wise minimal among the vertex sets of non-zero directed cycles. Thus, it does not yield an algorithm for Weighted Non-Zero Common Basis. In fact, the complexity of the problem remains open for a group Γ with $\mathbb{Z}_2 \not\leq \Gamma$. Nevertheless, we use Lemma 3.8 to give a 2-approximation if the weight function w is nonnegative. The proof of the result is given in the full version.

▶ **Theorem 3.9.** *Let Γ be a group with $\mathbb{Z}_2 \not\leq \Gamma$. Let M_1 and M_2 be matroids on a common ground set E, $\psi\colon E \to \Gamma$ a group labeling, and $w\colon E \to \mathbb{R}_{\geq 0}$ a weight function. Then, there exists a polynomial-time algorithm that computes a non-zero common basis B that satisfies $w(B) \leq 2w(B^*)$ for every non-zero common basis B^* or correctly outputs that there exists no non-zero common basis.*

3.2.2 Certificate for All Common Bases Being Zero

Given a strongly connected digraph having labels on its arcs, the fact that all directed cycles have label zero can be certified by a certain labeling of its vertices. Using this result and assuming a property of the matroid pair ensuring strong connectivity, we get a characterization for all directed cycles of $D_{M_1,M_2}(B)$ having label zero, which is analogous

to the weight-splitting theorem of Frank [19]. By Theorem 3.7, this provides the following certificate for the "NO" instances of Non-Zero Common Basis if $\mathbb{Z}_2 \not\leq \Gamma$. We give details in the full paper.

▶ **Theorem 3.10.** *Let M_1 and M_2 be matroids on a common ground set E and the same rank r. Assume that $r_{M_1}(X) + r_{M_2}(E \setminus X) > r$ holds for every $\emptyset \neq X \subsetneq E$. Let B be a common basis of M_1 and M_2, and $\psi\colon E \to \Gamma$ a group labeling. Then, $D_{M_1,M_2}(B)$ contains no non-zero directed cycle if and only if there exist labelings $\psi_1, \psi_2\colon E \to \Gamma$ such that $\psi = \psi_1 + \psi_2$ and ψ_i is constant on each connected component of M_i for $i = 1, 2$.*

3.2.3 Partition matroids

When both matroids are partition matroids, we can drop the assumption $\mathbb{Z}_2 \not\leq \Gamma$ from Theorem 3.7 and extend it to the weighted setting. The proof of the next result is given in the full version.

▶ **Theorem 3.11.** Weighted Non-Zero Common Basis *is polynomially solvable if M_1 and M_2 are partition matroids.*

Given a graph G and a function $b\colon V(G) \to \mathbb{Z}_{\geq 0}$, a *perfect b-matching* is an edge set $F \subseteq E(G)$ such that $d_F(v) = b(v)$ for each $v \in V$. If G is bipartite, then its perfect b-matchings form the family of common bases of two partition matroids. Therefore, Theorem 3.11 can be formulated as having a polynomial-time algorithm for finding a minimum weight non-zero perfect b-matching in a bipartite graph with weights and labels on its edges. For perfect matchings and the group $\mathbb{Z}_2$, the idea of essentially the same algorithm as ours was briefly mentioned in [26], where the authors noted that it can also be derived from results of [1]. A formal description of the algorithm and a proof of its correctness were given in [18] for a special weight function.

Next, we consider the special case of Weighted Non-Zero Common Basis when we only assume that one of the matroids is a partition matroid. Without further assumptions, this problem is not easier than the general one: a construction similar to that of Harvey, Király and Lau [23] shows that the general problem can be reduced to the special case when one of the matroids is a unitary partition matroid and all partition classes have size two. In what follows, we will consider the case when the partition matroid is defined by a partition having two classes. In this special case, the polynomial solvability of Non-Zero Common Basis follows from the corresponding lattice basis characterization of Rieder [45]. We extend this result by solving the weighted version of the problem.

▶ **Theorem 3.12.** Weighted Non-Zero Common Basis *is polynomially solvable if M_2 is a partition matroid defined by a partition having two classes.*

The proof of the theorem is given in the full version. The proof of both Theorem 3.11 and Theorem 3.12 relies on Lemma 3.8 by observing that the special property of the matroid pair guarantees that $B_0 \triangle V(C)$ is a common basis whenever B_0 is a minimum-weight common basis having label zero and C is a shortest non-zero directed cycle in $D_{M_1,M_2}(B)$.

4 Algebraic Algorithm for F-avoiding Basis and Common Basis

We present a randomized polynomial-time algorithm for F-avoiding Common Basis for representable matroids given as matrices over a field $\mathbb{F}$ and a finite group Γ given as an operation table. Our algorithm is a generalization of the exact-weight matroid intersection algorithm for representable matroids by Camerini, Galtiati, and Maffioli [11]. A similar algebraic algorithm has also been considered by Webb [51, Section 3.7].

Before describing the algorithm, we introduce needed algebraic notions and results. We assume that the arithmetic operations and the zero test over $\mathbb{F}$ can be performed in constant time. In this section, we use the multiplicative notation for the operation of Γ, and let e denote the group unit (zero) of Γ instead of 0. The *group ring* $K[\Gamma]$ of Γ over a field K is the set of formal K-coefficient linear combinations of the elements in Γ, i.e., $K[\Gamma] \coloneqq \{ \sum_{g\in\Gamma} a_g g \mid a_g \in K \ (g \in \Gamma) \}$. The addition and multiplication of $f = \sum_{g\in\Gamma} a_g g \in K[\Gamma]$ and $h = \sum_{g\in\Gamma} b_g g \in K[\Gamma]$ are naturally defined as $f + h = \sum_{g\in\Gamma}(a_g + b_g)g$ and $fh = \sum_{g,g'\in\Gamma} a_g b_{g'} gg'$. With these operations, $K[\Gamma]$ forms a commutative ring, containing K as a subring under the natural identification $K \ni a \mapsto ae \in K[\Gamma]$. Note that the operations of $K[\Gamma]$ and the zero test can be performed in polynomially many operations of K and Γ.

For finite sets R and C, we mean by an $R \times C$ *matrix* a matrix of size $|R| \times |C|$ whose rows and columns are identified with R and C, respectively. We simply write $[r] \times C$ as $r \times C$ for $r \in \mathbb{Z}_{\geq 0}$. Given a ground set E and a labeling $\psi\colon E \to \Gamma$, we define an $E \times E$ diagonal matrix D_ψ as follows: for every $j \in E$, we set the (j, j) diagonal entry of D_ψ as $x_j\psi(j)$, where x_j is an indeterminate (variable) whose actual value comes from $\mathbb{F}$. Then, D_ψ is regarded as a matrix over the group ring $\mathbb{F}(\{x_e\}_{e\in E})[\Gamma]$, where $\mathbb{F}(\{x_e\}_{e\in E})$ denotes the rational function field over $\mathbb{F}$ in $|E|$ indeterminates $\{x_e\}_{e\in E}$.

The following is a modification of a claim of Tomizawa and Iri [50], who first used the Cauchy–Binet formula in the context of linear matroid intersection.

▶ **Lemma 4.1.** *Let $\mathbb{F}$ be a field, M_1 and M_2 $\mathbb{F}$-representable matroids with the common ground set E and the same rank r, A_k an $r \times E$ matrix representing M_k for $k = 1, 2$, and $\psi\colon E \to \Gamma$ a labeling. Let $\Xi = A_1 D_\psi A_2^\top$. Then, the coefficient of $g \in \Gamma$ in $\det(\Xi)$ is a non-zero polynomial in $\{x_j\}_{j\in E}$ if and only if a common basis with label g exists.*

Proof. By the Cauchy–Binet formula, we can expand $\det(\Xi)$ as

$$\det(\Xi) = \sum_{B\in\binom{E}{r}} \det(A_1[B]) \det(A_2[B]) \prod_{j\in B} x_j \cdot \psi(B), \tag{1}$$

where $A_k[B]$ denotes the submatrix of A_k obtained by extracting the columns in B for $k = 1, 2$. Observe that $\det(A_1[B]) \det(A_2[B])$ is non-zero if and only if B is a common basis of M_1 and M_2, and the terms coming from different common bases do not cancel out thanks to the factor $\prod_{j\in B} x_j$, proving the claim. ◀

Lemma 4.1 together with the Schwartz–Zippel lemma [36, 48, 53], division-free determinant algorithm [27], search-to-decision reduction, and the field extension for small fields give rise to a randomized algebraic algorithm for F-avoiding Common Basis. The proof of the result is given in the full version.

▶ **Theorem 4.2.** *Let $\mathbb{F}$ be a field and M_1 and M_2 $\mathbb{F}$-representable matroids with the common ground set E. There is a randomized algorithm that, given matrices A_1 and A_2 over $\mathbb{F}$ representing M_1 and M_2, respectively, the operation table of a finite abelian group Γ, a group labeling $\psi\colon E \to \Gamma$, and a forbidden label set $F \subseteq \Gamma$, solves F-avoiding Common Basis in expected polynomial time.*

A *Pfaffian pair* is a pair of $r \times n$ matrices A_1, A_2 such that $\det(A_1[B]) \det(A_2[B])$ is a non-zero constant for any common basis B [51]. This property implies that if $\mathbb{F} = \mathbb{Q}$ and matroids are given as a Pfaffian pair, then no cancel-out occurs in the summands of $\det(\Xi)$ in the equation (1) even if we substitute 1 for all x_i. Therefore, we can derandomize the algorithm given in Theorem 4.2. Examples of common bases of matroid pairs representable by Pfaffian pairs include spanning trees, regular matroid bases, arborescences, perfect matchings in Pfaffian-orientable bipartite graphs, and node-disjoint S–T paths in planar graphs [39].

▶ **Theorem 4.3.** *F-avoiding Common Basis is polynomially solvable for $\mathbb{Q}$-representable matroids if matroids are given as a Pfaffian pair and a group is given as the operation table.*

We can also generalize Theorem 4.2 to a randomized pseudo-polynomial-time algorithm for the weighted problem as follows. See the full version for the proof.

▶ **Theorem 4.4.** *Let $\mathbb{F}$ be a field and M_1 and M_2 $\mathbb{F}$-representable matroids on a common ground set E. There is a randomized algorithm that, given matrices A_1 and A_2 over $\mathbb{F}$ representing M_1 and M_2, respectively, the operation table of a finite abelian group Γ, a group labeling $\psi\colon E \to \Gamma$, a forbidden label set $F \subseteq \Gamma$, and a weight function $w\colon E \to \mathbb{Z}$, solves Weighted F-avoiding Common Basis in pseudo-polynomial time in expectation. The algorithm can be derandomized if $\mathbb{F} = \mathbb{Q}$ and (A_1, A_2) is a Pfaffian pair.*

5 F-avoiding Basis with Fixed $|F|$

As we will see in Theorem 6.3, Zero Basis is hard for groups given by operation tables. This implies the hardness of F-avoiding Basis if the set F of forbidden labels is part of the input. In this section, we study the problem when F has a fixed size. Note that in contrast to the setting of Section 4, we assume that Γ is given as an operation oracle (and it is not necessarily finite).

Related to the notion of k-closeness recently introduced by Liu and Xu [35], we propose the following conjecture.

▶ **Conjecture 5.1.** *Let M be a matroid, $\psi\colon E(M) \to \Gamma$ a labeling, and $F \subseteq \Gamma$ a finite collection of forbidden labels. Then, for any basis B, there exists an F-avoiding basis B^* with $|B \setminus B^*| \leq |F|$, provided that at least one F-avoiding basis exists.*

Note that Lemma 3.2 (applied with a constant weight function) implies that Conjecture 5.1 holds for $|F| = 1$. A tightness example for Conjecture 5.1 can be found in the full version. We can relax Conjecture 5.1 as follows.

▶ **Conjecture 5.2.** *There exists a function $f\colon \mathbb{N} \to \mathbb{N}$ with the following property: If M is a matroid, $\psi\colon E(M) \to \Gamma$ is a group labeling, and $F \subseteq \Gamma$ is a finite collection of forbidden labels, then, for any basis B, there exists an F-avoiding basis B^* with $|B \setminus B^*| \leq f(|F|)$, provided that at least one F-avoiding basis exists.*

Conjectures 5.1 and 5.2 have algorithmic implications due to the following simple observation.

▶ **Lemma 5.3.** *Let α be a fixed positive integer. Further, let M be a matroid, $\psi\colon E(M) \to \Gamma$ a group labeling, and $F \subseteq \Gamma$ a finite collection of forbidden labels, such that, for any basis B, there exists an F-avoiding basis B^* with $|B \setminus B^*| \leq \alpha$, provided that at least one F-avoiding basis exists. Then, an F-avoiding basis of M can be found in polynomial time, if one exists.*

Proof. We first compute an arbitrary basis B of M. Then, for every $X \subseteq B$ and every $Y \subseteq E(M) \setminus B$ with $|X| = |Y| \leq \alpha$, we test whether $(B \setminus X) \cup Y$ is an F-avoiding basis of M. As there are at most $1 + n^\alpha$ choices for each of X and Y, the desired running time follows. If we find an F-avoiding basis during this procedure, we return it. Otherwise, no F-avoiding basis exists by assumption. ◀

The following is an immediate consequence of Lemma 5.3.

▶ **Corollary 5.4.** *If Conjecture 5.2 holds, then F-avoiding Basis is solvable in polynomial time if $|F|$ is fixed.*

Liu and Xu [35] defined a finite group Γ to be *k-close* for an integer $k \geq 1$, if for any matroid M, group labeling $\psi\colon E(M) \to \Gamma$, element $g \in \Gamma$ and basis B, there exists a basis B^* with $|B \setminus B^*| \leq k$ and $\psi(B^*) = g$, provided that M has at least one basis with label g. Observe that Conjecture 5.1 would imply $(|\Gamma|-1)$-closeness, and Conjecture 5.2 would imply $f(|\Gamma|-1)$-closeness of each finite group Γ for some function $f\colon \mathbb{N} \to \mathbb{N}$. This would imply an FPT algorithm for Zero Basis when parameterized with $|\Gamma|$ due to the following result, which is a consequence of [35, Theorem 1].

▶ **Theorem 5.5** (see Liu–Xu [35]). *Assume that for each finite group Γ, there exists an integer k such that Γ is k-close. Then,* Zero Basis *is in FPT for finite groups when parameterized by $|\Gamma|$.*

Liu and Xu [35] observed that if all subgroups of Γ satisfy a conjecture by Schrijver and Seymour [47], then Γ is $(|\Gamma|-1)$-close. By the results of DeVos, Goddyn, and Mohar [13], this implies that any cyclic group Γ is $(|\Gamma|-1)$-close whose order is a prime power or the product of two primes. The proof of [35, Theorem 4] does not seem to generalize to our setting. Thus, it is not clear whether the conjecture of Schrijver and Seymour implies Conjecture 5.1. If Γ has prime order, then Liu and Xu [35, Theorem 3] gave a simpler proof of $(|\Gamma|-1)$-closeness. That proof also generalizes to show that Conjecture 5.1 holds for such groups.

Using the results of Lemos [34], we can also prove that Conjecture 5.1 holds for *ordered groups*, which is a group Γ equipped with a total order $\leq$ on Γ such that $a \leq b$ implies $a + c \leq b + c$ for all $a, b, c \in \Gamma$. The result is restated in the following theorem, whose proof can be found in the full version.

▶ **Theorem 5.6.** *Let M be a matroid, $\psi\colon E(M) \to \Gamma$ a labeling to an ordered group Γ, $F \subseteq \Gamma$ a finite collection of forbidden labels, B a basis of M, and suppose that M has an F-avoiding basis. Then, there exists an F-avoiding basis B^* of M with $|B^* \setminus B| \leq |F|$.*

5.1 Strongly Base Orderable Matroids and Relaxations

In this section, we introduce a relaxed notion of strong base-orderability, called (α, k)-weak base orderability, where α and k are positive integers. In Section 5.1.1, we define this notion and show its relation to strong base-orderability and group-restricted bases. In Section 5.1.2 and Section 5.1.3, we conclude results for matroids representable over fixed finite fields and graphic matroids, respectively.

5.1.1 (α, k)-Weakly Base Orderable Matroids

A matroid is called *strongly base orderable* if for any two bases B_1, B_2, there exists a bijection $\varphi\colon B_1 \setminus B_2 \to B_2 \setminus B_1$ such that $(B_1 \setminus Z) \cup \varphi(Z)$ is a basis for each $Z \subseteq B_1 \setminus B_2$. For some positive integer k, we say that the ordered basis pair (B_1, B_2) has the *k-exchange property* if there exist pairwise disjoint nonempty subsets $X_1, \ldots, X_k \subseteq B_1 \setminus B_2$ and $Y_1, \ldots, Y_k \subseteq B_2 \setminus B_1$ such that $\left(B_1 \setminus \bigcup_{i \in Z} X_i\right) \cup \bigcup_{i \in Z} Y_i$ is a basis for each $Z \subseteq [k]$. For positive integers α and k, we define a matroid M to be *weakly (α, k)-base orderable* if the ordered pair (B_1, B_2) has the k-exchange property for any two bases B_1, B_2 of M with $|B_1 \setminus B_2| \geq \alpha$. We note that (α, k)-weak base orderability is a relaxation of *k-base orderability* defined by Bonin and Savitsky [7], and our definition of the k-exchange property differs from their definition of k-exchange-ordering. Observe that strongly base orderable matroids are precisely the matroids that are (k, k)-weakly base orderable for each $k \geq 1$.

For a matroid M and two disjoint bases B_1, B_2 of M with $B_1 \cup B_2 = E(M)$, we say that (B_1, B_2) is a *basis partition* of M. For a basis B of a matroid M, we say that a minor M' of M is a *B-minor* if it is obtained by contracting some elements of B and deleting some elements of $E(M) \setminus B$. We use the following simple observation later, whose proof can be found in the full version.

▶ **Lemma 5.7.** *Let B_1 and B_2 be two bases of a matroid M. Further, let M' be a B_1-minor of M such that (B_1', B_2') is a basis partition of M' and has the k-exchange property for some $k \in \mathbb{N}$, where $B_i' \coloneqq B_i \cap E(M')$ for $i = 1, 2$. Then (B_1, B_2) has the k-exchange property in M.*

The following result is our main motivation to consider weak base orderability. It establishes a connection between weak base orderability and Conjecture 5.2.

▶ **Theorem 5.8.** *Let M be a matroid, $\psi\colon E(M) \to \Gamma$ a group labeling, and $F \subseteq \Gamma$ a finite collection of forbidden labels. If M is $(\alpha, |F|+1)$-weakly base orderable, then for each basis B, there exists an F-avoiding basis B^* with $|B \setminus B^*| \leq \alpha - 1$, provided that at least one F-avoiding basis exists.*

For the proof, we need the following result, which is most likely routine; see the full version for the proof.

▶ **Proposition 5.9.** *Let S be a finite set, $\psi\colon S \to \Gamma$ a group labeling, and $0 \notin F \subseteq \Gamma$ satisfying $|F| \leq |S| - 1$. Then, there exists some nonempty $S' \subseteq S$ with $\psi(S') \notin F$.*

Proof of Theorem 5.8. Let $k \coloneqq |F|$ and let B be a basis and B' an F-avoiding basis minimizing $|B' \setminus B|$. If $|B' \setminus B| \leq \alpha - 1$, there is nothing to prove. We may hence suppose that $|B' \setminus B| \geq \alpha$. Then, as M is $(\alpha, k+1)$-weakly base orderable, there exist pairwise disjoint nonempty subsets $X_1, \dots, X_{k+1} \subseteq B' \setminus B$ and $Y_1, \dots, Y_{k+1} \subseteq B \setminus B'$ such that $\left(B' \setminus \bigcup_{i \in Z} X_i\right) \cup \bigcup_{i \in Z} Y_i$ is a basis for each $Z \subseteq [k+1]$. We define $\psi'\colon [k+1] \to \Gamma$ by $\psi'(i) = \psi(Y_i) - \psi(X_i)$ for all $i \in [k+1]$. Observe that $0 \notin F' \coloneqq \{ f - \psi(B') \mid f \in F \}$, as B' is an F-avoiding basis. It hence follows from Proposition 5.9 that there exists some nonempty $Z \subseteq [k+1]$ with $\psi'(Z) \notin F'$. Let $B'' \coloneqq \left(B' \setminus \bigcup_{i \in Z} X_i\right) \cup \bigcup_{i \in Z} Y_i$. By the definition of $X_1, \dots, X_{k+1}$ and $Y_1, \dots, Y_{k+1}$, we obtain that B'' is a basis of M. Further, we have $\psi(B'') = \psi(B') + \psi'(Z) \notin F$. Finally, we have $|B'' \setminus B| < |B' \setminus B|$ since Z is nonempty. This contradicts the choice of B'. ◀

As strongly base orderable matroids are (k, k)-weakly base orderable for any $k \geq 1$, we also get the following.

▶ **Corollary 5.10.** *Strongly base orderable matroids satisfy Conjecture 5.1.*

A connection of the results obtained in this section with certain orderings of elements of matroids introduced by Baumgart [3] can be found in the full version.

5.1.2 Matroids Representable over Finite Fields

In this section, we prove that the concept of weakly base orderability allows us to deal with a large class of matroids, namely all those which are representable over a fixed finite field. More precisely, we prove the following result.

▶ **Theorem 5.11.** *There is a function $f\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that for every prime power q, every $\mathrm{GF}(q)$-representable matroid is weakly $(f(q,k), k)$-orderable for any positive integer k.*

On a high level, the proof works in the following way. First, relying on results of [15] on the existence of certain submatrices of large matrices over finite fields, we show that every $\mathrm{GF}(q)$-representable matroid has a certain substructure. We then show that this substructure has the desired property. From this, we can conclude the theorem.

In order to find this substructure, we deal with the matrices representing the matroids in consideration. We first need some notation for these matrices. For two matrices A and A', we say that A contains A' as a *permuted submatrix* if A' can be obtained from A by deleting and permuting rows and columns. For a square matrix, we refer by its *size* to its number of rows. Let q be a prime power. We say that a triple (α, β, γ) of elements of $\mathrm{GF}(q)$ is *feasible* if $\alpha \neq \beta$ and at least one of $\beta \neq 0$ and $\gamma \neq 0$ hold. For a triple (α, β, γ) and a positive integer t, the (α, β, γ)*-diagonal matrix* of size t is the $t \times t$ matrix $A = (A_{ij})$ such that for $i, j \in [t]$, we have $A_{ij} = \alpha$ if $i < j$, $A_{ii} = \beta$ if $i = j$, and $A_{ij} = \gamma$ if $i > j$. We now collect some properties of (α, β, γ)-diagonal matrices. We first need the following result showing that (α, β, γ)-diagonal matrices can always be found in sufficiently large matrices over a fixed finite field. The following result can easily be concluded from a slightly weaker result due to Ding, Oporowski, Oxley, and Vertigan [15]. Its detailed proof can be found in the full version.

▶ **Proposition 5.12.** *There is a computable function $f_1 \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ with the following property: Let q be a prime power, t a positive integer, and A a matrix over $\mathrm{GF}(q)$ having at least $f_1(q, t)$ columns no two of which are identical. Then, A contains a permuted square submatrix A' of size t which is (α, β, γ)-diagonal for a feasible triple (α, β, γ).*

We are now ready to give the following result showing that every sufficiently large matroid that is representable over a fixed finite field has a certain substructure. The approach is to choose a matrix representing the matroid and find a particular submatrix in this matrix using Proposition 5.12. After, we show that a minor represented by this matrix can be obtained by applying certain deletions and contractions. The detailed proof can be found in the full version.

▶ **Lemma 5.13.** *There is a computable function $f_1 \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ with the following properties: Let q be a prime power, k a positive integer, M a $\mathrm{GF}(q)$-representable matroid of rank at least $f_1(q, k)$ and (B_1, B_2) a basis partition of M. Then, there exists a B_1-minor M' of M that can be represented by a matrix of the form $[I_k \ A]$, where I_k is the identity matrix of size k and A is an (α, β, γ)-diagonal matrix for a feasible triple (α, β, γ), and the columns of I_k correspond to the elements of B_1' and those of A correspond to the elements of B_2', where $B_i' \coloneqq B_i \cap E(M')$ for $i = 1, 2$.*

We will prove Theorem 5.11 by showing that matroids representable by a very specific class of matrices satisfy its conclusion. For this, we need a statement showing that certain matrices are nonsingular, which we derive from an explicit formula for the determinants of (α, β, γ)-triangular matrices due to Efimov [17]. The detailed proof can be found in the full version.

▶ **Proposition 5.14.** *Let q be a prime power, (α, β, γ) a feasible triple, t a multiple of $q(q-1)$, and A the (α, β, γ)-diagonal matrix of size t. Then, A is nonsingular.*

We are now ready to conclude the result for the specific class of matroids.

▶ **Lemma 5.15.** *There is a computable function $f_2 \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ with the following properties: Let q be a prime power, k a positive integer, and M a matroid that can be represented by $[I \ A]$ over $\mathrm{GF}(q)$, where I is an identity matrix of size $f_2(q, k)$ and A is an (α, β, γ)-diagonal*

matrix of the same size for a feasible triple (α, β, γ)*. Next, let* B_1 *and* B_2 *be the subsets of* $E(M)$ *corresponding to* I *and* A*, respectively. Then,* (B_1, B_2) *is a basis partition of* M *and has the* k*-exchange property.*

Proof. Let f_2 be the function defined by $f_2(q,k) \coloneqq q(q-1)k$ for all positive integers q and k. By Proposition 5.14, we have that A is nonsingular and hence (B_1, B_2) is a basis partition of M. For $i \in [k]$, let X_i be the subset of B_1 and Y_i be the subset of B_2 that corresponds to the columns of indices $q(q-1)(i-1)+1$ to $q(q-1)i$ of I and A, respectively. For $Z \subseteq [k]$, let $B_Z \coloneqq \left(B_1 \setminus \bigcup_{i \in Z} X_i\right) \cup \bigcup_{i \in Z} Y_i$. It suffices to prove that B_Z is a basis of M for every $Z \subseteq [k]$. To this end, consider some fixed $Z \subseteq [k]$. Observe that the matrix obtained from restricting $[I\ A]$ to the columns corresponding to B_Z can be transformed into a matrix of the form $A^* = \begin{bmatrix} I' & A_1 \\ O & A' \end{bmatrix}$ by exchanging rows and columns. Here, I' is the identity matrix of size $q(q-1)(k-|Z|)$, O is a zero matrix, A' is an (α, β, γ)-diagonal matrix of size $q(q-1)|Z|$, and A_1 is an arbitrary matrix. As the size of A' is divisible by $q(q-1)$, we obtain by Proposition 5.14 that A' is nonsingular. It follows that A^* is nonsingular, and hence B_Z is independent. As $|B_Z| = |B_1|$ by construction, we obtain that B_Z is a basis of M. This finishes the proof. ◀

Finally, we combine Lemmas 5.7, 5.13, and 5.15 to conclude Theorem 5.11.

Proof of Theorem 5.11. We prove the statement for the function $f\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ defined by $f(q,k) \coloneqq f_1(q, f_2(q,k))$ for each $k \in \mathbb{N}$ and prime power q. Let B_1 and B_2 be bases of a $\mathrm{GF}(q)$-representable matroid M with $|B_1 \setminus B_2| \geq f(q,k)$. We need to prove that (B_1, B_2) has the k-exchange property. Let $M' \coloneqq M / (B_1 \cap B_2) \setminus (E(M) \setminus (B_1 \cup B_2))$. Further, for $i = 1, 2$, let $B_i' \coloneqq B_i \cap E(M')$ and observe that (B_1', B_2') is a basis partition of M'. It follows from Lemma 5.13 that there exists a B_1'-minor M'' of M' that can be represented by a matrix of the form $[I\ A]$, where I is the identity matrix of size $f_2(q,k)$, A is an (α, β, γ)-diagonal matrix of size $f_2(q,k)$ for a feasible triple (α, β, γ) and the columns of I and A correspond to the elements of B_1'' and B_2'', respectively, where $B_i'' \coloneqq B_i' \cap E(M'')$ for $i = 1, 2$. We now obtain from Lemma 5.15 that (B_1'', B_2'') is a basis partition of M'' and has the k-exchange property in M''. As M'' is a B_1-minor of M, we now obtain from Lemma 5.7 that (B_1, B_2) has the k-exchange property in M. ◀

Combining Theorems 5.8 and 5.11, Lemma 5.3, and Theorem 5.5, we get the following.

▶ **Corollary 5.16.** *Let* q *be a prime power,* M *a* $\mathrm{GF}(q)$*-representable matroid,* $\psi\colon E \to \Gamma$ *a group labeling and* $F \subseteq \Gamma$ *a finite set of forbidden labels. When* $|F|$ *is fixed,* F-avoiding Basis *is solvable in polynomial time. Moreover, if* $|\Gamma|$ *is finite, then* Zero Basis *is in FPT when parameterized by* $|\Gamma|$*.*

We note that Corollary 5.16 is not implied by Theorem 4.2. The former applies to arbitrary groups, while the latter is limited to finite groups given by an operation table. Furthermore, Corollary 5.16 gives a deterministic polynomial-time algorithm, in contrast to the randomized algorithm in Theorem 4.2.

5.1.3 Graphic matroids

As a strengthening of the k-exchange property, we say that the basis pair (B_1, B_2) of a matroid has the *elementary* k*-exchange property* if there exist k-element subsets $X \subseteq B_1 \setminus B_2$ and $Y \subseteq B_2 \setminus B_1$ and a bijection $\varphi\colon X \to Y$ such that $(B_1 \setminus Z) \cup \varphi(Z)$ is a basis for each $Z \subseteq X$. Note that this is equivalent to requiring $|X_i| = |Y_i| = 1$ for each $i \in [k]$ in the

definition of the k-exchange property. We define a matroid M to be *elementarily* (α, k)*-weakly base orderable* if (B_1, B_2) has the elementary k-exchange property for any pair of basis B_1 and B_2 with $|B_1 \setminus B_2| \geq \alpha$.

It turns out that all regular matroids are elementarily $(f(k), k)$-weakly base orderable for some large function $f\colon \mathbb{N} \to \mathbb{N}$, while the same is not true for binary matroids. The proofs of these results can be found in the full version. As graphic matroids are regular, they are elementarily $(f(k), k)$-weakly base orderable for some large function $f\colon \mathbb{N} \to \mathbb{N}$. We give a proof in the full version, independent from the proof of Theorem 5.11, which shows that for graphic matroids, there exists such a function f satisfying $f(k) = O(k^3)$.

▶ **Theorem 5.17.** *Graphic matroids are elementarily* $(3k^3, k)$*-weakly base orderable for any* $k \geq 1$.

5.2 Two forbidden labels

The objective of this section is to prove the following restatement of the case $|F| = 2$ of Conjecture 5.1. Its proof can be found in the full version. Vaguely speaking, we first reduce the problem to matroids on six elements and then combine some earlier results with a particular treatment for the cycle matroid of K_4.

▶ **Theorem 5.18.** *Let M be a matroid, $\psi\colon E(M) \to \Gamma$ a group labeling, and let F be a 2-element subset of Γ. For any basis B, there exists an F-avoiding basis B^* such that $|B \setminus B^*| \leq 2$, provided that there exists at least one F-avoiding basis.*

6 Hardness and Negative Results

In this section, we give the algorithmic hardness results and counterexamples contained in this work. Sections 6.1 and 6.2 contain algorithmic intractability results and Section 6.3 contains a counterexample to a conjecture of Liu and Xu [35]. All proofs can be found in the full version.

6.1 Hardness of Non-zero Common Basis with $\mathbb{Z}_2 \leq \Gamma$

We here show that Non-Zero Common Basis is intractable for any group Γ such that $\mathbb{Z}_2 \leq \Gamma$. This implies that the condition on Γ in Theorem 3.7 is crucial indeed.

▶ **Theorem 6.1.** Non-Zero Common Basis *requires an exponential number of independence queries for any fixed group Γ such that $\mathbb{Z}_2 \leq \Gamma$.*

Our proof of Theorem 6.1 provides a new and simpler proof of the result of Bérczi and Schwarcz [5] showing that the problem of partitioning the ground set into common bases is hard. In addition to this new proof, we also describe the relation of a relaxation of that problem to a problem on non-zero common bases via dual lattices in the full version.

6.2 Hardness of Zero Basis

In this section, we show two hardness results for Zero Basis. The first one shows that the problem is hard even for uniform matroids by using the hardness of the well-known Subset Sum problem.

▶ **Theorem 6.2.** Zero Basis *is* NP-hard *for a uniform matroid and $\Gamma = \mathbb{Z}$.*

It can be derived from [16, Theorem 1.3] that Zero Basis is also hard for finite groups given as operation tables.

▶ **Theorem 6.3** (see Doron-Arad–Kulik–Shachnai [16, Theorem 1.3]). Zero Basis *requires an exponential number of independence queries for a finite group* Γ *given as an operation table.*

Recall that (Weighted) Zero Basis is solvable if Γ is a fixed, finite group [35]. In contrast, Theorem 6.1 implies that Non-Zero Common Basis is hard for any fixed group Γ such that $\mathbb{Z}_2 \leq \Gamma$. By modifying that proof, the hardness of Zero Common Basis follows even when the assumption $\mathbb{Z}_2 \leq \Gamma$ is dropped.

▶ **Theorem 6.4.** Zero Common Basis *requires an exponential number of independence queries for any nontrivial fixed group* Γ.

6.3 Counterexample to a Conjecture of Liu and Xu

Liu and Xu [35] proposed a conjecture which is even stronger than the implications from Conjectures 5.1 and 5.2. In order to state their conjecture, we need the following definition. For a finite abelian group Γ its *Davenport constant* $D(\Gamma)$ is defined as the minimum value such that every sequence of elements from Γ of length $D(\Gamma)$ contains a nonempty subsequence with sum 0. Liu and Xu proposed the following conjecture.

▶ **Conjecture 6.5** (Liu–Xu [35]). *Let* Γ *be a finite abelian group. Then,* Γ *is* $(D(\Gamma)-1)$*-close.*

We provide a counterexample for Conjecture 6.5. More precisely, we prove the following result.

▶ **Theorem 6.6.** *Let* $\Gamma = \mathbb{Z}_2^d$ *for some* $d \geq 4$. *Then,* Γ *is not* $(D(\Gamma)-1)$*-close.*

7 Conclusion

In this work, we have treated several problem settings on finding bases of group-labeled matroids whose labels satisfy certain conditions. Many questions remain open. In Section 3.2, we deal with Weighted Non-Zero Common Basis for groups Γ with $\mathbb{Z}_2 \not\leq \Gamma$ and give an approximation algorithm and exact algorithms for some special cases. However, the general complexity of Weighted Non-Zero Basis for $\mathbb{Z}_2 \not\leq \Gamma$ remains open. In Section 4, randomized algebraic algorithms turn out to be a powerful tool for finding bases and common bases of certain labels. It would be interesting to see whether more of the problems that can be solved by these randomized algorithms can also be solved deterministically. For example, one could consider Non-Zero Common Basis for arbitrary groups when one of the matroids is graphic, and the other one is a partition matroid. Finally, while Conjectures 5.1 and 5.2 remain wide open, the following stronger conjecture can be formulated analogously to the notion of strongly k-closeness introduced by Liu and Xu [35]. Note that the conjecture holds for $|F| = 1$ by Lemma 3.2, and it can also be shown that it holds for strongly base orderable matroids.

▶ **Conjecture 7.1.** *Let* M *be a matroid on a ground set* E, $\psi\colon E \to \Gamma$ *a group labeling,* $F \subseteq \Gamma$ *a finite subset, and* $w\colon E \to \mathbb{R}$ *a weight function. Suppose that* M *has an* F*-avoiding basis. Then, for any minimum weight basis* B*, there exists a minimum weight* F*-avoiding basis* B^* *such that* $|B \setminus B^*| \leq |F|$.

References

1 Stephan Artmann, Robert Weismantel, and Rico Zenklusen. A strongly polynomial algorithm for bimodular integer linear programming. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2017)*. ACM, 2017. doi:10.1145/3055399.3055473.

2 Francisco Barahona and William R. Pulleyblank. Exact arborescences, matchings and cycles. *Discrete Applied Mathematics*, 16(2):91–99, 1987. doi:10.1016/0166-218x(87)90067-9.

3 Matthias Baumgart. *Ranking and ordering problems of spanning trees*. PhD thesis, Technische Universität München, München, 2009.

4 Kristóf Bérczi, Gergely Csáji, and Tamás Király. On the complexity of packing rainbow spanning trees. *Discrete Mathematics*, 346(4):113297, 2023. doi:10.1016/j.disc.2022.113297.

5 Kristóf Bérczi and Tamás Schwarcz. Complexity of packing common bases in matroids. *Mathematical Programming*, 188(1):1–18, 2021. doi:10.1007/s10107-020-01497-y.

6 Nayantara Bhatnagar, Dana Randall, Vijay V. Vazirani, and Eric Vigoda. Random bichromatic matchings. *Algorithmica*, 50(4):418–445, 2007. doi:10.1007/s00453-007-9096-4.

7 Joseph E. Bonin and Thomas J. Savitsky. An infinite family of excluded minors for strong base-orderability. *Linear Algebra and its Applications*, 488:396–429, 2016. doi:10.1016/j.laa.2015.09.055.

8 André Bouchet. Greedy algorithm and symmetric matroids. *Mathematical Programming*, 38(2):147–159, 1987. doi:10.1007/BF02604639.

9 Carl Brezovec, Gérard Cornuéjols, and Fred Glover. Two algorithms for weighted matroid intersection. *Mathematical Programming*, 36(1):39–53, 1986. doi:10.1007/bf02591988.

10 Richard A. Brualdi. Comments on bases in dependence structures. *Bulletin of the Australian Mathematical Society*, 1(2):161–167, 1969. doi:10.1017/s000497270004140x.

11 Paolo M. Camerini, Giulia Galbiati, and Francesco Maffioli. Random pseudo-polynomial algorithms for exact matroid problems. *Journal of Algorithms*, 13(2):258–273, 1992. doi:10.1016/0196-6774(92)90018-8.

12 Maria Chudnovsky, Jim Geelen, Bert Gerards, Luis Goddyn, Michael Lohman, and Paul Seymour. Packing non-zero A-paths in group-labelled graphs. *Combinatorica*, 26(5):521–532, 2006. doi:10.1007/s00493-006-0030-1.

13 Matt DeVos, Luis Goddyn, and Bojan Mohar. A generalization of Kneser's Addition Theorem. *Advances in Mathematics*, 220(5):1531–1548, 2009. doi:10.1016/j.aim.2008.11.003.

14 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Berlin, fifth edition, 2017.

15 Guoli Ding, Bogdan Oporowski, James Oxley, and Dirk Vertigan. Unavoidable minors of large 3-connected binary matroids. *Journal of Combinatorial Theory, Series B*, 66(2):334–360, 1996. doi:10.1006/jctb.1996.0026.

16 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. Lower bounds for matroid optimization problems with a linear constraint. In *51th International Colloquium on Automata, Languages, and Programming (ICALP 2024)*, 2024. To appear. arXiv:2307.07773.

17 Dmitry Efimov. Determinant of three-layer Toeplitz matrices. *Journal of Integer Sequences*, 24(2):3, 2021.

18 Nicolas El Maalouly, Raphael Steiner, and Lasse Wulf. Exact matching: Correct parity and FPT parameterized by independence number. In *34th International Symposium on Algorithms and Computation (ISAAC 2023)*, pages 28:1–28:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ISAAC.2023.28.

19 András Frank. A weighted matroid intersection algorithm. *Journal of Algorithms*, 2(4):328–336, 1981. doi:10.1016/0196-6774(81)90032-8.

20 András Frank. *Connections in Combinatorial Optimization*, volume 38 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2011.

21 Satoru Fujishige. A primal approach to the independent assignment problem. *Journal of the Operations Research Society of Japan*, 20(1):1–15, 1977. doi:10.15807/jorsj.20.1.

22 Anna Galluccio and Martin Loebl. On the theory of Pfaffian orientations. I. Perfect matchings and permanents. *The Electronic Journal of Combinatorics*, 6(1), 1998. doi:10.37236/1438.

23 Nicholas J. A. Harvey, Tamás Király, and Lap Chi Lau. On disjoint common bases in two matroids. *SIAM Journal on Discrete Mathematics*, 25(4):1792–1803, 2011. doi:10.1137/100815232.

24 Florian Hörsch, Tomáš Kaiser, and Matthias Kriesell. Rainbow bases in matroids. *arXiv preprint*, 2022. arXiv:2206.10322.

25 Yoichi Iwata and Yutaro Yamaguchi. Finding a shortest non-zero path in group-labeled graphs. *Combinatorica*, 42(S2):1253–1282, 2022. doi:10.1007/s00493-021-4736-x.

26 Xinrui Jia, Ola Svensson, and Weiqiang Yuan. The exact bipartite matching polytope has exponential extension complexity. In *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2023)*, pages 1635–1654. SIAM, 2023. doi:10.1137/1.9781611977554.ch61.

27 Erich Kaltofen and Gilles Villard. On the complexity of computing determinants. *Computational Complexity*, 13(3-4):91–130, 2005. doi:10.1007/s00037-004-0185-3.

28 Yasushi Kawase, Yusuke Kobayashi, and Yutaro Yamaguchi. Finding a path with two labels forbidden in group-labeled graphs. *Journal of Combinatorial Theory, Series B*, 143:65–122, 2020. doi:10.1016/j.jctb.2019.12.001.

29 Donggyu Kim, Duksang Lee, and Sang-il Oum. Γ-graphic delta-matroids and their applications. *Combinatorica*, 43(5):963–983, 2023. doi:10.1007/s00493-023-00043-6.

30 Stein Krogdahl. A combinatorial base for some optimal matroid intersection algorithms. Technical Report STAN-CS-74-468, Computer Science Department, Stanford University, Stanford, CA, 1974.

31 Stein Krogdahl. A combinatorial proof for a weighted matroid intersection algorithm. Technical Report Computer Science Report 17, Institute of Mathematical and Physical Sciences, University of Tromso, Tromso, 1976.

32 Stein Krogdahl. The dependence graph for bases in matroids. *Discrete Mathematics*, 19(1):47–59, 1977. doi:10.1016/0012-365X(77)90118-2.

33 Andrea S. Lapaugh and Christos H. Papadimitriou. The even-path problem for graphs and digraphs. *Networks*, 14(4):507–513, 1984. doi:10.1002/net.3230140403.

34 Manoel Lemos. Weight distribution of the bases of a matroid. *Graphs and Combinatorics*, 22(1):69–82, 2006. doi:10.1007/s00373-005-0648-6.

35 Siyue Liu and Chao Xu. On the congruency-constrained matroid base. In *Proceedings of the 25th Conference on Integer Programming and Combinatorial Optimization (IPCO '24)*, 2024. To appear. arXiv:2311.11737.

36 László Lovász. On determinants, matchings, and random algorithms. In Lothar Budach, editor, *Fundamentals of Computation Theory, FCT '79, Proceedings of the Conference on Algebraic, Arthmetic, and Categorial Methods in Computation Theory*, pages 565–574. Akademie-Verlag, Berlin, 1979.

37 László Lovász. Some algorithmic problems on lattices. In László Lovász and Endre Szemerédi, editors, *Theory of algorithms*, volume 44 of *Colloquia Mathematica Societatis János Bolyai*, pages 323–337. North-Holland, Amsterdam, 1985.

38 László Lovász. Matching structure and the matching lattice. *Journal of Combinatorial Theory, Series B*, 43(2):187–222, 1987. doi:10.1016/0095-8956(87)90021-9.

39 Kazuki Matoya and Taihei Oki. Pfaffian pairs and parities: counting on linear matroid intersection and parity problems. *SIAM Journal on Discrete Mathematics*, 36(3):2121–2158, 2022. doi:10.1137/21M1421751.

40 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987. doi:10.1007/bf02579206.

41 Martin Nägele, Richard Santiago, and Rico Zenklusen. Congruency-constrained TU problems beyond the bimodular case. *Mathematics of Operations Research*, 2023. doi:10.1287/moor.2023.1381.

42 Martin Nägele, Benny Sudakov, and Rico Zenklusen. Submodular minimization under congruency constraints. *Combinatorica*, 39(6):1351–1386, 2019. doi:10.1007/s00493-019-3900-1.

43 Martin Nägele and Rico Zenklusen. A new contraction technique with applications to congruency-constrained cuts. *Mathematical Programming*, 183(1):455–481, 2020. doi:10.1007/s10107-020-01498-x.

44 Christos H. Papadimitriou and Mihalis Yannakakis. The complexity of restricted spanning tree problems. *Journal of the ACM*, 29(2):285–309, 1982. doi:10.1145/322307.322309.

45 Jörg Rieder. The lattices of matroid bases and exact matroid bases. *Archiv der Mathematik*, 56(6):616–623, 1991. doi:10.1007/bf01246778.

46 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, Berlin, 2003.

47 Alexander Schrijver and Paul D. Seymour. Spanning trees of different weights. In William J. Cook and Paul D. Seymour, editors, *Polyhedral Combinatorics*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 281–288. DIMACS/AMS, 1990. doi:10.1090/DIMACS/001/21.

48 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.

49 Ola Svensson and Jakub Tarnawski. The matching problem in general graphs is in quasi-NC. In *Proceedings of the 58th Annual Symposium on Foundations of Computer Science (FOCS 2017)*. IEEE, 2017. doi:10.1109/focs.2017.70.

50 Nobuaki Tomizawa and Masao Iri. An algorithm for determining the rank of a triple matrix product AXB with application to the problem of discerning the existence of the unique solution in a network (in Japanese). *Electronics and Communications in Japan*, 57(11):50–57, 1974.

51 Kerri P. Webb. *Counting Bases*. PhD thesis, University of Waterloo, Waterloo, ON, 2004.

52 Raphael Yuster. Almost exact matchings. *Algorithmica*, 63(1–2):39–50, 2012. doi:10.1007/s00453-011-9519-0.

53 Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward W Ng, editor, *Symbolic and Algebraic Computation*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, Berlin, 1979. doi:10.1007/3-540-09519-5_73.

Finding Most-Shattering Minimum Vertex Cuts of Polylogarithmic Size in Near-Linear Time

Kevin Hua
University of Michigan, Ann Arbor, MI, USA

Daniel Li
University of Michigan, Ann Arbor, MI, USA

Jaewoo Park
University of Michigan, Ann Arbor, MI, USA

Thatchaphol Saranurak
University of Michigan, Ann Arbor, MI, USA

Abstract

We show the first near-linear time randomized algorithms for listing all minimum vertex cuts of polylogarithmic size that separate the graph into at least *three* connected components (also known as *shredders*) and for finding the *most shattering* one, i.e., the one maximizing the number of connected components. Our algorithms break the quadratic time bound by Cheriyan and Thurimella (STOC'96) for both problems that has been unimproved for more than two decades. Our work also removes an important bottleneck to near-linear time algorithms for the vertex connectivity augmentation problem (Jordan '95) and finding an even-length directed cycle in a graph, a problem shown to be equivalent to many other fundamental problems (Vazirani and Yannakakis '90, Robertson et al. '99). Note that it is necessary to list only minimum vertex cuts that separate the graph into at least three components because there can be an exponential number of minimum vertex cuts in general.

To obtain a near-linear time algorithm, we have extended techniques in local flow algorithms developed by Forster et al. (SODA'20) to list shredders on a local scale. We also exploit fast queries to a pairwise vertex connectivity oracle subject to vertex failures (Long and Saranurak FOCS'22, Kosinas ESA'23). This is the first application of using connectivity oracles subject to vertex failures to speed up a static graph algorithm.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Graphs, Flows, Randomized Algorithms, Vertex Connectivity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.87

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2405.03801`

Funding *Thatchaphol Saranurak*: Supported by NSF grant CCF-2238138.

1 Introduction

Given an undirected graph G with n vertices and m edges, a *minimum vertex cut* is a smallest set of vertices whose removal disconnects G. The *vertex connectivity* of G is the size of any minimum vertex. The problem of efficiently computing vertex connectivity and finding a corresponding minimum vertex cut has been extensively studied for more than half a century [16, 25, 9, 8, 12, 7, 21, 1, 19, 5, 22, 4, 13, 14, 11, 2]. Let k denote the vertex connectivity of G. Recently, a $\tilde{\mathcal{O}}(m + nk^3)$-time algorithm was shown in [10], which is near-linear when $k = \mathcal{O}(\text{polylog}(n))$. Then, an almost-linear $\mathcal{O}(m^{1+o(1)})$-time algorithm for the general case was finally discovered [18, 3]. In this paper, we show new algorithms for two closely related problems:

1. Find a *most-shattering* minimum vertex cut, i.e., a minimum vertex cut S such that the number of (connected) components of $G \setminus S$ is maximized over all minimum vertex cuts.
2. List all minimum vertex cuts S such that $G \setminus S$ has at least three components.

In the latter problem, the restriction to at least three components is natural for polynomial-time algorithms. This is because there are at most n many minimum vertex cuts whose removal results in at least three components [15], but the total number of minimum vertex cuts can be exponential or, more specifically, at least $2^k(n/k)^2$ [24][1]. We say that a vertex set S is a *separator* if $G \setminus S$ is not connected and a *shredder* if $G \setminus S$ has at least three components. An s-separator (s-shredder) is a separator (shredder) of size s. In other words, the second problem is to list all k-shredders[2].

The state-of-the-art algorithm for both Problems 1 and 2 was discovered by Cheriyan and Thurimella [6] and runs in $\mathcal{O}(k^2n^2 + k^3n\sqrt{n})$ time. This bound has remained unimproved for over two decades. Their approach inherently requires quadratic time because their algorithm makes $\Omega(n + k^2)$ max flow calls. Naturally, one may ask whether a subquadratic algorithm exists.

Our Contribution

We answer the above question in the affirmative by showing a randomized algorithm for listing all k-shredders and computing a most shattering min-cut in near-linear time for all $k = \mathcal{O}(\text{polylog}(n))$. Our main results are stated below.

▶ **Theorem 1.1.** *Let $G = (V, E)$ be an n-vertex m-edge undirected graph with vertex connectivity k. There exists an algorithm that takes G as input and correctly lists all k-shredders of G with probability $1 - n^{-97}$ in $\mathcal{O}(m + k^5 n \log^4 n)$ time.*

▶ **Theorem 1.2.** *Let $G = (V, E)$ be an n-vertex m-edge undirected graph with vertex connectivity k. There exists a randomized algorithm that takes G as input and returns a most shattering minimum-cut (if one exists) with probability $1 - n^{-97}$. The algorithm runs in $\mathcal{O}(m + k^5 n \log^4 n)$ time.*

▶ Remark 1.3. The $k^5 n \log^4 n$ term in both theorems can be improved to $k^3 n^{1+o(1)}$ using a pairwise vertex connectivity oracle developed by Long and Saranurak in [20], which is faster for large values of k.

Given recent developments in fast algorithms for computing vertex connectivity, one might expect that some of these modern techniques (e.g. local flow [23, 10], sketching [18], expander decomposition [26]) will be useful for listing shredders and finding most shattering min-cuts. It turns out that they are indeed useful, but *not enough*.

We have extended the techniques developed for local flow algorithms [10] to list k-shredders and compute the number of components that they separate. Specifically, our local algorithm lists k-shredders that separate the graph in an unbalanced way in time *proportional to the smaller side* of the cut. To this end, we generalize the structural results related to shredders from [6] to the local setting. To carry out this approach, we bring a new tool into the area – our algorithm queries a pairwise connectivity oracle subject to vertex failures [20, 17]. Surprisingly, this is the first application of using connectivity oracles subject to vertex failures to speed up a static graph algorithm.

---

[1] Both problems are specific to vertex cuts; recall that every minimum *edge* cut always separates a graph into two components.

[2] E.g. in a tree, vertices with degree at least three are 1-shredders. In the complete bipartite graph $K_{k,k}$ with $k \geq 3$, both the left and right halves of the bipartition are k-shredders.

2 Technical Overview

Let $G = (V, E)$ be an n-vertex m-edge undirected graph with vertex connectivity k. Cheriyan and Thurimella developed a deterministic algorithm called `All-`k`-shredders`$(\cdot)$ that takes G as input and lists all k-shredders of G in $\mathcal{O}(knm + k^2\sqrt{n}m)$ time. They improved this bound by using the sparsification routine developed in [22] as a preprocessing step. Specifically, there exists an algorithm that takes G as input and produces an edge subgraph G' on $\mathcal{O}(kn)$ edges such that all k-shredders of G are k-shredders of G' and vice versa. The algorithm runs in $\mathcal{O}(m)$ time. Using this preprocessing step, they obtained the bound for listing all k-shredders in $\mathcal{O}(m) + \mathcal{O}(k(kn)n + k^2\sqrt{n}(kn)) = \mathcal{O}(k^2n^2 + k^3n\sqrt{n})$ time.

In this paper, we resolve a bottleneck of `All-`k`-shredders`$(\cdot)$, improving the time complexity of listing all k-shredders from $\mathcal{O}(knm + k^2\sqrt{n}m)$ to $\mathcal{O}(k^4m\log^4 n)$. Using the same sparsification routine, our algorithm runs in $\mathcal{O}(m + k^5n\log^4 n)$ time. The proofs of the theorems and lemmas presented here have been omitted to the full version of the paper.

2.1 The Bottleneck

A key subroutine of `All-`k`-shredders`$(\cdot)$ is a subroutine called `Shredders`$(\cdot,\cdot)$ that takes a pair of vertices (x, y) as input and lists all k-shredders that separate x and y. This subroutine takes $\mathcal{O}(m)$ time plus the time to compute a flow of size k, which is at most $\mathcal{O}(mk)$ time. The idea behind `All-`k`-shredders`$(\cdot)$ is to call `Shredders`$(\cdot,\cdot)$ multiple times to list all k-shredders. `All-`k`-shredders`$(\cdot)$ works as follows. Let Y be an arbitrary set of $k + 1$ vertices. Any k-shredder S will either separate a pair of vertices in Y, or separate a component containing $Y \setminus S$ from the rest of the graph.

For the first case, they call `Shredders`(u, v) for all pairs of vertices $(u, v) \in Y \times Y$. This takes $\mathcal{O}(k^2)$ calls to `Shredders`$(\cdot,\cdot)$, which in total runs in $\mathcal{O}(k^3m)$ time. For the second case, we can add a dummy vertex z and connect z to all vertices in Y. Notice that any k-shredder S separating a component containing $Y \setminus S$ from the rest of the graph must separate z and a vertex $v \in V \setminus (Y \cup S)$. We can find these k-shredders by calling `Shredders`(z, v) for all $v \in V \setminus Y$. The bottleneck of Cheriyan's algorithm is listing k-shredders that fall into the second case. They call `Shredders`$(\cdot,\cdot)$ $\Omega(n)$ times as $|V \setminus Y| = n - \mathcal{O}(k) = \Theta(n)$ for small values of k. This step is precisely where the quadratic $\mathcal{O}(knm)$ factor comes from. To bypass this bottleneck, we categorize the problem into different cases and employ randomization.

2.2 Quantifying a Notion of Balance

Let S be a k-shredder of G. A useful observation is that if we obtain vertices x and y in different components of $G \setminus S$, then `Shredders`(x, y) will list S. To build on this observation, one can reason about the relative *sizes* of the components of $G \setminus S$ and employ a random sampling approach. Let C denote the "largest" component of $G \setminus S$. If C does not greatly outsize the remaining components, we can obtain two vertices in different components of $G \setminus S$ without too much difficulty. To capture the idea of relative *size* between components of $G \setminus S$, we define a quantity called *volume*. This definition is used commonly throughout the literature, but we have slightly altered it here.

▶ **Definition 2.1** (Volume). *For a vertex set Q, we refer to the* volume *of Q, denoted by* $\mathrm{vol}(Q)$*, to denote the quantity*

$$\mathrm{vol}(Q) = |\{(u, v) \in E \mid u \in Q\}|.$$

If $\mathcal{Q}$ is a collection of disjoint vertex sets, then we define the volume of $\mathcal{Q}$ as the quantity

$$\mathrm{vol}(\mathcal{Q}) = \mathrm{vol}\left(\bigcup_{Q\in\mathcal{Q}} Q\right).$$

For convenience, we will say that a k-shredder S admits *partition* $(C, \mathcal{R})$ to signify that C is the largest component of $G \setminus S$ by volume and $\mathcal{R}$ is the set of remaining components. We will also say $x \in \mathcal{R}$ to denote a vertex in a component of $\mathcal{R}$. We can categorize a k-shredder S by comparing the volume ratio between the largest component of $G \setminus S$ and the union of remaining components (the ratio between $\mathrm{vol}(C)$ and $\mathrm{vol}(\mathcal{R})$).

▶ **Definition 2.2** (Balanced/Unbalanced k-Shredders). *Let S be a k-shredder of G with partition $(C, \mathcal{R})$. We say S is* balanced *if $\mathrm{vol}(\mathcal{R}) \geq m/k$. Conversely, we say S is* unbalanced *if $\mathrm{vol}(\mathcal{R}) < m/k$.*

Suppose that S is a k-shredder with partition $(C, \mathcal{R})$. Building on our discussion above, if S is balanced, then C does not greatly outsize the rest of the graph (by a factor of k at most). Conversely, if S is unbalanced, then C greatly outsizes the rest of the graph.

2.3 Listing Balanced k-Shredders via Edge Sampling

Listing all balanced k-shredders turns out to be straightforward via random edge sampling. Let S be a k-shredder that admits partition $(C, \mathcal{R})$. Suppose that S is balanced. The idea is to sample an edges (x, x') and (y, y') such that x and y are in different components of $G \setminus S$. Then, `Shredders`(x, y) will list S. Intuitively, this approach works because we know that each component of $G \setminus S$ cannot be too large, as S is a balanced k-shredder. Hence, if we sample two edges (x, x') and (y, y') at random, the probability that x and y live in different components of $G \setminus S$ is some constant. We can then boost the success probability by repeating the procedure for a polylogarithmic number of times. The formal statement is given below.

▶ **Lemma 2.3.** *Let $G = (V, E)$ be an n-vertex m-edge undirected graph with vertex connectivity k. There exists a randomized algorithm that takes G as input and returns a list $\mathcal{L}$ that satisfies the following. If S is a balanced k-shredder of G, then $S \in \mathcal{L}$ with probability $1 - n^{-100}$. Additionally, every set in $\mathcal{L}$ is a k-shredder of G. The algorithm runs in $\mathcal{O}(k^2 m \log n)$ time.*

Because handling balanced k-shredders is simple, we omit the algorithm for listing balanced k-shredders and its analysis. We will primarily focus on our methods of listing unbalanced k-shredders.

2.4 Listing Unbalanced k-Shredders via Local Flow

Unbalanced k-shredders are more difficult to list than balanced k-shredders. For balanced k-shredders S with partition $(C, \mathcal{R})$, we can obtain two vertices in different components of $G \setminus S$ without much difficulty. This is no longer the case for unbalanced k-shredders, mainly because $\mathrm{vol}(\mathcal{R})$ may be arbitrarily small. Hence, most sampled edges will be incident to C. To handle unbalanced k-shredders, we introduce a new structural definition.

▶ **Definition 2.4** (Capture). *Let S be an unbalanced k-shredder of a graph $G = (V, E)$ with partition $(C, \mathcal{R})$. Consider an arbitrary tuple (x, ν, Π), where x is a vertex in V, ν is a positive integer, and Π is a set of paths. We say that the tuple (x, ν, Π)* captures *S if the following holds.*

1. *x is in a component of $\mathcal{R}$.*
2. *$\frac{1}{2}\nu < \mathrm{vol}(\mathcal{R}) \leq \nu$.*
3. *Π is a set of k openly-disjoint simple paths, each starting from x and ending at a vertex in C, such that the sum of lengths over all paths is at most $k^2\nu$.*

At a high level, we will spend some time constructing random tuples (x, ν, Π) in the hopes that one of the tuples captures S. The main result is stated below.

▶ **Lemma 2.5.** *Let G be a graph with vertex connectivity k. Let (x, ν, Π) be a tuple where x is a vertex, ν is a positive integer, and Π is a set of paths. There exists a deterministic algorithm that takes (x, ν, Π) as input and outputs a list $\mathcal{L}$ of k-shredders and one set U such that the following holds. If S is a k-shredder that is captured by (x, ν, Π), then $S \in \mathcal{L}$ or $S = U$. The algorithm runs in $\mathcal{O}(k^2\nu \log \nu)$ time.*

What is most important about this result is that we have constructed an algorithm that can identify k-shredders on a *local* scale. This means it spends time proportional to an input volume parameter ν instead of a global quantity like n or m. The idea behind Lemma 2.5 is to modify `Shredders`$(\cdot, \cdot)$ using recent advancements in local flow algorithms.

2.5 Verification via Pairwise Connectivity Oracles

In our algorithm, we will obtain a list of k-shredders and a list of unverified sets. The union of these two sets will include all k-shredders of G with high probability. However, within the list of unverified sets, there may be some false k-shredders. To filter out the false k-shredders, we utilize a pairwise connectivity oracle subject to vertex failures developed by Kosinas in [17]. To determine whether an unverified set S is a k-shredder, we will make some pairwise connectivity queries between vertex pairs (u, v) to determine whether u, v are disconnected in $G \setminus S$. These local queries, along with some more structural observations, will help us determine whether $G \setminus S$ contains at least three components.

3 Preliminaries

This paper concerns finite, undirected, and unweighted graphs with vertex connectivity k. We use standard graph-theoretic definitions. Let $G = (V, E)$ be a graph with vertex connectivity k. For a vertex subset $S \subseteq V$, we use $G \setminus S$ to denote the subgraph of G induced by removing all vertices in S. A *connected component* (component for short) of a graph refers to any maximally-connected subgraph, or the vertex set of such a subgraph. Suppose that S is a k-shredder of G. The *largest component* of $G \setminus S$ is the component with the greatest *volume*, where volume is defined in Definition 2.1. We break ties arbitrarily. For convenience, we will say that a k-shredder S admits partition $(C, \mathcal{R})$ to signify that C is the largest component of $G \setminus S$ and $\mathcal{R}$ is the set of remaining components. We will also write $x \in \mathcal{R}$ to indicate a vertex in a component of $\mathcal{R}$.

For a vertex subset $Q \subseteq V$, we define the set of neighbors of Q as the set $N(Q) = \{v \in V \setminus Q \mid (u, v) \in E, u \in Q\}$. For vertex subsets $Q_1, Q_2 \subseteq V$, we define $E(Q_1, Q_2) = \{(u, v) \in E \mid u \in Q_1, v \in Q_2\}$.

Let π be a simple path in G. We refer to the *length* of π as the number of edges in π. Suppose π starts at a vertex $x \in V$. We say the *far-most endpoint* of π to denote the other endpoint of π. Although all paths we refer to are undirected, our usage of the far-most endpoint will be unambiguous. Let x and y be two arbitrary vertices in π. We denote $\pi[x \rightsquigarrow y]$ as the subpath of π from x to y. We denote $\pi(x \rightsquigarrow y)$ as the subpath $\pi[x \rightsquigarrow y]$ *excluding* the

vertices x and y. We say two paths π, π' are *openly-disjoint* if they share no vertices except their endpoints. Let Π be a set of paths. Then, Π is openly disjoint if all pairs of paths in Π are openly disjoint. We say $v \in \Pi$ to denote a vertex among one of the paths in Π and $(u, v) \in \Pi$ to denote an edge used by one of the paths in Π.

4 Cheriyan and Thurimella's Algorithm

We will use $\texttt{Shredders}(\cdot, \cdot)$ as a subroutine and extend it to a *localized setting*. To do this, we need to review the terminology and ideas presented in [6].

▶ **Theorem 4.1** ([6, Algorithm 2]). *Let G be an n-vertex m-edge undirected graph with vertex connectivity k. Let x and y be two distinct vertices. There exists a deterministic algorithm $\texttt{Shredders}(\cdot, \cdot)$ that takes (x, y) as input and returns all k-shredders of G separating x and y in $\mathcal{O}(km)$ time.*

At a high level, $\texttt{Shredders}(\cdot, \cdot)$ works as follows. Let x and y be two vertices in a k-vertex-connected graph G. Firstly, we use a flow algorithm to obtain a set Π of k openly-disjoint simple paths from x to y. Observe that any k-shredder S of G separating x and y must contain exactly one vertex from each path of Π. More crucially, at least one component of $G \setminus S$ must also be a component of $G \setminus \Pi$. That is, for some component Q of $G \setminus \Pi$, we have $N(Q) = S$. This is the main property that $\texttt{Shredders}(\cdot, \cdot)$ exploits. It lists potential k-shredders by finding components Q of $G \setminus \Pi$ such that $|N(Q)| = k$ and $N(Q)$ consists of exactly one vertex from each path in Π (see Figure 1).

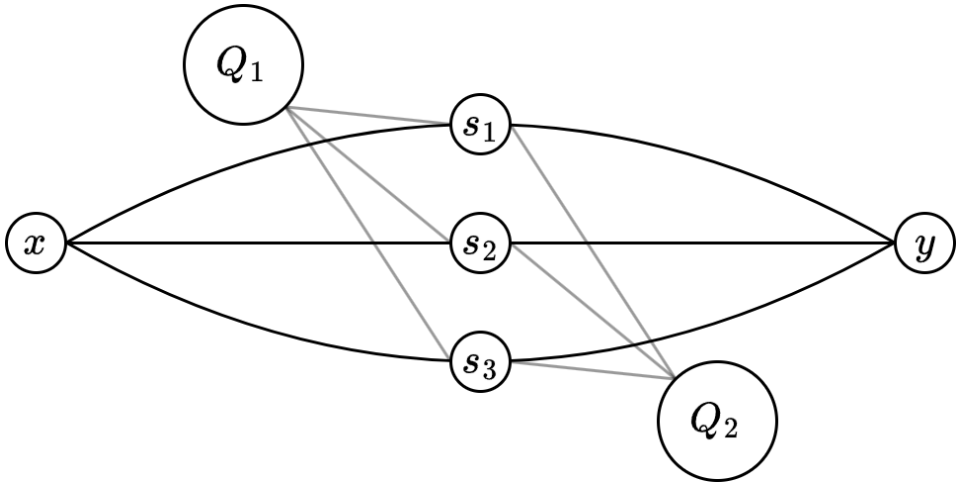


Figure 1 Let Π be the set of three openly-disjoint simple paths from x to y. A call to $\texttt{Shredders}(x, y)$ will identify $N(Q_1) = N(Q_2)$ as potential 3-shredders.

We can state these facts formally with the following definitions.

▶ **Definition 4.2** (Bridge). *A* bridge Γ *of* Π *is a component of $G \setminus \Pi$ or an edge $(u, v) \in E$ such that $(u, v) \notin \Pi$, but $u \in \Pi$ and $v \in \Pi$.*

If Γ is an edge, we define $\text{vol}(\Gamma) = 1$. Otherwise, $\text{vol}(\Gamma)$ is defined according to Definition 2.1.

▶ **Definition 4.3** (Attachments). *Let Γ be a bridge of Π. If Γ is a component of $G \setminus \Pi$, then the set of* attachments *of Γ is the vertex set $N(\Gamma)$. Otherwise, if Γ is an edge $(u, v) \in E$, the set of* attachments *of Γ is the vertex set $\{u, v\}$.*

For convenience, we refer to a single vertex among the attachments of Γ as an attachment of Γ. For a path $\pi \in \Pi$, we denote $\pi(\Gamma)$ as the set of attachments of Γ that are in π. If Q is an arbitrary vertex set, then we use $\pi(Q)$ to denote the set $Q \cap \pi$. If $\pi(Q)$ is a singleton set, then we may use $\pi(Q)$ to represent the unique vertex in $Q \cap \pi$. In all contexts, it will be clear what object the notation is referring to.

▶ **Definition 4.4** (k-Tuple). *A set S is called a k-tuple with respect to Π if $|S| = k$ and for all $\pi \in \Pi$, we have $|S \cap \pi| = 1$.*

We adopt the following notation as in $\texttt{Shredders}(\cdot,\cdot)$. If a component Γ of $G \setminus \Pi$ is such that $N(\Gamma)$ forms a k-tuple with respect to Π, then $N(\Gamma)$ is called a *candidate k-shredder* or *candidate* for short. In general, not all candidates are true k-shredders. To handle this, $\texttt{Shredders}(\cdot,\cdot)$ performs a *pruning* phase by identifying fundamental characteristics of false candidates. To identify false candidates, we formalize some of the key definitions in [6].

▶ **Definition 4.5** (δ_π). *Let π be a path starting from a vertex x. For a vertex $v \in \pi$, we define $\delta_\pi(v)$ as the distance between x and v along path π.*

▶ **Definition 4.6** (Straddle). *Let Γ_1, Γ_2 be two bridges of Π. We say Γ_1* straddles *Γ_2 (or Γ_2 straddles Γ_1) if there exist paths $\pi, \pi' \in \Pi$ such that there exist vertex pairs $(v_1, v_2) \in \pi(\Gamma_1) \times \pi(\Gamma_2)$ satisfying $\delta_\pi(v_1) < \delta_\pi(v_2)$ and $(u_1, u_2) \in \pi'(\Gamma_1) \times \pi'(\Gamma_2)$ satisfying $\delta_{\pi'}(u_1) > \delta_{\pi'}(u_2)$.*

See figure Figure 2 for an example.

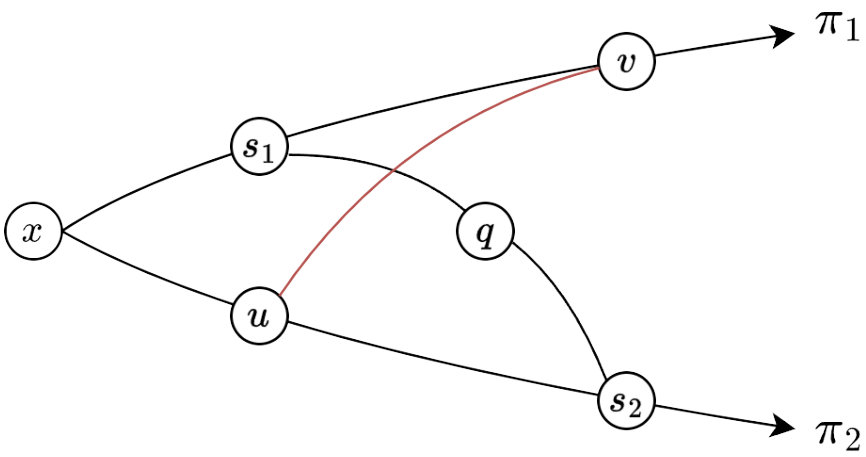


Figure 2 Here we have $\Pi = \{\pi_1, \pi_2\}$. The candidate k-shredder $\{s_1, s_2\}$ is straddled by the edge (u, v) because $\delta_{\pi_2}(u) < \delta_{\pi_2}(s_2)$ and $\delta_{\pi_1}(v) > \delta_{\pi_1}(s_1)$.

It is useful to note that Definition 4.6 is still well-defined even for candidates (i.e. we can use the $\preceq, \succeq$ operators to compare bridges to candidates and candidates to candidates).

Candidate Pruning

The crucial observation is that a candidate S reported by $\texttt{Shredders}(x, y)$ is a k-shredder of G if and only if no bridge of Π straddles S. Hence, to prune false candidates, $\texttt{Shredders}(x, y)$ finds all bridges of Π that are straddled using standard sorting and interval merging. These notions are formalized in the following two lemmas.

▶ **Lemma 4.7.** *Let x and y be two distinct vertices and let Π denote a set of k openly disjoint simple paths from x to y. Let S be a candidate with respect to Π. Then, S is a k-shredder separating x and y if and only if no bridge of Π straddles S.*

▶ **Lemma 4.8.** *Let x be a vertex in G and let Π denote a set of k openly disjoint simple paths starting from x whose sum of lengths over all paths is at most ℓ. Let $\mathcal{C}, \mathcal{B}$ be a set of candidates and a set of bridges of Π, respectively. There exists a deterministic algorithm that, given $(\mathcal{C}, \mathcal{B}, \Pi)$ as input, lists all candidates $S \in \mathcal{C}$ such that S is not straddled by another candidate in $\mathcal{C}$ nor by any bridge in $\mathcal{B}$. The algorithm runs in $\mathcal{O}(k|\mathcal{C}| \log |\mathcal{C}| + \ell + \mathrm{vol}(\mathcal{B}))$ time.*

Challenges

Fix an unbalanced k-shredder S with partition $(C, \mathcal{R})$. Unfortunately, the same probabilistic approach we used for listing balanced k-shredders does not work here. Specifically, if we sample two edges $(x, x'), (y, y')$, the probability that x and y are in different components of $G \setminus S$ can be arbitrarily small. What this dilemma implies is that we must spend at least a linear amount of time just to collect samples. More critically, we must spend a sublinear amount of time processing an *individual* sample to make any meaningful improvement. This time constraint rules out the possibility of calling $\texttt{Shredders}(\cdot, \cdot)$ per sample. Instead, we must develop a *localized* version of $\texttt{Shredders}(\cdot, \cdot)$ that spends time relative to a parameter of our choice, instead of a global value such as m or n.

In detail, consider an unbalanced k-shredder S with partition $(C, \mathcal{R})$. Observe that there must exist a power of two 2^i such that $2^{i-1} < \mathrm{vol}(\mathcal{R}) \leq 2^i$. If we sample $\tilde{\mathcal{O}}(m/2^i)$ edges (x, y), we will sample a vertex $x \in \mathcal{R}$ with high probability due to the classic hitting set lemmas. The goal is to spend only $\tilde{\mathcal{O}}(\mathrm{poly}(k) \cdot 2^i)$ time processing each sample to list S. Suppose that such a local algorithm exists. Although we do not know the exact power of two 2^i, we know that $\mathrm{vol}(\mathcal{R}) < \frac{m}{k}$. Hence, we can simply try all powers of two up to $\frac{m}{k}$. This gives us the near-linear runtime bound:

$$\sum_{i=0}^{\lceil \log \frac{m}{k} \rceil} \tilde{\mathcal{O}}\left(\frac{m}{2^i}\right) \cdot \tilde{\mathcal{O}}(\mathrm{poly}(k) \cdot 2^i) = \sum_{i=0}^{\lceil \log \frac{m}{k} \rceil} \tilde{\mathcal{O}}(\mathrm{poly}(k) \cdot m) = \tilde{\mathcal{O}}(\mathrm{poly}(k) \cdot m).$$

5 Local Techniques

Let S be an unbalanced k-shredder with partition $(C, \mathcal{R})$. Our challenge is to list S in time relative to a parameter of our choice rather than the size of the entire graph. To achieve this, we will implement a local search procedure from a sampled vertex. To formalize the procedure, recall Definition 2.4.

Suppose that we have obtained a tuple (x, ν, Π) that captures S. Let Q denote the component in $\mathcal{R}$ containing x. It is straightforward to see that all remaining components of $\mathcal{R} \setminus \{Q\}$ will also be components of $G \setminus \Pi$. Thus, we may attempt to explore components of $G \setminus \Pi$ and apply Lemma 5.3 (a local version of Lemma 4.7) and Lemma 4.8 to list S. An important distinction from $\texttt{Shredders}(\cdot, \cdot)$ is that in this case, Π is no longer a set of k openly-disjoint simple paths from x to a single vertex y, but rather from x to some vertices in C.

▶ **Lemma 2.5.** *Let G be a graph with vertex connectivity k. Let (x, ν, Π) be a tuple where x is a vertex, ν is a positive integer, and Π is a set of paths. There exists a deterministic algorithm that takes (x, ν, Π) as input and outputs a list $\mathcal{L}$ of k-shredders and one set U such that the following holds. If S is a k-shredder that is captured by (x, ν, Π), then $S \in \mathcal{L}$ or $S = U$. The algorithm runs in $\mathcal{O}(k^2 \nu \log \nu)$ time.*

Our goal is to show that the inner workings of `Shredders`$(\cdot, \cdot)$ can be efficiently translated to our localized setting. We show how to implement this in Algorithm 1. A major caveat concerns the set U. We can think of U as an *unverified set*. Why U exists is a consequence of locality. Essentially, because the algorithm in Lemma 2.5 is time-limited by a volume parameter ν, we may discover a set U but not have enough time to verify whether U is a k-shredder. We first state two helpful lemmas that will provide some understanding of how Algorithm 1 works. These lemmas summarize and translate a large portion of Section 4 in terms of our new setting.

▶ **Fact 5.1.** *Let S be a k-shredder with partition $(C, \mathcal{R})$ captured by (x, ν, Π). For every path $\pi \in \Pi$, we have $|S \cap \pi| = 1$.*

▶ **Lemma 5.2.** *Let S be a k-shredder with partition $(C, \mathcal{R})$ captured by (x, ν, Π). Let Q denote the component in $\mathcal{R}$ containing x. Every component in $\mathcal{R} \setminus \{Q\}$ is a component of $G \setminus \Pi$.*

▶ **Lemma 5.3.** *Let S be a k-shredder with partition $(C, \mathcal{R})$ captured by (x, ν, Π). Then, no bridge of Π straddles S.*

Algorithm Outline

Consider Algorithm 1. At a high level, Algorithm 1 works as follows. Let (x, ν, Π) be a tuple given as input to the algorithm. For each path $\pi \in \Pi$, we can traverse the path from x to the far-most endpoint of π. At a given vertex $u \in \pi$, we can explore the bridges of Π attached to u using a breadth-first search (BFS). While doing so, we maintain a list of bridges of Π. We also maintain a list of candidate k-shredders of Π by checking whether the attachment set of a bridge forms a k-tuple. Among the list of candidate k-shredders, false candidates are then pruned correctly and efficiently via Lemma 5.3 and Lemma 4.8. So far, we have not deviated from `Shredders`$(\cdot, \cdot)$.

The key modification is to impose a restriction on the number of edges we are allowed to explore via BFS. We can explore at most ν edges along each path. Suppose we are processing a vertex u along path π. If the number of edges explored for this path exceeds ν while exploring bridges of Π attached to u, we terminate early and mark u as an *unverified vertex*. Intuitively, the unverified vertex u serves as a boundary of exploration. It signifies that bridges attached to u were not fully explored, so we flag u and treat it with extra caution during a later step. While this may seem dubious, the key is that we are concerned only with k-shredders that are *captured* by (x, ν, Π). Let S be a k-shredder with partition $(C, \mathcal{R})$. For each path $\pi \in \Pi$, if we walk in increasing distance from x and explore attached bridges of Π, observe that it is impossible to visit more than $\text{vol}(\mathcal{R})$ edges prior to reaching a vertex in S. Since $\text{vol}(\mathcal{R}) \leq \nu$, this implies we must either identify S as a candidate k-shredder, or as the unverified set U.

At the end of processing all the paths in Π, we will have obtained a list of bridges of Π, a list of candidate k-shredders of Π, and an *unverified set* consisting of all the unverified vertices. The final step is to black box Lemma 4.8 to prune false candidates among the list of candidate k-shredders.

Algorithm 1 Finding the unbalanced k-shredders with small components containing x.

Input: x - a sample vertex
ν - volume parameter
Π - a set of k openly disjoint paths starting from x
Output: $\mathcal{L}$ - a list of k-shredders captured by (x, ν, Π)
U - an unverified set

$\mathcal{L} \leftarrow \varnothing$
$U \leftarrow \varnothing$
for each path $\pi \in \Pi$ **do**
    reset explored edges and vertices to null
    for each vertex $u \in \pi$ in increasing distance from x **do**
        if u is the far-most endpoint of π **then**
            $U \leftarrow U \cup \{u\}$
            break
        explore all unexplored bridges of Π attached to u with BFS:
            terminate early the moment more than ν edges are explored
        if BFS terminated early **then**
            $U \leftarrow U \cup \{u\}$
            break
        else
            for each bridge Γ explored by BFS **do**
                if the attachment set A of Γ forms a k-tuple of Π **then**
                    $\mathcal{L} \leftarrow \mathcal{L} \cup \{A\}$
if $x \in U$ **then**
    return $(\varnothing, \varnothing)$

prune false k-shredders in $\mathcal{L} \cup \{U\}$ using Lemma 4.8
$F \leftarrow$ the set of far-most endpoints of all paths in Π
if U was not pruned $U \cap F = \varnothing$ **then**
    return $(\mathcal{L}, U)$
else
    return $(\mathcal{L}, \varnothing)$

We state two more lemmas that will be useful in the later sections.

▶ **Lemma 5.4.** *Suppose that Algorithm 1 on input (x, ν, Π) returns an unverified set U such that $U \neq \varnothing$. Then, U is a k-separator. Specifically, let z be the far-most endpoint of an arbitrary path $\pi \in \Pi$. Then, x and z are not connected in $G \setminus U$.*

▶ **Lemma 5.5.** *Suppose that Algorithm 1 returns a nonempty unverified set U on input (x, ν, Π). Let Q denote the component of $G \setminus U$ containing x. There exists a modified version of Algorithm 1 that also computes* $\mathrm{vol}(Q)$ *in $\mathcal{O}(k\nu)$ time.*

6 Resolving Unverified Sets

We now address the case where Algorithm 1 returns an unverified set U on input (x, ν, Π). The difference between U and the list of returned k-shredders $\mathcal{L}$ is that the algorithm did not find a component Q of $G \setminus \Pi$ such that $N(Q) = U$. In some sense, this means that

we "lose" a component of $G \setminus U$. The danger in guessing that U is a k-shredder is that U might only be a k-separator. Imagine that $G \setminus U$ has exactly two components: C_1 and C_2. If $\text{vol}(C_2) \gg \text{vol}(C_1)$, then it becomes quite challenging to determine whether there exists a third component of $G \setminus U$ in $\mathcal{O}(\text{vol}(C_1))$ time. To cope with this uncertainty, we make another structural classification.

▶ **Definition 6.1** (Low-Degree, High-Degree). *Let S be a k-shredder with partition $(C, \mathcal{R})$. Let ν be the unique power of two satisfying $\frac{1}{2}\nu < \text{vol}(\mathcal{R}) \leq \nu$. We say that S has* low-degree *if there exists a vertex $s \in S$ such that $\deg(s) \leq \nu$. Otherwise, we say S has* high-degree.

One aspect of this definition may seem strange: we have specifically described ν as a power of two. This is because in the later sections, we will use geometric sampling to capture k-shredders. The sampling parameters we use will be powers of two. Hence, we have imposed this slightly arbitrary detail on Definition 6.1. For now, all that matters is that $\frac{1}{2}\nu < \text{vol}(\mathcal{R}) \leq \nu$.

Organization

There are two main lemmas for this section. Lemma 6.3 handles low-degree unverified sets and is presented in Section 6.1. Lemma 6.7 handles high-degree unverified sets and is presented in Section 6.2. The idea is that after Algorithm 1 returns an unverified set U, we will use Lemma 6.3 to test whether U is a low-degree k-shredder. If not, we will leave keep U as a potential high-degree k-shredder. After all unverified sets have been returned, we use Algorithm 3 to extract all high-degree k-shredders from the remaining unverified sets.

6.1 Low-Degree

Suppose that U is a low-degree k-shredder with partition $(C, \mathcal{R})$. Suppose that the tuple (x, ν, Π) captures U and Algorithm 1 reports U as an unverified set. Our goal is to design an algorithm that confirms U is a k-shredder in the same time complexity as Algorithm 1 up to $\text{polylog}(n)$ and $\text{poly}(k)$ factors.

Lemma 5.4 gives us two vertices in different components of $G \setminus U$: x and z (where z is the far-most endpoint of any path in Π). Since U is a k-shredder, there must exist a third component of $G \setminus U$ and every vertex in U must be adjacent to a vertex in this third component. Because U is low-degree, there must exist a vertex $u \in U$ with $\deg(u) \leq \nu$. The idea is to scan through the edges adjacent to this low-degree vertex u and find an edge (u, y) such that y is neither connected to x nor z in $G \setminus U$. To make the scanning procedure viable, we need to efficiently answer pairwise connectivity queries in $G \setminus U$. The key ingredient is a result obtained by Kosinas in [17]. Specifically, the following holds.

▶ **Theorem 6.2.** *There exists a deterministic data structure for an undirected graph G on n vertices and m edges with $\mathcal{O}(km \log n)$ preprocessing time that supports the following operations.*

1. *Given a set F of k vertices, perform a data structure update in time $\mathcal{O}(k^4 \log n)$.*
2. *Given a pair of vertices (x, y) return* `true` *if x is connected to y in $G \setminus F$ in time $\mathcal{O}(k)$.*

We defer the method of capturing k-shredders for later. For now, let us assume that the tuple (x, ν, Π) captures a k-shredder U and U is reported as an unverified set by Algorithm 1. We show an auxiliary algorithm that will be helpful in listing low-degree k-shredders.

▶ **Lemma 6.3.** *After preprocessing the input graph using the data structure from Theorem 6.2, there exists a deterministic algorithm that takes in as input (x, ν, Π, U), where U is an unverified set returned by Algorithm 1 on input (x, ν, Π). The algorithm outputs* `true` *if U is a k-shredder and there exists a vertex $u \in U$ such that $\deg(u) \leq \nu$ in $\mathcal{O}(k^4 \log n + k\nu)$ time.*

▶ Remark 6.4. In [20], Long and Saranurak showed the existence of a pairwise connectivity oracle subject to vertex failures with $\mathcal{O}(k^2 n^{o(1)})$ update time and $\mathcal{O}(k)$ query time. Our usage of Kosinas's oracle involves making one update operation with a set of size k and $\mathcal{O}(\nu)$ many connectivity queries. Using Long and Saranurak's version, we can improve the dependency on k in the time complexity of Algorithm 2 to $\mathcal{O}(k^2 n^{o(1)} + k\nu)$ time.

Algorithm Outline

As mentioned above, the idea is to scan through the edges adjacent to u in order to find a vertex y such that y is neither connected to x nor z in $G \setminus U$. We will be utilizing Theorem 6.2 to determine whether a pair of vertices (x, y) are connected in $G \setminus U$. Pseudocode is given in Algorithm 2.

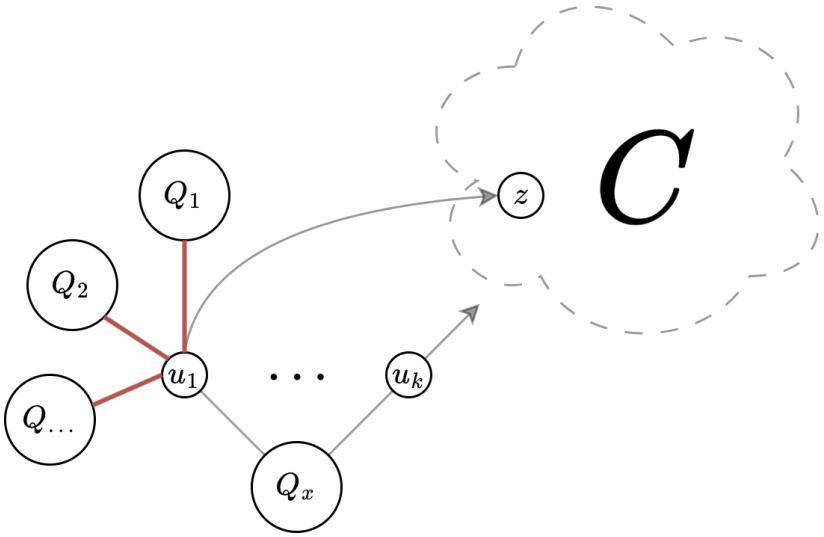


Figure 3 Here the set $U = \{u_1, \dots, u_k\}$ was reported as an unverified set. Suppose there exist a vertex v_1 without loss of generality such that $\deg(u_1) \leq \nu$. Then, we can simply scan through the neighbors of u_1 in $\mathcal{O}(\nu)$ iterations to obtain an edge incident to a component of $G \setminus U$ that is not Q_x nor C.

6.2 High-Degree

It is quite difficult to determine locally whether U is a high-degree k-shredder. We can no longer hope for a low-degree vertex $u \in U$ and scan through its edges to find a third component of $G \setminus U$. To tackle high-degree k-shredders reported as unverified sets, our idea is to ignore them as they are reported and filter them later using one subroutine. Let U be a high-degree k-shredder with partition $(C, \mathcal{R})$. Let ν be the power of two satisfying $\frac{1}{2}\nu < \mathrm{vol}(\mathcal{R}) \leq \nu$. Every vertex $x \in \mathcal{R}$ has $\deg(x) \leq \nu$ because $\mathrm{vol}(\mathcal{R}) \leq \nu$. Furthermore, every vertex $u \in U$ has $\deg(u) > \nu$ because U is high-degree. We can exploit this structure by noticing that U forms a "wall" of high-degree vertices. That is, if we obtain a vertex $x \in \mathcal{R}$, we can use BFS seeded at x to explore a small area of vertices with degree at most ν. The high degree vertices of U would prevent the graph traversal from escaping the component of $G \setminus U$ containing x. At the end of the traversal, the set of explored vertices would compose a component Q of $G \setminus U$ and we can report that $N(Q) = U$ might be a high-degree k-shredder. To obtain a vertex $x \in \mathcal{R}$, we can use the classic hitting set lemmas via random edge sampling.

Algorithm 2 Auxiliary algorithm for low-degree k-shredders.

Input: x - a sample vertex
ν - volume parameter
Π - a set of k openly-disjoint paths starting from x
U - unverified set returned by Algorithm 1 on (x, ν, Π)
Output: `true` if (1) and (2) are satisfied, `false` otherwise
(1) U is a k-shredder
(2) there exists a vertex $u \in U$ such that $\deg(u) \leq \nu$

if all vertices in U have degree greater than ν **then**
  return `false`
$u \leftarrow$ a vertex in U with $\deg(u) \leq \nu$
$\pi \leftarrow$ the path in Π containing u
$z \leftarrow$ the far-most endpoint of π

$\Phi \leftarrow$ the pairwise connectivity oracle from Theorem 6.2 (already preprocessed)
update Φ on vertex failure set U
for each edge (u, y) adjacent to u such that $y \notin U$ **do**
  $q_{x,y} \leftarrow$ connectivity query between x and y in $G \setminus U$
  $q_{y,z} \leftarrow$ connectivity query between y and z in $G \setminus U$
  if $q_{x,y}$ is `false` **and** $q_{y,z}$ is `false` **then**
    return `true`
return `false`

▶ **Lemma 6.5** (Hitting Set Lemma). *Let Q be a set of vertices. Let ν be a positive integer satisfying $\frac{1}{2}\nu < \mathrm{vol}(Q) \leq \nu$. If we independently sample $400\frac{m}{\nu}\log n$ edges (u, v) uniformly at random, we will obtain an edge (u, v) such that $u \in Q$ with probability $1 - n^{-100}$.*

We are not finished yet, as we need to connect this idea with the unverified sets reported by Algorithm 1. Suppose that U was reported by Algorithm 1 on input (x, ν, Π). We have that x is in some component Q of $G \setminus U$. Our goal is to sample a vertex y in a *different* component Q' of $G \setminus U$. As described above, we can exploit the fact that U is a high-degree k-shredder by exploring low-degree vertices in the neighborhood of y to recover Q'. At this point, we have essentially recovered two components of $G \setminus U$: Q and Q'. To make further progress, we present a short, intuitive lemma.

▶ **Lemma 6.6.** *Let U be a k-separator. Let Q, Q' be two distinct components of $G \setminus U$. Then U is a k-shredder if and only if $\mathrm{vol}(Q) + \mathrm{vol}(Q') + |E(U, U)| < m$.*

We are finally ready to handle high-degree k-shredders reported as unverified sets. We present the main result.

▶ **Lemma 6.7.** *There exists a randomized Monte Carlo algorithm that takes as input a list $\mathcal{U}$ of tuples of the form (x, ν, Π, U), where U is an unverified set returned by Algorithm 1 on (x, ν, Π). The algorithm returns a list of k-shredders $\mathcal{L}$ that satisfies the following. If $(x, \nu, \Pi, U) \in \mathcal{U}$ is a k-tuple such that U is a high-degree k-shredder, then $U \in \mathcal{L}$ with probability $1 - n^{-100}$. The algorithm runs in $\mathcal{O}(k^2 m \log^2 n)$ time.*

Algorithm Outline

We summarize the argument made above. Fix a tuple (x, ν, Π, U) in $\mathcal{U}$ such that U is a high-degree k-shredder with partition $(C, \mathcal{R})$. Let ν denote the unique power of two satisfying $\frac{1}{2}\nu < \mathrm{vol}(\mathcal{R}) \leq \nu$. Because U is high-degree, we have for all $u \in U$, $\deg(u) > \nu$. Let Q

denote the component of $G \setminus U$ containing x. Suppose we have obtained a vertex y in a component $Q' \in \mathcal{R} \setminus \{Q\}$. Since all vertices in Q' have degree at most ν, the idea is to explore all vertices connected to y that have degree at most ν. Because $N(Q') = U$ and each vertex in U has degree greater than ν, we will compute Q' as the set of explored vertices. After computing Q', we will have obtained two components of $G \setminus U$: Q and Q'. In order to confirm that U is a k-shredder, all that is left to do is to make sure that Q and Q' are not the only components of $G \setminus U$. Lemma 6.6 implies this can be done by verifying that $\text{vol}(Q) + \text{vol}(Q') + |E(U, U)| < m$. Now, the final step is to find such a vertex y. We use a random sampling procedure for this task. Consider Algorithm 3. We omit the proofs of correctness and time complexity.

Algorithm 3 Extracting all high-degree k-shredders from unverified sets.

Input: $\mathcal{U}$ - a list of tuples of the form (x, ν, Π, U)
Output: $\mathcal{L}$ - a list of k-shredders that satisfies the following:
if $(x, \nu, \Pi, U) \in \mathcal{U}$ is such that U is a high-degree k-shredder,
then $U \in \mathcal{L}$ with probability $1 - n^{-100}$

$\mathcal{L} \leftarrow \varnothing$
for $i \leftarrow 0$ to $\lceil \log\left(\frac{m}{k}\right) \rceil$ **do**
  $\nu' \leftarrow 2^i$
  for $400\frac{m}{\nu'} \log n$ times **do**
    independently sample an edge (y, z) uniformly at random
    $Q' \leftarrow$ set of vertices explored by BFS seeded at y:
      ignore vertices v such that $\deg(v) > \nu'$
      terminate early the moment more than ν' edges are explored
    if BFS terminated early **then**
      skip to next sample
    else if there exists a tuple $(x, \nu, \Pi, U) \in \mathcal{U}$ such that $N(Q') = U$ **then**
      $Q \leftarrow$ component of $G \setminus U$ containing x
      if $x \notin Q'$ **and** $\text{vol}(Q) + \text{vol}(Q') + |E(U, U)| < m$ **then**
        $\mathcal{L} \leftarrow \mathcal{L} \cup \{U\}$
return $\mathcal{L}$

7 Capturing and Listing Unbalanced k-Shredders

In the previous sections, we showed an algorithm that takes as input a tuple (x, ν, Π), and lists all k-shredders that are captured by (x, ν, Π) as well as an unverified set. We then showed how to resolve unverified sets using casework on the structural properties of k-shredders. There is one piece of the puzzle left: the method for capturing unbalanced k-shredders. For this, we will leverage random sampling and recent developments in local flow algorithms as in [10].

7.1 Leveraging Local Flow Algorithms

Let S be an unbalanced k-shredder with partition $(C, \mathcal{R})$. At a high level, we use geometric sampling to obtain a seed vertex $x \in \mathcal{R}$ and a volume parameter ν satisfying $\frac{1}{2}\nu < \text{vol}(\mathcal{R}) \leq \nu$. This obtains two items necessary for capturing S. We are still missing a set of k openly-disjoint paths Π, each starting from x and ending at a vertex in C such that the sum of lengths over all paths is at most $k^2\nu$. This is precisely a core tool developed in [10].

▶ **Definition 7.1** (Vertex Cut). *A vertex cut (L, S, R) of a graph $G = (V, E)$ is a partition of V such that for all vertex pairs $(u, v) \in L \times R$, u is not connected to v in $G \setminus S$.*

▶ **Theorem 7.2** ([10, implicit in Theorem 4.1]). *Let $G = (V, E)$ be an undirected n-vertex m-edge graph with vertex connectivity k. Let (L, S, R) be a vertex cut of G such that $\mathrm{vol}(R) < \frac{m}{k}$. There exists a randomized algorithm `LocalVC`$(\cdot, \cdot)$ that takes as input a pair (x, ν) where x is a vertex in R and ν is a positive integer satisfying $\frac{1}{2}\nu < \mathrm{vol}(R) \leq \nu$. The algorithm outputs a set Π of k openly-disjoint paths such that each path satisfies the following.*
1. *The sum of lengths over all paths in Π is at most $k^2\nu$.*
2. *The path starts from x and ends at a vertex in L.*

The algorithm outputs Π with probability $1/4$ in $\mathcal{O}(k^2\nu)$ time.

Theorem 4.1 of [10] only states that their algorithm returns a vertex cut. But they also construct the set of paths Π. Their algorithm is simple, and we will briefly explain it here. First, we perform the standard vertex-splitting reduction and reduce the problem to finding *directed edge-disjoint* paths instead. To find the first path, we perform any graph search, says DFS, from a vertex x to explore $k\nu$ volume and sample a random endpoint y_1 among all explored edges. Note that y_1 is in L with probability at least $1 - 1/k$ since $\mathrm{vol}(R) \leq \nu$. Then, we reverse the direction of edges on the path from x to y_1 in the DFS tree and obtain a "residual" graph. Then, we repeat the process in the residual graph to construct the next path from x to y_2. After k iterations, the endpoints $y_1, \ldots, y_k$ of these k paths are in L with probability $(1 - 1/k)^k \geq 1/4$. These paths in the residual graphs can be decomposed (like the flow-decomposition) into k directed edge-disjoint paths in the original graph whose total length is $k^2\nu$. These, in turn, correspond k openly vertex-disjoint paths by the standard reduction in the beginning.

Notice how the output of the algorithm described in Theorem 7.2 directly corresponds to Definition 2.4. Let R denote the union of all components in $\mathcal{R}$. Notice that (C, S, R) forms a vertex cut such that $\mathrm{vol}(R) < \frac{m}{k}$. The idea is to obtain a seed vertex $x \in R$ using a linear amount of random samples. For each sample, we can directly apply `LocalVC`$(\cdot, \cdot)$ to obtain the desired set of paths. Furthermore, we can boost the success rate of the algorithm by repeating it a polylogarithmic number of times. In the following section, we formalize this idea.

7.2 The Algorithm for Unbalanced k-Shredders

The main result is stated below.

▶ **Lemma 7.3.** *There exists a randomized algorithm that takes as input $G = (V, E)$, an n-vertex m-edge undirected graph with vertex connectivity k. The algorithm correctly lists all unbalanced k-shredders of G with probability $1 - n^{-98}$ in $\mathcal{O}(k^4 m \log^4 n)$ time.*

We first give a high level outline for listing unbalanced k-shredders. Let S be an unbalanced k-shredder with partition $(C, \mathcal{R})$. With geometric sampling, by the hitting set lemma we will sample a vertex $x \in \mathcal{R}$ with a volume parameter ν that satisfies $\frac{1}{2}\nu < \mathrm{vol}(\mathcal{R}) \leq \nu$. Then, we use Theorem 7.2 to obtain a set of k openly-disjoint paths Π, each starting from x and ending at a vertex in C such that the sum of lengths over all paths is at most $k^2\nu$. We now have a tuple (x, ν, Π) that captures S. After capturing S, we call Algorithm 1 to list S as a k-shredder or an unverified set. In the latter case, we can verify whether S is a low-degree k-shredder using Algorithm 2. If this verification step fails, S must be a high-degree k-shredder. In this case, we can add S to a global list of unverified sets. This list will be processed after all unbalanced k-shredders have been captured. Lastly, we can extract all high-degree k-shredders from the list of unverified sets using Algorithm 3. Pseudocode describing this process is given in Algorithm 4.

Algorithm 4 Listing all unbalanced k-shredders of a graph.

Input: $G = (V, E)$ - an undirected k-vertex-connected graph
Output: $\mathcal{L}$ - a list containing all unbalanced k-shredders of G

$\mathcal{L} \leftarrow \varnothing$ ▷ *list of k-shredders*
$\mathcal{U} \leftarrow \varnothing$ ▷ *unverified sets*
$\Phi \leftarrow$ initialize a pairwise connectivity oracle as in [17]

for $i \leftarrow 0$ to $\lceil \log \left(\frac{m}{k}\right) \rceil$ **do**
  $\nu \leftarrow 2^i$
  for $400 \frac{m}{\nu} \log n$ times **do**
    independently sample an edge (x, y) uniformly at random
    for $100 \log n$ times **do**
      $\Pi \leftarrow \texttt{LocalVC}(x, \nu)$ as in [10]
      $(\mathcal{L}_{local}, U) \leftarrow$ call Algorithm 1 on input (x, ν, Π)
      $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_{local}$
      if $U \neq \varnothing$ **then**
        call Algorithm 2 on input (x, ν, Π, U)
        if Algorithm 2 returned `true` **then**
          $\mathcal{L} \leftarrow \mathcal{L} \cup \{U\}$
        else
          $\mathcal{U} \leftarrow \mathcal{U} \cup \{(x, \nu, \Pi, U)\}$

$\mathcal{L}_{high-degree} \leftarrow$ call Algorithm 3 on input $\mathcal{U}$
$\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_{high-degree}$
return $\mathcal{L}$

8 Listing All k-Shredders

At last, we are ready to present the algorithm for listing all k-shredders. See Algorithm 5.

▶ **Lemma 8.1.** *Let $G = (V, E)$ be an n-vertex m-edge undirected graph with vertex connectivity k. There exists a randomized algorithm that takes G as input and correctly lists all k-shredders of G with probability $1 - n^{-97}$ in $\mathcal{O}(k^4 m \log^4 n)$ time.*

The idea is that we can classify any k-shredder as either balanced or unbalanced. We have described how to use random edge sampling to handle the former case, and Algorithm 4 handles the latter case. Because the algorithm for listing balanced k-shredders is quite trivial, we have omitted its pseudocode. Given Lemma 8.1, Theorem 1.1 follows by the standard sparsification [22].

9 Most Shattering Min-Cut

We have presented a randomized near-linear time algorithm that lists all k-shredders with high probability, but we have not yet shown how to count the number of components each k-shredder separates. Counting components proves to be a bit trickier than it was in [6]. What we will do is modify our algorithms such that whenever we list a k-shredder S, we also return the number of components of $G \setminus S$. We will do this for Algorithm 1, Algorithm 2, and Algorithm 3. Note that if there are no k-shredders of G, we can simply return a k-separator of G. This can be done in $\tilde{\mathcal{O}}(m + nk^3)$ time as shown in [10].

Algorithm 5 Listing all k-shredders of a graph.

Input: $G = (V, E)$ - an undirected k-vertex-connected graph
Output: $\mathcal{L}$ - a list of all k-shredders of G

1: $\mathcal{L} \leftarrow \varnothing$

2: $\mathcal{L}_{balanced} \leftarrow$ edge sampling and `Shredders`$(\cdot, \cdot)$
3: $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_{balanced}$

4: $\mathcal{L}_{unbalanced} \leftarrow$ call Algorithm 4 on input G
5: $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_{unbalanced}$

6: **return** $\mathcal{L}$

9.1 Counting Components in the Local Algorithm

At a high level, we can modify Algorithm 1 as follows. Suppose Algorithm 1 identifies a candidate k-shredder S on line 16. By line 13, all bridges of Π attached to some vertex $s \in S$ must have been explored. The modification is as follows. We can keep a dictionary M whose keys are candidate k-shredders (in k-tuple form) mapped to nonnegative integers initialized to zero. For each bridge Γ of Π attached to s, if Γ is a component of $G \setminus \Pi$ such that $N(\Gamma) = S$, then we increment $M[S]$. We claim that if $S \in \mathcal{L}$ is a k-shredder captured by (x, ν, Π), then $M[S] + 2$ is the number of components of $G \setminus S$.

▶ **Lemma 9.1.** *Suppose that Algorithm 1 returns $(\mathcal{L}, U)$ on input (x, ν, Π). There exists a modified version of Algorithm 1 such that for each k-shredder $S \in \mathcal{L}$ that is captured by (x, ν, Π), the modified version also computes the number of components of $G \setminus S$. The modification requires $\mathcal{O}(k^2\nu)$ additional time, subsumed by the running time of Algorithm 1.*

9.2 Counting Components for Low-Degree Unverified Sets

If Algorithm 2 returned `true` on input (x, ν, Π, U), then U is a k-shredder with a vertex $u \in U$ such that $\deg(u) \leq \nu$. Notice that all components of $G \setminus U$ must contain a vertex that is adjacent to u. Immediately, we have that the number of components of $G \setminus U$ is upper bounded by ν, as $\deg(u) \leq \nu$. However, looking at two arbitrary edges $(u, v_1), (u, v_2)$ adjacent to u, it is unclear whether v_1 and v_2 are in the same component of $G \setminus U$. Although we can query whether v_1 and v_2 are connected in $G \setminus U$ in $\mathcal{O}(k)$ time, we cannot afford to make these queries for all pairs of vertices in $N(u)$. Such a procedure would require us to make $\mathcal{O}(\nu^2)$ many queries, which is too costly. To solve this issue, we use a convenient property of DFS trees to bypass making $\mathcal{O}(\nu^2)$ connectivity queries. Much of the machinery was inspired by [17], so we omit the details.

▶ **Lemma 9.2.** *Suppose that Algorithm 2 returns `true` on input (x, ν, Π, U). There exists a modified version of Algorithm 2 that also computes the number of components of $G \setminus U$. The modification requires $O(k^2\nu)$ additional time, subsumed by the running time of Algorithm 2.*

9.3 Counting Components for High-Degree Unverified Sets

Recall from Section 6.2 that the key observation is that S forms a wall of high-degree vertices. We exploited this by sampling vertices in $\mathcal{R}$ and exploring regions of vertices with low degree. The idea is that S will restrict the graph exploration within one component of $\mathcal{R}$, and we can

recover that component. We can actually extend this idea a bit further to count the number of components in $\mathcal{R}$. For all components $Q \in \mathcal{R}$, our algorithm will eventually sample a vertex $x \in Q$. By exploring low-degree vertices, we will recover Q. We can verify that $N(Q)$ is a k-shredder as by now we have listed all of them. Hence, we will eventually count all components of $G \setminus S$ throughout the sampling and exploring procedure.

▶ **Lemma 9.3.** *Suppose that Algorithm 3 returns a list $\mathcal{L}$ of k-shredders on input $\mathcal{U}$. There exists a modified version of Algorithm 3 such that for each k-shredder $S \in \mathcal{L}$, the modified version also computes the number of components of $G \setminus S$. The modification requires $\mathcal{O}(m \log^2 n)$ additional time, subsumed by the time complexity of Algorithm 3.*

Given Lemmas 9.1–9.3, we can compute a most shattering minimum-cut with high probability in $O(k^4 m \log^4 n)$ time and, hence, Theorem 1.2 follows by the standard sparsification [22].

References

1 Michael Becker, W. Degenhardt, Jürgen Doenhardt, Stefan Hertel, Gerd Kaninke, W. Kerber, Kurt Mehlhorn, Stefan Näher, Hans Rohnert, and Thomas Winter. A probabilistic algorithm for vertex connectivity of graphs. *Inf. Process. Lett.*, 15(3):135–136, 1982.

2 Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. Distributed connectivity decomposition. In *PODC*, pages 156–165. ACM, 2014.

3 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time, 2022. `arXiv:2203.00671`.

4 Joseph Cheriyan and John H. Reif. Directed s-t numberings, rubber bands, and testing digraph k-vertex connectivity. *Combinatorica*, 14(4):435–451, 1994. Announced at SODA'92.

5 Joseph Cheriyan and Ramakrishna Thurimella. Algorithms for parallel k-vertex connectivity and sparse certificates (extended abstract). In *STOC*, pages 391–401. ACM, 1991.

6 Joseph Cheriyan and Ramakrishna Thurimella. Fast algorithms for k-shredders and k-node connectivity augmentation. *Journal of Algorithms*, 33(1):15–50, 1999. `doi:10.1006/jagm.1999.1040`.

7 Abdol-Hossein Esfahanian and S. Louis Hakimi. On computing the connectivities of graphs and digraphs. *Networks*, 14(2):355–366, 1984.

8 Shimon Even. An algorithm for determining whether the connectivity of a graph is at least k. *SIAM J. Comput.*, 4(3):393–396, 1975.

9 Shimon Even and Robert Endre Tarjan. Network flow and testing graph connectivity. *SIAM J. Comput.*, 4(4):507–518, 1975.

10 Sebastian Forster, Danupon Nanongkai, Thatchaphol Saranurak, Liu Yang, and Sorrachai Yingchareonthawornchai. Computing and testing small connectivity in near-linear time and queries via fast local cut algorithms. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2046–2065. SIAM, 2020.

11 Harold N. Gabow. Using expander graphs to find vertex connectivity. *J. ACM*, 53(5):800–844, 2006. Announced at FOCS'00.

12 Zvi Galil. Finding the vertex connectivity of graphs. *SIAM J. Comput.*, 9(1):197–199, 1980.

13 Monika Rauch Henzinger. A static 2-approximation algorithm for vertex connectivity and incremental approximation algorithms for edge and vertex connectivity. *J. Algorithms*, 24(1):194–220, 1997.

14 Monika Rauch Henzinger, Satish Rao, and Harold N. Gabow. Computing vertex connectivity: New bounds from old techniques. *J. Algorithms*, 34(2):222–250, 2000. Announced at FOCS'96.

15 Tibor Jordán. On the number of shredders. *Journal of Graph Theory*, 31(3):195–200, 1999. `doi:10.1002/(SICI)1097-0118(199907)31:3<195::AID-JGT4>3.0.CO;2-E`.

16 D Kleitman. Methods for investigating connectivity of large graphs. *IEEE Transactions on Circuit Theory*, 16(2):232–233, 1969.

17 Evangelos Kosinas. Connectivity Queries Under Vertex Failures: Not Optimal, but Practical. In *31st Annual European Symposium on Algorithms (ESA 2023)*, volume 274 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 75:1–75:13, 2023. doi:10.4230/LIPIcs.ESA.2023.75.

18 Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Vertex connectivity in poly-logarithmic max-flows. *CoRR*, abs/2104.00104, 2021. arXiv:2104.00104.

19 Nathan Linial, László Lovász, and Avi Wigderson. Rubber bands, convex embeddings and graph connectivity. *Combinatorica*, 8(1):91–102, 1988. Announced at FOCS'86.

20 Yaowei Long and Thatchaphol Saranurak. Near-optimal deterministic vertex-failure connectivity oracles. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1002–1010, 2022. doi:10.1109/FOCS54457.2022.00098.

21 David W. Matula. Determining edge connectivity in o(nm). In *FOCS*, pages 249–251. IEEE Computer Society, 1987.

22 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of ak-connected graph. *Algorithmica*, 7(1-6):583–596, 1992.

23 Danupon Nanongkai, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Breaking quadratic time for small vertex connectivity and an approximation scheme. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 241–252, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316394.

24 Seth Pettie and Longhui Yin. The structure of minimum vertex cuts. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 105:1–105:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.105.

25 VD Podderyugin. An algorithm for finding the edge connectivity of graphs. *Vopr. Kibern*, 2:136, 1973.

26 Thatchaphol Saranurak and Sorrachai Yingchareonthawornchai. Deterministic small vertex connectivity in almost linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 789–800. IEEE Computer Society, 2022. doi:10.1109/FOCS54457.2022.00080.

Satisfiability to Coverage in Presence of Fairness, Matroid, and Global Constraints

Tanmay Inamdar
Indian Institute of Technology Jodhpur, India

Pallavi Jain
Indian Institute of Technology Jodhpur, India

Daniel Lokshtanov
University of California Santa Barbara, CA, USA

Abhishek Sahu
National Institute of Science, Education and Research Bhubaneswar, HBNI, India

Saket Saurabh
The Institute of Mathematical Sciences, HBNI, Chennai, India
University of Bergen, Norway

Anannya Upasana[1]
The Institute of Mathematical Sciences, HBNI, Chennai, India

Abstract

In the MaxSAT with Cardinality Constraint problem (CC-MaxSAT), we are given a CNF-formula Φ, and a positive integer k, and the goal is to find an assignment β with at most k variables set to true (also called a weight k-assignment) such that the number of clauses satisfied by β is maximized. Maximum Coverage can be seen as a special case of CC-MaxSat, where the formula Φ is monotone, i.e., does not contain any negative literals. CC-MaxSat and Maximum Coverage are extremely well-studied problems in the approximation algorithms as well as the parameterized complexity literature.

Our first conceptual contribution is that CC-MaxSat and Maximum Coverage are equivalent to each other in the context of FPT-Approximation parameterized by k (here, the approximation is in terms of the number of clauses satisfied/elements covered). In particular, we give a randomized reduction from CC-MaxSat to Maximum Coverage running in time $\mathcal{O}(1/\epsilon)^k \cdot (m+n)^{\mathcal{O}(1)}$ that preserves the approximation guarantee up to a factor of $(1-\epsilon)$. Furthermore, this reduction also works in the presence of "fairness" constraints on the satisfied clauses, as well as matroid constraints on the set of variables that are assigned true. Here, the "fairness" constraints are modeled by partitioning the clauses of the formula Φ into r different colors, and the goal is to find an assignment that satisfies at least t_j clauses of each color $1 \leq j \leq r$.

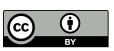
Armed with this reduction, we focus on designing FPT-Approximation schemes (FPT-ASes) for Maximum Coverage and its generalizations. Our algorithms are based on a novel combination of a variety of ideas, including a carefully designed probability distribution that exploits sparse coverage functions. These algorithms substantially generalize the results in Jain et al. [SODA 2023] for CC-MaxSat and Maximum Coverage for $K_{d,d}$-free set systems (i.e., no d sets share d elements), as well as a recent FPT-AS for Matroid Constrained Maximum Coverage by Sellier [ESA 2023] for frequency-d set systems.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Partial Vertex Cover, Max SAT, FPT Approximation, Matroids

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.88

[1] corresponding author



51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 88; pp. 88:1–88:18

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2403.07328

Funding *Tanmay Inamdar*: Part of this work was done when the author was affiliated with the University of Bergen, Norway and was funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416).
Pallavi Jain: The author acknowledges the support from SERB-SUPRA (grant number SPR/2021/000860) and ITJ Seed Grant (grant number I/SEED/PJ/20210119).
Daniel Lokshtanov: The author is supported by NSF award CCF-2008838.
Saket Saurabh: The author is supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416); and he also acknowledges the support of Swarnajayanti Fellowship grant DST/SJF/MSA-01/2017-18.

1 Introduction

Two problems that have gained considerable attention from the perspective of Parameterized Approximation [11] are the classical MaxSAT with cardinality constraint (CC-MaxSat) problem and its monotone version, the Maximum Coverage problem. In the CC-MaxSat problem, we are given a CNF-formula Φ over m clauses and n variables, and a positive integer k, and the objective is to find a weight k assignment that maximizes the number of satisfied clauses. We use $\mathsf{var}(\Phi)$ and $\mathsf{cla}(\Phi)$ to denote the set of variables and clauses in Φ, respectively. An *assignment* to a CNF-formula Φ is a function $\beta : \mathsf{var}(\Phi) \to \{0, 1\}$. The *weight* of an assignment β is the number of variables that have been assigned 1.

The classical Maximum Coverage problem is a special case of the CC-MaxSat problem. Indeed, it is a monotone variant of CC-MaxSat, where negated literals are not allowed. An input to the Maximum Coverage problem consists of a family of m sets, $\mathcal{F}$, over a universe U of size n, and an integer k, and the goal is to find a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size k such that the number of elements *covered* (belongs to some set in $\mathcal{F}'$) by $\mathcal{F}'$ is maximized. Observe that when the goal is to cover every element in U, the Maximum Coverage problem corresponds to Set Cover. A natural question that has guided research on these problems is whether CC-MaxSat or Maximum Coverage admits an algorithm with running time $f(k)n^{\mathcal{O}(1)}$? That is, whether CC-MaxSat or Maximum Coverage is fixed parameter tractable (FPT) with solution size k? Unfortunately, these problems are W[2]-hard [9]. That is, we do not expect these problems to admit an algorithm with running time $f(k)n^{\mathcal{O}(1)}$. This negative result sets the platform for studying these problems from the viewpoint of Parameterized Approximation [11]. It is well known that both CC-MaxSat and Maximum Coverage admit a polynomial time $(1 - \frac{1}{e})$-approximation algorithm [26], which is in fact optimal. [10]. So, in the realm of Parameterized Approximation, we ask does there exist an $\epsilon > 0$, such that CC-MaxSat or Maximum Coverage admits an approximation algorithm with factor $(1 - \frac{1}{e} + \epsilon)$ and runs in time $f(k, \epsilon)n^{\mathcal{O}(1)}$. While there has been a lot of work on Maximum Coverage [17, 20, 25, 15, 24], Jain et al. [17] studied CC-MaxSat and designed a standalone algorithm for the problem. Our first result, a bit of a surprise to us, shows that in the world of Parameterized Approximation, CC-MaxSat and Maximum Coverage are *"equivalent"*.

▶ **Theorem 1.1** (Informal). *Let $\epsilon > 0$. There is a polynomial time randomized algorithm that given an instance (Φ, k) of CC-MaxSat produces an instance $(U, \mathcal{F}, k)$ of* Maximum Coverage *such that the following holds with probability $\frac{1}{2}(\frac{\epsilon}{2})^k$. Given a $(1 - \epsilon)\mathsf{OPT}_{\mathsf{cov}}$ solution to $(U, \mathcal{F}, k)$ we can obtain a $(1-\epsilon)\mathsf{OPT}_{\mathsf{sat}}$ solution to (Φ, k) in polynomial time. Here, $\mathsf{OPT}_{\mathsf{cov}}$ ($\mathsf{OPT}_{\mathsf{sat}}$) denotes the value of the maximum number of covered elements (satisfied clauses) by a k-sized family of subsets (weight k assignment).*

Theorem 1.1 allows us to focus on Maximum Coverage, rather than CC-MaxSat, at the expense of $\epsilon^{-\mathcal{O}(k)}$ in the running time. Further, there is no assumption on the input formulas in Theorem 1.1. This reduction immediately implies faster algorithms for CC-MaxSat by utilizing the known good algorithms for Maximum Coverage [17, 20, 25, 15, 24]. The Maximum Coverage problem has been generalized in several directions by adding either fairness constraints or asking our solution to be an independent set of a matroid. In what follows, we take a closer look at progresses on Maximum Coverage and its generalizations and then design algorithms that generalize and unify all the known results for CC-MaxSat and Maximum Coverage.

1.1 Tractability Boundaries for Maximum Coverage

Cohen-Addad et al. [8] studied Maximum Coverage and showed that there is no $\epsilon > 0$, such that Maximum Coverage admits an approximation algorithm with factor $(1 - \frac{1}{e} + \epsilon)$ and runs in time $f(k, \epsilon)(m + n)^{\mathcal{O}(1)}$ [2]. Later, this was also studied by Manurangsi [20], who obtained the following strengthening over [8]: for any constant $\epsilon > 0$ and any function h, assuming Gap-ETH, no $h(k)(n + m)^{o(k)}$ time algorithm can approximate Maximum Coverage with n elements and m sets to within a factor $(1 - \frac{1}{e} + \epsilon)$, even with a promise that there exist k sets that fully cover the whole universe. This negative result sets the contour for possible positive results. In particular, if we hope for an FPT algorithm that improves over a factor $(1 - \frac{1}{e})$ then we must assume some additional structure on the input families. This automatically leads to the families wherein each set has bounded size, or each element appears in bounded sets which was considered earlier.

Skowron and Faliszewski [25] showed that, if we are working on set families, such that each element in U appears in at most p sets, then there exists an algorithm, that given an $\epsilon > 0$, runs in time $(\frac{p}{\epsilon})^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ and returns a subfamily $\mathcal{F}'$ of size k that is a $(1-\epsilon)$-approximation. These kind of FPT-approximation algorithms are called FPT-approximation Schemes (FPT-ASes). For $p = 2$, Manurangsi [20] independently obtained a similar result. Jain et al. [17] generalized these two settings by looking at $K_{d,d}$-free set systems (i.e., no d sets share d elements). They also considered $K_{d,d}$-free formulas (that is, the clause-variable incidence bipartite graph of the formula excludes $K_{d,d}$ as an induced subgraph). They showed that for every $\epsilon > 0$, there exists an algorithm for $K_{d,d}$-free formulas with approximation ratio $(1 - \epsilon)$ and running in time $2^{\mathcal{O}((\frac{dk}{\epsilon})^d)}(n + m)^{\mathcal{O}(1)}$. For, Maximum Coverage on $K_{d,d}$-free set families, they obtain an FPT-AS with running time $(\frac{dk}{\epsilon})^{\mathcal{O}(dk)} n^{\mathcal{O}(1)}$. Using these results together with Theorem 1.1 we get the following.

▶ **Corollary 1.2.** *Let $\epsilon > 0$. Then, CC-MaxSat admits a randomized* FPT-AS *with running time $(\frac{dk}{\epsilon})^{\mathcal{O}(dk)} n^{\mathcal{O}(1)}$ on $K_{d,d}$-free formulas. Furthermore, if the size of clauses is bounded by p or every variable appears in at most p clauses then CC-MaxSat admits randomized* FPT-AS *with running time $(\frac{p}{\epsilon})^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$. Both results hold with constant probability.*

We note that the deterministic version of the first result in the corollary was given in [21] in a recent independent work.

Corollary 1.2 follows by utilizing Theorem 1.1 and repurposing the known results about Maximum Coverage ([17, 5, 25, 20]). We will return to the case of $K_{d,d}$-free set systems later. Apart from extending the classes of set families where Maximum Coverage admits FPT-ASes, the study on the Maximum Coverage problem has been extended in many directions.

[2] Throughout the paper, the approximation factor will refer to the number of elements covered/number of satisfied clauses, unless explicitly stated otherwise

1.1.1 Matroid Constraints

Note that Maximum Coverage is a special case of submodular function maximization subject to a cardinality constraint. In the latter problem, we are given (an oracle access to) a submodular function $f : 2^V \to \mathbb{R}_{\geq 0}$ [3], and the goal is to find a subset $U \subseteq V$ that maximizes $f(U)$ over all subsets of size at most k. Indeed, coverage functions are submodular and monotone (i.e., adding more sets cannot decrease the number of elements covered). There has been a plethora of work on monotone submodular maximization subject to cardinality constraints, starting from Wolsey [27]. In a further generalization, we are interested in monotone submodular maximization subject to a matroid constraint – in this setting, we are given a matroid $\mathcal{M} = (U, \mathcal{I})$ [4] via an *independence oracle*, i.e., an algorithm that answers queries of the form "Is $P \in \mathcal{I}$?" for any $P \subseteq U$ in one step, and we want to find an independent set $S \in I$ that maximizes $f(S)$. Note here that a uniform matroid of rank k [5] exactly captures the cardinality constraint. Calinescu et al. [6] gave an optimal $(1 - 1/e)$-approximation.

More recently, Huang and Sellier [15] and Sellier [24] studied the problem of maximizing a coverage function subject to a matroid constraint, called Matroid Constrained Maximum Coverage. In this problem, which we call M-MaxCov (M for "matroid" constraint), we are given a set system $(U, \mathcal{F})$ and a matroid $\mathcal{M} = (\mathcal{F}, I)$ of rank k, and the goal is to find a subset $\mathcal{F}' \subseteq \mathcal{F}$ such that $\mathcal{F}' \in I$ and $\mathcal{F}'$ maximizes the number of elements covered. Note that M-MaxCov is a generalization of Maximum Coverage. In the latter paper, Sellier [24] designed an FPT-AS for M-MaxCov, running in time $(d/\epsilon)^{\mathcal{O}(k)} \cdot (m+n)^{\mathcal{O}(1)}$ for frequency-d set systems. Note that this result generalizes that of [25, 20] from a uniform matroid consraint to an arbitrary matroid constraint of rank k.

Analogous to M-MaxCov, one can define a matroid constrained version of CC-MaxSat, called M-MaxSAT. In this problem, we are given a CNF-SAT formula Φ and a matroid $\mathcal{M}$ of rank k on the set of variables. The goal is to find an assignment that satisfies the maximum number of clauses, with the restriction that, the set of variables assigned 1 must be an independent set in $\mathcal{M}$. Note that M-MaxSAT generalizes M-MaxCov as well as CC-MaxSat. We obtain the following result for M-MaxSAT, by combining the results on a variant of Theorem 1.1 with the corresponding result on M-MaxCov.

▶ **Theorem 1.3.** *There exists an* FPT-AS *for M-MaxSAT parameterized by k, d, and ϵ, on d-CNF formulas, where k denotes the rank of the given matroid.*

1.1.2 Fairness or Multiple Coverage Constraints

Now we consider an orthogonal generalization of Maximum Coverage. Note that an optimal solution for Maximum Coverage may leave many elements uncovered. However, such a solution may be deemed *unfair* if the elements are divided into multiple colors (representing, say, people of different demographic groups), and the set uncovered elements are biased against a specific color. To address these constraints, the following generalization of Maximum Coverage, which we call F-MaxCov (F stands for "fair"), has been studied in the literature. Here, we are given a set system $(U, \mathcal{F})$, a coloring function $\chi : U \to [r]$, a

[3] $f : 2^V \to \mathbb{R}$ is submodular if it satisfies $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ for all $A, B \subseteq V$

[4] Recall that a matroid is a pair $\mathcal{M} = (U, \mathcal{I})$, where U is the ground set, and $\mathcal{I}$ is a family of subsets of U satisfying the following three axioms: (i) $\emptyset \in \mathcal{I}$, (ii) If $A \in \mathcal{I}$, then $B \in \mathcal{I}$ for all subsets $B \subseteq A$, and (iii) for any $A, B \in \mathcal{I}$ with $|B| > |A|$, then there exists an element $e \in B \setminus A$ such that $A \cup \{e\} \in \mathcal{I}$.

[5] Rank of a matroid is equal to the maximum size of any independent set in the matroid.

coverage requirement function $t : [r] \to \mathbb{N}$, and an integer k; and the goal is to find a subset $\mathcal{F}' \subseteq \mathcal{F}$ of size at most k such that, for each $i \in [r]$, the union of elements in $\mathcal{F}'$ is at least $t(i)$ (or t_i).

Since F-MaxCov is a generalisation of Maximum Coverage, it inherits all the lower bounds known for Maximum Coverage. Furthermore, we can mimic the algorithm for Maximum Coverage (Partial Set Cover) parameterized by t (where you want to cover at least t elements with k sets) [5] to obtain an algorithm for Partition Maximum Coverage parameterized by $\sum_{j\in[r]} t_j$. However, the problem is NP-hard even when $t_j \leq 1$, $j \in [r]$, via a simple reduction from Set Cover.

F-MaxCov has been studied under multiple names in the approximation algorithms literature; however much of the focus has been on approximating the *size* of the solution, rather than the coverage. Notable exception include Chekuri et al. [7] who gave a "bicriteria" approximation, that outputs a solution of size at most $\mathcal{O}(\log r/\epsilon)$ times the optimal size, and covers at least $(1 - 1/e - \epsilon)$ fraction of the required coverage of each color. Very recently, Bandyapadhyay et al. [3] recently designed an FPT-AS for F-MaxCov for the set systems of frequency 2, running in time $2^{\mathcal{O}(\frac{rk^2 \log k}{\epsilon})} \cdot (m+n)^{\mathcal{O}(1)}$. We obtain the following result on F-MaxCov.

▶ **Theorem 1.4.** *There exists a randomized* FPT-AS *for F-MaxCov running in time* $\left(dr\left(\frac{\log k}{\epsilon}\right)^r\right)^{\mathcal{O}(k)} \cdot (m+n)^{\mathcal{O}(1)}$*, on set systems with frequency bounded by* d*.*

Note that this result generalizes the result of [3] to frequency-d set systems, and in the case of $d = 2$, our running time is faster than that of [3] (albeit our algorithm is randomized).

One can also define *fair* version of CC-MaxSat in an analogous way, which we call F-MaxSAT. In this problem, we are given a CNF-formula Φ, a coloring function $\chi : \mathsf{cla}(\Phi) \to [r]$, a coverage demand function $t : [r] \to \mathbb{N}$, and an integer k. The goal is to find a weight-k assignment that satisfies at least $t(j)$ (also denoted as t_j) clauses of each color $j \in [r]$. By combining Theorem 1.4 with a slightly more general version of the reduction theorem (Theorem 1.1) also yields FPT-AS for F-MaxSAT with a similar running time.

1.2 Our New Problem: Combining Matroid and Fairness Constraints

As discussed in the previous subsections, Maximum Coverage has been generalized in two orthogonal directions, namely, matroid constraints on the sets chosen in the solution, and fairness constraints on the elements covered by the solution. Although the corresponding variants of CC-MaxSat have not been studied in the literature, we mentioned that our techniques readily imply FPT-ASes for these problems for many "sparse" formulas. Given this, the following natural question arises.

> Can we find good approximations for the variants of CC-MaxSat (resp. Maximum Coverage) that combines the two orthogonal generalizations, namely, matroid constraint on the variables assigned 1, and fairness constraints on the satisfied clauses (resp. matroid constraint on the sets chosen in the solution, and fairness constraints on the elements covered)?

In the following, we formally define the common generalization of M-MaxSAT and F-MaxSAT, which we call (M, F)-MaxSAT.

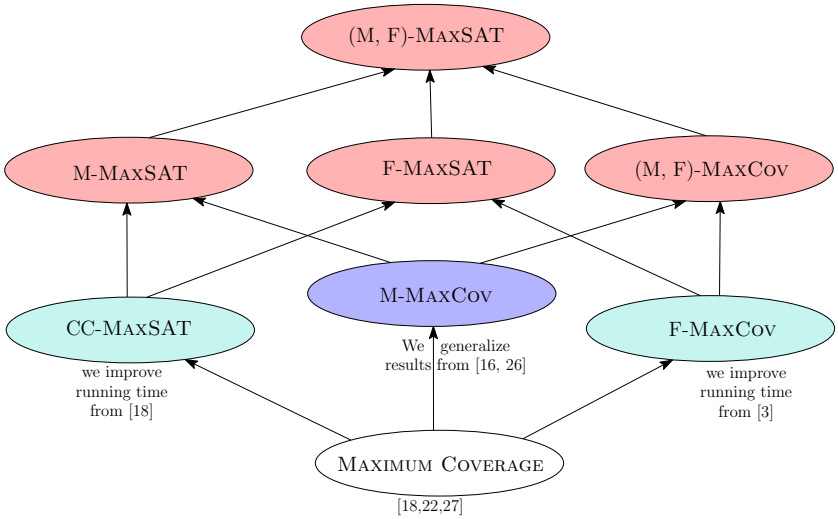


Figure 1 If there is an arrow of the form $A \to B$, then problem B generalizes problem A. FPT-ASes for the problems in red bubbles are not known in the literature, and we study in this paper. For all the other problems FPT-ASes are known in the literature for some cases. This paper improves the results in cyan and blue.

(M, F)-MaxSAT
Input. A CNF-SAT formula Φ where the clauses $\mathsf{cla}(\Phi)$ of Φ are partitioned into r colors. Each color $j \in [r]$ has an associated demand t_j. Additionally, we are provided the independence oracle to a matroid $\mathcal{M} = (\mathsf{var}(\Phi), I)$ of rank k.
Question. Does there exist an assignment $\Psi : \mathsf{var}(\Phi) \to \{0, 1\}$, such that
- The number of clauses satisfied by Ψ of color j is at least t_j, for each $j \in [r]$,
- The set of variables assigned 1 must be independent in $\mathcal{M}$, i.e., $\Psi^{-1}(1) \in I$.

In the special case where the CNF-SAT formula is monotone (i.e., does not contain negated literals), we obtain (M, F)-MaxCov, which generalizes all the variants of Maximum Coverage discussed earlier. We obtain the following result for (M, F)-MaxCov.

▶ **Theorem 1.5.** *There exists a randomized* FPT-AS *for (M, F)-MaxCov on set systems with maximum frequency d, that runs in time $\left(\frac{d \log k}{\epsilon}\right)^{\mathcal{O}(kr)} \cdot (m+n)^{\mathcal{O}(1)}$ and returns a $(1-\epsilon)$-approximation with at least a constant probability.*

Finally, by reducing (M, F)-MaxSAT on d-CNF formulas to (M, F)-MaxCov with frequency d set systems, using the randomized reduction, and then using the results of Theorem 1.5, we obtain our most general result, as follows.

▶ **Theorem 1.6.** *There exists a randomized* FPT-AS *for (M, F)-MaxSAT on d-CNF formulas, that runs in time $\left(\frac{d \log k}{\epsilon}\right)^{\mathcal{O}(kr)} \cdot (m+n)^{\mathcal{O}(1)}$ and returns a $(1-\epsilon)$-approximation with at least a constant probability.*

We give a summary of how the various problems are related to each other, and a comparison of our results with the literature in Figure 1.

1.3 Related Results

Max k-VC or Partial Vertex Cover has been extensively studied in Parameterized Complexity. In this problem we are given a graph and the task is to select a subset of k vertices covering as many of the edges as possible. The problem is known to be approximable within 0.929 and is hard to approximate within 0.944, assuming UGC [20]. Max k-VC is known to be W[1]-hard [14], parameterized by k, but admits FPT algorithms on planar graphs, graphs of bounded degeneracy, $K_{d,d}$-free graphs, and bipartite graphs, parameterized by k [1, 13, 18]. Indeed, it is among the first problems to admit FPT-AS [23, 20, 25]. It is also known to have "lossy kernels" [23, 19], a lossy version of classical kernelization.

Bera et al. [4] considered the special case of Partition Vertex Cover, where the set of edges of a graph are divided into r colors, and we want to find a subset of vertices that covers at least a certain number of edges from each color class. For this problem, they gave a polynomial-time $\mathcal{O}(\log r)$-approximation algorithm. Hung and Kao [16] generalized this to F-MaxCov, and gave a $\mathcal{O}(d \log r)$-approximation, where each element of the universe is contained in at most d sets (i.e., d is the maximum *frequency*). Bandyapadhyay et al. [2] studied this problem under the name of Fair Covering, and designed a $\mathcal{O}(d)$-approximation, but their running time is XP in the number of colors. Chekuri et al. [7] designed a general framework for F-MaxCov, yielding tight approximation guarantees for a variety of set systems satisfying certain property; in particular, they improve the approximation guarantee for frequency-d set systems to $\mathcal{O}(d + \log r)$, which is tight in polynomial time.

2 Overview of Our Results and Techniques

2.1 Reduction from CC-MaxSat to Maximum Coverage: An overview of Theorem 1.1

This theorem is essentially a randomized *approximation-preserving reduction* from CC-MaxSat to Maximum Coverage. Given an instance $\mathcal{I} = (\Phi, k)$ of CC-MaxSat, we first compute a random assignment Ψ that assigns a variable independently to be 1 with probability $p = \epsilon/2$ and 0 with probability $1 - p$. Let V^* be the set of at most k variables set to be 1 by an optimal assignment Ψ^*. It is straightforward to see that, the probability that all the variables in V^* are set to be 1 by the random assignment Ψ is p^k – we say that this is the good event $\mathcal{G}$. Now, consider a clause that is satisfied negatively by Ψ^*, i.e., a clause C that contains a negative literal $\neg x$ and $\Psi^*(x) = 0$. It is also easy to see that, conditioned on the good event $\mathcal{G}$, the probability that such a clause C is also satisfied negatively by Ψ is at least $1 - p$. Thus, the expected number of clauses that are satisfied negatively by Ψ, conditioned on $\mathcal{G}$, is at least $1 - p$ times the number of clauses satisfied negatively by Ψ^*. Markov's inequality implies that, with probability at least $1/2$, the actual number of such clauses is close to its expected value. Thus, conditioned on $\mathcal{G}$, and the previous event, we can focus on the positively satisfied clauses (note that the probability that both of these events occur is at least $1/2 \cdot (\epsilon/2)^k$. To this end, we can eliminate all the negatively satisfied clauses, and we can also prune the remaining clauses by eliminating any negative literals and the variables that are set to 0 by Ψ. Thus, all the remaining clauses only contain positive literals, which can be seen as an instance $\mathcal{I}'$ of Maximum Coverage. Furthermore, conditioned on $\mathcal{G}$, the variables set to 1 by Φ^* correspond to a family $\mathcal{F}^*$ of size k, and the elements covered by $\mathcal{F}^*$ correspond to the set of clauses satisfied only positively by Φ^*. Thus, if we find a $(1 - \epsilon)$-approximate solution to $\mathcal{I}'$, and set the corresponding variables to 1, and the rest of the variables to 0, then we get a weight-k assignment that satisfies at least $(1 - \epsilon) \cdot \mathsf{OPT}_{sat}$ clauses. Note that this reduction, combined with the algorithm of [17] gives the proof of Corollary 1.2.

Furthermore, this reduction is robust enough that it can accommodate the fairness constraints on the clauses, as defined above. To be precise, one can give a similar reduction from $\mathcal{C}$-MaxSAT to $\mathcal{C}$-MaxCov, where $\mathcal{C} \in \{\text{M, F, (M, F)}\}$ – note that when we have fairness constraints, the success probability now becomes $(r/\epsilon)^{\mathcal{O}(k)}$. Essentially, these reductions translate a constraint on the variables set to 1 (for CC-MaxSat and variants), to the corresponding family of sets (for Maximum Coverage and variants). Thus, at the expense of a multiplicative $(r/\epsilon)^{\mathcal{O}(k)}$ factor in the running time, we can focus on Maximum Coverage and its variants, which is what we do in this section, as well as in the paper. As a warm-up, we start in Section 2.2 with the vanilla Maximum Coverage on frequency-d set systems (where our algorithms *do not* improve over the known algorithms in the literature), and give a complete formal proof. Then, we will gradually introduce the ideas required to handle fairness (Section 2.3) and matroid (Section 2.4) constraints – first separately, and then together. Finally, in Section 2.5, we briefly discuss the ideas required to these results to $K_{d,d}$-free set systems and multiple matroid constraints.

2.2 Deterministic and Randomized Branching using a Largest Set

To introduce our ideas in a clean and gradual way, we start with the simplest setting of Maximum Coverage where the maximum frequency of the elements is bounded by d. Recall that we are given an instance $(U, \mathcal{F}, k)$ and the goal is to find a sub-family of $\mathcal{F}$ of size k that covers the maximum number of elements. For any sub-family $\mathcal{R} \subseteq \mathcal{F}$, let $U(\mathcal{R})$ denote the subset of elements covered by $\mathcal{R}$, and $\mathsf{OPT}_k(\mathcal{R})$ denote the maximum number of elements that can be covered by a subset of $\mathcal{R}$ of size k. Further, for a set $S \in \mathcal{F}$, we denote by $\mathcal{F} - S$, the family obtained by removing S, as well as the elements of S from each of the remaining sets. Our approach is inspired by the approaches of Skowron and Faliszewski [25] and Manurangsi [20] who show that $\mathcal{O}(kd/\epsilon)$ sets of the largest size is guaranteed to contain a $(1-\epsilon)$-approximate solution. This naturally begs the question, "*why not start by adding the largest set into the solution?*" (in a sense, the following presentation is closer in spirit to Jain et al. [17].) Let us inspect this question more closely. Let L be a largest set in $\mathcal{F}$. By looking at the contribution of coverage of each set in an optimal solution, say $\mathcal{O}$, we can easily see that $|L| \geq \frac{\mathsf{OPT}_k(\mathcal{F})}{k}$. We say that a set $S \in \mathcal{F}$ is *heavy* w.r.t. L if $|L \cap S| \geq \frac{\epsilon|L|}{k}$ (note that L is heavy w.r.t. itself). However, since the frequency of each element is bounded by d, each element in L can appear in at most d (in fact, $d-1$) sets $L \cap S$ for different $R \in \mathcal{F}$. This implies that at most $\frac{kd}{\epsilon}$ sets in $\mathcal{F}$ are heavy w.r.t. L.

Algorithm. Our algorithm simply branches on the sets in $\mathcal{H}(L)$, which is the family of heavy sets w.r.t. L. Specifically, in the branch corresponding to a heavy set $S \in \mathcal{H}(L)$, we include it in the solution, and recursively call the algorithm on the residual instance $(U \setminus S, \mathcal{F} - S, k-1)$. If any of the sets in $\mathcal{O}$ is heavy w.r.t. L, then in the branch corresponding to such a set yields an approximate solution via induction. The main idea is that, if no set in $\mathcal{O}$ is heavy w.r.t. L, then the branch corresponding to L yields a good solution. This is justified as the effect of any of the $k-1$ sets in $\mathcal{O}$ is too small. For the sake of clarity, we formally analyze this algorithm below via induction.

We want to show that, for a given input $(U', \mathcal{F}', k')$ the recursive algorithm returns a family $\mathcal{R} \subseteq \mathcal{F}'$ of size k' such that $|U'(\mathcal{R})| \geq (1-\epsilon) \cdot \mathsf{OPT}_{k'}(\mathcal{F}')$. The base case for $k' = 0$ is trivial, since the algorithm returns an empty set. Suppose that the claim is true for all inputs with budget $k-1$, and we want to prove it for $(U, \mathcal{F}, k)$. Let $\mathcal{O}$ denote an optimal solution of size k with $|\mathsf{OPT}_k(\mathcal{F})| = |U(\mathcal{O})|$.

Approximation Ratio in the Easy Case. If $\mathcal{O} \cap \mathcal{H}(L) \neq \emptyset$, then there exists a branch corresponding to a set $S \in \mathcal{O} \cap \mathcal{H}(L)$. This is the *easy case* (for the analysis). In this case, $\mathsf{OPT}_{k-1}(\mathcal{F} - S) = \mathsf{OPT}_k(\mathcal{F}) - |S|$, and hence the approximation ratio is:

$$\begin{aligned}\frac{|S| + (1-\epsilon)\mathsf{OPT}_{k-1}(\mathcal{F} \setminus S)}{\mathsf{OPT}_k(\mathcal{F})} &= \frac{|S| + (1-\epsilon)(\mathsf{OPT}_k(\mathcal{F}) - |S|)}{\mathsf{OPT}_k(\mathcal{F})} \\ &\geq \frac{(1-\epsilon)(\mathsf{OPT}_k(\mathcal{F})}{\mathsf{OPT}_k(\mathcal{F})} = (1-\epsilon)\end{aligned}$$

Approximation Ratio in the Hard Case. The more hard case (for analysis) is when $\mathcal{O} \cap \mathcal{H}(S) = \emptyset$. In this case, we argue that the branch that includes the element L is good enough. As in the easy case, we first lower bound the value of $\mathsf{OPT}_{k-1}(\mathcal{F} - L)$. By counting the unique contributions to the solution, there exists a *light* set $S_l \in \mathcal{O}$ such that for $\mathcal{O}' = \mathcal{O} \setminus \{S_l\}$, it holds $|U'(\mathcal{O}')| \geq \frac{k-1}{k} \cdot \mathsf{OPT}_k(\mathcal{F})$. Because no set in $\mathcal{O}$ is heavy w.r.t. L, it follows that for each $R \in \mathcal{O}'$, it holds that $|R \cap L| < \frac{\epsilon|L|}{k}$, and therefore by counting it holds that $|U(\mathcal{O}') \cap L| < \epsilon \cdot |L|$. Therefore,

$$\mathsf{OPT}_{k-1}(\mathcal{F} \setminus L) \geq |U'(\mathcal{O}') \setminus L| \geq |U(\mathcal{O}')| - |U(\mathcal{O}') \cap L| \geq \frac{k-1}{k} \cdot \mathsf{OPT}_k(\mathcal{F}) - \epsilon \cdot |L|.$$

Therefore, the approximation ratio of the branch that includes L is as follows.

$$\begin{aligned}\frac{|L| + (1-\epsilon)\mathsf{OPT}_{k-1}(\mathcal{F} - L)}{\mathsf{OPT}_k(\mathcal{F})} &= \frac{|L| + (1-\epsilon)\left(\frac{k-1}{k} \cdot \mathsf{OPT}_k(\mathcal{F}) - \epsilon \cdot |L|\right)}{\mathsf{OPT}_k(\mathcal{F})} \\ &\geq \frac{|L| + (1-\epsilon)\left(\frac{k-1}{k} \cdot \mathsf{OPT}_k(\mathcal{F})\right) - \epsilon|L|}{\mathsf{OPT}_k(\mathcal{F})} \\ &= \frac{(1-\epsilon)|L| + (1-\epsilon)\left(\frac{k-1}{k} \cdot \mathsf{OPT}_k(\mathcal{F})\right)}{\mathsf{OPT}_k(\mathcal{F})} \\ &\geq \frac{(1-\epsilon)(\mathsf{OPT}_k(\mathcal{F})}{\mathsf{OPT}_k(\mathcal{F})} = (1-\epsilon)\end{aligned}$$

The second last inequality holds from the fact that $|L| \geq \frac{\mathsf{OPT}_k(\mathcal{F})}{k}$.

This leads to a deterministic $(1-\epsilon)$-approximation algorithm with running time $((\frac{kd}{\epsilon})^k) \cdot (n+m)^{\mathcal{O}(1)}$.

Insight into the probabilistic branching

A closer inspection of the analysis reveals that the reason L may not a good choice is that the sets of $\mathcal{O}$ *together* cover more than a certain threshold fraction of elements covered by L. We utilize this idea through a smoothening process that captures the effect of the size of the intersection of a set S with L in a more nuanced manner. Let us define the weight $h_L(S)$ of a set $S \in \mathcal{F} \setminus \{L\}$ as $h_L(S) = \frac{|S \cap L|}{|L|}$. Our algorithm now instead does "randomized branching", i.e., it samples one set to be included in the solution according to some probability distribution, and then continues recursively. Note that the single run of the algorithm finishes in polynomial time. The probability distribution used by the algorithm is as follows: the set L is sampled with probability $1/2$, and any other set $S \in \mathcal{F} \setminus \{L\}$ is sampled with probability proportional to its weight $h(S)$ (the constant of proportionality is chosen such that this is a valid probability distribution, that is, the probabilities sum up to 1). In particular, observe that $\sum_{S \in \mathcal{F} \setminus \{L\}} h_L(S) \leq \frac{d|L|}{|L|} = d$. Thus, the probability of selecting S is at least $\frac{h_L(S)}{2d}$. Note that due to the way the weights $h_L(S)$ are defined, the sets with a large intersection with L have a greater chance of being sampled, as compared to the sets with a small intersection with L.

We will show that the algorithm returns a $(1-\epsilon)$-approximate solution with probability at least $(\frac{\epsilon}{2d})^k$, and runs in polynomial time. This implies that by repeating the algorithm $(\frac{2d}{\epsilon})^k$ times, we obtain a $(1-\epsilon)$-approximation with probability at least a positive constant. This leads to a randomized algorithm with running time $\mathcal{O}^*((\frac{2d}{\epsilon})^k)$.

The proof is again by induction. We want to show that, for any input $(U', \mathcal{F}', k')$, the algorithm returns a solution $\mathcal{R} \subseteq \mathcal{F}'$ of size k' such that $|U'(\mathcal{R})| \geq (1-\epsilon) \cdot \mathsf{OPT}_{k'}(\mathcal{F}')$, with probability at least $(\frac{\epsilon}{2d})^{k'}$. We reuse much of the notation from the previous analysis. Let $\mathcal{O}$ be an optimal solution of size k. First, the case when $L \in \mathcal{O}$, since our algorithm samples and includes L in the solution with probability 1/2. Then, conditioned on this event (that is, L being sampled), the approximation ratio analysis proceeds similarly to the *easy case* of the previous analysis. By induction, the recursive algorithm returns a $(1-\epsilon)$-approximate solution with probability at least $(\frac{\epsilon}{2d})^{k-1}$. Thus, overall, the algorithm returns a $(1-\epsilon)$-approximation with probability at least $\frac{1}{2}(\frac{\epsilon}{2d})^{k-1} \geq (\frac{\epsilon}{2d})^k$.

Now suppose $L \notin \mathcal{O}$. As before, let $S_l \in \mathcal{O}$ be a light set as defined earlier, and $\mathcal{O}' = \mathcal{O} \setminus \{S_l\}$. We analyze by considering the following two cases: either (i) $|U(\mathcal{O}') \cap L| \leq \epsilon \cdot |L|$, or (ii) $|U(\mathcal{O}') \cap L| > \epsilon \cdot |L|$.

In case (i), we are effectively in the same situation as the *hard case* of the previous analysis – as before, the algorithm samples L with probability at least 1/2, and as argued in the *hard case*, conditioned on the previous event, the branch corresponding to L returns a $(1-\epsilon)$-approximate solution, but now with probability at least $(\frac{\epsilon}{2d})^{k-1}$ by induction. Therefore, we obtain an $(1-\epsilon)$-approximate solution with probability at least $\frac{1}{2}(\frac{\epsilon}{2d})^{k-1} \geq (\frac{\epsilon}{2d})^k$.

In case (ii), we have that $|U(\mathcal{O}') \cap L| > \epsilon \cdot |L|$. Notice that,

$$\sum_{S \in \mathcal{O}'} |S \cap L| > \epsilon.$$

This implies that

$$\sum_{S \in \mathcal{O}'} h_L(S) = \sum_{S \in \mathcal{O}'} \frac{|S \cap L|}{|L|} \geq |U(\mathcal{O}') \cap L| > \epsilon |L|.$$

Therefore, the total weight of the sets in $\mathcal{O}'$ is at least ϵ. Therefore, the probability that the algorithm samples a set from $\mathcal{O}'$ is at least $\frac{\epsilon}{2d}$. Conditioned on this event, the approximation ratio analysis now proceeds as in the *easy case*, and the algorithm returns a $(1-\epsilon)$-approximate solution with probability at least $\frac{\epsilon}{2d}(\frac{\epsilon}{2d})^{k-1} = (\frac{\epsilon}{2d})^k$.

2.3 Handling fairness constraints via the Bucketing trick

The aforementioned idea of prioritizing the largest set L, or sets that are heavy w.r.t. it, fails to generalize when we have multiple coverage constraints in F-MaxCov. This is simply because there is no notion of "the largest set" even when we want to cover elements of *two* different colors, each with different coverage requirements. To handle such multiple coverage constraints, our idea is to use multidimensional-knapsack-style bucketing technique to group the sets of $\mathcal{F}$ into *approximate equivalence classes*, called *bags* for short. At a high level, all the vertices belonging to a particular bag contain *approximately equal* (i.e., within a factor of $(1+\epsilon)$) number of elements of *all* of the r colors. Thus, in isolation, any two sets L_1 and L_2 belonging to the same bag are interchangeable, since we tolerate an ϵ-factor loss in the coverage. Since the total number of bags can be shown to be $\left(\frac{\log k}{\epsilon}\right)^{\mathcal{O}(rk)}$, and hence we can "guess" a bag $\mathcal{B}$ containing a set from a solution $\mathcal{O}$. However, due to different amount

of overlap with an optimal solution $\mathcal{O}$, two sets $L_1, L_2 \in \mathcal{B}$ may not be interchangeable w.r.t. $\mathcal{O}$. That is, if $L_1 \in \mathcal{O}$, then $\mathcal{O} \setminus \{L_1\} \cup \{L_2\}$ may not be a good solution. However, assuming that we have correctly guessed a bag that intersects with $\mathcal{O}$, we can then select a set $L \in \mathcal{B}$, and define the heavy sets (for deterministic algorithm) or weights $h_L(\cdot)$ (for the randomized algorithm) w.r.t. L. Note that, since we have multiple coverage constraints, we cannot simply look at the total size of the intersection $|S \cap L|$. Instead, we need to tweak the notion of *heaviness* that takes into account the number of elements of each color in the intersection $S \cap L$. To summarize, we need two additional ideas to handle multiple colors in F-MaxCov: (1) "guessing" over buckets, and (2) a suitable generalization of the notion of heaviness. Modulo this, the rest of the analysis is again similar to the *easy* and *hard* cases as before. Using these ideas, we can prove the first part of Theorem 1.4.

2.4 Handling Matroid Constraints

First, we consider M-MaxCov (note that this is an orthogonal generalization of Maximum Coverage, without multiple coverage constraints), where the solution is required to be an independent set in the given matroid $\mathcal{M} = (\mathcal{F}, I)$ of rank k. We assume that we are given an *oracle access* to $\mathcal{M}$ in the form of an algorithm that answers the queries of the form "Is $\mathcal{R}$ an independent set?" for any subset $\mathcal{R} \subseteq \mathcal{F}$. Let us revisit the initial deterministic FPT-AS for Maximum Coverage and try to generalize it to M-MaxCov. Recall that this algorithm branches on each set $S \in \mathcal{H}(L)$, where L is a largest set. The analysis of easy case goes through even in presence of the matroid constraint, since we branch on a set from an optimal solution $\mathcal{O}$. However, in the hard case, our analysis replaces a $S_l \in \mathcal{O}$ with L, and argues that the branch corresponding to L returns a $(1-\epsilon)$-approximate solution. However, this does not work for M-MaxCov, since $(\mathcal{O} \setminus \{S_l\}) \cup \{L\}$ may not be an independent set in $\mathcal{M}$. To summarize, although L handles the coverage constraints (approximately), it may fail to handle the matroid constraint. In fact, it may just so happen that L is not a good set *at all*, in the sense that, for *any* set $S \in \mathcal{O}$, $(\mathcal{O} \setminus \{S\}) \cup \{L\}$ is not independent in $\mathcal{M}$, which is crucial for the induction to go through.

To solve this issue, we resort to the bucketing idea as in the fair coverage case (thus, the subsequent arguments generalize to (M, F)-MaxCov in a straightforward manner; although let us stick to the special case of M-MaxCov for now). Indeed, branching w.r.t. the largest set L is an overkill – it suffices to pin down a bag $\mathcal{B}$ containing a set in $\mathcal{O}$ (it does not even have to be the largest set), by "guessing" from $\mathcal{O}\left(\frac{\log k}{\epsilon}\right)$ bags. However, we again cannot select an arbitrary set $S \in \mathcal{B}$ and define heavy sets w.r.t. S, precisely due to the matroid compatibility issues mentioned earlier. Therefore, we resort to the idea of *representative sets* from matroid theory [22] [6] Assuming our guess for $\mathcal{B}$ is correct, there exists some $S \in \mathcal{B} \cap \mathcal{O}$. However, we cannot further "guess" S, since the size of the bag may be too large. Instead, we compute a inclusion-wise maximal independent set $\mathcal{B}' \subseteq \mathcal{B}$. Note that the size of $\mathcal{B}'$ is at most k, and it can be computed using polynomially many queries to the independence oracle. However, it may very well happen that $S \notin \mathcal{B}'$. Nevertheless, using matroid properties, we can argue that, there exists a set $S' \in \mathcal{B}'$, such that $(\mathcal{O} \setminus \{S\}) \cup \{S'\}$ is an independent set. Thus, $\mathcal{B}'$ is a *representative set* of $\mathcal{B}$. Furthermore, since both S and S' come from the same bag, they cover approximately the same number of elements. Thus, our modified deterministic

[6] Although this is a powerful hammer in its full generality – which we do use to handle multiple matroid constraints – our specialized setting lets us use much simpler arguments to handle single matroid constraint in M-MaxCov/(M, F)-MaxCov.

algorithm works as follows. First, it computes maximal independent set $\mathcal{B}' \subseteq \mathcal{B}$, and for each $S' \in \mathcal{B}'$, it computes the heavy family $\mathcal{H}(S')$. Then, it branches over all sets in $\bigcup_{S' \in \mathcal{B}'} \mathcal{H}(S')$. If one of the branches corresponds to branching on a set from $\mathcal{O}$, then the analysis is similar to the easy case. Otherwise, we know that $(\mathcal{O} \setminus \{S\}) \cup \{S'\}$ is an $(1-\epsilon)$-approximate solution that is also an independent set. Therefore, the branch corresponding to S' yields the required $(1-\epsilon)$-approximation. We can improve the running time via doing a randomized branching in two steps: first we pick a set $S'' \in \mathcal{B}'$ uniformly at random, and then we perform the probabilistic branching using the weights $h_{S''}(\cdot)$.

Both deterministic and randomized variants incur a further multiplicative overhead of $(\frac{k \log k}{\epsilon})^k$ due to first guessing a bag, and then computing a representative set $\mathcal{B}' \subseteq \mathcal{B}$ of the bag, and thus do not improve over the results of Sellier [24] in terms of running time for M-MaxCov. However, this idea naturally generalizes to (M, F)-MaxCov, with the appropriate modifications in bucketing (as mentioned in the previous paragraph) to handle the multiple coverage requirements of different colors. This leads to the proof of Theorem 1.5.

2.5 Further Extensions

The ideas mentioned in the previous subsections can be extended to even more general settings in a couple of ways. First, we describe how to extend the ideas from frequency-d set systems for Maximum Coverage to $K_{d,d}$-free set systems, i.e., set system $(U, \mathcal{F})$, where no d sets in $\mathcal{F}$ contain d elements of U in common. Note that frequency-d set systems are $K_{d+1,d+1}$-free. Then, we describe how the linear algebraic toolkit of *representative sets* can be used to handle multiple (linear) matroid constraints.

$K_{d,d}$-free Set Systems

Next, we consider $K_{d,d}$-free set systems $(U, \mathcal{F})$, where no d sets of $\mathcal{F}$ contain d common elements of U. To design the FPT-AS on $K_{d,d}$-free set systems, we combine the bucketing idea along with the combinatorial properties of $K_{d,d}$-free graphs to bound the number of heavy neighbors of a set. To this end, however, we need to modify the precise definition of *heaviness* (as in Jain et al. [17]). This leads to a somewhat cumbersome branching algorithm that handles colors differently based on their coverage requirement. For colors with small coverage requirement, we highlight the covered vertices using the standard technique of *label coding*[7]. Now, vertices in a bag cover the elements with the same label and for colors with high coverage requirements, the sizes of the sets in the same bag are "almost" the same. Then, we pick an *arbitrary* set L from the bag $\mathcal{B}$, and we branch on (suitably defined) *heavy* sets w.r.t. S. Since the number of heavy sets is bounded by a function of k, d, and ϵ, this leads to a deterministic version of Theorem 1.4 to $K_{d,d}$-free set systems. Note that since frequency-d set systems is a special case of this, this implies a deterministic FPT-AS in this case; however with a much worse running time compared to Theorem 1.4. This leads to the proof of Theorem 1.5.

[7] The technique is better known as *color coding*. However, this creates an unfortunate clash of terminology – these *colors* have nothing to do with the original colors corresponding to coverage constraints. Bandyapadhyay et al. [3] instead use the term "label coding", and we also adopt the same terminology

Handling Multiple Matroid Constraints

Our results on M-MaxCov and (M, F)-MaxCov can be generalized to handle multiple matroid constraints on the solution, in the case when the matroids are *linear* or representable[8]. In this more general problem, we are given q linear matroids $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_q$, where $\mathcal{M}_i = (\mathcal{F}, I_i)$, each of rank at most k, and the solution S is required to be independent in all q matroids, i.e., $S \in \bigcap_{i \in [q]} I_i$. In this case, we can use linear algebraic tools ([22, 12]) to compute a representative set of size qk instead of k, and the computation requires $2^{\mathcal{O}(qk)} \cdot n^{\mathcal{O}(1)}$ time. Thus, FPT-ASes for this problem now have a factor of q in the exponent.

Note that our FPT-AS improves upon the polynomial-time approximation guarantee of $1 - 1/e$ of Calinescu et al. [6] for monotone submodular maximization subject to a matroid constraint, in the special case of $K_{d,d}$-free coverage functions. To the best of our knowledge, this is the largest class of monotone submodular functions and matroid constraints for which the lower bound of $1 - 1/e$ can be overcome, even in FPT time. Further, the analogous results to (M, F)-MaxSAT generalize these results to maximization of non-monotone/non-submodular functions.

3 Preliminaries

For a positive integer q, let $[q] \coloneqq \{1, 2, \ldots, q\}$.

Convention. We consistently use index j to refer to a color from the range $[r]$, and may often write "for a color j" instead of "for a color $j \in [r]$". Finally, for a coverage requirement function t (resp. variations such as $t', \widetilde{t}$), and a color j, we shorten $t(j)$ to t_j (resp. $t'_j, \widetilde{t}_j$).

4 Reduction from (M, F)-MaxSAT to (M, F)-MaxCov

In this section, we begin with a polynomial time approximate preserving randomized reduction from F-MaxSAT to F-MaxCov. The success probability of the reduction is $\mathcal{O}((\epsilon/r)^k)$. Recall that in the F-MaxSAT problem, given a CNF-formula Φ with $\chi : \mathsf{cla}(\Phi) \to [r]$, a coverage demand function $t : [r] \to \mathbb{N}$ and an integer k, the goal is to find an assignment of weight at most k that satisfies at least $t(i)$ (also denoted as t_i) clauses of color class i (an assignment Ψ satisfying these properties is called *optimal weight k assignment*).

We begin with some basic definitions. Let Φ be a CNF-formula. By $\mathsf{var}(\Phi)$ and $\mathsf{cla}(\Phi)$, we denote the set of variables and clauses in the formula Φ, respectively. An assignment to a CNF-formula Φ is a function $\Psi : \mathsf{var}(\Phi) \to \{0, 1\}$. The weight of an assignment is the number of variables that have been assigned 1. By $T(\Psi)$ and $F(\Psi)$, we denote the set of variables assigned 1 and 0 by the assignment Ψ, respectively. For a clause $c \in \mathsf{cla}(\Phi)$, $\mathsf{var}(c)$ is the set of variables that occur in the clause c as a positive or negative literal. Similarly, for a set of clauses $C \in \mathsf{cla}(\Phi)$, $\mathsf{var}(C)$ is the set of variables that occur as a positive or negative literal in a clause $c \in C$.

Our reduction (Algorithm 1) takes an instance (Φ, χ, t, k) of F-MaxSAT as input. It constructs a random assignment Ψ by setting each variable to 1 with probability p and 0 with probability $1 - p$. It constructs a new formula by first removing the set of clauses that are

[8] A matroid $\mathcal{M} = (E, \mathcal{I})$ is representable over a field $\mathbb{F}$ if there exists a matrix M such that there exists a bijection between E and the columns on M with the property that, a subset $E' \subseteq E$ is independent in $\mathcal{M}$ iff the corresponding set of columns are linearly independent over $\mathbb{F}$.

Algorithm 1 Reduction Algorithm($\mathcal{I} = (\Phi, \chi, t, k)$ of F-MaxSAT).

1: Construct a random assignment Ψ as follows. For each variable $x \in \mathsf{var}(\Phi)$, independently set $\Psi(x)$ to 1 with probability p and 0 with probability $1-p$. ▷ We will later set $p = \frac{\epsilon}{2r}$.
2: Construct a new formula Φ' as follows:
- Let $N \subseteq \mathsf{cla}(\Phi)$ be the set of clauses that are satisfied negatively by Ψ. Then, $\mathsf{cla}(\Phi') = \mathsf{cla}(\Phi) \setminus N$.
- For each $c \in \mathsf{cla}(\Phi')$, remove all the variables in c that occur either as a negative literal or set to 0 by Ψ.
- For each $c \in \mathsf{cla}(\Phi')$, add $\mathsf{var}(c)$ to $\mathsf{var}(\Phi')$.

3: Construct an instance $\mathcal{J}_\Psi = (\mathcal{U}, \mathcal{F}, \chi', t', k')$ of F-MaxCov as follows:
- Set $\mathcal{U} = \mathsf{cla}(\Phi')$.
- For each $v \in \mathsf{var}(\Phi')$, add a set f_v to $\mathcal{F}$ where $f_v = \{c \in \mathsf{cla}(\Phi') \colon v \in \mathsf{var}(c)\}$.
- For each $c \in \mathsf{cla}(\Phi')$, if the corresponding element in $\mathcal{U}$ is e_c, set $\chi'(e_c) = \chi(c)$.
- Set $t'(i) = t(i) - \frac{|N \cap \chi^{-1}(i)|}{1-\epsilon}$ for each $i \in [r]$.
- Set $k' = k$.

satisfied negatively by Ψ, followed by removing negative literals from the remaining clauses. It reduces the formula to an instance $(\mathcal{U}, \mathcal{F}, \chi', t', k')$ of F-MaxCov as described in Step 3 of Algorithm 1. Clearly, this reduction takes polynomial time.

Next, we prove the correctness of our reduction. For a Yes-instance $\mathcal{I}$ of F-MaxSAT, let $\Psi^\star$ be an optimal weight k assignment. Let $N^\star$ be the set of clauses satisfied negatively by $\Psi^\star$, i.e., every clause in $N^\star$ contains a negative literal that is set to 1, and let $P^\star$ be the set of clauses satisfied only positively by $\Psi^\star$, i.e., every clause in $P^\star$ contains a positive literal that is set to 1 and no negative literal in this clause is set to 1. By $N_i^\star$, we mean the set of clauses in color class i satisfied negatively by $\Psi^\star$ and by $P_i^\star$, we mean the set of clauses in color class i satisfied only positively by $\Psi^\star$. We call a random assignment, constructed in Algorithm 1, *good* if each variable in $T(\Psi^\star)$ (positive variables under $\Psi^\star$) is assigned 1 by Ψ, i.e., $T(\Psi^\star) \subseteq T(\Psi)$, which occurs with probability at least p^k. For a good assignment Ψ, let N_i denote the set of clauses in color class i satisfied negatively by Ψ and P_i denote the set of clauses in color class i satisfied only positively by Ψ. We say that an event $\mathcal{G}$ is *good* if a good assignment Ψ is generated in Algorithm 1. We begin with the following claim.

▷ Claim 4.1. Given a Yes-instance $\mathcal{I}$ of F-MaxSAT, with probability at least 1/2, a good assignment Ψ satisfies at least $(1-\epsilon)|N_i^\star|$ clauses negatively, for each $i \in [r]$.

Proof. Let (Φ, χ, t, k) be a Yes-instance of F-MaxSAT. Let Ψ be a good assignment, which occurs with probability at least p^k. We show that Ψ satisfies at least $(1-\epsilon)|N^\star|$ clauses negatively, with probability at least 1/2. Let X_i be the number of clauses in $N_i^\star$ that are satisfied negatively by Ψ. We define an indicator random variable x_j, for each $j \in [|N_i^\star|]$, as follows.

$$x_j = \begin{cases} 1 & \text{clause } c_j \in N_i^\star \text{ is satisfied negatively by } \Psi \\ 0 & \text{otherwise} \end{cases}$$

$$\mathsf{Pr}(x_j|\mathcal{G}) = \mathsf{Pr}(\text{clause } c_j \in N_i^\star \text{ is satisfied negatively by } \Psi|\mathcal{G}) \geq (1-p)$$

$$\mathbb{E}[X_i|\mathcal{G}] = \sum_{j \in [|N_i^\star|]} x_j \times \mathsf{Pr}(x_j|\mathcal{G}) \geq (1-p)|N_i^\star|$$

Let $Y_i = |N_i^\star| - |N_i|$, where N_i is the set of clauses satisfied negatively by Ψ. Note that,

$$Y_i = |N_i^\star| - |N_i| \leq |N_i^\star \setminus N_i| = |N_i^\star| - X_i$$

Thus,

$$\mathbb{E}[Y_i|\mathcal{G}] \leq |N_i^\star| - \mathbb{E}[X_i|\mathcal{G}] \leq p|N_i^\star|$$

Since $Y_i \geq 0$, we can use Markov's inequality and get

$$\mathsf{Pr}(Y_i \geq 2rp|N_i^\star||\mathcal{G}) \leq \frac{\mathbb{E}[Y_i]}{2rp|N_i^\star|} \leq \frac{1}{2r}$$

Since $Y_i = |N_i^\star| - |N_i|$, we get

$$\mathsf{Pr}(|N_i| \leq (1-2rp)|N_i^\star||\mathcal{G}) \leq \frac{1}{2r}$$

By union bound,

$$\mathsf{Pr}\left(\exists i \in [r], |N_i| \leq (1-2rp)|N_i^\star||\mathcal{G}\right) \leq \sum_{i\in[r]} \mathsf{Pr}(|N_i| \leq (1-2rp)|N_i^\star||\mathcal{G}) \leq \frac{1}{2}$$

This implies that

$$\mathsf{Pr}\left(\forall i \in [r], |N_i| > (1-2rp)|N_i^\star||\mathcal{G}\right) \geq \frac{1}{2}$$

Setting $p = \frac{\epsilon}{2r}$ gives us the required result, i.e., with probability at least $1/2$, for all colors $i \in [r]$, $|N_i| > (1-\epsilon)|N_i^\star|$. ◁

▶ **Lemma 4.2.** *If $\mathcal{I} = (\Phi, \chi, t, k)$ is a yes-instance of F-MaxSAT, then with probability at least $(\frac{\epsilon}{2r})^k$, the reduced instance $\mathcal{J}_\Psi = (\mathcal{U}, \mathcal{F}, \chi', t', k')$ is a yes-instance of F-MaxCov.*

Proof. Let $\mathcal{I}$ be a Yes-instance of F-MaxSAT and let $\Psi^\star$ be an optimal weight k assignment. Further, let $N^\star$ be the set of clauses satisfied negatively by $\Psi^\star$, i.e., every clause in $N^\star$ contains a negative literal that is set to 1, and let $P^\star$ be the set of clauses satisfied only positively by $\Psi^\star$, i.e., every clause in $P^\star$ contains a positive literal that is set to 1 and no negative literal in this clause is set to 1. Then, there exists a set $V_{P^\star} \subseteq \mathsf{var}(P^\star)$ of size at most k that satisfies all the clauses in $P^\star$ positively, i.e., for each clause in $P^\star$, there is a variable in $V_{P^\star}$ that occurs as a positive literal in it and is assigned 1 under $\Psi^\star$. Let Ψ be a good assignment which is generated with probability at least $(\frac{\epsilon}{2r})^k$. Since Ψ is a good assignment, $T(\Psi^\star) \subseteq T(\Psi)$ and $F(\Psi) \subseteq F(\Psi^\star)$. Hence, $P^\star \subseteq P$. Thus, $P^\star \subseteq \mathcal{U}$ and for each variable in $\mathsf{var}(P^\star)$, we have a set in the family $\mathcal{F}$. Let $Z = \{f_v \in \mathcal{F} \colon v \in V_{P^\star}\}$. We claim that Z is a solution to $\mathcal{J}_\Psi$. Clearly, Z covers at least $|P_i^\star|$ elements in $\mathcal{U}$, for each $i \in [r]$. We claim that $t_i' \leq |P_i^\star|$. Since $\Psi^\star$ is a solution to $\mathcal{I}$, it satisfies t_i clauses for each $i \in [r]$. Since $t_i = |P_i^\star| + |N_i^\star|$, due to Claim 4.1, we know that $t_i \leq |P_i^\star| + \frac{|N_i|}{1-\epsilon}$. Thus, $|P_i^\star| \geq t_i - \frac{|N_i|}{1-\epsilon}$. Since $t_i' = t_i - \frac{|N_i|}{1-\epsilon}$, $t_i' \leq |P_i^\star|$. This completes the proof. ◀

▶ **Lemma 4.3.** *Assume that $\mathcal{I} = (\Phi, \chi, t, k)$ is a yes-instance of F-MaxSAT. If there exists $(1-\epsilon)$-approximate solution for $\mathcal{J}_\Psi = (\mathcal{U}, \mathcal{F}, \chi', t', k')$, where Ψ is a good assignment, then there exists $(1-\epsilon)$-approximate solution for $\mathcal{I}$ with probability at least $\frac{1}{2}$.*

Proof. Let $\Psi^\star$ be an optimal assignment. Due to Claim 4.1, Ψ satisfies at least $(1-\epsilon)|N_i^\star|$ clauses negatively, for each $i \in [r]$, with probability at least $1/2$. Let S be a $(1-\epsilon)$-approximate solution to $\mathcal{J}_\Psi$. We construct an assignment σ as follows: if $f_x \in S$, then $\sigma(x) = 1$, otherwise 0. We claim that σ is a $(1-\epsilon)$-approximate solution to $\mathcal{I}$. Due to the construction of $\mathcal{J}_\Psi$, note that $\mathcal{F}$ does not contain a set corresponding to the variable that is set to 0 by Ψ. Thus, if $\Psi(x) = 0$, then $\sigma(x) = 0$. Hence, σ satisfies at least $(1-\epsilon)|N_i^\star|$ clauses negatively, for each $i \in [r]$, with probability at least $1/2$. Next, we argue that σ satisfies at least $(1-\epsilon)|P_i^\star|$ clauses only positively, for each $i \in [r]$. Since S is a $(1-\epsilon)$-approximate solution to $\mathcal{J}_\Psi$, for each $i \in [r]$, S covers at least $(1-\epsilon)t_i'$ elements. Recall that $\mathcal{U}$ contains an element corresponding to each clause in $\cup_{i\in[r]} P_i$. Thus, σ satisfies at least $(1-\epsilon)t_i'$ clauses only positively for each $i \in [r]$. Recall that $t_i' = t_i - \frac{|N_i|}{1-\epsilon}$. Thus, $|P_i| + |N_i| \geq (1-\epsilon)(t_i - \frac{|N_i|}{1-\epsilon}) + |N_i| = (1-\epsilon)t_i$. Hence, σ is a factor $(1-\epsilon)$-approximate solution for $\mathcal{I}$. ◀

Due to Lemma 4.2 and 4.3, we have the following result.

▶ **Theorem 4.4.** *There exists a polynomial time randomized algorithm that given a* Yes*-instance $\mathcal{I}$ of F-MaxSAT generates a* Yes*-instance $\mathcal{J}$ of F-MaxCov with probability at least $(\frac{\epsilon}{2r})^k$. Furthermore, given a factor $(1-\epsilon)$-approximate solution of $\mathcal{J}$, it can be extended to a $(1-\epsilon)$-approximate solution of $\mathcal{I}$ with probability at least $1/2$.*

Note that if the variable-clause incidence graph of the input formula belongs to a subgraph closed family $\mathcal{H}$, then the incidence graph of the resulting instance of F-MaxCov will also belong to $\mathcal{H}$. Thus, due to Theorem 4.4 and Theorem 1.2 in [17], we have the following result, which is an improvement over Theorem 1.1 in [17].

▶ **Theorem 4.5.** *There is a randomized algorithm that given a* Yes*-instance $\mathcal{I}$ of* CC-Max-SAT*, where the variable-clause incidence graph is $K_{d,d}$-free, returns a factor $(1-\epsilon)$-approximate solution with probability at least $(1-\frac{1}{e})$, and runs in time $(\frac{rdk}{\epsilon})^{\mathcal{O}(dk)}(n+m)^{\mathcal{O}(1)}$.*

The $(\frac{r}{\epsilon})^{\mathcal{O}(k)}$ factor in the running time of the algorithm in Theorem 4.5 comes by repeating the algorithm in Theorem 4.4, followed by Theorem 1.2 in [17], independently $(\frac{r}{\epsilon})^{\mathcal{O}(k)}$ many times. This also boosts the success probabilty to at least $(1-\frac{1}{e})$.

▶ Remark 4.6. Note that the reduction from F-MaxSAT to F-MaxCov also works in presence of matroid constraint(s) on the set of variables assigned 1. Recall that in the former (resp. latter) problem, we are given a matroid $\mathcal{M}$ on the set of variables (resp. sets), and the set of at most k variables assigned 1 (resp. at most k sets chosen in the solution) is required to be an independent set in $\mathcal{M}$. This follows from the fact that the randomized algorithm preserves the optimal independent set in the set cover instance with good probability.

5 Conclusion

In this paper, we designed FPT-approximation schemes for (M, F)-MaxSAT, which is a generalization of the CC-MaxSat problem with fairness and matroid constraints. In particular, we designed FPT-AS for the classes of formulas where the maximum frequency of a variable in the clause is bounded by d, and more generally, for $K_{d,d}$-free formulas. Our algorithm for F-MaxCov on the set systems of frequency bounded by d is substantially faster compared to the recent result of Bandyapadhyay et al. [3], even for the special case of $d = 2$. We use a novel combination of the bucketing trick and a carefully designed probability distribution in order to obtain this faster FPT-AS.

Our work naturally leads to the following intriguing questions. Firstly, our approximation-preserving reduction from CC-MaxSat (and variants) to Maximum Coverage (and variants) is inherently randomized. Is it possible to derandomize this reduction? A similar

question of derandomization is also interesting for our aforementioned algorithm for F-MaxCov on bounded-frequency set systems. In this case, can we design an FPT-AS for the problem running in time single-exponential in k?

References

1 Omid Amini, Fedor V. Fomin, and Saket Saurabh. Implicit branching and parameterized partial cover problems. *J. Comput. Syst. Sci.*, 77(6):1159–1171, 2011. doi:10.1016/j.jcss.2010.12.002.

2 Sayan Bandyapadhyay, Aritra Banik, and Sujoy Bhore. On colorful vertex and edge cover problems. *Algorithmica*, pages 1–12, 2023.

3 Sayan Bandyapadhyay, Zachary Friggstad, and Ramin Mousavi. A parameterized approximation scheme for generalized partial vertex cover. In Pat Morin and Subhash Suri, editors, *WADS 2023*, pages 93–105, 2023.

4 Suman Kalyan Bera, Shalmoli Gupta, Amit Kumar, and Sambuddha Roy. Approximation algorithms for the partition vertex cover problem. *Theor. Comput. Sci.*, 555:2–8, 2014. doi:10.1016/j.tcs.2014.04.006.

5 Markus Bläser. Computing small partial coverings. *Inf. Process. Lett.*, 85(6):327–331, 2003. doi:10.1016/S0020-0190(02)00434-9.

6 Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011. doi:10.1137/080733991.

7 Chandra Chekuri, Tanmay Inamdar, Kent Quanrud, Kasturi R. Varadarajan, and Zhao Zhang. Algorithms for covering multiple submodular constraints and applications. *J. Comb. Optim.*, 44(2):979–1010, 2022. doi:10.1007/s10878-022-00874-x.

8 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight FPT approximations for k-median and k-means. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *ICALP 2019*, pages 42:1–42:14, 2019.

9 M. Cygan, F. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

10 Uriel Feige. A threshold of ln n for approximating set cover (preliminary version). In Gary L. Miller, editor, *STOCS, 1996*, pages 314–318. ACM, 1996. doi:10.1145/237814.237977.

11 Andreas Emil Feldmann, Karthik C. S., Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020. doi:10.3390/a13060146.

12 F. V. Fomin, D. Lokshtanov, and S. Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *proceedings of SODA*, pages 142–151, 2014.

13 Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Subexponential algorithms for partial cover problems. *Inf. Process. Lett.*, 111(16):814–818, 2011. doi:10.1016/j.ipl.2011.05.016.

14 Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Parameterized complexity of vertex cover variants. *Theory Comput. Syst.*, 41(3):501–520, 2007. doi:10.1007/s00224-007-1309-3.

15 Chien-Chung Huang and François Sellier. Matroid-constrained maximum vertex cover: Approximate kernels and streaming algorithms. In Artur Czumaj and Qin Xin, editors, *SWAT 2022*, pages 27:1–27:15, 2022. doi:10.4230/LIPIcs.SWAT.2022.27.

16 Eunpyeong Hung and Mong-Jen Kao. Approximation algorithm for vertex cover with multiple covering constraints. *Algorithmica*, 84(1):1–12, 2022. doi:10.1007/s00453-021-00885-w.

17 Pallavi Jain, Lawqueen Kanesh, Fahad Panolan, Souvik Saha, Abhishek Sahu, Saket Saurabh, and Anannya Upasana. Parameterized approximation scheme for biclique-free max k-weight SAT and max coverage. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3713–3733. SIAM, 2023. doi:10.1137/1.9781611977554.ch143.

18 Tomohiro Koana, Christian Komusiewicz, André Nichterlein, and Frank Sommer. Covering many (or few) edges with k vertices in sparse graphs. In Petra Berenbrink and Benjamin Monmege, editors, *STACS 2022*, pages 42:1–42:18, 2022. doi:10.4230/LIPIcs.STACS.2022.42.

19 Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *STOC 2017*, pages 224–237. ACM, 2017. doi:10.1145/3055399.3055456.

20 Pasin Manurangsi. A note on max k-vertex cover: Faster fpt-as, smaller approximate kernel and improved approximation. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *SOSA 2019*, pages 15:1–15:21, 2019.

21 Pasin Manurangsi. Improved FPT approximation scheme and approximate kernel for biclique-free max k-weight SAT: greedy strikes back. *CoRR*, abs/2403.06335, 2024. doi:10.48550/ARXIV.2403.06335.

22 Dániel Marx. A parameterized view on matroid optimization problems. In Hajo Broersma, Stefan S. Dantchev, Matthew Johnson, and Stefan Szeider, editors, *ACiD 2006*, volume 7 of *Texts in Algorithmics*, page 158. King's College, London, 2006.

23 Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008.

24 François Sellier. Parameterized matroid-constrained maximum coverage. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *ESA 2023*, volume 274 of *LIPIcs*, pages 94:1–94:16, 2023.

25 Piotr Skowron and Piotr Faliszewski. Chamberlin-courant rule with approval ballots: Approximating the maxcover problem with bounded frequencies in FPT time. *J. Artif. Intell. Res.*, 60:687–716, 2017. doi:10.1613/jair.5628.

26 Maxim Sviridenko. Best possible approximation algorithm for MAX SAT with cardinality constraint. *Algorithmica*, 30(3):398–405, 2001. doi:10.1007/s00453-001-0019-5.

27 Laurence A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Comb.*, 2(4):385–393, 1982. doi:10.1007/BF02579435.

Breaking a Barrier in Constructing Compact Indexes for Parameterized Pattern Matching

Kento Iseri
Kyushu Institute of Technology, Japan

Tomohiro I
Kyushu Institute of Technology, Japan

Diptarama Hendrian
Tokyo Medical and Dental University, Japan

Dominik Köppl
University of Yamanashi, Japan

Ryo Yoshinaka
Tohoku University, Sendai, Japan

Ayumi Shinohara
Tohoku University, Sendai, Japan

Abstract

A parameterized string (p-string) is a string over an alphabet $(\Sigma_s \cup \Sigma_p)$, where Σ_s and Σ_p are disjoint alphabets for static symbols (s-symbols) and for parameter symbols (p-symbols), respectively. Two p-strings x and y are said to parameterized match (p-match) if and only if x can be transformed into y by applying a bijection on Σ_p to every occurrence of p-symbols in x. The indexing problem for p-matching is to preprocess a p-string T of length n so that we can efficiently find the occurrences of substrings of T that p-match with a given pattern. Let σ_s and respectively σ_p be the numbers of distinct s-symbols and p-symbols that appear in T and $\sigma = \sigma_s + \sigma_p$. Extending the Burrows-Wheeler Transform (BWT) based index for exact string pattern matching, Ganguly et al. [SODA 2017] proposed parameterized BWTs (pBWTs) to design the first compact index for p-matching, and posed an open problem on how to construct the pBWT-based index in compact space, i.e., in $O(n \lg |\Sigma_s \cup \Sigma_p|)$ bits of space. Hashimoto et al. [SPIRE 2022] showed how to construct the pBWT for T, under the assumption that $\Sigma_s \cup \Sigma_p = [0..O(\sigma)]$, in $O(n \lg \sigma)$ bits of space and $O(n \frac{\sigma_p \lg n}{\lg \lg n})$ time in an online manner while reading the symbols of T from right to left. In this paper, we refine Hashimoto et al.'s algorithm to work in $O(n \lg \sigma)$ bits of space and $O(n \frac{\lg \sigma_p \lg n}{\lg \lg n})$ time in a more general assumption that $\Sigma_s \cup \Sigma_p = [0..n^{O(1)}]$. Our result has an immediate application to constructing parameterized suffix arrays in $O(n \frac{\lg \sigma_p \lg n}{\lg \lg n})$ time and $O(n \lg \sigma)$ bits of working space. We also show that our data structure can support backward search, a core procedure of BWT-based indexes, at any stage of the online construction, making it the first compact index for p-matching that can be constructed in compact space and even in an online manner.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases Index for parameterized pattern matching, Parameterized Burrows-Wheeler Transform, Online construction

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.89

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2308.05977

Funding *Tomohiro I*: KAKENHI (Grant Numbers 19K20213, 22K11907).
Dominik Köppl: KAKENHI (Grant Number 23H04378).
Ryo Yoshinaka: KAKENHI (Grant Numbers 18K11150, 20H05703, 23K11325, 24K14827).
Ayumi Shinohara: KAKENHI (Grant Number 21K11745).

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 89; pp. 89:1–89:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A *parameterized string* (*p-string*) is a string over an alphabet $(\Sigma_s \cup \Sigma_p)$, where Σ_s and Σ_p are disjoint alphabets for *static symbols* (*s-symbols*) and for *parameter symbols* (*p-symbols*), respectively. Two p-strings x and y are said to *parameterized match* (*p-match*) if and only if x can be transformed into y by applying a bijection on Σ_p to every occurrence of p-symbols in x. For example with $\Sigma_s = \{\mathtt{a}, \mathtt{b}\}$ and $\Sigma_p = \{\mathtt{X}, \mathtt{Y}, \mathtt{Z}\}$, two p-strings `aXYbZXaY` and `aZYbXZaY` p-match because `aXYbZXaY` can be transformed into `aZYbXZaY` by replacing `X`, `Y` and `Z` with `Z`, `Y` and `X`, respectively. The concept of p-matching was introduced by Baker aiming at software maintenance and plagiarism detection [1, 2, 3], and has been extensively studied in the last decades (see a recent survey [28] and references therein).

The indexing problem for p-matching is to preprocess a p-string T of length n so that we can efficiently find the occurrences of substrings of T that p-match with a given pattern. Solutions proposed for this problem adapt and extend indexes initially devised for exact string pattern matching, e.g., parameterized suffix trees [1, 25, 2, 3], parameterized suffix arrays [8, 20, 4, 12], parameterized suffix trays [14], parameterized DAWGs [31], parameterized position heaps [9, 11, 13] and parameterized Burrows-Wheeler transforms (pBWTs) based indexes [16, 24, 18].

Among these indexes, pBWT-based indexes are the most space economic, consuming $n \lg |\Sigma_s \cup \Sigma_p| + O(n)$ bits [16] or $2n \lg |\Sigma_s \cup \Sigma_p| + 2n + o(n)$ bits with a simplified version proposed in [24]. Let σ_s and respectively σ_p be the numbers of distinct s-symbols and p-symbols that appear in T and $\sigma = \sigma_s + \sigma_p$. The pBWT-based index of T can be constructed via the parameterized suffix tree of T for which $O(n(\lg \sigma_s + \lg \sigma_p))$-time or randomized $O(n)$-time construction algorithms are known [25, 7, 26], but the intermediate memory footprint of $O(n \lg n)$ bits could be intolerable when it is significantly larger than the resulting index. Hashimoto et al. [19] showed how to compute the pBWT of [24] for T, under the assumption that $\Sigma_s \cup \Sigma_p = [0..O(\sigma)]$, in $O(n \lg \sigma)$ bits and $O(n \frac{\sigma_p \lg n}{\lg \lg n})$ time in an online manner while reading the symbols of T from right to left. Here we note that the work of [19] lacks details in terms of pBWT-based index construction because any pBWT-based index to date [16, 24, 18] requires additional data structures other than the pBWT, and the pBWTs alone does not seem to be enough to support p-matching queries efficiently.

In this paper, we refine the algorithm of [19] to work in $O(n \lg \sigma)$ bits and $O(n \frac{\lg \sigma_p \lg n}{\lg \lg n})$ time in a more general assumption that $\Sigma_s \cup \Sigma_p = [0..n^{O(1)}]$. While working in compact space, i.e., $O(n \lg \sigma)$ bits, it achieves $o(n\sigma_p)$ time when $\sigma_p = \omega(\lg n)$. This is of great interest because the time complexity of $o(n\sigma_p)$ has not been achieved in the construction for p-matching indexes even in the offline setting unless we resort to a fast construction algorithm for parameterized suffix trees using $O(n \lg n)$ bits. In particular, the currently best worst-case result for the direct construction of parameterized suffix arrays is $O(n\sigma_p)$ time and $O(n \lg n)$ bits of working space [12]. Since our online-built data structure for T can be used to compute the parameterized suffix array of T in $O(n \frac{\lg \sigma_p \lg n}{\lg \lg n})$ time, we obtain a new way to construct parameterized suffix arrays in $O(n \frac{\lg \sigma_p \lg n}{\lg \lg n})$ time and $O(n \lg \sigma)$ bits of working space.

We also show that our data structure can support backward search, a core procedure of BWT-based indexes, at any stage of the online construction, making it the first compact index for p-matching that can be constructed in compact space and even in an online manner. This cannot likely be achieved with the previous work [19] due to the lack of support for 2D range counting queries in the data structure it uses.

Our computational assumptions are as follows:

- We assume a standard Word-RAM model with word size $\Omega(\lg n)$.
- Each symbol in $(\Sigma_s \cup \Sigma_p)$ is represented by $O(\lg n)$ bits, namely, a symbol is from the universe $[0..n^{O(1)}]$.
- We can check membership for a given symbol ($\in \Sigma_s \cup \Sigma_p$) in Σ_s and Σ_p in $O(1)$ time, e.g., by having some flag bits or thresholds separating both alphabet sets.
- The order of two s-symbols can be determined in $O(1)$ time based on their bit representations.

An index of a p-string T for p-matching is to support, given a pattern w,

1. the *counting query* that asks to compute the number of occurrences of substrings in T that p-match with w and
2. the *locating query* that asks to compute the positions of these counted occurrences in T.

The number of occurrences returned for a locating query of w is the answer to the counting query of w. Since these occurrences can be at arbitrary positions of T in general, the time complexity for the locating query depends usually on the number of these occurrences. In contrast, most indexes based on the BWT can answer counting queries in time independent to this number, by levering the so-called *backward search*. By using backward search, our time complexities for both queries resemble those of other BWT-based indexes, with some additional logarithmic terms. In detail, our main result is as follows:

▶ **Theorem 1.** *For a p-string T of length n over an alphabet $(\Sigma_s \cup \Sigma_p)$ of size $n^{O(1)}$, an index of T for p-matching can be constructed online in $O(n\frac{\lg \sigma_p \lg n}{\lg\lg n})$ time and $O(n \lg \sigma)$ bits of space, where σ_s and respectively σ_p are the numbers of distinct s-symbols and p-symbols used in the p-string and $\sigma = \sigma_s + \sigma_p$. At any stage of the online construction, it can support the counting queries in $O(m\frac{\lg \sigma_p \lg n}{\lg\lg n})$ time, where m is the length of a given pattern for queries. By building an additional data structure of $O(\frac{n}{\Delta}\lg n)$ bits of space for a chosen parameter $\Delta \in \{1, 2, \ldots, n\}$ the locating queries can be supported in $O(m\frac{\lg \sigma_p \lg n}{\lg\lg n} + \mathsf{occ}\frac{\Delta \lg n}{\lg\lg n})$ time, where occ is the number of occurrences to be reported.*

We also obtain the following result for constructing the parameterized suffix array:

▶ **Theorem 2.** *For a p-string T of length n over an alphabet $(\Sigma_s \cup \Sigma_p)$ of size $n^{O(1)}$, the parameterized suffix array of T can be constructed in $O(n\frac{\lg \sigma_p \lg n}{\lg\lg n})$ time and $O(n \lg \sigma)$ bits of space, where σ_s and respectively σ_p are the numbers of distinct s-symbols and p-symbols used in the p-string and $\sigma = \sigma_s + \sigma_p$.*

2 Preliminaries

2.1 Basic notations and tools

We denote with $\lg = \log_2$ the logarithm with base two. An integer interval $\{i, i+1, \ldots, j\}$ is denoted by $[i..j]$, where $[i..j]$ represents the empty interval if $i > j$.

Let Σ be an ordered finite *alphabet*. An element of Σ^* is called a *string* over Σ. The length of a string w is denoted by $|w|$. The empty string ε is the string of length 0, that is, $|\varepsilon| = 0$. Let $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ and $\Sigma^k = \{x \in \Sigma^* \mid |x| = k\}$ for any non-negative integer k. The concatenation of two strings x and y is denoted by $x \cdot y$ or simply xy. When a string w is represented by the concatenation of strings x, y and z (i.e., $w = xyz$), then x, y and z are called a *prefix*, *substring*, and *suffix* of w, respectively. A substring x of w is called *proper* if $x \neq w$.

The i-th symbol of a string w is denoted by $w[i]$ for $1 \leq i \leq |w|$, and the substring of a string w that begins at position i and ends at position j is denoted by $w[i..j]$ for $1 \leq i \leq j \leq |w|$, i.e., $w[i..j] = w[i]w[i+1]\cdots w[j]$. For convenience, let $w[i..j] = \varepsilon$ if $j < i$; further let $w[..i] = w[1..i]$ and $w[i..] = w[i..|w|]$ denote abbreviations for the prefix of length i and the suffix starting at position i, respectively. For two strings x and y, let $\mathsf{lcp}(x, y)$ denote the length of the longest common prefix between x and y. We consider the lexicographic order over Σ^* by extending the strict total order $<$ defined on Σ: x is lexicographically smaller than y (denoted as $x < y$) if and only if either x is a proper prefix of y or $x[\mathsf{lcp}(x, y) + 1] < y[\mathsf{lcp}(x, y) + 1]$ holds. In this paper, we will ignore the former case since we mainly consider the lexicographic order between distinct strings that have a sentinel (end-marker) at the end of the strings so that x cannot be a proper prefix of y.

For any string w, character c, and position i ($1 \leq i \leq |w|$), $\mathsf{rank}_c(w, i)$ returns the number of occurrences of c in $w[..i]$ and $\mathsf{select}_c(w, i)$ returns the i-th occurrence of c in w. For $1 \leq i \leq j \leq |w|$, a *range minimum query* $\mathsf{RmQ}_w(i, j)$ asks for $\arg\min_{i \leq k \leq j}\{w[k]\}$. We also consider *find previous/next queries* $\mathsf{FPQ}_p(w, i)$ and $\mathsf{FNQ}_p(w, i)$, where p is a predicate either in the form of "c" (equal to c), "$< c$" (less than c) or "$\geq c$" (larger than or equal to c): $\mathsf{FPQ}_p(w, i)$ returns the largest position $j \leq i$ at which $w[j]$ satisfies the predicate p. Symmetrically, $\mathsf{FNQ}_p(w, i)$ returns the smallest position $j \geq i$ at which $w[j]$ satisfies the predicate p. For example with the integer string $w = [2, 5, 10, 6, 8, 3, 14, 5]$, $\mathsf{FNQ}_5(w, 4) = 8$, $\mathsf{FNQ}_6(w, 4) = 4$, $\mathsf{FPQ}_5(w, 4) = 2$, $\mathsf{FNQ}_{<5}(w, 4) = 6$, $\mathsf{FPQ}_{<5}(w, 4) = 1$, $\mathsf{FNQ}_{\geq 9}(w, 4) = 7$ and $\mathsf{FPQ}_{\geq 9}(w, 4) = 3$.

If the answer of $\mathsf{select}_c(w, i)$, $\mathsf{FPQ}_p(w, i)$ or $\mathsf{FNQ}_p(w, i)$ does not exist, it is just ignored. To handle this case of non-existence, we would use them in an expression with min or max: For example, $\max\{1, \mathsf{FPQ}_p(w, i)\}$ returns 1 if $\mathsf{FPQ}_p(w, i)$ does not exist.

Dynamic strings should support insertion/deletion of a symbol to/from any position as well as fast random access. We use the following result:

▶ **Lemma 3** ([29]). *A dynamic string of length n over an alphabet $[0..U]$ can be implemented while supporting random access, insertion, deletion,* rank *and* select *queries in $(n + o(n)) \lg U$ bits of space and $O(\frac{\lg n}{\lg \lg n})$ query and update times.*

Dynamic binary strings equipped with rank and select queries can be used as a building block for the dynamic wavelet matrix [6] of a string over an alphabet $[0..U]$ to support queries beyond rank and select. The idea is that each of the other queries can be simulated by performing one of the building block queries on every level of the wavelet matrix, which has $\lceil \lg U \rceil$ levels, cf. [32, Section 6.2.].

▶ **Lemma 4.** *A dynamic string of length n over an alphabet $[0..U]$ with $U = O(n)$ can be implemented while supporting random access, insertion, deletion,* rank, select, RmQ, FPQ *and* FNQ *queries in $(n + o(n))\lceil \lg U \rceil$ bits of space and $O(\frac{\lg U \lg n}{\lg \lg n})$ query and update times.*

2.2 Parameterized strings

Let Σ_s and Σ_p denote two disjoint sets of symbols. We call a symbol in Σ_s a *static symbol* (*s-symbol*) and a symbol in Σ_p a *parameter symbol* (*p-symbol*). A *parameterized string* (*p-string*) is a string over $(\Sigma_s \cup \Sigma_p)$. Let \$ be the smallest s-symbol, which will be used as an end-marker of p-strings. Let ∞ represent a symbol that is larger than any integer, and let $\mathbf{N}_\infty = \mathbf{N}_+ \cup \{\infty\}$ be the set of positive integers $\mathbf{N}_+$ including infinity (∞). Logically we assume that $\mathbf{N}_\infty \cap \Sigma_s = \emptyset$ and $(\mathbf{N}_\infty \cup \Sigma_s)$ is an ordered alphabet such that all s-symbols are smaller than any element in $\mathbf{N}_\infty$. For practical implementations, we require that s-symbols and integers can be distinguished in constant time (e.g., by shifting the ranges of the domains). Also, the conceptual symbol ∞ can be treated as the finite value $\sigma_p + 1$.

For any p-string w the *p-encoded string* $\langle w \rangle$ of w, also proposed as $\texttt{prev}_\infty(w)$ in [24], is the string in $(\mathbf{N}_\infty \cup \Sigma_s)^{|w|}$ such that

$$\langle w \rangle[i] = \begin{cases} w[i] & \text{if } w[i] \in \Sigma_s, \\ \infty & \text{if } w[i] \in \Sigma_p \text{ and } w[i] \text{ does not appear in } w[..i-1], \\ i - j & \text{otherwise,} \end{cases}$$

where j is the largest position in $[1..i-1]$ with $w[i] = w[j]$. To put in words, we transformed each occurrence of a p-symbol into the distance to the previous occurrence of the same p-symbol, or ∞ if it is the leftmost occurrence. Two p-strings x and y p-match if and only if $\langle x \rangle = \langle y \rangle$. On the one hand, the transformation from w to $\langle w \rangle$ is *prefix-consistent*, i.e., $\langle w \rangle = \langle wc \rangle[..|w|]$ for any symbol $c \in (\Sigma_s \cup \Sigma_p)$. On the other hand, $\langle w \rangle$ and $\langle cw \rangle[2..]$ differ if and only if $c \in \Sigma_p$ occurs in w. If it is the case, the leftmost occurrence h of c in w is the unique position such that $\langle w \rangle$ and $\langle cw \rangle[2..]$ differ with $\langle w \rangle[h] = \infty$ and $(\langle cw \rangle[2..])[h] = \langle cw \rangle[h+1] = h$, i.e., $h = \mathsf{select}_c(w, 1)$ and $h + 1 = \mathsf{select}_c(cw, 2)$.

For any p-string w, let $|w|_p$ denote the number of distinct p-symbols in w, i.e., $|w|_p = \mathsf{rank}_\infty(\langle w \rangle, |w|)$. We define a function π that maps a non-empty p-string $w \in (\Sigma_s \cup \Sigma_p)^+$ to an element in $(\Sigma_s \cup [1..|w|_p])$ such that $\pi(w)$ is $w[1]$ if $w[1]$ is an s-symbol; otherwise $\pi(w)$ is the number of *distinct* p-symbols in $w[..h+1]$, where $h+1$ is either the position of the second occurrence of $w[1]$ in w or $|w|$ if $w[1]$ is unique in w. More formally,

$$\pi(w) = \begin{cases} w[1] & \text{if } w[1] \in \Sigma_s, \\ |w[..h+1]|_p & \text{otherwise,} \end{cases}$$

where $h + 1 = \min\{|w|, \mathsf{select}_{w[1]}(w, 2)\}$. In the second case, $\pi(w)$ is considered to represent the rank of p-symbol $w[1]$ when p-symbols are sorted in increasing order of the leftmost positions they appear in $w[2..]$, considering the rank of p-symbols not in $w[2..]$ to be $|w|_p$. If $\mathsf{select}_{w[1]}(w, 2)$ exists, it holds that $h = \mathsf{select}_\infty(\langle w[2..] \rangle, \pi(w))$. For convenience, we extend the domain of π to handle the empty string with $\pi(\varepsilon) = \$$.

For two p-strings x and y, $\mathsf{lcp}^\infty(\langle x \rangle, \langle y \rangle)$ denotes the number of ∞'s in the longest common prefix of $\langle x \rangle$ and $\langle y \rangle$.

Our algorithm heavily relies on the properties of the p-string encoding and π. For any p-strings x and y, Table 1 shows a complete list of cases for $\mathsf{lcp}(\langle x \rangle, \langle y \rangle)$, $\mathsf{lcp}^\infty(\langle x \rangle, \langle y \rangle)$ and the lexicographic order between $\langle x \rangle$ and $\langle y \rangle$. The correctness immediately follows from the definition of the p-string encoding and π (see Figure 1 for illustrations). It is worth noting that Case (B3) is the only case in Cases (B1)-(B4) where we have $\langle y \rangle < \langle x \rangle$, i.e., the lexicographic order is changed after extension.

By Table 1, we have the following corollaries:

▶ **Corollary 5.** *For any p-strings x and y, $\mathsf{lcp}^\infty(\langle x \rangle, \langle y \rangle) \leq \mathsf{lcp}^\infty(\langle x[2..] \rangle, \langle y[2..] \rangle) + 1$.*

▶ **Corollary 6.** *For any p-strings x and y with $\pi(x) = \pi(y)$, $\langle x \rangle < \langle y \rangle$ if and only if $\langle x[2..] \rangle < \langle y[2..] \rangle$.*

▶ **Corollary 7.** *For any p-strings x and y (whether $\langle x[2..] \rangle < \langle y[2..] \rangle$ or $\langle x[2..] \rangle > \langle y[2..] \rangle$) with $\pi(x) \leq \mathsf{lcp}^\infty(\langle x[2..] \rangle, \langle y[2..] \rangle)$ and $\pi(x) < \pi(y)$, it holds that $\langle x \rangle < \langle y \rangle$. Note that $\pi(x)$ and/or $\pi(y)$ can be s-symbols.*

Table 1 All cases for $\mathsf{lcp}(\langle x\rangle, \langle y\rangle)$, $\mathsf{lcp}^{\infty}(\langle x\rangle, \langle y\rangle)$ and the lexicographic order between $\langle x\rangle$ and $\langle y\rangle$ for p-strings x and y over $(\Sigma_s \cup \Sigma_p)$ with $\lambda = \mathsf{lcp}(\langle x[2..]\rangle, \langle y[2..]\rangle) < \min\{|x|, |y|\}$, $e = \mathsf{lcp}^{\infty}(\langle x[2..]\rangle, \langle y[2..]\rangle)$ and $\langle x[2..]\rangle < \langle y[2..]\rangle$. On the one hand, a case starting with letter A assumes that at least one of $\pi(x)$ and $\pi(y)$ is in Σ_s, while on the other hand, a case starting with letter B assumes that none of $\pi(x)$ and $\pi(y)$ is in Σ_s. We let $h = \mathsf{select}_{x[1]}(x, 2) - 1$ in Case (B2) and $h' = \mathsf{select}_{y[1]}(y, 2) - 1$ in Case (B3), both of which always exist because the conditions of Cases (B2) and (B3) imply that $\pi(x) \neq \infty$ and $\pi(y) \neq \infty$, respectively.

cases	additional conditions	$\mathsf{lcp}(\langle x\rangle, \langle y\rangle)$	$\mathsf{lcp}^{\infty}(\langle x\rangle, \langle y\rangle)$	lexicographic order
(A1)	$\pi(x) \neq \pi(y)$	0	0	$\langle x\rangle < \langle y\rangle$ iff $\pi(x) < \pi(y)$
(A2)	$\pi(x) = \pi(y)$	$\lambda + 1$	e	$\langle x\rangle < \langle y\rangle$
(B1)	$\pi(x) = \pi(y) \leq e$	$\lambda + 1$	e	$\langle x\rangle < \langle y\rangle$
(B2)	$\pi(x) \leq e$ and $\pi(x) < \pi(y)$	h	$\pi(x)$	$\langle x\rangle < \langle y\rangle$
(B3)	$\pi(y) \leq e$ and $\pi(y) < \pi(x)$	h'	$\pi(y)$	$\langle y\rangle < \langle x\rangle$
(B4)	$e < \min\{\pi(x), \pi(y)\}$	$\lambda + 1$	$e + 1$	$\langle x\rangle < \langle y\rangle$

Table 2 An example of $\mathsf{R}_T^{-1}(i)$, LCP_T^{∞}, L_T and F_T for a p-string $T =$ `XYaZYXaZXZa$` with $\Sigma_s = \{$`a`$\}$ and $\Sigma_p = \{$`X`, `Y`, `Z`$\}$.

i	$T[i..]$	$\langle T[i..]\rangle$	$\mathsf{R}_T^{-1}(i)$	$\mathsf{LCP}_T^{\infty}[i]$	$\mathsf{L}_T[i]$	$\mathsf{F}_T[i]$	$\langle T[\mathsf{R}_T^{-1}(i)..]\rangle$
1	`XYaZYXaZXZa$`	`⍟a⍟35a432a$`	12	0	`a`	`$`	`$`
2	`YaZYXaZXZa$`	`⍟a⍟3⍟a432a$`	11	0	1	`a`	`a$`
3	`aZYXaZXZa$`	`a⍟⍟⍟a432a$`	7	0	2	`a`	`a⍟⍟2a$`
4	`ZYXaZXZa$`	`⍟⍟⍟a432a$`	3	2	2	`a`	`a⍟⍟⍟a432a$`
5	`YXaZXZa$`	`⍟⍟a⍟32a$`	10	0	2	1	`⍟a$`
6	`XaZXZa$`	`⍟a⍟32a$`	6	1	3	2	`⍟a⍟32a$`
7	`aZXZa$`	`a⍟⍟2a$`	2	2	3	2	`⍟a⍟3⍟a432a$`
8	`ZXZa$`	`⍟⍟2a$`	9	1	2	2	`⍟⍟a$`
9	`XZa$`	`⍟⍟a$`	5	2	3	3	`⍟⍟a⍟32a$`
10	`Za$`	`⍟a$`	1	3	`$`	3	`⍟⍟a⍟35a432a$`
11	`a$`	`a$`	8	2	`a`	2	`⍟⍟2a$`
12	`$`	`$`	4	2	`a`	3	`⍟⍟⍟a432a$`

Let T be a p-string that has the smallest s-symbol \$ as its end-marker, i.e., $T[|T|] =$ \$ and \$ does not appear anywhere else in T. The suffix rank function $\mathsf{R}_T : [1..|T|] \to [1..|T|]$ for T maps a position i $(1 \leq i \leq |T|)$ to the lexicographic rank of $\langle T[i..]\rangle$ in $\{\langle T[j..]\rangle \mid 1 \leq j \leq |T|\}$. Its inverse function $\mathsf{R}_T^{-1}(i)$ returns the starting position of the lexicographically i-th p-encoded suffix of T. [1]

The *parameterized Burrows-Wheeler Transform (pBWT)* of T is the string L_T of length $|T|$ over $(\Sigma_s \cup [1..|T|_p])$ such that $\mathsf{L}_T[i] = \pi(T[\mathsf{R}_T^{-1}(i) - 1..]))$, where we assume that $T[0..] =$ \$. Another string F_T of length $|T|$ is defined as $\mathsf{F}_T[i] = \pi(T[\mathsf{R}_T^{-1}(i)..])$. [2] Since $\{T[\mathsf{R}_T^{-1}(i)..] \mid 1 \leq i \leq |T|\} = \{T[\mathsf{R}_T^{-1}(i) - 1..] \mid 1 \leq i \leq |T|\}$ is equivalent to the set of all non-empty suffixes of T, F_T is a permutation of L_T.

[1] R_T^{-1} and R_T are essentially equivalent to parameterized suffix arrays and inverse parameterized suffix arrays, respectively.

[2] Previous studies [16, 24, 19] define pBWTs based on sorted cyclic rotations, but our suffix-based definition is more suitable for online construction to prevent unnecessary updates on F_T and L_T.

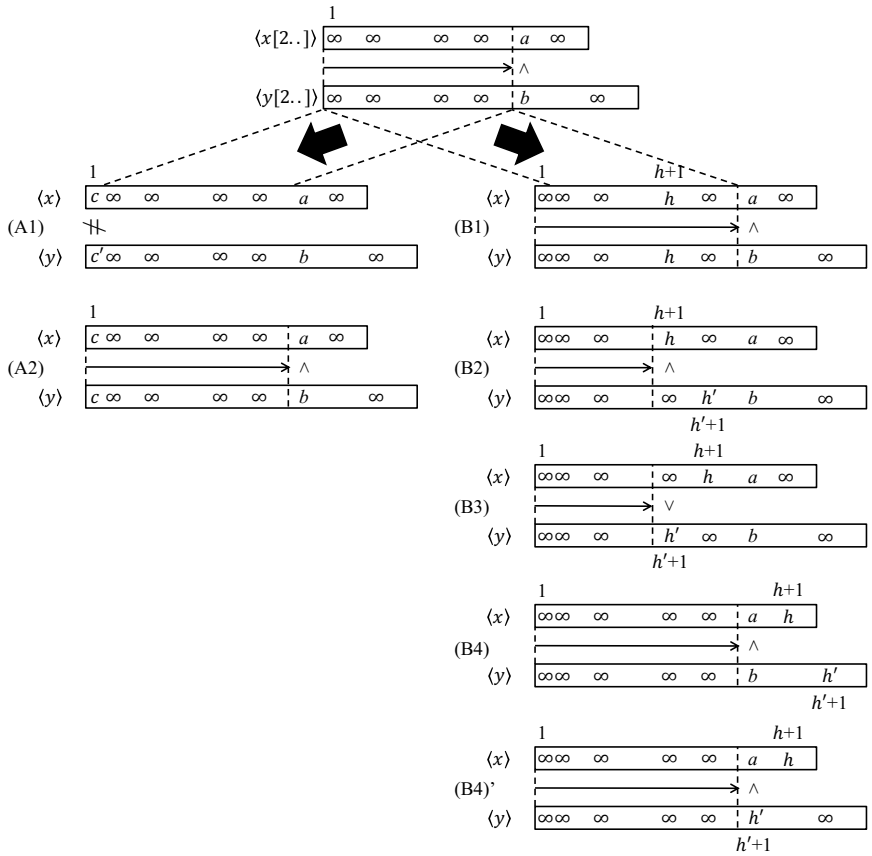


Figure 1 Illustrations for the cases of Table 1. The two bold diagonal arrows on the top separate the cases starting with letter A (left side) from the others, starting with B (right side). Each horizontal right-facing arrow represents the longest common prefix of two p-encoded strings, and the lexicographic order between them is determined by the following p-encoded symbols. Particularly, we let $a = \langle x[2..]\rangle[\lambda+1]$ and $b = \langle y[2..]\rangle[\lambda+1]$, where $\lambda = \mathsf{lcp}(\langle x[2..]\rangle, \langle y[2..]\rangle)$. Since $a < b$, it holds that $a \le \lambda$ while $b \le \lambda$ or $b = \infty$. For Case (B1), $h = \mathsf{select}_{x[1]}(x,2) - 1 = \mathsf{select}_{y[1]}(y,2) - 1$. For Case (B2)-(B4) and (B4)', $h = \mathsf{select}_{x[1]}(x,2) - 1$ and $h' = \mathsf{select}_{y[1]}(y,2) - 1$, some of which are not necessarily defined (when $\pi(x)$ or $\pi(y)$ is ∞) but assumed to be present in illustrations. Case (B4)' illustrates the case with $b = \infty$ and $h' = \lambda + 1$, which is included in Case (B4).

The so-called LF-mapping LF_T maps a position i to $\mathsf{R}_T(\mathsf{R}_T^{-1}(i) - 1)$ if $\mathsf{R}_T^{-1}(i) > 1$, and otherwise $\mathsf{R}_T(|T|) = 1$. By definition and Corollary 6, we have:

▶ **Corollary 8.** *For any p-string T and any integers i, j with $1 \le i < j \le |T|$, $\mathsf{LF}_T(i) < \mathsf{LF}_T(j)$ if $\mathsf{L}_T[i] = \mathsf{L}_T[j]$.*

Thanks to Corollary 8, it holds that $\mathsf{LF}_T(i) = \mathsf{select}_c(\mathsf{F}_T, \mathsf{rank}_c(\mathsf{L}_T, i))$, where $c = \mathsf{L}_T[i]$. The inverse function FL_T of LF_T can be computed by $\mathsf{FL}_T(i) = \mathsf{select}_c(\mathsf{L}_T, \mathsf{rank}_c(\mathsf{F}_T, i))$, where $c = \mathsf{F}_T[i]$.

Let LCP_T^∞ be the string of length $|T|$ such that $\mathsf{LCP}_T^\infty[1] = 0$ and $\mathsf{LCP}_T^\infty[i] = \mathsf{lcp}^\infty(\langle T[\mathsf{R}_T^{-1}(i-1)..]\rangle, \langle T[\mathsf{R}_T^{-1}(i)..]\rangle)$ for every $1 < i \le |T|$. An example of all explained arrays is given in Table 2.

3 Online construction algorithm

To construct our index for p-matching online, we maintain F_T, L_T, and LCP^∞_T with dynamic data structures while prepending a symbol to the current p-string T. The details of the data structures will be presented in Subsection 3.3. In what follows, we focus on a single step of updating T to $\hat{T} = cT$ for some symbol c in $\Sigma_s \cup \Sigma_p$. Note that F_T, L_T and LCP^∞_T are strongly related to the sorted p-encoded suffixes of a p-string and $\hat{T} = cT$ is the only suffix that was not in the suffixes of T. Let $k = \mathsf{R}_T(1)$ and $\hat{k} = \mathsf{R}_{\hat{T}}(1)$. In order to deal with the new emerging suffix $\hat{T}$, we compute the lexicographic rank $\hat{k}$ of $\langle\hat{T}\rangle$ among the non-empty p-encoded suffixes of $\hat{T}$. Then $\mathsf{F}_{\hat{T}}$ and $\mathsf{L}_{\hat{T}}$ can be obtained by replacing \$ in L_T at k by $\pi(\hat{T})$ and inserting \$ and $\pi(\hat{T})$ into the $\hat{k}$-th position of L_T and F_T, respectively. In Subsection 3.1, we propose our algorithm to compute $\hat{k}$. For updating LCP^∞, we have to compute the lcp^∞-values for $\langle\hat{T}\rangle$ with its lexicographically adjacent p-encoded suffixes, which will be treated in Subsection 3.2.

3.1 How to compute $\hat{k}$

Unlike previous work [19] that computes $\hat{k}$ by counting the number of p-encoded suffixes that are lexicographically smaller than $\langle\hat{T}\rangle$, we get $\hat{k}$ indirectly by computing the rank of a lexicographically closest (smaller or larger) p-encoded suffix to $\langle\hat{T}\rangle$. The lexicographically smaller (resp. larger) closest element in $\{\langle T[i..]\rangle \mid 1 \le i \le |T|\}$ to $\langle\hat{T}\rangle$ is called the *p-pred* (resp. *p-succ*) of $\langle\hat{T}\rangle$. If the lexicographic rank of the p-pred (resp. p-succ) of $\langle\hat{T}\rangle$ is k_- (resp. k_+), then it holds that $\hat{k} = k_+ = k_- + 1$.

We start with the easy case that the prepended symbol c is an s-symbol.

▶ **Lemma 9.** *Let $\hat{T} = cT$ be a p-string with $c \in \Sigma_s$. If $p := \mathsf{FPQ}_c(\mathsf{L}_T, k)$ exists, the rank k_- of the p-pred of $\hat{T}$ is $\mathsf{LF}_T(p)$. Otherwise, $k_- = \mathsf{select}_b(\mathsf{F}_T, \mathsf{rank}_b(\mathsf{F}_T, |T|))$, where b is the largest s-symbol that appears in T and is smaller than c.*

Proof. By Case (A2) of Table 1, the lexicographic order of p-encoded suffixes starting with c does not change by removing their first characters, which are all c. If p exists, $\langle T[\mathsf{R}_T^{-1}(p)..]\rangle$ is the lexicographically smaller closest p-encoded suffix to $\langle T\rangle$ that is preceded by c. Hence, $\langle T[\mathsf{R}_T^{-1}(\mathsf{LF}_T(p))..]\rangle = \langle c(T[\mathsf{R}_T^{-1}(p)..])\rangle$ is the p-pred of $\langle cT\rangle = \langle\hat{T}\rangle$, which means that $k_- = \mathsf{LF}_T(p)$.

If p does not exist, it implies that $\langle\hat{T}\rangle$ is the lexicographically smallest p-encoded suffix that starts with c. Since $\langle\hat{T}\rangle$ lexicographically comes right after the p-encoded suffixes starting with an s-symbol smaller than c, k_- is the last occurrence of b in F_T, that is, $k_- = \mathsf{select}_b(\mathsf{F}_T, \mathsf{rank}_b(\mathsf{F}_T, |T|))$. ◀

In the rest of this subsection, we consider the case that c is a p-symbol. If T contains no p-symbol, it is clear that $k_- = |T|$. Hence, in what follows, we assume that there is a p-symbol in T.

Since $\langle\hat{T}\rangle$ has the longest lcp-value with its p-pred or p-succ among all the suffixes of T, we search for such p-encoded suffixes of T using the following lemmas to leverage the information stored in LCP^∞_T.

▶ **Lemma 10.** *Given two positions i and j with $1 \le i < j \le |T|$,*

$$\mathsf{lcp}^\infty(\langle T[\mathsf{R}_T^{-1}(i)..]\rangle, \langle T[\mathsf{R}_T^{-1}(j)..]\rangle) = \mathsf{RmQ}_{\mathsf{LCP}^\infty_T}(i+1, j).$$

Algorithm 1 Algorithm to compute the maximal interval $[l..r]$ such that $\mathsf{lcp}^\infty(\langle T[\mathsf{R}_T^{-1}(i)..]\rangle, \langle T[\mathsf{R}_T^{-1}(j)..]\rangle) \geq e$ for any $j \in [l..r]$. It returns $[i..i]$ if $e > |T[\mathsf{R}_T^{-1}(i)..]|_p$.

```
1 Function GetMI(i, e):
2     l ← max{1, FPQ_{<e}(LCP^∞_T, i)};
3     r ← min{|T|, FNQ_{<e}(LCP^∞_T, i + 1) − 1};
4     return [l..r];
```

Proof. By Lemma 1 of [22], $\mathsf{lcp}(x,z) = \min\{\mathsf{lcp}(x,y), \mathsf{lcp}(y,z)\}$ for any strings $x < y < z$, and thus, $\mathsf{lcp}^\infty(x,z) = \min\{\mathsf{lcp}^\infty(x,y), \mathsf{lcp}^\infty(y,z)\}$. Since LCP_T^∞ holds the lcp^∞-values of lexicographically adjacent p-encoded suffixes, we get $\mathsf{lcp}^\infty(\langle T[\mathsf{R}_T^{-1}(i)..]\rangle, \langle T[\mathsf{R}_T^{-1}(j)..]\rangle) = \min\{\mathsf{LCP}_T^\infty[g]\}_{g=i+1}^{j} = \mathsf{RmQ}_{\mathsf{LCP}_T^\infty}(i+1, j)$ by applying the previous argument successively. ◀

▶ **Lemma 11.** *For given $i, e \in [1..n]$, if $e \leq |T[\mathsf{R}_T^{-1}(i)..]|_p$, then Algorithm 1 computes the maximal interval $[l..r]$ such that $\mathsf{lcp}^\infty(\langle T[\mathsf{R}_T^{-1}(i)..]\rangle, \langle T[\mathsf{R}_T^{-1}(j)..]\rangle) \geq e$ for any $j \in [l..r]$. If $e > |T[\mathsf{R}_T^{-1}(i)..]|_p$, then Algorithm 1 returns $[i..i]$.*

▶ **Lemma 12.** *Algorithm 2 correctly returns $\hat{k}$.*

Proof.

Outline. Let $h_i = \mathsf{select}_\infty(\langle T\rangle, i)$ for any $1 \leq i \leq \min\{|T|_p, \pi(\hat{T})\}$, and $h_i = |T|+1$ for any $i > \min\{|T|_p, \pi(\hat{T})\}$. Also let $\lambda = \max\{\mathsf{lcp}(\langle \hat{T}\rangle, \langle T[i..]\rangle) \mid 1 \leq i \leq |T|\}$. Although Algorithm 2 does not intend to compute the exact value of λ, it checks if λ falls in $[h_e..h_{e+1}]$ in decreasing order of e starting from $\min\{\pi(\hat{T}), \max\{\mathsf{LCP}_T^\infty[k], \mathsf{LCP}_T^\infty[k+1]\}\}$. One of the necessary conditions to have $\mathsf{lcp}(\langle \hat{T}\rangle, \langle T[i..]\rangle) > h_e$ is that $\mathsf{lcp}(\langle T\rangle, \langle T[i+1..]\rangle) \geq h_e$, or equivalently $\mathsf{lcp}^\infty(\langle T\rangle, \langle T[i+1..]\rangle) \geq e$. Line 2 computes the maximal interval $[l..r]$ that represents the ranks of the p-encoded suffixes having an lcp^∞-value larger than or equal to e. The basic idea is to find a p-encoded suffix in $\{\langle T[\mathsf{R}_T^{-1}(p)..]\rangle\}_{p=l}^{r}$ that comes closest to $\langle \hat{T}\rangle$ when extended by adding its preceding symbol. Here let us call $\langle T[\mathsf{R}_T^{-1}(p)-1..]\rangle$ the *extended suffix* of $\langle T[\mathsf{R}_T^{-1}(p)..]\rangle$. When Algorithm 2 decreases e to the value with $\lambda \in [h_e+1..h_{e+1}]$, $\hat{k}$ is returned in one of the if-then-blocks at Lines 4, 5, 8 and 13.

If $\mathsf{lcp}(\langle \hat{T}\rangle, \langle T[i..]\rangle) = h_{\hat{e}}$ for an integer $\hat{e}$, there are two possible scenarios (see Figure 2 for an illustration):

(H1) $\mathsf{lcp}^\infty(\langle T\rangle, \langle T[i+1..]\rangle) \geq \hat{e}$ and either $\pi(\hat{T}) > \pi(T[i..]) = \hat{e}$ or $\pi(T[i..]) > \pi(\hat{T}) = \hat{e}$, and

(H2) $\mathsf{lcp}(\langle T\rangle, \langle T[i+1..]\rangle) = h_{\hat{e}} - 1$ and both $\pi(\hat{T})$ and $\pi(T[i..])$ are at least $\hat{e}$.

Case (H1) is processed in one of the if-then-blocks at Lines 6 and 18 when $e = \hat{e}$. while Case (H2) at Lines 4, 5, 8 and 13 when $e = \hat{e} - 1$. Note that p-encoded suffix of Case (H1) is never farther from $\langle \hat{T}\rangle$ than that of Case (H2) because the lexicographic order between $\langle \hat{T}\rangle$ and $\langle T[i..]\rangle$ is determined by ∞ and $h_{\hat{e}}$ at $h_{\hat{e}}+1$ in Case (H1), while it is by ∞ and something smaller than $h_{\hat{e}}$ in Case (H2). Since Algorithm 2 processes Case (H1) first, it guarantees that the algorithm finds the closer one first.

In what follows, we delve into the details of each code block.

If-then-block at Line 3. The case with $e = \pi(\hat{T})$ is treated differently than other cases in the if-then-block at Line 3 since $h_{\pi(\hat{T})}$ is the unique position where $\langle T\rangle[h_{\pi(\hat{T})}] = \infty$ turns into $\langle \hat{T}\rangle[h_{\pi(\hat{T})}+1] = h_{\pi(\hat{T})}$. For a p-encoded suffix $\langle T[\mathsf{R}_T^{-1}(q')..]\rangle \in \{\langle T[\mathsf{R}_T^{-1}(p)..]\rangle\}_{p=l}^{r}$, having $\mathsf{L}_T[q'] = \pi(\hat{T})$ is necessary and sufficient for its extended suffix $\langle T[\mathsf{R}_T^{-1}(q')-1..]\rangle$ to have an lcp-value larger than $h_{\pi(\hat{T})}$ with $\langle \hat{T}\rangle$. By Corollary 6, p-encoded suffixes satisfying this condition must preserve their lexicographic order after extension, and hence, it is enough to search for the closest one ($q \leftarrow \mathsf{FPQ}_e(\mathsf{L}_T, k)$ or $q \leftarrow \mathsf{FNQ}_e(\mathsf{L}_T, k)$) to $\langle T\rangle$ and compute the rank of its

extended suffix by $\mathsf{LF}_T(q)$. If Lines 4 and 5 do not return a value, we know that $\lambda \leq h_{\pi(\hat{T})}$. The if-block at Line 6 checks if there exists a p-encoded suffix $\langle T[i{+}1..]\rangle$ that satisfies the condition of Case (H1) to be $\mathsf{lcp}(\langle\hat{T}\rangle, \langle T[i..]\rangle) = h_{\pi(\hat{T})}$. It is enough to find one $\langle T[\mathsf{R}_T^{-1}(q)..]\rangle$ with $\mathsf{L}_T[q] > \pi(\hat{T})$ because it is necessary and sufficient to have $\mathsf{lcp}(\langle\hat{T}\rangle, \langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle) = h_{\pi(\hat{T})}$ and $\langle\hat{T}\rangle[h_{\pi(\hat{T})}+1] = \infty \neq h_{\pi(\hat{T})} = \langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle[h_{\pi(\hat{T})}+1]$. Note that there could be two or more p-encoded suffixes that satisfy the condition and their lexicographic order may change by extension. In the then-block at Line 6, the algorithm computes the rank of the lexicographically smallest p-encoded suffix that has an lcp^∞-value larger than $\pi(\hat{T})$ with $\langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle = \langle T[\mathsf{R}_T^{-1}(\mathsf{LF}_T(q))..]\rangle$, which is the p-succ of $\langle\hat{T}\rangle$ in this case.

Precondition to enter Line 7. The case with $e \neq \pi(\hat{T})$ is processed in the else-block at Line 7. Here, it is good to keep in mind that when we enter this else-block, $\mathsf{lcp}^\infty(\langle T\rangle, \langle T[i..]\rangle) \leq e$ or $\pi(T[i-1..]) \leq e$ holds for any proper suffix $T[i..]$ of T, since otherwise $\hat{k}$ must be reported in a previous round of the foreach loop.

If-then-block at Line 8. When the if-condition at Line 8 holds, $\langle T[\mathsf{R}_T^{-1}(q)..]\rangle$ is the lexicographically smaller closest p-encoded suffix to $\langle T\rangle$ such that $\mathsf{lcp}^\infty(\langle\hat{T}\rangle, \langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle) \geq e+1$, or equivalently $\mathsf{lcp}(\langle\hat{T}\rangle, \langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle) > h_e$. Note that $\langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle < \langle\hat{T}\rangle$ must hold, since otherwise, $T[\mathsf{R}_T^{-1}(q)-1..]$ and $\langle\hat{T}\rangle$ would fall into Case (B3) with $\mathsf{lcp}^\infty(\langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle, \langle\hat{T}\rangle) = \pi(\hat{T})$, and $\hat{k}$ should be reported at Line 6 in a previous round. For any p-encoded suffix in $\{\langle T[\mathsf{R}_T^{-1}(p)..]\rangle\}_{p=q+1}^{k-1}$ its extended suffix is lexicographically smaller than $\langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle$ due to Corollary 7, and never closer to $\langle\hat{T}\rangle$ than $\langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle$. If $|T[\mathsf{R}_T^{-1}(q)..]|_p \geq e+1$, the interval $[l'..r']$ computed at Line 9 is the maximal interval such that every p-encoded suffix in $\{\langle T[\mathsf{R}_T^{-1}(p)..]\rangle\}_{p=l'}^{r'}$ shares the common prefix of length $h' := \mathsf{select}_\infty(\langle T[\mathsf{R}_T^{-1}(q)..]\rangle, e+1)$ with $\langle T[\mathsf{R}_T^{-1}(q)..]\rangle$. In the case with $|T[\mathsf{R}_T^{-1}(q)..]|_p = e$, `GetMI(`$q$`,` $e+1$`)` returns $[q..q]$ and let us define h' to be $|T[\mathsf{R}_T^{-1}(q)..]|$.

Since any $\langle T[i..]\rangle \in \{\langle T[\mathsf{R}_T^{-1}(p)..]\rangle\}_{p=1}^{l'-1}$ has an lcp-value smaller than h' with $\langle T[\mathsf{R}_T^{-1}(q)..]\rangle$, it follows from Table 1 that $\langle T[i-1..]\rangle < \langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle$. Also, for any $\langle T[i..]\rangle \in \{\langle T[\mathsf{R}_T^{-1}(p)..]\rangle\}_{p=k+1}^{|T|}$, the aforementioned precondition to enter the else-block at Line 7 implies that $\langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle < \langle T[i-1..]\rangle < \langle\hat{T}\rangle$ cannot happen: If $\mathsf{lcp}^\infty(\langle T\rangle, \langle T[i..]\rangle) < e$ or $\pi(T[i-1..]) \leq e$, then $\mathsf{lcp}(\langle\hat{T}\rangle, \langle T[i-1..]\rangle) \leq h_e < \mathsf{lcp}(\langle\hat{T}\rangle, \langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle)$ leads to the conclusion. For the remaining case with $\mathsf{lcp}^\infty(\langle T\rangle, \langle T[i..]\rangle) = e$ and $\pi(T[i-1..]) > e$, it holds that $\langle\hat{T}\rangle < \langle T[i-1..]\rangle$ due to Case (B4) of Table 1.

In the previous paragraph we have confirmed that the lcp-value between $\langle\hat{T}\rangle$ and its p-pred is at most h', which implies that the p-pred is the largest p-encoded suffix that is prefixed by $x := \langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle[..h']$. If $q' \leftarrow \mathsf{FPQ}_{\geq e+2}(\mathsf{L}_T, r')$ computed at Line 10 is in $[l'..r']$, $\langle T[\mathsf{R}_T^{-1}(q')-1..]\rangle = \langle T[\mathsf{LF}_T(q')..]\rangle$ is prefixed by $x\cdot\infty$ and the p-pred is the largest p-encoded suffix that is prefixed by $x\cdot\infty$, which can be computed by $\max \mathtt{GetMI}(\mathsf{LF}_T(q'), e+2)$ because $\langle T[\mathsf{R}_T^{-1}(q')-1..]\rangle[..h'+1] = \langle T[\mathsf{R}_T^{-1}(\mathsf{LF}_T(q')..]\rangle[..h'+1] = x\cdot\infty$ contains exactly $e+2$ ∞'s. If $q' \notin [l'..r']$, the p-pred is the largest p-encoded suffix that is prefixed by $x\cdot h'$ (or $x\cdot\$$ for the case with $|T[\mathsf{R}_T^{-1}(q)..]|_p = e$), which is $\langle T[\mathsf{R}_T^{-1}(\mathsf{LF}_T(q))..]\rangle$.

If-then-block at Line 13. When the if-condition at Line 13 holds, $\langle T[\mathsf{R}_T^{-1}(q)..]\rangle$ is the lexicographically larger closest p-encoded suffix to $\langle T\rangle$ such that $\mathsf{lcp}^\infty(\langle\hat{T}\rangle, \langle T[\mathsf{R}_T^{-1}(q){-}1..]\rangle) \geq e+1$, or equivalently $\mathsf{lcp}(\langle\hat{T}\rangle, \langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle) > h_e$. Note that $\langle\hat{T}\rangle < \langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle$ must hold, since otherwise, $\langle\hat{T}\rangle$ and $T[\mathsf{R}_T^{-1}(q)-1..]$ would fall into Case (B3) with $\mathsf{lcp}(\langle\hat{T}\rangle, \langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle) = h_{\hat{e}}$ for some $\hat{e} > e$, and $\hat{k}$ should be reported at Line 18 of a previous round. For any p-encoded suffix in $\{\langle T[\mathsf{R}_T^{-1}(p)..]\rangle\}_{p=k+1}^{q-1}$ its extended suffix is lexicographically smaller than $\langle\hat{T}\rangle$ due to Corollary 7, and never come lexicographically between $\langle\hat{T}\rangle$ and $\langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle$. If $|T[\mathsf{R}_T^{-1}(q)..]|_p \geq e+1$, the interval $[l'..r']$ computed

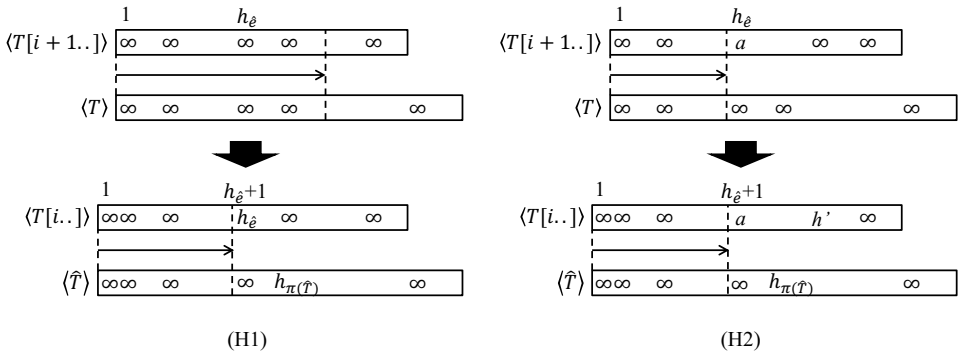


Figure 2 Illustration for Cases (H1) and (H2) in which $\mathsf{lcp}(\langle\hat{T}\rangle, \langle T[i..]\rangle) = h_{\hat{e}}$ for an integer $\hat{e}$. The left part shows one of the situations for Case (H1) where $\mathsf{lcp}^{\infty}(\langle T\rangle, \langle T[i+1..]\rangle) \geq \hat{e}$ and $\pi(\hat{T}) > \pi(T[i..]) = \hat{e}$. The right part shows one of the situations for Case (H2) where $\mathsf{lcp}(\langle T\rangle, \langle T[i+1..]\rangle) = h_{\hat{e}} - 1$, $\pi(\hat{T}) > h_{\hat{e}}$ and $\pi(T[i..]) > h_{\hat{e}}$.

at Line 14 is the maximal interval such that every p-encoded suffix in $\{\langle T[\mathsf{R}_T^{-1}(p)..]\rangle\}_{p=l'}^{r'}$ shares the common prefix of length $h' := \mathsf{select}_{\infty}(\langle T[\mathsf{R}_T^{-1}(q)..]\rangle, e+1)$ with $\langle T[\mathsf{R}_T^{-1}(q)..]\rangle$. In the case with $|T[\mathsf{R}_T^{-1}(q)..]|_p = e$, `GetMI(`$q$, $e+1$`)` returns $[q..q]$ and let us define h' to be $|T[\mathsf{R}_T^{-1}(q)..]|$.

For any $\langle T[i..]\rangle \in \{\langle T[\mathsf{R}_T^{-1}(p)..]\rangle\}_{p=1}^{k-1}$, the aforementioned precondition to enter the else-block at Line 7 implies that $\langle T[i-1..]\rangle < \langle\hat{T}\rangle$ because Case (B3) of Table 1 cannot hold under the condition of $\mathsf{lcp}^{\infty}(\langle T\rangle, \langle T[i..]\rangle) \leq e$ or $\pi(T[i-1..]) \leq e$. For any $\langle T[i..]\rangle \in \{\langle T[\mathsf{R}_T^{-1}(p)..]\rangle\}_{p=r'+1}^{|T|}$, we show that $\langle\hat{T}\rangle < \langle T[i-1..]\rangle < \langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle$ cannot happen: Note that $\mathsf{lcp}^{\infty}(\langle T[\mathsf{R}_T^{-1}(q)..]\rangle, \langle T[i..]\rangle) \leq e$ by definition, and $\mathsf{lcp}^{\infty}(\langle T[\mathsf{R}_T^{-1}(q)..]\rangle, \langle T[i..]\rangle) < e$ implies $\mathsf{lcp}^{\infty}(\langle T\rangle, \langle T[i..]\rangle) = \mathsf{lcp}^{\infty}(\langle T[\mathsf{R}_T^{-1}(q)..]\rangle, \langle T[i..]\rangle)$. If $\mathsf{lcp}^{\infty}(\langle T[i..]\rangle, \langle T[\mathsf{R}_T^{-1}(q)..]\rangle) < e$ or $\pi(T[i-1..]) \leq e$, $\mathsf{lcp}(\langle\hat{T}\rangle, \langle T[i-1..]\rangle) \leq h_e < \mathsf{lcp}(\langle\hat{T}\rangle, \langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle)$ leads to the conclusion. For the remaining case with $\mathsf{lcp}^{\infty}(\langle T[i..]\rangle, \langle T[\mathsf{R}_T^{-1}(q)..]\rangle) = e$ and $\pi(T[i-1..]) > e$, it holds that $\langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle < \langle T[i-1..]\rangle$ due to Case (B4) of Table 1.

In the previous paragraph we have confirmed that the lcp-value between $\langle\hat{T}\rangle$ and its p-succ is at most h', which implies that the p-succ is the smallest p-encoded suffix that is prefixed by $x := \langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle[..h']$. If $q' \leftarrow \mathsf{FNQ}_{e+1}(\mathsf{L}_T, l')$ computed at Line 15 is in $[l'..r']$, the p-succ is the smallest p-encoded suffix that is prefixed by $x \cdot h'$ (or $x \cdot \$$ for the case with $|T[\mathsf{R}_T^{-1}(q)..]|_p = e$), which is $\langle T[\mathsf{R}_T^{-1}(\mathsf{LF}_T(q'))..]\rangle$. If $q' \notin [l'..r']$, the p-succ is the smallest p-encoded suffix that is prefixed by $x \cdot \infty$, which can be computed by $\min$ `GetMI(`$\mathsf{LF}_T(q), e+2$`)` because $\langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle[..h'+1] = \langle T[\mathsf{R}_T^{-1}(\mathsf{LF}_T(q))..]\rangle[..h'+1] = x \cdot \infty$ contains exactly $e+2$ ∞'s.

If-then-block at Line 18. When we enter the if-then-block at Line 18, it is guaranteed that $\lambda \leq h_e$. In order to check if there exists a p-encoded suffix $\langle T[i+1..]\rangle$ that satisfies the condition of Case (H1) to be $\mathsf{lcp}(\langle\hat{T}\rangle, \langle T[i..]\rangle) = h_e$, the algorithm computes $q \leftarrow \mathsf{FPQ}_e(\mathsf{L}_T, r)$. If $q \in [l..r]$, $\langle T[\mathsf{R}_T^{-1}(q)..]\rangle$ is the lexicographically largest p-encoded suffix that satisfies the condition, and by Corollary 6, its extended suffix $\langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle$ must be the largest one to have $\mathsf{lcp}(\langle\hat{T}\rangle, \langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle) = h_e$. Therefore, $\langle T[\mathsf{R}_T^{-1}(q)-1..]\rangle$ is the p-pred of $\langle\hat{T}\rangle$, and $\hat{k} = 1 + \mathsf{LF}_T(q)$. ◀

Algorithm 2 Algorithm to compute $\hat{k}$.

```
1  foreach e ← min{π(T̂), max{LCP∞_T[k], LCP∞_T[k+1]}} down to 1 do
2      [l..r] ← GetMI(k, e);
3      if e = π(T̂) then
4          if (q ← FPQ_e(L_T, k)) ∈ [l..r] then return 1 + LF_T(q) ;
5          if (q ← FNQ_e(L_T, k)) ∈ [l..r] then return LF_T(q) ;
6          if (q ← FNQ_{≥e+1}(L_T, l)) ∈ [l..r] then return min GetMI(LF_T(q), e + 1) ;
7      else
8          if (q ← FPQ_{≥e+1}(L_T, k)) ∈ [l..r] then
9              [l'..r'] ← GetMI(q, e + 1);
10             q' ← FPQ_{≥e+2}(L_T, r');
11             if q' ∈ [l'..r'] then return 1 + max GetMI(LF_T(q'), e + 2) ;
12             else return 1 + LF_T(q) ;
13         if (q ← FNQ_{≥e+1}(L_T, k)) ∈ [l..r] then
14             [l'..r'] ← GetMI(q, e + 1);
15             q' ← FNQ_{e+1}(L_T, l');
16             if q' ∈ [l'..r'] then return LF_T(q') ;
17             else return min GetMI(LF_T(q), e + 2) ;
18         if (q ← FPQ_e(L_T, r)) ∈ [l..r] then return 1 + LF_T(q) ;
```

3.2 How to maintain $\mathsf{LCP}^{\infty}_{T}$

Suppose that we have $k = \mathsf{R}_T(1)$, $\hat{k} = \mathsf{R}_{\hat{T}}(1)$, L_T, F_T. We show how to compute the lcp^{∞}-values of $\langle \hat{T} \rangle$ with its p-pred $\langle T[\mathsf{R}_T^{-1}(\hat{k}) - 1..] \rangle$ and p-succ $\langle T[\mathsf{R}_T^{-1}(\hat{k})..] \rangle$ to maintain $\mathsf{LCP}^{\infty}_{T}$.

We focus on $\mathsf{lcp}^{\infty}(\langle \hat{T} \rangle, \langle T[\mathsf{R}_T^{-1}(\hat{k})..] \rangle)$ because the other one can be treated similarly. We apply Table 1 by setting $x = \hat{T}$ and $y = T[\mathsf{R}_T^{-1}(\hat{k})..]$ if $k < \mathsf{FL}_T(\hat{k})$ (otherwise we swap their roles for x and y). In order to get $\mathsf{lcp}^{\infty}(\langle x \rangle, \langle y \rangle)$, all we need are $\pi(x) = \pi(\hat{T})$, $\pi(y) = \mathsf{F}[\hat{k}]$ and $e = \mathsf{lcp}^{\infty}(\langle x[2..] \rangle, \langle y[2..] \rangle)$. For the computation of e we use Lemma 10, i.e., $e = \mathsf{lcp}^{\infty}(\langle x[2..] \rangle, \langle y[2..] \rangle) = \mathsf{RmQ}_{\mathsf{LCP}^{\infty}_{T}}(k+1, \mathsf{FL}_T(\hat{k}))$.

3.3 Dynamic data structures and analysis

Let σ_s and respectively $\sigma_p := |T|_p$ be the numbers of distinct s-symbols and p-symbols that appear in T and $\sigma = \sigma_s + \sigma_p$. We consider constructing F_T, L_T and $\mathsf{LCP}^{\infty}_{T}$ online for a p-string T of length n over an alphabet $(\Sigma_s \cup \Sigma_p)$ of size $n^{O(1)}$. To this end, we introduce data structures for implementing our algorithm, which we presented in the previous subsections.

To obtain our claimed space bounds of $O(n \lg \sigma)$ bits, we maintain a naming function that maps the set of distinct s-symbols indexed in the pBWT from Σ_s to the range of ranks $[1..\sigma_s]$. By doing so, we can represent and store each s-symbol in the pBWT by its rank. Thus, each s-symbol in F_T and L_T consumes $O(\lg \sigma_s)$ bits instead of $O(\lg n)$ bits.

In the following we present two alternative implementations for the naming function. The first one imposes a new order on the s-symbols such that the computed F_T and L_T arrays may arrange s-symbols differently than the standard pBWT built on the plain s-symbols. The second one keeps the order of the s-symbols, but needs an additional scan of the input text T to determine the order *prior to* the computation of the pBWT.

1. Our first approach updates the naming function in the same online fashion as computing the pBWT. To this end, we represent the naming function by a dynamic associative array using $O(\sigma_s \lg n) = O(n \lg \sigma_s)$ bits of space and taking $O(\frac{\lg \sigma_s}{\lg \lg \sigma_s})$ operation time like [21]. Setting initially $\sigma_s = 0$, we update σ_s and the naming function whenever we read a new s-symbol. In detail, when we read a new s-symbol with integer representation x with a yet-undefined rank (meaning it is not yet stored in the associative array), we increment σ_s by one, and insert the integer key x associated with the value σ_s into the associative array. The assignment of the ranks is done in a *first come first served* order, meaning that the order of two s-symbols is determined by whose rightmost occurrence in the text has the larger text position (since we build the pBWT online reading symbols from the right end). Changing the alphabet order neither invalidates the LF-mapping nor the backward search.[3] However, there is a subtle caveat in that this approach needs to know σ_s in advance (or at least a close upper bound σ'_s with $\sigma'_s = c\sigma$ for a constant $c \geq 1$). Otherwise, we need to spend some extra time on reconstructing all dynamic data structures working with s-symbols. That is because, for steady increases of σ_s, there is point where we no longer can represent a s-symbol rank in just $\lceil \lg \sigma_s \rceil$ bits, but need $1 + \lceil \lg \sigma_s \rceil$ bits instead. Instead of rebuilding all data structures storing s-symbol ranks on every increase of $\lceil \lg \sigma_s \rceil$, we initially accommodate each s-symbol with $2\lceil \lg \sigma_s \rceil$ bits, and double this space whenever necessary.[4] By doing so, the number of bits per s-symbol increases from constant to $\Theta(\lg \sigma_s)$ exponentially, and thus the number of total rebuilding steps is bounded by $O(\lg \lg \sigma_s)$, where σ_s denotes the final number of distinct s-symbols indexed by the pBWT. Thus the final construction time stated in Theorem 1 becomes $O(n \frac{(\lg \sigma_p + \lg \lg \sigma_s) \lg n}{\lg \lg n})$, based on the fact that querying or updating the dynamic data structures representing F_T and L_T needs $O(\frac{\lg n}{\lg \lg n})$ time per entry, as we will later see in Lemma 13.
   For computing b in Lemma 9 for a given s-symbol c, we process as follows. Given c has been assigned the rank r, then $b = r - 1$. Otherwise, c has not been ranked, and thus $b = \sigma_s$ since c will receive a rank larger than all other s-symbols indexed in the pBWT.
2. However, if this imposed order is not desirable in some setting, it is possible to assign ranks reflecting the initial order of Σ_s. For that, we note that the aforementioned implementation [21] also supports a sorted traversal of the keys. Thus, it is sufficient to (a) build this associative array while scanning the entire text T, (b) reassign each key a new rank determined by a sorted traversal of the associative array, and finally (c) start the pBWT computation. This, of course, needs to read T twice instead of once.
   Unfortunately, since we keep the initial alphabet order of the s-symbols, determining b in Lemma 9 becomes nontrivial. For computing the value of b, we maintain the set of s-symbols used in the currently computed pBWT by a dynamic fusion tree [34] taking $O(\sigma_s \lg n) = O(n \lg \sigma_s)$ bits. The fusion tree allows us then to compute b in $O(\frac{\lg \sigma_s}{\lg \lg \sigma_s})$ time.

Next, we maintain F_T by a dynamic string of Lemma 3 supporting random access, insertion, rank and select queries in $O(\frac{\lg n}{\lg \lg n})$ time and $O(n \lg \sigma)$ bits of space. For LCP^{∞}_T we maintain a dynamic string of Lemma 4 to support random access, insertion, RmQ, FPQ and FNQ queries in $O(\frac{\lg \sigma_p \lg n}{\lg \lg n})$ time and $O(n \lg \sigma_p)$ bits of space.

[3] Changing the alphabet order for optimizing the compressibility of the BWT is actually an actively researched topic [5].

[4] While this seems like a standard trick for amortizing the costs of dynamic arrays, the amortization argument does not hold here in general because the cost parameter σ_s and the array length n can be independent. For instance, imagine that we first read $n/2$ times the same s-symbol from the input, and then start to read only distinct s-symbols.

If we build a dynamic string of Lemma 4 for L_T, the query time would be $O(\frac{\lg \sigma \lg n}{\lg \lg n})$. Since our algorithm does not use RmQ, FPQ and FNQ queries for s-symbols, we can reduce the query time to $O(\frac{\lg \sigma_p \lg n}{\lg \lg n})$ as follows. We represent L_T with one level of a wavelet tree, where a bit vector partitions the alphabet into Σ_s and $[1..|T|_p]$ and thus has pointers to X_T and Y_T storing respectively the sequence over Σ_s and that over $[1..|T|_p]$ of L_T. We represent the former and the latter by the data structures described in Lemmas 3 and 4, respectively, since we only need the aforementioned queries such as RmQ on Y_T. Then, queries on L_T can be answered in $O(\frac{\lg \sigma_p \lg n}{\lg \lg n})$ time using $O(n \lg \sigma)$ bits of space.

In addition to these dynamic strings for F_T, L_T and LCP^{∞}_T, we consider maintaining another dynamic string Z_T, a string that is obtained by extracting the leftmost occurrence of every p-symbol in T. Note that $|Z_T| = \sigma_p \leq n$. A dynamic string Z_T of Lemma 3 enables us to compute $\pi(cT)$ for a p-symbol c by $\pi(cT) = \min\{\infty, \mathsf{select}_c(Z_T, 1)\}$ in $O(\frac{\lg \sigma_p}{\lg \lg \sigma_p})$ time and $O(\sigma_p \lg \sigma_p)$ bits of space.

We are now ready to prove the following lemma.

▶ **Lemma 13.** *F_T, L_T and LCP^{∞}_T for a p-string of length n over an alphabet $(\Sigma_s \cup \Sigma_p)$ of size $n^{O(1)}$ can be constructed online in $O(n\frac{\lg \sigma_p \lg n}{\lg \lg n})$ time and $O(n \lg \sigma)$ bits of space, where σ_s and respectively σ_p are the numbers of distinct s-symbols and p-symbols used in the p-string and $\sigma = \sigma_s + \sigma_p$.*

Proof. We maintain the dynamic data structures of $O(n \lg \sigma)$ bits described in this subsection while prepending a symbol to the current p-string. For a single step of updating T to $\hat{T} = cT$ with $c \in (\Sigma_s \cup \Sigma_p)$, we compute $\hat{k} = \mathsf{R}_{\hat{T}}(1)$ as described in Subsection 3.1 and obtain $\mathsf{F}_{\hat{T}}$ and $\mathsf{L}_{\hat{T}}$ by replacing \$ in L_T at $k = \mathsf{R}_T(1)$ by $\pi(\hat{T})$ and inserting \$ and $\pi(\hat{T})$ into the $\hat{k}$-th position of L_T and F_T, respectively. LCP^{∞}_T is updated as described in Subsection 3.2.

If $c \in \Sigma_s$, the computation of $\hat{k}$ based on Lemma 9 requires a constant number of queries. If $c \in \Sigma_p$, Algorithm 2 computes $\hat{k}$ invoking $O(2 + e - \hat{e})$ queries, where $e = \max\{\mathsf{LCP}^{\infty}_T[k], \mathsf{LCP}^{\infty}_T[k+1]\}$ and $\hat{e} = \max\{\mathsf{LCP}^{\infty}_{\hat{T}}[\hat{k}], \mathsf{LCP}^{\infty}_{\hat{T}}[\hat{k}+1]\}$. The value e can be seen as a potential held by the current string T, which upper bounds the number of queries. The number of queries in a single step can be $O(\sigma_p)$ in the worst case when e and $\hat{e}$ are close to σ_p and respectively 0, but this will reduce the potential for later steps, which allows us to give an amortized analysis. Since a single step increases the potential at most 1 by Corollary 5, the total number of queries can be bounded by $O(n)$.

Since we invoke $O(n)$ queries that take $O(\frac{\lg \sigma_p \lg n}{\lg \lg n})$ time each, the overall time complexity is $O(n\frac{\lg \sigma_p \lg n}{\lg \lg n})$. ◀

4 Extendable compact index for p-matching

In this section, we show that L_T, F_T and LCP^{∞}_T can serve as an index for p-matching.

First we show that we can support backward search, a core procedure of BWT-based indexes, with the data structures for L_T, F_T and LCP^{∞}_T described in Subsection 3.3. For any p-string w, let *w-interval* be the maximal interval $[l..r]$ such that $\langle T[\mathsf{R}_T^{-1}(p)..]\rangle$ is prefixed by $\langle w \rangle$ for any $p \in [l..r]$. We show the next lemma for a single step of the backward search, which computes *cw-interval* from w-interval.

▶ **Lemma 14.** *Suppose that we have data structures for L_T, F_T and LCP^{∞}_T described in Subsection 3.3. Given w-interval $[l..r]$ and $c \in (\Sigma_s \cup \Sigma_p)$, we can compute cw-interval $[l'..r']$ in $O(\frac{\lg \sigma_p \lg n}{\lg \lg n})$ time.*

Proof. We show that we can compute cw-interval from w-interval using a constant number of queries supported on L_T, F_T and $\langle \hat{T} \rangle$, which takes $O(\frac{\lg \sigma_p \lg n}{\lg \lg n})$ time each.

- When c is in Σ_s: A p-encoded suffix $\langle T[\mathsf{R}_T^{-1}(p)-1..]\rangle = \langle T[\mathsf{R}_T^{-1}(\mathsf{LF}_T(p))..]\rangle$ is prefixed by $\langle cw\rangle$ if and only if $\langle T[\mathsf{R}_T^{-1}(p)..]\rangle$ is prefixed by $\langle w\rangle$ and $\mathsf{L}_T[p] = c$. In other words, $\mathsf{LF}_T(p) \in [l'..r']$ if and only if $p \in [l..r]$ and $\mathsf{L}_T[p] = c$. Then it holds that $l' = \mathsf{LF}_T(\mathsf{FNQ}_c(\mathsf{L}_T, l))$ and $r' = \mathsf{LF}_T(\mathsf{FPQ}_c(\mathsf{L}_T, r))$ due to Corollary 6.
- When c is a p-symbol that appears in w: Similar to the previous case, $\mathsf{LF}_T(p) \in [l'..r']$ if and only if $p \in [l..r]$ and $\mathsf{L}_T[p] = \pi(cw)$. Then it holds that $l' = \mathsf{LF}_T(\mathsf{FNQ}_{\pi(cw)}(\mathsf{L}_T, l))$ and $r' = \mathsf{LF}_T(\mathsf{FPQ}_{\pi(cw)}(\mathsf{L}_T, r))$ due to Corollary 6.
- When c is a p-symbol that does not appear in w: Let $e = |w|_p$. Since $p \in [l..r]$ and $\mathsf{L}_T[p] > e$ are necessary and sufficient conditions for $\mathsf{LF}_T(p)$ to be in $[l'..r']$, we can compute $r' - l' + 1$, the width of $[l'..r']$, by counting the number of positions p such that $\mathsf{L}_T[p] > e$ with $p \in [l..r]$. This can be done with 2D range counting queries, which can also be supported with the wavelet matrix of Lemma 4. If $s = \mathsf{FNQ}_{\geq e+1}(\mathsf{L}_T, l)$ is in $[l..r]$, it holds that $r' - l' + 1 \neq 0$ and $\mathsf{LF}_T(s) \in [l'..r']$. Note that $\mathsf{LF}_T(s)$ is not necessarily l' because p-encoded suffixes $\langle T[\mathsf{R}_T^{-1}(p)..]\rangle$ with $\mathsf{L}_T[p] > e$ in $[l..r]$ do not necessarily preserve the lexicographic order when they are extended by one symbol to the left, making it non-straightforward to identify the position l'.
  To tackle this problem, we consider

  $$[l_e..r_e] = \mathtt{GetMI}(s, e) \text{ and } [l'_{e+1}..r'_{e+1}] = \mathtt{GetMI}(\mathsf{LF}_T(s), e+1),$$

  and show that $l' = l'_{e+1} + x$, where x is the number of positions p in $[l_e..l-1]$ with $\mathsf{L}_T[p] > e$. Observe that $[l..r] \subseteq [l_e..r_e]$ and $[l'..r'] \subseteq [l'_{e+1}..r'_{e+1}]$ by definition, and that $\mathsf{LF}_T(p) \in [l'_{e+1}..r'_{e+1}]$ if and only if $p \in [l_e..r_e]$ and $\mathsf{L}_T[p] > e$ (see Figure 3 for an illustration). Also, it holds that $\langle T[\mathsf{R}_T^{-1}(\mathsf{LF}_T(p))..]\rangle < \langle T[\mathsf{R}_T^{-1}(\mathsf{LF}_T(q))..]\rangle$ for any $p \in [l_e..l-1]$ and $q \in [l..r]$ satisfying $\mathsf{L}_T[p] > e$ and $\mathsf{L}_T[q] > e$ because $\mathsf{lcp}^\infty(\langle T[\mathsf{R}_T^{-1}(p)..]\rangle, \langle T[\mathsf{R}_T^{-1}(q)..]\rangle) = e$, and they fall into Case (B4) of Table 1. Similarly for any $p \in [l..r]$ and $q \in [r+1..r_e]$ satisfying $\mathsf{L}_T[p] > e$ and $\mathsf{L}_T[q] > e$, we have $\langle T[\mathsf{R}_T^{-1}(\mathsf{LF}_T(p))..]\rangle < \langle T[\mathsf{R}_T^{-1}(\mathsf{LF}_T(q))..]\rangle$. Hence, $l' = l'_{e+1} + x$ holds.

This concludes the proof. ◀

We are now ready to prove Theorem 1, which we restate here:

▶ **Theorem 1.** *For a p-string T of length n over an alphabet $(\Sigma_s \cup \Sigma_p)$ of size $n^{O(1)}$, an index of T for p-matching can be constructed online in $O(n\frac{\lg \sigma_p \lg n}{\lg\lg n})$ time and $O(n \lg \sigma)$ bits of space, where σ_s and respectively σ_p are the numbers of distinct s-symbols and p-symbols used in the p-string and $\sigma = \sigma_s + \sigma_p$. At any stage of the online construction, it can support the counting queries in $O(m\frac{\lg \sigma_p \lg n}{\lg\lg n})$ time, where m is the length of a given pattern for queries. By building an additional data structure of $O(\frac{n}{\Delta}\lg n)$ bits of space for a chosen parameter $\Delta \in \{1, 2, \ldots, n\}$ the locating queries can be supported in $O(m\frac{\lg \sigma_p \lg n}{\lg\lg n} + \mathsf{occ}\frac{\Delta \lg n}{\lg\lg n})$ time, where occ is the number of occurrences to be reported.*

Proof of Theorem 1. If we only need counting queries, Lemmas 13 and 14 are enough: While we build L_T, F_T and LCP_T^∞ online, we can compute w-interval $[l..r]$ for a given pattern w of length m using Lemma 14 successively m times, spending $O(m\frac{\lg \sigma_p \lg n}{\lg\lg n})$ time in total.

Since $\{\mathsf{R}_T^{-1}(i) \mid i \in [l..r]\}$ is the set of occurrences of w in T, we consider how to access $\mathsf{R}_T^{-1}(i)$ in $O(\frac{\Delta \lg n}{\lg\lg n})$ time to support locating queries. As is common in BWT-based indexes, we employ a sampling technique (e.g., see [10]): For every $\Theta(\Delta)$ text positions we store the values so that if we apply LF/FL-mapping to i successively at most $\Theta(\Delta)$ times we hit one of the sampled text positions. A minor remark is that since our online construction proceeds from right to left, it is convenient to start sampling from the right-end of T and store the distance to the right-end instead of the text position counted from the left-end of T.

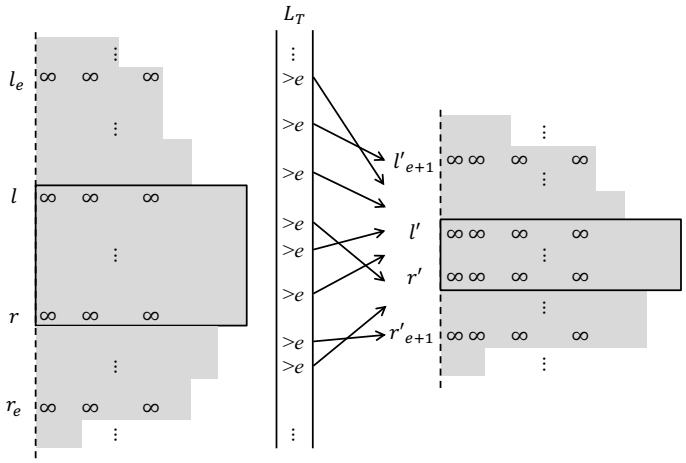


Figure 3 Illustration for the computation of cw-interval $[l'..r']$ from w-interval $[l..r]$ for the case when c is a p-symbol that does not appear in w with $e = |w|_p = 3$. The left (resp. right) part shows sorted p-encoded suffixes around $[l..r]$ (resp. $[l'..r']$) with grayed areas representing the longest common prefix between w (resp. cw) and each p-encoded suffix. The figure illustrates that each position $p \in [l_e..l-1]$ with $\mathsf{L}_T[p] > e$ is mapped to $[l'_{e+1}..l'-1]$ by the LF-mapping.

During the online construction of the data structures for L_T, F_T and LCP^{∞}_T, we additionally maintain a dynamic bit vector of length n and a dynamic integer string V_T of length $O(n/\Delta)$, which marks the sampled positions and stores sampled values, respectively. We implement V_T with the dynamic string of Lemma 3 in $O(\frac{n}{\Delta}\lg n)$ bits with $O(\frac{\lg n}{\lg\lg n})$ query times. In order to support LF/FL-mapping in $O(\frac{\lg n}{\lg\lg n})$ time, we also maintain L_T by an instance of the dynamic string of Lemma 3. Using these data structures, we can access $\mathsf{R}^{-1}_T(i)$ in $O(\frac{\Delta\lg n}{\lg\lg n})$ time as we use LF/FL-mapping at most $O(\Delta)$ times. This leads to the claimed time bound for locating queries. ◀

5 Constructing parameterized suffix arrays in compact space

The parameterized suffix array of a p-string T of length n is the n-length integer array PSA_T with $\mathsf{PSA}_T[i] = \mathsf{R}^{-1}_T(i)$ for any $1 \le i \le n$. We now prove Theorem 2:

▶ **Theorem 2.** *For a p-string T of length n over an alphabet $(\Sigma_s \cup \Sigma_p)$ of size $n^{O(1)}$, the parameterized suffix array of T can be constructed in $O(n\frac{\lg\sigma_p\lg n}{\lg\lg n})$ time and $O(n\lg\sigma)$ bits of space, where σ_s and respectively σ_p are the numbers of distinct s-symbols and p-symbols used in the p-string and $\sigma = \sigma_s + \sigma_p$.*

Proof of Theorem 2. Using Lemma 13, we can build F_T and L_T in $O(n\frac{\lg\sigma_p\lg n}{\lg\lg n})$ time and $O(n\lg\sigma)$ bits of working space, which supports the LF-mapping in $O(\frac{\lg\sigma_p\lg n}{\lg\lg n})$ time. After reserving $n\lg n$ bits space for the array PSA_T, we fill PSA_T in decreasing order of its values starting from $\mathsf{PSA}_T[1] = n$. By definition of the LF-mapping, given a position i with $\mathsf{PSA}_T[i] = x > 1$, the position i' with $\mathsf{PSA}_T[i'] = x-1$ can be computed by $i' = \mathsf{LF}_T(i)$. Therefore, all values of PSA_T can be filled with n applications of the LF-mappings, leading to $O(n\frac{\lg\sigma_p\lg n}{\lg\lg n})$ time in total. ◀

6 Concluding remarks

We have proposed a construction of a BWT-based compact index for p-matching, which works in compact space and in an online manner. BWT-based indexes have been proposed for other generalized pattern matching like structural pattern matching [17], order preserving matching [15], Cartesian tree matching [23], palindrome pattern matching [30] and circular parameterized pattern matching [33]. Generalized pattern matching listed above has a common feature that their underlying equivalent relations are substring consistent [27]. Since previous work has shown that similar techniques can often be applied to this class of generalized pattern matching, it is of great interest to see if the techniques presented in this paper can also be used for constructing other BWT-based indexes.

References

1 Brenda S. Baker. A theory of parameterized pattern matching: algorithms and applications. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 71–80. ACM, 1993. doi:10.1145/167088.167115.

2 Brenda S. Baker. Parameterized pattern matching: Algorithms and applications. *Journal of Computer and System Sciences*, 52(1):28–42, 1996. doi:10.1006/jcss.1996.0003.

3 Brenda S. Baker. Parameterized duplication in strings: Algorithms and an application to software maintenance. *SIAM J. Comput.*, 26(5):1343–1362, 1997. doi:10.1137/S0097539793246707.

4 Richard Beal and Donald A. Adjeroh. p-suffix sorting as arithmetic coding. *J. Discrete Algorithms*, 16:151–169, 2012. doi:10.1016/j.jda.2012.05.001.

5 Jason W. Bentley, Daniel Gibney, and Sharma V. Thankachan. On the complexity of BWT-runs minimization via alphabet reordering. In *Proc. ESA*, volume 173 of *LIPIcs*, pages 15:1–15:13, 2020. doi:10.4230/LIPIcs.ESA.2020.15.

6 Francisco Claude, Gonzalo Navarro, and Alberto Ordóñez Pereira. The wavelet matrix: An efficient wavelet tree for large alphabets. *Inf. Syst.*, 47:15–32, 2015. doi:10.1016/j.is.2014.06.002.

7 Richard Cole and Ramesh Hariharan. Faster suffix tree construction with missing suffix links. *SIAM J. Comput.*, 33(1):26–42, 2003. doi:10.1137/S0097539701424465.

8 Satoshi Deguchi, Fumihito Higashijima, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda. Parameterized suffix arrays for binary strings. In *Proc. Prague Stringology Conference (PSC) 2008*, pages 84–94, 2008. URL: http://www.stringology.org/event/2008/p08.html.

9 Diptarama, Takashi Katsura, Yuhei Otomo, Kazuyuki Narisawa, and Ayumi Shinohara. Position heaps for parameterized strings. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *Proc. 28th Annual Symposium on Combinatorial Pattern Matching (CPM) 2017*, volume 78 of *LIPIcs*, pages 8:1–8:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.8.

10 Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proc. 41st Annual Symposium on Foundations of Computer Science (FOCS) 2000*, pages 390–398, 2000. doi:10.1109/SFCS.2000.892127.

11 Noriki Fujisato, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Right-to-left online construction of parameterized position heaps. In Jan Holub and Jan Zdárek, editors, *Proc. Prague Stringology Conference (PSC) 2018*, pages 91–102. Czech Technical University in Prague, Faculty of Information Technology, Department of Theoretical Computer Science, 2018. URL: http://www.stringology.org/event/2018/p09.html.

12 Noriki Fujisato, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Direct linear time construction of parameterized suffix and LCP arrays for constant alphabets. In Nieves R. Brisaboa and Simon J. Puglisi, editors, *Proc. 26th International Symposium on String Processing and Information Retrieval (SPIRE) 2019*, volume 11811 of *Lecture Notes in Computer Science*, pages 382–391. Springer, 2019. doi:10.1007/978-3-030-32686-9_27.

13 Noriki Fujisato, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. The parameterized position heap of a trie. In Pinar Heggernes, editor, *Proc. 11th International Conference on Algorithms and Complexity (CIAC) 2019*, volume 11485 of *Lecture Notes in Computer Science*, pages 237–248. Springer, 2019. doi:10.1007/978-3-030-17402-6_20.

14 Noriki Fujisato, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. The parameterized suffix tray. In Tiziana Calamoneri and Federico Corò, editors, *Proc. 12th International Conference on Algorithms and Complexity (CIAC) 2021*, volume 12701 of *Lecture Notes in Computer Science*, pages 258–270. Springer, 2021. doi:10.1007/978-3-030-75242-2_18.

15 Travis Gagie, Giovanni Manzini, and Rossano Venturini. An encoding for order-preserving matching. In *Proc. 25th Annual European Symposium on Algorithms (ESA) 2017*, pages 38:1–38:15, 2017. doi:10.4230/LIPIcs.ESA.2017.38.

16 Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. pBWT: Achieving succinct data structures for parameterized pattern matching and related problems. In *Proc. 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) 2017*, pages 397–407, 2017. doi:10.1137/1.9781611974782.25.

17 Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. Structural pattern matching - succinctly. In *Proc. 28th International Symposium on Algorithms and Computation (ISAAC) 2017*, pages 35:1–35:13, 2017. doi:10.4230/LIPIcs.ISAAC.2017.35.

18 Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. Fully functional parameterized suffix trees in compact space. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *Proc. 49th International Colloquium on Automata, Languages, and Programming, (ICALP) 2022*, volume 229 of *LIPIcs*, pages 65:1–65:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.65.

19 Daiki Hashimoto, Diptarama Hendrian, Dominik Köppl, Ryo Yoshinaka, and Ayumi Shinohara. Computing the parameterized Burrows-Wheeler transform online. In Diego Arroyuelo and Barbara Poblete, editors, *Proc. 29th International Symposium on String Processing and Information Retrieval (SPIRE) 2022*, volume 13617 of *Lecture Notes in Computer Science*, pages 70–85. Springer, 2022. doi:10.1007/978-3-031-20643-6_6.

20 Tomohiro I, Satoshi Deguchi, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda. Lightweight parameterized suffix array construction. In *Proc. 20th International Workshop on Combinatorial Algorithms (IWOCA) 2009*, pages 312–323, 2009. doi:10.1007/978-3-642-10217-2_31.

21 Tomohiro I and Dominik Köppl. Load-balancing succinct B trees, 2021. arXiv:2104.08751. doi:10.48550/arXiv.2104.08751.

22 Robert W. Irving and Lorna Love. The suffix binary search tree and suffix AVL tree. *J. Discrete Algorithms*, 1(5-6):387–408, 2003. doi:10.1016/S1570-8667(03)00034-0.

23 Sung-Hwan Kim and Hwan-Gue Cho. A compact index for cartesian tree matching. In Pawel Gawrychowski and Tatiana Starikovskaya, editors, *Proc. 32nd Annual Symposium on Combinatorial Pattern Matching (CPM) 2021*, volume 191 of *LIPIcs*, pages 18:1–18:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CPM.2021.18.

24 Sung-Hwan Kim and Hwan-Gue Cho. Simpler FM-index for parameterized string matching. *Inf. Process. Lett.*, 165:106026, 2021. doi:10.1016/j.ipl.2020.106026.

25 S. Rao Kosaraju. Faster algorithms for the construction of parameterized suffix trees (preliminary version). In *Proc. 36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 631–637. IEEE Computer Society, 1995. doi:10.1109/SFCS.1995.492664.

26 Taehyung Lee, Joong Chae Na, and Kunsoo Park. On-line construction of parameterized suffix trees for large alphabets. *Inf. Process. Lett.*, 111(5):201–207, 2011. doi:10.1016/j.ipl.2010.11.017.

27 Yoshiaki Matsuoka, Takahiro Aoki, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Generalized pattern matching and periodicity under substring consistent equivalence relations. *Theor. Comput. Sci.*, 656:225–233, 2016. doi:10.1016/j.tcs.2016.02.017.

28 Juan Mendivelso, Sharma V. Thankachan, and Yoan J. Pinzón. A brief history of parameterized matching problems. *Discret. Appl. Math.*, 274:103–115, 2020. doi:10.1016/j.dam.2018.07.017.

29 J. Ian Munro and Yakov Nekrich. Compressed data structures for dynamic sequences. In *Proc. 23rd Annual European Symposium on Algorithms (ESA) 2015*, pages 891–902, 2015. doi:10.1007/978-3-662-48350-3_74.

30 Shinya Nagashita and Tomohiro I. PalFM-Index: FM-index for palindrome pattern matching. In Laurent Bulteau and Zsuzsanna Lipták, editors, *Proc. 34th Annual Symposium on Combinatorial Pattern Matching (CPM) 2023*, volume 259 of *LIPIcs*, pages 23:1–23:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.CPM.2023.23.

31 Katsuhito Nakashima, Noriki Fujisato, Diptarama Hendrian, Yuto Nakashima, Ryo Yoshinaka, Shunsuke Inenaga, Hideo Bannai, Ayumi Shinohara, and Masayuki Takeda. Parameterized DAWGs: Efficient constructions and bidirectional pattern searches. *Theor. Comput. Sci.*, 933:21–42, 2022. doi:10.1016/j.tcs.2022.09.008.

32 Gonzalo Navarro. Wavelet trees for all. *J. Discrete Algorithms*, 25:2–20, 2014. doi:10.1016/j.jda.2013.07.004.

33 Eric M. Osterkamp and Dominik Köppl. Extending the parameterized Burrows–Wheeler transform. In *Proc. DCC*, pages 143–152, March 2024.

34 Mihai Patrascu and Mikkel Thorup. Dynamic integer sets with optimal rank, select, and predecessor search. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 166–175, 2014. doi:10.1109/FOCS.2014.26.

Dynamic PageRank: Algorithms and Lower Bounds

Rajesh Jayaram
Google Research, New York, NY, USA

Jakub Łącki
Google Research, New York, NY, USA

Slobodan Mitrović
University of California Davis, CA, USA

Krzysztof Onak
Boston University, USA

Piotr Sankowski
IDEAS NCBR, University of Warsaw, Poland
MIM Solutions, Warsaw, Poland

Abstract

We consider the PageRank problem in the dynamic setting, where the goal is to explicitly maintain an approximate PageRank vector $\pi \in \mathbb{R}^n$ for a graph under a sequence of edge insertions and deletions. Our main result is a complete characterization of the complexity of dynamic PageRank maintenance for both multiplicative and additive (L_1) approximations.

First, we establish matching lower and upper bounds for maintaining additive approximate PageRank in both incremental and decremental settings. In particular, we demonstrate that in the worst-case $(1/\alpha)^{\Theta(\log \log n)}$ update time is necessary and sufficient for this problem, where α is the desired additive approximation. On the other hand, we demonstrate that the commonly employed ForwardPush approach performs substantially worse than this optimal runtime. Specifically, we show that ForwardPush requires $\Omega(n^{1-\delta})$ time per update on average, for any $\delta > 0$, even in the incremental setting.

For multiplicative approximations, however, we demonstrate that the situation is significantly more challenging. Specifically, we prove that any algorithm that explicitly maintains a constant factor multiplicative approximation of the PageRank vector of a directed graph must have amortized update time $\Omega(n^{1-\delta})$, for any $\delta > 0$, even in the incremental setting, thereby resolving a 13-year old open question of Bahmani et al. (VLDB 2010). This sharply contrasts with the undirected setting, where we show that $\operatorname{poly}\log n$ update time is feasible, even in the fully dynamic setting under oblivious adversary.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Information systems → Page and site ranking; Theory of computation → Random walks and Markov chains; Mathematics of computing → Graph algorithms

Keywords and phrases PageRank, dynamic algorithms, graph algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.90

Category Track A: Algorithms, Complexity and Games

Funding *Slobodan Mitrović*: Supported by the Google Research Scholar and NSF Faculty Early Career Development Program #2340048.
Piotr Sankowski: Partially supported by the National Science Center (NCN) grant no. 2020/37/B/ST6/04179 and the ERC CoG grant TUgbOAT no 772346.

1 Introduction

The notion of PageRank was introduced by Brin and Page 25 years ago to rank web search results [7]. Since then, computing the PageRank of a network has become a fundamental task in data mining [23]. At a high level, PageRank is a probability distribution over the vertices

of a directed graph which assigns higher probability to more "central" vertices; see Section 2 for a formal definition. We write $\pi \in \mathbb{R}^n$ to denote PageRank probability vector, where π_i is the probability mass on the i-th vertex. Due to its importance, it has been studied extensively in a number of computational models. In this paper, we consider the PageRank problem in the dynamic setting, in which the goal is to maintain an approximate PageRank vector $\tilde{\pi} \in \mathbb{R}^n$ of a graph undergoing a sequence of edge insertions and deletions. We focus primarily on explicit maintainance of the PageRanks, meaning that the algorithm explicitly maintains $\tilde{\pi}$ in its memory contents at all time steps; we remark that all prior algoriths for the problem of maintaining all PageRanks in the dynamic setting have been of this form.

We consider *three* different settings, which differ in the allowed sets of operations. In the *incremental* setting, edges can only be added to the graph. Analogously, in the *decremental* setting, edges can only be deleted. The most general setting is the *fully dynamic* setting in which we allow both types of updates. We also consider two notions of approximation. A $1+\alpha$ *multiplicative approximation* to the PageRank vector π is a vector $\tilde{\pi}$, such that for every vertex v it holds $\tilde{\pi}_v \in [(1-\alpha)\pi_v, (1+\alpha)\pi_v]$. An *additive* α *approximation* is a vector $\tilde{\pi}$ such that $\|\tilde{\pi} - \pi\|_1 \leq \alpha$.[1] We note that a multiplicative guarantee is strictly stronger, as a multiplicative $1+\alpha$ approximation implies an additive α approximation.

Previous work on dynamic PageRank [4, 11, 24, 5, 10] resulted in two main approaches to the problem. The first one is based on sampling random walks. Specifically, it is well-known that one can approximate PageRank by sampling $O(\log n)$ random walks of length $O(\log n)$ from each vertex in the graph (see Algorithm 1).

In a seminal paper, Bahmani et al. [4] showed that this approach can be made dynamic. Specifically, the algorithm of Bahmani et al. maintains a *multiplicative* $1+\alpha$ approximation of incremental (or decremental) PageRank when the updates *arrive in a random order*. However, their analysis crucially relies on the random arrival of updates, and it was not clear whether this assumption could be removed. The authors of [4] explicitly posed the question of whether it is possible to extend their results for multiplicative approximations to the case of adversarially ordered updates; to date, this question has remained open.

The second approach to computing dynamic PageRank is a dynamic version of the ForwardPush algorithm [25, 1, 9], which is a variant of a classical local push approach proposed by [3]. This algorithm was developed for the problem of maintaining Personalized PageRank, but can also be naturally used to maintain an additive PageRank approximation. While this approach is highly effective in practice, no running time bounds faster than running a static algorithm from scratch after each update have been developed for maintaining PageRank using the dynamic ForwardPush method.[2]

Thus, despite the above line of work, many fundamental questions regarding the computational cost of maintaining PageRank in a dynamic setting remain open. Specifically, it is still open whether there exists an algorithm for maintaining a approximation to PageRank in $o(n)$ time per update. This question is open even if one considers only incremental or decremental updates, or if one allows additive approximation. In this paper, we answer each of these open questions. More precisely, we characterize the complexity of solving the dynamic PageRank problem in each of these settings by providing new upper and lower bounds.

---

[1] Note that this coincides with the total variational distance between distributions.

[2] We note that the paper introducing the dynamic ForwardPush algorithm gives a good running time bound for running the algorithm in *undirected* graphs. However, this bound only holds for computing Personalized PageRank from a *uniformly random* source vertex. Even though PageRank can be reduced to Personalized PageRank, the reduction requires computing Personalized PageRank from a *fixed* vertex, and so the bound does not carry over.

1.1 Our contributions

We provide new lower and upper bounds on the complexity of explicitly maintaining an approximate PageRank vector both under additive and multiplicative approximation. Throughout this section, we use n to denote the number of vertices in a graph, m to denote the number of edges and ϵ to denote the jumping probability used to define PageRank.[3]

1.1.1 Additive Approximation

We provide (essentially) matching lower and upper bounds for explicitly maintaining additive approximation of PageRank in both incremental and decremental setting.

▶ **Theorem 1.** *Fix $\epsilon \in (0.01, 0.99)$. For any sufficiently large $n \geq 1$ and any α such that $1/\alpha = n^{o(1/\log\log n)}$, any algorithm which explicitly maintains α-additive approximation of PageRank must run in $n \cdot (1/\alpha)^{\Omega(\log\log n)}$ total time.*

Our lower bound, which we prove in Section 3.1, is obtained by constructing a graph and an update sequence for which the PageRank vector undergoes a large number of significant changes. The changes to the vector are large to the point that even an approximate PageRank vector must be often updated in linear time. We note that the lower bound, and all other lower bounds that we state, applies to the setting when the PageRank vector is maintained explicitly, i.e., after each update algorithm outputs the changes that the PageRank vector undergoes.

We note that it is easy to come up with an example in which a single edge update significantly changes the PageRanks of a large fraction of vertices (see Figure 1). This immediately rules out efficient incremental and decremental algorithms that maintain approximate PageRank with *worst-case* update time guarantees. This also rules out fully dynamic algorithms with amortized update time guarantees. However, proving a strong lower bound for the *amortized* update time bound in the incremental or decremental setting is far more involved, as it requires showing a long sequence of updates in which, on average, every edge insertion (or deletion) changes the PageRank of many vertices.

We complement our lower bound with the following algorithmic result proved in Section 5.

▶ **Theorem 2.** *For any $\epsilon \in (0, 1)$, there is an algorithm that with high probability explicitly maintains an α additive approximation of PageRank of any graph G in either incremental or decremental setting. The algorithm processes the entire sequence of updates in $O(m) + n \cdot (1/\alpha)^{O_\epsilon(\log\log n)}$ total time and works correctly against an oblivious adversary.*

Furthermore, we study the complexity of the dynamic ForwardPush algorithm [25]. This algorithm, when run with parameter $\tilde{\alpha}$ maintains an $\tilde{\alpha} \cdot m$ additive approximation to PageRank (and so to obtain α additive approximation, one needs to use $\tilde{\alpha} = \alpha/m$). By using a similar construction of a hard instance, we show that the algorithm takes $\Omega(n^{2-\delta})$ time, for any $\delta > 0$, to handle a sequence of $O(n)$ operations, even in incremental or decremental settings (see Theorem 9).

[3] The probability of not-jumping (in our notation, $1 - \epsilon$) is sometimes called the *damping factor* of PageRank.

1.1.2 Multiplicative Approximation

Our next result is a lower bound showing that any algorithm explicitly maintaining a constant multiplicative approximation to PageRank, even in the incremental or decremental setting, must in the worst case take $\Omega(n^{2-\delta})$ total time, for any $\delta > 0$, to process a sequence of n updates to an n-vertex graph. Specifically, we prove the following in Section 3.2:

▶ **Theorem 3.** *There exists a sequence of $\Theta(n)$ edge insertions applied to an initially empty graph on n vertices for which the following holds. For any constant $\delta > 0$, any algorithm that maintains a vector $\tilde{\pi} \in \mathbb{R}^n$ such that $(1/2)\pi_v < \tilde{\pi}_v \leq 2\pi_v$ at all time steps, must take time $\Omega(n^{2-\delta})$ to process the sequence. In particular, the amortized update time of any such algorithm is $\Omega(n^{1-\delta})$.*

We note that, by symmetry, the above theorem also applies to the decremental setting.

Theorem 3 gives a negative resolution to the 13-year-old open question of Bahmani et al. [4], who asked whether their polylogarithmic update time bounds for maintaining PageRank under a sequence of updates coming in *random* order can be extended to the general case. Previously, the only negative results for this problem were given by Lofgren [12] who showed that the specific algorithm of Bahmani et al. requires $\Omega(n^c)$ update time for some $c \in (0, 1)$, but this did not rule out the existence of a better algorithm. We extend this lower bound to *every* algorithm which explicitly maintains an approximate PageRank vector, and strengthen the bound from $\Omega(n^c)$ to $\Omega(n^{1-\delta})$ for any $\delta > 0$.

To complement the above lower bound, in Section 6, we give a simple analysis of the Bahmani et al. algorithm in *undirected* graphs, and show that in this case maintaining multiplicative approximation can be done in polylogarithmic time per update even in the fully dynamic setting. This algorithm also assumes an oblivious adversary. While the analysis is based on a simple observation, to the best of our knowledge it has not been explicitly given before.

▶ **Theorem 4.** *For any $\epsilon \in (0, 1)$, there is an algorithm that with high probability explicitly maintains a $1 + \alpha$ multiplicative approximation of PageRank of any undirected graph G in the fully dynamic setting. The algorithm handles each update in $O(\log^5 n/(\epsilon^2\alpha^2))$ time and works correctly against an oblivious adversary.*

It is an open question whether it is possible to design dynamic PageRank algorithms that bypass our lower bounds, for example, by not maintaining PageRank explicitly or looking beyond worst-case bounds and studying restricted graph classes.

1.2 Related Work

The dynamic PageRank problem has been studied in a number of recent works [4, 5, 11, 24, 8, 17, 18, 13, 19, 20, 9] studying both the theoretical and empirical aspects of the problem. One line of study considered the incremental and decremental settings with updates performed in *random* order [4, 24] and obtained algorithms that achieve $O(\log n/\epsilon)$ update time. The result of [24] is applicable in a non-random order as well, although in that case it requires $\Omega(d_v)$ running time per update done on a vertex v of degree d_v. Bahmani et al. [5] analyze their algorithm in a random graph model in which high PageRank vertices are more likely to receive new neighbors. We note that attempts at designing faster algorithms have been undertaken in [11] as well as [24]. However, these algorithms come with no provable approximation guarantees.

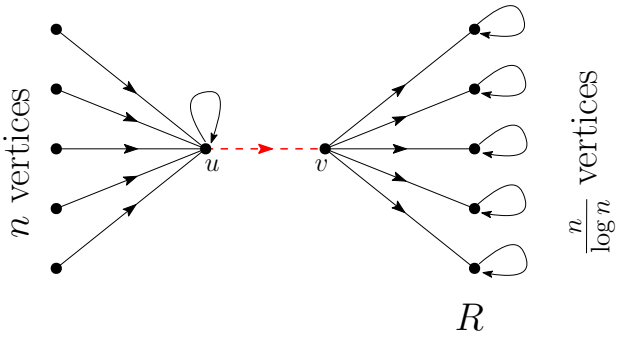


Figure 1 An example illustrating that maintaining multiplicative approximation or even an L_1 approximation of PageRank in the worst case requires $\Theta(n/\log n)$ running time even after a single deletion/insertion of edge uv. For details, see Section 1.3.

Another line of work [2, 14, 15, 25, 21] focuses on computing Personalized PageRank, which is PageRank computed from the point of view of a single vertex. For instance, [15] show that if each entry of a Personalized PageRank is lower-bounded by δ, then the Personalized PageRank of a vertex can be approximated in time $O(\sqrt{d/\delta})$, where d is the average graph degree.

Finally, PageRank was also studied in the context of sublinear algorithms [6, 22]. For instance, for a graph on m edges and omitting poly dependence on $\log m$ and α^{-1}, the very recent algorithm presented in [22] requires $O(n^{1/2} \cdot \min\{m^{1/4}, \Delta^{1/2}\})$ running time for approximating the PageRank of a single vertex, where Δ is the maximum degree in the graph.

1.3 Impossibility of Non-Trivial Worst-Case Bounds

A wealth of literature on designing dynamic algorithms for approximate PageRank, including our results, focuses on *amortized* running time complexity. It is natural to wonder whether non-trivial worst-case update running times do not exists due to lack of techniques or due to fundamental reasons. As our example in Figure 1 illustrates, non-trivial update running times are not possible even on very sparse graphs and even if one's goal is to maintain an L_1-approximate PageRank vector.

Namely, on the one hand, for the graph G in Figure 1, it can be shown that $\pi_u, \pi_v \in \Omega(\epsilon)$ and $\pi_x \in \Omega((\log n)/n)$ for each vertex $x \in R$. On the other hand, consider graph G' obtained from G, i.e., from the graph in Figure 1, by removing the red-dashed (u, v) edge, and let π' be the PageRank of G'. It is not hard to show that $\pi'_u \in \Omega(1)$, $\pi'_v \in O(\epsilon/n)$ and $\pi'_x \in O(1/n)$ for each $x \in R$. This example illustrates the following: there exists a directed graph in which after *a single* edge removal one has to update $\Omega(n/\log n)$ vertices if the goal is to maintain a multiplicative and even an L_1 approximation of the PageRank for sufficiently small constant ϵ. Moreover, if random walks are used to estimate the PageRank – which to the best of our knowledge is the only other used approach than Power method – then maintaining an additive or multiplicative approximate PageRank of a single vertex still requires $\Omega(n)$ worst-case time. To see that, observe that there are $\Theta(n)$ times more random walks passing through v in G than in G'.

Algorithm 1 An algorithm for computing approximate PageRank using random walks.
Input: A graph G, and parameters ϵ, α and ℓ.

1: Sample a set W of $R = \lceil \frac{9 \ln n}{\epsilon \alpha^2} \rceil$ random walks starting from each vertex of G. Each walk length is chosen from geometric distribution with parameter $1 - \epsilon$.
2: Remove from W all walks longer than ℓ.
3: **for** $v \in V$ **do**
4: $\quad \mathbf{X}_v \leftarrow$ the number of times the walks from W visit v.
5: $\quad \tilde{\pi}(v) \leftarrow \frac{\mathbf{X}_v}{|W|/\epsilon}$.
6: **end for**
7: Return $\tilde{\pi}$

1.4 Organization of the Paper

The rest of this paper is organized as follows. In Section 2 we formally define PageRank and review a random-walk based algorithm for approximating it in the static setting. In Section 3 we give the lower bounds on the time required to explicitly maintain PageRank and on the running time of the dynamic ForwardPush algorithm. Section 4 reviews the algorithm for approximating PageRank by maintaining random walks. While the algorithm is essentially the same as the algorithm by Bahmani et al. [4], we present a full analysis, since the previous papers on dynamic PageRank did not prove the correctness of this approach. In the following two sections we analyze this algorithm in two settings. First, in Section 5 we show that this algorithm achieves near-optimal update time while maintaining additive approximation to PageRank. Second, in Section 6 we present a simple analysis showing that in undirected graphs maintaining even a constant multiplicative approximation to PageRank in the fully dynamic setting is possible with polylogarithmic update time.

2 Preliminaries

We begin by defining the PageRank of a directed graph $G = (V, E)$. Formally, the PageRank of G, denoted by $\pi \in \mathbb{R}^n_{\geq 0}$, is the stationary distribution of a random walk on G, where at each step the walk *jumps* to another uniformly random vertex with probability $\epsilon \in (0, 1)$. The jump probability ϵ is a parameter, which we will fix for the remainder. If $\deg(i)$ is the out-degree of the i-th vertex in V, then the corresponding non-symmetric transition matrix $M \in \mathbb{R}^{n \times n}$ has entries $M_{i,j} = \frac{\epsilon}{n} + (1 - \epsilon)\frac{1}{\deg(i)}$ if $(i, j) \in E$, and $M_{i,j} = \epsilon/n$ otherwise. We make the standard assumption (required for PageRank to be well-defined) that each vertex has $\deg(i) \geq 1$, which can be accomplished by adding self-loops.

PageRank can be approximated by sampling $O(\log n/\epsilon)$ relatively short random walks from each vertex. One such approach is provided as Algorithm 1, for which the following can be shown.

▶ **Proposition 5** ([4, 16]). *Let π be the PageRank vector of a graph G. The estimate $\tilde{\pi}$ computed by Algorithm 1 (with $\ell = \infty$) satisfies (a) for all $v \in V$ we have $\mathbb{E}[\tilde{\pi}_v] = \pi_v$, and (b) with probability $1 - 1/\operatorname{poly}(n)$, simultaneously for all $v \in V$, we have $\tilde{\pi}_v = (1 \pm \alpha)\pi_v$.*

3 Lower Bounds

In this section we present our lower bounds for maintaining explicit approximation to PageRank and for the running time of the dynamic ForwardPush algorithm [25]. We now describe a generic construction of a hard instance, which we instantiate with different parameters in each of the individual lower bounds. Throughout the section, we consider the case of $\epsilon \in (0.01, 0.99)$, which is the usual case in the applications of PageRank.

The Graph. The graph G is a union of the graphs H, R, S_0, S_1. First, H is a directed tree. Each non-leaf vertex v has exactly t children, with p^i parallel directed edges from v to the i-th child of v (where i is 0-based). We require that $p \geq \max(1/\epsilon, 2)$. Hence, the total-out degree of each internal vertex in H is $O(p^t)$. The depth of H is d, and so H has $\Theta(t^d)$ vertices and $\Theta(t^d \cdot p^t)$ edges.

The graph R consists of $n/4$ vertices v, each with no in-edges, and each with a single out edge vr where r is the root of the directed tree H. Finally, the sets S_0, S_1 are both directed star graphs on $s+1$ vertices (with the edges directed away from the center of the star), where S_i has the center c_i for $i \in \{0, 1\}$. Additionally, each leaf of S_0 and S_1 has a single outgoing edge, which is a self-loop. We then order the leaf vertices of H as $\ell_1, \ell_2, \ldots$, and create a directed edge from ℓ_i to $c_{i \bmod 2}$. We will set the parameters, such that the total number of vertices in H, R, S_0, and S_1 is less than n. One can then add an additional $O(n)$ isolated vertices (with self-loops), so that the total number of vertices is precisely n.

Update Sequence The initial graph has all vertices and edges of H, R, S_0, S_1, except that each non-leaf vertex of H only has an edge to its leftmost child (i.e., one with index 0). Observe that each vertex has at least one outgoing edge, and so PageRank is well-defined.

The update sequence is as follows. Let $v_1, \ldots, v_{|H|}$ be the sequence of vertices visited on a pre-order traversal of H, such that $\ell_1, \ell_2, \ldots$ is a subsequence of $v_1, \ldots, v_{|H|}$. We insert the edges of H in $|H|$ rounds: in the ith round we insert all incoming edges of v_i (unless they have already been in the graph from the beginning).

To prove the lower bounds, we use the following way of interpreting PageRank, which is a continuous version of Algorithm 1 and follows from Proposition 5. Each vertex has some *probability mass*, which it either generates or receives from its in-neighbors. Specifically, each vertex of the graph generates a probability mass of $1/n$. A $1-\epsilon$ fraction of the probability mass of a vertex v (either generated by v or incoming to v from other vertices) is divided uniformly among the outgoing edges of v and sent to the neighbors of v. The PageRank of each vertex is exactly ϵ fraction of its probability mass.

Note that if a vertex is on a cycle, some probability mass enters it multiple times. In this case, each time the mass enters the vertex, it increases the total probability mass. In particular, we have the following.

▶ **Observation 6.** *Let v be a vertex, whose only outgoing edge is a self loop. Assume that v receives a probability mass of p along its incoming edges other than the self-loop. Then, the PageRank of v is $p + 1/n$.*

▶ **Lemma 7.** *Consider the graph G^τ obtained right after inserting all edges on the path from R to ℓ_i. Let m_i be the probability mass that reaches ℓ_i from R in G^τ. Then $m_i \geq (1-\epsilon)^{2d+2}/4$.*

Moreover, out of the probability mass that reaches the leaves of H from R, at least $(1-1/p)^d$ fraction reaches ℓ_i.

Proof. Observe that a path from any vertex $u \in R$ to ℓ_i first follows the edge to r, which is the only outgoing edge of u, and then, thanks to the order of adding edges of H, at each step uses the rightmost edge of each vertex in H. Consider an internal vertex $w \in H$. By the construction it has p^i edges to the ith child (0-based). Assuming that we have added edges to j children so far, we have that there are p^{j-1} edges to the rightmost child and so the fraction of outgoing edges of w that go to the rightmost child is:

$$p^{j-1} / \left(\sum_{k=0}^{j-1} p^k \right) = p^{j-1} \cdot \frac{p-1}{p^j - 1} \geq p^{j-1} \cdot \frac{p-1}{p^j} = 1 - 1/p. \tag{1}$$

The path from w to ℓ_i has $d+1$ edges. At each step $1-\epsilon$ fraction of the probability mass is forwarded to the children, out of which, as shown above, at least $1-1/p \geq 1-\epsilon$ fraction follows the path to ℓ_i. Hence, the fraction of probability mass that reaches ℓ_i from w is $(1-\epsilon)^{2d+2}$. Since vertices of R generate a total probability mass of $1/4$, we get the desired.

The second claim follows directly from Equation (1) and the fact that H has depth d. ◀

3.1 Lower Bound for Maintaining Additive Approximation

We first show the following auxiliary lemma which we will use to argue when an additive α-approximate PageRank vectors must be updated in linear time.

▶ **Lemma 8.** *Consider four vectors $v^1, \tilde{v}^1, v^2, \tilde{v}^2 \in \mathbb{R}^n$, such that $\|v^1 - \tilde{v}^1\|_1 \leq \alpha$, $\|v^2 - \tilde{v}^2\|_1 \leq \alpha$ and v^1 and v^2 differ by at least $100 \cdot \alpha/n$ on at least $n/4$ coordinates. Then $\tilde{v}^1$ and $\tilde{v}^2$ differ on $\Omega(n)$ coordinates.*

Proof. The proof goes by contradiction. Assume that $\tilde{v}^1$ and $\tilde{v}^2$ differ on at most $n/1000$ coordinates. Thus, they have at least $0.999 \cdot n$ coordinates in common. Moreover, $\|v^1 - \tilde{v}^1\|_1 \leq \alpha$ implies that v^1, and $\tilde{v}^1$ differ by more than $10 \cdot \alpha/n$ on less than $0.1 \cdot n$ coordinates. Clearly, a similar property is satisfied by v^2, and $\tilde{v}^2$.

Let I be the set of coordinates where

1. $\tilde{v}^1$ and $\tilde{v}^2$ are equal (there are at least $0.999 \cdot n$ such coordinates),
2. v^1 and v^2 differ by at least $100 \cdot \alpha/n$ (at least $n/4$ such coordinates),
3. v^1 and $\tilde{v}^1$ differ by at most $10 \cdot \alpha/n$ (at least $0.9 \cdot n$ coordinates),
4. v^2 and $\tilde{v}^2$ differ by at most $10 \cdot \alpha/n$ (at least $0.9 \cdot n$ coordinates).

Observe that since the vectors have n coordinates, I is nonempty. By using first the triangle inequality, and then items 2-4 above, for any coordinate $i \in I$ we have

$$\begin{aligned} |\tilde{v}_i^1 - \tilde{v}_i^2| &\geq |v_i^1 - v_i^2| - |v_i^1 - \tilde{v}_i^1| - |v_i^2 - \tilde{v}_i^2| \\ &\geq 100 \cdot \alpha/n - 10 \cdot \alpha/n - 10 \cdot \alpha/n \\ &= 80 \cdot \alpha/n. \end{aligned}$$

which contradicts item 1. The lemma follows. ◀

▶ **Theorem 1.** *Fix $\epsilon \in (0.01, 0.99)$. For any sufficiently large $n \geq 1$ and any α such that $1/\alpha = n^{o(1/\log\log n)}$, any algorithm which explicitly maintains α-additive approximation of PageRank must run in $n \cdot (1/\alpha)^{\Omega(\log\log n)}$ total time.*

Proof. We instantiate our construction using the following parameters. The number of edges from a vertex to its ith child is $(1/\epsilon)^i$ ($p = 1/\epsilon$). Each vertex of H has $t = 1/2 \cdot \log_p n$ children. The tree H has depth $d = \frac{\log(101\alpha)}{2\log(1-\epsilon)} - 2 \geq 1$. Note that $d = \Theta(\log(1/\alpha))$. Finally, S_0, S_1 have $s = n/4$ leaves.

Let us now bound the size of the graph. The number of leaves of H is

$$t^d = (1/2 \cdot \log_{1/\epsilon} n)^d = \left(\frac{\log n}{2\log 1/\epsilon}\right)^{\Theta(\log(1/\alpha))} = \log^{\Theta(\log(1/\alpha))} n = (1/\alpha)^{\Theta(\log\log n)}, \qquad (2)$$

where in the third step we use the fact that $\frac{\log n}{2\log 1/\epsilon} = \log^{\Theta(1)} n$ for sufficiently large n.

R, S_0 and S_1 have $n/4$ vertices each. By Equation (2) and the assumption on α, H has $o(n)$ vertices, and so with the additional isolated vertices, the graph has exactly n vertices.

The number of edges incident to R, S_0 and S_1 is $O(n)$. The number of children of each internal vertex of H is

$$\Theta(p^t) = \Theta(p^{1/2 \cdot \log_p n}) = \Theta(n^{1/2}).$$

Thus, the total number of edges in H is $(1/\alpha)^{\Theta(\log\log n)} \cdot n^{1/2} = n^{o(1)} \cdot \Theta(n^{1/2})$. Hence, we conclude that the graph has n vertices and $O(n)$ edges.

Observe that as we add edges, leaves $\ell_1, \ell_2, \ldots$ become reachable from R exactly in the order of their indices. Fix any leaf ℓ_j of H. Denote by π^b and π^a, respectively, the PageRank vectors just before ℓ_j is reachable from R and just after all edges on the path from R to ℓ_j are added.

We use the interpretation of PageRank based on probability mass. Before ℓ_j is reachable from R, it may receive probability mass only from its ancestors in H. Hence,

$$\pi^b_{\ell_j} \leq (d+1)/n = \Theta(\log(1/\alpha))/n = o(\log n)/n.$$

Moreover, since PageRank is a ϵ fraction of the probability mass entering each vertex, by Lemma 7,

$$\pi^a_{\ell_j} \geq \epsilon \cdot (1-\epsilon)^{2d+2}/4 = \epsilon \cdot (1-\epsilon)^{\frac{\log(101\alpha)}{\log(1-\epsilon)}-2}/4 = 101 \cdot \epsilon \cdot \alpha \cdot (1-\epsilon)^{-2}.$$

The increase to PageRank of ℓ_j is thus at least $\pi^a_{\ell_j} - \pi^b_{\ell_j} \geq 100 \cdot \epsilon \cdot \alpha(1-\epsilon)^{-2}$. Hence, after the insertion there is at least $100\alpha(1-\epsilon)^{-2}/4$ "new" probability mass at ℓ_j. Since every two hop path from j leads to a leaf in $S_{j \bmod 2}$, each of these leaves will receive a least

$$100 \cdot \alpha \cdot (1-\epsilon)^{-2}/4 \cdot (1-\epsilon)^2/s = 100 \cdot \alpha/(4s).$$

new probability mass (since only $(1-\epsilon)$ fraction of the probability mass is transferred along each hop). By Observation 6 all of that probability mass ends up increasing the PageRank of the leaf. Therefore the PageRank of each of these s leaves increases by $100 \cdot \alpha/(4s) = 100 \cdot \alpha/(4 \cdot n/4) = 100 \cdot \alpha/n$.

We now use Lemma 8 with $v_1 = \pi^b$, $v_2 = \pi^a$ and $\tilde{v}_1$ and $\tilde{v}_2$ being any PageRank vectors giving α-additive approximation and infer that $\Omega(n)$ coordinates of any approximate PageRank vector must be updated in order to maintain α-additive approximation. This happens for each leaf of H, and so by Equation (2) the Lemma follows. ◀

3.2 Lower Bound for Maintaining Multiplicative Approximation

▶ **Theorem 3.** *There exists a sequence of $\Theta(n)$ edge insertions applied to an initially empty graph on n vertices for which the following holds. For any constant $\delta > 0$, any algorithm that maintains a vector $\tilde{\pi} \in \mathbb{R}^n$ such that $(1/2)\pi_v < \tilde{\pi}_v \leq 2\pi_v$ at all time steps, must take time $\Omega(n^{2-\delta})$ to process the sequence. In particular, the amortized update time of any such algorithm is $\Omega(n^{1-\delta})$.*

Proof. We instantiate our construction as follows. Each non-leaf vertex v of H has exactly $t = \delta/2 \log n/\log\log n$ children, with $(\log^2 n)^i$ parallel directed edges from v to the i-th child of v ($p = \log^2 n$). It follows that the total outdegree of each internal vertex in H is $O(n^\delta)$. The depth of H is set to be $d = \log_t(n^{1-2\delta}) = \Theta(\log n/\log\log n)$, so that H has $n^{1-2\delta}$ vertices, and the total number of edges in H is $O(n^{1-\delta})$. Finally, both S_0 and S_1 have $s = n^{1-2\delta}$ vertices.

Fix a leaf ℓ_j of H and consider the state of the algorithm right after all on the path from the root of H to ℓ_j have been added. By Lemma 7, the probability mass entering ℓ_j is at least.

$$(1-\epsilon)^{2d+2}/4 = (1-\epsilon)^{\Theta(\log n/\log\log n)} = n^{\Theta(-1/\log\log n)}.$$

Out of this probability mass a constant fraction reaches the leaves of $S_{j \bmod 2}$. In particular, the PageRank of each such leaf is at least $\epsilon \cdot n^{\Theta(-1/\log\log n)}/n^{1-2\delta} \geq n^{-\delta}$.

Moreover, out of the probability mass from R the fraction that reaches ℓ_j is at least

$$(1-1/p)^d = (1-1/\log^2 n)^{\Theta(\log n/\log\log n)} \geq 1 - 1/\log n.$$

out of all probability mass that reaches the leaves of H from R. Observe that compared to this probability mass (which is a constant), the total probability mass generated by all vertices of H is negligible. As a result, the ratio of probability mass that reaches $S_{j \bmod 2}$ to the probability mass that reaches $S_{(j+1) \bmod 2}$ is

$$\frac{1-1/\log n}{1/\log n} = \Theta(\log n).$$

This implies that when we add all edges on a path from R to ℓ_j, the PageRanks of leaves of $S_{j \bmod 2}$ increase by a factor of $\Theta(\log n)$ and so the PageRank estimates of all these $\Omega(n^{1-2\delta})$ vertices must be changed. Since a total of $m = O(n)$ edges are added, and since this occurs once for each of the $\Omega(n^{1-\delta})$ leaf vertices in H, we obtain a total of $\Omega(n^{2-3\delta})$ PageRank estimate updates, which is the desired result after rescaling δ by a constant. ◀

3.3 Lower bound for the ForwardPush algorithm

▶ **Theorem 9.** *Consider running the ForwardPush [25] algorithm whose error parameter is set to ensure that the algorithm maintains additive α approximation of PageRank. For any $\delta > 0$, each sufficiently large $n \geq 1$ and $\epsilon \in (0.01, 0.99)$ there exists a graph on n vertices and a sequence of $O(n)$ edge insertions, such that the algorithm runs in $\Omega(n^{2-\delta})$ time.*

Proof. We use our construction with the same settings as in the proof of Theorem 3. Specifically, $t = \delta/2 \log n/\log\log n$, $p = \log^2 n$, $d = \log_t(n^{1-2\delta}) = \Theta(\log n/\log\log n)$ and $s = n^{1-2\delta}$.

The ForwardPush algorithm can be explained using the probability mass interpretation. The algorithm maintains a *residual* on each vertex u, denoted by R_u. This residual can be positive or negative. Initially, the residual of each vertex is $1/n$.

The residual is a probability mass that still has to be pushed to the neighbors of u. The algorithm maintains two invariants

1. $|R_u| \leq \gamma \deg(u)$ for each vertex $u \in V$, where γ is an accuracy parameter.
2. If we keep pushing the residuals, the PageRank estimates converge to the exact PageRank values.

For any vertex u that violates the invariant, that is satisfies $|R_u|/\deg(u) > \gamma$, the algorithm executes a *push* operation, which takes time $\Theta(\deg(u))$ and pushes a $1-\epsilon$ fraction of the residual to the outneighbors of u and uses a ϵ fraction of the residual to increase the PageRank of u. The residual of u is then set to 0. Upon an insertion of an edge uv, the algorithm decreases R_u by $\Delta = \Theta(\pi_u)/\deg(u)$ and increases R_v by Δ. Then, it restores the invariant by executing push operations.

In the following part of the proof we use the following observation, which follows from the second algorithm invariant.

▶ **Observation 10.** *Fix a vertex v and denote by D_v the set of vertices that have a directed path to v. We assume $v \in D_v$. Then, the total additive error of the PageRank estimate maintained by the ForwardPush algorithm is at most $\sum_{u \in D_v} |R_u|$.*

By using the second algorithm invariant, we get that ForwardPush ensures that the total additive error is $\sum_{u \in V} |R_u| \leq \sum_{u \in V} \gamma \deg(u) = \gamma m$. Therefore, to ensure an additive α approximation of PageRank, we set $\gamma m \leq \alpha$, implying $\gamma \leq \alpha/m$. We note that it is easy to come up with an example where this analysis is tight up to a constant factor.

We now analyze ForwardPush algorithm on our hard instance. Since the number of edges in our graph is $\Theta(n)$, we invoke ForwardPush with the approximation parameter $\gamma = \Theta(1/n)$. We claim that with this value of γ, the residual values are propagated often enough so that over $\Theta(n)$ edge insertions described above, ForwardPush makes $\Omega(n^{2-\delta})$ updates.

We use the observations from the proof of Theorem 3 that the PageRank of a vertex c_i ($i \in \{0, 1\}$) is $n^{\Theta(-1/\log\log n)}$ and, as we add edges, increases by a $\Theta(\log n)$ factor each time we fully add a path from R to a leaf ℓ_j, such that $i = j \bmod 2$.

We now use Observation 10 to show that the ForwardPush maintains a constant factor approximate of the PageRank estimates of c_0 and c_1. Indeed, these vertices can only be reached from R, H or from themselves. We now bound the residuals of these vertices. The residuals of the vertices of R are set to 0 the moment each of these vertices performs the first push operation and are then never updated. The residual of each vertex v of H satisfies $|R_v|/\deg(v) \leq \alpha/m$ which implies $|R_v| \leq \Theta(\deg(v))/m = \Theta(n^{\delta-1})$. Finally, the residual of c_0 (and, similarly c_1) satisfies $|R_{c_0}|/\deg(c_0) \leq \alpha/m$, which gives $|R_{c_0}| \leq \Theta(n^{1-2\delta})/m = \Theta(n^{-2\delta})$. By applying Observation 10 we have that the additive error the PageRank estimates of c_0 and c_1 is at most

$$\Theta(n^{\delta-1}) \cdot \Theta(n^{1-2\delta}) + \Theta(n^{-2\delta}) = \Theta(n^{-\delta}).$$

These additive errors are negligible comared to the PageRanks of these vertices, which is $n^{\Theta(-1/\log\log n)}$. Hence, the algorithm maintains constant-factor estimates of the PageRanks of c_0 and c_1. As a result, when the exact PageRank values change by a factor of $\Theta(\log n)$, the algorithm updates their estimates. However, the ForwardPush algorithm only updates a PageRank estimate of a vertex u when either it executes a push operation on u or adds an outgoing edge from u. Since all outgoing edges of c_0 and c_1 have been added in the beginning, we get that the algorithm executes a push operation on c_0 for half of leaves of H. Each such operation takes $\Theta(\deg(c_0)) = \Theta(n^{1-2\delta})$ time and so the overall running time of the algorithm is $\Theta(n^{1-2\delta} \cdot n^{1-2\delta}) = \Theta(n^{2-4\delta})$ which, after tweaking δ by a constant factor, gives the desired. ◀

4 Approximating PageRank by Maintaining Dynamic Random Walks

In this section we review the algorithm for approximating PageRank by maintaining random walks. This algorithm is a dynamic version of Algorithm 1 and has been previously described by Bahmani et al. [4]. We provide a detailed proof of correctness of the algorithm, which to the best of our knowledge has not been included in any prior work.

The algorithm relies on maintaining $O_\epsilon(n \log n)$ random walks and re-sampling their parts as necessary. In this section, we present data structures that we use to efficiently maintain and re-sample those random walks. Section 4.1 presents our approach on an edge insertion, while Section 4.2 describes how our algorithms handle edge deletions. We being by describing the problem setup.

Setup. Following Proposition 5, to approximate the PageRank it suffices to sample $R = O(\log n/(\epsilon\alpha^2))$ PageRank walks from each vertex. A PageRank walk is a random walk w, whose length ℓ_w is sampled from geometric distribution with parameter $1 - \epsilon$. Even though a

given walk may get re-routed after edge insertions or deletions, it is crucial that the the length of each walk remains **fixed** throughout the entire execution of the algorithm. Otherwise, it is easy to construct examples where the lengths of the maintained walks no longer follow the right distribution.

We maintain two types of data structures. For each each vertex v and $t = 0 \dots O(\log n/\epsilon)$, we maintain a binary search tree $S_{v,t}$ which stores all the walks whose t-th vertex if v. For each edge e, we maintain the binary search tree W_e consisting of the walks passing through e.

4.1 Edge Insertion

When an edge (u, v) is inserted, we re-sample some of the walks passing through u. This re-sampling is done by first performing rejection sampling on each walk and, second, by choosing an appropriate position where each of the rejected walks should be re-sampled. Choosing an appropriate position from where to re-sample w is trivial in case when w passes through u once. However, it might be the case that w passes through that vertex multiple times, and a more careful consideration is required. At a high level, we iterate through all segments of w and for each segment of w that leaves u we toss a coin. Then, with probability $1/d_u$, where d_u is the degree of u after the update, we reroute w starting from the considered segment, and terminate the update procedure for w.

Each walk has a unique ID associated with it. These IDs are integers ranging from 1 through the number of walks we maintain. Each vertex and each edge keeps track of which walks are passing through them.

Given a vertex v and integers i and t, it will be convenient to be able to sample the i-th walk whose t-th vertex is v. It will become clear why such operation is needed when we describe how to handle edge insertions. To be able to implement this operation efficiently, we store the IDs of walks whose t-th vertex is v in a binary tree; we use $S_{v,t}$ to refer to this binary tree. Then, the i-th walk can be easily fetched via a search within that tree. The maximum value of t to consider is upper-bounded by the maximum length of the walks.

Assume that we insert an edge $e = (u, v)$. Let d_u be the out-degree of u *after* adding e. Consider a walk w that at some point got to u and continued to u's neighbors. If e was present in the graph at that point, with probability $1/d_u$ the walk w would have continued along e, and with probability $1 - 1/d_u$ the walk w would have chosen some other neighbor of u. However, w was sampled before e was in the graph, and our aim now is to correct this distribution and account for the insertion of e. The idea is to use rejection sampling, which we provide as Algorithm 2.

The for-loop on Line 3 of Algorithm 2 is in an efficient way of selecting walks passing through u and v that need to be re-sampled. Since the length of each walk follows a geometirc distribution with parameter $1 - \epsilon$, it is easy to see that with high probability the walks have length $O(\log n/\epsilon)$, and hence $\ell \in O(\log n/\epsilon)$.

Remark: To the best of our understanding, on an insertion of an edge (u, v), the prior work [24] re-samples a walk passing through u from the first occurrence of u in the walk, if there is any such occurrence (for details, see [24]). Such re-sampling does not account for the case when a walk passes through u multiple times and leads to biases in randomness.

4.2 Edge Deletions

Algorithm 3 presents our procedure executed after deleting an edge.

Let e be a deleted edge, and let $W_e \subseteq W$ be the list of walks passing through e. Clearly each $w \in W_e$ needs to be rerouted. The following lemma states that W updated by executing Algorithm 3 is a set of independent random walks.

Algorithm 2 A procedure executed after edge $e = (u, v)$ is inserted.

1: $W \leftarrow \emptyset$
2: Let ℓ be the length of longest generated walk.
3: **for** $t = 1 \dots \ell$ **do**
4: Sample each walk from $S_{u,t}$ with probability $1/d_u$ in the following way. First, select an integer $r_{u,t}$ from the binomial distribution with parameters $|S_{u,t}|$ and $1/d_u$. Second, select $r_{u,t}$ integers uniformly at random and without repetition from $[1, |S_{u,t}|]$. Then, for each of those integers i select the i-th walk from $S_{u,t}$. If e is an undirected edge, apply the same steps for $S_{v,t}$.
5: For each walk w selected in the last step such that $w \notin W$, add w to W and label w by t.
6: **end for**
7: **for** each $w \in W$ **do**
8: Let j be the label remembered for w on Line 5.
9: Generate walk w' with the following properties:
- The walks w and w' have the same length.
- The vertex-prefixes of length j of w and w' are the same.
- After that prefix, if w has more than j vertices, w' walks along e.
- The remaining edges of w' are chosen randomly, i.e., the rest of w' is a newly generated random walk.

10: Update the data structures by removing w and inserting w'.
11: **end for**

Algorithm 3 A procedure executed after edge e is deleted.

1: Let $W_e \subseteq W$ be the list of walks passing through e.
2: **for** $w \in W_e$ **do**
3: Let w_p be the longest prefix of w not containing e.
4: Let w' be a walk of length $|w|$ such that w' has w_p as its prefix, and the remainder of w' is a random walk.
5: To update W, remove w from W and the corresponding data structures, and insert w'.
6: **end for**

▶ **Lemma 11.** *Let W be the set of walks that our algorithm maintains. Assume that e gets deleted, and let W' be the updated list of walks as described in Algorithm 3. If W consists of random walks sampled independently, then W' is also a set of random walks sampled independently.*

Proof. The edges of walks throughout the algorithm are sampled independently of each other, so walks are independent by construction. We focus on showing how deletion of an edge affects randomness of a single walk.

Consider a walk $w \in W$ originating at vertex w_1. Let w_i be the i-th vertex of w, $w_{1\dots i}$ be the prefix of length i of w, and k be the length of w. Walk w is random iff for each $i \geq 2$ and each $u \in N(w_{i-1})$ it holds

$$\Pr\left[w_i = u \mid w_{1\dots i-1}\right] = \frac{1}{d(w_{i-1})}. \tag{3}$$

Let w' be the updated walk w, $d'(v)$ be the updated degree of vertices after e gets deleted and u' be a neighbor of w'_{i-1} after deletion of e. Note: we are **not** assuming that w contains e, so it might be the case that $w = w'$. We want to show that $\Pr\left[w'_i = u' \mid w'_{1\dots i-1}\right] = 1/d'(w'_{i-1})$. We have

$$\begin{aligned}
&\Pr\left[w'_i = u' \mid w'_{1\dots i-1}\right] && (4)\\
=&\Pr\left[w'_i = u' \mid w'_{1\dots i-1}, e \in w_{1\dots i}\right] \cdot \Pr\left[e \in w_{1\dots i}\right] && (5)\\
&+ \Pr\left[w'_i = u' \mid w'_{1\dots i-1}, e \notin w_{1\dots i}\right] \cdot \Pr\left[e \notin w_{1\dots i}\right]. && (6)
\end{aligned}$$

Analyzing (5)**.** We first handle (5). Recall that w' is constructed by keeping only the prefix of w up to the first occurrence of e, and the rest of the walk of w' is random and independent of any other state of the algorithm (see Algorithm 3). Hence, we have

$$\Pr\left[w'_i = u' \mid w'_{1\dots i-1}, e \in w_{1\dots i}\right] = \frac{1}{d'(w'_{i-1})}.$$

Analyzing (6)**.** Now consider term (6). If $w_{1\dots i}$ does not contain e, then $w'_{1\dots i} = w_{1\dots i}$ and we have

$$\begin{aligned}
&\Pr\left[w'_i = u' \mid w'_{1\dots i-1}, e \notin w_{1\dots i}\right]\\
=&\Pr\left[w_i = u' \mid w_{1\dots i-1}, e \notin w_{1\dots i-1}, e \neq \{w_{i-1}, w_i\}\right].
\end{aligned}$$

There are two cases:

(a) Case $w_{i-1} \notin e$: from (3) we have

$$\begin{aligned}
&\Pr\left[w_i = u' \mid w_{1\dots i-1}, e \notin w_{1\dots i-1}, e \neq \{w_{i-1}, w_i\}, w_{i-1} \notin e\right]\\
=&\Pr\left[w_i = u' \mid w_{1\dots i-1}, e \neq \{w_{i-1}, w_i\}, w_{i-1} \notin e\right]\\
=&\frac{1}{d(w_{i-1})} = \frac{1}{d'(w_{i-1})} = \frac{1}{d'(w'_{i-1})}.
\end{aligned}$$

In the last chain of equalities we used that once we condition on $w_{1\dots i-1}$, then (3) is a function of only w_{i-1} and not on any other content of $w_{1\dots i-1}$, e.g., whether $e \in w_{1\dots i-1}$ or not.

Note: The choice of e is independent of our data structures and the randomness the algorithm uses. However, in the case of non-oblivious adversary, i.e., in case of the adversary who sees the state of our algorithm, the updated edge e could be chosen based on the randomness used to generate w, and hence the above sequence of equalities would not hold.

(b) Case $w_{i-1} \in e$: we have the following

$$\begin{aligned}
&\Pr\left[w_i = u' \mid w_{1\dots i-1}, e \notin w_{1\dots i-1}, e \neq \{w_{i-1}, w_i\}, w_{i-1} \in e\right]\\
=&\frac{\Pr\left[w_i = u' \wedge e \neq \{w_{i-1}, w_i\} \mid w_{1\dots i-1}, e \notin w_{1\dots i-1}, w_{i-1} \in e\right]}{\Pr\left[e \neq \{w_{i-1}, w_i\} \mid w_{1\dots i-1}, e \notin w_{1\dots i-1}, w_{i-1} \in e\right]}\\
=&\frac{1/d(w_{i-1})}{(d(w_{i-1})-1)/d(w_{i-1})} = \frac{1}{d(w_{i-1})-1} = \frac{1}{d'(w'_{i-1})}.
\end{aligned}$$

Showing (3) for w'. The analysis of (5) and (6) together with (4) implies

$$\begin{aligned}&\Pr\left[w'_i = u' \mid w'_{1\dots i-1}\right]\\ =&\frac{1}{d'(w'_{i-1})}\cdot\Pr\left[e \in w_{1\dots i}\right] + \frac{1}{d'(w'_{i-1})}\cdot\Pr\left[e \notin w_{1\dots i}\right]\\ =&\frac{1}{d'(w'_{i-1})}.\end{aligned}$$

◀

4.2.1 Re-sampling Walks from Scratch

We now give a simple example that shows why re-sampling affected walks from scratch after a deletion would not properly maintain random walks. We note that this approach was suggested as a valid alternative by Bahmani et al. [4].

Consider a path graph on 5 vertices; let the graph be $1-2-3-4-5$. Consider a random walk w of length 2 originating at vertex 3 and visiting vertices w_1, w_2, w_3, i.e., $w_1 = 3$. Next, a deletion of $e = \{4,5\}$ occurs. Let w' be obtained from w as follows: if w contains e, then w' is a new random walk of length 2 originating at 3; otherwise, w' equals w. Now, if we denote the vertices on w' by w'_1, w'_2, w'_3, we have

$$\begin{aligned}\Pr\left[w'_2 = 4\right] =& \Pr\left[w'_2 = 4 \mid \{4,5\} \notin w\right]\Pr\left[\{4,5\} \notin w\right]\\ &+ \Pr\left[w'_2 = 4 \mid \{4,5\} \in w\right]\Pr\left[\{4,5\} \in w\right]\\ =& \Pr\left[w_2 = 4 \mid \{4,5\} \notin w\right]\Pr\left[\{4,5\} \notin w\right]\\ &+ \Pr\left[w'_2 = 4 \mid \{4,5\} \in w\right]\Pr\left[\{4,5\} \in w\right]\\ =& \Pr\left[w_2 = 4 \text{ and } \{4,5\} \notin w\right] + \frac{1}{2}\cdot\frac{1}{4}\\ =& \frac{1}{4} + \frac{1}{8}.\end{aligned}$$

However, for w' to be random it should hold $\Pr\left[w'_2 = 4\right] = 1/2$.

5 Near-Optimal Additive Approximation Algorithm

In this section, we analyze the algorithm from Section 4 in the context of dynamically maintaining *additive* approximation of PageRank. Namely, we show that when considering the incremental or decremental setting for directed graphs, an α additive PageRank approximation can be maintained in $(1/\alpha)^{O_\epsilon(\log\log n)}$ amortized update time, even for an adversarially chosen graph and a sequence of edge updates. Perhaps surprisingly, Theorem 1 shows that, for a constant ϵ, this running time complexity is essentially tight.

▶ **Theorem 2.** *For any $\epsilon \in (0,1)$, there is an algorithm that with high probability explicitly maintains an α additive approximation of PageRank of any graph G in either incremental or decremental setting. The algorithm processes the entire sequence of updates in $O(m) + n \cdot (1/\alpha)^{O_\epsilon(\log\log n)}$ total time and works correctly against an oblivious adversary.*

Our new analysis is based on two ideas. First, we show that if we *limit the lengths* of walks in Algorithm 1 to a constant, we obtain a constant additive approximation of the PageRank vector. This is thanks to the fact that a constant fraction of all walks have length $O(1/\epsilon)$, and so this truncation only affects a constant factor of the walks.

▶ **Lemma 12.** *Let π be the PageRank of a directed graph G. Then, with high probability, Algorithm 1 for $\ell = \lceil 2/\epsilon \cdot \log(2/(\alpha\epsilon))\rceil$ outputs a vector π_{ADD} such that $\|\pi - \pi_{ADD}\|_1 \le 5\frac{\alpha}{1-\epsilon}$.*

To keep the flow of high-level ideas uninterrupted, the proof of Lemma 12 is given in Section 5.1.

The second idea is an observation which bounds the maximum number of times a walk can be affected by adding edges (edge deletions can use a symmetric argument). To explain the idea let us see what happens when we want to maintain a random outgoing edge e of a vertex undergoing insertions of outgoing edges. Clearly when we insert the d-th outgoing edge we need to update e to be equal to d with probability $1/d$. By a harmonic sum argument, the expected number of times e needs to be updated in the course if k insertions is only $O(\log k)$. We generalize this argument to walks of length ℓ as follows.

▶ **Lemma 13.** *Let G be a directed graph undergoing edge insertions (or deletions). The total number of times a random walk of length ℓ is being regenerated is bounded by $O(\log^{\ell} n)$ in expectation.*

Proof. We are going to prove this bound by induction, i.e., let us denote by $f(i)$ the upper bound on expected number of times the walk of length i is regenerated. Consider a random walk w of length 1 starting in a vertex v. Consider insertion of an edge incident to v. The probability that w is regenerated at this moment is $1/d_v$. As we consider incremental setting the expected number of times w is regenerated is bounded by

$$f(1) = \sum_{i=1}^{n} \frac{1}{i} \leq \ln n.$$

Now consider a walk w of length ℓ starting at v. Similarly as above we can bound the number of changes to w as

$$f(\ell) = \sum_{i=1}^{n} \frac{1}{i} \cdot f(\ell - 1) \leq \ln n \cdot f(\ell - 1) = \ln^{\ell} n,$$

what finishes the proof. Symmetric argument can be applied in the decremental case. ◀

The above lemma implies that for $\ell = \lceil 2/\epsilon \cdot \log(2/(\alpha\epsilon)) \rceil$ the amortized cost of maintaining each walk is $(1/\alpha)^{O(\log\log n)}$ for a constant ϵ. As we generate $O(n \log n)$ walks in Algorithm 1 the total cost of maintaining $5\alpha/(1-\epsilon)$-approximation in incremental or decremental setting is $O(m + n \cdot (1/\alpha)^{O(\log\log n)})$.

5.1 Proof of Lemma 12

Define $\hat{\ell} = \lceil 2/\epsilon \cdot \log(2/(\alpha\epsilon)) \rceil$. Let $\tilde{\pi}$ be the output of Algorithm 1 for $\ell = \infty$, and π_{ADD} the output for $\ell = \hat{\ell}$. As discussed, it is known, e.g., see [4, 16], that $|\pi_v - \tilde{\pi}_v| \leq \alpha\pi_v$. As $\sum_v \pi_v = 1$, this further implies $\|\pi - \tilde{\pi}\|_1 \leq \alpha$.

Next, we compare π_{ADD} and $\tilde{\pi}$. Difference between these two vectors can be expressed by the following two quantities: (1) $|W|$, which in turn affects the scaling on Line 5; and (2) the value of $\mathbf{X}_v$, which affects the numerator on Line 5. We analyze both of these quantities.

Analysis for $|W|$. For $\ell = \hat{\ell}$, a walk has length at most $\hat{\ell}$ with probability $\epsilon \sum_{j=0}^{\hat{\ell}} (1-\epsilon)^j = 1 - (1-\epsilon)^{\hat{\ell}+1} \geq 1 - \epsilon/2$, where we used that $1 - x \leq e^{-x}$ for $x \in [0, 1/2]$. Hence, $\mathbb{E}[|W|] \geq nR(1 - \epsilon/2)$. By using a Chernoff bound we can prove that with high probability it holds $|W| \geq nR(1-\epsilon)$. The proof proceeds as follows. In the summation above, there are only ℓ different values of j that affect $\mathbb{E}[|W|]$. For a fixed j, the contribution to $|W|$ can be expressed as a sum of **independent** 0/1 random variables – a random variable per each

of the nR walks, denoting whether the given walk has length length j or not. Hence, for a fixed j we apply the Chernoff bound to show it concentrates well, and then by the union bound over all ℓ values of j we get the desired concentration for $|W|$.

Analysis for $\mathbf{X}_v$. By definition, π_{ADD} only accounts for the contribution to $\mathbf{X}_v$ by the appearances of v which are within walks of length at most ℓ; $\mathbf{X}_v$ is defined in Algorithm 1. Let $\mathbf{Y}_v$ be the appearances of v for which π_{ADD} does not but $\tilde{\pi}$ does account for.

Now, we upper-bound $\sum_v \mathbf{Y}_v$:

$$\begin{aligned}
\mathbb{E}\left[\sum_v \mathbf{Y}_v\right] =& nR(1-\epsilon)^{\hat{\ell}+1}\cdot(\hat{\ell}+1) + \sum_{j=\hat{\ell}+2}^{\infty} nR(1-\epsilon)^j \\
\leq& 2nR\alpha + nR(1-\epsilon)^{\hat{\ell}+2}\sum_{j=0}^{\infty}(1-\epsilon)^j \\
=& 2nR\alpha + \frac{nR}{\epsilon}(1-\epsilon)^{\ell+2} \\
\leq& 2nR\alpha + nR\epsilon\alpha^2/4 \\
\leq& 3nR\alpha.
\end{aligned}$$

In the derivation above, we used $(1-\epsilon)^{\hat{\ell}+1}(\hat{\ell}+1) \leq (\alpha\epsilon/2)^2(\hat{\ell}+1) \leq (\alpha\epsilon/2)^2 2\hat{\ell} \leq 2\alpha$. To prove that $\sum_v \mathbf{Y}_v \leq 4nR\alpha$ with high probability, it suffices to proceed the same way as for our analysis of $\mathbb{E}[|W|]$. In the analysis, we need the observation that $\sum_{j>c\log n/\epsilon} nR(1-\epsilon)^j < 1/n$ for a sufficiently large constant c. In other words, there are only $O(\log n)$ different values of j that substantially contribute to $\sum_v \mathbf{Y}_v$ and over which is needed to take the union bound.

Our analysis now implies that additive approximation of Algorithm 1 for $\ell = \hat{\ell}$ is with high probability upper-bounded by $\frac{\alpha}{1-\epsilon} + 4\frac{\alpha\epsilon}{1-\epsilon} \leq 5\frac{\alpha}{1-\epsilon}$. The first term is coming from the fact that π_{ADD} is computed by rescaling $\mathbf{X}_v$ by $|W|/\epsilon \geq (1-\epsilon)nR/\epsilon$ as opposed to rescaling by nR/ϵ, as it is done when computing $\tilde{\pi}$. The second term is coming from the fact that the loss between $\tilde{\pi}$ and π_{ADD} in the numerator of Line 5 is at most $4nR\alpha$ with high probability, which is divided by $|W|/\epsilon \geq nR(1-\epsilon)/\epsilon$.

6 Efficient Multiplicative Approximation in Undirected Graphs

In this section, we describe how to maintain approximate PageRank of undirected graphs under edge deletions and insertions even if the goal is to maintain a multiplicative approximation. Our approach takes $\operatorname{polylog} n$ time per update and is also based on the algorithm from Section 4.

▶ **Theorem 4.** *For any $\epsilon \in (0,1)$, there is an algorithm that with high probability explicitly maintains a $1+\alpha$ multiplicative approximation of PageRank of any undirected graph G in the fully dynamic setting. The algorithm handles each update in $O(\log^5 n/(\epsilon^2\alpha^2))$ time and works correctly against an oblivious adversary.*

Our analysis relies on the following (folklore) claim, which states that the number of the walks passing through an edge is fairly small.

▶ **Lemma 14** (Folklore). *Let G be an undirected graph. Consider a set of random walks W of length $\ell < n$ each, such that there are d_v walks originating at vertex v. Then, with high probability an edge e is contained in $O(\ell \cdot \log n)$ of those walks.*

Proof. Observe that the number of walks in W originating at each vertex v is proportional to the stationary distribution of v. Hence, the number of walks of W whose i-th vertex is v in expectation equals d_v, for each $1 \leq i \leq \ell$. Therefore, the number of walks of W whose i-th edge is $e = \{u, v\}$ (either as $u \to v$ or $v \to v$) in expectation equals 2, for each $1 \leq i \leq \ell$.

Let $X_{e,i}$ be the number of walks whose i-th edge equals i. From our discussion, $\mathbb{E}[X_{e,i}] = 2$. Also, $X_{e,i}$ is a sum of 0/1 independent random variables $Y_{v,j,i}$, where $Y_{v,j,i}$ means that the i-th edge of the j-th walk originating at v equals e. Hence, by applying the Chernoff bound, we obtain that with high probability it holds that $X_{e,i} \in O(\log n)$. By taking the union bound over all $1 \leq i \leq \ell$ and over all the vertices, we prove the desired claim. ◀

As a direct consequence of Lemma 14 we obtain the following claim.

▶ **Corollary 15.** *Consider $n \cdot t$ independent random walks of length $\ell \in O(\log n/\epsilon)$ such that from each vertex there are t walks originating. Then, with high probability an edge e is contained in $O(t \log^2 n/\epsilon)$ of those walks.*

In Section 4, we describe how to update our data structures in $O(\ell \cdot \log n)$ time per an update of an ℓ-length walk. Since Algorithm 1 runs $t = R = O(\log n/(\epsilon\alpha^2))$ random walks per vertex, by Corollary 15 there are $O(\log^3 n/(\epsilon\alpha^2))$ walks passing through each edge. Thus by the fact that walks have lengths $O(\log n/\epsilon)$ with high probability, the dynamic algorithm requires $O(\log^5 n/(\epsilon^2\alpha^2))$ time for each update, which yields Theorem 4.

References

1 Madhav Aggarwal, Bingyi Zhang, and Viktor Prasanna. Performance of local push algorithms for personalized pagerank on multi-core platforms. In *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pages 370–375. IEEE, 2021.

2 Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcraft, Vahab S Mirrokni, and Shang-Hua Teng. Local computation of PageRank contributions. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 150–165. Springer, 2007.

3 Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 475–486. IEEE, 2006.

4 Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized PageRank. *Proc. VLDB Endow.*, 4(3):173–184, December 2010. doi:10.14778/1929861.1929864.

5 Bahman Bahmani, Ravi Kumar, Mohammad Mahdian, and Eli Upfal. PageRank on an evolving graph. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 24–32, 2012.

6 Marco Bressan, Enoch Peserico, and Luca Pretto. Sublinear algorithms for local graph centrality estimation. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 709–718. IEEE, 2018.

7 Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, April 1998. doi:10.1016/S0169-7552(98)00110-X.

8 Soumen Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *Proceedings of the 16th international conference on World Wide Web*, pages 571–580, 2007.

9 Wentian Guo, Yuchen Li, Mo Sha, and Kian-Lee Tan. Parallel personalized PageRank on dynamic graphs. *Proceedings of the VLDB Endowment*, 11(1):93–106, 2017.

10 Kyung Soo Kim and Yong Suk Choi. Incremental iteration method for fast PageRank computation. In *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication*, IMCOM '15, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2701126.2701165.

11 Qun Liao, ShuangShuang Jiang, Min Yu, Yulu Yang, and Tao Li. Monte Carlo based incremental PageRank on evolving graphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 356–367. Springer, 2017.

12 Peter Lofgren. On the complexity of the monte carlo method for incremental pagerank. *Inf. Process. Lett.*, 114(3):104–106, 2014. doi:10.1016/J.IPL.2013.11.006.

13 Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. Personalized pagerank estimation and search: A bidirectional approach. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 163–172, 2016.

14 Peter Lofgren and Ashish Goel. Personalized PageRank to a target node. *arXiv preprint*, 2013. arXiv:1304.4658.

15 Peter A Lofgren, Siddhartha Banerjee, Ashish Goel, and C Seshadhri. FAST-PPR: Scaling personalized PageRank estimation for large graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1436–1445, 2014.

16 Jakub Łącki, Slobodan Mitrović, Krzysztof Onak, and Piotr Sankowski. Walking randomly, massively, and efficiently. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 364–377, 2020.

17 Amit Pathak, Soumen Chakrabarti, and Manish Gupta. Index design for dynamic personalized pagerank. In *2008 IEEE 24th International Conference on Data Engineering*, pages 1489–1491. IEEE, 2008.

18 Ryan A Rossi and David F Gleich. Dynamic pagerank using evolving teleportation. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 126–137. Springer, 2012.

19 Subhajit Sahu, Kishore Kothapalli, and Dip Sankar Banerjee. Dynamic batch parallel algorithms for updating pagerank. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1129–1138. IEEE, 2022.

20 Scott Sallinen, Juntong Luo, and Matei Ripeanu. Real-time pagerank on dynamic graphs. In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*, pages 239–251, 2023.

21 Hanzhi Wang, Zhewei Wei, Junhao Gan, Sibo Wang, and Zengfeng Huang. Personalized PageRank to a target node, revisited. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 657–667, 2020.

22 Hanzhi Wang, Zhewei Wei, Ji-Rong Wen, and Mingji Yang. Revisiting local computation of pagerank: Simple and optimal. In *STOC'24*, 2024.

23 Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, January 2008. doi:10.1007/s10115-007-0114-2.

24 Zexing Zhan, Ruimin Hu, Xiyue Gao, and Nian Huai. Fast incremental pagerank on dynamic networks. In *International Conference on Web Engineering*, pages 154–168. Springer, 2019.

25 Hongyang Zhang, Peter Lofgren, and Ashish Goel. Approximate personalized PageRank on dynamic graphs. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1315–1324, 2016.

A Sublinear Time Tester for Max-Cut on Clusterable Graphs

Agastya Vibhuti Jha
École polytechnique fédérale de Lausanne, Switzerland

Akash Kumar[1]
Indian Institute of Technology, Bombay, India

Abstract

One natural question in the area of sublinear time algorithms asks whether we can distinguish between graphs with max-cut value at least $1-\varepsilon$ from graphs with max-cut value at most $1/2+\varepsilon$ in the adjacency list model where we can make degree queries and neighbor queries. Chiplunkar, Kapralov, Khanna, Mousavifar, and Peres (FOCS' 18) showed that in graphs of bounded degree, one cannot hope for a factor $1/2+\varepsilon$ approximation to the max-cut value in time $n^{1/2+o(\varepsilon)}$. Recently, Peng and Yoshida (SODA '23) obtained $o(n)$ time algorithms which can distinguish expanders with max-cut value at least $1-\varepsilon$ from expanders with small max-cut value (their running time is $n^{1/2+O(\varepsilon)}$). In this paper, going beyond the results of Peng-Yoshida, we develop sublinear time algorithms for this problem on clusterable graphs (which is a graph class with a good community structure). Our algorithms run in $\approx n^{0.5001+O(\varepsilon)}$ time.

A natural extension of Peng-Yoshida approach does not seem to work for clusterable graphs. Indeed, their random walk based technique tracks the ℓ_2 length of random walk vectors and they exploit the difference in the length of these vectors to tell apart expanders with large cut value from expanders with small cut-value. Such approaches fail to be reliable when graph has loosely connected clusters. Taking inspiration from [4], we exploit the more refined geometry of spectra of clusterable graphs which leads to our sublinear time implementation. We prove a novel spectral lemma which shows that in a spectral expander $2-\lambda_{n-1} \geq \Omega(\lambda_2)$. This lemma is leveraged to show that there is a suitable difference between spectra of clusterable graphs with large cut value and spectra of clusterable graphs with small cut value. We use this gap to obtain our sublinear time implementation. To do this, we obtain a nuanced understanding of the eigenvector structure of clusterable graphs and in particular, we show that the eigenvectors of the normalized Laplacian of a clusterable graph, corresponding to eigenvalues which are close to 2 have a small infinity norm.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Mathematics of computing → Spectra of graphs

Keywords and phrases Sublinear Algorithms, Graph Algorithms, Clusterable Graphs, Property Testung

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.91

Category Track A: Algorithms, Complexity and Games

Acknowledgements We sincerely thank Michael Kapralov for insightful discussions at the beginning of the project. We also would like to thank Kshiteej Sheth and Weronika Wrozs-Kominska for being helping us bounce off ideas.

1 Introduction

Max-Cut is a fundamental algorithmic problem and has several applications in computer science. In this problem, we are given a graph $G=(V,E)$ as input and we are asked to find a bipartition $(S,\overline{S})$ of vertices which has the maximum number of edges going across. Let

[1] Part of this work was done when the author was a postdoc at EPFL.


Max-Cut(G) denote the fraction of edges cut by the maximizing bipartition. The decision version of Max-Cut was shown to be NP-Complete by Karp in his famous list of 21 problems in [11]. A 0.878 approximation algorithm for Max-Cut was achieved in the seminal work of [8] which was shown to be tight assuming the unique games conjecture.

While Max-Cut is interesting on general graphs, it is also intriguing when restricted to important graph classes. For instance, [2] provided algorithms for finding cuts in expanders with Max-Cut(G) $\geq 1-\gamma$ (for some sufficiently small $\gamma > 0$) that are crossed by at least $(1 - O(\gamma))$ fraction of the edges, which improves (in small γ regime) upon the Goemans-Williamson bound of $(1 - O(\sqrt{\gamma}))$ fraction of edges. In another direction, more relevant to this paper, a crucially important step was taken by [9] who presented algorithms for testing bipartiteness in bounded-degree graphs, assuming query access to the adjacency list of the input. This algorithm decides in sublinear time whether the input bounded-degree graph has Max-Cut(G) $= 1$ or whether has Max-Cut(G) $< 1-\gamma$. The authors also proposed a two-step rule of thumb for approaching a wide variety of property testing problems in bounded-degree graphs, which involves developing property testing algorithms assuming the input graph is an expander, and then using tools from expander decompositions to break the graph into a collection of expanding components with inverse-polylogarithmic expansion.

Until recently however, no sublinear time algorithms were known for approximating Max-Cut even on expanding graphs which approximate the cut-value to within a factor better than 1/2. This was remedied by [13] who gave sublinear algorithms for approximating Max-Cut on expanders in the adjacency list model. In this work, we focus on the adjacency list model and provide sublinear time algorithms for Max-Cut on a natural relaxation of expanders, namely, the family of $(k, \varphi, \varepsilon)$-clusterable graphs. Briefly, a degree d-bounded graph $G = (V, E)$ is $(k, \varphi, \varepsilon)$-clusterable if the vertex set can be partitioned into k sets, each with *inner conductance* at least φ and *outer conductance* at most ε. This graph class has been considered in several recent works on property testing [5, 4, 7]. Our main theorem (informal version below) concerns this graph class and asserts the following:

▶ **Theorem 1.** *Fix $k \in \mathbb{N}, \varphi < 1$ and $0 < \varepsilon, \gamma < \delta\varphi^2$ where $\delta = 10^{-5}$. Then there exists an algorithm which on input a $(k, \varphi, \varepsilon)$-clusterable graph runs in time $\approx n^{1/2+100\delta+O(\varepsilon/\varphi^2)}$ and returns*

- *Yes, if Max-Cut(G) $\geq 1-\gamma$*
- *No, if Max-Cut(G) $\leq 1/2+\gamma$*

Broadly speaking, this problem of distinguishing clusterable graphs with large max-cut value from clusterable graphs with small max-cut value is a special sub-problem of the more general question which seeks to develop tolerant testers for max-cut. Some complexity considerations related to the Unique Games Conjecture seem to suggest that this problem does not admit a $(1-\gamma, 1-\sqrt{\gamma})$ tolerant tester in the adjacency list query model. [3] even showed that there is no sublinear time algorithm for the Max-Cut problem with approximation ratio better than 16/17. It is an open question to chase down the parameter range for which one might expect a sublinear time algorithm for a better than one-half approximation of Max-Cut on a class of graphs richer than expanders. Our results can be viewed as taking the first step in this direction.

2 Preliminaries

In the following, we will let $G = (V, E)$ denote a graph.

▶ **Definition 2.** *The normalized adjacency matrix $\bar{\boldsymbol{A}}$ is $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$, where $\boldsymbol{D}$ is the diagonal of degrees. The normalized Laplacian is $\bar{\boldsymbol{L}} = \boldsymbol{I} - \bar{\boldsymbol{A}}$.*

The *random walk associated with* G is defined to be the random walk with transition matrix $\boldsymbol{AD}^{-1}$. Note that, unlike the previous works in *property-testing*, we do a *non-lazy* walk over G.

▶ **Definition 3** ([10], Rayleigh Quotient). *Let $\boldsymbol{A}$ be a matrix in $\mathbb{R}^{n\times n}$ and let $\boldsymbol{x}$ be a non-zero vector in $\mathbb{R}^n$. Then, the Rayleigh quotient of $\boldsymbol{x}$ with respect to $\boldsymbol{A}$ is defined as:*

$$\mathcal{R}_{\boldsymbol{A}}(\boldsymbol{x}) = \frac{\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}}.$$

- For any arbitrary matrix $\boldsymbol{B}$ in $\mathbb{R}^{n\times n}$, we use $\mu_1 \leq \mu_2 \leq \ldots \leq \mu_n$ to denote its eigenvalues in ascending order, and $\nu_1 \geq \nu_2 \geq \ldots \geq \nu_n$ to denote its eigenvalues in descending order.
- Given a graph $G = (V, E)$, we let $\mathbf{1}_x \in \mathbb{R}^V$ to denote the indicator vector for a vertex x in V. For a multi-set of vertices $\{x_1, x_2, \ldots x_k\}$, we let $S \in \mathbb{R}^{n\times k}$ denote the matrix of indicators, where the j^{th} column of S is the vector $\mathbf{1}_{x_j}$ for $1 \leq j \leq k$.
- (*Informal*) Given a graph $G = (V, E)$, and its normalized Laplacian $\bar{\boldsymbol{L}}$, we will refer to the eigenvectors with corresponding eigenvalues close to 0 (resp. eigenvalues close to 2) as the *clusterability* eigenvectors (resp. *Max-Cut* eigenvectors). The notion of close to 0 (resp. close to 2) will be made clear in the context.

▶ **Theorem 4** ([10], Spectral Theorem). *Let $\boldsymbol{A}$ be a real symmetric matrix. Then, there exists an orthonormal basis of $\mathbb{R}^n$ consisting of eigenvectors of $\boldsymbol{A}$ and all the eigenvalues of $\boldsymbol{A}$ are real.*

▶ **Theorem 5** ([10], Courant-Fischer). *Let $\boldsymbol{A}$ be a real symmetric matrix in $\mathbb{R}^{n\times n}$, let $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ be its eigenvalues. Then, for any $1 \leq k \leq n$,*

$$\lambda_k = \min_{\mathcal{U}} \max_{x\neq 0}\{\{R_{\boldsymbol{A}}(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{U}\} \mid dim\, \mathcal{U} = k\},$$

and

$$\lambda_{n-k+1} = \max_{\mathcal{U}} \min_{\boldsymbol{x}\neq 0}\{\{R_{\boldsymbol{A}}(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{U}\} \mid dim\, \mathcal{U} = k\}.$$

▶ **Lemma 6** ([10], Weyl's Inequality). *Let $\boldsymbol{A}$ and $\boldsymbol{E}$ be real symmetric matrices in $\mathbb{R}^{n\times n}$. Then, for all $i \in \{1, 2, \ldots, n\}$,*

$$\lambda_i(\boldsymbol{A}) + \lambda_{min}(\boldsymbol{E}) \leq \lambda_i(\boldsymbol{A} + \boldsymbol{E}) \leq \lambda_i(\boldsymbol{A}) + \lambda_{max}(\boldsymbol{E}).$$

▶ **Lemma 7** ([10]). *For any $m \times n$ matrix $\boldsymbol{A}$ and $n \times m$ matrix $\boldsymbol{B}$, the multisets of nonzero eigenvalues of $\boldsymbol{AB}$ and $\boldsymbol{BA}$ are the same. In particular, if one of $\boldsymbol{AB}$ and $\boldsymbol{BA}$ is positive semi-definite, then $\mu_h(\boldsymbol{AB}) = \mu_h(\boldsymbol{BA})$.*

▶ **Lemma 8** (Folklore). *Let $\boldsymbol{A}$ and $\boldsymbol{B}$ denote two positive semidefinite matrices in $\mathbb{R}^{n\times n}$. Then $\nu_{max}(\boldsymbol{AB}) \leq \nu_{max}(\boldsymbol{A})\,\nu_{max}(\boldsymbol{B})$.*

▶ **Definition 9.** *Given a graph $G = (V, E)$ and a set $S \subseteq V$, we define $vol(S) = \sum_{i\in S} deg(i)$.*

▶ **Definition 10** (Inner and Outer Conductance). *Let $G = (V, E)$ be a graph. For a set $S \subseteq C \subseteq V$, we define the conductance of S within C as $\varphi^C_{in}(S) = \frac{|E(S,C\setminus S)|}{\sum_{i\in S} deg(i)} = \frac{|E(S,C\setminus S)|}{vol(S)}$. The inner conductance of a set $C \subseteq V$ is defined as*

$$\varphi_{in}(C) = \min_{\substack{S\subseteq C \\ 0<vol(S)\leq vol(C)/2}} \varphi^C_{in}(S).$$

We define the outer conductance of a set $C \subseteq V$ to be $\varphi_{out}(C) = \frac{|E(C,V\setminus C)|}{vol(C)}$.

▶ **Theorem 11** (Cheeger's Inequality, Folklore). *Let G be a graph. Let $\bar{\boldsymbol{L}}$ denote its normalized Laplacian. Then,*

$$\frac{\phi^2(G)}{2} \leq \lambda_2 \leq 2\phi(G),$$

where λ_2 denotes the second smallest eigenvalue of the normalized Laplacian of G.

▶ **Definition 12** ((k,φ,ε)-clusterable graphs). *A graph $G=(V,E)$ is said to admit a (k,φ,ε)-clustering if there exists a partition of V into k sets $C_1, C_2, \ldots C_k$ such that each C_i satisfies $\varphi_{in}(C_i) \geq \varphi$ and $\varphi_{out}(C_i) \leq \varepsilon$ and for all $i,j \in [k]$ it holds that $\frac{|C_i|}{|C_j|} = O(1)$.*

When the parameters φ and ε are clear from the context, we will often refer to these graphs as k-clusterable graphs and sometimes even as clusterable graphs when the parameter k is also clear from the context. [7] consider clusterable graphs where $\varepsilon/\varphi^2 \leq \delta := 10^{-5}$. We also consider a similar parameter regime.

▶ **Theorem 13** ([7], Clusterability Eigengap). *Let G be a graph that admits a (k,φ,ε)-clustering. Let $0 \leq \lambda_1 \leq \lambda_2 \leq \ldots \lambda_n$ denote the spectrum of its normalized Laplacian. Then, $\lambda_k \leq 2\varepsilon$ and $\lambda_{k+1} \geq \varphi^2/2$.*

3 Technical Overview

The algorithmic problem of getting a better than $1/2$ approximation algorithm for the max-cut-value of a degree bounded graph was ushered to the frontlines of research in sublinear-time algorithms in the work of [4]. This paper shows that any algorithm that returns an estimate to the max-cut-value that is at least $1/2+\varepsilon$ must make an $n^{1/2+\Omega(\varepsilon)}$ number of queries to the adjacency list of G. After this work, one natural next step is to ask whether there are algorithms that approximate max-cut to within some approximation factor better than $1/2$ on some rich enough class of interesting graphs. A progress was reported on this endeavor in the work of [13] who obtained a such an estimate to the max-cut-value on expanders of bounded degree.

As our starting point, we describe at a high level, the approach used in [13] for deciding whether the max-cut value of an input expander is large or whether it is small. The starting point of this work adapts the techniques used in the pioneering work of [9] to obtain a tester for deciding whether Max-Cut(G) is close to 1 or bounded away from 1 on expanders. One can think of a φ-expander as a (k,φ,ε)-clusterable graph with $k=1$ and $\varepsilon=0$ (see Definition 12). Let us now describe the high level ideas that underlie Peng-Yoshida algorithm. In particular, [13] note that on an expanding instance with large max-cut value, the following distributions over end-points of ℓ length lazy walks are fairly far:

- $\mathcal{D}_{v,e}$: The end-point distribution supported over vertices reached in an ℓ-step walk with the effective length (that is, number of steps left after deleting all loops) being an even number.
- $\mathcal{D}_{v,o}$: The end-point distribution supported over vertices reached in an ℓ-step walk with the effective length being an odd number.

As mentioned ealier, the intuition behind this argument comes from [9] which considers the case where Max-Cut(G) $= 1$. In this case, note that the distributions are disjointly supported and thus indeed the ℓ_2^2 distance between the distributions is large.

While the Peng-Yoshida algorithm extends the Goldreich-Ron bipartiteness testing algorithm in a very elegant way, unfortunately, this algorithm does not extend to the (k,φ,ε)-clusterable case even for $k=2$. To see this, let us take the following graph. It contains two

disjoint isomorphic $(1/\varepsilon - 2)$-regular[2] bipartite φ-expanders which are sparsely connected (and we will describe what these corss-edges are momentarily). We denote the first bipartite graph as (A_1, B_1) and the other one as (A_2, B_2) where $|A_1| = |B_1| = |A_2| = |B_2| = n/4$. Next, we connect A_1 and A_2 with a perfect matching and we also connect A_1 and B_2 with another perfect matching. We also add a perfect matching between B_1 and A_2 and another one between B_1 and B_2. In all, this gives us a $1/\varepsilon$-regular $(2, \varphi, \varepsilon)$-clusreable graph which has $\mathsf{Max\text{-}Cut(G)} = 1 - O(\varepsilon)$. Now, consider performing a lazy walk of logarithmic length from *any* start vertex in this instance. Note that the walk reveses its "polarity" once every $2/\varepsilon$ steps in expectation. In particular, this means that the distributions $\mathcal{D}_{v,o}$ and $\mathcal{D}_{v,e}$ are fairly close and one can no longer use the distance between these distributions as a reliable indicator for whether the max-cut-value is large or whether it is small.

We want to circumvent this obstacle and obtain a better than 1/2-approximation to max-cut-value in sublinear time for k-clusterable graphs. To this end, for the ease of presentation in this overview, it will be convenient to make the following simplications listed in the remark below.

▶ Remark 14. We emphasize that all the simplifications made in this remark are only for the ease of presentation in this overview. Our main result (Theorem 16) and its proof does not rely on these simplifications.

1. *We will assume that graph is d-regular.*
2. *We will assume that all clusters in the input k-clusterable graph have the same size.*
3. *Recall we are trying to distinguish $(k, \varphi, \varepsilon)$-clusterable graphs with max-cut-value at least $1 - \gamma$ from graphs with max-cut-value at most $1/2 + \gamma$. It will additionally be convenient to assume that $\varphi = \Omega(1)$ and that ε and γ are sufficiently small constants with $\gamma = \Theta(\varepsilon)$. As stated in Theorem 16, we only need to have both ε and γ being at most $\delta\varphi^2$.*

Now, towards getting a better than 1/2 approximation to the max-cut-value, let us consider the following intuition: Suppose we are given a k-clusterable graph G with high max-cut-value. That is, we are told that $\mathsf{Max\text{-}Cut(G)} \geq 1 - \gamma$. In this case, by averaging, one notices that G has at least $\ell := 2k/3$ clusters which have induced max-cut-value at least $1 - O(\gamma)$. Now consider the graph that one gets after doing a two-step non-lazy walk on G. This is the graph G^2 where one puts a (parallel) edge between every pair of vertices between which there is a path of length two. Consider what this process does to a component with high induced max-cut-value. Intuitively, since (almost) all the edges run between the maximizing bipartition in this component, we get two sparsely connected components – one induced on each bipartition. And both of these bipartitions induce expanders as well. This way, we get one additional sparse cut in G^2 corresponding to every component with large induced max-cut-value. In particular, this means the $(k + \ell)$-th smallest eigenvalue of the Normalized Laplacian of G^2 is close to zero. Thus, this intuition suggests that one can track the $(k + \ell)$-th smallest eigenvalue of the normalized Laplacian of the graph which results after a non-lazy walk of some even length. Indeed, our algorithms are built off on this intuition.

Towards showing that this algorithm can reliably distinguish between k-clusterable graphs G with $\mathsf{Max\text{-}Cut(G)} \geq 1 - \gamma$ and k-clusterable graphs with $\mathsf{Max\text{-}Cut(G)} \leq 1/2 + \gamma$ (recall we assumed $\gamma = \Theta(\varepsilon)$ in Remark 14), we need to understand the spectra of instances in both of these regimes. Additionally, we need to show that the graph spectra in these two cases are appreciably different that a non-lazy random walk based algorithm can detect this difference. We now outline our algorithm. The algorithm proceeds by taking a multiset

[2] We ignore the integrality issues.

$S \subseteq V$ of samples with $|S| = poly(k) \cdot n^{O(\varepsilon)}$ in the hope of getting enough vertices from every cluster. Next up, setting the length of the walk to be $t = \frac{C \log n}{\varphi^2}$, the algorithm computes the Gram Matrix of collision probabilities $\boldsymbol{W}_S = (\boldsymbol{M}^t \boldsymbol{I}_S)^T (\boldsymbol{M}^t \boldsymbol{I}_S) \in \mathbb{R}^{|S| \times |S|}$ (here $\boldsymbol{I}_S$ denotes the identity matrix restricted to vertices of S). Finally, the algorithm just checks whether the $(k+\ell)$-th largest eigenvalue of $n/s \cdot \boldsymbol{W}_S$ is at least $n^{-O(\varepsilon)}$. If yes, the algorithm reports that the graph had max-cut-value close to 1 otherwise it reports that the graph had max-cut-value close to 1/2. In the following remark, we collect the remaining ingredients our analysis relies upon. In the remainder of the tech-overview we elaborate upon the ideas stressed in this remark.

▶ Remark 15. The intuition here comes from considering the matrix $\boldsymbol{W} = (\boldsymbol{M}^t)^T(\boldsymbol{M}) = \boldsymbol{M}^{2t}$. For a set $S \subseteq V$, we let $\boldsymbol{W}_S$ denote the matrix we obtain when we restrict the matrix $\boldsymbol{W}$ to rows and columns indexed by S.

1. *In Theorem 20, we show the following two items.*
   a. *If* Max-Cut(G) $\geq 1-\gamma$, *then the $(k+\ell)$-th eigenvalue of $\boldsymbol{M}^{2t}$ is at least $(1-\varepsilon)^{2t}$ which by the choice of $t = \frac{C \log n}{\varphi^2}$ means we get a lower bound of $n^{-O(\varepsilon)}$ on the $(k+\ell)$-th largest eigenvalue of $\boldsymbol{W}$.*
   b. *If* Max-Cut(G) $\leq 1/2+\gamma$, *we show the $(k+\ell)$-th largest eigenvalue of $\boldsymbol{W}$ is at most $(1-O(\varphi^2))^{2t}$ which by the choice of t can be shown to be at most n^{-C}.*
2. *Finally, one shows that the eigenvalues of $\boldsymbol{W}$ are very close to the corresponding eigenvalues of $n/s \cdot \boldsymbol{W}_S$. This goes via an application of Matrix Bernstein Bounds. Using these bounds requires a little more understanding of the eigenvector structure of the Laplacian of Clusterable instances with high max-cut-value which we also develop.*

Towards showing Item 1.(a) and Item 1.(b) mentioned in Remark 15, it is helpful to introduce a little notation. Let $\nu_1 \geq \nu_2 \geq \cdots \geq \nu_n$ denote the eigenvalues of the random walk matrix $\boldsymbol{M}$. For showing item 1.(a), note that using the easy direction of higher order Cheeger, we already have $\nu_k(\boldsymbol{M}) \geq 1 - 2\varepsilon$. In case 1.(a), we also know Max-Cut(G) $\geq 1-\varepsilon$ which additionally means that the last ℓ eigenvalues of $\boldsymbol{M}$ are close to -1 (and in particular we have $\nu_{n-\ell+1} \leq -1 + O(\varepsilon)$). This is because G has ℓ nearly bipartite components and therefore, we have ℓ disjointly supported vectors all of which have Rayleigh Quotient close to -1. In all, this means that $(k+\ell)$-th largest eigenvalue of $\boldsymbol{M}^{2t}$ is at least $(1-O(\varepsilon))^{2t}$ as desired.

Towards showing Item 1.(b), we prove an important *eigengap transportation* lemma (Lemma 21) in spectral graph theory which asserts that for any expander graph on n vertices we have $\lambda_{n-1} < 2 - \Omega(\lambda_2)$ (recall that λ's denote the eigenvalues of the Normalized Laplacian). Although, fairly intuitive, this seems to be a novel result. Indeed, a direct adaptation of techniques from [14] produces a bound saying $\lambda_{n-1} \leq 2 - \Omega(\lambda_2^2)$ as obtained in [12]. One can use this lemma to conclude that in case 1.(b), where Max-Cut(G) $\leq 1/2+\varepsilon$, $\nu_{n-k+1} \gg -1$. Additionally, since G has such a small max-cut-value, we can show that there at least $\ell := 2k/3$ clusters in G which have induced max-cut-value close to 1/2. It can be shown that corresponding to every one of these ℓ components, we have an additional eigenvalue of $\boldsymbol{M}$ which is bounded away from -1. In all, using Lemma 21, we get $\nu_{k+\ell}(\boldsymbol{M}^{2t}) \leq (1 - O(\varphi^2)^{2t}$ as desired.

Finally, we turn to item 2 in Remark 15. Towards relating eigenvalues of $\boldsymbol{W}$ and $n/s \cdot \boldsymbol{W}_S$ using Matrix Bernstein, we need to control the Euclidean length of columns of $\boldsymbol{M}^{2t}$. Thus, we want to understand collision statistics of random walks performed from all start vertices in G. We do this by following techniques used in [4, 7] which encounters a similar situation. The main goal in [4] was to test k-clusterability and lazy walks were fine for this objective. The main idea there was to show that the eigenvectors of the random walk matrix corresponding to

eigenvalues close to 1 (that is, eigenvectors which reveal clusterability information) are mostly uniform, in absolute value, over the corresponding cluster. [7] formalize this by proving an ℓ_∞ norm bound on such eigenvectors which is later leveraged towards understanding the collision statistics of random walk behaviors from an arbitrary pair a, b of vertices. Oversimplifying a little, this allows them to approximate powers of random walk matrices which can then be used to test for an eigengap which in turn allows us to test clusterability. However, bounding this Euclidean length in our situation requires a more nuanced adaptation of techniques from [4] since our walks are non-lazy and we need to track eigenvectors of the random walk matrix with corresponding eigenvalue close to -1.

Indeed, we show a similar statement for such eigenvectors. This is done by proving ℓ_∞ norm bounds on these eigenvectors which we later use to approximate appropriate powers of random walk matrices. Again, using the same oversimplification as above, this allows us to approximate powers of random walk matrices which we then use to distinguish clusterable graphs with large max-cut value from clusterable graphs with small max-cut value. More precisely, what we show that the eigengaps between the original clusterable graphs (one with large max-cut value and the other one with a small max-cut value) are preserved under sampling. And this finishes the high level description of our approach. As a final aside, there are a few additional technical challenges/interesting features of this work which we enumerate below.

- **Challenges in Bounding Euclidean Length of Random Walk Vectors:** Recall that in [4], the goal was to test k-clusterability. To this end, [4] exploits that for a k-clusterable graph, there is a large gap between the k-th largest eigenvalue (which is at least $1 - 2\varepsilon$) and $(k+1)$-st largest eigenvalue (at most $1 - \varphi^2/2$)) of the random walk matrix, $\boldsymbol{M}$. On the other hand, in graphs which are far from being k-clusterable, the $(k+1)$-st eigenvalue of the random walk matrix is also reasonably large and this can be used a reliable estimator to distinguish between the two cases. However, in our setup, there is no such sharp threshold after which we necessarily witness any sharp drop between two successive eigenvalues of $\boldsymbol{M}$. And therefore, this non-existence of an eigengap between successive eigenvalues remains a problem with $\boldsymbol{M}^{2t}$ as well. Indeed, if $\mathsf{Max\text{-}Cut(G)}$ is large, we can have more than ℓ clusters with relatively large induced max-cut-value (say with value at least $(1 - 1000\varepsilon)$) – and each of these clusters implies a yet another large eigenvalue of $\boldsymbol{M}$. To allay this, we consider all eigenvectors with eigenvalue at least $1 - \delta\varphi^2$ and we leverage our ℓ_∞ bounds on the eigenvectors to upperbound the contribution to $\|\boldsymbol{M}^t \mathbf{1}_x\|_2^2$ from such eigenvectors. For other eigenvectors, the contribution to the walk length can be handled by choosing walk length suitably (which depends inversely on the δ value we choose). This also explains why our analysis carries the parameter δ around.
- **Showing an Eigengap in Presence of Crossedges:** Remark 15 emphasizes that we have an eigenvalue gap between the two cases, 1(a) and 1(b). However, recall that we wanted to consider instances with outer conductance ε and the instance just described had outer conductance 0. Two problems emerge when we consider instances with large outer conductance. All of the argument so far assumes there are no cross edges running between these k components. We show when an ε fraction of cross edges between various components are added, the $(k+\ell)$-th eigenvalue of $\boldsymbol{M}^{2t}$ *still remains* a reliable indicator for the max-cut-value. This does not follow immediately from Frobenius norm bounds on the Laplacian corresponding to the cut edges.
- **Necessity of Non-Lazy Walks:** An essential feature of our algorithm is that it crucially involves performing non-lazy random walks. To the best of our knowledge, there is no other work in sublinear algorithms where analyzing non-lazy walks is tied with the algorithmic guarantees in such a fundamental way. Classic results in property testing

Algorithm 1 TestMaxCut$(G, k, \varphi, \varepsilon, d)$ ▷ Need: $\varepsilon/\varphi^2 \leq \delta = \frac{1}{10^5}$, a constant $\chi \gg 1$.
▷ Set constants $a = \frac{2000 \cdot \chi \cdot d^4}{\delta}$, $b = \frac{4000 \cdot \chi^2 d^8}{\delta^2}$.

$\ell = \lceil 2k/3 \rceil$
$\xi = n^{-a \cdot \varepsilon/\varphi^2}$.
$s = 10^{20} k^4 d^6 \cdot n^{80\delta + b \cdot \varepsilon^2/\varphi^4}$
$t = 10/\delta \cdot 1/\varphi^2 \cdot \chi d^3 \log n$.
Sample s vertices from V uniformly at random. Let S be the multiset of sampled vertices.
Compute $\boldsymbol{Z} = (n/s) \left(\boldsymbol{D}^{-1/2}\boldsymbol{M}^t\boldsymbol{S}\right)^T \left(\boldsymbol{D}^{-1/2}\boldsymbol{M}^t\boldsymbol{S}\right)$ using the oracle.
$\mu_{thres} = \frac{0.99}{d} \cdot n^{-2000 \cdot \varepsilon \cdot \chi d^4 \delta^{-1} \varphi^{-2}}$.
if $\nu_{k+\ell}(Z) \geq \mu_{thres}$ **then**
  Accept G.
else
  Reject G.

on bounded degree graphs often make the simplification of making the graph regular by adding loops (which again makes any random walks considered lazy). However, we unfortunately cannot use this simplification of adding loops as this again risks shrinking the eigengap our approach hopes to exploit. Thus, for non-regular input graphs, *our analysis can not even assume the random walk Matrix $\boldsymbol{M}$ to be symmetric* (a common assumption which can be made if G could be made regular by adding loops).

4 Algorithm Under the Oracle Assumption

The goal of this section is to present an algorithm for testing Max-Cut(G) under a simplifying assumption. We assume that we have the following oracle at our disposal: the oracle takes a vertex v as input, and returns $\boldsymbol{D}^{-1/2}\boldsymbol{M}^t\mathbf{1}_v$.

5 Proof Under the Oracle Assumption

We state below the main theorem which asserts that the above algorithm is a bonafide distinguisher which reliably tells apart graphs with large max-cut-value from graphs with small max-cut-value. This provides the proof of correctness for the algorithm described in Section 4.

▶ **Theorem 16.** *Let $G = (V, E)$ be a $(k, \varphi, \varepsilon)$-clusterable graph where*
- *The maximum degree of G is at most some constant, d.*
- *$\varepsilon \leq \frac{\delta \cdot \varphi^2}{10^4 \cdot d^4 \cdot \chi}$.*
  Here $\delta = 10^{-5}$ and $\chi > 1$ is sufficiently large.

Then the algorithm TestMaxCut$(G, k, \varphi, \varepsilon, d)$ runs in time $\frac{\chi \cdot d^3 \cdot \log n}{\varphi^2} n^{1/2 + 100\delta + O(\varepsilon/\varphi^2)}$ and with probability at least $2/3$, returns
- *Accept if Max-Cut(G) $\geq 1 - \varepsilon$.*
- *Reject if Max-Cut(G) $\leq 1/2 + \varepsilon$.*

▶ Remark 17. As noted in item 3 of Remark 14, one can show Theorem 16 assuming both $\varepsilon, \gamma \ll \delta\varphi^2$ (where $\delta = 10^{-5}$). It is more easily shown assuming $\gamma \leq \varepsilon$ which is what the theorem above assumes.

Towards proving this, we first prove the following theorem:

▶ **Theorem 18.** *Let $G = (V, E)$ be a $(k, \varphi, \varepsilon)$-clusterable graph where*
- *The maximum degree of G is at most some constant, d.*
- *$\varepsilon \leq \frac{\delta \cdot \varphi^2}{10^4 \cdot d^4 \cdot \chi}$. Here $\delta = 10^{-5}$ and $\chi > 1$ is sufficiently large.*

Then with probability taken over its internal randomness, 1 satisfies the following.
- *If Max-Cut(G) $\geq 1 - \varepsilon$, $\nu_{k+\ell}\left(\left(\boldsymbol{D}^{-1/2}\boldsymbol{M}^t\boldsymbol{S}\right)^T\left(\boldsymbol{D}^{-1/2}\boldsymbol{M}^t\boldsymbol{S}\right)\right) \geq \frac{0.99}{d} \cdot n^{-2000 \cdot \varepsilon \cdot \chi d^4 \delta^{-1} \varphi^{-2}}$ with probability at least $2/3$,*
- *If Max-Cut(G) $\leq 1/2 + \varepsilon$, $\nu_{k+\ell}\left(\left(\boldsymbol{D}^{-1/2}\boldsymbol{M}^t\boldsymbol{S}\right)^T\left(\boldsymbol{D}^{-1/2}\boldsymbol{M}^t\boldsymbol{S}\right)\right) \leq n^{-100}$,*

The proof of Theorem 16 is immediate by Theorem 18.

Proof of Theorem 16. As ε is upper bounded by $10^{-4}\delta\varphi^2\chi^{-1}d^{-4}$, in the YES case our estimator is lower bounded by $0.99d^{-1} \cdot n^{-2000 \cdot \varepsilon \cdot \chi d^4 \delta^{-1} \varphi^{-2}}$ with probability at least $2/3$. While in the NO case, our estimator is upper bounded by n^{-100} with probability 1. ◀

We begin by stating the following proposition:

▶ **Proposition 19.** *Let $G = (V, E)$ be a bounded degree $(k, \varphi, \varepsilon)$-clusterable graph with ε at most $10^{-4}\delta\varphi^2\chi^{-1}d^{-4}$ where $\delta = 10^{-5}$ and d is the degree bound. Then,*
1. *If Max-Cut(G) $\geq 1 - \varepsilon$, then at least $\lceil 2k/3 \rceil$ of the clusters have induced Max-Cut value at least $(1 - 10\varepsilon d)$.*
2. *If Max-Cut(G) $\leq 1/2 + \varepsilon$, then at least $\lceil 2k/3 \rceil$ of the clusters have induced Max-Cut value at most $(1/2 + 10\varepsilon d)$.*

N.B. In the following, we will denote $\lceil 2k/3 \rceil$ by ℓ for simplicity.

A simple markov argument shows that in the case with large max-cut value, most of the clusters are nearly bipartite (far from bipartite in the case with small max-cut value resp.).

5.1 Eigengaps in the Spectrum of Random Walk Matrix

For simplicity of the reader, we collect all the parameters we use throughout the paper.
- $\delta = 10^{-5}$.
- $\chi > 1$. a sufficiently large constant.
- A degree bound, d.
- $k \in \mathbb{N}$, the number of clusters in our $(k, \varphi, \varepsilon)$ clusterable graph.
- $\ell = \lceil 2k/3 \rceil$.
- A bound on ε, namely $\varepsilon \leq \delta\varphi^2/(10^4 d^4 \chi)$.

We state the main result of this section below. The proof is given in the appendix. In this section, we will prove one key lemma (Lemma 21) which is crucially used in proving the theorem below.

▶ **Theorem 20.** *Let G be a bounded degree graph that admits a $(k, \varphi, \varepsilon)$-clustering such that $\varepsilon \leq \delta \cdot \varphi^2/\left(10^4 \chi d^4\right)$ where d is the degree bound (and $\delta = 10^{-5}$). Then,*
1. *If Max-Cut(G) $\geq 1 - \varepsilon$, then $\nu_{k+\ell}\left(\left(\boldsymbol{D}^{-1/2}\boldsymbol{M}^t\right)^T\left(\boldsymbol{D}^{-1/2}\boldsymbol{M}^t\right)\right) \geq (1 - 100\varepsilon d)^{2t}/d$,*
2. *If Max-Cut(G) $\leq 1/2 + \varepsilon$, then $\nu_{k+\ell}\left(\left(\boldsymbol{D}^{-1/2}\boldsymbol{M}^t\right)^T\left(\boldsymbol{D}^{-1/2}\boldsymbol{M}^t\right)\right) \leq \left(1 - \varphi^2/\left(100\chi d^3\right)\right)^{2t}$,*

where t is any even number.

We begin by proving the key technical lemma required in the proof of Theorem 20.

▶ **Lemma 21** (Eigengap Transportation). *Let $G = (V, E)$ be a bounded degree graph, and $0 \leq \lambda_1 \leq \lambda_2 \leq \ldots \lambda_n \leq 2$ denote the spectrum of $\boldsymbol{L}_G$. Then $\lambda_{n-1} \leq 2 - \frac{\lambda_2}{\chi \cdot d^2}$, where d is the degree bound and χ is an absolute constant.*

Let us do a little setup before we prove Lemma 21. Suppose $\boldsymbol{v}_{n-1}, \boldsymbol{v}_n$ denote the last two eigenvectors of $\bar{\boldsymbol{L}} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$. Suppose for a sufficiently small $\gamma > 0$, we have for every $\boldsymbol{x} \in span(\boldsymbol{v}_{n-1}, \boldsymbol{v}_n)$, $R(\boldsymbol{x}) \geq 2 - \gamma$. We will show that in this case we also have $\lambda_2 \leq O(\gamma d^2)$ where d is the maximum degree in the graph G. Before we prove this result, we first develop some intuition which will help with the formal proof which is presented in Section Subsubsection 5.1.2.

5.1.1 Intuition for Proving Lemma 21

Towards getting some intuition, it will be helpful to assume that the graph is d-regular. Denote the eigenvectors corresponding to λ_{n-1} (resp λ_n) as $\boldsymbol{v}_{n-1}$ (resp $\boldsymbol{v}_n$). Recall that in this (d-regular) case the eigenvector corresponding to λ_1 is given $\boldsymbol{v}_1 = \mathbf{1}/\sqrt{n}$. As mentioned above, we would like to produce a vector $\boldsymbol{x} \perp \boldsymbol{v}_1$ with small Rayleigh Quotient. Consider a d-regular graph with two disjoint bipartite components each on n vertices – denoted $G_1 = (L_1, R_1, E_1)$ and $G_2 = (L_2, R_2, E_2)$. The eigenvectors $\boldsymbol{v}_{n-1}, \boldsymbol{v}_n$ satisfy:

$$\boldsymbol{v}_{n-1}(u) = \begin{cases} +1/\sqrt{n} \text{ if } u \in L_2 \\ -1/\sqrt{n} \text{ if } u \in R_2 \\ 0 \text{ Otherwise.} \end{cases} \quad \text{and} \quad \boldsymbol{v}_n(u) = \begin{cases} +1/\sqrt{n} \text{ if } u \in L_1 \\ -1/\sqrt{n} \text{ if } u \in R_1 \\ 0 \text{ Otherwise.} \end{cases} \tag{1}$$

Now consider the vector $\boldsymbol{x}$ (resp $\boldsymbol{y}$) obtained by reversing the signs of all entries in the vector $\boldsymbol{v}_n$ (resp $\boldsymbol{v}_{n-1}$). Thus, the vector $\boldsymbol{x}$ equals a copy of the all-ones vector over G_1 and $\boldsymbol{y}$ equals a copy of all-ones vector over G_2 where $\boldsymbol{x}$ and $\boldsymbol{y}$ have disjoint supports and thus $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = 0$. This gives a two-dimensional space of vectors with small Rayleigh Quotient which in turn means $\lambda_2 = 0$. While taking the absolute values gives a vector with small Rayleigh Quotient, in general, we cannot expect this to produce vectors with disjoint supports. Suppose we only have the vector $\boldsymbol{y}$ we obtained above by taking the absolute values of every entry in $\boldsymbol{v}_{n-1}$. Suppose we want to use only this vector towards bounding λ_2. The difficulty is this vector is not orthogonal to $\boldsymbol{v}_1$ as all coordinates in both of these vectors are all positive. To fix this, we subtract off a multiple of the projection of the $\boldsymbol{y}$ along the all-ones vector to obtain a vector $\boldsymbol{z} \perp \mathbf{1}$. We would like to bound the Rayleigh Quotient of $\boldsymbol{z}$. Since all the coordinates in $\boldsymbol{z}$ are a shift of corresponding coordinates in $\boldsymbol{y}$ (by the same additive amount) the numerator of the corresponding Rayleigh Quotients of the two vectors are equal. Towards bounding the Rayleigh Quotient, the main idea is to lower bound the length of $\boldsymbol{z}$ after this shift. This is precisely what we achieve in Lemma 27. Details follow.

5.1.2 Proof of Lemma 21

The high-level idea in the argument is to exhibit a two-dimensional subspace all vectors in which have a small Rayleigh Quotient with respect to the Laplacian. We already know $\boldsymbol{D}^{1/2}\mathbf{1}$ is one such vector. So, it suffices to produce a vector $\boldsymbol{t} \perp \boldsymbol{D}^{1/2}\mathbf{1}$ which has a small Rayleigh Quotient. At a high-level, our proof uses the following strategy. If a suitable non-linear transform applied to vectors $\boldsymbol{v}_{n-1}$ or $\boldsymbol{v}_n$ does not give us the desired vector $\boldsymbol{t}$, then that same transform applied to an equal weight linear combination of $\boldsymbol{D}^{-1/2}\boldsymbol{v}_{n-1}$ and $\boldsymbol{D}^{-1/2}\boldsymbol{v}_n$ gives us the desired vector $\boldsymbol{t}$.

Let $R(\boldsymbol{x}) = R_{\bar{\boldsymbol{L}}}(\boldsymbol{x})$. Note that the map $\boldsymbol{x} \to \boldsymbol{D}^{1/2}\boldsymbol{x}$ is a bijection and as noted in [15], this means

$$\lambda_{n-1} = \max_{\substack{S \subseteq \mathbb{R}^n \\ S \text{ 2-dimensional}}} \min_{\boldsymbol{x} \in S \setminus 0} R_{\bar{\boldsymbol{L}}}(\boldsymbol{x}) \tag{2}$$

$$= \max_{\substack{S \subseteq \mathbb{R}^n \\ S \text{ 2-dimensional}}} \min_{\boldsymbol{x} \in S \setminus 0} R_{\bar{\boldsymbol{L}}}(\boldsymbol{D}^{1/2}\boldsymbol{x}) \tag{3}$$

$$= \frac{\sum_{(u,v)\in E}(\boldsymbol{x}_u - \boldsymbol{x}_v)^2}{\sum d_v \boldsymbol{x}_v^2}. \tag{4}$$

The following observation is immediate.

▶ **Observation 22.** *Suppose $\lambda_{n-i+1}(\bar{\boldsymbol{L}}) \geq 2 - \gamma$. Let $S = span(\boldsymbol{v}_{n-i+1}, \boldsymbol{v}_{n-i+2}, \ldots, \boldsymbol{v}_n)$. Consider a i-dimensional subspace $S' = span(\boldsymbol{D}^{-1/2}\boldsymbol{v}_{n-i+1}, \boldsymbol{D}^{-1/2}\boldsymbol{v}_{n-i+2}, \ldots \boldsymbol{D}^{-1/2}\boldsymbol{v}_n) \subseteq \mathbb{R}^n$. Then, for all non-zero vectors $\boldsymbol{x} \in S'$, we have $R(\boldsymbol{D}^{1/2}\boldsymbol{x}) \geq 2 - \gamma$.*

▷ Claim 23. Let $\boldsymbol{x} \in span(\boldsymbol{D}^{-1/2}\boldsymbol{v}_{n-1}, \boldsymbol{D}^{-1/2}\boldsymbol{v}_n)$. Now consider the vector $\boldsymbol{x}' = |\boldsymbol{x}|$ obtained by letting $\boldsymbol{x}'(u) = |\boldsymbol{x}(u)|$ for each $u \in V$. Then, $R(\boldsymbol{D}^{1/2}\boldsymbol{x}') \leq \gamma$.

Proof. Since $\boldsymbol{x} \in S \stackrel{\text{def}}{=} span(\boldsymbol{D}^{-1/2}\boldsymbol{v}_{n-1}, \boldsymbol{D}^{-1/2}\boldsymbol{v}_n)$, by Observation 22, it holds that $R(\boldsymbol{D}^{1/2}\boldsymbol{x}) \geq 2-\gamma$. This means $\sum_{(u,v)\in E}(\boldsymbol{x}(u)+\boldsymbol{x}(v))^2 \leq \gamma \sum d_v \boldsymbol{x}_v^2$. Note that $\|\boldsymbol{D}^{1/2}\boldsymbol{x}'\|_2 = \|\boldsymbol{D}^{1/2}\boldsymbol{x}\|_2$ as the two vectors have same absolute value in each coordinate. We will show that $R(\boldsymbol{D}^{1/2}\boldsymbol{x}') = \frac{\sum_{(u,v)\in E}\left(\boldsymbol{x}'(u)-\boldsymbol{x}'(v)\right)^2}{\sum d_v \boldsymbol{x}'(v)^2} \leq \gamma$ which will settle the claim. To do this, pick an edge $(u, v) \in E$ and note that we have the following cases.

1. **Case 1:** $\boldsymbol{x}'(u) = \boldsymbol{x}(u), \boldsymbol{x}'(v) = \boldsymbol{x}(v)$. In this case, we note that $\left(\boldsymbol{x}'(u) - \boldsymbol{x}'(v)\right)^2 \leq \left(\boldsymbol{x}(u) + \boldsymbol{x}(v)\right)^2$.
2. **Case 2:** $\boldsymbol{x}'(u) = -\boldsymbol{x}(u), \boldsymbol{x}'(v) = -\boldsymbol{x}(v)$.
   In this case as well, it holds that $\left(\boldsymbol{x}'(u) - \boldsymbol{x}'(v)\right)^2 \leq \left(\boldsymbol{x}(u) + \boldsymbol{x}(v)\right)^2$.
3. **Case 3 and 4:** $\boldsymbol{x}'(u) = -\boldsymbol{x}(u), \boldsymbol{x}'(v) = \boldsymbol{x}(v)$ and vice versa. In this case it holds that $\left(\boldsymbol{x}'(u) - \boldsymbol{x}'(v)\right)^2 = \left(\boldsymbol{x}(u) + \boldsymbol{x}(v)\right)^2$.

Thus, it follows that in all, we have $R(\boldsymbol{D}^{1/2}\boldsymbol{x}') = \frac{\sum_{(u,v)\in E}\left(\boldsymbol{x}'(u)-\boldsymbol{x}'(v)\right)^2}{\sum d_v \boldsymbol{x}'(v)^2} \leq \gamma$ which settles the claim. ◁

Thus, given any vector $\boldsymbol{x} \in S \stackrel{\text{def}}{=} span(\boldsymbol{D}^{-1/2}\boldsymbol{v}_{n-1}, \boldsymbol{D}^{-1/2}\boldsymbol{v}_n)$ we can produce a vector $\boldsymbol{x}'$ for which $\boldsymbol{D}^{1/2}\boldsymbol{x}'$ has small Rayleigh Quotient. However, this vector is not orthogonal to the trivial eigenvector $\boldsymbol{D}^{1/2}\mathbf{1}$ of $\boldsymbol{L}$. To fix this, we obtain a vector $\boldsymbol{t}$ in two steps. As a first step, consider the following vector obtained by shifting $\boldsymbol{x}'$ around which is orthogonal to the all ones vector, $\mathbf{1}$:

$$\boldsymbol{s} = \boldsymbol{x}' - \|\boldsymbol{x}\|_1 \cdot \frac{\mathbf{1}}{n}.$$

To obtain a vector orthogonal to $\boldsymbol{D}^{1/2}\mathbf{1}$, consider the vector $\boldsymbol{t} = \boldsymbol{D}^{-1}\boldsymbol{s}$. Observation 26 shows that this vector is orthogonal to $\boldsymbol{D}^{1/2}\mathbf{1}$. One notes that $\boldsymbol{t}$ does not necessarily have small length and this is an obstacle to upperbound $R(\boldsymbol{D}^{1/2}\boldsymbol{t})$. To handle this, we make the following definition.

▶ **Definition 24.** *Let $\alpha > 0$ be a sufficiently small constant. Take a unit vector $\boldsymbol{x} \in \mathbb{R}^n$. Consider the vector $\boldsymbol{s}$ obtained by taking absolute values in each coordinate and then shifting it to obtain a vector orthogonal to all 1's vector. That is,*

$$\boldsymbol{s}(i) = |\boldsymbol{x}(i)| - \frac{\|\boldsymbol{x}\|_1}{n}.$$

Let $\boldsymbol{t} = \boldsymbol{D}^{-1}\boldsymbol{s}$. The vector $\boldsymbol{x}$ is called (α, d)-bad if $\|\boldsymbol{t}\|_2 < \alpha/d$. If $\|\boldsymbol{t}\|_2 \geq \alpha/d$, then the vector $\boldsymbol{x}$ is not (α, d)-bad and is called (α, d)-good. If the parameter d is clear from context, we will call such these vectors α-good or α-bad.

We make the following observations about (α, d)-good vectors.

▶ **Observation 25.** *Let $\boldsymbol{x} \in \mathbb{R}^n$ be a unit vector. Obtain the vector $\boldsymbol{s} = |\boldsymbol{x}| - \|\boldsymbol{x}\|_1 \cdot \frac{\mathbf{1}}{n}$ and the vector $\boldsymbol{t} = \boldsymbol{D}^{-1}\boldsymbol{s}$. If $\|\boldsymbol{t}\| \geq \beta$, then $\|\boldsymbol{s}\|_2 \geq \beta$. Also, if $\|\boldsymbol{s}\| \geq \alpha$, then $\boldsymbol{t}$ is α-good.*

Proof. Note that $\|\boldsymbol{t}\|_2^2 = \sum \boldsymbol{s}_i^2/d_i \leq \sum \boldsymbol{s}_i^2 = \|\boldsymbol{s}\|_2^2$. Thus, if $\|\boldsymbol{t}\|_2 \geq \beta$, $\|\boldsymbol{s}\|_2 \geq \|\boldsymbol{t}\|_2 \geq \beta$. In the other direction, we are told $\|\boldsymbol{s}\|_2^2 = \sum \boldsymbol{s}_i^2 \geq \alpha^2$. Note that $\|\boldsymbol{t}\|_2^2 = \sum \boldsymbol{s}_i^2/d_i^2 \geq \|\boldsymbol{s}\|_2^2/d^2$ and the result follows. ◀

▶ **Observation 26.** *Let $\boldsymbol{x} \in \mathbb{R}^n$ be a unit vector. Let $\boldsymbol{t}$ be a vector obtained as above. We have $\boldsymbol{D}^{1/2}\boldsymbol{t} \perp \boldsymbol{D}^{1/2}\mathbf{1}$.*

Proof. Note

$$\langle \boldsymbol{D}^{1/2}\boldsymbol{t}, \boldsymbol{D}^{1/2}\mathbf{1}\rangle = \langle \boldsymbol{t}, \boldsymbol{D}\mathbf{1}\rangle = \sum \boldsymbol{t}_i d_i = \sum \boldsymbol{s}_i = 0.$$ ◀

In the rest of this section, we will prove the following lemma.

▶ **Lemma 27.** *Let $\alpha > 0$ be a sufficiently small constant. Then there exists a vector $\boldsymbol{x} \in span(\boldsymbol{D}^{-1/2}\boldsymbol{v}_{n-1}, \boldsymbol{D}^{-1/2}\boldsymbol{v}_n)$ which is α-good.*

With this lemma in hand, Lemma 21 follows as an immediate corollary.

Proof of Lemma 21. By Lemma 27, there exists a vector $\boldsymbol{x} \in span(\boldsymbol{D}^{-1/2}\boldsymbol{v}_{n-1}, \boldsymbol{D}^{-1/2}\boldsymbol{v}_n)$ which is α-good. As before, define the vectors $\boldsymbol{s}$ and $\boldsymbol{t}$. Recall from Observation 26 that $\boldsymbol{D}^{1/2}\boldsymbol{t} \perp \boldsymbol{D}^{1/2}\mathbf{1}$. Towards showing that $\lambda_2 \leq O(\gamma d^2)$, it suffices to show that $R(\boldsymbol{D}^{1/2}\boldsymbol{t}) \leq O(\gamma d^2)$. First, let us note that $\boldsymbol{x}$ being α-good, we have $\|\boldsymbol{t}\|_2 = \left\|\boldsymbol{D}^{-1}\boldsymbol{s}\right\|_2 \geq \alpha/d$. Letting $\boldsymbol{x}' = |\boldsymbol{x}|$, by Claim 23, we know $\sum_{(u,v)\in E} \left(\boldsymbol{x}'(u) - \boldsymbol{x}'(v)\right)^2 \leq \gamma \sum d_v \boldsymbol{x}'(v)^2$. And since $\boldsymbol{s}$ is obtained by shifting each coordinate in $\boldsymbol{x}'$ by the same amount, it follows that

$$\sum_{(u,v)\in E} \left(\boldsymbol{s}(u) - \boldsymbol{s}(v)\right)^2 = \sum_{(u,v)\in E} \left(\boldsymbol{x}'(u) - \boldsymbol{x}'(v)\right)^2.$$

Next, write

$$R(\boldsymbol{D}^{1/2}\boldsymbol{t}) = \frac{\sum_{(u,v)\in E}(\boldsymbol{t}_u - \boldsymbol{t}_v)^2}{\sum d_v \boldsymbol{t}_v^2}.$$

We observe that for each edge $(u, v) \in E$, $(\boldsymbol{t}_u - \boldsymbol{t}_v)^2 \leq (\boldsymbol{s}_u - \boldsymbol{s}_v)^2$. Finally, note $\sum d_v \boldsymbol{t}_v^2 \geq \sum \boldsymbol{t}_v^2 = \|\boldsymbol{t}\|_2^2 \geq \alpha^2/d^2$. Thus, it follows that $R(\boldsymbol{D}^{1/2}\boldsymbol{t}) \leq \gamma d^2/\alpha^2$. This means that $\lambda_2 \leq R(\boldsymbol{D}^{1/2}\boldsymbol{t}) \leq \gamma \cdot d^2/\alpha^2$. ◀

Now, in the rest of this document, we will prove Lemma 27. The following claim will be useful.

▷ Claim 28. Suppose $\boldsymbol{x}$ is an α-bad vector of unit length (in l_2). Let

$$\boldsymbol{s} = |\boldsymbol{x}| - \|\boldsymbol{x}\|_1 \cdot \frac{\mathbf{1}}{n}$$

(where $|\boldsymbol{x}|$ is a vector with $|\boldsymbol{x}|(u) = |\boldsymbol{x}(u)| \forall u \in V$.) Then $\|\boldsymbol{s}\|_1 \geq \left(1 - \alpha^2/4\right)\sqrt{n}$.

Proof. By the definition of bad vectors and Observation 25, we have $\|\boldsymbol{s}\|_2 \leq \alpha$. On expanding out,

$$\|\boldsymbol{s}\|_2^2 = \sum_{i \in V} \left(\boldsymbol{x}(i)^2 + \frac{\|\boldsymbol{x}\|_1^2}{n^2} - \frac{2|\boldsymbol{x}(i)| \cdot \|\boldsymbol{x}\|_1}{n} \right) = \left(\|\boldsymbol{x}\|_2^2 - \frac{\|\boldsymbol{x}\|_1^2}{n} \right) = \left(1 - \frac{\|\boldsymbol{x}\|_1^2}{n} \right) \leq \alpha^2.$$

Rearranging, this gives $\|\boldsymbol{x}\|_1 \geq \sqrt{n} \cdot \sqrt{1 - \alpha^2}$. For sufficiently small α, and taking Taylor expansion, this gives

$$\|\boldsymbol{x}\|_1 \geq \left(1 - \frac{\alpha^2}{4}\right) \cdot \sqrt{n} \tag{5}$$

◁

Now suppose $\boldsymbol{x}$ is indeed an α-bad vector. Since, $\boldsymbol{x}$ is a unit vector with $\|\boldsymbol{x}\|_1$ pretty close to $\sqrt{n}$, it follows that the absolute value of $\boldsymbol{x}$ in each coordinate is almost $1/\sqrt{n}$. This is shown below.

▷ Claim 29. Suppose $\boldsymbol{x}$ is an α-bad vector with $\|\boldsymbol{x}\|_2 = 1$. Let $\beta > 0$ be sufficiently small. Then for at least $\left(1 - \alpha^2/2\beta^2\right) \cdot n$ coordinates in $i \in [n]$, it holds that

$$\frac{1-\beta}{\sqrt{n}} \leq |\boldsymbol{x}(i)| \leq \frac{1+\beta}{\sqrt{n}}.$$

Proof. From Claim 28, it follows that $\|\boldsymbol{x}\|_1 \geq (1 - \alpha^2/4)\sqrt{n}$. This means

$$\begin{aligned}
\frac{\|\boldsymbol{x}\|_1}{\sqrt{n}} = \sum_{i \in [n]} |\boldsymbol{x}(i)| \cdot \frac{1}{\sqrt{n}} &\geq (1 - \alpha^2/4) \\
\implies 1/2 + 1/2 - \sum_{i \in [n]} |\boldsymbol{x}(i)|/\sqrt{n} &\leq \alpha^2/4 \qquad (6) \\
\implies \frac{1}{2} \cdot \sum_{i \in [n]} |\boldsymbol{x}(i)^2| + \frac{1}{2} \cdot \sum_{i \in [n]} \left(\frac{1}{\sqrt{n}}\right)^2 - \sum_{i \in [n]} |\boldsymbol{x}(i)|/\sqrt{n} &\leq \alpha^2/4 \qquad (7) \\
\implies \sum_{i \in [n]} \left(|\boldsymbol{x}(i)| - \frac{1}{\sqrt{n}} \right)^2 &\leq \frac{\alpha^2}{2} \qquad (8)
\end{aligned}$$

Now let

$$B_{\boldsymbol{x}} = \{i \in [n] : |\boldsymbol{x}(i)| \geq \frac{1+\beta}{\sqrt{n}}\} \cup \{i \in [n] : |\boldsymbol{x}(i)| \leq \frac{1-\beta}{\sqrt{n}}\}.$$

Note that for each such $i \in B_{\boldsymbol{x}}$, we have that $(|\boldsymbol{x}(i)| - 1/\sqrt{n})^2 \geq \beta^2/n$. Together with (8), this means that $|B_{\boldsymbol{x}}| \leq \frac{1}{2} \cdot \left(\frac{\alpha}{\beta}\right)^2 n$ which settles the claim. ◁

Thus, Claim 28 means that if a vector $\boldsymbol{x}$ is α-bad, it would have a pretty large l_1 mass which as shown in Claim 29 means that $\boldsymbol{x}$ takes on values close to $\pm\frac{1}{\sqrt{n}}$ almost everywhere in the support. This gives us all the ammunition we need to prove Lemma 27. We will proceed by contradiction. That is, we will produce a vector $\boldsymbol{x} \in S \stackrel{\text{def}}{=} span(\boldsymbol{D}^{-1/2}\boldsymbol{v}_{n-1}, \boldsymbol{D}^{-1/2}\boldsymbol{v}_n)$ with small l_1 norm. And this means this vector is α-good. Details follow.

Proof of Lemma 27. For ease of indexing, let $\boldsymbol{z}_1 = \boldsymbol{D}^{-1/2}\boldsymbol{v}_n, \boldsymbol{z}_2 = \boldsymbol{D}^{-1/2}\boldsymbol{v}_{n-1}$. If either of $\boldsymbol{z}_1, \boldsymbol{z}_2$ is α-good, we choose that vector and the proof is finished. So, suppose both $\boldsymbol{z}_1, \boldsymbol{z}_2$ are α-bad. In this case, let S denote the $span(\boldsymbol{z}_1, \boldsymbol{z}_2) = span(\boldsymbol{D}^{-1/2}\boldsymbol{v}_n, \boldsymbol{D}^{-1/2}\boldsymbol{v}_{n-1})$. We normalize all non-zero vectors in S to have length 1. We will show that the unit vector $\boldsymbol{x} = \frac{1}{\sqrt{2}}(\boldsymbol{z}_1 + \boldsymbol{z}_2) \in S$ is in fact α-good. By way of contradiction, suppose $\boldsymbol{x}$ is α-bad and thus by Claim 28 $\|\boldsymbol{x}\|_1$ is close to $\sqrt{n}$. We will obtain a contradiction to this. Set a parameter $\beta = \sqrt{\alpha}$ and define the set of "bad" coordinates in $\boldsymbol{z}_1$ as

$$B_1 = \left\{ i \in [n] \colon |\boldsymbol{z}_1(i)| \geq \frac{(1+\beta)}{\sqrt{n}} \text{ OR } |\boldsymbol{z}_1(i)| \leq \frac{(1-\beta)}{\sqrt{n}} \right\}.$$

Similarly, define B_2 as the set of bad coordinates in $\boldsymbol{z}_2$. By Claim 29, note that $|B_1|, |B_2|$ both have size at most $\alpha/2n$. Let $B = B_1 \cup B_2$ and set $G = [n] \setminus B$. Note that $\boldsymbol{z}_1 \perp \boldsymbol{z}_2$ and thus

$$\langle \boldsymbol{z}_1, \boldsymbol{z}_2 \rangle = 0 = \sum_{i \in [n]} \boldsymbol{z}_1(i)\boldsymbol{z}_2(i) = \sum_{i \in B} \boldsymbol{z}_1(i)\boldsymbol{z}_2(i) + \sum_{i \in G} \boldsymbol{z}_1(i)\boldsymbol{z}_2(i) \tag{9}$$

We will show that the first term above is small in absolute value (and therefore, so is the second term). For notational convenience, denote the restriction of $\boldsymbol{z}_1$ on B as $\boldsymbol{z}_{1,B} \in \mathbb{R}^{|B|}$. Similarly, define $\boldsymbol{z}_{1,G}, \boldsymbol{z}_{2,B}$, and $\boldsymbol{z}_{2,G}$. By Cauchy Schwartz,

$$\left| \sum_{i \in B} \boldsymbol{z}_1(i)\boldsymbol{z}_2(i) \right| \leq \|\boldsymbol{z}_{1,B}\|_2 \|\boldsymbol{z}_{2,B}\|_2 \tag{10}$$

We now upperbound the right hand side by upperbounding each of the two norms above. We do this, for instance for $\boldsymbol{z}_{1,B}$ by noting $1 = \|\boldsymbol{z}_1\|_2^2 = \|\boldsymbol{z}_{1,G}\|_2^2 + \|\boldsymbol{z}_{1,B}\|_2^2$ and noting that for each $i \in G$, $\boldsymbol{z}_1(i)^2 \geq \frac{(1-\beta)^2}{n}$. This way, we get

$$\|\boldsymbol{z}_{1,B}\|_2^2 \leq 1 - \frac{(1-\beta)^2}{n} \cdot |G| \leq 1 - \frac{(1-\sqrt{\alpha})^2}{n} \cdot (1-\alpha)n \leq 1 - (1-\alpha) \cdot (1 - 2\sqrt{\alpha}) \leq 3\sqrt{\alpha}.$$

The second inequality uses that $\beta = \sqrt{\alpha}$. Similarly, $\|\boldsymbol{z}_{2,B}\|_2^2 \leq 3\sqrt{\alpha}$ as well. This means $\langle \boldsymbol{z}_{1,B}, \boldsymbol{z}_{2,B} \rangle \leq 3\sqrt{\alpha}$. Thus, the inner product of $\boldsymbol{z}_{1,B}$ and $\boldsymbol{z}_{2,B}$ is indeed small in magnitude and the same holds for the inner product of $\boldsymbol{z}_{1,G}$ and $\boldsymbol{z}_{2,G}$ which means the restrictions to the good parts of $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ are nearly orthogonal. We will now show that $\|\boldsymbol{x}\|$ is small and thus, by Claim 28, $\boldsymbol{x}$ cannot be α-bad. To this end, write

$$\|\boldsymbol{x}\|_1 = \frac{1}{\sqrt{2}} \cdot \sum_{i \in G} |\boldsymbol{z}_1(i) + \boldsymbol{z}_2(i)| + \sum_{i \in B} |\boldsymbol{z}_1(i) + \boldsymbol{z}_2(i)|.$$

Let $P = \{i \in G \colon \boldsymbol{z}_1(i), \boldsymbol{z}_2(i) \text{ have the same sign.}\}$ and let $N = G \setminus P$. We have

$$\begin{aligned} \|\boldsymbol{x}\|_1 =& \frac{1}{\sqrt{2}} \cdot \sum_{i \in P} |\boldsymbol{z}_1(i) + \boldsymbol{z}_2(i)| + \sum_{i \in N} |\boldsymbol{z}_1(i) + \boldsymbol{z}_2(i)| + \sum_{i \in B} |\boldsymbol{z}_1(i) + \boldsymbol{z}_2(i)| \\ \leq& \frac{1}{\sqrt{2}} \left(\frac{2 + 2\beta}{\sqrt{n}} \right) \cdot |P| + \frac{1}{\sqrt{2}} \cdot 2\beta \cdot |N| + \sum_{i \in B} |\boldsymbol{z}_1(i) + \boldsymbol{z}_2(i)| \end{aligned}$$

We now bound the RHS above. For the last term, note that by triangle inequality and Cauchy Schwatrz,

$$\sum_{i \in B} |\boldsymbol{z}_1(i)| + \boldsymbol{z}_2(i)| \leq \|\boldsymbol{z}_{1,B}\|_1 + \|\boldsymbol{z}_{2,B}\|_1 \leq (\|\boldsymbol{z}_{1,B}\|_2 + \|\boldsymbol{z}_{2,B}\|_2) \cdot \sqrt{|B|}$$

which is at most $2\sqrt{3} \cdot \sqrt[4]{\alpha} \cdot \sqrt{n} \leq 4\sqrt[4]{\alpha} \cdot \sqrt{n}$. Finally, we will show that $|P| \approx |N| \approx n/2$ from which it will follow that $\|\boldsymbol{x}\|_1 \approx \left(1/\sqrt{2} + O(\sqrt[4]{\alpha})\right) \cdot \sqrt{n}$ which is multiplicatively bounded away from $\sqrt{n}$ (and thus implies $\boldsymbol{x}$ cannot be α-bad).

To see this, recall $\beta = \sqrt{\alpha}$ and that

$$3\beta \geq \left|\sum_{i \in G} \boldsymbol{z}_1(i)\boldsymbol{z}_2(i)\right| \geq \left|\left|\sum_{i \in P} \boldsymbol{z}_1(i)\boldsymbol{z}_2(i)\right| - \left|\sum_{i \in N} \boldsymbol{z}_1(i)\boldsymbol{z}_2(i)\right|\right|.$$

We will show if $|P| \notin [n/2 \pm 10\beta \cdot n]$ then the absolute inner product restricted to the good coordinates is much larger than 3β which means that both $|P|$ and $|N|$ have size around $n/2$.

Suppose $|P| \geq n/2 + \delta n$. Note

$$\left|\sum_{i \in P} \boldsymbol{z}_1(i)\boldsymbol{z}_2(i)\right| \geq \frac{(1-\beta)^2}{n}|P| \geq \frac{(1+\beta^2-2\beta)}{n} \cdot \left(\frac{n}{2} + \delta n\right) \geq (1-2\beta)\left(\frac{1}{2} + \delta\right).$$

The last term can be expanded as $(1/2) - \beta + \delta - 2\delta\beta$.

Also, note

$$\left|\sum_{i \in N} \boldsymbol{z}_1(i)\boldsymbol{z}_2(i)\right| \leq \frac{(1+\beta)^2}{n} \cdot |N| \leq \frac{(1+\beta^2+2\beta)}{n} \cdot \left(\frac{n}{2} - \delta n\right) \leq (1+3\beta)\left(\frac{1}{2} - \delta\right).$$

The last term can be expanded as $(1/2) + 3\beta/2 - \delta - 3\delta\beta$.

For $\delta = 10\beta$, using

- $|\sum_{i \in G} \boldsymbol{z}_1(i)\boldsymbol{z}_2(i)| \geq |\sum_{i \in P} \boldsymbol{z}_1(i)\boldsymbol{z}_2(i)| - |\sum_{i \in N} \boldsymbol{z}_1(i)\boldsymbol{z}_2(i)|$.
- The above lowerbound on $|\sum_{i \in P} \boldsymbol{z}_1(i)\boldsymbol{z}_2(i)|$, and
- The above upperbound on $|\sum_{i \in N} \boldsymbol{z}_1(i)\boldsymbol{z}_2(i)|$

we conclude $|\sum_{i \in G} \boldsymbol{z}_1(i)\boldsymbol{z}_2(i)| \geq 15\beta$ which is a contradiction. A similar contradiction is reached when $|P| \leq n/2 - \delta n$ (for $\delta = 10\beta$). Thus, overall we have $n/2 - 10\beta n \leq |P|, |N| \leq n/2 + 10\beta n$. Plugging back the upperbounds on $|P|$ and $|N|$ in

$$\|\boldsymbol{x}\|_1 \leq \frac{\sqrt{n}}{\sqrt{2}}\left((2+2\beta) \cdot (1/2 + 10\beta) + 2\beta \cdot (1/2 + 10\beta)\right) + 4\sqrt[4]{\alpha}n \leq \frac{\sqrt{n}}{\sqrt{2}} \cdot (1 + 8\sqrt[4]{\alpha}).$$

The last inequality uses that $\beta = \sqrt{\alpha}$. This confirms that $\boldsymbol{x}$ is α-good as desired. ◀

6 Discussion and Concluding Remarks

As mentioned in the tech-overview, the remainder of the proof is deferred to the arXiv version. In this last section, we want to explain why exploring the better than 1/2 approximability of max-cut over clusterable graphs seems to be an important step. Indeed, as one might already see, the current paper only presents an algorithm achieving better than a factor 1/2 approximation for clusterable graphs. After the seminal work of Goldreich and Ron ([9]), and the recent work of Peng and Yoshida ([13]), the question of obtaining a better than 1/2 approximation algorithms for max-cut in sublinear time got ushered to the frontiers of research. Perhaps the most natural graph class to extend a better than 1/2 approximation guarantee on, is the class of *low threshold rank* graphs as defined in the seminal work of [1].

However, it is important to consider the nuances of the problem: the problem asks –given a graph with small threshold rank, is its max-cut value close to 1 or is it close to 1/2. Approaching this problem in the sublinear time regime appears highly non-trivial and it seems

the current techniques have some limitations. For instance, one might try a Peng-Yoshida style approach which leverages the ℓ_2 distance between distributions induced by walks of odd-lengths and even-lengths. However, as we described in our tech-overview, this technique does not *even* extend to the case of graphs with threshold rank 2 (since clusterable graphs form a sub-class of low-threshold rank graphs). The spectral approach pioneered in the work of Chinplunkar et al. in FOCS 18, (which was refined in our submission), attempts to relate the cut-value of the graph to an appropriate eigenvalue close to -1. This approach fails for low-threshold rank graphs. Since, even for graphs with threshold rank 2, there could be 2 small bipartite components, which would lead to 2 eigenvalues being -1. Thus, it is not immediately clear, if there is an appropriate eigenvalue which is a good indicator of the actual max-cut value.

Taking inspiration from the celebrated work of [6], [5] defined the notion of a (k, φ)-clusterable graphs which has more immediate relevance for the property-testing community. With all of this in mind, we think this is an important stepping stone towards obtaining better than 1/2 approximation algorithms for max-cut on low threshold rank graphs. In particular, getting a handle on the "combinatorics of low-threshold rank graphs" and understanding the structure of small non-expanding sets therein appears quite hard. Clusterable graphs help us leverage a much neater structure and improve our understanding of how random walks might behave in non-expanding graphs, without making the problem of testing max-cut on them trivial.

References

1. Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential algorithms for unique games and related problems. *J. ACM*, 62(5):42:1–42:25, 2015.
2. Boaz Barak, Prasad Raghavendra, and David Steurer. Rounding semidefinite programming hierarchies via global correlation. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 472–481. IEEE Computer Society, 2011.
3. Andrej Bogdanov, Kenji Obata, and Luca Trevisan. A lower bound for testing 3-colorability in bounded-degree graphs. In *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*, pages 93–102. IEEE Computer Society, 2002.
4. Ashish Chiplunkar, Michael Kapralov, Sanjeev Khanna, Aida Mousavifar, and Yuval Peres. Testing graph clusterability: Algorithms and lower bounds. *CoRR*, abs/1808.04807, 2018. `arXiv:1808.04807`.
5. Artur Czumaj, Pan Peng, and Christian Sohler. Testing cluster structure of graphs. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 723–732. ACM, 2015.
6. Shayan Oveis Gharan and Luca Trevisan. Partitioning into expanders. In Chandra Chekuri, editor, *SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1256–1266. SIAM, 2014.
7. Grzegorz Gluch, Michael Kapralov, Silvio Lattanzi, Aida Mousavifar, and Christian Sohler. Spectral clustering oracles in sublinear time, 2021. `arXiv:2101.05549`.
8. Michel X. Goemans and David P. Williamson. .879-approximation algorithms for MAX CUT and MAX 2sat. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, pages 422–431. ACM, 1994.
9. Oded Goldreich and Dana Ron. A sublinear bipartiteness tester for bunded degree graphs. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 289–298. ACM, 1998.

10 Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985. doi:10.1017/CBO9780511810817.

11 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, New York, USA*, pages 85–103. Plenum Press, New York, 1972.

12 Shiping Liu. Multi-way dual cheeger constants and spectral bounds of graphs. *Advances in Mathematics*, 268:306–338, 2015.

13 Pan Peng and Yuichi Yoshida. Sublinear-time algorithms for max cut, max e2lin(q), and unique label cover on expanders, 2022. arXiv:2210.12601.

14 Luca Trevisan. Max cut and the smallest eigenvalue. *SIAM J. Comput.*, 41(6):1769–1786, 2012.

15 Luca Trevisan. Graph theory lecture notes, lecture 02. *Notes*, 2022. URL: https://lucatrevisan.github.io/41000-22/lecture02.pdf.

Algorithms for the Generalized Poset Sorting Problem

Shaofeng H.-C. Jiang
School of Computer Science, Peking University, Beijing, China

Wenqian Wang
School of Electronic, Information and Electrical Engineering, Shanghai Jiao Tong University, China

Yubo Zhang
School of Computer Science, Peking University, Beijing, China

Yuhao Zhang
John Hopcroft Center for Computer Science, Shanghai Jiao Tong University, China

Abstract

We consider a generalized poset sorting problem (GPS), in which we are given a query graph $G = (V, E)$ and an *unknown poset* $\mathcal{P}(V, \prec)$ that is defined on the same vertex set V, and the goal is to make as few queries as possible to edges in G in order to fully recover $\mathcal{P}$, where each query (u, v) returns the relation between u, v, i.e., $u \prec v$, $v \prec u$ or $u \not\sim v$. This generalizes both the poset sorting problem [Faigle et al., SICOMP 88] and the generalized sorting problem [Huang et al., FOCS 11].

We give algorithms with $\tilde{O}(n \operatorname{poly}(k))$ query complexity when G is a complete bipartite graph or G is stochastic under the Erdős-Rényi model, where k is the *width* of the poset, and these generalize [Daskalakis et al., SICOMP 11] which only studies complete graph G. Both results are based on a unified framework that reduces the poset sorting to partitioning the vertices with respect to a given pivot element, which may be of independent interest. Moreover, we also propose novel algorithms to implement this partition oracle. Notably, we suggest a randomized BFS with vertex skipping for the stochastic G, and it yields a nearly-tight bound even for the special case of generalized sorting (for stochastic G) which is comparable to the main result of a recent work [Kuszmaul et al., FOCS 21] but is conceptually different and simplified.

Our study of GPS also leads to a new $\tilde{O}(n^{1-1/(2W)})$ competitive ratio for the so-called weighted generalized sorting problem where W is the number of distinct weights in the query graph. This problem was considered as an open question in [Charikar et al., JCSS 02], and our result makes important progress as it yields the first nontrivial sublinear ratio for general weighted query graphs (for any bounded W). We obtain this via an $\tilde{O}(nk + n^{1.5})$ query complexity algorithm for the case where every edge in G is guaranteed to be comparable in the poset, which generalizes a $\tilde{O}(n^{1.5})$ bound for generalized sorting [Huang et al., FOCS 11].

2012 ACM Subject Classification Theory of computation → Sorting and searching; Theory of computation → Online algorithms

Keywords and phrases sorting, poset sorting, generalized sorting

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.92

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2304.01623` [19]

Funding *Shaofeng H.-C. Jiang*: Research partially supported by a national key R&D program of China No. 2021YFA1000900 and a startup fund from Peking University.
Yuhao Zhang: Yuhao Zhang is supported by the National Natural Science Foundation of China (NSFC) under Grant No. 62102251.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 92; pp. 92:1–92:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

We consider a generalized poset sorting problem and obtain various new algorithmic results. In the *generalized poset sorting* problem (GPS), we are given an undirected *query graph* $G = (V, E)$ and an unknown *poset* $\mathcal{P} = (V, \prec)$. The goal is to fully recover the poset $\mathcal{P}$, that is, to figure out the relation between all $x, y \in V$, through the smallest number of queries to the edges in G. Here, when the algorithm makes a query $(u, v) \in E$, the relation of u and v in the poset, i.e., $u \prec v$, $v \prec u$ or $u \not\sim v$ (which stands for u and v are not comparable), is returned. [1]

When the comparison graph G is a complete graph, GPS reduces to a special case called the *poset sorting* problem which was suggested by [11]. This poset sorting is a fundamental problem since it captures the presence of incomparable elements in a partially ordered set which does not have a linear ordering. For this problem, an algorithm with optimal $\Theta(nk + n \log n)$ query complexity was given in [10] where k is the *width* of the poset (the width of a poset is defined as the size of its largest antichain).[2] However, this $\tilde{O}(nk)$ bound heavily relies on the fact that G is complete, and does not work for our general case where the query graph G can have missing edges (u, v) which forbid the query of the relation between u and v (we shall provide a more detailed technical discussion later).

In fact, the missing edges in the query graph already introduce significant challenges even when the poset $\mathcal{P}$ is a total order (where every two elements are comparable). This special case (general graph G and total order) is called *generalized sorting* whose study was initiated by [18]. The state-of-the-art algorithm for this generalized sorting needs to use $\tilde{O}(\sqrt{mn})$ queries [22] for general graphs G, far from matching the $\Theta(n \log n)$ bound for the classic sorting. On the other hand, a parallel research theme aims to explore whether $\tilde{O}(n)$ query complexity can be obtained for generalized sorting on special graph families. Notably, such algorithms were obtained for complete bipartite graphs [1, 7, 2, 21] and Erdős-Rényi stochastic graphs [18, 22].

Our focus. Thus, a fundamental question is to figure out which families of query graphs G admit algorithms with the optimal $\tilde{O}(nk)$ queries (to match that for the complete graphs [10]) for GPS, where k is the width of the poset. An ideal goal is to achieve this $\tilde{O}(nk)$ bound for general query graphs, but as we mentioned, even for the total order case it is already difficult to improve over $\sqrt{mn}$. Therefore, we instead focus on complete bipartite and Erdős-Rényi stochastic query graphs, which are fundamental cases and were very well studied in the special case of generalized sorting. Moreover, we also study how GPS connects to other settings, especially its implications for variants of generalized sorting. This connection is plausible since a natural way for sorting is to build a partially sorted solution and then solve the remaining sub-problem, and this sub-problem may often be modeled as a poset sorting problem.

Technical challenges. However, designing algorithms for GPS turns out to be nontrivial and requires new approaches. Below, we briefly discuss why the existing techniques from tightly related problems, including poset sorting and generalized sorting problems, cannot be readily applied.

---

[1] To make sure the problem is well-defined, we need to define the correspondence between $\mathcal{P}$ and query results of G, see Section 3 for more details.

[2] Throughout, $\tilde{O}(f) := O(f \operatorname{poly} \log f)$.

- **Techniques from poset sorting.** The missing edges in G can increase the query complexity of existing algorithms for poset sorting [11, 10]. In these algorithms, the overall framework is to incrementally add elements to the current sorting, and when an element x is to be added, a binary search is used to figure out the relation between x and every other element added so far. A crucial step to bound the query complexity is that there always exists a path cover of size k (which is the width) in the induced subgraph of the added elements, and this ensures only k binary search suffices. While this is true for complete graphs, the width k can no longer upper bound the size of the path cover only by using edges in an induced subgraph of a general G.
- **Techniques from generalized sorting.** Incomparable edges $(u, v) \in E$ ($u \not\sim v$) reveal very little information about ordering, hence algorithms for generalized sorting should avoid querying these edges. However, existing algorithms [18, 23, 22] for generalized sorting (designed for general query graph G) relies on a rough estimation of the relation between elements, and "useful" edges may be wrongly classified as incomparable edges. If this happens, then it is very difficult to detect the "useful" edge without querying a lot of incomparable edges, making existing algorithms less efficient.

1.1 Our Results

We give efficient algorithms for GPS that make $\tilde{O}(n \operatorname{poly}(k))$ queries for Erdős-Rényi stochastic query graphs (Theorem 1.1) and complete bipartite query graphs (Theorem 1.2), where k is the width of the poset throughout (see Section 3 for formal definitions of these query graph models). These are the first results for GPS parameterized by the width of the poset k, and the query complexity bound is nearly-optimal in n (and up to a factor of k). We obtain our results via a unified framework and it may be of independent interest (will be discussed in Section 4). These results are our main technical contributions.

▶ **Theorem 1.1** (Erdős-Rényi Stochastic Graphs). *There exists an algorithm that solves GPS on Erdős-Rényi stochastic query graphs and width-k posets using $\tilde{O}(nk^2)$ queries with high probability. This holds regardless of the probability parameter $0 < p \leq 1$ in Erdős-Rényi $G(n, p)$.*

Roughly, the Erdős-Rényi stochastic query graph is a union of an adversarily choosen base graph and an Erdős-Rényi graph $G(n, p)$. Although our bound for Erdős-Rényi stochastic query graph does not depend on p, it still relies on the structural property of the Erdős-Rényi graph where edges are i.i.d. generated. This case of Erdős-Rényi query graph has been well studied in (total order) generalized sorting (i.e., $k = 1$), where [18] and [22] are milestones. Compared with [18], our result is significantly better than their $\min\{np^{-2}, n^{1.5}\sqrt{p}\}$, especially that our algorithm is $\tilde{O}(n)$ regardless of p and theirs can obtain near-linear query complexity only for a very limited range of p. On the other hand, compared to the more recent work [22] whose bound is $O(n \log(np))$, our result is worse by a $\operatorname{poly} \log n$ factor. However, our slightly worse bound generalizes to poset sorting and is also technically different. See Section 4 for a more detailed discussion. Finally, we remark that a unique feature of [22] is that when p is very small, say $p = 1/n$, then the complexity of sorting can even be better than $O(n \log n)$ which is the well-known lower bound for classic sorting. We leave it as an open question to figure out if one can achieve a similar bound for GPS with Erdős-Rényi query graphs.

▶ **Theorem 1.2** (Complete Bipartite Graphs). *There exists an algorithm that solves GPS on complete bipartite query graphs and width-k posets using $\tilde{O}(nk)$ queries with high probability.*

This result for complete bipartite graphs is tight up to $\text{poly}(\log n)$ factors, since the $\Omega(nk)$ lower bound for complete query graph in [10] still holds in the complete bipartite case. Our result is also a generalization of a series work of nuts-and-bolts problems [1, 7, 2, 21], and our bound for $k = 1$ nearly matches the state-of-the-art bound for this problem (up to poly log factors, compared to the best $O(n \log n)$ result in (total order) generalized sorting problem on complete bipartite graphs by [21]). Note that there is a difference between the nuts-and-bolts problem and the generalized sorting problem on complete bipartite graphs, where the nuts-and-bolts problem has an additional assumption that every node is assigned an edge with the query result "equal". With the help of the "equal" edge, a natural randomized quicksort-like algorithm achieves the query complexity $O(n \log n)$, by an $O(n)$ partition algorithm to partition nodes based on a randomly selected pivot. However, if the "equal" edge is not provided, it is already a non-trivial task to design a partition algorithm. This is also noted by [21], and they resolve the missing "equal" edges and design an $O(n \log n)$ sorting algorithm with some other indirect methods. However, this indirect method does not yield a partition algorithm, and we find it hard to generalize these indirect methods to the setting of poset. In our result, we devised a partition algorithm for complete bipartite graphs without "equal" edges, and this type of partiton algorithm was unknown even for the total order setting. This partition step turns out to be useful and naturally generalizable to posets.

Weighted generalized sorting. Apart from the significance in its own right, another important implication of GPS is that it can be used as an intermediate step for other (total order) sorting problems. We showcase this idea by presenting new results for the weighted generalized sorting problem.

In the weighted (total order) generalized sorting problem, the query graph is weighted (with weight function $w : E \to \mathbb{R}_{\geq 0}$), and each query $(u, v) \in E$ incurs a weighted cost $w(u, v)$ instead of a unit cost. Since the objective is weighted, we measure the performance of the algorithm using the *competitive ratio* [16], defined as the total cost incurred by the algorithm divided by $\sum_i w(v_i, v_{i+1})$ ($v_1 \prec \ldots \prec v_n$ is the total order), which is the cost of cheapest proof, i.e. comparing every consecutive elements in the sorted order. We obtain the following result for weighted generalized sorting.

▶ **Theorem 1.3.** *There exists an algorithm that solves the weighted (total order) generalized sorting problem with competitive ratio $\tilde{O}(n^{1-1/(2W)})$, where W is the number of distinct weights in the graph.*

Indeed, obtaining nontrivial bounds for this weighted generalized sorting has been suggested as an *open question* by [9], and it received significant attention in various subsequent works [16, 3, 4, 13, 14]. However, these existing works mostly focus on understanding certain special cases of weights, such as bounded number of distinct weights [4, 13, 14] or structured/random weights [16, 17, 3]. For the general case, we are only aware of an $O(n)$ ratio [16], which is trivial in the unweighted case since one can query all edges, but is already nontrivial in the weighted setting.

Our result makes progress on the weighted generalized sorting problem, and our bound implies a strictly sublinear ratio when the number of distinct weights is bounded (and the weights can take any non-negative values). This improves over the known $O(n)$ ratio in [16], and our ratio also matches ratios for several notable special cases. When $W = 1$ which reduces to (total order) generalized sorting, our bound matches the $\tilde{O}(n^{1.5})$ query complexity in [18] which is the state-of-the-art for dense graphs (for sparse graphs a $\sqrt{mn}$ bound was

obtained in [22], where m is the number of edges in the query graph). Moreover, we give an improved analysis of our algorithm for the case when the weights are well-separated, and this result matches an $\tilde{O}(n^{3/4})$ ratio, obtained in a recent work [13, 14], for the case when the weights are picked from $\{0, 1, n, \infty\}$ (where ∞ weight can be interpreted as "missing edge" in the query graph in our case).

▶ Remark 1.4. In several previous works [16, 18, 22] it has been mentioned that the ratio for finding a maximum element in a weighted query graph is $\Omega(n)$, and this seems to suggest that $\Omega(n)$ is also a lower bound for sorting (since sorting implies finding the maximum). However, this is not true, for two reasons. One is that the ratio in the maximum-finding problem is defined with respect to the cost of a minimum-weight certificate for the maximum, which can be much smaller than the $\sum_i w(v_i, v_{i+1})$ cost for identifying total order in sorting. Secondly, the hard instance in the lower bound of maximum-finding [16] only uses three types of weights, and by our upper bound, this type of instance cannot be hard for sorting. A similar discussion of this gap was also made in [13, 14]. In fact, a further implication of our result is that, in order to prove $\Omega(n)$ lower bound for weighted generalized sorting, one must use at least $\Omega(\log n)$ distinct weights in the hard instance.

Auxiliary problem: GPS with comparable edges (GPSC). As we mentioned, GPS is used as an important intermediate step for obtaining Theorem 1.3. In particular, we consider a special case of GPS whose query graph consists of *comparable* edges only, i.e., $(u, v) \in E$ only if $u \prec v$ or $v \prec u$, and we call this special case the GPS with comparable edges (GPSC). Due to the fact that GPSC is in between GPS and (total order) generalized sorting, and that it may be useful for other sorting problems, the result of this problem, stated below, may be of independent interest.

▶ **Theorem 1.5** (GPSC). *There exists an algorithm that solves GPSC on general query graphs and width-k posets using $\tilde{O}(nk + n^{1.5})$ queries with high probability.*[3]

As mentioned, Theorem 1.5 is a crucial subroutine for Theorem 1.3, but to obtain a sublinear ratio for weighted generalized sorting (provided that the number of distinct weights is bounded), any $n^{2-\epsilon}$ query bound for GPSC suffices, although it may lead to a worse constant in the exponent of n in the ratio (i.e., worse than $O(n^{1-1/(2W)})$ but still $o(n)$). We give a detailed overview on how this can be used to obtain Theorem 1.3 in Section 4.

▶ Remark 1.6 (Comparison to [13, 14]). A recent work [13] independently studies similar problems that are relevant to our results.[4] The meta problem suggested in their paper is a weighted version of the GPSC problem, which they call universal sorting. This weighted GPSC model is more general than the unweighted GPSC (which is the model of our Theorem 1.5), but is incomparable with GPS which we focus on. However, due to a lower bound of $\Omega(n)$ for the universal sorting model, their concrete results/upper bounds only deal with special cases. We compare with each of their main results in the following; in short, all their results concern problems that are either special cases or immediate extensions of ours, but the bounds may not be comparable (i.e., they may obtain better bounds for special cases or use a different measure of complexity).

[3] This algorithm may run in exponential time.

[4] [14] is another version of [13] and is to appear in ITCS'24. We focus on [13] since [14] only contains a subset of results of [13].

- **Weighted generalized sorting with special weights** [13, Theorem 1]. They study a special case of weighted generalized sorting, where all weights are among $\{0, 1, F\}$ ($F > n^{3/4}$), and they achieve a $\tilde{O}(n^{3/4})$ competitive ratio. Our algorithm for general weights achieves the same competitive ratio for this special case.
- **Bichromatic sorting** [13, Theorem 2]. They study a special case of weighted generalized sorting, where the comparison costs are among $\{1, \alpha, \beta\}$ and are assigned to query graph in a structured way. They achieve a nearly optimal ratio of $O(\log^3 n)$ for this special case, and our general algorithm for weighted generalized sorting (our Theorem 1.3) can also be applied which yields a ratio of $O(n^{5/6})$.
- **GPSC with weight-0 edges** [13, Theorem 6]. They propose an algorithm for GPSC that additionally allows weight-0 edges. This algorithm is used as a subroutine in their sorting algorithm. Our algorithm for GPSC (our Theorem 1.5), even though not explicitly stated to be able to handle the weight-0 edges, can still be applied by treating all weight-0 edges as weight-1. Hence, we still obtain the same bound of $\tilde{O}(nk + n^{1.5})$ as in no weight-0 edges, and this is actually better than their $O(n^{1.5}k \log n)$ bound.
- **GPSC on complete bipartite graphs** [13, Theorem 22]. They study GPSC problem on complete bipartite graphs. They measure the performance by instance optimality instead of query complexity, and their algorithm is $O(\log^3 n)$ instance optimal. Since GPSC is a special case of GPS problem, our algorithm for GPS problem on complete bipartite graphs (our Theorem 1.2) can be applied which yields a query complexity of $\tilde{O}(nk)$. This is also a near optimal result because optimal solution requires $\Omega(nk)$ queries in the worst case.

1.2 Technical Contribution

We give a highlight of technical challenges and our technical contributions. A more detailed overview of the proof can be found in Section 4.

A general framework for GPS. Previous algorithms for generalized sorting [18, 23, 22] use an incremental method to iteratively discover the (nearly-)minimum element, but this does not work directly in GPS due to incomparable edges and non-unique minimal elements. We develop a general framework for GPS, which reduces GPS to finding a linear extension, and we further show this linear extension can be found by a quicksort-like algorithm proposed by [18, 10]. The quicksort-like algorithm relies on a partition algorithm that receives as input a subgraph and a pivot, and it partitions the subgraph into three parts: less than, larger than, and incomparable to the pivot. We require the partition algorithm to have a refined query complexity that only depends on the width of the part of the input subgraph (to the partition algorithm) that is comparable to the pivot. This refined property of partition algorithms enables us to obtain $\tilde{O}(nk)$ query complexity for the quicksort-like algorithm. Hence, this framework is capable of obtaining (nearly) tight bounds when combined with carefully designed partition algorithms (for instance, for Erdős-Rényi query graphs and complete bipartite graphs), which may be of independent interest.

Novel partition algorithms for Erdős-Rényi graphs based on stochastic BFS. Our partition algorithm for Erdős-Rényi graphs is based on a stochastic BFS, where the key idea is to skip a vertex from the BFS queue if that vertex has been visited by sufficiently many other vertices. This still guarantees the correctness with high probability due to the property of Erdős-Rényi graph. To make sure we trim most vertices in a few iterations, we also run the BFS in a random order of vertices. Previously, algorithms for Erdős-Rényi graphs were only known

for generalized sorting (without considering a poset), and the techniques are not readily applicable. In particular, the framework of [18] requires an algorithm with a subquadratic query complexity for general query graph, which is not available in GPS. Another recent work [22] uses a very different approach, but the efficiency of one of its subroutines relies on the uniqueness of the minimal element. Hence it is highly nontrivial to generalize to the poset setting while achieving subquadratic complexity.

Weighted generalized sorting via new algorithms for GPSC. To achieve the $\tilde{O}(n^{1-\frac{1}{2W}})$ competitive ratio, we partition the edge set into cheap and expensive edges according to a threshold, and the two cases are balanced and solved by one of the following two algorithms: a) a new sorting algorithm that may receive a partially sorted graph (i.e. a partial order) as extra input and the competitive ratio depends on the width of the input partial order; and b) a new GPSC algorithm as in Theorem 1.5. In our new GPSC algorithm, we employ the framework of sorting with predictions [18, 23, 22] (which was proposed for generalized sorting), where we construct a prediction graph that "guesses" the direction of the edges, and make decisions and refine the prediction in an iterative manner. To achieve better query complexity, we devise a stronger predictor that has an "everywhere" guarantee for each vertex, as opposed to having a collective bound on the total number of wrongly predicted edges, as considered in [23, 22].

1.3 Related Work

Parameterization other than the width of the poset which we use was also considered in the literature, and they are generally not comparable to our results. In [5, 6], the GPS is parameterized by the number of missing edges $q = \binom{n}{2} - m$ (where m is the number of edges in the query graph) while there is no restriction on the poset, and nearly-tight bounds were obtained with respect to q. In a recent work [24], the query graph can be general but the poset is assumed to be a tree and is parameterized by the maximum degree d, and they also obtained nearly tight query complexity bounds.

In addition to generalized sorting problems, other related problems were also considered. Examples include noisy sorting/selection [12, 8, 15] and generalized/weighted selection [16, 20, 3, 10].

2 Preliminaries

Throughout, we use $\mathcal{P} = (V, \prec)$ to denote a poset, and we let $k_{\mathcal{P}}$ denote the width of $\mathcal{P}$. By Dilworth's Theorem, a poset of width k can be decomposed to k chains, say $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$, where the elements are comparable to each other on each chain. Suppose $\mathcal{P} = (V, \prec)$ is the underlying poset of GPS problem, we directly use k to denote $k_{\mathcal{P}}$. For every $X \subseteq V$, let k_X denote the width of poset $(X, \prec)$. For every $X \subseteq V, v \in X$, let $X_{\prec v}, X_{\nsim v}, X_{\succ v}$ denote the elements of X that are smaller than v, incomparable with v and larger than v respectively. Recall that $x \nsim y$ denotes "x is incomparable with y". For some set X, denote the set of permutations of X by $\mathrm{perm}(X)$. For every set X with a total order $(X, \prec)$ and every $x \in X$, define $\mathrm{rank}_X(x) = |\{y \in X \mid y \prec x\}| + 1$ as the rank of x. Similarly, let $\mathrm{rank}_{\mathbf{p}}(x)$ be the rank of x in permutation $\mathbf{p}$.

For a graph $G = (V, E)$ and a vertex subset $S \subseteq V$, denote $G[S]$ as the induced subgraph of G on S, whose vertex set is S and the edge set is $\{(u, v) \in E : u, v \in S\}$. Given a directed acyclic graph (DAG) $\vec{G}(V, \vec{E})$, let $\mathcal{P}(\vec{G})$, which stands for induced poset of $\vec{G}$, be a poset $\mathcal{P}'(V, \prec)$, such that $\forall u, v \in V$, $u \prec v$ if and only if there is a directed path from u to v in

$\vec{G}$. This implies that $u \not\prec v$ if and only if u cannot reach v and v cannot reach u in $\vec{G}$. By Dilworth's Theorem, a DAG can be covered by k paths, and these paths form a path cover of $\vec{G}$, where k is the width of $\mathcal{P}(\vec{G})$, i.e., every vertex is contained in at least one path (and may be contained in multiple paths).

3 Models

Generalized poset sorting (GPS). Formally, in the GPS problem, we are given an n-element underlying (unknown) poset $\mathcal{P} = (V, \prec)$ and a graph $G = (V, E)$. An oracle receives queries of the form $(u, v) \in E$, and returns the relation of u, v in $\mathcal{P}$. The goal of GPS is to use the minimum number of queries to fully recover $\mathcal{P}$, i.e., $\forall u, v \in V$, correctly determine the relation of u, v in $\mathcal{P}$.

Model of query graphs in GPS. To make sure the problem is well-defined, e.g., G has sufficient edges to recover $\mathcal{P}$, we need to add some further constraints on the query graph G. Specifically, we enforce the following: let $\vec{E} := \{(u, v) \in E : u \prec v\}$ and $\vec{G} = (V, \vec{E})$ (noting that $\vec{G}$ is defined with respect to both G and $\mathcal{P}$) then

$$\mathcal{P} = \mathcal{P}(\vec{G}). \tag{1}$$

This is well-defined if G is deterministic (for instance G is a complete bipartite graph), but for stochastic case enforcing this directly may cause randomness issues. Hence, we discuss how we define the Erdős-Rényi stochastic query graph in more detail in the following.

Model of query graphs in GPS: Erdős-Rényi stochastic case. Let $G(n, p)$ denote the Erdős-Rényi random graph with n vertices and probability parameter $0 \leq p \leq 1$. Specifically, this $G(n, p)$ is generated by independently adding an undirected edge (u, v) with probability p for every vertex pair $u \neq v$. Clearly, this random graph is unlikely to be able to uniquely identify $\mathcal{P}$. Hence, we still wish to enforce the property stated in (1). Specifically, we need to add to the Erdős-Rényi graph a *minimal* DAG G_{base} which is a "base graph". Here, we say a DAG $\vec{G}$ is minimal if there is no redundant edge in $\vec{G}$, where we call an edge $u \to v$ redundant if we have $\exists x \notin \{u, v\}$, $u \to x$ and $x \to v$. Formally, we have the following definition, and it indeed satisfies (1) (stated in Fact 3.2).

▶ **Definition 3.1.** *Fix some minimal DAG $\vec{G}_{\text{base}}$ such that $\mathcal{P} = \mathcal{P}(\vec{G}_{\text{base}})$, denoting its underlying undirected graph as G_{base}, the Erdős-Rényi stochastic query graph G is a union of G_{base} and $G(n, p)$.*

▶ **Fact 3.2.** *The random query graph $G = (V, E)$ defined in Definition 3.1 satisfies* (1) *with probability* 1*, namely,* $\Pr[\mathcal{P} = \mathcal{P}(\vec{G})] = 1$ *where* $\vec{G} := \{(u, v) \in E \mid u \prec v\}$.

Indeed, this definition can be viewed as a generalization of the stochastic setting in the (total order) generalized sorting [18, 22], here they use a directed Hamiltonian Path as a base graph (which is the only minimal choice in the total order setting).

Generalized poset sorting with comparable edges (GPSC). In this model, all edges in G are comparable edges. Specifically, when an edge in E is queried, the answer will only be $v \prec u$ or $u \prec v$, corresponding to $\mathcal{P}$. We remark that when $\mathcal{P}$ is a total order set, then the model draws back to the generalized sorting model, so GPSC is already a generalization of generalized sorting.

Weighted generalized (total order) sorting. In this model, the poset is total order, and the query graph is weighted by a weight function $w : E \to \mathbb{R}_{\geq 0}$. We aim to minimize the sum of costs we pay to solve the GPS under this setting. We evaluate our algorithms by the competitive ratio, which is the maximum ratio taken over all possible inputs, measured by the cost of the algorithm, denoted as ALG, divided by OPT which is the sum of costs $\sum_i w(v_i, v_{i+1})$ where $(v_1, \ldots, v_n)$ is the total order defined by $\mathcal{P}$.

4 Proof Overview

Due to space limit, we give a sketch for the proofs of main results in this section, and the full proof, except for a more detailed presentation of our main framework (in Section 5), can be found in the full version.

A general framework for generalized poset sorting. As mentioned, we obtain algorithms for GPS via a new unified framework. In this framework, we first reduce the GPS to finding a *linear extension* (Lemma 5.1). A linear extension for the poset $\mathcal{P} = (V, \prec)$ is a total order such that $\forall x, y \in V$, if $x \prec y$ then x appears before y in the total order. Finding a linear extension is an interesting problem in its own right, and it has also been studied in [18, 10]. However, previous studies did not establish the connection between GPS and linear extension, which we do in our framework.

To find the linear extension, we employ a quicksort-like algorithm to randomly select a pivot element $v \in V$ and partition the elements into three parts, elements smaller than v, elements incomparable with v and elements larger than v. Given this partition, one can compute the linear extension of these three parts recursively and combine them in the order of smaller-incomparable-larger to obtain the linear extension of $\mathcal{P}$.

This quicksort-like algorithm was also used in [10] to find a linear extension for complete query graphs. While their analysis may be adapted to the general query graph case, it only leads to sub-optimal bounds with respect to k. We give new analysis to this quicksort-like algorithm, and we are able to obtain an improved dependence in k, provided that the partition algorithm have a refined query complexity that only depends on the width of the part of the input subgraph (to the partition algorithm) that is comparable to the pivot (see Lemmas 5.3 and 5.4 for more details). We manage to design partition algorithms with refined query complexity for both Erdős-Rényi and complete bipartite query graphs.

Now we explain our new steps in the analysis to the quicksort-like algorithm. In [10], it is observed that the depth of the recursion tree is $O(k + \log n)$. This is good enough for complete query graphs, since the partition step can be done in $O(n)$, and this, combined with the depth of the recursion tree, translates to an $O(nk + n \log n)$ bound. However, when G is not a complete graph, the partition problem often requires $\Omega(n)$ queries, say $O(nk^c)$ queries, then the analysis in [10] leads to an $\tilde{O}(nk^{c+1})$ bound, which introduces an additional k factor. In order to avoid this additional k factor, we require partition algorithms to use $O(nk_v^c)$ queries that depend on k_v, which denotes the width of elements comparable with pivot vertex v. A crucial observation is that, if k_v is small, then the partition algorithm uses few queries, and if k_v is large, then the next pivot v' (in the incomparable part) is likely to have a small $k_{v'}$.

Partition algorithms. For the partition step, if it were the complete graph case, we could directly query the relations between the pivot and every other element using $n - 1$ queries. However, this simple but efficient bound is no longer easily obtainable when the query graph

is not complete. Nonetheless, we introduce novel ideas for this partition step, and we manage to obtain algorithms that use $\tilde{O}(nk^2)$ queries for Erdős-Rényi query graphs and $\tilde{O}(nk)$ queries for complete bipartite query graphs.

Partition algorithms: Erdős-Rényi graphs. It is helpful to interpret the problem as a graph problem. We define a directed graph $\vec{G}$ from G, by defining the direction of every edge $(u, v) \in E$ according to the relation between u, v, i.e. the direction is $u \to v$ if and only if $u \prec v$. Then for every vertex u, u is smaller than the pivot vertex if and only if there exists a path from u to pivot in $\vec{G}$. Hence, the partition problem reduces to finding all vertices that can be reached from the pivot vertex. This graph problem may be solved using BFS, but a vanilla BFS needs to query all edges, which is too costly. To resolve this issue, we design a variant of BFS that can make use of the structure of Erdős-Rényi graphs, called Skip-BFS.

We start by giving the overall intuition by assuming we are given a chain decomposition of the poset (which is of size k, guaranteed by Dilworth's Theorem). An important property of Erdős-Rényi $G(n, p)$ is that, if we select $O(p^{-1} \log n)$ arbitrary vertices, then every vertex is adjacent to at least one selected vertex[5]. Hence, if we select the $O(p^{-1} \log n)$-largest vertices from each chain in the chain decomposition of the poset then every vertex has outgoing edges to at least one selected vertex. Exploring (the neighbors of) these selected vertices only takes $\tilde{O}(kp^{-1} \cdot np) = \tilde{O}(nk)$ queries, and this finishes the partition.

However, the chain decomposition is not known to our algorithm a priori. Thus, we need a method to gauge whether a vertex is worth exploring, i.e., it is sufficiently large in its chain. To this end, Skip-BFS maintains a counter $c[v]$ for each vertex v, which is initialized as some parameter $R = \Theta(\log n)$. Then, if some vertex v becomes the current vertex for which we start to explore its neighbors, we decrease the counter $c[u]$ by 1 for every v's neighbor u such that u is smaller than v. When the counter of some vertex u is decreased to 0, we skip this vertex u by removing it from the BFS queue. Such u can be safely skipped since Skip-BFS has already explored $R \geq \Omega(\log n)$ vertices that are larger than u, and these vertices are likely to cover all incoming vertices of u. To see this, since u's counter is decreased $R \geq \Omega(\log n)$ times, we already visited $O(p^{-1} \log n)$ vertices that are larger than u, and that each such vertex connects to p fraction of vertices smaller than u. Hence, these already-visited $O(p^{-1} \log n)$ vertices connect to/cover all vertices that are smaller than u. Finally, to guarantee the efficiency of this process, we need to explore vertices in a random order for each BFS step (in step i we explore all vertices with distance i from the starting vertex), in order to trim most u's in only a few steps. This eventually leads to an $\tilde{O}(nk^2)$ time partition algorithm for Erdős-Rényi query graphs.

Compared with the approach in [22] who gave an algorithm for the total order case that uses $O(n \log(np))$ queries which is tight, our bound is comparable, but our approach is conceptually different. In fact, it is unclear if their approach can be efficiently generalized to the poset case. In their algorithm, they repeatedly find the minimum vertex of the current graph and remove it. To find the minimum vertex, they identify a set of candidate vertices and then trim the wrong ones by testing if there is an incoming edge, which requires querying the edges between the candidates and other vertices. However, in GPS, there are multiple minimal vertices, and it is nontrivial to bound the number of candidates and the adjacent edges to query since one cannot stop before one is certain that the surviving candidates are minimal. Hence, it is nontrivial to generalize their approach to GPS using even subquadratic queries.

[5] This does not always happen and only with high probability, but in the following discussions we ignore this and talk about the typical behavior.

Partition algorithm: complete bipartite graphs. Suppose the two parts of the bipartite graph are A and B, and suppose the pivot is $b \in B$. Let $A_{\succ b}$ and $B_{\succ b}$ be the elements in A and B that are greater than b, respectively. We focus the discussion on finding the elements that are larger than b, i.e., $A_{\succ b} \cup B_{\succ b}$. Since the graph is complete bipartite, it is easy to obtain $A_{\succ b}$, but it is nontrivial to obtain $B_{\succ b}$ since one cannot directly compare any other point in B with the pivot b. A natural idea to deal with this is to find the minimal elements in $A_{\succ b}$ so that one can figure out $B_{\succ b}$ from these elements. However, finding the minimal element is technically nontrivial even in the total order setting (whose minimal is unique), let alone there may be multiple minimal elements in posets. Indeed, the existing algorithm for the total order setting does not seem to make progress on this simple and fundamental task, and they solved the problem via other indirect methods [1, 7, 2, 21]. We provide a completely new algorithm to find the minimal elements for poset in bipartite complete graphs, which is a technical contribution to the study of sorting and selection for bipartite graphs.

We first devise a FindMin procedure that finds a "local" minimal element in $A_{\succ b}$ (the "local" is due to the fact that we may make iterative calls and only run the procedure on an induced subgraph). Then, we apply this FindMin iteratively to both find the minimal elements of $A_{\succ b}$ and construct $B_{\succ b}$. In particular, every time we run FindMin to obtain a vertex a, we try to find $B_{\succ a}$ and expand the currently found $B_{\succ b}$, and remove a from A to continue. Each iteration takes $\tilde{O}(n)$ queries. Then, using the property and the randomness of FindMin, the new element a' that we find must be smaller than a if they are comparable, and this also shrinks the distance from a to the pivot by a constant factor with good probability. Finally, if one takes one chain in a chain decomposition, this entire process would typically run on a vertex from this chain for $O(\log n)$ iterations. Summing over k chains, the total query time is $\tilde{O}(nk)$ in the typical case.

GPSC. Recall that there are no incomparable edges in the query graph (which means every edge $(u, v) \in E$ satisfies either $u \prec v$ or $v \prec u$) of the GPSC problem. This conceptually simplifies the problem since this avoids the issue of gaining essentially no information from querying an incomparable edge. Technically, this allows us to apply techniques/frameworks developed for generalized sorting problems, which crucially relies on the information gained from querying an edge. Specifically, we use an idea proposed in [18] and further developed in [22], where one first constructs a prediction graph which "guesses" the direction/relation of all edges in the query graph. Then, an incremental algorithm that iteratively adds a currently "minimal" element, i.e., an element u with a small number of "incoming edges" (which are the edges (v, u) such that $v \prec u$) in the prediction, is employed to generate the sorting.

To apply this framework to GPSC, especially to achieve a linear dependence in k, we cannot use [18, 22] in a black-box way, since we need a stronger predictor such that it has a bounded number of wrongly predicted edges *everywhere*: $\forall v$, there are $\tilde{O}(\sqrt{n})$ wrongly predicted edges among all adjacent edges to v. This is stronger than the previously designed predictors [18, 22], since [18] only guarantees an $\tilde{O}(\sqrt{n})$ absolute error for the in-degree of every vertex (instead of the edge predictions), and [22] only guarantees an overall number of wrong edges (instead of our "everywhere" guarantee). We manage to obtain such a stronger predictor.

Then, with this predictor, we iteratively maintain a current set A of sorted vertices, and we show it is possible to identify a key vertex $v \in A$, whose incoming vertices (with respect to the prediction) can be partitioned into $X_v \subseteq A$ and $Y_v \subseteq V \backslash A$, such that the poset induced by X_v still has width k, and that $|Y_v| = O(\sqrt{n})$ (by the stronger guarantee of the predictor).

This, together with the fact that X_v can be decomposed into k chains, implies that one can verify/discover all incoming edges (from the predictor) to v using $O(k \log n + \sqrt{n})$ queries. In total, this entire iterative process of updating A happens $O(n)$ times, which leads to our final query bound $\tilde{O}(nk + n^{1.5})$.

Weighted generalized sorting. We start with designing an $O(k \operatorname{poly} \log n)$-competitive algorithm $\mathcal{A}$, whose input consists of a chain decomposition (of the total order) of size k in addition to the weighted query graph (and the underlying total order), for weighted generalized sorting. Notice that one can always feed a trivial chain decomposition of size n to $\mathcal{A}$ and obtain $\tilde{O}(n)$ competitive ratio, which is already nontrivial as we mention in Section 1.1. Although the algorithm by [16] can also achieve an $O(n)$ ratio for weighted generalized sorting, it only works for the case when all chains are single nodes (i.e., $k = n$), hence it is not useful for obtaining a sublinear ratio.

Next, we employ a threshold algorithm to "combine" $\mathcal{A}$ with Theorem 1.5 to obtain a sublinear ratio when the number of distinct weights is bounded. Suppose the weights are $w_1 < \ldots < w_W$. We use a threshold parameter $1 \leq \tau \leq W$, and define G_τ as the subgraph of the query graph with edge weights *at most* w_τ. We also consider the poset $\mathcal{P}_\tau$ induced by G_τ, and let k_τ denote its width. We start with running Theorem 1.5 on $(G_\tau, \mathcal{P}_\tau)$ and ignoring the weight, which takes $\tilde{O}(nk_\tau)$ queries (assuming $k_\tau \geq \sqrt{n}$ in this discussion), and it generates a chain decomposition of $\mathcal{P}_\tau$. Notice that this chain decomposition of $\mathcal{P}_\tau$ is also a chain decomposition of $\mathcal{P}$ since they are supported on the same element set. Then, we feed this chain decomposition to $\mathcal{A}$, and use the output of $\mathcal{A}$ as the result. The entire algorithm achieves an $\tilde{O}(\frac{nk_\tau \cdot w_\tau}{\mathrm{OPT}} + k_\tau)$ ratio, and we can further show this ratio is at most $O(\frac{nw_\tau}{w_{\tau+1}})$ (assuming that k_τ is not the dominating factor), which depends on the "gap" between two adjacent weights. The final result can be achieved by fine-tuning of τ to minimize this ratio: if all weights are of a small gap, then one can view it as the unweighted case and run Theorem 1.5 directly, and otherwise, we have a significant gap which still allows a sublinear ratio.

5 A General Framework for Generalized Poset Sorting

In this section, we present our framework for GPS. As mentioned, this framework consists of two steps: it first reduces GPS to finding a linear extension, and eventually reducing the task of finding a linear extension to constructing a partition oracle. We start with formally defining the mentioned linear extension problem and the partition problem. We establish two sets of technical lemmas that relate GPS with linear extension Lemma 5.1 and the partition problem Lemmas 5.3 and 5.4, respectively.

Linear extension problem. In linear extension problem, there is an underlying poset $\mathcal{P} = (V, \prec)$ and query graph $G = (V, E)$. The algorithm receives G as input and has access to an oracle, which accepts queries u, v such that $(u, v) \in E$ and answers the relation of u, v in $\mathcal{P}$. The algorithm needs to compute a linear extension of $\mathcal{P}$ by making as few queries as possible. We call $p_1, p_2, \ldots, p_n \in \operatorname{perm}(V)$ a linear extension of $\mathcal{P}$ if and only if for every $1 \leq i < j \leq n$, $p_i \not\succ p_j$ (i.e. either $p_i \prec p_j$ or $p_i \nsim p_j$).

▶ **Lemma 5.1** (Linear Extension to Poset). *There exists an algorithm that given a linear extension of the underlying width-k poset $\mathcal{P}$ solves GPS in $\tilde{O}(nk)$ queries.*

Lemma 5.1 shows that given any linear extension of $\mathcal{P}$, we can solve $\mathcal{P}$ using $\tilde{O}(nk)$ queries. Specifically, our algorithm maintains a vertex set X such that all directions of edges in $G[X]$ is determined. Initially, X is empty. The vertices are added to X by their order in the linear

extension. When vertex v is added to X, our algorithm needs to determine all directions of edges between X and v. In the proof of Lemma 5.1, we show that X can always be decomposed into at most k paths. For every vertex v and every path $P = p_1 \prec p_2 \prec \ldots \prec p_s$, there exists a, b such that

- For every $1 \leq i \leq a$, $p_i \prec v$.
- For every $a < i < b$, $p_i \nsim v$.
- For every $b \leq i \leq s$, $p_i \succ v$.

a, b can be found by binary search. Hence the directions of edges between X and v can be determined in $O(k \log n)$ queries. Our algorithm is shown as Algorithm 1.

Algorithm 1 Constructing GPS using Linear Extension.

```
procedure GPS(p)
    input: p, a linear extension of 𝒫
    for i ← 1, 2, ..., n − 1 do
        let X_i be {p_1, p_2, ..., p_i}
        compute a path cover of induced subgraph G⃗[X_i], denoted by P_1, P_2, ..., P_{w_i} (w_i
    denotes the width of poset 𝒫(G⃗[X_i]))
        determine the directions of edges between X_i and p_{i+1} by applying binary search
    between P_j and p_{i+1} for every 1 ≤ j ≤ w_i
    end for
end procedure
```

▶ **Lemma 5.2.** *For every linear extension $p_1, p_2, \ldots, p_n$ of $\mathcal{P}$ and every $1 \leq i \leq n$, we have $\mathcal{P}(\vec{G}[X_i]) = (X_i, \prec)$ where $X_i = \{p_1, p_2, \ldots, p_i\}$.*

Proof. To prove $\mathcal{P}(\vec{G}[X_i]) = (X_i, \prec)$, we show that for every $u, v \in X_i$, $u \prec v$ if and only if u can reach v in $\vec{G}[X_i]$.

- If u can reach v in $\vec{G}[X_i]$, then u can also reach v in $\vec{G}$, which implies $u \prec v$.
- If $u \prec v$, then there exists a path $u \to q_1 \to q_2 \to \ldots \to q_\ell \to v$ in $\vec{G}$. For every $1 \leq j \leq \ell$, suppose $q_j \notin X_i$, let $q_j = p_a, v = p_b$, we have $p_a \prec p_b$ and $b \leq j < a$, which contradicts the definition of linear extension. Hence we have $q_1, q_2, \ldots, q_\ell \in X_i$, which implies u can reach v in $\vec{G}[X_i]$. ◀

Proof of Lemma 5.1. By Lemma 5.2, for every $1 \leq i < n$, $\mathcal{P}(\vec{G}[X_i]) = (X_i, \prec)$. The width of poset $(X_i, \prec)$ is no more than k since $X_i \subseteq V$. Hence, for every i, all directions of edges between $p_1, p_2, \ldots, p_i$ and p_{i+1} can be determined by applying at most k binary searches on the path cover of $\vec{G}[X_i]$. Clearly, this entire process takes $O(nk \log n)$ queries in total, which finishes the proof. ◀

Partition problem. The partition algorithm is defined on an underlying DAG $\vec{G} = (V, \vec{E})$ and a query graph $G = (V, E)$. Notice that partition algorithm is a pure graph problem (there is no poset in the problem definition). The algorithm receives G and vertex $p \in V$ as input and has access to an oracle, which accepts queries u, v such that $(u, v) \in E$ and answers the relation of u, v in $\vec{G}$. There are three possible relations, u can reach v, v can reach u, neither u nor v can reach each other. The algorithm needs to compute $V_{\to p}, V_{\nrightarrow p}, V_{\leftarrow p}$ by making as few queries as possible, where

- $V_{\to p} = \{u \in V \mid u \neq p, u \text{ can reach } p\}$
- $V_{\leftarrow p} = \{u \in V \mid u \neq p, p \text{ can reach } u\}$
- $V_{\nrightarrow p} = \{u \in V \mid u \neq p, \text{neither } u \text{ nor } p \text{ can reach each other}\}$

The following two lemmas reduce the problem of finding linear extensions to finding a partition of the elements with respect to a given pivot. These two versions of lemmas are essentially the same, except that one allows the partition oracle to make mistakes and with a randomized query complexity (but needs to succeed with high probability), and the other requires the correctness (with probability 1) and a good query complexity in expectation. We need these two since we find it is not trivial to convert one to the other, and our downstream algorithms may need both of them.

▶ **Lemma 5.3.** *If for every $X \subseteq V$ and $p \in X$, $\textsc{Partition}(G[X], p)$ correctly outputs $X_{\to p}, X_{\nleftrightarrow p}, X_{\leftarrow p}$ within $O(|X|(k_{X_{\prec v}} + k_{X_{\succ v}})f(n,k))$ queries with probability $1-\varepsilon$ (f is some function of n, k), then $\textsc{Part-to-LE}(V)$ outputs a linear extension of $\mathcal{P}$ in $O(nkf(n,k)\log^2 n)$ queries with probability of at least $1 - n\varepsilon - 2n^{-6}$.*

▶ **Lemma 5.4.** *If for every $X \subseteq V$ and $p \in X$, $\textsc{Partition}(G[X], p)$ always correctly outputs $X_{\to p}, X_{\nleftrightarrow p}, X_{\leftarrow p}$ and uses $O(|X|(k_{X_{\prec v}} + k_{X_{\succ v}})f(n,k))$ queries in expectation, then $\textsc{Part-to-LE}(V)$ outputs a linear extension of $\mathcal{P}$ in $O(nkf(n,k)\log^2 n)$ queries in expectation.*

References

1 Noga Alon, Manuel Blum, Amos Fiat, Sampath Kannan, Moni Naor, and Rafail Ostrovsky. Matching nuts and bolts. In *SODA*, pages 690–696. ACM/SIAM, 1994.

2 Noga Alon, Phillip G. Bradford, and Rudolf Fleischer. Matching nuts and bolts faster. *Inf. Process. Lett.*, 59(3):123–127, 1996.

3 Stanislav Angelov, Keshav Kunal, and Andrew McGregor. Sorting and selection with random costs. In *LATIN*, volume 4957 of *Lecture Notes in Computer Science*, pages 48–59. Springer, 2008.

4 Indranil Banerjee and Dana Richards. Sorting under 1-∞ cost model. *CoRR*, abs/1508.03698, 2015.

5 Indranil Banerjee and Dana S. Richards. Sorting under forbidden comparisons. In *SWAT*, volume 53 of *LIPIcs*, pages 22:1–22:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.

6 Arindam Biswas, Varunkumar Jayapaul, and Venkatesh Raman. Improved bounds for poset sorting in the forbidden-comparison regime. In *CALDAM*, volume 10156 of *Lecture Notes in Computer Science*, pages 50–59. Springer, 2017.

7 Phillip G Bradford. Matching nuts and bolts optimally. Technical report, Max-Planck-Institut für Informatik, 1995.

8 Mark Braverman and Elchanan Mossel. Noisy sorting without resampling. In *SODA*, pages 268–276. SIAM, 2008.

9 Moses Charikar, Ronald Fagin, Venkatesan Guruswami, Jon M. Kleinberg, Prabhakar Raghavan, and Amit Sahai. Query strategies for priced information. *J. Comput. Syst. Sci.*, 64(4):785–819, 2002.

10 Constantinos Daskalakis, Richard M. Karp, Elchanan Mossel, Samantha J. Riesenfeld, and Elad Verbin. Sorting and selection in posets. *SIAM J. Comput.*, 40(3):597–622, 2011.

11 Ulrich Faigle and György Turán. Sorting and recognition problems for ordered sets. *SIAM J. Comput.*, 17(1):100–113, 1988.

12 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, 1994.

13 Mayank Goswami and Riko Jacob. Sorting with priced comparisons: The general, the bichromatic, and the universal. *CoRR*, abs/2211.04601v2, 2023. A subset of results in this paper has been accepted to ITCS 2024, as [14]. `arXiv:2211.04601`.

14 Mayank Goswami and Riko Jacob. An algorithm for bichromatic sorting with polylog competitive ratio. In *ITCS*, 2024. See also `arXiv:2311.05773`. A prelimary version [13] has appeared in arXiv.

15 Yuzhou Gu and Yinzhan Xu. Optimal bounds for noisy sorting. In *STOC*. ACM, 2023.

16 Anupam Gupta and Amit Kumar. Sorting and selection with structured costs. In *FOCS*, pages 416–425. IEEE Computer Society, 2001.

17 Anupam Gupta and Amit Kumar. Where's the winner? max-finding and sorting with metric costs. In *APPROX-RANDOM*, volume 3624 of *Lecture Notes in Computer Science*, pages 74–85. Springer, 2005.

18 Zhiyi Huang, Sampath Kannan, and Sanjeev Khanna. Algorithms for the generalized sorting problem. In *FOCS*, pages 738–747. IEEE Computer Society, 2011.

19 Shaofeng H. C. Jiang, Wenqian Wang, Yubo Zhang, and Yuhao Zhang. Algorithms for the generalized poset sorting problem, 2023. `arXiv:2304.01623`.

20 Sampath Kannan and Sanjeev Khanna. Selection with monotone comparison cost. In *SODA*, pages 10–17. ACM/SIAM, 2003.

21 János Komlós, Yuan Ma, and Endre Szemerédi. Matching nuts and bolts in o(n log n) time. *SIAM J. Discret. Math.*, 11(3):347–372, 1998.

22 William Kuszmaul and Shyam Narayanan. Stochastic and worst-case generalized sorting revisited. In *FOCS*, pages 1056–1067. IEEE, 2021.

23 Pinyan Lu, Xuandi Ren, Enze Sun, and Yubo Zhang. Generalized sorting with predictions. In *SOSA*, pages 111–117. SIAM, 2021.

24 Jishnu Roychoudhury and Jatin Yadav. Efficient algorithms for sorting in trees. *CoRR*, abs/2205.15912, 2022. `arXiv:2205.15912`.

Streaming Algorithms for Connectivity Augmentation

Ce Jin[1]
MIT, Cambridge, MA, USA

Michael Kapralov
EPFL, Lausanne, Switzerland

Sepideh Mahabadi
Microsoft Research–Redmond, WA, USA

Ali Vakilian
Toyota Technological Institute at Chicago (TTIC), IL, USA

Abstract

We study the k-connectivity augmentation problem (k-CAP) in the single-pass streaming model. Given a $(k-1)$-edge connected graph $G = (V, E)$ that is stored in memory, and a stream of weighted edges (also called links) L with weights in $\{0, 1, \ldots, W\}$, the goal is to choose a minimum weight subset $L' \subseteq L$ of the links such that $G' = (V, E \cup L')$ is k-edge connected. We give a $(2+\epsilon)$-approximation algorithm for this problem which requires to store $O(\epsilon^{-1} n \log n)$ words. Moreover, we show the tightness of our result: Any algorithm with better than 2-approximation for the problem requires $\Omega(n^2)$ bits of space even when $k = 2$. This establishes a gap between the optimal approximation factor one can obtain in the streaming vs the offline setting for k-CAP.

We further consider a natural generalization to the fully streaming model where both E and L arrive in the stream in an arbitrary order. We show that this problem has a space lower bound that matches the best possible size of a spanner of the same approximation ratio. Following this, we give improved results for spanners on weighted graphs: We show a streaming algorithm that finds a $(2t-1+\epsilon)$-approximate weighted spanner of size at most $O(\epsilon^{-1} n^{1+1/t} \log n)$ for integer t, whereas the best prior streaming algorithm for spanner on weighted graphs had size depending on $\log W$. We believe that this result is of independent interest. Using our spanner result, we provide an optimal $O(t)$-approximation for k-CAP in the fully streaming model with $O(nk + n^{1+1/t})$ words of space.

Finally we apply our results to network design problems such as Steiner tree augmentation problem (STAP), k-edge connected spanning subgraph (k-ECSS) and the general Survivable Network Design problem (SNDP). In particular, we show a single-pass $O(t \log k)$-approximation for SNDP using $O(kn^{1+1/t})$ words of space, where k is the maximum connectivity requirement.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases streaming algorithms, connectivity augmentation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.93

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2402.10806

1 Introduction

In the (weighted) *k-connectivity augmentation* problem (k-CAP), given a $(k-1)$-edge-connected n-vertex graph $G = (V, E)$ (possibly with parallel edges) together with a set of weighted candidate edges (also called *links*) denoted by $L \subseteq \binom{V}{2}$ and their weights

[1] Work done during an internship at Microsoft Research, Redmond.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 93; pp. 93:1–93:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$w\colon L \to \{0, 1, \ldots, W\}$, the goal is to find a minimum weight subset $S \subseteq L$ of the links such that $(V, E \cup S)$ is k-edge-connected. Augmenting connectivity is a crucial task for enhancing network reliability which can be used for strengthening the resilience of a network and ensuring uninterrupted access for all users. k-CAP is among the most elementary questions in Network Design, which is an important area of discrete optimization. The iterative rounding method of [29] provides a 2-approximation for a more general problem of *survivable network design* problem (SNDP). Untill very recently, nothing better than 2 approximation was known even for weighted *tree augmentation problem* (TAP). In a recent development, weighted k-CAP has witnessed breakthroughs with approximation factors below 2 [48, 49, 50]. The state-of-the-art for weighted k-CAP is $1.5 + \epsilon$ approximation.

In this work, we consider weighted k-CAP in the streaming model, which is one of the most common models for processing real-time and large-scale data. A graph streaming algorithm operates by processing a sequence of graph edges presented in any order (or in some applications in random order), reading them one by one. The primary objective is to design algorithms that can process the entire edge sequence and output an approximately efficient solution, making just one (or a few passes), while utilizing limited memory resources. Ideally, the space usage of the algorithm should be significantly smaller than the size of the n-vertex input graph (with possibly $O(n^2)$ edges), preferably $O(n \cdot \text{polylog}(n))$ memory, which is referred to as the *semi-streaming* model [18].

While graph problems such as minimum spanning tree [2, 47, 42], matching [39, 25, 5, 4, 30], spanners, sparsifiers and shortest paths [19, 7, 15, 2, 33, 28, 20, 21] have received significant attention in the streaming model, the connectivity augmentation problem, has received comparatively very limited study in this context. Prior to our result, only testing k-connectivity in streaming was studied [52, 13, 47], which showed that testing k-edge-connectivity in streaming requires $\tilde{\Theta}(nk)$ space in one pass, and $\tilde{\Theta}(n)$ space in two passes [46, 3]. See Appendix C for more discussion on related work.

1.1 Our Computational Models

In this work, we study graph augmentation problems in the streaming model of computation. The input to the k-CAP problem consists of two pieces of information, namely the $(k-1)$-connected network G and the set of links that can be used to augment connectivity.

Link arrival streaming. In the link arrival streaming model the graph G is presented to the algorithm first, and the cost of storing it does not count towards the space complexity of the algorithm. This is akin to the oracle model that is routinely used to study submodular function maximization in the streaming model (e.g., in [6, 43]): One thinks of having an oracle for the function being maximized. For submodular function maximization it is not always clear how to implement this oracle in small space, but in our case the actual cost of storing a sufficient representation of the graph G can be easily made $O(nk)$, and, with some work, even $O(n)$, as we now explain.

Note that a minimally k-connected graph has size $O(nk)$. So if the graph has larger size, one can process the edges of G (even in a streaming fashion) using a k-connectivity certificate of G that preserves all cuts of value at most k, and store this compact representation in $O(nk)$ space. Finally, one can apply even a more efficient preprocessing that preserves a similar information via a *cactus* graph with $O(n)$ edges. Then the problem becomes streaming *cactus augmentation*. The cactus augmentation problem itself is a well-studied problem in particular for designing approximation algorithms for k-CAP. To simplify the notation, throughout the paper, we assume the latter compact representation of size $O(n)$.

Fully streaming. Besides the most natural link arrival model defined above, we study the more general model where the edges of G and the links that can be used for augmentation may arrive in an interleaved fashion. This model is quite general: in particular, it allows for the edges of G to arrive *after* the links, in which case the algorithm must maintain a compressed representation of the stream of links that allows augmenting any given graph G presented later!

For the other graph problems studied in this paper, namely spanner, SNDP and k-edge connected spanning subgraph (k-ECSS), we consider the standard edge arrival streams in which edges of the input graph arrives one by one in an arbitrary order stream.

1.2 Our Results

In this paper, we focus on insertion-only streams, and provide the first streaming algorithms for k-CAP in link arrival streams and fully streaming. Table 1 summarizes our results.

Graph augmentation in link arrival. We show tight results for weighted k-CAP in link arrival streams (see first row in Table 1). Note that, while we can achieve a factor $2+\epsilon$ approximation in $O(\frac{n}{\epsilon}\log n)$ words of space, our lower bound shows that getting better than 2 approximation requires $\Omega(n^2)$ bits of memory. This establishes a gap between the streaming setting and the offline setting where strictly better than 2 approximation algorithms are known (e.g., see [50]). An easy argument shows that $\Omega(n)$ bits of space is necessary for achieving any approximation for k-CAP in link arrival streams (Proposition 2.12 in the full versoin), so our algorithm has nearly-tight space complexity. If one picks a k-connectivity certificate as the compact representation of G, the space complexity of the upper bound becomes $O(nk+\frac{n}{\epsilon}\log n)$.

Further, we study the Steiner tree augmentation problem (STAP) which is a generalization of the tree augmentation problem (TAP) in link arrival streams and provide matching upper and lower bounds (See the second row in Table 1). While our lower bound holds for link arrival streams, our algorithm works even in the more general fully streaming too. We remark that, while in the offline setting TAP and STAP admit similar approximations [45], there is a gap in their complexities in the streaming model.

Graph augmentation in fully streaming. We further show matching upper and lower bounds (up to a polylog(n) factor) for k-CAP in the fully streaming setting (see the lower section in the first row of Table 1). The main component in our algorithm for solving k-CAP is an improved streaming algorithm for constructing *spanners on weighted graphs*. In particular, our upperbound implies that spanner is an optimal "universal" augmentation set for k-CAP.

Improved streaming spanner in weighted graphs. Given an n-vertex graph $G=(V,E)$ with a weight function $w\colon E\to\{0,\dots,W\}$, a subgraph $H\subseteq G$ is a t-spanner of G if for every $(u,v)\in E$, the shortest uv-path in H has weight at most $t\cdot w(uv)$. In streaming spanner, which is a well-studied problem [7, 15, 2, 33, 20], edges of E arrive in an arbitrary order stream. While by using the standard weight-based partitioning trick, constructing an $O(t)$-spanner in $O(n^{1+1/t}\cdot\log W)$ words of space in one pass over the stream is straightforward (e.g., mentioned in [21]), it was not known whether the dependence on $\log W$ is crucial.[2]

[2] We remark that our contribution in removing the dependence on $\log W$ from the number of edges in spanner (and consequently from k-CAP) is conceptually interesting, as most graph streaming algorithms are mainly designed for unweighted graphs, and extending them to the weighted case typically incurs a $\log W$ loss.

Table 1 Summary of our results for k-CAP, STAP, Spanner and SNDP in steaming models. All our problems are *weighted*. The space upper bounds are measured in words, while the lower bounds are in bits. We use $\tilde{O}(f)$ to mean $O(f \cdot \text{polylog} f)$ (it does not hide $\log W$ factors). All our algorithms are deterministic, whereas all lower bounds hold for randomized algorithms with constant success probability.

Problem	Pass	Approx.	Space	Stream	Notes
k-CAP	1	$2+\epsilon$	$O(\frac{n}{\epsilon}\log n)$	link arrival	Theorem 1
		$2-\epsilon$	$\Omega(n^2)$ bits		Theorem 10
		$O(t)$	$\tilde{O}(kn+n^{1+\frac{1}{t}})$	fully streaming	Theorem 15
			$\Omega(kn+n^{1+\frac{1}{t}})$ bits		Theorem 11
STAP	1	$O(t)$	$\tilde{O}(n^{1+\frac{1}{t}})$	fully streaming	Corollary 20
			$\Omega(n^{1+\frac{1}{t}})$ bits	link arrival	Corollary 21
Spanner	1	$O(t)$	$\tilde{O}(n^{1+\frac{1}{t}})$	edge arrival	Theorem 16
			$\Omega(n^{1+\frac{1}{t}})$ bits		Erdős' girth conjecture
SNDP	1	$O(t\log k)$	$\tilde{O}(kn^{1+\frac{1}{t}})$	edge arrival	Theorem 25
		$O(t)$	$\Omega(n^{1+\frac{1}{t}})$ bits		Corollary 21
k-ECSS	k	$O(\log k)$	$O(kn\log n)$	edge arrival	Corollary 26

Exploiting an even-odd bucketing approach, we provide a streaming algorithm with space complexity $O(n^{1+1/t}\cdot\log\min(W,n))$ words which by the well-known Erdős girth conjecture is basically the best one can hope for up to logarithmic factors. We further apply this even-odd bucketing to the k-CAP problem in the link arrival setting, and obtain a (more technical) algorithm (Theorem 1) with no dependence on $\log W$ in its space complexity.

Streaming SNDP. Finally, we describe an application of our results for designing the *first* one-pass streaming algorithms for the problem in insertion only edge arrival streams, where the edges of the input graph arrive in an arbitrary order stream.

In SNDP, given a graph $G=(V,E)$ with a weight function $w: E\to\{0,1,\ldots,W\}$ together with a *connectivity requirement* $r\colon V\times V\to\mathbb{Z}_{\geq 0}$, the goal is to find a minimum weight subgraph $H\subseteq G$ so that for every $s,t\in V$, H contains $r(st)$ edge-disjoint paths connecting s and t. A parameter of interest in SNDP is the maximum connectivity requirement $k=\max_{st} r(st)$. SNDP is a classic problem in combinatorial optimization and generalizes several well-studied problems such as MST, Steiner tree, k-edge connected spanning subgraph (k-ECSS), and k-CAP.

The fourth row of Table 1 shows our results for SNDP in edge arrival streams. In fact, our streaming algorithm works even for the more general problem of covering proper functions of the form $f: 2^V\to\{0,1,\ldots,k\}$ using the edges of G (see Section B.2 for more details). k-ECSS, which itself is a basic problem in discrete optimization, is a variant of SNDP in which for every $s,t\in V$, $r(st)=k$. As a straightforward application of our algorithm for k-CAP in link arrival streams, we get a k-pass, $O(\log k)$-approximation for k-ECSS using $O(kn\log n)$ words of space. (See last row of Table 1).

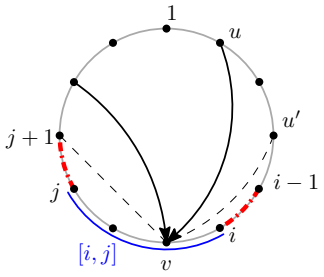
Unweighted variant. We remark that while we get tight algorithms for weighted k-CAP in both link arrival and fully streaming models, our lower bounds for link arrival does not hold for unweighted graphs. By a reduction from bipartite matching and invoking the result of [30], we observe a weaker lower bound that no streaming algorithm with $n\text{polylog}n$ space can achieve an approximation factor better than 1.409. Therefore, it remains an interesting open question to close the gap between 1.409 and 2 for unweighted k-CAP. Again, given that this lower bound is for tree augmentation, and the best known algorithm for (offline) TAP in unweighted graphs achieves an approximation factor of 1.326 [24], this again shows a gap between the two models for the problem in the unweighted variant.

1.3 Our Techniques

Given a streaming algorithm for the unweighted variants of both k-CAP and the spanner problems, an easy generalization to the weighted graphs is by partitioning the set of weights into $\log_{1+\epsilon} W$ number of classes and roughly running the unweighted sparsification on each class, resulting in $\epsilon^{-1} \log W$ blow up in the space usage. To remove the dependency on $\log W$ from the number of words, we follow an *even-odd bucketing* approach. More precisely, we partition the weights into much larger classes (i.e., *buckets*), such that the minimum and maximum weight in each class differ by $\text{poly}(n)/\epsilon$. This ensures that first, inside each class one can perform the weight-based partitioning to solve the problem while having only $\log n$ dependence in the space. Second, even picking all the edges from the $(i-2)$-th class E_{i-2} is cheaper than picking any edge in the i-th class E_i (i.e., it only introduces an extra $(1+\epsilon)$ multiplicative factor). This assumption allows us to infer additional properties about the graph once we are processing the edges in the class E_i, and shrink the problem significantly from each level E_{i-2} to E_i. Thus our algorithm proceeds by separating the sparsification for the even-indexed buckets E_{2i} and the odd-indexed buckets E_{2i-1}, and processes the buckets from smallest to largest weights.

Spanner. First, consider the spanner problem, and let $\mathcal{C} = \{C_1, \ldots, C_r\}$ be the set of connected components created by the edges from the classes upto E_{i-2}. The even-odd bucketing ensures that we only need to consider the edges from E_i that are between two different components of $\mathcal{C}$. Thus, we shrink each connected component into a super-node and use the standard spanner algorithm with weight-based partitioning on this reduced graph. Note that the space usage of the algorithm is proportional to the number of *super-nodes with non-zero degree*. However, all such super-nodes will merge into bigger components for the next bucket E_{i+2}. Therefore the space usage of the algorithm for processing E_i can be charged to the reduction in the number of super-nodes. Since the number of super-nodes starts from n and goes down to 1, the total space usage of the algorithm can be bounded as a function of n. Finally, we need to perform the above process in a streaming setting: As we receive more edges in the stream, the components in $\mathcal{C}_i$ change but it is easy to maintain all required information in a streaming fashion.

Link arrival k-CAP. Our algorithm for k-CAP is more involved. First, by standard results in the literature, the problem reduces to cycle augmentation: given a cycle C, the goal is to augment it with a subset of edges from L such that the resulting graph becomes 3-edge-connected. Let the nodes on the cycle be indexed 1 to n in this order with vertex 1 being called the *root*. Now every cut of size 2 corresponds to two edges on the cycle. We specify such a cut with the interval $[i, j]$ with $1 < i \leq j \leq n$ that does not include the root. The goal is to cover all such cuts specified by these intervals.



First, using known ideas from [35, 36], we present a simple streaming algorithm for the unweighted variant of the problem as follows. We replace every link uv by two directed links $\vec{uv}$ and $\vec{vu}$, (this is where the factor 2 in the approximation comes from), and we say that $\vec{uv}$ covers a cut $[i, j]$ if $v \in [i, j]$ and $u \notin [i, j]$. Now one can show that for $1 \leq u < u' < v$, it is always better to keep the edge $\vec{uv}$ than $\vec{u'v}$. Similarly, for $v < u' < u \leq n$, it is always better to keep the edge $\vec{uv}$ than $\vec{u'v}$. As a result, for each vertex, we keep at most two incoming edges. Therefore, the total space usage of the algorithm is only $O(n)$ in this case. Again this algorithm can be generalized to the weighted graphs using a weight-based partitioning, introducing a factor $\log W$.

To remove the dependency on $\log W$, again we consider the even-odd bucketing. This time, for each weight class E_i, we consider the 3-edge-connected components $C_1, \ldots, C_r$ formed by the edges in buckets upto E_{i-2}. Again using the even-odd bucketing plus the fact that the cycle is already 2-edge-connected, we can show that shrinking each of the 3-connected components into a super-node still works. The main challenge is that as opposed to the spanner setting, the problem on the super-node does not reduce to the same problem of cycle augmentation. This is because a single super-node does not necessarily span a consecutive set of vertices on the cycle. However, we note that in this case, the min-cuts on the cycle that do not fully include or fully exclude the vertices in a single super-node do not need to be considered. This allows us to reduce the space usage of the algorithm again to be proportional to the number of super-nodes and thus bound the total space usage of the algorithm as a function of n.

Fully streaming k-CAP. Our algorithm in this setting maintains two sketches. First, it keeps a k-connectivity certificate on the set of edges E using a folklore streaming algorithm that keeps k disjoint forests, which contains the information of all min-cuts of E that need to be augmented in k-CAP. Second, employing our results on weighted spanners, the algorithm maintains a spanner for the set of (weighted) links. This means that every link ℓ of weight/length w that we miss, can be replaced with a path of weight at most $O(t) \cdot w$, thus covering all the min-cuts originally covered by ℓ. We show that this is a near-optimal algorithm one can get in this setting.

Lower bounds. Most of our lower bounds are via simple reductions from the INDEX problem in a two-party communication model, where we embed the bit-string held by Alice into edges of a graph, where by asking augmentation queries, Bob is able to tell whether edge (u, v) exists in Alice's graph for any pair of vertices u, v. The most interesting one of our lower bounds (Theorem 12) shows that, in the fully streaming model, the space complexity for storing a spanner is essentially necessary. In the proof we let Alice hold a subgraph of a high-girth graph, and Bob wants to estimate the distance in this graph between u, v (which is sufficient for telling whether (u, v) is an edge, due to the high girth). Our proof reduces

this problem of estimating the distance between u, v to the problem of augmenting a chain with end points u, v into a 2-edge-connected graph. However, we also need rule out potential augmentation solutions that do not correspond to a uv-path.

Applications. Our algorithms for streaming connectivity augmentation also imply streaming algorithms for problems such as STAP, k-ECSS and SNDP. In particular, our one-pass algorithm for SNDP works by running k instances of our streaming spanner algorithm in parallel, which store k disjoint sparse subgraphs of the input graph that satisfy certain approximation guarantee. In particular, we show these k disjoint "spanner-like" objects forms a coreset for SNDP instances with maximum connectivity requirement at most k.[3] Our approach follows the augmentation framework of [51, 26] to show the existence of an approximately good solution using edges from these k sparse subgraphs.

1.4 Organization

In Section 2, we present our $(2+\epsilon)$-approximate algorithms for k-CAP in the link arrival model, and present a lower bound showing our approximation ratio is close to optimal. In Section 3, we study k-CAP in the fully streaming model, and present matching space lower bounds and upper bounds assuming our weighted spanner result. In Appendix A, we present our weighed spanner algorithm in the streaming model with better $\log W$ dependence. Finally in Appendix B we present further applications to other network design problems such as k-ECSS and SNDP. All missing proofs are deferred to the full version of the paper.

2 Connectivity Augmentation in Link Arrival Streams

In this section, we consider k-CAP, the problem of augmenting the connectivity of a given graph $G = (V, E)$ from $k-1$ to k using a subset of weighted links $L \subseteq \binom{V}{2}$ in link arrival streams. To recall, in the link arrival model, a cactus representation of the graph G, which is of size $O(n)$ (see Definition 2 for the formal definition of cactus), is given to us in advance and the set L arrives in the stream (see Section 1.1).

▶ **Theorem 1.** *The k-connectivity augmentation problem (k-CAP) on $(G = (V, E), L)$ in the link arrival model admits a one-pass $(2+\epsilon)$-approximation algorithm with total memory space $O(\frac{n}{\epsilon} \log \min(n, W))$ words, where $W = \max_{e \in E} w(e)$.*

Note that the augmentation set itself may have size $\Omega(n)$[4], so any algorithm that explicitly stores a solution must consume $\Omega(n)$ space. Moreover, we will show that just approximating the optimal total weight of the augmentation solution to any factor already requires $\Omega(n)$ bits of space. Hence, the space of our algorithm is tight up to a poly-logarithmic factor.

2.1 Preliminaries

Cactus representation of min-cuts. To increase the edge-connectivity of a $(k-1)$-connected graph G to k, we need to add links to *cover* all min-cuts of size $k-1$. That is, for each cut S of size $k-1$ (i.e., $|\delta_G(S)| = k-1$), we must add a link $e \in L$ such that $e \in \delta(S)$. Dinits, Karzanov, and Lomonosov [14] showed there is a compact representation of all min-cuts of an undirected graph by a *cactus graph*.

[3] In fact, the coreset guarantee holds even for the more general covering proper functions of the form $f : 2^V \to \{0, 1, \ldots, k\}$.

[4] As an example, consider a graph $G = (V, E)$ where $V = \{0, 1, \ldots, n-1\}$ and $E = \{(i, j) : j - i \in \{1, 2, \ldots, k\}\}$ (where indices are modulo n), which has edge connectivity $2k$. If the link set is $L = \{(i, i+1) : i \in [n]\}$, then at least $\lceil n/2 \rceil$ links are necessary to increase the edge connectivity by one.

▶ **Definition 2** (Cactus Graph). *A cactus graph is a 2-edge-connected graph $C = (V_C, E_C)$ where each edge in E_C belongs to exactly one simple cycle. Note that we allow cycles of length 1 or 2 too.*

▶ **Lemma 3** ([14]). *Let $G = (V, E)$ be an undirected graph. There is a loopless cactus $C = (V_C, E_C)$ of size at most $2n - 1$ and a mapping $\varphi : V \to V_C$ so that a subset $S \in V$ is a min-cut of G if and only if $\varphi(S)$ is a min-cut of C.*

Moreover, when the min-cut size of G is an odd integer, the cactus representation of G is a spanning tree (we may still treat it as a cactus by duplicating each tree edge).

The cactus representation is particularly useful for connectivity augmentation problems:

▶ **Corollary 4.** *Let $G = (V, E)$ be an undirected graph. Let C denote the cactus representation of min-cuts in G. Then a link $(u, v) \in E \setminus E_H$ crosses a min-cut S in G if and only if the corresponding link $(\varphi(u), \varphi(v))$ crosses $\varphi(S)$ in C.*

▶ Remark 5. We remark that there is a simple streaming algorithm for constructing the cactus representation with space complexity $\tilde{O}(kn)$: First, construct a k-connectivity certificate H of G (recall that a k-connectivity certificate for a graph G is a subgraph H of G that contains all edges crossing cuts of size k or less in G, and at least k edges from each cut of size more than k) with $O(kn)$ edges with space complexity $O(kn)$ words in polynomial time, using a simple algorithm by [41]. Then, we apply the algorithm of [34] for computing the cactus representation of the subgraph H in $\tilde{O}(|E(H)|) = \tilde{O}(kn)$ time and space. It is straightforward to verify that the constructed cactus is a cactus representation of G, given G is a $(k-1)$-connected graph.

We then get the following as a corollary of Theorem 1: If the algorithm receives a k-connectivity certificate as a representation of G or the edges of G arrive in the stream before any link arrives, we can construct a cactus representation of G in $O(kn)$ space first and then run our algorithm in this section for cactus augmentation and the overall space complexity will be $O(nk + \frac{n}{\epsilon} \log n)$.

Transforming cactus to cycle. In the (weighted) *cactus augmentation* problem, without loss of generality, we can assume the cactus is a single cycle. The latter problem is known as weighted *cycle augmentation*. To reduce an instance on a general cactus to the single cycle case (without losing approximation factor), we apply the technique observed in [23, 50]: Unfold the cactus into its Eulerian circuit, then add additional zero-weight edges (which we can use to augment at no cost) to connect the nodes corresponding to the same junction node in the cactus. See Section 3 in [50] for a detailed description.

▶ **Lemma 6** (Theorem 3 in [23]; see also Lemma 2.2 in [50]). *Let $\alpha > 1$. If there is an α-approximation algorithm for the weighted cycle augmentation problem, then the weighted cactus augmentation problem admits an α-approximation.*

Note that this reduction only produces $O(n)$ extra zero-weight edges, so it does not affect the space complexity of the streaming algorithm. We can apply the unfolding technique in the preprocessing step and in the rest of this section, we assume that the cactus is a single cycle.

2.2 Main Step: Cycle Augmentation in Link Arrival Streams

We arbitrarily assign a root node on the cycle, and let its index be 0. Then let the vertices of the cycle be $V = \{0, 1, \ldots, n-1\}$, with edges $C = \{e_1, e_2, \ldots, e_n\}$ where $e_i = (i-1, i)$ (with indices modulo n). We first describe a 2-approximation for the unweighted case, using an idea from [35, 36].

▶ **Theorem 7.** *There exists a one-pass 2-approximation algorithm for the cycle augmentation problem on unweighted graphs with total memory space $O(n)$ edges.*

Proof. Following [35, 36], we consider a directed version of the problem defined as follows: given a set E of *directed* edges, augment a minimum size subset $E' \subseteq E$ to the cycle, such that for every 2-cut $(L, V \setminus L)$ of the cycle where $0 \in V \setminus L$ (i.e., $L = \{l, l+1, \ldots, r\}$ for some $1 \leq l \leq r \leq n-1$), there exists $x\vec{y} \in E'$ with $y \in L$ and $x \in V \setminus L$ (we say $x\vec{y}$ covers L in this case). To reduce the original (undirected) cycle augmentation instance to this directed problem, simply replace each input edge (u, v) by two arcs $u\vec{v}, v\vec{u}$, incurring a 2-factor approximation: any directed solution $\{x\vec{y}\}$ implies an undirected solution $\{(x, y)\}$ of the same cost, and any undirected solution $\{(x, y)\}$ implies a directed solution $\{x\vec{y}\} \cup \{y\vec{x}\}$ of twice the cost.

Now we solve the directed instance exactly by an $O(n)$-space streaming algorithm. For each $v \in V$, we only need to keep the input arc $u\vec{v}$ with minimum indexed u, and keep the input arc $u\vec{v}$ with maximum indexed u. In this way we store only $O(n)$ arcs in total, and finally we run an offline exact algorithm (e.g., [22], which was also used by [36]) for the directed problem on these stored arcs. This does not affect optimality, because when $0 \leq u < u' < v$, any 2-cut, $U = \{l, l+1, \ldots, r\}$ covered by $u'\vec{v}$ is also covered by $u\vec{v}$, so we can discard $u'\vec{v}$ if we already have $u\vec{v}$ (a similar argument applies to the $v < u' < u \leq n-1$ case). ◀

By a simple scaling, this algorithm can be modified into a $(2+\epsilon)$-approximate algorithm for the weighted case with total space $O(\frac{n}{\epsilon} \log W)$ edges. Now we improve this $\log W$ dependency.

▶ **Theorem 8.** *The cycle augmentation problem on weighted graphs admits a one-pass $(2+\varepsilon)$-approximation streaming algorithm with total memory space $O(\frac{n}{\epsilon} \log \min(W, n))$ edges.*

Proof. We assume $\epsilon > 1/n$; otherwise use the trivial $O(n^2)$-space algorithm that stores the cheapest edge between every pair of vertices.

Define weight intervals $I_k = [(n/\epsilon)^k, (n/\epsilon)^{k+1})$. Let E_k be the set of input edges e that have arrived so far with weights $w(e) \in I_k$. Note that[5]

$$\frac{\min_{e \in E_{k+2}} w(e)}{\max_{e \in E_k} w(e)} > n/\epsilon. \tag{1}$$

These weight intervals do not contain zero, so we separately use a zero-weight class E_{-1} to hold edges of zero weight. But for notational simplicity, we will not specially mention this zero weight class in later description. One can check that this does not affect the correctness of the algorithm.

Recall C is the base cycle of length n. For each $k \in \{0, 1, \ldots, \lceil \log_{n/\epsilon} W \rceil\}$, define graph

$$G_k := C \cup \bigcup_{i \geq 0} E_{k-2i}. \tag{2}$$

Let Q_k denote the collection of 3-edge-connected components of G_k, which form a partition of the n vertices. See Figure 1 for an illustration. Let $1 \leq |Q_k| \leq n$ denote the number of components. Since $G_k \subseteq G_{k+2}$, Q_k refines Q_{k+2}, and $|Q_k| \geq |Q_{k+2}|$.

[5] This inequality is meaningful only if both E_k and E_{k+2} are nonempty. This issue does not affect our overall argument since our algorithm can simply ignore the empty weight classes.

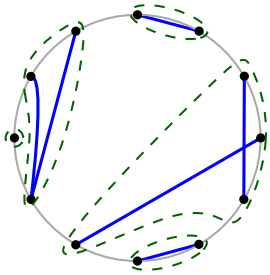


Figure 1 An example of 3-edge-connected components Q_k of the graph G_k. Thin black edges denote the base cycle, and thick blue edges denote the links from the set $\bigcup_{i\geq 0} E_{k-2i}$; together they form G_k. The dashed green lines describe the 3-edge-connected components of graph G_k.

Algorithm description. At any point, our streaming algorithm always stores a subset of the input edges $E = \bigcup_k E_k$, which includes the following:

1. **Undirected edges F_k:** We store edge subsets $F_k \subseteq E_k$, such that for all k the subgraph $C \cup \bigcup_{i\geq 0} F_{k-2i} \subseteq G_k$ has the same 3-edge-connected components as Q_k.
2. **Directed arcs S_k:** For each k and 3-edge-connected component $U \in Q_k$, and every weight interval $J_i = [(1+\epsilon)^i, (1+\epsilon)^{i+1}) \subseteq I_{k+2}$, we store the arc $x\vec{y}$ with minimum (and maximum) indexed x where $(x, y) \in E_{k+2}$, $y \in U, x \notin U$, and $w(x, y) \in J_i$. The set of these arcs is denoted by S_k.

Now we describe how to maintain this information when a new edge $(u, v) \in E_{k'}$ arrives.

- **Maintain Item 1:** Note that adding this edge could potentially cause the components in $Q_{k'+2i}$ $(i = 0, 1, 2, \dots)$ to merge. To maintain Item 1 (and hence the knowledge of all Q_k), we insert (u, v) into the current $F_{k'}$, and then run a clean up procedure to remove redundant edges: Start from the graph $H \leftarrow C \cup \bigcup_{j\geq 1} F_{k'-2j}$ which encodes the 3-connectivity information of the graph formed using edges prior to $E_{k'}$, and iterate over the edges $e \in F_{k'+2i}$ (in increasing order of $i = 0, 1, 2, \dots$). If adding e to H does not change the 3-edge-connected components of H, then remove e from $F_{k'+2i}$. Otherwise add e to H. It is clear that this clean up procedure preserves all the 3-connectivity information, since we start from the base graph C which is already 2-edge-connected.
- **Maintain Item 2:** To maintain Item 2, we simply use arcs $u\vec{v}$ and $v\vec{u}$ to replace the existing ones that become dominated. When two 3-edge-connected components $U, U' \in Q_k$ merge, we also merge the stored information for U, U' (compare the best arcs stored for these two components and keep the better one).
- **Offline step:** In the end, we run an offline exact algorithm (such as [22]) that solves the directed problem (see proof of Theorem 7) on the stored arcs in Item 2 and directed versions of the stored edges in Item 1.

Space complexity. For Item 1 the total space is $\sum_k |F_k| = \sum_j |F_{2j}| + \sum_j |F_{2j+1}|$ edges. We bound both terms separately. Due to our clean up procedure, there should be no redundant edges in $F_{\text{even}} = \bigcup_j F_{2j}$: starting from the base cycle $H \leftarrow C$, we can iterate over the edges $e \in F_{\text{even}}$ in certain order so that adding edge e to H always strictly decreases the number of 3-edge-connected components of H. Hence $|F_{\text{even}}| \leq n - 1$, and similarly $|F_{\text{odd}}| \leq n - 1$, so

$$\sum_k |F_k| \leq 2(n-1). \tag{3}$$

Define c_{k+2} to be the number of 3-edge-connected components $U \in Q_k$ for which there exists $(u,v) \in E_{k+2}$ with $u \in U$ and $v \notin U$. Then the space for Item 2 is $\sum_k 2c_{k+2} \cdot \log_{1+\epsilon} \frac{\max_{e\in E_{k+2}} w(e)}{\min_{e\in E_{k+2}} w(e)} \leq \sum_k c_{k+2} \cdot O(\log(n/\epsilon)/\epsilon) \leq \sum_k c_{k+2} \cdot O(\log(n)/\epsilon)$ edges. We need the following lemma.

▶ **Lemma 9.** $c_{k+2} \leq 2(|Q_k| - |Q_{k+2}|)$.

Using this lemma, the space complexity for Item 2 is

$$\begin{aligned} O(\frac{\log n}{\epsilon}) \sum_k c_{k+2} &\leq O(\frac{\log n}{\epsilon}) \sum_k (|Q_k| - |Q_{k+2}|) \\ &\leq O(\frac{\log n}{\epsilon}) \cdot 2(n-1) \\ &\leq O(\frac{n \log n}{\epsilon}) \qquad \triangleright \text{ by summing over even and odd } k \text{ separately} \end{aligned}$$

So the total space complexity is $O(\epsilon^{-1} n \log n)$ edges.

Proof of Lemma 9. We first shrink the graph $G_k = C \cup \bigcup_{i\geq 0} E_{k-2i}$ into graph H_k. Let each node of H_k represent a 3-edge-connected component $U \in Q_k$, and for every $(u,v) \in C$ (recall C is the set of edges on the base cycle) with $u \in U \in Q_k$ and $v \in V \in Q_k$, we connect U, V in H_k by an edge (allowing self-loops and parallel edges). As a standard fact, H_k is a cactus (allowing self loops), and C corresponds to an Eulerian circuit of H_k.

Let $\mathcal{C}_k$ denote the collection of simple cycles (cycles with distinct vertices; we view a self loop as a simple cycle as well) of the cactus H_k. Then $\mathcal{C}_k$ can be viewed as a partition of the n edges on the base cycle C, where $e, e' \in C$ belong to the same partition if and only if $\{e, e'\}$ is a 2-cut of G_k. Observe that $|\mathcal{C}_k| = n + 1 - |Q_k|$.[6] Hence, in the following it suffices to prove $c_{k+2} \leq 2 \cdot (|\mathcal{C}_{k+2}| - |\mathcal{C}_k|)$. Note that $\mathcal{C}_{k+2}$ is a finer partition of C than $\mathcal{C}_k$.

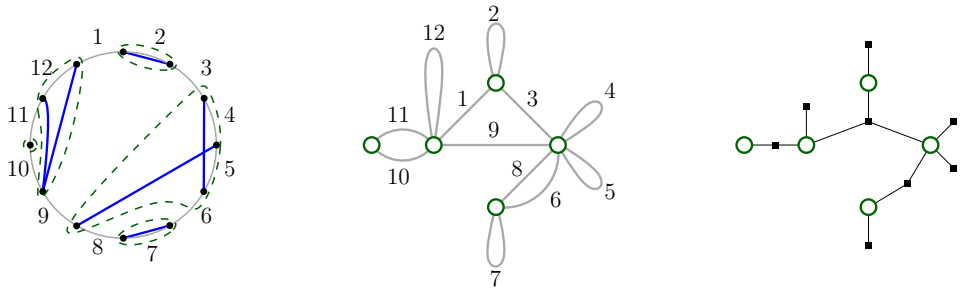


Figure 2 A picture of the cactus H_k (middle) produced by shrinking G_k (left). The tree T_k (right) is produced from cactus H_k.

We now consider how adding edges E_{k+2} into G_k can refine $\mathcal{C}_k$. Convert the cactus H_k into a tree T_k as follows. Let T_k be a bipartite graph with vertex bipartition $(\mathcal{C}_k, Q_k)$, in which $D \in \mathcal{C}_k$ is connected to every $U \in Q_k$ that lies on the simple cycle D in cactus H_k. Observe this bipartite graph T_k is indeed a tree. For each $(u,v) \in E_{k+2}$, let $u \in U \in Q_k$ and $v \in V \in Q_k$, and we mark all the tree-edges on the unique path connecting U and V in T_k.

[6] To see this equality, consider removing one arbitrary edge from each simple cycle of the cactus, and the remaining edges should form a tree. As there are $|Q_k|$ vertices, the number of edges in the remaining tree is $|Q_k| - 1$, so the number of edges n in the original cactus equals $|Q_k| - 1 + |\mathcal{C}_k|$, since we removed $|\mathcal{C}_k|$ edges in the removal step.

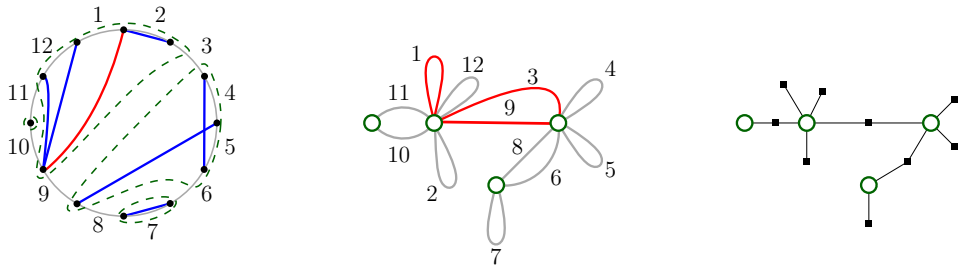


Figure 3 After adding an edge from E_{k+2} (depicted in red), the partition $\mathcal{C}_{k+2}$ refines the old partition $\mathcal{C}_k$: $\{1,3,9\}$ breaks into $\{1\}$ and $\{3,9\}$.

For each $D \in \mathcal{C}_k$, let $d(D)$ denote the number of marked tree-edges incident to the tree-node D in T_k. Then, observe that $d(D) \in \{0\} \cup \{2,3,4,\dots\}$, and $D \in \mathcal{C}_k$ (viewed as a subset of C) breaks into $\max\{d(D),1\}$ subsets in the partition $\mathcal{C}_{k+2}$.

By assumption, there are at least c_{k+2} many tree-nodes $U \in Q_k$ that are incident to at least one marked tree-edge in T_k, so T_k contains at least c_{k+2} marked tree-edges. Hence,

$$\begin{aligned}|\mathcal{C}_{k+2}| = \sum_{D\in\mathcal{C}_k} \max\{d(D),1\} &\geq \sum_{D\in\mathcal{C}_k} (1+d(D)/2) \quad \triangleright \text{since } d(D) \in \{0\}\cup\{2,3,4,\dots\} \\ &= |\mathcal{C}_k| + \frac{1}{2}\sum_{D\in\mathcal{C}_k} d(D) \geq |\mathcal{C}_k| + \frac{1}{2}c_{k+2},\end{aligned}$$

which completes the proof. ◀

Approximation factor. Let $\mathrm{OPT} \subseteq E = \bigcup_{k=-\infty}^{+\infty} E_k$ denote the optimal solution for the (undirected) cycle augmentation problem. Let k^* be the maximum k^* such that $\mathrm{OPT} \cap E_{k^*} \neq \varnothing$. Then by (1) we have

$$w(\mathrm{OPT}) > (n/\epsilon) \cdot \max_{e\in E_{k^*-2}} w(e). \tag{4}$$

Bidirecting OPT gives a solution OPT' for the directed problem with total cost $w(\mathrm{OPT}') = 2w(\mathrm{OPT})$. In the following we convert OPT' into a solution SOL for the directed problem that only uses arcs stored by the streaming algorithm, with total cost $w(\mathrm{SOL}) \leq (1 + O(\epsilon))w(\mathrm{OPT}') \leq (2 + O(\epsilon))w(\mathrm{OPT})$. This establishes that our streaming algorithm achieves $2 + O(\epsilon)$ approximation ratio for the (undirected) cycle augmentation problem.

In SOL we first include both directed versions of all $(u,v) \in \bigcup_{k\leq k^*-2} F_k$, with total cost at most

$$\begin{aligned}\sum_{k\leq k^*-2} 2|F_k| \cdot \max_{e\in F_k} w(e) &\leq \sum_k 2|F_k| \cdot \max_{e\in E_{k^*-2}} w(e) \\ &\leq 4(n-1)\cdot\frac{\epsilon}{n} w(\mathrm{OPT}) \qquad \triangleright \text{by (3) and (4)} \\ &\leq 4\epsilon w(\mathrm{OPT}).\end{aligned}$$

Then, for every arc $x\vec{y} \in \mathrm{OPT}'$ with weight $w(x\vec{y}) \in I_k$ where $k \in \{k^*-1, k^*\}$, we will find a replacement arc $x'\vec{y}' \in S_k$ stored by Item 2: Let $y \in U \in Q_k$. If $x \notin U$, then by Item 2 we can pick a stored arc $x'\vec{y}' \in S_k$ with $y' \in U$ and $w(x'\vec{y}') < (1+\epsilon)w(x\vec{y})$, such that $x' \leq x$ (if $x < y$) or $x' \geq x$ (if $x > y$). We include $x'\vec{y}'$ in SOL. (in the case of $x \in U$ we do not need to do anything)

By definition we immediately have $w(\text{SOL}) \leq 4\epsilon w(\text{OPT}) + (1+\epsilon)w(\text{OPT}') = (2 + 6\epsilon)w(\text{OPT})$. To show SOL is a feasible solution for the directed problem, we verify that each 2-cut $L = \{l, l+1, \ldots, r\}$ (where $1 \leq l \leq r \leq n-1$) is covered. There are three cases:

- **Case 1: $(L, V \setminus L)$ is not a 2-cut of G_{k^*-2}.** By Item 1, G_{k^*-2} and $C \cup \bigcup_{i \geq 0} F_{k^*-2-2i}$ have the same 3-edge-connected components, and hence have the same 2-cuts, so $(L, V \setminus L)$ is also not a 2-cut of $C \cup \bigcup_{i \geq 0} F_{k^*-2-2i}$. Hence, there exists $(u', v') \in \bigcup_{i \geq 0} F_{k^*-2-2i}$ such that $u' \in V \setminus L, v' \in L$. Then, $\vec{u'v'}$ covers L, and by construction we have $\vec{u'v'} \in \text{SOL}$.
- **Case 2: $(L, V \setminus L)$ is not a 2-cut of G_{k^*-3}.** This case is similar to case 1.
- **Case 3: Otherwise.** In this case, $(L, V \setminus L)$ is a 2-cut of both G_{k^*-2} and G_{k^*-3}. From the feasibility of OPT, we know there must exist arc $\vec{xy} \in \text{OPT}'$ that covers L (i.e., $y \in L, x \in V \setminus L$) with weight $w(\vec{xy}) \in I_k$ where $k \in \{k^*-1, k^*\}$. Let $y \in U \in Q_{k-2}$. Since $(L, V \setminus L)$ is a 2-cut of G_{k-2}, we know x, y cannot be in the same 3-edge-connected component of G_{k-2}, so $x \notin U$. Now let $\vec{x'y'} \in \text{SOL}$ be the replacement arc we found for $\vec{xy}$. By definition, $y' \in U$. We consider the case of $x < y$ (the other case $y < x$ is similar), and hence $x' \leq x$. In this case we must have $x < l \leq y \leq r$, so $x' < l$ and hence $x' \notin L$. Suppose for contradiction that $\vec{x'y'}$ does not cover L. Then we must have $y' \notin L$. But this would mean $(L, V \setminus L)$ is a 2-cut in G_{k-2} separating y and y', contradicting the assumption that $y, y' \in U$ belong to the same 3-edge-connected component of G_{k-2}. This proves that the replacement arc $\vec{x'y'} \in \text{SOL}$ indeed covers L. ◀

The following shows that the approximation factor of our algorithm is close to optimal.

▶ **Theorem 10.** *Any streaming algorithm that solves the weighted TAP in the link arrival model with better than 2-approximation needs $\Omega(n^2)$ bits of space.*

3 Connectivity Augmentation in the Fully Streaming Setting

In this section, we first prove a space lower bound for k-CAP in the fully streaming model. Then, we show a streaming algorithm with nearly matching space complexity.

3.1 Lowerbound for Estimating Connectivity Augmentation Cost

Our main lower bound statement is the following.

▶ **Theorem 11.** *For any constant integer $t \geq 1$, the (unweighted) k-CAP (even when k is known) in the fully streaming model requires space complexity $\Omega(kn + n^{1+1/t})$ bits (assuming the Erdős's girth conjecture) to approximate the solution size to a factor better than $2t+1$.*

It follows from combining two lower bound results Theorem 12 and Theorem 13.

Lower bound in terms of approximation factor (t). We first describe the space lower bound in terms of the approximation factor. As is standard in the spanner literature, the proof is based on high-girth graphs, but here we need to be more careful to make the connection between tree-augmentation and shortest paths.

▶ **Theorem 12.** *Consider the (unweighted) TAP where E is the base tree and L is the set of edges to augment, and $E \cup L$ arrive as a stream in an arbitrary order.*

For any constant integer $t \geq 1$, any (randomized) streaming algorithm $\mathcal{A}$ that can output the size of a better than $(2t+1)$-approximate solution requires $\Omega(\gamma(n, 2t+1))$ bits of space, where $\gamma(n, 2t+1)$ denotes the maximum possible number of edges in an n–vertex graph with girth $> 2t+1$.

We remark that the same lower bound of Theorem 12 also generalizes to k-CAP for higher values of $k > 2$, provided that we allow the base graph E to have parallel edges.

Lower bound in terms of connectivity parameter (k). Zelke [53] gave a simple proof that computing the size of the minimum cut of an (unweighted) undirected graph requires $\Omega(n^2)$ bits of space for any one-pass streaming algorithm. In Zelke's construction the input graph has minimum cut size as large as $\Theta(n)$. Here we observe that Zelke's proof can be adapted to graphs with minimum cut size $\Theta(k)$, and show lower bounds for the connectivity augmentation problem.

We remark that [47] also obtained an $\Omega(kn)$-bit randomized lower bound and an $\Omega(kn \log n)$-bit deterministic lower bound for the k-CAP using a different proof.

▶ **Theorem 13.** *The k-CAP (where k is known) in the fully streaming model (with unweighted links) requires $\Omega(nk)$ bits of space to approximate to any finite factor.*

▶ Remark 14. We remark that the same $\Omega(nk)$ lower bound also holds for the task of constructing a cactus representation of a graph (Lemma 3), even assuming the edge connectivity value k is known. This is because the cactus representation immediately allows to distinguish between the cases of having two minimum cuts C_1, C_2 or one minimum cut C_2, and thus the proof above still applies.

3.2 Tight Algorithm

The main result of this section, whose details are deferred to the full version of the paper, is a single-pass algorithm that outputs a $(2t-1+\epsilon)$-approximate solution in $O(nk+\epsilon^{-1}n^{1+1/t}\log n)$ space, nearly matching the lower bounds of Theorem 12 and 13.

▶ **Theorem 15.** *The k-CAP in the fully-streaming model can be solved by a single-pass streaming algorithm with approximation ratio $(2t-1+\epsilon)$ in $O(nk+\epsilon^{-1}n^{1+1/t}\log n)$ space.*

References

1 Abu Reyan Ahmed, Greg Bodwin, Faryad Darabi Sahneh, Keaton Hamm, Mohammad Javad Latifi Jebelli, Stephen G. Kobourov, and Richard Spence. Graph spanners: A tutorial review. *Comput. Sci. Rev.*, 37:100253, 2020.

2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Symposium on Principles of Database Systems*, pages 5–14, 2012.

3 Sepehr Assadi and Aditi Dudeja. A simple semi-streaming algorithm for global minimum cuts. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 172–180. SIAM, 2021.

4 Sepehr Assadi, Sanjeev Khanna, and Yang Li. On estimating maximum matching size in graph streams. In *Proceedings of the Symposium on Discrete Algorithms*, pages 1723–1742, 2017.

5 Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1345–1364, 2016.

6 Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the international conference on Knowledge discovery and data mining*, pages 671–680, 2014.

7 Surender Baswana. Streaming algorithm for graph spanners–single pass and constant processing time per edge. *Inf. Process. Lett.*, 106(3):110–114, 2008.

8 Jarosław Byrka, Fabrizio Grandoni, and Afrouz Jabal Ameli. Breaching the 2-approximation barrier for connectivity augmentation: a reduction to steiner tree. In *Symposium on Theory of Computing*, pages 815–825, 2020.

9 Federica Cecchetto, Vera Traub, and Rico Zenklusen. Bridging the gap between tree and connectivity augmentation: unified and stronger approaches. In *Symposium on Theory of Computing*, pages 370–383, 2021.

10 Chandra Chekuri, Alina Ene, and Ali Vakilian. Prize-collecting survivable network design in node-weighted graphs. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 98–109, 2012.

11 Chandra Chekuri, Alina Ene, and Ali Vakilian. Node-weighted network design in planar and minor-closed families of graphs. *ACM Transactions on Algorithms (TALG)*, 17(2):1–25, 2021.

12 Joseph Cheriyan and László A Végh. Approximating minimum-cost k-node connected subgraphs via independence-free graphs. *SIAM Journal on Computing*, 43(4):1342–1362, 2014.

13 Michael S. Crouch, Andrew McGregor, and Daniel M. Stubbs. Dynamic graphs in the sliding-window model. In *Algorithms - ESA 2013 - 21st Annual European Symposium*, volume 8125 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2013.

14 E A Dinits, Alexander V Karzanov, and Micael V Lomonosov. On the structure of a family of minimal weighted cuts in a graph. *Studies in Discrete Optimization*, pages 290–306, 1973.

15 Michael Elkin. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. *ACM Trans. Algorithms*, 7(2):20:1–20:17, 2011.

16 Guy Even, Jon Feldman, Guy Kortsarz, and Zeev Nutov. A 1.8 approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2. *Transactions on Algorithms (TALG)*, 5(2):1–17, 2009.

17 Jittat Fakcharoenphol and Bundit Laekhanukit. An $O(\log^2 k)$-approximation algorithm for the k-vertex connected spanning subgraph problem. In *Proceedings of the Symposium on Theory of Computing*, pages 153–158, 2008.

18 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. In *International Colloquium on Automata, Languages, and Programming*, pages 531–543, 2004.

19 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the data-stream model. *Journal on Computing*, 38(5):1709–1727, 2008.

20 Manuel Fernández V, David P Woodruff, and Taisuke Yasuda. Graph spanners in the message-passing model. In *Innovations in Theoretical Computer Science Conference*, 2020.

21 Arnold Filtser, Michael Kapralov, and Navid Nouri. Graph spanners by sketching in dynamic streams and the simultaneous communication model. In *Proceedings of the Symposium on Discrete Algorithms (SODA)*, pages 1894–1913, 2021.

22 Harold N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *J. Comput. Syst. Sci.*, 50(2):259–273, 1995.

23 Waldo Gálvez, Fabrizio Grandoni, Afrouz Jabal Ameli, and Krzysztof Sornat. On the cycle augmentation problem: hardness and approximation algorithms. *Theory of Computing Systems*, 65:985–1008, 2021.

24 Mohit Garg, Fabrizio Grandoni, and Afrouz Jabal Ameli. Improved approximation for two-edge-connectivity. In *Symposium on Discrete Algorithms (SODA)*, pages 2368–2410, 2023.

25 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In *Symposium on Discrete Algorithms (SODA)*, pages 468–485, 2012.

26 MX Goemans, AV Goldberg, S Plotkin, DB Shmoys, É Tardos, and DP Williamson. Improved approximation algorithms for network design problems. In *Symposium on Discrete Algorithms (SODA)*, pages 223–232, 1994.

27 Fabrizio Grandoni, Christos Kalaitzis, and Rico Zenklusen. Improved approximation for tree augmentation: saving by rewiring. In *Symposium on Theory of Computing*, pages 632–645, 2018.

28 Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. *Algorithmica*, 76:654–683, 2016.

29 K. Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

30 Michael Kapralov. Space lower bounds for approximating maximum matching in the edge arrival model. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1874–1893, 2021.

31 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. *SIAM J. Comput.*, 46(1):456–477, 2017.

32 Michael Kapralov, Aida Mousavifar, Cameron Musco, Christopher Musco, Navid Nouri, Aaron Sidford, and Jakab Tardos. Fast and space efficient spectral sparsification in dynamic streams. In *Proceedings of the Symposium on Discrete Algorithms (SODA)*, pages 1814–1833, 2020.

33 Michael Kapralov and David Woodruff. Spanners and sparsifiers in dynamic streams. In *Proceedings of the Symposium on Principles of Distributed Computing*, pages 272–281, 2014.

34 David R. Karger and Debmalya Panigrahi. A near-linear time algorithm for constructing a cactus representation of minimum cuts. In Claire Mathieu, editor, *Proceedings of the Symposium on Discrete Algorithms (SODA)*, pages 246–255, 2009.

35 Samir Khuller and Ramakrishna Thurimella. Approximation algorithms for graph augmentation. *Journal of Algorithms*, 14(2):214–225, 1993.

36 Samir Khuller and Uzi Vishkin. Biconnectivity approximations and graph carvings. *J. ACM*, 41(2):214–235, 1994.

37 Guy Kortsarz and Zeev Nutov. Approximating k-node connected subgraphs via critical graphs. *SIAM Journal on Computing*, 35(1):247–257, 2005.

38 Guy Kortsarz and Zeev Nutov. A simplified 1.5-approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2. *Transactions on Algorithms*, 12(2):1–20, 2015.

39 Andrew McGregor. Finding graph matchings in data streams. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 170–181. Springer, 2005.

40 Hiroshi Nagamochi. An approximation for finding a smallest 2-edge-connected subgraph containing a specified spanning tree. *Discrete Applied Mathematics*, 126(1):83–113, 2003.

41 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992.

42 Jelani Nelson and Huacheng Yu. Optimal lower bounds for distributed and streaming spanning forest computation. In *Proceedings of the Symposium on Discrete Algorithms*, pages 1844–1860, 2019.

43 Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavifar, and Ola Svensson. Beyond 1/2-approximation for submodular maximization on massive data streams. In *International Conference on Machine Learning*, pages 3829–3838, 2018.

44 Zeev Nutov. Approximating steiner networks with node-weights. *SIAM Journal on Computing*, 39(7):3001–3022, 2010.

45 R Ravi, Weizhong Zhang, and Michael Zlatin. Approximation algorithms for steiner tree augmentation problems. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2429–2448, 2023.

46 Aviad Rubinstein, Tselil Schramm, and S. Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In *9th Innovations in Theoretical Computer Science Conference, ITCS*, volume 94 of *LIPIcs*, pages 39:1–39:16, 2018.

47 Xiaoming Sun and David P Woodruff. Tight bounds for graph problems in insertion streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.

48 Vera Traub and Rico Zenklusen. A better-than-2 approximation for weighted tree augmentation. In *Foundations of Computer Science (FOCS)*, pages 1–12, 2022.

49 Vera Traub and Rico Zenklusen. Local search for weighted tree augmentation and steiner tree. In *Symposium on Discrete Algorithms (SODA)*, pages 3253–3272, 2022.

50 Vera Traub and Rico Zenklusen. A $(1.5+\varepsilon)$-approximation algorithm for weighted connectivity augmentation. In *Symposium on Theory of Computing*, pages 1820–1833, 2023.

51 David P Williamson, Michel X Goemans, Milena Mihail, and Vijay V Vazirani. A primal-dual approximation algorithm for generalized steiner network problems. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 708–717, 1993.

52 Mariano Zelke. k-connectivity in the semi-streaming model. *arXiv preprint cs/0608066*, 2006. arXiv:cs/0608066.

53 Mariano Zelke. Intractability of min-and max-cut in streaming graphs. *Information Processing Letters*, 111(3):145–150, 2011.

A Streaming Algorithm for Spanners on Weighted Graphs

In this section, we prove the following theorem on computing spanners for weighted graphs in the streaming model.

▶ **Theorem 16.** *For any integer $t \geq 1$, there is a one-pass streaming algorithm for computing a $(2t-1+\epsilon)$-spanner of size $O(\epsilon^{-1}n^{1+1/t}\log n)$ of a weighted graph, with space complexity $O(\epsilon^{-1}n^{1+1/t}\log n)$ words.*

Let $G=(V,E)$ be a weighted graph. We denote the weight function by $w : E \to \mathbb{R}^+$. Moreover, We normalize the weights so that $w(e) \in \{0\} \cup [1, W]$. For each $j \in [0, \lceil \log_{1+\epsilon} W \rceil]$, our algorithm stores E_j, a subset of edges of G that have weights in $[(1+\epsilon)^j, (1+\epsilon)^{j+1})$. (These intervals do not contain zero, so we separately use a zero-weight class E_{-1} to hold edges of zero weight. For notational simplicity, we will not mention this zero weight class in later description. One can check that this does not affect the correctness of the algorithm.)

Our algorithm is as follows (see Algorithm 2). As an edge e arrives, round its weight to the nearest power of $(1+\epsilon)$ and place it in the corresponding weight class E_j. As usual, we keep the edge e iff it does not close a cycle of length at most $2t$ in E_j, for some given parameter t. After processing the edge, we run the Sparsify subroutine described below in Algorithm 1.

Sparsify subroutine. Let $C > 0$ be a sufficiently large constant. Define intervals $I_k = [k \cdot (C/\epsilon)\log n, (k+1)\cdot(C/\epsilon)\log n]$. For all k let $\tilde{E}_k := \bigcup_{j \in I_k} E_j$. For each k let $E^{\text{even}}_{\leq k} = \bigcup_{j=-\infty}^{k} \tilde{E}_{2j}$ and $E^{\text{odd}}_{\leq k} = \bigcup_{j=-\infty}^{k} \tilde{E}_{2j+1}$. Let $E^{\text{even}} = \bigcup_j \tilde{E}_{2j}$, and we define E^{odd} similarly. Our Sparsify procedure operates independently on these two sets. We will ensure that each set contains $O(\epsilon^{-1}n^{1+1/t}\log n)$ edges, independent of the weight bound W. We now describe how Sparsify operates on E^{even} (the operations are the same for E^{odd}).

▷ Claim 17. Let the constant C in the definition of the sets $\tilde{E}_k$ be chosen sufficiently large. Let k be an integer. Let $H = (V, E^{\text{even}}_{\leq k-1})$. Then for any edge $e=(u,v) \in \tilde{E}_{2k}$ such that u and v belong to the same connected component in H, one has $w_e \geq \text{dist}_H(u,v)$.

Our procedure Sparsify(k) performs the following step for each k from $k_{\max}$ down to $k_{\min}$. Collapse the connected components induced by $E^{\text{even}}_{\leq k-1}$ into supernodes, and consider the multigraph with edges $\tilde{E}_{2k}$ on this set of supernodes. We convert this multigraph into a simple graph in the following natural way. For each edge $e=(u,v) \in \tilde{E}_{2k}$,

- delete e if it is a self loop in this graph (i.e. u, v belong to the same connected component)
- delete e if there is a shorter edge that is parallel to e.

This is summarized in Algorithm 1. The algorithm is summarized in Algorithm 2.

▶ **Lemma 18.** *The edges stored by Algorithm 2 form a $(2t-1)\cdot(1+\epsilon)$-spanner of G.*

▶ **Lemma 19.** *Throughout the algorithm, the total number of edges stored by Algorithm 2 is always at most $O(\epsilon^{-1}n^{1+1/t}\log n)$.*

Algorithm 1 SPARSIFY.

```
1: procedure SPARSIFY
2:     for k = k_max down to k_min do    ▷ The same procedure for the set E^odd
3:         Let H = (V, E^even_{≤k−1}) and let C_1, ..., C_r be the connected components of H.
4:         for e = (u, v) ∈ Ẽ_{2k} do
5:             if u, v ∈ C_i for some i then
6:                 delete e from Ẽ_{2k}
7:             end if
8:             if ∃(u', v') ∈ Ẽ_{2k} s.t. w_{(u',v')} ≤ w_e, and u, u' ∈ C_i, v, v' ∈ C_j for some i, j
   then
9:                 delete e from Ẽ_{2k}
10:            end if
11:        end for
12:    end for
13: end procedure
```

Algorithm 2 Overall algorithm.

```
1: procedure SPANNER
2:     for each edge e = (u, v) in the stream do
3:         Round weight of e to power of 1 + ε. Let j be the weight class of e.
4:         Add e to E_j iff dist_{E_j}(u, v) > (2t − 1) · w_e.
5:         Call SPARSIFY
6:     end for
7: end procedure
```

B Further Applications of Streaming Connectivity Augmentation

In this section, we show applications of our streaming algorithms for k-CAP for following well-studied *network design* problems: STAP, SNDP and k-ECSS.

B.1 Steiner Tree Augmentation Problem (STAP) in Streaming

In STAP, we are given a set of vertices V partitioned into *terminal* nodes (R) and *Steiner* nodes ($V \setminus R$), and a Steiner tree T spanning the terminal set R. Then given a set of weighted links $L \subseteq \binom{V}{2}$, the goal is to find a minimum weight set of links $S \subseteq L$ such that $H = (V, E(T) \cup S)$ has 2 edge-disjoint paths between any pair of terminals. The problem is a special case of SNDP and can be approximate within a factor of 2 by iterative rounding method of Jain [29]. In light of recent developments for approximating tree augmentation and connectivity augmentation problems [49], Ravi, Zhang, and Zlatin [45] provided a $(1.5 + \epsilon)$-approximation for Steiner tree augmentation problem in polynomial time.

Algorithm in fully streaming setting. First, we observe that our results imply an algorithm for STAP in the fully streaming setting.

▶ **Corollary 20.** *STAP in the fully streaming model can be solved by a single-pass streaming algorithm with approximation ratio $(2t-1+\epsilon)$ and space complexity $O(\epsilon^{-1}n^{1+1/t}\log n)$ words.*

Note that the same fully streaming algorithm from Corollary 20 can also be used to solve STAP in the easier link arrival streams.

Lower bound in link arrival streams. Now we show that STAP has a lower bound nearly matching Corollary 20 *even in link arrival streams*. This shows a separation of STAP from the easier TAP: the latter problem has a better streaming algorithm in link arrival streams than in the fully streaming setting, whereas the former problem does not.

▶ **Corollary 21.** *For any constant integer $t \geq 1$, weighted STAP in link arrival streams requires space complexity $\Omega(n^{1+1/t})$ bits (assuming the Erdős's girth conjecture) to approximate the solution cost to a factor better than $2t+1$.*

B.2 SNDP in Edge Arrival Streams

In this section, using our results and techniques from k-CAP and weighted spanners, we present a streaming algorithm for the general SNDP problem in edge arrival streams. We remark that our result in this section provide coresets for *covering functions* defined on cuts.

▶ **Lemma 22.** *Consider a weighted graph $G = (V, E)$ in an edge arrival stream. For integer $k \geq 1$ there is a one-pass streaming algorithm that computes k disjoint edge subsets $S_1 \uplus S_2 \uplus \cdots \uplus S_k \subseteq E$ each of size $|S_i| \leq O(\epsilon^{-1} n^{1+1/t} \log n)$, in total space $O(k\epsilon^{-1} n^{1+1/t} \log n)$ words such that, for every $i \in [k]$ and every $e = (u, v) \in E \setminus (S_1 \cup S_2 \cup \cdots \cup S_i)$, there is be a path $P \subseteq S_i$ connecting u, v with total length $w(P) \leq (2t - 1 + \epsilon)w(e)$.*

One of the main algorithmic approaches for SNDP is the augmentation framework pioneered by [51]. In this approach, the solution is constructed in k phases and by the end of the phase ℓ, the connectivity of every pair u, v in the so-far-constructed solution is at least $\min\{\ell, r(st)\}$. So, the optimization problem of each phase is to increase connectivity of subset of pairs by one. More precisely, in each phase ℓ, we need to pick a minimum-weight subgraph H to cover a function $f_\ell : 2^V \to \{0, 1\}$. We say that a subgraph H covers f iff for every $U \subset V$, $\delta_H(U) \geq f(s)$. In the case of SNDP, for every $\ell \leq k$, f_ℓ is a skew-supermodular function and admits a 2-approximation via a primal-dual algorithm [51].

Next, We use Lemma 22 to show a coreset for covering $\{0, 1\}$ functions $f : 2^V \to \{0, 1\}$:

▶ **Definition 23.** *Given a weighted graph $G = (V, E)$, and a function $f : 2^V \to \{0, 1, \ldots, k\}$, find an edge subset $H \subseteq E$ with minimum total weight such that for all $U \subseteq V$ it holds that $|\delta_H(U)| \geq f(U)$. Throughout this section, we consider the functions f arising from an instance of SNDP on G with connectivity requirement function r with maximum requirement k. Then, for every $U \subset V$, $f(U) := \max_{s \in U, t \in V \setminus U} r(st)$.*[7]

▶ **Lemma 24.** *Given a weighted graph $G = (V, E)$, let $S = S_1 \cup \cdots \cup S_k$ be the set of edges returned by the algorithm of Lemma 22. Then, the optimal solution for covering a function $f : 2^V \to \{0, 1, \cdots, k\}$ (arising from a SNDP instance on G) on graph $G' = (V, S)$ is an $O(t \log k)$-approximation of the optimal solution for covering f on $G = (V, E)$.*

▶ **Theorem 25.** *SNDP with maximum connectivity requirement k on a weighted graph $G = (V, E)$ admits a single-pass streaming algorithm with space complexity $O(kn^{1+1/t})$ words and approximation ratio $O(t \log k)$.*

Note that SNDP generalizes STAP, so the same lower bound for STAP from Corollary 21 also applies to SNDP. Specifically, for any constant integer $t \geq 1$, weighted SNDP requires space complexity $\Omega(n^{1+1/t})$ bits (assuming the Erdős's girth conjecture) to approximate the solution cost to a factor better than $2t + 1$.

[7] All results hold for a more general class of *proper* functions too. The function f is called proper if $f(V) = 0$, $f(U) = f(V \setminus U)$ for every $U \subset V$ (symmetry), and $f(U_1 \cup U_2) \leq \max\{f(U_1), f(U_2)\}$ whenever U_1 and U_2 are disjoint (maximality).

Min-Weight k-ECSS. As a corollary of Theorem 1 for CAP in link arrival streams, we have the following guarantee for the problem of finding minimum-weight k-edge-connected spanning subgraph (k-ECSS), where given a graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}_{\geq 0}$, the goal is to find a minimum-weight k-edge-connected subgraph $H \subseteq G$.

▶ **Corollary 26.** *There exists a k-pass $O(\log k)$-approximation algorithm for minimum-weight k-ECSS with total memory space $O(nk + n \log \min(n, W))$ where $W = \max_{e \in E} w(e)$.*

C Related Work

Approximation algorithms of k-CAP. The edge-connectivity of a graph plays a central role in a wide range of network design problems, spanning both classical and modern problems. While the celebrated iterative rounding technique of [29] provides a 2-approximation for most of these problems, any better than 2-approximation for them are among main open problems within the field of approximation algorithms.

Significant progress has been made in achieving better than a 2-approximation for specific instances of the weighted k-CAP.Notably, extensive research focusing on the well-studied unweighted TAP has led to breakthroughs [40, 16, 38, 27, 9], culminating in an approximation factor of 1.326 [24]. Remarkably, this same factor has also been achieved for the unweighted k-CAP [9], a problem that recently saw significant advancements surpassing the 2-approximation barrier [8]. Moreover, in a recent development, the weighted TAP and k-CAP have witnessed breakthroughs with approximation factors below 2 [48, 49, 50]. It is noteworthy that these advancements in the weighted variants are relatively recent in the research landscape.

The Steiner tree augmentation problem, in which given a Steiner tree $T \subset G = (V, E)$ over terminals $R \subset V$ the goal is to find a minimum weight set of edges $H \subseteq G \setminus T$ that increases the connectivity of the set R to 2, has also been studied and recently [45] provides $(1.5 + \epsilon)$-approximation generalizing some of the techniques in [49].

SNDP. Similarly to k-ECSS, the augmentation variant of SNDP has been extensively studied and is significant in the development of approximation algorithms for different variations of SNDP. Notably, the augmentation variant of SNDP generalizes well-studied problems such as TAP, STAP and k-CAP. The augmentation variant of SNDP was originally studied to analyze the primal-dual methods for SNDP, leading to k and $\log k$ approximations [51, 26], and compared to the state-of-the-art 2-approximation iterative rounding technique of [29] has the advantage of applicability to other variants of SNDP such as node-weighted SNDP [44, 11, 10] or vertex-connectivity SNDP [37, 17, 12].

Spanners and sparsifiers. Graph spanners are important tools for graph compression in which the distances between the nodes are preserved. See [1] for a survey on graph spanners in general. Spanners have also been studied extensively in the streaming setting, see e.g., [7, 2, 15, 33, 21]. For other notions of graph sparsifiers in the streaming model, see e.g., [31, 32].

A Faster Algorithm for Pigeonhole Equal Sums

Ce Jin
MIT, Cambridge, MA, USA

Hongxun Wu
University of California Berkeley, CA, USA

Abstract

An important area of research in exact algorithms is to solve Subset-Sum-type problems faster than meet-in-middle. In this paper we study *Pigeonhole Equal Sums*, a total search problem proposed by Papadimitriou (1994): given n positive integers $w_1, \ldots, w_n$ of total sum $\sum_{i=1}^{n} w_i < 2^n - 1$, the task is to find two distinct subsets $A, B \subseteq [n]$ such that $\sum_{i \in A} w_i = \sum_{i \in B} w_i$.

Similar to the status of the Subset Sum problem, the best known algorithm for Pigeonhole Equal Sums runs in $O^*(2^{n/2})$ time, via either meet-in-middle or dynamic programming (Allcock, Hamoudi, Joux, Klingelhöfer, and Santha, 2022).

Our main result is an improved algorithm for Pigeonhole Equal Sums in $O^*(2^{0.4n})$ time. We also give a polynomial-space algorithm in $O^*(2^{0.75n})$ time. Unlike many previous works in this area, our approach does not use the representation method, but rather exploits a simple structural characterization of input instances with few solutions.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Subset Sum, Pigeonhole, PPP

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.94

Category Track A: Algorithms, Complexity and Games

Funding *Ce Jin*: Supported by NSF grants CCF-2129139 and CCF-2127597. Work done while visiting the Simons Institute for the Theory of Computing.
Hongxun Wu: Supported by Avishay Tal's Sloan Research Fellowship, NSF CAREER Award CCF-2145474, and Jelani Nelson's ONR grant N00014-18-1-2562.

Acknowledgements We thank Ryan Williams for useful discussions.

1 Introduction

The Subset Sum problem is an important NP-hard problem in computer science: given positive integers $w_1, w_2, \ldots, w_n$ and a target integer t, find a subset $A \subseteq [n]$ such that $\sum_{i \in A} w_i = t$. Subset Sum can be solved in $O(2^{n/2})$ time by a simple meet-in-middle algorithm [14], and an important open problem is to improve it to $O(2^{(1/2-\varepsilon)n})$. A long line of research attempts to solve Subset Sum faster using the representation method [15] and connections to uniquely decodable code pairs [3, 4, 22], but these techniques have so far only succeeded on average-case inputs [15, 8, 9] or restricted classes of inputs [2, 3]. Nevertheless, significant progress has been made for other variants of Subset Sum, including Equal Sums [17], 2-Subset Sum and Shifted Sums [1] and more general subset balancing problems [12], as well as Subset Sum in other computational settings such as Merlin–Arthur protocols [18], low-space algorithms [6, 19], quantum algorithms [1], and algorithms with lower-order run time improvements [13]. The general hope is that the tools developed for solving these variant problems might one day help solve the original Subset Sum problem.


In this paper we study an interesting variant of Subset Sum called *Pigeonhole Equal Sums*:

> PIGEONHOLE EQUAL SUMS [20]
> **Input:** positive integers $w_1, w_2, \ldots, w_n$, with promise $\sum_{i=1}^{n} w_i < 2^n - 1$.
> **Output:** two different subsets $A, B \subseteq [n]$ such that $\sum_{i \in A} w_i = \sum_{i \in B} w_i$.

Since there are 2^n subsets $S \subseteq [n]$ with only $2^n - 1$ possible subset sums $\sum_{i \in S} w_i \in \{0, 1, \ldots, 2^n - 2\}$ due to the promise, the pigeonhole principle guarantees that there exists a pair of subsets with the same subset sum.

Pigeonhole Equal Sums was introduced by Papadimitriou [20] as a natural example problem in the total search complexity class PPP. This problem has received attention in the TFNP literature [5, 21], and is conjectured to be PPP-complete [20].

From the algorithmic point of view, the current status of Pigeonhole Equal Sums is quite similar to that of the Subset Sum problem: a simple binary search with meet-in-middle solves Pigeonhole Equal Sums in $O^*(2^{n/2})$ time (see Section 2).[1] Allcock, Hamoudi, Joux, Klingelhöfer, and Santha [1, Theorem 6.2] gave another $O^*(2^{n/2})$-time algorithm based on dynamic programming (which is analogous to the alternative $O^*(2^{n/2})$-time Subset Sum algorithm from [3][2]). It remains open whether $O(2^{(1/2-\varepsilon)n})$ time is possible for Pigeonhole Equal Sums. Improvement of such type was achieved for the Equal Sums problem (without the pigeonhole promise) by Mucha, Nederlof, Pawlewicz, and Węgrzycki [17] via the representation method with $O(3^{(1/2-\varepsilon)n})$ run time for some $\varepsilon > 0.01$, but this result has no direct implications for Pigeonhole Equal Sums (for which the known $O^*(2^{n/2})$ time bound is already much better than $O(3^{n/2})$).

1.1 Our results

We give an algorithm that solves Pigeonhole Equal Sums faster than the previous $O^*(2^{n/2})$ running time [1].

▶ **Theorem 1** (Main). *Pigeonhole Equal Sums can be solved by a randomized algorithm in $O^*(2^{0.4n})$ time.*

Surprisingly, unlike previous works on other variants of Subset Sum, our algorithm does not use the representation method [15] or tools from coding theory [3, 4, 22]. Instead, our main insight is a simple structural characterization of Pigeonhole Equal Sums instances with few solutions.

Our techniques also yield a fast polynomial-space algorithm for Pigeonhole Equal Sums, in an analogous way to the previous $O(3^{(1-\varepsilon)n})$-time polynomial-space algorithm for Equal Sums [17].

▶ **Theorem 2.** *Pigeonhole Equal Sums can be solved by a randomized algorithm in $O^*(2^{0.75n})$ time and $\mathrm{poly}(n)$ space.*

For comparison, a straightforward algorithm based on binary search solves Pigeonhole Equal Sums in $\mathrm{poly}(n)$ space and $O^*(2^n)$ time (see the beginning of Section 4).

Theorem 1 and Theorem 2 will be proved in Section 3 and Section 4 respectively.

---

[1] We use $O^*(\cdot)$ to hide $\mathrm{poly}(n)$ factors.
[2] See also `https://youtu.be/cHimhXXIwcg?t=454`.

2 Preliminaries

Denote $[n] = \{1, \dots, n\}$. Let $O^*(\cdot), \Omega^*(\cdot)$ hide $\mathrm{poly}(n)$ factors, where n is the number of input integers in the Pigeonhole Equal Sums problem.

Denote $w(A) = \sum_{i \in A} w_i$ for $A \subseteq [n]$. The pigeonhole promise states $w([n]) < 2^n - 1$.

For a predicate p we define $\mathbf{1}[p] = 1$ if p is true and $\mathbf{1}[p] = 0$ if p is false.

We need the following well-known lemma.

▶ **Lemma 3** (Counting subset sums via meet-in-middle [14]). *Given integers $w_1, \dots, w_n$ and t, we can compute $\#\{S \subseteq [n] : w(S) \le t\}$ in $O^*(2^{n/2})$ time. Moreover, we can list $S \subseteq [n]$ such that $w(S) \le t$ in $O^*(1)$ additional time per S.*

Proof. Divide $[n]$ into $S_1 = \{1, \dots, \lfloor n/2 \rfloor\}$ and $S_2 = [n] \setminus S_1$, and every subset $S \subseteq [n]$ can be represented as $X \uplus Y, X \subseteq S_1, Y \subseteq S_2$. Compute and sort the two lists $A = \{w(X)\}_{X \subseteq S_1}$ and $B = \{w(Y)\}_{Y \subseteq S_2}$ of length $O(2^{n/2})$ each. Then for each $w(X) \in A$ we accumulate $|B \cap (-\infty, t - w(X)]|$ to the answer. It is easy to augment this algorithm to support listing. ◀

Pigeonhole Equal Sums via binary search

The following simple binary-search algorithm (described in [1, Remark 6.9 of arXiv version] and attributed to an anonymous referee) solves Pigeonhole Equal Sums in $O^*(2^{n/2})$ time: Maintain an interval $\{\ell, \ell+1, \dots, r\}$ (initialized to $\ell = 0, r = 2^n - 2$) that satisfies the pigeonhole invariant $r - \ell + 1 < \#\{S \subseteq [n] : \ell \le w(S) \le r\}$. Initially this invariant is satisfied due to $w([n]) \le 2^n - 2$. While $r > \ell$, pick the middle point $m = \lfloor \frac{\ell + r}{2} \rfloor$, and use meet-in-middle (Lemma 3) to compute $c_1 = \#\{S \subseteq [n] : \ell \le w(S) \le m\}$ and $c_2 = \#\{S \subseteq [n] : m+1 \le w(S) \le r\}$ in $O^*(2^{n/2})$ time. Then we shrink the interval to $\{\ell, \dots, m\}$ if $m - \ell + 1 < c_1$, or to $\{m+1, \dots, r\}$ if $r - m < c_2$ (the invariant guarantees that at least one holds). After $\lceil \log_2(2^n - 1) \rceil = n$ iterations we shrink to a singleton interval $\ell = r$. By the invariant, there exist two different $S_1, S_2 \subseteq [n]$ such that $w(S_1) = w(S_2) = \ell$, and we can report such S_1, S_2 using meet-in-middle (Lemma 3).

This binary-search strategy will be used in our improved algorithms as well.

3 The improved algorithm

Let the n input integers be sorted as $0 < w_1 < w_2 < \cdots < w_n$ (assuming no trivial solution $w_i = w_j$ exists).

An assumption on prefix sums

If any proper prefix $\{w_1, \dots, w_i\}$ $(i \le n-1)$ already satisfies the pigeonhole promise $w([i]) < 2^i - 1$, then we can instead solve the smaller Pigeonhole Equal Sums instance $\{w_1, \dots, w_i\}$ and obtain $A, B \subseteq [i], A \ne B$ with $w(A) = w(B)$. Hence, without loss of generality we assume such prefix does not exist, i.e.,

$$w([i]) \ge 2^i - 1 \text{ for all } i \in [n-1]. \tag{1}$$

Frequencies f_t and parameter d

The *frequency* (also called bin size) of $t \in \mathbb{N}$ is the number of input subsets achieving sum t, denoted as $f_t = \#\{S \subseteq [n] : w(S) = t\}$. Since $w([n]) < 2^n - 1$, we know $f_t = 0$ for all $t \ge 2^n - 1$, and

$$\sum_{0 \le t < 2^n} f_t = 2^n. \tag{2}$$

Two different subsets achieving equal subset sum t imply $f_t > 1$. This motivates the following parameter,

$$d = \sum_{0 \le t < 2^n} \max\{0, f_t - 1\}, \tag{3}$$

which counts the (non-trivial) equality relations among all the 2^n subset sums. Using Equation (2), we can rewrite Equation (3) as $d = \sum_{0 \le t < 2^n} (f_t - \mathbf{1}[f_t \ge 1]) = 2^n - \sum_{0 \le t < 2^n} \mathbf{1}[f_t \ge 1]$, and thus obtain

$$d = \#\{0 \le t < 2^n : f_t = 0\}, \tag{4}$$

which counts the non-subset-sums in $\{0, 1, \ldots, 2^n - 1\}$. In particular, $d < 2^n$.

The equivalence between Equation (3) and Equation (4) is powerful. In the following we will give two different algorithms for Pigeonhole Equal Sums. The first one works for small d by analyzing the structure of input instances with few non-subset-sums (by Equation (4)). The second one works when d is large and hence there are many solutions (by Equation (3)) which allow a subsampling approach. These two algorithms are summarized as follows:

▶ **Lemma 4.** *Given parameter $\Delta \le 2^n/(3n^2)$, Pigeonhole Equal Sums with $d \le \Delta$ can be solved deterministically in $O^*(\sqrt{\Delta})$ time.*

▶ **Lemma 5.** *Given parameter $2^{n/2} \le \Delta < 2^n$, Pigeonhole Equal Sums with $d \ge \Delta$ can be solved in $O^*((2^{2n}/\Delta)^{1/3})$ time by a randomized algorithm.*

Combining these two lemmas implies our main result:

Proof of Theorem 1. Set $\Delta = 2^{0.8n}$ so that the two time bounds in Lemma 4 and Lemma 5 are balanced to $O^*(2^{0.4n})$. Given an instance of Pigeonhole Equal Sums (with unknown d), we run both algorithms in parallel, and return the answer of whichever terminates first. ◀

3.1 Small d case via structural characterization

In this section we prove Lemma 4. Assume $d \le \Delta \le 2^n/(3n^2)$ and Δ is known.

Since $f_t = 0$ for all $w([n]) < t < 2^n$, from Equation (4) we know $d \ge 2^n - 1 - w([n])$, and hence $w([n]) \ge 2^n - 1 - d \ge 2^n - 1 - \Delta$. Combined with Equation (1) for $i \in [n-1]$, we get the following lower bound

$$w([i]) \ge 2^i - 1 - \Delta \text{ for all } i \in [n]. \tag{5}$$

The key step is to complement Equation (5) with a nearly matching upper bound:

▶ **Lemma 6.** *For all $i \in [n]$,*

$$w_i \le 2^{i-1} + \Delta. \tag{6}$$

Summing Equation (6) over i gives

$$w([i]) \le 2^i - 1 + i\Delta \tag{7}$$

for all $i \in [n]$.

Proof. Fix $i \in [n]$. Let M be the number of subsets $S \subseteq [n]$ with $w(S) < w_i$. Since $w_i < w_{i+1} < \cdots < w_n$, any such S must be contained in $[i-1]$, and thus $M \le 2^{i-1}$. On the other hand, $M = \sum_{t=0}^{w_i - 1} f_t \ge w_i - \#\{0 \le t < w_i : f_t = 0\} \ge w_i - d$ by Equation (4). Hence, $w_i \le M + d \le 2^{i-1} + \Delta$. ◀

Comparing Equation (5) with Equation (7) gives the lower bound

$$w_i = w([i]) - w([i-1]) \geq (2^i - 1 - \Delta) - (2^{i-1} - 1 + (i-1)\Delta) = 2^{i-1} - i\Delta,$$

which is very close to the upper bound from Equation (6). Together we get

$$w_i - 2^{i-1} \in [-i\Delta, \Delta] \tag{8}$$

for all $i \in [n]$.

Equation (8) gives a very rigid structure of the large input numbers. In the next lemma we exploit this structure to improve the naive meet-in-middle subset sum counting algorithm from Lemma 3.

▶ **Lemma 7.** *For any given $T < 2^n$, we can compute $\sum_{t=0}^{T} f_t$ in $O^*(\sqrt{\Delta})$ time.*

Proof. Let i^* be the minimum $i^* \in [n]$ such that $2^{i^*} \geq 3n^2\Delta$, which exists by our assumption $\Delta \leq 2^n/(3n^2)$. Let $A = \{1, 2, \ldots, i^*\}$ and $B = \{i^* + 1, \ldots, n\}$.

By Equation (7), $w(A) < 2^{i^*} + n\Delta$.

For every $B' \subseteq B$, by Equation (8) we have

$$\left|w(B') - \sum_{j \in B'} 2^{j-1}\right| \leq \sum_{j \in B'} |w_j - 2^{j-1}| \leq \sum_{j \in B'} j\Delta \leq n^2\Delta.$$

In other words, the subset sums of $\{w_j\}_{j \in B}$ are $n^2\Delta$-additively approximated by the subset sums of $\{2^{j-1}\}_{j \in B}$. The subset sums of the latter set form an arithmetic progression $\{k \cdot 2^{i^*} : 0 \leq k < 2^{n-i^*}\}$, namely all n-bit binary numbers whose lowest i^* bits are zeros. Notably, this arithmetic progression is very sparse: its difference 2^{i^*} is large enough compared to $w(A) < 2^{i^*} + n\Delta$.

Given query T, we want to count the number of pairs (A', B') $(A' \subseteq A, B' \subseteq B)$ such that $w(A') + w(B') \leq T$. To do this, we enumerate $B' \subseteq B$, and consider three cases (the non-trivial case is Case 3, where $w(B')$ and $\sum_{j \in B'} 2^{j-1}$ are close to T):

- **Case 1:** $\sum_{j \in B'} 2^{j-1} \leq T - 2^{i^*} - (n + n^2)\Delta$.
  Then, for all $A' \subseteq A$, we have $w(A') + w(B') \leq w(A) + w(B') \leq (2^{i^*} + n\Delta) + (n^2\Delta + \sum_{j \in B'} 2^{j-1}) \leq T$. Hence B' contributes $2^{|A|}$ many pairs (A', B').
- **Case 2:** $\sum_{j \in B'} 2^{j-1} > T + n^2\Delta$.
  Then, for all $A' \subseteq A$, we have $w(A') + w(B') \geq w(B') \geq \sum_{j \in B'} 2^{j-1} - n^2\Delta > T$. Hence B' does not contribute any pairs (A', B').
- **Case 3:** otherwise, $\sum_{j \in B'} 2^{j-1} \in (T - 2^{i^*} - (n + n^2)\Delta, T + n^2\Delta]$.
  This interval has length $2^{i^*} + (n + n^2)\Delta + n^2\Delta \leq 2 \cdot 2^{i^*}$ by our choice of i^*. Since $\sum_{j \in B'} 2^{j-1}$ is a multiple of 2^{i^*} in this interval, it has at most two possibilities, namely $2^{i^*} \cdot \lfloor \frac{T-(n+n^2)\Delta}{2^{i^*}} \rfloor$ and $2^{i^*} \cdot \left(\lfloor \frac{T-(n+n^2)\Delta}{2^{i^*}} \rfloor + 1\right)$, and then B' is uniquely determined by the binary decomposition of $\sum_{j \in B'} 2^{j-1}$. For each possible B', we count the number of $A' \subseteq A$ such that $w(A') \leq T - w(B')$ using meet-in-middle (Lemma 3) with time complexity $O^*(2^{|A|/2}) = O^*(2^{i^*/2}) = O^*(\sqrt{\Delta})$ by the definition of i^*.

Note that in $O^*(1)$ time we can easily find the (at most two) subsets B' satisfying Case 3, and also count the total contribution of Case 1. Hence the overall time complexity is $O^*(\sqrt{\Delta})$. ◀

Using Lemma 7 we can solve Pigeonhole Equal Sums using binary search, in the same way as described in the last paragraph of Section 2. The running time is $O^*(\sqrt{\Delta})$. This finishes the proof of Lemma 4.

3.2 Large d case via subsampling

In this section we prove Lemma 5. Assume $2^{n/2} \le \Delta \le d < 2^n$, and Δ is known. We first use $d = \sum_{0 \le t < 2^n} \max\{0, f_t - 1\}$ (Equation (3)) to show that many subset sums t have large f_t, which then allows us to use subsampling to speed up the modular dynamic programming approach of [1, 3].

▶ **Lemma 8.** *There exists a $j \in \{0, 1, \ldots, n-1\}$ such that $\#\{t : f_t > 2^j\} > \frac{\Delta}{2^{j+1}n}$.*

Proof. By definition of d in Equation (3),

$$\Delta \le d = \sum_{t : f_t > 1} (f_t - 1) \le \sum_{0 \le j < n} \#\{t : 2^j < f_t \le 2^{j+1}\} \cdot (2^{j+1} - 1). \tag{9}$$

If the claimed inequality fails for all j, then

$$[\text{RHS of Equation (9)}] \le \sum_{0 \le j < n} \frac{\Delta}{2^{j+1}n} \cdot (2^{j+1} - 1) < \Delta,$$

a contradiction. ◀

Our algorithm enumerates all $j \in \{0, 1, \ldots, n-1\}$ (increasing the time complexity by a factor of $n = O^*(1)$), and from now on we assume j satisfies the inequality in Lemma 8. Define

$$h := 2^j + 1 \ge 2, \quad m := \left\lceil \frac{\Delta}{2^{j+1}n} \right\rceil > \frac{\Delta}{2hn}, \quad \text{and } X := \{t \in [2^n] : f_t \ge h\}. \tag{10}$$

Here we defined the set X of frequent subset sums only for the sake of analysis. By Lemma 8,

$$|X| \ge m. \tag{11}$$

Readers are encouraged to focus on the case of $h = 2$ and $m \ge \Omega^*(\Delta)$ (which is the hardest case for our algorithm) at first read.

We first describe the behavior of our algorithm: Let $p \in [P, 2P]$ be a uniformly random prime (for some parameter $2 \le P \le 2m$ to be determined later in the "Time complexity" paragraph). For each $r \in \mathbb{Z}_p$, define bin $B_r := \{S \subseteq [n] : w(S) \equiv r \pmod p\}$. The algorithm picks a random bin index $r^* \in \mathbb{Z}_p$, and subsamples $C \subseteq B_{r^*}$ by keeping each $S \in B_{r^*}$ with probability α independently (for some $0 < \alpha \le \frac{1}{2h}$ to be determined later in the "Success probability" paragraph). Finally, a pair of distinct $S, S' \in C$ with $w(S) = w(S')$ is reported (if exists).

Now we explain how to implement the algorithm above via dynamic programming (DP) similarly to [1, 3]. Build the DP table $D_{i,r} = \#\{S \subseteq [i] : w(S) \equiv r \pmod p\}$ (where $0 \le i \le n$ and $r \in \mathbb{Z}_p$) in $O^*(p)$ overall time via the transition $D_{i,r} = D_{i-1,r} + D_{i-1,(r-w_i) \bmod p}$ with initial values $D_{0,r} = \mathbf{1}[r = 0]$. This DP computes the size of every bin $|B_r| = D_{n,r}$. Furthermore, for any bin B_r and integer $k \in [|B_r|]$, we can report the rank-k set S in B_r (in lexicographical order, where larger indices are compared first) by backtracing in the DP table in $O^*(1)$ time. Then, in order to subsample a collection of sets $C \subseteq B_{r^*}$ at rate α, we can first subsample their ranks in $[|B_{r^*}|]$ (in near-linear time in the output size, see e.g., [10]), and then recover the actual sets by backtracing.

Success probability

We study how the frequent subset sums, $X = \{t : f_t \geq h\}$, are distributed to the bins modulo a random prime p, using an argument similar to [3]. Setting

$$k := \lceil \frac{m}{4P} \rceil, \tag{12}$$

the following lemma shows that the bin B_{r^*} receives at least k frequent subset sums, with $\Omega^*(1)$ probability.

▶ **Lemma 9.** *With at least $\Omega(1/n)$ probability over the choice of prime $p \in [P, 2P]$ and $r^* \in \mathbb{Z}_p$, there are at least k integers $t \in \mathbb{N}$ such that $\#\{S \in B_{r^*} : w(S) = t\} \geq h$.*

Proof. Since $|X| \geq m$ by Equation (11), we arbitrarily pick $X' \subseteq X$ with $|X'| = m$ for the sake of analysis. Let $c_{r,p} := \{t \in X' : t \equiv r \pmod p\}$. Then,

$$\begin{aligned}
\mathop{\mathbf{E}}_{p\in[P,2P]}\Big[\sum_{r\in\mathbb{Z}_p} c_{r,p}^2\Big] &= \sum_{x\in X', y\in X'} \mathop{\mathbf{Pr}}_{p\in[P,2P]}[p \mid x-y] \\
&\leq m + m^2 \cdot \frac{\log_P 2^n}{\Omega(P/\ln P)} \qquad \text{(by } |x-y| \leq 2^n \text{ and the density of primes)} \\
&\leq O(n \cdot m^2/P). \qquad \text{(by the assumption that } P \leq 2m)
\end{aligned}$$

Then by Markov's inequality, with 0.9 success probability over the choice of p, we have $\sum_{r\in\mathbb{Z}_p} c_{r,p}^2 \leq O(n \cdot m^2/P)$. Conditioned on this happening, by Cauchy–Schwarz inequality we have

$$\begin{aligned}
\sum_{r\in\mathbb{Z}_p} \mathbf{1}[c_{r,p} \geq \tfrac{m}{2p}] &\geq \frac{\left(\sum_{r\in\mathbb{Z}_p} \mathbf{1}[c_{r,p} \geq \frac{m}{2p}] \cdot c_{r,p}\right)^2}{\sum_{r\in\mathbb{Z}_p} c_{r,p}^2} \\
&\geq \frac{\left((\sum_{r\in\mathbb{Z}_p} c_{r,p}) - p \cdot \frac{m}{2p}\right)^2}{O(n \cdot m^2/P)} = \frac{(|X'| - m/2)^2}{O(n \cdot m^2/P)} = \frac{(m/2)^2}{O(n \cdot m^2/P)} = \Omega(P/n),
\end{aligned}$$

and hence, by our choice of $k = \lceil \frac{m}{4P} \rceil \leq \lceil \frac{m}{2p} \rceil$,

$$\mathop{\mathbf{Pr}}_{r^*\in\mathbb{Z}_p}[c_{r^*,p} \geq k] \geq \mathop{\mathbf{Pr}}_{r^*\in\mathbb{Z}_p}[c_{r^*,p} \geq \tfrac{m}{2p}] \geq \frac{\Omega(P/n)}{p} = \Omega(1/n).$$

Conditioned on $c_{r^*,p} \geq k$ happening, we have at least k integers $t \in X' \subseteq X$ such that $t \equiv r^* \pmod p$. By definitions of B_{r^*} and X, this implies that there are at least k integers $t \in \mathbb{N}$ such that $\#\{S \in B_{r^*} : w(S) = t\} \geq h$, with overall success probability at least $0.9 \cdot \Omega(1/n) = \Omega(1/n)$ over the choice of p and r^*. ◀

Recall our algorithm subsamples $C \subseteq B_{r^*}$ at rate $\alpha \in (0, \frac{1}{2h}]$, and fails iff $w(S)$ are distinct for all $S \in C$. The failure probability of this step can be derived from the following lemma:

▶ **Lemma 10.** *Let B' be a collection of kh colored balls ($h \geq 2, k \geq 1$), with exactly h balls of color i for each color $i \in [k]$. Let $C' \subseteq B'$ be an i.i.d. subsample at rate $\alpha \in [0, \frac{1}{2h}]$. Then C' contains distinct colors with at most $\exp(-kh(h-1)\alpha^2/4)$ probability.*

Proof. For each color $i \in [k]$, by Bernoulli's inequality, the probability that C' includes exactly two balls of color i is $\binom{h}{2}\alpha^2(1-\alpha)^{h-2} \geq \binom{h}{2}\alpha^2(1-(h-2)\alpha) \geq \binom{h}{2}\alpha^2/2$. Hence, the probability that C' includes at most one ball of every color $i \in [k]$ is at most $\left(1-\binom{h}{2}\alpha^2/2\right)^k \leq \exp\left(-k\binom{h}{2}\alpha^2/2\right) = \exp(-kh(h-1)\alpha^2/4)$. ◀

We think of each set $S \in B_{r^*}$ as a ball of color $w(S)$, and apply Lemma 10 to the k integers (colors) $t \in \mathbb{N}$ ensured by Lemma 9, each having at least h sets (balls) $S \in B_{r^*}$ with $w(S) = t$. We set the sample rate to be

$$\alpha := \frac{1}{2h\sqrt{k}} \le \frac{1}{2h}. \tag{13}$$

Then the failure probability of the subsampling step is at most

$$\exp(-kh(h-1)\alpha^2/4) = \exp(-\tfrac{h-1}{16h}) \le \exp(-1/32).$$

Overall, the probability that the algorithm successfully finds a solution is at least $\Omega(1/n) \cdot (1 - \exp(-1/32)) \ge \Omega(n^{-1})$.

Time complexity

The mod-p DP runs in $O^*(p) \le O^*(P)$ time. Since the bins have total size $\sum_{r\in\mathbb{Z}_p} |B_r| = 2^n$, the chosen bin B_{r^*} has expected size $\mathbf{E}_{r^*\in\mathbb{Z}_p}[|B_{r^*}|] = 2^n/p \le 2^n/P$, and hence the subsample $C \subseteq B_{r^*}$ has expected size $\mathbf{E}[|C|] \le \alpha 2^n/P$. To detect a solution $S, S' \in C$ with $w(S) = w(S')$, we simply sort C in near-linear time. Hence the total expected running time is $O^*(P + \alpha 2^n/P)$. By Markov's inequality, with probability at least $1 - n^{-10}$, the algorithm terminates in $O^*(P + \alpha 2^n/P)$ time. By a union bound, the algorithm successfully finds a solution in time $O^*(P + \alpha 2^n/P)$ with probability at least $\Omega(n^{-1}) - n^{-10} \ge \Omega(n^{-1})$. This success probability can be boosted to 0.99 by repeating the algorithm $O(n)$ times.

Recall from Equations (12) and (13) that $\alpha = \frac{1}{2h\sqrt{k}} = \frac{1}{2h\sqrt{\lceil m/4P\rceil}} \le \frac{\sqrt{P}}{h\sqrt{m}}$, so the run time is (ignoring poly(n) factors)

$$P + \alpha 2^n/P \le P + \frac{2^n}{h\sqrt{mP}}.$$

Recall $h = 2^j + 1$ (where $0 \le j \le n-1$) and $m = \lceil \frac{\Delta}{2^{j+1}n} \rceil$, and hence $hm < h(1 + \frac{\Delta}{2^{j+1}n}) \le h + \frac{\Delta}{n} < (2^{n-1}+1) + \frac{2^n}{n} \le 2^n$ (assuming $n \ge 3$). Now we set

$$P := 2m \cdot \min\left\{1, \left(\frac{2^n}{hm^2}\right)^{2/3}\right\},$$

and we first need to verify the requirement $2 \le P \le 2m$ introduced earlier: The upper bound is obvious. To see the lower bound, note that $2m \ge 2$ and $2m \cdot \left(\frac{2^n}{hm^2}\right)^{2/3} = 2\left(\frac{2^{2n}}{h^2 m}\right)^{1/3} \ge 2\left(\frac{2^{2n}}{(hm)^2}\right)^{1/3} \ge 2$ (using the inequality $hm \le 2^n$ we just showed).

Hence, the overall running time is at most (ignoring poly(n) factors)

$$\begin{aligned}
P + \frac{2^n}{h\sqrt{mP}} &\le 2m\left(\frac{2^n}{hm^2}\right)^{2/3} + \frac{2^n}{h\sqrt{m\cdot 2m}} \cdot \max\left\{1, \left(\frac{hm^2}{2^n}\right)^{1/3}\right\} \\
&= 2\cdot\frac{2^{2n/3}}{h^{2/3}m^{1/3}} + \frac{1}{\sqrt{2}}\max\left\{\frac{2^n}{hm}, \frac{2^{2n/3}}{h^{2/3}m^{1/3}}\right\} \\
&\le O\left(\frac{2^{2n/3}}{h^{2/3}m^{1/3}} + \frac{2^n}{hm}\right) \\
&\le O^*\left(\frac{2^{2n/3}}{h^{1/3}\Delta^{1/3}} + \frac{2^n}{\Delta}\right) && \text{(by } hm > \tfrac{\Delta}{2n} \text{ from Equation (10))} \\
&\le O^*\left(\frac{2^{2n/3}}{\Delta^{1/3}}\right). && \text{(by } h > 1 \text{ and the assumption that } \Delta \ge 2^{n/2})
\end{aligned}$$

This finishes the proof of Lemma 5.

4 A polynomial-space algorithm

We now consider poly(n)-space algorithms for Pigeonhole Equal Sums. The straightforward binary search approach (described at the end of Section 2) can be adapted to run in $O^*(2^n)$ time and poly(n) space: instead of using meet-in-middle (Lemma 3, which requires large space), we count the number of valid subsets $S \subseteq [n]$ by brute force in $O^*(2^n)$ time and only poly(n) space.

We improve this $O^*(2^n)$ running time using the ideas from earlier sections. Again, consider two cases depending on whether parameter d from Equation (3) is small or large.

▶ **Lemma 11.** *Given parameter $\Delta \le 2^n/(3n^2)$, Pigeonhole Equal Sums with $d \le \Delta$ can be solved deterministically in* poly(n) *space and $O^*(\Delta)$ time.*

Proof Sketch. The proof is almost the same as Lemma 4 (see Section 3.1), with the only difference in Case 3 from the proof of Lemma 7: instead of using meet-in-middle, here we count the valid subsets $A' \subseteq A$ by brute force in $O^*(2^{|A|}) = O^*(2^{i^*}) = O^*(\Delta)$ time and only poly(n) space. ◀

To solve the large d case, we need the low-space element distinctness algorithm by Beame, Clifford, and Machmouchi [7] (generalized in [6], and with a non-standard assumption removed by [11, 16]). This algorithm was also previously used for Subset Sum [6] and Equal Sums [17]. The following statement can be inferred from [11, Section 4.2 (proof of Theorem 1.1)].

▶ **Theorem 12** (Low-space Element Distinctness, [7, 6, 11])**.** *Given random access to an integer list $a_1, \dots, a_N$ (where $a_i \in [\text{poly}(N)]$) that contains at least one pair $(i, j) \in [N] \times [N]$ with $a_i = a_j, i \ne j$, there is a randomized algorithm that reports such a pair using* poly log N *working memory and*

$$O\left(\frac{N\sqrt{F_2}}{F_2 - N} \cdot \text{poly}\log N\right)$$

time, where $F_2 = \sum_{i=1}^{N} \sum_{j=1}^{N} \mathbf{1}[a_i = a_j] \in [N + 2, N^2]$.[3]

▶ **Lemma 13.** *Given parameter $1 \le \Delta \le 2^n$, Pigeonhole Equal Sums with $d \ge \Delta$ can be solved in $O^*(2^{1.5n}/\Delta)$ time and* poly(n) *space by a randomized algorithm.*

Proof. Apply Theorem 12 to the list $\{w(A)\}_{A \subseteq [n]}$ of length $N = 2^n$ and we obtain a pair of distinct $A, A' \subseteq [n]$ with $w(A) = w(A')$ as desired. The space complexity is poly log$(2^n) =$ poly(n). To analyze the time complexity, note that

$$F_2 - 2^n = \sum_{A \subseteq [n]} \sum_{\substack{B \subseteq [n] \\ B \ne A}} \mathbf{1}[w(A) = w(B)] = \sum_{0 \le t < 2^n} f_t(f_t - 1) \ge \sum_{0 \le t < 2^n} \max\{0, f_t - 1\} \overset{\text{Eq. (3)}}{=} d \ge \Delta,$$

so the time bound is (ignoring poly(n) factors)

$$\frac{2^n\sqrt{F_2}}{F_2 - 2^n} < \frac{2^{0.5n}F_2}{F_2 - 2^n} = 2^{0.5n}\left(1 + \frac{2^n}{F_2 - 2^n}\right) \le 2^{0.5n}\left(1 + \frac{2^n}{\Delta}\right) \le \frac{2 \cdot 2^{1.5n}}{\Delta}$$

as claimed. ◀

[3] We have $F_2 \ge N + 2$ due to the following $(N + 2)$ pairs: $(1, 1), (2, 2), \dots, (N, N)$ and $(i, j), (j, i)$, where $a_i = a_j$ $(i \ne j)$.

Combining the two lemmas gives the desired result.

Proof of Theorem 2. Set $\Delta = 2^{0.75n}$ so that the two time bounds in Lemma 11 and Lemma 13 are balanced to $O^*(2^{0.75n})$. Given an instance of Pigeonhole Equal Sums (with unknown d), we run both algorithms in parallel, and return the answer of whichever terminates first. ◀

5 Open problems

Allcock et al. [1] proposed a modular variant of the Pigeonhole Equal Sums problem: given integers $w_1, \ldots, w_n$ and a modulus $m \leq 2^n - 1$, find two distinct subsets $A, B \subseteq [n]$ such that $\sum_{i \in A} w_i \equiv \sum_{i \in B} w_i \pmod{m}$. They obtained a $O^*(2^{n/2})$-time algorithm for this problem. Can this result be improved as well?

Can we obtain faster algorithms for other problems in PPP (e.g., [5, 21])?

References

1 Jonathan Allcock, Yassine Hamoudi, Antoine Joux, Felix Klingelhöfer, and Miklos Santha. Classical and quantum algorithms for variants of subset-sum via dynamic programming. In *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 6:1–6:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.6.

2 Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Subset sum in the absence of concentration. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 48–61. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.48.

3 Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Dense subset sum may be the hardest. In *Proceedings of the 33rd Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 47 of *LIPIcs*, pages 13:1–13:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.13.

4 Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Sharper upper bounds for unbalanced uniquely decodable code pairs. *IEEE Trans. Inf. Theory*, 64(2):1368–1373, 2018. doi:10.1109/TIT.2017.2688378.

5 Frank Ban, Kamal Jain, Christos H. Papadimitriou, Christos-Alexandros Psomas, and Aviad Rubinstein. Reductions in PPP. *Inf. Process. Lett.*, 145:48–52, 2019. doi:10.1016/j.ipl.2018.12.009.

6 Nikhil Bansal, Shashwat Garg, Jesper Nederlof, and Nikhil Vyas. Faster space-efficient algorithms for subset sum, k-sum, and related problems. *SIAM J. Comput.*, 47(5):1755–1777, 2018. doi:10.1137/17M1158203.

7 Paul Beame, Raphaël Clifford, and Widad Machmouchi. Element distinctness, frequency moments, and sliding windows. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 290–299, 2013. doi:10.1109/FOCS.2013.39.

8 Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 364–385. Springer, 2011. doi:10.1007/978-3-642-20465-4_21.

9 Xavier Bonnetain, Rémi Bricout, André Schrottenloher, and Yixin Shen. Improved classical and quantum algorithms for subset-sum. In *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 633–666. Springer, 2020. doi:10.1007/978-3-030-64834-3_22.

10 Karl Bringmann and Konstantinos Panagiotou. Efficient sampling methods for discrete distributions. *Algorithmica*, 79(2):484–508, 2017. doi:10.1007/S00453-016-0205-0.

11 Lijie Chen, Ce Jin, R. Ryan Williams, and Hongxun Wu. Truly low-space element distinctness and subset sum via pseudorandom hash functions. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1661–1678, 2022. doi:10.1137/1.9781611977073.67.

12 Xi Chen, Yaonan Jin, Tim Randolph, and Rocco A. Servedio. Average-case subset balancing problems. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 743–778. SIAM, 2022. doi:10.1137/1.9781611977073.33.

13 Xi Chen, Yaonan Jin, Tim Randolph, and Rocco A. Servedio. Subset sum in time $2^{n/2}$ / poly(n). In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2023, September 11-13, 2023, Atlanta, Georgia, USA*, volume 275 of *LIPIcs*, pages 39:1–39:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.APPROX/RANDOM.2023.39.

14 Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM*, 21(2):277–292, 1974. doi:10.1145/321812.321823.

15 Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 235–256. Springer, 2010. doi:10.1007/978-3-642-13190-5_12.

16 Xin Lyu and Weihao Zhu. Time-space tradeoffs for element distinctness and set intersection via pseudorandomness. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 5243–5281. SIAM, 2023. doi:10.1137/1.9781611977554.ch190.

17 Marcin Mucha, Jesper Nederlof, Jakub Pawlewicz, and Karol Węgrzycki. Equal-subset-sum faster than the meet-in-the-middle. In *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 73:1–73:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.73.

18 Jesper Nederlof. A short note on merlin-arthur protocols for subset sum. *Inf. Process. Lett.*, 118:15–16, 2017. doi:10.1016/j.ipl.2016.09.002.

19 Jesper Nederlof and Karol Węgrzycki. Improving Schroeppel and Shamir's algorithm for subset sum via orthogonal vectors. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1670–1683. ACM, 2021. doi:10.1145/3406325.3451024.

20 Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994. doi:10.1016/S0022-0000(05)80063-7.

21 Katerina Sotiraki, Manolis Zampetakis, and Giorgos Zirdelis. PPP-completeness with connections to cryptography. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 148–158. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00023.

22 Henk C. A. van Tilborg. An upper bound for codes in a two-access binary erasure channel (corresp.). *IEEE Trans. Inf. Theory*, 24(1):112–116, 1978. doi:10.1109/TIT.1978.1055814.

Fully Dynamic Strongly Connected Components in Planar Digraphs

Adam Karczmarz
University of Warsaw, Poland
IDEAS NCBR, Warsaw, Poland

Marcin Smulewicz
University of Warsaw, Poland

Abstract

In this paper we consider maintaining strongly connected components (SCCs) of a directed planar graph subject to edge insertions and deletions. We show a data structure maintaining an implicit representation of the SCCs within $\widetilde{O}(n^{6/7})$ worst-case time per update. The data structure supports, in $O(\log^2 n)$ time, reporting vertices of any specified SCC (with constant overhead per reported vertex) and aggregating vertex information (e.g., computing the maximum label) over all the vertices of that SCC. Furthermore, it can maintain global information about the structure of SCCs, such as the number of SCCs, or the size of the largest SCC.

To the best of our knowledge, no fully dynamic SCCs data structures with sublinear update time have been previously known for any major subclass of digraphs. Our result should be contrasted with the $n^{1-o(1)}$ amortized update time lower bound conditional on SETH, which holds even for dynamically maintaining whether a general digraph has more than two SCCs.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases dynamic strongly connected components, dynamic strong connectivity, dynamic reachability, planar graphs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.95

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2406.10420

Funding *Adam Karczmarz*: Partially supported by the ERC CoG grant TUgbOAT no 772346 and the National Science Centre (NCN) grant no. 2022/47/D/ST6/02184.

1 Introduction

Two vertices of a directed graph $G = (V, E)$ are called strongly connected if they can reach each other using paths in G. Pairwise strong connectivity is an equivalence relation and the strongly connected components (SCCs) of G are its equivalence classes. Computing the SCCs is among the most classical and fundamental algorithmic problems on digraphs and there exists a number of linear-time algorithms for that [14, 33, 35]. Therefore, it is no surprise that *maintaining* SCCs has been one of the most actively studied problems on *dynamic directed graphs* [1, 2, 3, 4, 5, 7, 8, 18, 23, 25, 30, 31].

When maintaining the strongly connected components, the information we care about may vary. First, we could be interested in efficiently answering *pairwise* strong connectivity queries: given $u, v \in V$, decide whether u and v are strongly connected. Pairwise strong connectivity queries, however, cannot easily provide any information about the *global* structure of SCCs (such as the number of SCCs, the size of the largest SCC). Neither they enable, e.g., listing the vertices strongly connected to some $u \in V$. This is why, in the following, we distinguish between *dynamic pairwise strong connectivity* and *dynamic SCCs* data structures which

provide a more global view. In particular, all the data about the SCCs can be easily accessed if the SCCs are maintained *explicitly*, e.g., if the SCC identifier of every vertex is stored at all times and explicitly updated.

1.1 Previous work

In the following, let $n = |V|$ and $m = |E|$. Dynamic graph data structures are traditionally studies in *incremental*, *decremental* or *fully dynamic* settings, which permit the graph to evolve by either only edge insertions, only deletions, or both, respectively. A decremental data structure maintaining SCCs with near-optimal total update time is known [5]. Very recently, a deterministic data structure with $m^{1+o(1)}$ total update time has been obtained also for the incremental setting [8]. Both these state-of-the art data structures maintain the SCCs explicitly.

The *fully dynamic* variant – which is our focus in this paper – although the most natural, has been studied the least. First of all, there is strong evidence that a non-trivial dynamic SCCs data structure for sparse graphs cannot exist. If the SCCs have to be maintained explicitly, then a single update can cause a rather dramatic $\Omega(n)$-sized amortized change in the set of SCCs[1]. As a result, an explicit update may be asymptotically as costly as recomputing SCCs from scratch. This argument – applicable also for maintaining *connected components* of an undirected graph – does not exclude the possibility of maintaining an *implicit* representation of the SCCs in sublinear time, though. After all, there exist very efficient fully dynamic connectivity data structures, e.g., [20, 21, 38], typically maintaining also an explicit spanning forest which allows retrieving any "global" component-wise information one can think of rather easily. However, Abboud and Vassilevska Williams [1] showed that even for maintaining a single-bit information whether G has *more than two SCCs*, a data structure with $O(n^{1-\epsilon})$ amortized time is unlikely, as it would break the Orthogonal Vectors conjecture implied by the SETH [22, 37].[2] This considerably limits the possible global information about the SCCs that can be maintained within sublinear time per update.

For denser graphs, Abboud and Vassilevska Williams [1] also proved that maintaining essentially any (even pairwise) information about SCCs dynamically within truly subquadratic update time has to rely on fast matrix multiplication. And indeed, that pairwise strong connectivity can be maintained this way follows easily from the dynamic matrix inverse-based dynamic *st*-reachability data structures [36, 32]. More recently, we [25] showed that in fact SCCs can be maintained explicitly in $O(n^{1.529})$ worst-case time per update. They also proved that maintaining whether G has just a single SCC (*dynamic SC*) is easier[3] and can be achieved within $O(n^{1.406})$ worst-case time per update. Both these bounds are tight conditional on the appropriate variants [36] of the OMv conjecture [19].

In summary, the complexity of maintaining SCCs in general directed graphs is rather well-understood now. In partially dynamic settings, the known bounds are near optimal unconditionally, whereas in the fully dynamic setting, the picture appears complete unless some popular hardness conjectures are proven wrong. In particular, for general sparse digraphs, no (asymptotically) non-trivial fully dynamic SCCs data structure can exist.

---

[1] Consider a directed cycle and switching its arbitrary single edge on and off.

[2] In [1], a conditional lower bound of the same strength is also derived for the dynamic #SSR problem where the goal is to dynamically count vertices reachable from a source $s \in V$.

[3] Interestingly, the SETH-based lower bound of [1] does not apply to the dynamic SC problem.

Planar graphs. It is thus natural to ask whether non-trivial dynamic SCCs data structures are possible if we limit our attention to some significant class of sparse digraphs. And indeed, this question has been partially addressed for *planar digraphs* in the past. Since pairwise s,t-strong connectivity queries reduce to two s,t-reachability queries, the known planar dynamic reachability data structures [9, 34] imply that sublinear ($\widetilde{O}(n^{2/3})$ or $\widetilde{O}(\sqrt{n})$-time, depending on whether embedding-respecting insertions are required) updates/queries are possible for *pairwise* strong connectivity. Another trade-off for dynamic pairwise strong connectivity has been showed by Charalampopoulos and Karczmarz [6]. Namely, they showed a fully dynamic data structure for planar graphs with $\widetilde{O}(n^{4/5})$ worst-case update time that can produce an identifier s_v of an SCC of a given query vertex v in $O(\log^2 n)$ time. Whereas this is slightly more general[4], it still not powerful enough to enable efficiently maintaining any of the global data about the SCCs of a dynamic planar digraph such as the SCCs count.

To the best of our knowledge, the question whether a more robust – that is, giving a more "global" perspective on the SCCs beyond only supporting pairwise queries – fully dynamic SCCs data structure for planar digraphs (or digraphs from any other interesting class) with sublinear update time is possible has not been addressed before.

1.2 Our results

In this paper, we address the posed question in the case of planar directed graphs. Specifically, our main result is a dynamic SCCs data structure summarized by the following theorem.

▶ **Theorem 1.** *Let G be a planar digraph subject to planarity-preserving edge insertions and deletions. There exists a data structure maintaining the strongly connected components of G implicitly in $\widetilde{O}(n^{6/7})$ worst-case time per update. Specifically:*

- *The data structure maintains the number of SCCs and the size of the largest SCC in G.*
- *For any query vertex v, in $O(\log^2 n)$ time the data structure can compute the size of the SCC of v, and enable reporting the elements of the SCC of v in $O(1)$ worst-case time per element.*

In particular, Theorem 1 constitutes the first known fully dynamic SCCs data structure with sublinear update time for any significant class of sparse digraphs. It also shows that the conditional lower bound of [1] does not hold in planar digraphs.

The data structure of Theorem 1 is deterministic and does not require the edge insertions to respect any fixed embedding of the graph (this also applies to side results discussed below). Obtaining more efficient data structures for fully dynamic embedding-respecting updates is an interesting direction (see, e.g., [9]) that is beyond the scope of this paper.

Related problems. Motivated by the discrepancies between the known bounds for dynamic SCCs and dynamic SC in general digraphs (both from the lower- [1] and upper bounds [25] perspective), we also complement Theorem 1 with a significantly simpler and faster data structure suggesting that the dynamic SC might be easier (than dynamic SCCs) in planar digraphs as well.[5]

[4] Than answering pairwise strong connectivity queries. Using the SCC-identifiers, one can, e.g., partition any k vertices of G into strongly connected classes in $\widetilde{O}(k)$ time, whereas using pairwise queries this requires $\Theta(k^2)$ queries.

[5] Clearly, one could use Theorem 1 for dynamic SC as well.

▶ **Lemma 2.** *Let G be a planar digraph subject to planarity-preserving edge insertions and deletions. One can maintain whether G has a single SCC in $\widetilde{O}(n^{2/3})$ worst-case time per update.*

Similarly, one could ask how *dynamic #SSR* (i.e., counting vertices reachable from a single source) relates to dynamic SCCs in planar digraphs. Especially since:

(1) in general directed graphs, dynamic SCCs and dynamic #SSR currently have matching lower- [1, 36] and upper bounds [25, 32] (up to polylogarithmic factors);

(2) the former problem is at least as hard as the latter in the sense that dynamic #SSR reduces to dynamic SCCs in general graphs easily[6], whereas an opposite reduction is not known.

Unfortunately, the aforementioned reduction of dynamic #SSR to dynamic SCCs breaks planarity rather badly. Interestingly, the *path net* technique we develop to obtain Theorem 1 does not seem to work for counting "asymmetric" reachabilities from a single source.

Nevertheless, the Voronoi diagram machinery developed for computing the diameter of a planar graph [15] almost immediately yields a more efficient data structure for dynamic #SSR in planar digraphs with $\widetilde{O}(n^{4/5})$ update time. We provide the details of that construction in the full version of this extended abstract.

It is worth noting that Voronoi diagrams-based techniques (as developed for distance oracles [16]) have been used in the pairwise strong connectivity data structure [6]. However, as we discuss later on, it is not clear how to apply those for the dynamic SCCs problem. This is why Theorem 1 relies on a completely different path net approach developed in this paper.

1.3 Organization

We review some standard planar graph tools in Section 2. Then, as a warm-up, we show the data structure for dynamic SC in Section 3. In Section 4 we define a path net data structure and show how it can be used to obtain a dynamic SCCs data structure. Finally, in Section 5 we describe the path net data structure. Due to space limit, some details and proofs are deferred to the full version.

2 Preliminaries

In this paper we deal with *directed* graphs. We write $V(G)$ and $E(G)$ to denote the sets of vertices and edges of G, respectively. We omit G when the graph in consideration is clear from the context. A graph H is a *subgraph* of G, which we denote by $H \subseteq G$, iff $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. We write $e = uv \in E(G)$ when referring to edges of G. By G^{R} we denote G with edges reversed.

A sequence of vertices $P = v_1 \dots v_k$, where $k \geq 1$, is called an $s \to t$ path in G if $s = v_1$, $v_k = t$ and there is an edge $v_i v_{i+1}$ in G for each $i = 1, \dots, k-1$. We sometimes view a path P as a subgraph of G with vertices $\{v_1, \dots, v_k\}$ and (possibly zero) edges $\{v_1v_2, \dots, v_{k-1}v_k\}$. For convenience, we sometimes consider a single edge uv a path. If P_1 is a $u \to v$ path and P_2 is a $v \to w$ path, we denote by $P_1 \cdot P_2$ (or simply P_1P_2) a path obtained by concatenating P_1 with P_2. A vertex $t \in V(G)$ is *reachable* from $s \in V(G)$ if there is an $s \to t$ path in G.

[6] Consider the graph G' obtained from G by adding a supersink t with a single outgoing edge ts and incoming edges vt for all $v \in V$. Then $v \in V$ is reachable from s in G iff s and v are strongly connected in G'. See also [36].

Planar graph toolbox. An *r-division* [13] $\mathcal{R}$ of a planar graph, for $r \in [1, n]$, is a decomposition of the graph into a union of $O(n/r)$ pieces P, each of size $O(r)$ and with $O(\sqrt{r})$ boundary vertices (denoted ∂P), i.e., vertices shared with some other piece of $\mathcal{R}$. We denote by $\partial\mathcal{R}$ the set $\bigcup_{P\in\mathcal{R}} \partial P$. If additionally G is plane-embedded, all pieces are connected, and the boundary vertices of each piece P of the r-division $\mathcal{R}$ are distributed among $O(1)$ faces of P that contain the vertices from ∂P *exclusively* (also called holes of P), we call $\mathcal{R}$ an *r-division with few holes*. Klein [27] showed that an r-division with few holes of a triangulated graph can be computed in linear time.

Fully dynamic r-divisions. Many dynamic algorithms for planar graphs maintain r-divisions and useful piecewise auxiliary data structures under dynamic updates. Let us slightly generalize the definition of an r-division with few holes to non-planar graphs by dropping the requirement that G as a whole is planar but retaining all the other requirements (in particular, the individual pieces are plane-embedded).

▶ **Theorem 3** ([6, 28, 34]). *Let $G = (V, E)$ be a weighted planar graph that undergoes edge deletions and edge insertions (assumed to preserve the planarity of G). Let $r \in [1, n]$.*

There is a data structure maintaining an r-division with few holes $\mathcal{R}$ of some G^+, where G^+ can be obtained from G by adding edges[7]*, such that each piece $P \in \mathcal{R}$ is accompanied with some auxiliary data structures that can be constructed in $T(r)$ time given P and use $S(r)$ space.*

The data structure uses $O\left(n + \frac{n}{r} \cdot S(r)\right)$ space and can be initialized in $O\left(n + \frac{n}{r} \cdot T(r)\right)$ time. After each edge insertion/deletion, it can be updated in $O(r + T(r))$ worst-case *time.*

3 Fully dynamic SC data structure

To illustrate the general approach and introduce some of the concepts used for obtaining Theorem 1, in this section we first prove Lemma 2. That is, we show that the information whether a planar graph G is strongly connected can be maintained in $\widetilde{O}(n^{2/3})$ time per update.

We build upon the following general template used previously for designing fully dynamic data structures supporting reachability, strong connectivity, and shortest paths queries in planar graphs, e.g., [34, 28, 11, 6, 24]. As a base, we will maintain dynamically an r-division with few holes $\mathcal{R}$ of G using Theorem 3 with auxiliary piecewise data structures to be fixed later. Intuitively, as long as the piecewise data structures are powerful enough to allow recomputing the requested graph property (e.g., strong connectivity, shortest path between a fixed source/target pair) while spending $r^{1-\epsilon}$ time per piece, for some choice of r we get a sublinear update bound of $\widetilde{O}(n/r^{\epsilon} + r + T(r))$. For example, if $T(r) = O(r^{9.9})$ and $\epsilon = 0.1$, for $r = n^{0.1}$ we get $\widetilde{O}(n^{0.99})$ worst-case update time bound.

Reachability certificates. Subramanian [34] described *reachability certificates* that sparsify reachability between a subset of vertices lying on $O(1)$ faces of a plane digraph G into a (non-necessarily planar) digraph of size near-linear in the size of the subset in question. Formally, we have the following.

[7] Note that G^+ need not be planar.

▶ **Lemma 4** ([34]). *Let H be a plane digraph with a distinguished set $\partial H \subseteq V(H)$ lying on some $O(1)$ faces of H. There exists a directed graph X_H, where $\partial H \subseteq V(X_H)$, of size $\widetilde{O}(|\partial H|)$ satisfying the following property: for any $u, v \in \partial H$, a path $u \to v$ exists in H if and only if there exists a $u \to v$ path in X_H. The graph X_H can be computed in $\widetilde{O}(|H|)$ time.*

▶ Remark 5. For Lemma 4 to hold, it is enough that ∂H lies on $O(1)$ Jordan curves in the plane, each of them having the embedding of H entirely (but not strictly) on one side of the curve. In particular, it is enough that ∂H lies on $O(1)$ faces of some plane supergraph H' with $H \subseteq H'$.

Roughly speaking, Subramanian [34] uses reachability certificates as auxiliary data structures in Theorem 3 in order to obtain a fully dynamic reachability data structure. Crucially, the union of the piecewise certificates preserves pairwise reachability between the boundary vertices $\partial\mathcal{R}$, or more formally (see e.g. [6] for a proof):

▶ **Lemma 6.** *For any $u, v \in \partial\mathcal{R}$, u can reach v in G if and only if u can reach v in $X = \bigcup_{P\in\mathcal{R}} X_P$.*

Strong connectivity data structure. The union of certificates X preserves reachability, and thus strong connectivity between the vertices $\partial\mathcal{R} := \bigcup_{P\in\mathcal{R}} \partial P$. As a result, if G is strongly connected, then so is $\partial\mathcal{R}$ in X. But the reverse implication might not hold. It turns out that for connected graphs, to have an equivalence, it is enough to additionally maintain, for each piece P, whether P is strongly connected conditioned on whether ∂P is strongly connected in G.

In the following, we give a formal description of the data structure. As already said, the data structure maintains a dynamic r-division $\mathcal{R}^+$ of a supergraph G^+ of G (i.e., the input graph), as given by Theorem 3. Since $G \subseteq G^+$, the pieces $\{P^+ \cap G : P^+ \in \mathcal{R}^+\}$ induce an r-division $\mathcal{R}$ of G; however, the boundary ∂P of a piece $P \in \mathcal{R}$ does not necessarily lie on $O(1)$ faces of P, so $\mathcal{R}$ is not technically an r-division with few holes. Nevertheless, ∂P still lies on $O(1)$ faces of a plane supergraph P^+ of P that do not contain vertices outside ∂P. Consequently, by Remark 5, we can still use Lemma 4 to construct a sparse reachability certificate for the piece $P \in \mathcal{R}$. For obtaining Lemma 2, we do not require anything besides beyond that, so for simplicity and wlog. we can assume we work with $\mathcal{R}$ instead of $\mathcal{R}^+$.

While $\mathcal{R}$ evolves, each piece P is accompanied with a reachability certificate X_P of Lemma 4. Note that since $|\partial P| = O(\sqrt{r})$, X_P has size $\widetilde{O}(\sqrt{r})$ and can be constructed in $\widetilde{O}(r)$ time. Moreover, for each P, let $C_{\partial P}$ be a directed simple cycle on the vertices ∂P. We additionally store the (1-bit) information whether the graph $P \cup C_{\partial P}$ is strongly connected. Clearly, this can be computed in $O(|P|) = O(r)$ time. All the accompanying data structures of a piece $P \in \mathcal{R}$ can be thus constructed in $\widetilde{O}(r)$ time. Therefore, by Theorem 3, they are maintained in $\widetilde{O}(r)$ time per update.

Finally, in a separate data structure, we maintain whether G is connected (in the undirected sense). This can be maintained within $n^{o(1)}$ worst-case update time deterministically even in general graphs [17]; in our case, also a less involved data structure such as [12] would suffice.

After $\mathcal{R}$ and the accompanying data structures are updated, strong connectivity of G can be verified as follows. First of all, the union X of all X_P, $P \in \mathcal{R}$, is formed. Note that we can test whether the vertices $\partial\mathcal{R}$ are strongly connected in X in $O(|X|) = \widetilde{O}(n/\sqrt{r})$ time by computing the strongly connected components $\mathcal{S}_X$ of X using any classical linear time algorithm. If G is not connected, or $\partial\mathcal{R}$ is not strongly connected in X, we declare G not strongly connected. If, on the other hand, $\partial\mathcal{R}$ is strongly connected in X, we simply

check whether $P \cup C_{\partial P}$ is strongly connected for each $P \in \mathcal{R}$ and if so, declare G strongly connected. This takes $O(n/r)$ time. Thus, testing strong connectivity takes $\widetilde{O}(n/\sqrt{r})$ time. The following lemma establishes the correctness.

▶ **Lemma 7.** *G is strongly connected if and only if G is connected, the vertices $\partial\mathcal{R}$ are strongly connected in X, and for all $P \in \mathcal{R}$, $P \cup C_{\partial P}$ is strongly connected.*

Proof. First suppose that G is strongly connected. Then, G is clearly connected. Moreover, by Lemma 6, $\partial\mathcal{R}$ is strongly connected in X. For contradiction, suppose that for some $P \in \mathcal{R}$ and $u, v \in V(P)$, u cannot reach v in $P \cup C_{\partial P}$. By strong connectivity of G, there exists some path $Q = u \to v$ in G. Since u cannot reach v in P, Q is not fully contained in P. As a result, Q can be expressed as $Q_1 \cdot R \cdot Q_2$, where $Q_1 = u \to a, Q_2 = b \to v$ are fully contained in P, and $a, b \in \partial P$. But there is a path $Z = a \to b$ in $C_{\partial P}$, so there is a $u \to v$ path $Q_1 \cdot Z \cdot Q_2$ in $P \cup C_{\partial P}$, a contradiction.

Now consider the " $\Longleftarrow$ " direction. Suppose G is connected, the vertices $\partial\mathcal{R}$ are strongly connected in X, and for all $P \in \mathcal{R}$, $P \cup C_{\partial P}$ is strongly connected. By Lemma 6, $\partial\mathcal{R}$ is strongly connected in G. Consider any $P \in \mathcal{R}$ and let $x, y \in V(P)$. We first prove that there exists a path $x \to y$ in G. Indeed, if an $x \to y$ path exists in P, it also exists in G. Otherwise, since $P \cup C_{\partial P}$ is strongly connected, there exists a path $Q = x \to y$ in $P \cup C_{\partial P}$ that can be expressed as $Q_1 \cdot R \cdot Q_2$, where $Q_1 = x \to a$ and $Q_2 = b \to y$ are fully contained in P and $a, b \in \partial P$. But since $a, b \in \partial\mathcal{R}$, by strong connectivity of $\partial\mathcal{R}$, there exists a path $R' = a \to b$ in G. Since $Q_1, Q_2 \subseteq G$, $Q_1 \cdot R' \cdot Q_2$ is an $x \to y$ path in G.

Now take arbitrary $u, v \in V(G)$. If there exists a piece in $\mathcal{R}$ containing both u and v, then we have already proved that there exists a path $u \to v$ in G. Otherwise, let P_u, P_v, $P_u \neq P_v$, be some pieces of $\mathcal{R}$ containing u, v, respectively. We have $P_u \neq G$ and $P_v \neq G$. Since G is connected, P_u has at least one boundary vertex $a \in \partial P_u$. Similarly, P_v has at least one boundary vertex $b \in \partial P_v$. We have proved that there exist paths $u \to a$ and $b \to v$ in G. But also $a, b \in \partial\mathcal{R}$, by the strong connectivity of $\partial\mathcal{R}$, there exists a path $a \to b$ in G as well. We conclude that there exists a path $u \to v$ in G. Since u, v were arbitrary, G is indeed strongly connected. ◀

The worst-case update time of the data structure is $\widetilde{O}(r + n/\sqrt{r}) + n^{o(1)}$. By setting $r = n^{2/3}$, we obtain Lemma 2.

4 Dynamic strongly connected components

The approach we take for maintaining strong connectivity in planar graphs does not easily generalize even to dynamic SCCs counting. This is the case for the following reason. Even if the piece P is fixed (static), there can be possibly an exponential number of different assignments of the vertices ∂P to the SCCs in G (when the other pieces are subject to changes), whereas for dynamic SC, a non-trivial situation arises only when all of ∂P lies within a single SCC. In order to achieve sublinear update time, for any assignment we need to be able to count the SCCs fully contained in P in time sublinear in r after preprocessing P in only *polynomial* (and not exponential) time.

The following notion will be crucial for all our developments.

▶ **Definition 8.** *Let $P \in \mathcal{R}$, and let $A \subseteq \partial P$. A* path net $\Pi_P(A)$ *induced by A is the set of vertices of P that lie on some directed path in P connecting some two elements of A.*

In other words, the path net $\Pi_P(A)$ contains vertices $v \in V(P)$ such that v can reach A and can be reached from A in P. We call a path net $\Pi_P(A)$ *closed* if $A = \Pi_P(A) \cap \partial P$, that is, there are no boundary vertices of P outside A that can reach and can be reached from A.

The following key lemma relates a piece's path net to the SCCs of G.

▶ **Lemma 9.** *Let S be an SCC of G containing at least one boundary vertex of P, i.e., $S \cap \partial P \neq \emptyset$. Then the path net $\Pi_P(S \cap \partial P)$ is closed and equals $S \cap V(P)$.*

Proof. Let us first argue that $\Pi_P(S \cap \partial P)$ is closed. If it was not, there would exist $b \in \partial P \setminus S$ such that there exist paths $b \to (S \cap \partial P)$ and $(S \cap \partial P) \to b$ in P. It follows that b can reach and be reached from S in G, i.e., b is strongly connected with S. Hence, $b \in S$, a contradiction.

Let $v \in \Pi_P(S \cap \partial P)$. Since v can reach and can be reached from $S \cap \partial P$ in P, then it is indeed strongly connected with S in G, since the vertices S are strongly connected in G. So $v \in S \cap V(P)$.

Now let $v \in S \cap V(P)$. Pick any $b \in S \cap \partial P$ (possibly $b = v$ if $v \in \partial P$). There exists paths $R = v \to b$ and $Q = b \to v$ in G. Note that R has some prefix $R_1 = v \to a$ that is fully contained in P and $a \in \partial P$. Similarly, Q has a suffix $Q_1 = c \to v$ that is fully contained in P and $c \in \partial P$. Since there exists paths $v \to a$, $a \to b$, $b \to c$, $c \to v$ in G, vertices a, b, c are strongly connected in G. So $a, c \in S \cap \partial P$. The paths R_1, Q_1 certify that v can be reached from and can reach $S \cap \partial P$ in P. Therefore, $v \in \Pi_P(S \cap \partial P)$ as desired. ◀

If an SCC S is as in Lemma 9, then since the vertices ∂P might be shared with other pieces of $\mathcal{R}$, $\Pi_P(S \cap \partial P) \setminus \partial P$ constitutes the vertices of S contained *exclusively in the piece P*. As there are only $\widetilde{O}(n/\sqrt{r})$ boundary vertices through all pieces, their affiliation to the SCCs of G can be derived from the (SCCs of the) certificate graph $X = \bigcup_{P \in \mathcal{R}} X_P$ (defined and maintained as in Section 3), i.e., they may be handled efficiently separately. Consequently, being able to efficiently aggregate labels, or report the elements, of the sets of the form $\Pi_P(A) \setminus \partial P$ (where $\Pi_P(A)$ is closed) is the key to obtaining an efficient implicit representation of the SCCs of G, as claimed in Theorem 1. Our main technical contribution (Theorem 10) is a *path net data structure* enabling precisely that. The data structure requires a rather large $\widetilde{O}(r^3)$ preprocessing time but achieves the goal by supporting queries about $A \subseteq \partial P$ in near-optimal $\widetilde{O}(|A|)$ time. Formally, we show:

▶ **Theorem 10.** *Let $P \in \mathcal{R}$ and let $\alpha : V(P) \to \mathbb{R}$ be a weight function. In $\widetilde{O}(r^3)$ time one can construct a data structure supporting the following queries. Given a subset $A \subseteq \partial P$, such that $\Pi_P(A)$ is closed, in $\widetilde{O}(|A|)$ time one can:*

- *create an* iterator *that enables listing elements of $\Pi_P(A) \setminus \partial P$ in $O(1)$ time per element,*
- *aggregate weights over $\Pi_P(A) \setminus \partial P$, i.e., compute $\sum_{v \in \Pi_P(A) \setminus \partial P} \alpha(v)$.*

▶ Remark 11. We do not require using subtractions to compute the aggregate weights. In fact, the data structure of Theorem 10 can be easily modified to aggregate weights coming from any semigroup, e.g., one can compute the max/min weight in $\Pi_P(A) \setminus \partial P$ within these bounds.

Our high-level strategy is to maintain the certificates and path net data structures accompanying individual pieces along with the r-division. Roughly speaking, to obtain the information about the SCCs of G beyond how the partition of $\partial\mathcal{R}$ into SCCs looks like, we will query the path net data structures for each piece P with the sets A equal to the SCCs of X having non-empty intersection with ∂P. We prove Theorem 10 later on, in Section 5.

In the remaining part of the section, we explain in detail how, equipped with Theorem 10, a dynamic (implicit) strongly connected components data structure can be obtained. As in Section 3, we maintain an r-division $\mathcal{R}$ dynamically, and maintain sparse reachability certificates X_P, along with the set $\mathcal{S}_X$ of SCCs of $X = \bigcup_{P \in \mathcal{R}} X_P$. Moreover, for each P

we store the strongly connected components $\mathcal{S}_P$ of P. Let $\mathcal{S}_{\partial P}$ be the elements of $\mathcal{S}_P$ that contain a boundary vertex, and $\mathcal{S}_{P\setminus\partial P}$ the elements of $\mathcal{S}_P$ that do not. Clearly, we have $\mathcal{S}_P = \mathcal{S}_{\partial P} \cup \mathcal{S}_{P\setminus\partial P}$ and $(\bigcup \mathcal{S}_P) \cap \left(\bigcup \mathcal{S}_{P\setminus\partial P}\right) = \emptyset$.

For each piece $P \in \mathcal{R}$, we additionally store a path net data structure $\mathcal{D}_P$ of Theorem 10 with an appropriately defined weight function (to be picked depending on the application later). Note that for a piece P, all the auxiliary data structures accompanying P that we have defined can be computed in $\widetilde{O}(r^3)$ time. We now consider the specific goals that can be achieved this way.

Finding the largest SCC. Denote by S^* the largest SCC of G. To be able to identify S^*, and e.g., compute its size, we additionally store and maintain the following. For each piece P, we also maintain the largest SCC S^*_P of P. The sizes of all the SCCs of P, in particular the size of S^*_P, can be easily found and stored after computing $\mathcal{S}_P$.

Note that if the largest SCC S^* of G is not contained entirely in any individual piece P (and thus is larger than $\max_{P\in\mathcal{R}} |S^*_P|$), it has to intersect $\partial\mathcal{R}$. More specifically, in this case for each piece P such that $S^* \cap V(P) \neq \emptyset$, we have $S^* \cap \partial P \neq \emptyset$.

Recall that by Lemma 6, $X = \bigcup_{P\in\mathcal{R}} X_P$ preserves the strong connectivity relation on the vertices $\partial\mathcal{R}$. Therefore, if S^* intersects $\partial\mathcal{R}$, it has to contain $B \cap \partial\mathcal{R}$ for some SCC B of X. For any such B, we can compute the size of the SCC S_B of G satisfying $B \cap \partial\mathcal{R} \subseteq S_B$ as follows. First of all, $|S_B \cap \partial\mathcal{R}| = |B \cap \partial\mathcal{R}|$ since B is an SCC of X. It is thus enough to compute, for all $P \in \mathcal{R}$, $|S_B \cap (V(P) \setminus \partial P)|$. Since the sets $V(P) \setminus \partial P$ are pairwise disjoint across the pieces, by adding these values, we will get the desired size $|S_B|$.

We have already argued that if $B \cap \partial P = \emptyset$, then $S_B \cap V(P) = \emptyset$. If, on the other hand, $B \cap \partial P$ is non-empty, by Lemma 9, if we use the weight function $\alpha(v) \equiv 1$ in the piecewise data structures $\mathcal{D}_P$ of Theorem 10, we can compute $|S_B \cap (V(P) \setminus \partial P)| = \sum_{v\in\Pi_P(B\cap\partial P)\setminus\partial P} \alpha(v)$ in $\widetilde{O}(|B \cap \partial P|)$ time using the input set $A := B \cap \partial P$. We conclude that the sizes S_B for all $B \in \mathcal{S}_X$ can be computed in time

$$\widetilde{O}\left(\sum_{B\in\mathcal{S}_X} \sum_{\substack{P\in\mathcal{R}\\ B\cap\partial P\neq\emptyset}} |B \cap \partial P|\right) = \widetilde{O}\left(\sum_{P\in\mathcal{R}} \sum_{\substack{B\in\mathcal{S}_X\\ B\cap\partial P\neq\emptyset}} |B \cap \partial P|\right) = \widetilde{O}\left(\sum_{P\in\mathcal{R}} |\partial P|\right) = \widetilde{O}(n/\sqrt{r}). \tag{1}$$

Finally, S^* is either equal to the largest S_B for $B \in \mathcal{S}_X$, or the largest S^*_P (through $P \in \mathcal{R}$). The latter is the case if $\max_{B\in\mathcal{S}_X} |S_B| < \max_{P\in\mathcal{R}} |S^*_P|$. Which case we fall into is easily decided once all the $O(n/\sqrt{r})$ sizes $|S_B|$ are computed.

Accessing the SCC of a specified vertex. Suppose first that we know the SCC S_v of G containing a query vertex v, and additionally whether S_v intersects $\partial\mathcal{R}$. If $S_v \cap \partial\mathcal{R} = \emptyset$, then v is a vertex of a unique piece P, and $S_v \in \mathcal{S}_{P\setminus\partial P}$. In this case, we can clearly compute the size of S_v and report the elements of S_v in $O(1)$ time since $S_v \in \mathcal{S}_P$ is stored explicitly.

Otherwise, if $S_v \cap \partial\mathcal{R} \neq \emptyset$, we reuse the information that we have computed for finding the largest SCC. We have already described how to compute the sizes of all the SCCs S of G intersecting $\partial\mathcal{R}$ (in particular S_v) along with these respective intersections in $\widetilde{O}(n/\sqrt{r})$ time upon update. Moreover, for all $P \in \mathcal{R}$ we have computed $|S \cap (V(P) \setminus \partial P)|$ using the data structure $\mathcal{D}_P$ of Theorem 10. As a result, we can store, for each such S, a subset $L(S) \subseteq \mathcal{R}$ of pieces P such that $S \cap (V(P) \setminus \partial P) \neq \emptyset$. Recall that for all $P \in L(S)$, we can also use $\mathcal{D}_P$ to create an iterator for reporting the elements of $S \cap (V(P) \setminus \partial P)$ in $O(1)$ time per

element. So, in order to efficiently report elements of any such S, we first report the vertices of $S \cap \partial\mathcal{R}$, and then the elements of each $S \cap (V(P) \setminus \partial P)$ for subsequent pieces $P \in L(S)$. Indeed, one needs only $O(1)$ worst-case time to find each subsequent vertex of S.

Finally, we are left with the task of of finding the SCC S_v of a query vertex $v \in V$. It is not clear how to leverage path net data structures for that in the case when S_v intersects $\partial\mathcal{R}$. Instead, we use the data structure of [6] in a black-box way. That data structure handles fully dynamic edge updates in $\widetilde{O}(n^{4/5})$ worst-case time, and provides $O(\log^2 n)$ worst-case time access to consistent (for queries issued between any two subsequent updates) SCC identifiers of individual vertices[8]. Using [6], after every update we find the identifiers I of the SCCs of vertices $\partial\mathcal{R}$ in G in $\widetilde{O}(n/\sqrt{r})$ time. Now, to find the SCC of v upon query, we find the identifier i_v of the SCC containing v in $O(\log^2 n)$ time. If $i_v \in I$ (which can be tested in $O(\log n)$ time), we obtain that v is in an SCC of G intersecting $\partial\mathcal{R}$ and some vertex b_v from the intersection. b_v can be in turn used to access $L(S_v)$ and thus enable reporting the elements of S_v. Otherwise, if $i_v \notin I$, $S \cap \partial\mathcal{R} = \emptyset$ and thus S equals the unique SCC from $S_{P \setminus \partial P}$ containing v in the unique piece P containing v.

Counting strongly connected components. Let us separately count SCCs $\mathcal{S}_{\partial\mathcal{R}}$ that intersect $\partial\mathcal{R}$ and those that do not. The former can be counted in $\widetilde{O}(n/\sqrt{r})$ time by counting the SCCs of X that intersect $\partial\mathcal{R}$ (that we maintain). The latter can be computed as follows. Consider the sum $\Phi = \sum_{P \in \mathcal{R}} |\mathcal{S}_{P \setminus \partial P}|$. If we wanted the sum Φ to count the SCCs not intersecting $\partial\mathcal{R}$, then an SCC $S \in \mathcal{S}_{P \setminus \partial P}$ contributes to the sum unnecessarily precisely when S is not an SCC of G. To see that, note that if an SCC S of P is not an SCC of G, it has to be a part of another SCC S' of G that also contains vertices of other pieces, i.e., S' intersects $\partial\mathcal{R}$. From Lemma 9, we conclude:

▶ **Corollary 12.** *An SCC $S \in \mathcal{S}_{P \setminus \partial P}$ is not an SCC of G iff there exists (precisely one) SCC B of X such that $S \subseteq \Pi_P(B \cap \partial P)$ (or equivalently, such that $S \cap \Pi_P(B \cap \partial P) \neq \emptyset$).*

As a result, we can count the number of SCCs in G that do not intersect $\partial\mathcal{R}$ by subtracting from Φ, for each $P \in \mathcal{R}$, and each $B \in \mathcal{S}_X$ the number $c_{P,B}$ of SCCs in $\mathcal{S}_{P \setminus \partial P}$ that intersect $\Pi_P(B \cap \partial P)$. To this end, we can use the data structure $\mathcal{D}_P$ of Theorem 10 built upon P (and maintained as described before) with a weight function α on $V(P)$ assigning 1 to an arbitrary single vertex v_S of each SCC $S \in \mathcal{S}_{P \setminus \partial P}$, and 0 to all other vertices. By Corollary 12, with such a weight function, $c_{P,B} = \sum_{v \in \Pi_P(B \cap \partial P) \setminus \partial P} \alpha(v)$ can be computed in $\widetilde{O}(|B \cap \partial P|)$ time using $\mathcal{D}_P$. Consequently, similarly as in (1), over all $B \in \mathcal{S}_X$, and $P \in \mathcal{R}$, computing all the values $c_{P,B}$ will take $\widetilde{O}(n/\sqrt{r})$ time. As mentioned before, the SCC count is obtained by subtracting those from Φ and adding the result to the count of $\mathcal{S}_{\partial\mathcal{R}}$.

Depending on the application, the worst-case update time of the data structure is $\widetilde{O}(n/\sqrt{r} + r^3)$ or $\widetilde{O}(n/\sqrt{r} + r^3 + n^{4/5})$. The bound is optimized for $r = n^{2/7}$ and this yields Theorem 1.

5 The path net data structure

This section is devoted to describing the below key component of our dynamic SCCs data structure.

[8] The query time of that data structure can be easily reduced to $O(\log n \cdot \log\log n)$ without affecting the $\widetilde{O}(n^{4/5})$ update bound if one simply replaces the classical MSSP data structure [26] used internally for performing point location queries in additively weighted Voronoi diagrams [16] with the MSSP data structure of [29].

▶ **Theorem 10.** *Let $P \in \mathcal{R}$ and let $\alpha : V(P) \to \mathbb{R}$ be a weight function. In $\widetilde{O}(r^3)$ time one can construct a data structure supporting the following queries. Given a subset $A \subseteq \partial P$, such that $\Pi_P(A)$ is closed, in $\widetilde{O}(|A|)$ time one can:*

- *create an* iterator *that enables listing elements of $\Pi_P(A) \setminus \partial P$ in $O(1)$ time per element,*
- *aggregate weights over $\Pi_P(A) \setminus \partial P$, i.e., compute $\sum_{v \in \Pi_P(A) \setminus \partial P} \alpha(v)$.*

5.1 Overview

Charalampopoulos and Karczmarz [6] showed that for any SCC S of G, $V(P) \cap S$ forms an intersection of two cells coming from two carefully prepared additively weighted Voronoi diagrams on P with sites ∂P. As a result, they could use Voronoi diagram point location mechanism [16, 29] for testing in $O(\operatorname{polylog} n)$ time whether a query vertex lies in such an intersection. If we tried to follow this approach, we would need to be able to aggregate/report vertices in such intersections of cells coming from two seemingly unrelated Voronoi diagrams. This is very different from just testing membership, and it is not clear whether this can be done efficiently.

Instead, in order to prove Theorem 10, we take a more direct approach. As is done typically, we first consider the situation when A lies on a single face of P. In the single-hole case, the first step is to reduce to the case when the input piece P is acyclic; note that if a vertex lies in the path net $\Pi_P(A)$, its entire SCC in P does. Acyclicity and appropriate perturbation [10] allows us to pick a collection of paths $\pi_{u,v}$, for all $u, v \in \partial P$, such that every two paths in the collection are either disjoint or their intersection forms a single subpath of both. This property makes the paths $\pi_{u,v}$ particularly convenient to use for cutting the piece P into smaller non-overlapping parts.

More specifically, the paths $\pi_{u,v}$ are used to partition – using a polygon triangulation-like procedure – a queried net $\Pi_P(A)$ into regions in the plane with vertices $B \subseteq A$ bounded by either fragments of the face of P containing ∂P or some paths $\pi_{u,v}$ for $u, v \in B$ (so-called *base instances*). A base instance has a very special structure guaranteeing that for a given vertex $v \in V(P) \setminus \partial P$, there are *only* $O(1)$ *pairs* $s, t \in B$ such that an $s \to v \to t$ path exists in P. At the end, this crucial property can be used to reduce a base instance query B even further to looking up $\widetilde{O}(|B|)$ preprocessed answers for base instances with at most 5 vertices from ∂P, of which there are at most $\widetilde{O}(|\partial P|^5) = \widetilde{O}(r^{5/2})$. The precomputation of small base instances can be done in $\widetilde{O}(r^3)$ time.

For efficiently implementing the polygon triangulation-like partition procedure – which repeatedly cuts off base instances from the "core" part of the problem – we develop a dynamic data structure for existential reachability queries on a single face of a plane digraph (Lemma 15). To this end, we leverage the single-face reachability characterization of [23].

As the final step, we show a reduction from the case when A can lie on k faces on P, to the case when A lies on $k-1$ faces. The reduction – which eventually reduces the problem to the single-hole case – can blow up the piece's size by a constant factor. However, since there are only $O(1)$ holes initially, the general case can still be solved only constant factor slower than the single-hole case.

5.2 Reducing to the acyclic case

Recall that P is a piece of an r-division with few holes $\mathcal{R}$ and thus has size $O(r)$. Moreover, ∂P lies on $O(1)$ faces (*holes*) of some supergraph P^+ of P and $|\partial P| = O(\sqrt{r})$. For simplicity we will not differentiate between the faces/holes of P and P^+. Whenever we refer to a hole h of P, we mean a closed curve h in the plane such that whole embedding of P lies weakly on a single side of h, and only vertices ∂P may lie on the curve h.

First of all, we compute the strongly connected components $\mathcal{S}_P$ of P. Observe that from the point of view of supported queries, all vertices of a single SCC $S \in \mathcal{S}_P$ are treated exactly the same: when a single vertex $v \in S \setminus \partial P$ is reported (or its weight $\alpha(v)$ is aggregated), all other vertices from $S \setminus \partial P$ are as well. For each $S \in \mathcal{S}_P$ we precompute the aggregate weight $\alpha(S) = \sum_{v \in S \setminus \partial P} \alpha(v)$ in $O(r)$ time. We then contract each strongly connected component $S \in \mathcal{S}_P$ into a single vertex v_S. Crucially, since each subgraph $P[S]$ is connected in the undirected sense, contractions can be performed in an embedding preserving way (via a sequence of single-edge contractions), so that:

- The obtained graph P' is acyclic.
- The vertices $\partial P' := \{v_S : S \in \mathcal{S}_P, S \cap \partial P \neq \emptyset\}$ lie on $O(1)$ faces of (a supergraph of) P'.

Indeed, to see the latter property, label each $b \in \partial P$ initially each with hole that b lies on and make non-boundary vertices unlabeled. When the vertices merge, label the resulting vertex using one of the involved vertices' labels, if any of them has one. At the end there will be still $O(1)$ labels, each vertex of $\partial P'$ will have one of these labels, and for each label, all the vertices holding that label will be incident to a single face of P'. Clearly, $|P'| = O(r)$ and $|\partial P'| = O(\sqrt{r})$.

The proof of the below lemma can be found in the full version.

▶ **Lemma 13.** *Suppose a data structure $\mathcal{D}'$ of Theorem 10 is built for the acyclic graph P' with the weight function $\alpha'(v_S) := \alpha(S) = \sum_{v \in S \setminus \partial P} \alpha(v)$. Then, a query $A \subseteq \partial P$ from Theorem 10 can be reduced, in $O(|A|)$ time, to a query about $A' = \{v_S : S \in \mathcal{S}_P, S \cap A \neq \emptyset\}$ issued to $\mathcal{D}'$.*

In the following we will assume that P is an acyclic graph. We will no longer need the assumption $\Pi_P(A) \cap \partial P = A$; this was only needed for the efficient reduction to the acyclic case. Consequently, we will design a data structure with query time $\widetilde{O}(|A|)$ even if $A \subsetneq \Pi_P(A) \cap \partial P$.

The single-hole assumption. In the remaining part of this section, we assume we only want to handle queries where the query set A lies on a *single hole* h of an acyclic piece P. Due to space constraints, we present the reduction of the general case to the single-hole case in the full version. Our strategy for the single-hole case will be to efficiently break the problem into subproblems for which we have the answer precomputed.

5.3 Picking non-crossing paths

We first fix, for any $u, v \in V(P)$ such that u can reach v, one particular $u \to v$ path $\pi_{u,v} \subseteq P$. We apply the perturbation scheme of [10] to P, so that the shortest paths in P become unique. For any $u, v \in V(P)$ such that u can reach v in P, we define $\pi_{u,v}$ to be the unique shortest $u \to v$ path in P. Note that each $\pi_{u,v}$ is a simple path in P. We have the following crucial property.

▶ **Lemma 14.** *Let $u, v, x, y \in V(P)$ be such that u can reach v and x can reach y in P. If $V(\pi_{u,v}) \cap V(\pi_{x,y}) \neq \emptyset$, then $\pi_{u,v}$ and $\pi_{x,y}$ share a single (possibly zero-edge) subpath.*

Proof. Let a (b) be the first (last, resp.) vertex on $\pi_{u,v}$ that is also a vertex of $\pi_{x,y}$. $\pi_{u,v}$ can be expressed as $Q_1 \cdot T \cdot Q_2$, where $T = a \to b$. Moreover, $V(\pi_{u,v}) \cap V(\pi_{x,y}) \subseteq V(T)$. If $a \neq b$, the vertex b cannot appear before a on $\pi_{x,y}$, because then a and b would lie on a cycle in P, contradicting acyclicity of P. As a result, $\pi_{x,y}$ has an $a \to b$ subpath as well. Since shortest paths in P are unique and shortest paths have optimal substructure, the $a \to b$ subpath of $\pi_{x,y}$ equals T. So, $\pi_{x,y}$ can be expressed as $R_1 \cdot T \cdot R_2$. Since $\pi_{x,y}$ is simple, and $V(\pi_{u,v}) \cap V(\pi_{x,y}) \subseteq V(T)$, we indeed have $\pi_{u,v} \cap \pi_{x,y} = T$. ◀

Observe that since P is a DAG, for $u, v \in V(P)$, $u \neq v$, $\pi_{u,v}$ and $\pi_{v,u}$ cannot both exist. When one of them exists, we will sometimes write $\pi_{\{u,v\}}$ to denote the one that exists.

5.4 Generalization

Before we proceed, we need to state our problem in a more general way that will enable decomposition into smaller subproblems of similar kind. Let us first fix a counterclockwise order of $V(h)$, as follows. Let $u_1, e_1, u_2, e_2, \ldots, u_r, e_r$ be the sequence of vertices and edges on a counterclockwise bounding walk of h with an arbitrary starting point. The encountered vertices u_i might not be all distinct since h can be a non-simple face in general. The order $\prec_h$ of $V(h)$ that we use is obtained by removing duplicate vertices from the sequence in an arbitrary way, i.e., for each $v \in V(h)$ we pick one index i_v such that $u_{i_v} = v$. If $x, y \in V(h)$ and $i_x < i_y$, then we define $C_{x,y}$ to be the curve $e_{i_x} e_{i_x+1} \ldots e_{i_y-1}$. Otherwise, if $i_x > i_y$, then $C_{x,y} = e_{i_x} e_{i_x+1} \ldots e_r e_1 e_2 \ldots e_{i_y-1}$. If h is simple, then $C_{x,y}$ is a simple curve. However, in general, $C_{x,y}$ can be self-touching but it does not cross itself.

Suppose $B \subseteq \partial P \cap V(h)$ is given in the counterclockwise order on h. Moreover, for each pair of neighboring (i.e., appearing consecutively in the *cyclic* order of B given by $\prec_h$) vertices x, y of B, a curve $\Phi_{x,y}$ connecting x to y is given. The curve $\Phi_{x,y}$ equals either $\pi_{\{x,y\}}$ or $C_{x,y}$. Let Φ be the closed curve formed by concatenating the subsequent curves $\Phi_{x,y}$ for all neighboring $x, y \in B$. For brevity, we also extend the notation $\Phi_{a,b}$ to non-neighboring vertices a, b of B, and define $\Phi_{a,b}$ to be a concatenation of the curves $\Phi_{x,y}$ for all neighboring pairs (x, y) between a and b on Φ. We don't require Φ to be a simple (Jordan) curve; parts of it may be overlapping. Whereas Φ is allowed to be self-touching, it does not cross itself, like e.g., the cycle bounding a non-simple face of a plane graph. Let $P[\Phi]$ denote the region of P that lies *weakly* inside the curve Φ.

The objective of the problem (B, Φ) is to aggregate weights of (or report) the vertices of $\Pi_P(B) \setminus \partial P$ that lie *strictly inside* Φ. More formally, we want to aggregate vertices v that lie, at the same time, strictly on the left side of each of the curves $\Phi_{x,y}$ (seen as a curve directed from x to y) where $x, y \in B$ are neighboring in the counterclockwise order on h.[9] Note that with such a defined problem, the original goal of the query procedure can be rephrased as (A, h) since all the vertices of $\Pi_P(A) \setminus \partial P$ indeed lie strictly inside h.

5.5 Preprocessing

Preprocessing for small instances and subproblems.

- For any tuple $B = (b_1, \ldots, b_q)$ of at most 4 vertices of ∂P appearing in that order on in $\prec_h$, and any out of at most $2^q = O(1)$ possible curves Φ that might constitute the problem (B, Φ), we precompute the aggregate weight and the list of vertices in $\Pi_P(B) \setminus \partial P$ that lie strictly inside Φ. This can clearly be done in $\widetilde{O}(|\partial P|^4 \cdot |P|) = \widetilde{O}(r^3)$ time.
- For any 5-tuple $\tau = (b_0, \ldots, b_4)$ ordered by $\prec_h$, and any out of at most 2^4 possible curves $\Phi_{b_i,b_{i+1}} \in \{\pi_{\{b_i,b_{i+1}\}}, C_{b_i,b_{i+1}}\}$, where $i \in \{0, \ldots, 3\}$, we precompute and store the set X_τ of vertices $x \in V(P) \setminus \partial P$ such that:
  1. x lies on some path connecting b_1 and b_2 in P,
  2. x *does not* lie on any path connecting b_0 and b_1 in P,
  3. x lies strictly to the left of each $\Phi_{b_i,b_{i+1}}$ for $i = \{0, \ldots, 3\}$.

[9] Even more formally, assuming P is embedded in such a way that h is the infinite face of a supergraph of P, we are interested in the vertices lying in the intersection of the regions strictly inside closed curves $\Phi_{x,y} \cdot C_{y,x}$ for all neighboring x, y in B.

We also store the aggregate weight $w(X_\tau)$. This preprocessing can be performed in a brute-force way in $\widetilde{O}(|\partial P|^5 \cdot |P|) = \widetilde{O}(r^{7/2})$ time. It can be also optimized fairly easily to $\widetilde{O}(r^3)$ time and space, as shown in the full version.

- For any $u, v \in \partial P$, we store the path $\pi_{\{u,v\}}$ itself, along with aggregate weights of all its subpaths. This data can be computed in $\widetilde{O}(|\partial P|^2 \cdot |P|^2) = \widetilde{O}(r^3)$ time in a brute-force way.
- Finally, for any two pairs $(u, v), (x, y) \in \partial P \times \partial P$, we compute the intersection of the paths $\pi_{\{u,v\}}$ and $\pi_{\{x,y\}}$. By Lemma 14, that intersection is either empty or is a subpath of both these paths. Hence, it is enough to store the two endpoints of the intersection subpath only. The desired intersections can be found in $\widetilde{O}(|\partial P|^4 \cdot |P|) = \widetilde{O}(r^3)$ time in a brute-force way.

Existential reachability data structure. We also build data structures $\mathcal{L}, \mathcal{L}^R$ of the following lemma for P and P^R, respectively.

▶ **Lemma 15.** *In $\widetilde{O}(r)$ time one can construct a data structure maintaining an (initially empty) set $Z \subseteq \partial P \cap V(h)$ and supporting the following operations in $O(\operatorname{polylog} n)$ time:*

- *insert or delete a single $b \in \partial P \cap V(h)$ either to or from Z.*
- *for any query vertex $v \in (\partial P \cap V(h)) \setminus Z$, find any $z \in Z$ (if exists) that v can reach in P.*

Due to space constraints, the proof of Lemma 15 is deferred to the full version.

5.6 Answering queries

We now proceed with the description of our algorithm solving the general problem (B, Φ).

Base case. Let the elements of $B = \{b_1, \ldots, b_k\}$ be sorted according to their order $\prec_h$ on Φ. For convenience, identify b_{k+l} with b_l for any integer l.

We first consider the easier *base case*, with the following additional requirement:

For any $1 \leq i < j \leq k$, if b_i can reach b_j or can be reached from b_j in P, then either $j = i + 1$ or $(i, j) = (1, k)$.

If $k \leq 4$, we will simply return the precomputed aggregate weight, or an iterator to a list of vertices in $\Pi_P(B) \setminus \partial P$ strictly inside Φ. So suppose $k \geq 5$. We start with the following lemma.

▶ **Lemma 16.** *Let $v \in \Pi_P(B) \setminus \partial P$ lie strictly inside Φ. Then:*

- *There exists such i that v lies on some $b_i \to b_{i+1}$ or on some $b_{i+1} \to b_i$ path in P.*
- *Moreover, there is at most one more pair $\{x, y\} \subseteq B$, $\{x, y\} \neq \{b_i, b_{i+1}\}$ such that v lies on some $x \to y$ or $y \to x$ path in P, and either $\{x, y\} = \{b_{i-1}, b_i\}$, or $\{x, y\} = \{b_{i+1}, b_{i+2}\}$.*

Proof. Item (1) follows easily by the additional requirement of the base case. Let v lie on some $b_j \to c$ path in P, where $c \in B$. Since P is acyclic, we have that $b_j \neq c$ and thus $c \in \{b_{j-1}, b_{j+1}\}$. If $c = b_{j+1}$, we put $i = j$ and if $c = b_{j-1}$, we put $i = j - 1$.

For item (2), consider the case when v lies on some $b_i \to b_{i+1}$; the case when v lies on a $b_{i+1} \to b_i$ path is symmetric. Suppose v also lies on some $x \to y$ path in P, where $x, y \in B$, $x \neq y$, $\{x, y\} \neq \{b_i, b_{i+1}\}$. Since b_i can reach y through v, $y \in \{b_{i-1}, b_{i+1}\}$ by the base case requirement. Similarly, since x can reach b_{i+1} through v, $x \in \{b_i, b_{i+2}\}$. We cannot have $x = b_{i+2}$ and $y = b_{i-1}$ since b_{i+2} reaching b_{i-1} is a contradiction with the first base case requirement for $k \geq 5$. As a result, $(x, y) \neq (b_i, b_{i+1})$ implies $(x, y) = (b_i, b_{i-1})$

or $(x, y) = (b_{i+2}, b_{i+1})$. Moreover, v cannot lie on both some $b_i \to b_{i-1}$ path and some $b_{i+2} \to b_{i+1}$ path, since then b_{i+2} could reach b_{i-1}, which is again a contradiction for $k \geq 5$. ◀

By Lemma 16, each $v \in \Pi_P(B) \setminus \partial P$ falls into exactly one of the k sets Y_i, for $i = 1, \ldots, k$, such that Y_i contains those v in $V(P) \setminus \partial P$ that lie on a path in P connecting b_i and b_{i+1} (in any direction), but at the same time *do not* lie on any path connecting b_{i-1} and b_i in P (in any direction). Indeed, if v appears only on paths connecting b_i and b_{i+1}, it will be included in Y_i. On the other hand, if v appears both on paths connecting b_i and b_{i+1} and on paths connecting b_{i+1} and b_{i+2}, it will be included only in Y_i, but not in Y_{i+1}.

▶ **Lemma 17.** *Suppose $v \in Y_i$. Then v lies strictly inside Φ if and only if v lies strictly on the left side of Φ_{b_{i-1},b_i}, $\Phi_{b_i,b_{i+1}}$, $\Phi_{b_{i+1},b_{i+2}}$, and $\Phi_{b_{i+2},b_{i+3}}$.*

Proof. Since strictly inside Φ means strictly to the left of all $\Phi_{b_j,b_{j+1}}$, the " $\Longrightarrow$ " direction is trivial.

Consider the " $\Longleftarrow$ " direction. For contradiction, suppose v does not lie strictly inside Φ. Then, for some $j \notin \{i-1, i, i+1, i+2\}$, v lies weakly to the right of $\Phi_{b_j,b_{j+1}}$. Since $v \in V(P) \setminus \partial P$ and the hole h contains only vertices of ∂P, this means that $\Phi_{b_j,b_{j+1}} = \pi_{\{b_j,b_{j+1}\}}$. Since b_j and b_{j+1} are consecutive in B, and $i \notin \{j, j+1\}$, b_i lies weakly to the left of $\pi_{\{b_j,b_{j+1}\}}$.

By $v \in Y_i$, the vertex v lies on a path Q connecting b_i and b_{i+1} in P. Since v and b_i lie weakly on different sides of $\pi_{\{b_j,b_{j+1}\}}$, the path Q has to cross $\pi_{\{b_j,b_{j+1}\}}$ at a vertex w appearing not later than v on Q (possibly $v = w$). If $Q = b_i \to b_{i+1}$, the existence of w implies that there exists an $s \to b_{i+1}$ path in P going through v such that $s \in \{b_j, b_{j+1}\}$. By $v \in Y_i$, this implies $s \in \{b_i, b_{i+2}\}$. As a result, $j \in \{i-1, i, i+1, i+2\}$, a contradiction. Similarly, if $Q = b_{i+1} \to b_i$, there exists an $s \to b_i$ path in P going through v such that $s \in \{b_j, b_{j+1}\}$. But on the other hand by $v \in Y_i$ we have $s = b_{i+1}$ so $j \in \{i, i+1\}$, a contradiction. ◀

As a result, we can equivalently aggregate vertices v in each Y_i under a (seemingly) weaker requirement that v lies strictly on the left side of Φ_{b_{i-1},b_i}, $\Phi_{b_i,b_{i+1}}$, $\Phi_{b_{i+1},b_{i+2}}$, and $\Phi_{b_{i+2},b_{i+3}}$. But this is, again, part of the precomputed data for the tuple $(b_{i-1}, b_i, b_{i+1}, b_{i+2}, b_{i+3})$.

Consequently, there are k disjoint sets Y_i to consider. We can thus aggregate weights or construct a list for reporting vertices from $v \in \Pi_P(B) \setminus \partial P$ strictly inside Φ in $O(k)$ time. We thus obtain:

▶ **Lemma 18.** *The base case can be handled in $O(|B|)$ time.*

General case. To solve the general case, we reduce it to a number of base case instances. To this end, we maintain a partition of $B = S \cup T$ such that S precedes T on Φ. Let $S = \{s_1, \ldots, s_p\}$ and $T = \{t_1, \ldots, t_q\}$. We will gradually simplify the problem while maintaining the following invariants:

(1) For any $u, v \in B$, if u can reach v in P, then $\pi_{u,v} \subseteq P[\Phi]$.
(2) For any $1 \leq i < j \leq p$, if s_i can reach s_j or can be reached from s_j, then $j = i + 1$.
(3) If $x, y \in B$ are neighbors in the counterclockwise order $\prec_h$ on Φ and $\Phi_{x,y} = \pi_{\{x,y\}}$, then $x, y \in S$.
(4) In the data structures $\mathcal{L}, \mathcal{L}^R$ of Lemma 15, the set Z satisfies $Z = S \setminus \{s_p\}$.

The algorithm will gradually modify B, S, T, Φ until we have $S = B$ and $T = \emptyset$. Note that when $T = \emptyset$, $(B, \Phi) = (S, \Phi)$ satisfies the requirement of the base case and can be solved in $\widetilde{O}(|B|)$ time.

Initially we put $\Phi = h$, $S = \{b_1, b_2\}$, and insert b_1 to Z to satisfy the invariants.

The main loop of the procedure runs while $T \neq \emptyset$ and does the following. Using $\mathcal{L}$ and $\mathcal{L}^R$, in $O(\text{polylog}\, n)$ time we test whether t_1 can reach Z or can be reached from Z in P. If this is not the case, we simply move t_1 to the end of S, an update Z accordingly. Note that $|T|$ decreases then.

Otherwise, we can find all vertices $X = \{x_1, \ldots, x_l\} \subseteq Z$ that t_1 can reach or can be reached from in $\widetilde{O}(|X|)$ by repeatedly extracting them from the data structures $\mathcal{L}, \mathcal{L}^R$. Additionally we sort X so that the order $x_1, \ldots, x_l$ matches the order of S on Φ. Let π_i denote the path $\pi_{\{t_1, x_i\}}$ possibly reversed to go from t_1 to x_i, and π_i^R denote the reverse π_i going from x_i to t_1. The vertices X are used to "cut off" l base case instances, as follows. For $i = 0, \ldots, l$, let S_i denote the vertices of S between x_i and x_{i+1} on Φ (inclusive), where we set $x_{l+1} := s_p$, $x_0 := s_1$. We split the problem (B, Φ) into the following subproblems (see Figure 1 for better understanding):

1. For each $i = 1, \ldots, l-1$, an instance $(S_i \cup \{t_1\}, \Phi_{x_i, x_{i+1}} \cdot \pi_{i+1}^R \cdot \pi_i)$.
2. An instance $(S_l \cup \{t_1\}, \Phi_{x_l, t_1} \cdot \pi_l)$.
3. An instance $(S_0 \cup T, \Phi_{s_1, x_1} \cdot \pi_1^R \cdot \Phi_{t_1, s_1})$.

▶ **Lemma 19.** *For each of the above subproblems (B', Φ'), and $u, v \in B'$, if u can reach v in P, then $\pi_{u,v} \subseteq P[\Phi']$.*

Proof. Note that (B', Φ') is obtained from (B, Φ) by cutting it out of (B, Φ) with at most two paths $\pi_{\{t_1, a\}}, \pi_{\{t_1, b\}}$, for some $t_1, a, b \in B' \subseteq B$. By our assumption, $\pi_{u,v} \subseteq P[\Phi]$. As a result, if $\pi_{u,v}$ was not contained in $P[\Phi']$, then $\pi_{u,v}$ would need to cross either $\pi_{\{t_1,a\}}$ or $\pi_{\{t_1,b\}}$. However, this is impossible by Lemma 14 and since $u, v \in V(P[\Phi'])$. ◀

▶ **Lemma 20.** *The obtained instances of types 1 and 2 above fall into the base case.*

Proof. To see that the base case requirement is satisfied, recall that by the invariant posed on S, if two elements of S_i, where $i \in \{1, \ldots, l\}$, are related (wrt. reachability in P), they are neighboring in S_i. By construction, t_1 can be only related to the first and the last element of S_i. ◀

Since the cutting is performed using non-crossing paths in P, the regions $P[\Phi']$ for the obtained subproblems can only share their boundaries, that is, if some v is strictly inside one of the curves Φ', then it it is not strictly inside another obtained curve Φ''. Therefore, if we proceeded with the above subproblems recursively, we would not aggregate or report any vertex of $v \in \Pi_P(B) \setminus \partial P$ twice. However, we still need to report/aggregate vertices that lie on paths $\pi_1, \ldots, \pi_l$ strictly inside the curve Φ. We now discuss how this strategy is implemented. Let us first consider solving the subproblems recursively. We handle all the obtained base case instance of types 1 and 2 as explained before. If $x_1 = s_g$, then by Lemma 18, the total time required for this is $O\left(\sum_{i=1}^{l}(|S_i| + 1)\right) = O((p-g) + l)$. But note that $l \leq p - g$, so in fact the bound is $O(p - g)$.

To handle the instance $(S_0 \cup T, \Phi_{s_1, x_1} \cdot \pi_1^R \cdot \Phi_{t_1, s_1})$, we simply replace (B, Φ) with it and proceed with solving it using the algorithm for the general case. To this end, we set $S := S_0 \cup \{t_1\}$, $T := \{t_2, \ldots, t_q\}$ and update Z in the data structures $\mathcal{L}, \mathcal{L}^R$ to S_0 by removing elements. Then, invariant (1) is satisfied by Lemma 19, and invariants (2), (3) and (4) are satisfied by construction. This way, in $\widetilde{O}(\Delta)$ time we reduce the instance (B, Φ) by $\Delta = |S| - |S_0| = p - g$ vertices. Recall that $Z = S \setminus \{s_p\}$ implies that $g < p$. Thus, $\Delta \geq 1$ and the sizes of B and T strictly decrease.

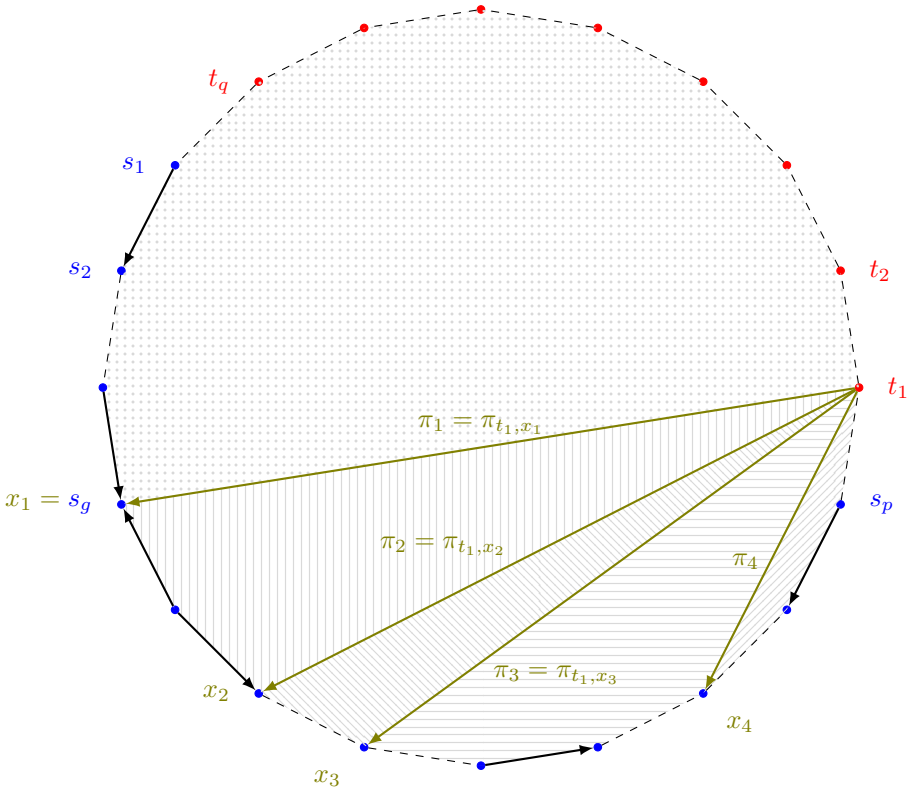


Figure 1 Splitting the instance (B, Φ), where $B = S \cup T$, into 5 smaller instances with paths $\pi_1, \ldots, \pi_l$ (either originating or ending in t_1) for $l = 4$. The vertices $S = \{s_1, \ldots, s_p\}$ are shown in blue, whereas the vertices $T = \{t_1, \ldots, t_q\}$ in red. The black arrows and dashed lines represent the individual parts of the curve Φ: paths of the form $\pi_{u,v}$ or parts of the curve h, respectively. Note that the black arrows appear only on the Φ_{s_1,s_p} part of Φ. The vertices $x_1, \ldots, x_4$, marked green, are precisely all the vertices of $S \setminus \{s_p\}$ that t_1 can reach or can be reached from. The obtained smaller instances are marked with distinct patterns. The instances marked with line patterns (types 1 or 2) are base instances. The instance marked using a dotted pattern (type 3) might constitute the only obtained instance that is not a base instance (for which the algorithm continues).

Let us now discuss how to aggregate/report the vertices of $\Pi_P(B) \setminus \partial P$ that lie on any of the paths $\pi_1, \ldots, \pi_l$ (that are not handled in any of the subproblems), but at the same time lie strictly inside Φ (before altering (B, Φ)). Since Φ is formed of either the edges of the hole h, or from the paths $\pi_{\{u,v\}}$, and each π_i is contained in $P[\Phi]$, equivalently we need to aggregate the vertices of $\Pi_P(B) \setminus \partial P$ on the paths $\pi_1, \ldots, \pi_l$ that do not lie on Φ.

Observe that since $x_1, \ldots, x_l$ lie on Φ in that order, and the paths $\pi_1, \ldots, \pi_l$ all lie in $P[\Phi]$ and are non-crossing by Lemma 14, for any three $i < j < k$, we have $V(\pi_i) \cap V(\pi_k) \subseteq V(\pi_j)$. As a result, any vertex on these paths is included in precisely one of the sets $V(\pi_i) \setminus V(\pi_{i-1})$, for $i = 1, \ldots, l$ and $V(\pi_0) := \emptyset$. Moreover, in Lemma 21 we will show that each π_i can possibly have a non-empty intersection with $O(1)$ parts (between neighboring elements of B) of Φ, that we can also identify in $O(1)$ time. Since, by Lemma 14, for every path $\pi_{\{u,v\}}$, the intersection of $\pi_{\{u,v\}}$ with π_i is either empty or forms a subpath of π_i, aggregating or reporting the required vertices of π_i boils down to aggregating or reporting the vertices of $V(P) \setminus \partial P$ on some $O(1)$ subpaths of π_i that form what remains from π_i after removing $O(1)$ of its intersections with other paths $\pi_{u,v}$.

▶ **Lemma 21.** *Consider the moment when the split into subproblems happens. Suppose $x_i = s_j$. Let $u, v \in B$ be neighboring on Φ, so that u comes before v in the counterclockwise order $\prec_h$ on Φ. Then $(V(\pi_i) \cap V(\Phi_{u,v})) \setminus \partial P \neq \emptyset$ implies that $u = s_{j'}$ for some $j' \in \{1, \ldots, p\}$ such that $|j - j'| \leq 2$.*

Proof. Recall that the curve is π_i is either the path $\pi_{\{x_i,t_1\}}$ or its reverse. Let us only consider the case when $\pi_{\{x_i,t_1\}} = \pi_{x_i,t_1}$; the case when $\pi_{\{x_i,t_1\}} = \pi_{t_1,x_i}$ is analogous.

By invariant (3), we have that $u, v \in S$ since otherwise $\Phi_{u,v}$ is a part of the curve h and therefore does not contain any vertices from outside $V(h)$. So $\{u, v\} = \{s_{j'}, s_{j'+1}\}$ for some $j' \in \{1, \ldots, p-1\}$. Let $z \in (V(\pi_i) \cap V(\Phi_{u,v})) \setminus \partial P$. If $\Phi_{u,v} = \pi_{s_{j'},s_{j'+1}}$, then the $z \to v$ subpath of $\Phi_{u,v}$ and the $x_i \to z$ subpath of π_i together form an $s_j \to s_{j'+1}$ path in P, which by invariant (2) implies $j' + 1 \in \{j-1, j, j+1\}$, and thus $j' \in [j-2, j]$. If $\Phi_{u,v} = \pi_{s_{j'+1},s_{j'}}$, then, analogously, there exists an $s_j \to s_{j'}$ path in P, so $j' \in [j-1, j+1]$. ◀

By Lemma 21, for each π_i, we need to report all vertices of π_i from outside ∂P, except those on the union of at most 6 subpaths of π_i. Since the subpaths are always intersections of some two paths $\pi_{u,v}$, we can identify these subpaths using precomputed data in $O(1)$ time. Aggregating vertex weights not on at most 6 subpaths of π_i is the same as aggregating weights on at most 7 disjoint subpaths of π_i. Recall that we have precomputed the aggregate weights for all the subpaths of all the possible $\pi_{u,v}$. As a result, aggregating or reporting vertices $\Pi_P(B) \setminus \partial P$ that lie on any of the paths $\pi_1, \ldots, \pi_l$ takes $O(l) = O(p - g)$ time.

▶ **Lemma 22.** *The general case can be handled in $\widetilde{O}(|B|)$ time.*

Proof. Recall that when $T = \emptyset$, we have a base instance that can be solved in $O(|B|)$ time.

Every iteration of the main loop that does not involve changing B takes $O(\text{polylog}\, n)$ time and reduces the size of T by one. But the set T can shrink at most $|B|$ times, so such iterations cost $\widetilde{O}(|B|)$ time in total. Every other iteration of the main loop involves reducing the size of B by some $\Delta > 0$ in $\widetilde{O}(\Delta)$ time. Such iterations clearly cost $\widetilde{O}(|B|)$ time in total as well. ◀

References

1. Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014*, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.
2. Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, and Robert E. Tarjan. A new approach to incremental cycle detection and related problems. *ACM Trans. Algorithms*, 12(2):14:1–14:22, 2016. doi:10.1145/2756553.
3. Aaron Bernstein, Aditi Dudeja, and Seth Pettie. Incremental SCC maintenance in sparse graphs. In *29th Annual European Symposium on Algorithms, ESA 2021*, volume 204 of *LIPIcs*, pages 14:1–14:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.14.
4. Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 1123–1134. IEEE, 2020. doi:10.1109/FOCS46700.2020.00108.
5. Aaron Bernstein, Maximilian Probst, and Christian Wulff-Nilsen. Decremental strongly-connected components and single-source reachability in near-linear time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 365–376. ACM, 2019. doi:10.1145/3313276.3316335.

6 Panagiotis Charalampopoulos and Adam Karczmarz. Single-source shortest paths and strong connectivity in dynamic planar graphs. *J. Comput. Syst. Sci.*, 124:97–111, 2022. doi:10.1016/j.jcss.2021.09.008.

7 Shiri Chechik, Thomas Dueholm Hansen, Giuseppe F. Italiano, Jakub Łącki, and Nikos Parotsidis. Decremental single-source reachability and strongly connected components in $\tilde{o}(m\sqrt{n})$ total update time. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016*, pages 315–324. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.42.

8 Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, and Maximilian Probst Gutenberg. Almost-linear time algorithms for incremental graphs: Cycle detection, sccs, *s-t* shortest path, and minimum-cost flow, 2023. arXiv:2311.18295.

9 Krzysztof Diks and Piotr Sankowski. Dynamic plane transitive closure. In *Algorithms - ESA 2007, 15th Annual European Symposium, Proceedings*, pages 594–604, 2007. doi:10.1007/978-3-540-75520-3_53.

10 Jeff Erickson, Kyle Fox, and Luvsandondov Lkhamsuren. Holiest minimum-cost paths and flows in surface graphs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 1319–1332, 2018. doi:10.1145/3188745.3188904.

11 Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *J. Comput. Syst. Sci.*, 72(5):868–889, 2006. doi:10.1016/j.jcss.2005.05.007.

12 Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.*, 14(4):781–798, 1985. doi:10.1137/0214055.

13 Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987. doi:10.1137/0216064.

14 Harold N. Gabow. Path-based depth-first search for strong and biconnected components. *Inf. Process. Lett.*, 74(3-4):107–114, 2000. doi:10.1016/S0020-0190(00)00051-X.

15 Pawel Gawrychowski, Haim Kaplan, Shay Mozes, Micha Sharir, and Oren Weimann. Voronoi diagrams on planar graphs, and computing the diameter in deterministic $\tilde{o}(n^{5/3})$ time. *SIAM J. Comput.*, 50(2):509–554, 2021. doi:10.1137/18M1193402.

16 Pawel Gawrychowski, Shay Mozes, Oren Weimann, and Christian Wulff-Nilsen. Better tradeoffs for exact distance oracles in planar graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 515–529. SIAM, 2018. doi:10.1137/1.9781611975031.34.

17 Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2212–2228. SIAM, 2021. doi:10.1137/1.9781611976465.132.

18 Bernhard Haeupler, Telikepalli Kavitha, Rogers Mathew, Siddhartha Sen, and Robert Endre Tarjan. Incremental cycle detection, topological ordering, and strong component maintenance. *ACM Trans. Algorithms*, 8(1):3:1–3:33, 2012. doi:10.1145/2071379.2071382.

19 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.

20 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001. doi:10.1145/502090.502095.

21 Shang-En Huang, Dawei Huang, Tsvi Kopelowitz, Seth Pettie, and Mikkel Thorup. Fully dynamic connectivity in $o(\log n(\log\log n)^2)$ amortized expected time. *TheoretiCS*, 2, 2023. doi:10.46298/THEORETICS.23.6.

22 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

23 Giuseppe F. Italiano, Adam Karczmarz, Jakub Łącki, and Piotr Sankowski. Decremental single-source reachability in planar digraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 1108–1121. ACM, 2017. doi:10.1145/3055399.3055480.

24 Haim Kaplan, Shay Mozes, Yahav Nussbaum, and Micha Sharir. Submatrix maximum queries in monge matrices and partial monge matrices, and their applications. *ACM Trans. Algorithms*, 13(2):26:1–26:42, 2017. doi:10.1145/3039873.

25 Adam Karczmarz and Marcin Smulewicz. On fully dynamic strongly connected components. In *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 68:1–68:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.68.

26 Philip N. Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005*, pages 146–155, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070454.

27 Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 505–514, 2013. doi:10.1145/2488608.2488672.

28 Philip N. Klein and Sairam Subramanian. A fully dynamic approximation scheme for shortest paths in planar graphs. *Algorithmica*, 22(3):235–249, 1998. doi:10.1007/PL00009223.

29 Yaowei Long and Seth Pettie. Planar distance oracles with better time-space tradeoffs. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2517–2537. SIAM, 2021. doi:10.1137/1.9781611976465.149.

30 Jakub Łącki. Improved deterministic algorithms for decremental reachability and strongly connected components. *ACM Trans. Algorithms*, 9(3):27:1–27:15, 2013. doi:10.1145/2483699.2483707.

31 Liam Roditty and Uri Zwick. Improved dynamic reachability algorithms for directed graphs. *SIAM J. Comput.*, 37(5):1455–1471, 2008. doi:10.1137/060650271.

32 Piotr Sankowski. Subquadratic algorithm for dynamic shortest distances. In *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings*, volume 3595 of *Lecture Notes in Computer Science*, pages 461–470. Springer, 2005. doi:10.1007/11533719_47.

33 M. Sharir. A strong-connectivity algorithm and its applications in data flow analysis. *Computers & Mathematics with Applications*, 7(1):67–72, 1981. doi:10.1016/0898-1221(81)90008-0.

34 Sairam Subramanian. A fully dynamic data structure for reachability in planar digraphs. In *Algorithms - ESA '93, First Annual European Symposium, Bad Honnef, Germany, September 30 - October 2, 1993, Proceedings*, pages 372–383, 1993. doi:10.1007/3-540-57273-2_72.

35 Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. doi:10.1137/0201010.

36 Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 456–480. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00036.

37 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

38 Christian Wulff-Nilsen. Faster deterministic fully-dynamic graph connectivity. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1757–1769. SIAM, 2013. doi:10.1137/1.9781611973105.126.

Minimizing Symmetric Convex Functions over Hybrid of Continuous and Discrete Convex Sets

Yasushi Kawase
University of Tokyo, Japan

Koichi Nishimura
CRESCO LTD., Japan

Hanna Sumita
Tokyo Institute of Technology, Japan

Abstract

We study the problem of minimizing a given symmetric strictly convex function over the Minkowski sum of an integral base-polyhedron and an M-convex set. This problem has a hybrid of continuous and discrete structures. This emerges from the problem of allocating mixed goods, consisting of both divisible and indivisible goods, to agents with binary valuations so that the fairness measure, such as the Nash welfare, is maximized. It is known that both an integral base-polyhedron and an M-convex set have similar and nice properties, and the non-hybrid case can be solved in polynomial time. While the hybrid case lacks some of these properties, we show the structure of an optimal solution. Moreover, we exploit a proximity inherent in the problem. Through our findings, we demonstrate that our problem is NP-hard even in the fair allocation setting where all indivisible goods are identical. Moreover, we provide a polynomial-time algorithm for the fair allocation problem when all divisible goods are identical.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Theory of computation → Algorithmic game theory

Keywords and phrases Integral base-polyhedron, Fair allocation, Matroid

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.96

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2306.05986 [29]

Funding This work was partially supported by JSPS KAKENHI Grant Numbers JP20K19739, JP21K17708, and JP21H03397, Japan, by JST PRESTO Grant Number JPMJPR2122, Japan, by JST ERATO Grant Number JPMJER2301, Japan, and by Value Exchange Engineering, a joint research project between R4D, Mercari, Inc. and the RIISE.

Acknowledgements We are grateful to Kazuo Murota, Warut Suksompong, and the anonymous reviewers for their helpful comments.

1 Introduction

In this paper, we study the hybrid problem of discrete and continuous structures. We are given a finite set N, a symmetric strictly convex function $\Phi\colon \mathbb{R}^N \to \mathbb{R}$ and two supermodular functions $f_C, f_M\colon 2^N \to \mathbb{Z}_+$. We assume that function values can be accessed by an oracle. For a supermodular function f, we call

$$\overline{\mathbf{B}} = \left\{x \in \mathbb{R}^N : x(N) = f(N) \quad \text{and} \quad x(X) \geq f(X) \ \ (\forall X \subseteq N)\right\}$$

the *integral base-polyhedron* of f, and $\ddot{\mathbf{B}} = \overline{\mathbf{B}} \cap \mathbb{Z}^N$ the *M-convex* set. Let $\overline{\mathbf{B}}_C$, $\overline{\mathbf{B}}_M$ be the integral base-polyhedra of f_C, f_M, respectively, and let $\ddot{\mathbf{B}}_M = \overline{\mathbf{B}}_M \cap \mathbb{Z}^N$. In addition, let

$\mathbf{B}_E$ be the Minkowski sum of $\ddot{\mathbf{B}}_M$ and $\overline{\mathbf{B}}_C$, i.e., $\mathbf{B}_E = \{x + y : x \in \ddot{\mathbf{B}}_M,\ y \in \overline{\mathbf{B}}_C\}$. The goal of the problem is to find a vector z that attains

$$\min_{z \in \mathbf{B}_E} \Phi(z)\ (= \min_{x \in \ddot{\mathbf{B}}_M} \min_{y \in \overline{\mathbf{B}}_C} \Phi(x + y)). \tag{1}$$

An optimal solution of this problem is called Φ-*minimizer* (on $\mathbf{B}_E$).

When $f_M = 0$ (i.e., $\mathbf{B}_E = \overline{\mathbf{B}}_C$), it is known that an integral base-polyhedron has a common unique minimizer independent of Φ, and the minimizer can be characterized by a structure called the *principal partition* [18, 34] (see Section 2.2 for the definition and details). By this structure, the problem (1) can be solved in polynomial time [38]. When $f_C = 0$, it is known that a Φ-minimizer on an M-convex set can be characterized by the *canonical partition* [15], which is an aggregation of the principal partition. Additionally, the set of Φ-minimizers does not depend on Φ [15] and a Φ-minimizer can be found in polynomial time [16]. Furthermore, a *proximity theorem* has been established [17]. This theorem states that a Φ-minimizer in an M-convex set lies within a unit hypercube that contains the Φ-minimizer in the corresponding integral base-polyhedron.

The hybrid problem (1) appears in the fair allocation of a mix of divisible and indivisible goods, which has recently been attracting attention [5,7,10,31,32,33,40]. Let $N = \{1, 2, \ldots, n\}$ be the set of agents. Let C and M be the sets of divisible and indivisible goods, respectively, and let also $E = C \cup M$. Each agent i has a binary valuation $v_{ie} \in \{0, 1\}$ for each good e. An allocation is a matrix $\pi \in [0, 1]^{N \times E}$ such that $\pi_{ie} \in \{0, 1\}$ for all $i \in N$ and $e \in M$. The entry π_{ie} means the allocated amount of good e to agent i. Throughout this paper, we only consider utilitarian optimum allocations, that is, $\pi_{ie} > 0$ only if $v_{ie} = 1$. Agents have additive utility, and the utility of agent i in allocation π is $\pi_i(E) = \sum_{e \in E} v_{ie}\pi_{ie}$. For an allocation π, a vector $z = (\pi_1(E), \ldots, \pi_n(E))$ is called a utility vector of π.

The problem of finding a utility vector with the maximum *Nash welfare* (MNW), which is a prominent fairness measure, can be reduced to our problem (1). Roughly speaking, the set of possible utility vectors by divisible goods C forms an integral base-polyhedron $\overline{\mathbf{B}}_C$, and the set by indivisible goods M forms an *M-convex set* $\ddot{\mathbf{B}}_M$, which is the set of integral vectors in an integral base-polyhedron. Maximizing the Nash welfare corresponds to minimizing a symmetric strictly convex function $\Phi(z) = -\sum_{i \in N} \log(z_i + \varepsilon)$ for sufficiently small $\varepsilon > 0$ (depending on the instance). Another standard fairness measure called *egalitarian social welfare* (max-min fairness) also can be represented by a symmetric strictly convex function. For a given symmetric strictly convex function Φ, we call an allocation π Φ-*fair* if its utility vector $z = (\pi_1(E), \ldots, \pi_n(E))$ is a Φ-minimizer. We will detail these in Section 2.

Unfortunately, the hybrid case does not inherit nice properties of continuous or discrete cases even in the fair allocation case. The set $\mathbf{B}_E$ is not necessarily an integral base-polyhedron or an M-convex set. It also does not work to find allocations of divisible and indivisible goods separately and combine them. We can observe these from the following example.

▶ **Example 1.** Suppose that there are one indivisible good g, one divisible good c, and three agents who desire both goods. Let $\Phi(z) = -z_1 \cdot z_2 \cdot z_3$. In this case, allocating c equally to the three agents minimizes Φ when considering only c. However, allocating g to agent 1 and c to agents 2 and 3 equally minimizes Φ for mixed goods. In addition, the set of possible utility vectors is not an M-convex set since it contains fractional utility vectors and not an integral base-polyhedron since it is not convex (see Figure 1).

In addition, we will see that the uniqueness of a Φ-minimizer set no longer holds (Example 7). Therefore, existing results are not applicable to our problem.

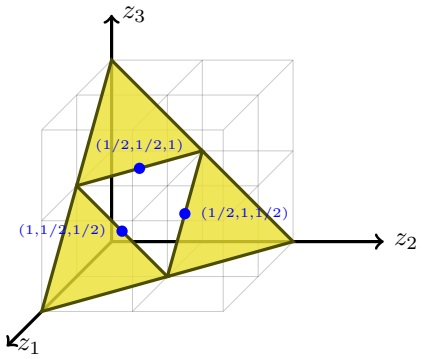


Figure 1 The set of possible utility vectors in Example 1. The blue points are minimizers of Φ.

1.1 Our contribution

First, we investigate the structure of the problem (1). Fortunately, we show that the hybrid problem still retains a structure of the canonical partition (Lemma 28), which is originally defined for the discrete case [15]. Namely, there exists integers $\beta_1 > \cdots > \beta_q$ and a partition $N_1, \ldots, N_q$ of N such that $\beta_j - 1 \leq z_i^* \leq \beta_j$ for any Φ-minimizer z^*, each $j = 1, \ldots, q$ and $i \in N_j$. The proof is based on exchanging element values of a solution. Since existing exchange properties in the discrete case are insufficient to deal with the hybrid problem, we need to introduce new exchange properties. Moreover, we discuss an optimality criterion in terms of an exchange graph, and unlike the discrete case, elaborate analysis of the graph is required. By this result, we can see that an optimal integral solution (i.e., $\arg\min_{z \in \ddot{\mathbf{B}}_M + \ddot{\mathbf{B}}_C} \Phi(z)$) is a good approximation solution for (1) in the sense that the ℓ_∞ distance from the optimal solution is at most 1. We remark that the canonical partition together can be found in polynomial time, and an optimal integral solution can be easily obtained from it.

In addition, by using the canonical partition, we can demonstrate a proximity theorem (Theorem 2). Namely, Φ-minimizer on $\mathbf{B}_E$ lies within a unit box containing the Φ-minimizer on an integral base-polyhedron $\overline{\mathbf{B}}_E$ of $f_E = f_M + f_C$. This generalizes a proximity theorem for the discrete case [17].

▶ **Theorem 2.** *Let Φ be a symmetric strictly convex function. For any $z^* \in \arg\min_{z \in \mathbf{B}_E} \Phi(z)$ and $\bar{z} \in \arg\min_{z \in \overline{\mathbf{B}}_E} \Phi(z)$, we have $\lfloor \bar{z}_i \rfloor \leq z_i^* \leq \lceil \bar{z}_i \rceil$ for all $i \in N$.*

Second, by applying the above results, we analyze the computational complexity of the problem (1) where $\overline{\mathbf{B}}_C$ and $\ddot{\mathbf{B}}_M$ arise from fair allocation. As a negative result, we show that the problem is NP-hard even when indivisible goods are *identical*, i.e., for each agent i, either $v_{ie} = 1$ $(\forall e \in M)$ or $v_{ie} = 0$ $(\forall e \in M)$.

▶ **Theorem 3.** *For any fixed symmetric strictly convex function Φ, finding a Φ-fair allocation is NP-hard even when indivisible goods are identical.*

We also prove that computing an MNW allocation and an optimal egalitarian allocation are both NP-hard. These results highlight the difficulty of the mixed goods case because the problems can be solved in polynomial time when there are only divisible goods or only indivisible goods.

As a positive result, we show the following tractability when divisible goods are identical. This class includes the case when the divisible goods are money.

▶ **Theorem 4.** *Let Φ be a symmetric strictly convex function. There exists a polynomial-time algorithm that finds a Φ-fair allocation if all the divisible goods are identical.*

This result may be interesting, because in fact, finding an allocation that attains a given utility vector is NP-hard (see Appendix B in the full version [29]). Nevertheless, Theorem 4 says that we can obtain not only a Φ-minimizer (utility vector) but also an allocation that attains the utility vector.

A key tool to construct our algorithm is the canonical partition for the mixed goods. By applying it, we can partition goods as $E_1, \dots, E_q$ and agents as $N_1, \dots, N_q$ so that goods in E_j are allocated to agents in N_j in a Φ-fair allocation (Theorem 29). Thanks to this structure, a Φ-minimizer can be found by independently solving the subproblems of assigning E_j to N_j for $j = 1, 2, \dots, q$. In each subproblem, the utility of every agent is almost the same. However, unlike the continuous or discrete case, an optimal allocation depends on Φ (see Examples 6 and 7). Thus, it is not easy to obtain a full characterization of minimizers.

Due to the space limitation, some proofs are omitted. They can be found in the full version [29].

1.2 Related work

The (integral) base-polyhedron has been studied in the theory of matroids and submodular functions [19]. The concept of M-convex sets [36] is defined as a set of integral vectors satisfying certain exchange axioms. *Discrete convex analysis* [37] is a framework of convex analysis in discrete settings, including M-convexity.

The concepts of continuous/discrete hybrid convexity have been proposed previously [35, 44]. In particular, an optimality criterion for an integral polyhedral hybrid M-convex function minimization is known [35]. However, the functions treated in the present paper are hybrid M-convex functions that are not necessarily integral polyhedral.

For the fair allocation of divisible homogeneous goods with additive valuations (not restricted to binary), an MNW allocation corresponds to a market equilibrium of a special case of the Fisher's market model (see, e.g., [39]). Moreover, an MNW allocation is *envy-free* (EF) [43, 45], that is, no agent envies any other agent. It is known that this problem can be solved in strongly polynomial time [41, 46].

For the fair allocation of indivisible goods with additive valuations, Caragiannis et al. [10] proved that an MNW allocation is *envy-free up to one good* (EF1), that is, each agent i does not envy another agent j if some indivisible good is removed from the bundle of agent j. Since computing an MNW allocation is hard in general [30], there is a series of research to design an approximation algorithm [1, 11, 12, 13, 22]. Benabbou et al. [6] proved that the set of MNW allocations coincides with that of minimizers of any symmetric strictly convex function, even when the utility of each agent is represented by a matroid rank function.[1] Harvey et al. [26] proposed efficient algorithms for computing an allocation that minimizes a certain symmetric strictly convex function. When agents have binary additive valuations, an MNW allocation can be computed in polynomial time [4, 14]. Truthful mechanisms to find an MNW allocation are also proposed [2, 24].

Fair allocation with a mixture of divisible and indivisible goods has recently gained attention. Bei et al. [5] introduced a fairness notion called *envy-freeness for mixed goods* (EFM) as a generalization of EF and EF1 notions. Very recently, Li et al. [31] proposed a truthful mechanism that outputs an EFM allocation for the case where agents have binary

[1] Note that Theorem 2 is an extension of this result. Specifically, we construct an "augmenting" path of [6, Section 3.2] for a hybrid situation.

additive valuations on indivisible goods and a common valuation on a single divisible good (e.g., money). They also showed that their mechanism runs in polynomial time, and its output achieves MNW. We remark that their algorithm does not work in our problem even when divisible goods are identical because we allow some agents to have value 0 on them. In addition, fair allocation of indivisible goods with subsidy [3, 8, 9, 23, 25, 28] is related to the problem since subsidy could be viewed as a divisible good. For more details, see a survey paper by Liu et al. [32].

2 Preliminaries

In this section, we explain the relationship between fair allocation of mixed goods and the hybrid problem (1). Then we introduce the canonical partition for the discrete case.

For $k \in \mathbb{N}$, we denote $[k] = \{1, 2, \ldots, k\}$. Let $N = [n]$ be a finite set. A set function f over N is called *supermodular* if

$$f(X) + f(Y) \leq f(X \cup Y) + f(X \cap Y) \quad (\forall X, Y \subseteq N)$$

and *submodular* if $-f$ is supermodular. For a subset $X \subseteq N$ and a vector $x \in \mathbb{R}^N$, we denote $x(X) = \sum_{i \in X} x_i$. For an integer-valued supermodular set function f on N for which $f(\emptyset) = 0$ (normalized), the *integral base-polyhedron* $\overline{\mathbf{B}}$ of f is defined as

$$\overline{\mathbf{B}} = \left\{ x \in \mathbb{R}^N : x(N) = f(N) \text{ and } x(X) \geq f(X) \ (\forall X \subseteq N) \right\}.$$

In addition, we call the set $\ddot{\mathbf{B}}$ of the integer vectors in an integral base-polyhedron $\overline{\mathbf{B}}$ an *M-convex set*. Note that an M-convex set $\ddot{\mathbf{B}}$ induces an integral base-polyhedron $\overline{\mathbf{B}}$ as its convex hull.

We say that a function $\Phi \colon \mathbb{R}^N \to \mathbb{R}$ is *symmetric* if

$$\Phi(z_1, z_2, \ldots, z_n) = \Phi(z_{\sigma(1)}, z_{\sigma(2)}, \ldots, z_{\sigma(n)})$$

for all permutations σ over $[n]$. We say that a function $\Phi \colon \mathbb{R}^N \to \mathbb{R}$ is strictly *convex* if

$$\lambda \Phi(z) + (1 - \lambda)\Phi(z') > \Phi(\lambda z + (1 - \lambda)z')$$

for all $z, z' \in \mathbb{R}^N$ and $\lambda \in (0, 1)$. A typical example of symmetric strictly convex functions is the square-sum $\Phi(z) = \sum_{i \in N} z_i^2$.

In general, for $z \in \mathbb{R}^N$ and $i, j \in N$ with $z_i > z_j$, we have $\Phi(z - \varepsilon(\chi_i - \chi_j)) < \Phi(z)$ for any $\varepsilon \in (0, z_i - z_j)$ because

$$\begin{aligned} \Phi(z) &= \lambda \Phi(z - (z_i - z_j)(\chi_i - \chi_j)) + (1 - \lambda)\Phi(z) \\ &> \Phi(\lambda(z - (z_i - z_j)(\chi_i - \chi_j)) + (1 - \lambda)z) = \Phi(z - \lambda(z_i - z_j)(\chi_i - \chi_j)) \end{aligned} \tag{2}$$

for any $\lambda \in (0, 1)$. Here, χ_i represents a unit vector where only the ith component is equal to 1, while all other components are equal to 0.

2.1 Relationship to fair allocation

Let $N = [n]$ represent the set of n agents. We have two types of goods: $M = \{g_1, g_2, \ldots, g_m\}$ represents the set of indivisible goods, and $C = \{c_1, c_2, \ldots, c_r\}$ denotes the set of homogeneous divisible goods, that is, the valuation for a piece of a good is proportional to its fraction. The set of all goods is denoted by $E = M \cup C$. Let v_{ie} be the valuation of good $e \in E$ for

agent $i \in N$. We assume that agents have binary valuations, that is, the valuation v_{ie} for the whole of good e is either 0 or 1 for all $i \in N$ and $e \in E$. An instance of the fair allocation we deal with in this paper is described as (N, M, C, v). Without loss of generality, we assume that, for any $e \in E$, there exists $i \in N$ such that $v_{ie} = 1$.

A *relaxed allocation* is defined as a matrix $\pi \in [0,1]^{N \times E}$ that satisfies (i) $\sum_{i \in N} \pi_{ie} = 1$ for all $e \in E$ and (ii) $\pi_{ie} = 0$ for any $i \in N$ and $e \in E$ with $v_{ie} = 0$. In a relaxed allocation π, each agent i receives each good e in the proportion of π_{ie}. Relaxed allocations treat indivisible goods as divisible. A relaxed allocation π is an *allocation* if it additionally satisfies $\pi_{ie} \in \{0,1\}$ for all $i \in N$ and $e \in M$. A relaxed allocation π is an *integral allocation* if it additionally satisfies $\pi_{ie} \in \{0,1\}$ for all $i \in N$ and $e \in E$. For an allocation π, an agent $i \in N$, and a subset of goods $E' \subseteq E$, let $\pi_i(E') = \sum_{e \in E'} \pi_{ie}$, which is the valuation of agent i's bundle from E'. For an allocation π, the utility of agent $i \in N$ is defined as $\pi_i(E)$. For an allocation π, let $\pi(E)$ be the utility vector $(\pi_1(E), \ldots, \pi_n(E))$.

Here we rewrite the set of possible utility vectors in terms of an integral base-polyhedron. We define $f_M, f_C, f_E \colon 2^N \to \mathbb{Z}_+$ as follows: for a subset $X \subseteq N$ of agents,

- $f_M(X) = |\{g \in M : v_{ig} = 0 \ (\forall i \notin X)\}|$ is the number of indivisible goods that must be allocated to agents in X,
- $f_C(X) = |\{c \in C : v_{ic} = 0 \ (\forall i \notin X)\}|$ is the number of divisible goods that must be allocated to agents in X, and
- $f_E(X) = f_M(X) + f_C(X)$ is the number of goods that must be allocated to agents in X.

It is not difficult to see that the functions f_M, f_C, f_E are normalized integer-valued supermodular.[2] Let $\ddot{\mathbf{B}}_M$ and $\overline{\mathbf{B}}_C$ be the M-convex set of f_M and the integral base-polyhedron of f_C, respectively. In addition, let $\mathbf{B}_E$ be the Minkowski sum of $\ddot{\mathbf{B}}_M$ and $\overline{\mathbf{B}}_C$, i.e., $\mathbf{B}_E = \{x + y : x \in \ddot{\mathbf{B}}_M, \ y \in \overline{B}_C\}$. Then, $\mathbf{B}_E$ is the set of possible utility vectors.

Recall that $\overline{\mathbf{B}}_M = \mathrm{conv}(\ddot{\mathbf{B}}_M)$ and $\ddot{\mathbf{B}}_C = \overline{B}_C \cap \mathbb{Z}^N$. We denote $\overline{\mathbf{B}}_E = \mathrm{conv}(\mathbf{B}_E)$, and $\ddot{\mathbf{B}}_E = \mathbf{B}_E \cap \mathbb{Z}^N$. Note that $\mathbf{B}_E$ is not necessarily an M-convex set or an integral base-polyhedron as we have seen in Example 1.

For a symmetric strictly convex function Φ, an allocation π is called Φ-*fair* if the utility vector $(\pi_1(E), \ldots, \pi_n(E))$ minimizes Φ among allocations.

Some prominent fairness notions are naturally represented as Φ-fairness for some Φ. An allocation π is said to achieve the *maximum Nash welfare* (MNW) if the number of agents with positive utilities is maximized, and subject to that, the Nash welfare $(\prod_{i \in N : \pi_i(E) > 0} \pi_i(E))^{1/n}$ is maximized. Finding a utility vector of an MNW allocation is equivalent to minimizing $-\prod_{i \in N : z_i > 0}(z_i + \varepsilon)$ for some sufficiently small $\varepsilon > 0$ (see Appendix A in the full version [29]). The egalitarian social welfare is defined by the smallest utility among agents. Maximizing the egalitarian social welfare is a weaker notion of *increasingly maximal* (*inc-max*, for short) allocations; an allocation is inc-max if its smallest utility is as large as possible, within this, its second smallest utility is as large as possible, and so on. Similarly, we say that an allocation is *decreasingly minimal* (*dec-min*, for short) if its largest utility is as small as possible, within this, its second largest utility is as small as possible, and so on. We show that a certain symmetric strictly convex function Φ induces the dec-min and inc-max solution as a Φ-fair allocation.

[2] Most of our results can be extended to the case when each agent evaluates indivisible goods with a matroid rank function and divisible goods with the concave closure of a matroid rank function because the functions f_M, f_C, f_E continue to be normalized integer-valued supermodular in this scenario.

▶ **Proposition 5.** *Let (N, M, C, v) be a fair allocation instance. There exists a symmetric strictly convex function Φ such that a dec-min allocation is Φ-fair. In addition, there exists a symmetric strictly convex function Φ' such that an inc-max allocation is Φ'-fair.*

Then for a symmetric strictly convex function $\Phi\colon \mathbb{R}^N \to \mathbb{R}$, we can rewrite finding a utility vector z^* of a Φ-fair allocation as the hybrid problem (1). If $M = \emptyset$ or $C = \emptyset$, the problem (1) can be solved in polynomial time [15, 16, 17, 18, 34].

When there are only divisible goods or only indivisible goods, once we obtain a Φ-mimizer z^* of (1), a Φ-fair allocation π is obtained by using the maximum flow problem. Since we can find an integral maximum flow in the indivisible goods case, π can be an integral allocation. However, when both types of goods exist, it is not straightforward to construct an allocation from a given utility vector. Indeed, given a vector u (not necessarily in $\mathbf{B}_E$), checking the existence of an allocation whose utility vector is u is NP-hard (see Appendix B in the full version [29]).

As mentioned in Introduction, when $\mathbf{B}_E = \overline{\mathbf{B}}_C$ (continuous case) or $\mathbf{B}_E = \ddot{\mathbf{B}}_M$ (discrete case), the set of Φ-minimizers is independent of Φ. However, this is not the case in general even in fair allocation with both types of goods (i.e., $M \neq \emptyset$ and $C \neq \emptyset$). Thus, our problem is challenging.

▶ **Example 6.** Consider an instance with five agents $N = \{1, 2, 3, 4, 5\}$, five indivisible goods $M = \{g_1, g_2, g_3, g_4, g_5\}$, and three divisible goods $C = \{c_1, c_2, c_3\}$. Suppose that agents 1, 2, 3, and 4 desire all the goods, but agent 5 desires only the indivisible goods (see Table 1). Then, an allocation π with $\pi(E) = (7/4, 7/4, 7/4, 7/4, 1)$ is dec-min. However, an allocation ρ with $\rho(E) = (6/4, 6/4, 6/4, 6/4, 2)$ is inc-max and square-sum minimizer. Indeed, $\sum_{i\in N} \pi_i(E)^2 = 13.25$ and $\sum_{i\in N} \rho_i(E)^2 = 13$.

▶ **Example 7.** Consider an instance with five agents $N = \{1, 2, 3, 4, 5\}$, five indivisible goods $M = \{g_1, g_2, g_3, g_4, g_5\}$, and *two* divisible goods $C = \{c_1, c_2\}$. Suppose that agents 1, 2, 3, and 4 desire all the goods, but agent 5 desires only the indivisible goods. Then, an allocation π with $\pi(E) = (\frac{6}{4}, \frac{6}{4}, \frac{6}{4}, \frac{6}{4}, 1)$ is dec-min and square-sum minimizer. However, an allocation ρ with $\rho(E) = (\frac{5}{4}, \frac{5}{4}, \frac{5}{4}, \frac{5}{4}, 2)$ is inc-max. Indeed, $\sum_{i\in N} \pi_i(E)^2 = 10$ and $\sum_{i\in N} \rho_i(E)^2 = 10.25$.

Table 1 Valuations in Example 6.

agents	g_1	g_2	g_3	g_4	g_5	c_1	c_2	c_3
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1
5	1	1	1	1	1	0	0	0

Table 2 Valuations in Example 7.

agents	g_1	g_2	g_3	g_4	g_5	c_1	c_2
1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1
5	1	1	1	1	1	0	0

2.2 Principal Partition and Canonical Partition

Consider the integral base-polyhedron $\overline{\mathbf{B}}$ and the M-convex set $\ddot{\mathbf{B}}$ of a supermodular function $f\colon 2^N \to \mathbb{Z}_+$. For any real number λ, let $\mathcal{L}(\lambda)$ be the set of all maximizers of $f(X) - \lambda|X|$, i.e., $\mathcal{L}(\lambda) = \arg\max_{X\subseteq N}(f(X) - \lambda|X|)$. Note that $\mathcal{L}$ has a lattice structure, i.e., $\mathcal{L}$ is closed under union and intersection. Let $\mathrm{L}(\lambda)$ be the smallest member in $\mathcal{L}(\lambda)$. It is known that $\mathrm{L}(\lambda) \subseteq \mathrm{L}(\lambda')$ for any $\lambda > \lambda'$ (see, e.g., [17, Proposition 3.1]).

Fujishige [18] characterized the optimal utility vectors by the *principal partition* of N. There are at most $|N|$ number of λ for which $|\mathcal{L}(\lambda)| \geq 2$. Let us denote such numbers as $\lambda_1 > \lambda_2 > \cdots > \lambda_r$, which are called the *critical values*. The principal partition $\hat{N}_1, \hat{N}_2, \ldots, \hat{N}_r$ is a partition of N defined by

$$\hat{N}_j = \mathrm{L}(\lambda'_j) - \mathrm{L}(\lambda_j) \quad (j = 1, 2, \ldots, r),$$

where λ'_j is an arbitrary real satisfying $\lambda_j > \lambda'_j > \lambda_{j+1}$ (assuming that $\lambda_{r+1} = -\infty$).

▶ **Theorem 8** (Fujishige [18] and Maruyama [34]). *The unique minimizer x^* of $\min_{x \in \overline{\mathbf{B}}} \Phi(x)$ satisfies $x^*_i = \lambda_j$ for each $i \in \hat{N}_j$ and $j \in [r]$.*

The principal partition and critical values can be found in strongly polynomial time by using the submodular function minimization [27, 42]; see also [38]. For more details of the principal partition, see a book and a survey of Fujishige [19, 20].

Frank and Murota [15] characterized the optimal utility vectors of $\min_{y \in \ddot{\mathbf{B}}} \Phi(y)$ by the *canonical partition* of N. There are at most $|N|$ number of $\beta \in \mathbb{Z}$ for which $\mathrm{L}(\beta) \neq \mathrm{L}(\beta - 1)$. Let us denote such numbers as $\beta_1 > \beta_2 > \cdots > \beta_q$, which are called the *essential values*. The canonical partition $N_1, N_2, \ldots, N_q$ is a partition of N defined by

$$N_i = \mathrm{L}(\beta_i - 1) - \mathrm{L}(\beta_i) \quad (i = 1, 2, \ldots, q).$$

Alternatively, the canonical partition and the essential values can be obtained by the following procedure [17, Section 3]: for $j = 1, 2, \ldots, q$, define

$$\beta_j = \max_{\emptyset \neq X \subseteq N \setminus \bigcup_{j'=1}^{j-1} N_{j'}} \left\lceil (f(X \cup \textstyle\bigcup_{j'=1}^{j-1} N_{j'}) - f(\bigcup_{j'=1}^{j-1} N_{j'}))/|X| \right\rceil, \tag{3}$$

$$h_j(X) = f(X \cup \textstyle\bigcup_{j'=1}^{j-1} N_{j'}) - (\beta_j - 1)|X| - f(\bigcup_{j'=1}^{j-1} N_{j'}) \quad (\forall X \subseteq N \setminus \textstyle\bigcup_{j'=1}^{j-1} N_{j'}),$$

$$N_j = \text{smallest subset of } N \setminus \textstyle\bigcup_{j'=1}^{j-1} N_{j'} \text{ maximizing } h_j.$$

They provided a strongly polynomial-time algorithm to compute the canonical partition and the essential values by using this structure and a strongly polynomial-time algorithm for the submodular function minimization [27, 42].

▶ **Theorem 9** (Frank and Murota [15, 16, 17]). *The essential values $\beta_1 > \beta_2 > \cdots > \beta_q$ are obtained from the critical values $\lambda_1 > \lambda_2 > \cdots > \lambda_r$ as the distinct members of the rounded-up integers $\lceil \lambda_1 \rceil \geq \lceil \lambda_2 \rceil \geq \cdots \geq \lceil \lambda_r \rceil$. Moreover, the canonical partition is an aggregation of the principal partition as $N_i = \bigcup_{j: \lceil \lambda_j \rceil = \beta_i} \hat{N}_j$ for each $i \in [q]$. Any minimizer y^* of $\min_{y \in \ddot{\mathbf{B}}} \Phi(y)$ satisfies $\beta_j - 1 \leq y^*_i \leq \beta_j$ for each $i \in N_j$ and $j \in [q]$. The minimizer y^*, the canonical partition, and essential values can be found in strongly polynomial time with respect to $|N|$.*

3 Exchange properties of integral base-polyhedra and M-convex sets

In this section, we review exchange properties of an integral base-polyhedron and an M-convex set. We also add new exchange properties for our hybrid case.

Let $f \colon 2^N \to \mathbb{Z}$ be a supermodular function. Let $\overline{\mathbf{B}}$ be an integral base-polyhedra and let $\ddot{\mathbf{B}}$ be an M-convex set defined by f. It is known that the M-convex set $\ddot{\mathbf{B}}$ and the integral base-polyhedron $\overline{\mathbf{B}}$ satisfy the *exchange properties*, respectively. For a vector $z \in \mathbb{R}^N$, define $\mathrm{supp}^+(z) \coloneqq \{i \in N : z_i > 0\}$ and $\mathrm{supp}^-(z) \coloneqq \{i \in N : z_i < 0\}$.

▶ **Proposition 10** ([37]). *For any $x, y \in \ddot{\mathbf{B}}$ and $i \in \mathrm{supp}^+(x - y)$, there exists some $j \in \mathrm{supp}^-(x - y)$ such that $x - \chi_i + \chi_j \in \ddot{\mathbf{B}}$ and $y + \chi_i - \chi_j \in \ddot{\mathbf{B}}$.*

▶ **Proposition 11** ([37]). *For any $x, y \in \overline{\mathbf{B}}$ and $i \in \mathrm{supp}^+(x - y)$, there exists some $j \in \mathrm{supp}^-(x - y)$ and a positive real α_0 such that $x - \alpha(\chi_i - \chi_j) \in \overline{\mathbf{B}}$ and $y + \alpha(\chi_i - \chi_j) \in \overline{\mathbf{B}}$ for all $\alpha \in [0, \alpha_0]$.*

In addition, we show the following variants of exchange properties.

▶ **Proposition 12.** *For any $x, y \in \ddot{\mathbf{B}}$ and $X \subseteq N$ with $x(X) > y(X)$, there exist $i \in X$ and $j \in N \setminus X$ such that $x - \chi_i + \chi_j \in \ddot{\mathbf{B}}$. Moreover, if $x(X) > f(X)$, there exist $i \in X$ and $j \in N \setminus X$ such that $x - \chi_i + \chi_j \in \ddot{\mathbf{B}}$.*

▶ **Proposition 13.** *For any $x, y \in \overline{\mathbf{B}}$ and $X \subseteq N$ with $x(X) > y(X)$, there exist $i \in X$, $j \in N \setminus X$, and $\varepsilon > 0$ such that $x - \varepsilon(\chi_i - \chi_j) \in \overline{\mathbf{B}}$. Moreover, if $x(X) > f(X)$, there exist $i \in X$, $j \in N \setminus X$, and $\varepsilon > 0$ such that $x - \varepsilon(\chi_i - \chi_j) \in \overline{\mathbf{B}}$.*

▶ **Proposition 14.** *For any $x \in \ddot{\mathbf{B}}$ and disjoint subsets $I, J \subseteq N$ such that $x(J) < f(I \cup J) - f(I)$, there exist $i \in I$ and $j \in J$ such that $x - \chi_i + \chi_j \in \ddot{\mathbf{B}}$.*

Then we show an exchange property for the hybrid situation.

▶ **Proposition 15.** *For any $x \in \ddot{\mathbf{B}}$ and $y \in \overline{\mathbf{B}}$, and $i \in \operatorname{supp}^+(x - y)$, there exists $j \in \operatorname{supp}^-(x-y)$ such that $x-\chi_i+\chi_j \in \ddot{\mathbf{B}}$. Also, for any $x \in \ddot{\mathbf{B}}$ and $y \in \overline{\mathbf{B}}$, and $i \in \operatorname{supp}^-(x-y)$, there exists $j \in \operatorname{supp}^+(x - y)$ such that $x + \chi_i - \chi_j \in \ddot{\mathbf{B}}$.*

Proof. We only provide a proof for the former part, as the latter part can be demonstrated in a similar manner. By Proposition 11, there exists some $j \in \operatorname{supp}^-(x - y)$ and a positive real α_0 such that $x' \coloneqq x - \alpha_0(\chi_i - \chi_j) \in \overline{\mathbf{B}}$. For any $X \subseteq N$ with $i \in X$ and $j \notin X$, we have $x'(X) = x(X) - \alpha_0 \geq f(X)$, and hence $x(X) - 1 \geq f(X)$ since $x(X)$ and $f(X)$ are integers. Therefore, $x - \chi_i + \chi_j \in \ddot{B}$ holds. ◀

4 Structure of Φ-minimizers

In this section, we prove a proximity theorem (Theorem 2) by using the structure (Lemma 28) based on the canonical partition. Moreover, in the case of fair allocation, we also show that the problem has a canonical partition of goods (Theorem 29).

Let f_M and f_C be two supermodular functions over M and C, respectively. Let also $f_E = f_M + f_C$. Recall that $\ddot{\mathbf{B}}_M$ and $\overline{\mathbf{B}}_C$ are a corresponding M-convex set and integral base-polyhedron, respectively. In addition, $\mathbf{B}_E$ is the Minkowski sum of $\ddot{\mathbf{B}}_M$ and $\overline{\mathbf{B}}_C$.

It should be noted that, for the integral base-polyhedron $\overline{\mathbf{B}}$ and the M-convex set $\ddot{\mathbf{B}}$ of a common supermodular function, the following proximity theorem has been shown by Frank and Murota [17].

▶ **Theorem 16** ([17, Theorem 4.1])**.** *Let Φ be a symmetric strictly convex function. For any $x^* \in \arg\min_{x \in \ddot{\mathbf{B}}} \Phi(x)$ and $y^* \in \arg\min_{y \in \overline{\mathbf{B}}} \Phi(y)$, we have $\lfloor y_i^* \rfloor \leq x_i^* \leq \lceil y_i^* \rceil$ for all $i \in N$.*

Note that this is a special case of our Theorem 2 when $f_C(X) = 0$ ($\forall X \subseteq N$). We prove Theorem 2 following the same approach as for Theorem 16. However, we need to conduct a more detailed analysis to handle $\overline{\mathbf{B}}_C$.

Throughout this section, we fix a symmetric strictly convex function $\Phi\colon \mathbb{R}^N \to \mathbb{R}$ and its minimizer $z^* \in \arg\min_{z \in \mathbf{B}_E} \Phi(z)$. By definition, z^* can be represented as $x^* + y^*$ by $x^* \in \ddot{\mathbf{B}}_M$ and $y^* \in \overline{\mathbf{B}}_C$. Moreover, let $N_1, \ldots, N_q$ and $\beta_1, \ldots, \beta_q$ be the canonical partition and the essential values of the M-convex set $\ddot{\mathbf{B}}_E$.

In subsequent subsections, we prove Theorem 2 through the following steps. First, in Section 4.1, we demonstrate that z_i^* lies within the interval $[\beta_1 - 1, \beta_1]$ for $i \in N_1$. Then, in Section 4.2, we decompose the problem of finding z^* into two independent problems on N_1 and $N \setminus N_1$. By iteratively applying the same procedure, we obtain the desired structure. In Section 4.3, we show an additional result when $\mathbf{B}_E$ emerges from the fair allocation.

4.1 The peak-set N_1

In this subsection, we prove that $\beta_1 - 1 \leq z^*_i \leq \beta_1$ $(\forall i \in N_1)$ and $z^*(N_1) = f_E(N_1)$. The main idea to prove these is to transfer some amounts from elements with high value to those with low value, which improves the objective value. Here, we need to be careful about the constraint.

We first introduce the basic properties for an M-convex set $\ddot{\mathbf{B}}$ and an integral base-polyhedron $\overline{\mathbf{B}}$.

▶ **Proposition 17.** *For any $x \in \ddot{\mathbf{B}}$, if $x - \chi_i + \chi_j \in \ddot{\mathbf{B}}$ and $x - \chi_j + \chi_k \in \ddot{\mathbf{B}}$, then it holds that $x - \chi_i + \chi_k \in \ddot{\mathbf{B}}$.*

▶ **Proposition 18** ([19, Lemma 4.5])**.** *Let $x \in \ddot{\mathbf{B}}$. Suppose that there exist a sequence $i_1, j_1, \ldots, i_r, j_r$ of $2r$ distinct elements in N such that $x - \chi_{i_h} + \chi_{j_k} \in \ddot{\mathbf{B}}$ if $h = k$ and $x - \chi_{i_h} + \chi_{j_k} \notin \ddot{\mathbf{B}}$ if $h > k$ for $h, k \in [r]$. Then, it holds that $x - \sum_{k\in[r]}(\chi_{i_k} - \chi_{j_k}) \in \ddot{\mathbf{B}}$.*

▶ **Proposition 19** ([37])**.** *For any $x \in \overline{\mathbf{B}}$, we have $x \in \mathrm{conv}(\overline{\mathbf{B}} \cap \{y \in \mathbb{Z}^N : \|x - y\|_\infty < 1\})$.*

▶ **Lemma 20.** *If $y^* + \varepsilon(\chi_i - \chi_j) \in \overline{\mathbf{B}}_C$ for some $i, j \in N$ and $\varepsilon > 0$, then $z^*_i = x^*_i + y^*_i \geq x^*_j + y^*_j = z^*_j$. In addition, for any $\beta \in \mathbb{R}$, it holds that $y^*(N') = f_C(N')$ with $N' = \{i \in N : z^*_i \geq \beta\}$.*

Proof. For the former statement, suppose to the contrary that $y^* + \varepsilon(\chi_i - \chi_j) \in \overline{\mathbf{B}}_C$ for some $i, j \in N$ such that $x^*_i + y^*_i < x^*_j + y^*_j$ and $\varepsilon > 0$. Then, $y = y^* + \min\{\varepsilon, (y^*_j - y^*_i)/2\} \cdot (\chi_i - \chi_j) \in \overline{\mathbf{B}}_C$ and $\Phi(x^* + y) < \Phi(x^* + y^*)$ by (2). This contradicts the assumption that $z^* = x^* + y^*$ is a Φ-minimizer.

For the latter statement, suppose that $y^*(N') > f_C(N')$ for some $N' = \{i \in N : z^*_i \geq \beta\}$ with $\beta \in \mathbb{R}$. Then, by Proposition 13, there exist $i \in N'$, $j \in N \setminus N'$, and $\varepsilon > 0$ such that $y^* + \varepsilon(\chi_j - \chi_i) \in \overline{\mathbf{B}}_C$. By the former statement, this implies $z^*_j \geq z^*_i$, contradicting $j \notin N'$. ◀

We define a graph whose edge represents that transferring a unit amount of elements does not violate the constraints. Then we show that transferring a unit amount along a path will improve the objective value.

▶ **Lemma 21.** *Let β be an integer and $N' \subseteq N$. Define $N^> = \{i \in N : z^*_i > \beta\}$, $N^= = \{i \in N : z^*_i = \beta\}$, and $N^< = \{i \in N : z^*_i < \beta\}$. Construct a graph*

$$G = \big(N', \big\{(i,j) \in N' \times N' : x^* - \chi_i + \chi_j \in \ddot{\mathbf{B}}_M \text{ or } y^* - \chi_i + \chi_j \in \overline{\mathbf{B}}_C\big\}\big).$$

If G has a path from some $i \in N' \cap N^>$ to some $j \in N' \cap N^>$ with $i \neq j$, then there exists a vector z'' such that $\Phi(z'') < \Phi(z^*)$.

Proof. We first observe that Lemma 20 implies $y^*(N^>) = f_C(N^>)$ and $y^*(N^> \cup N^=) = f_C(N^> \cup N^=)$.

Let $P = (i_1, i_2, \ldots, i_k)$ be a shortest path from some $i_1 \in N' \cap N^>$ to some $i_k \in N' \cap N^<$. Then we have $i_1 \in N^>$, $i_2, \ldots, i_{k-1} \in N^=$, and $i_k \in N^<$. Since $z^*_{i_2} < z^*_{i_1}$, Lemma 20 implies that $y^* - \chi_{i_1} + \chi_{i_2} \notin \overline{\mathbf{B}}_C$, and thus $x^* - \chi_{i_1} + \chi_{i_2} \in \ddot{\mathbf{B}}_M$. If $x^* - \chi_{i_2} + \chi_{i_3} \in \ddot{\mathbf{B}}_M$, then $x^* - \chi_{i_1} + \chi_{i_3} \in \ddot{\mathbf{B}}_M$ by Proposition 17, which leads to a shortcut of P. Thus, we have $x^* - \chi_{i_2} + \chi_{i_3} \notin \ddot{\mathbf{B}}_M$ and instead, $y^* - \chi_{i_2} + \chi_{i_3} \in \overline{\mathbf{B}}_C$ holds. Next, let us assume that $y^* - \chi_{i_3} + \chi_{i_4} \in \overline{\mathbf{B}}_C$ and $i_4 \in N^=$, and derive a contradiction. Consider

$$\overline{\mathbf{B}}^= = \left\{ \tilde{y} \in \mathbb{R}^{N^=} : \begin{array}{l} \tilde{y}(N^=) = f_C(N^= \cup N^>) - f_C(N^>), \\ \tilde{y}(X) \geq f_C(X \cup N^>) - f_C(N^>) \ (\forall X \subseteq N^=) \end{array} \right\},$$

which is also an integral base-polyhedron of a supermodular function. Since β is an integer, $y^*_{N^=}$ is also integral. Thus, the vectors $y^*_{N^=} - \chi_{i_2} + \chi_{i_3}$ and $y^*_{N^=} - \chi_{i_3} + \chi_{i_4}$ are contained in $\ddot{\mathbf{B}}^=$, since $y^*(N^>) = f_C(N^>)$ and $i_2, i_3, i_4 \in N^=$. Then $y^*_{N^=} - \chi_{i_2} + \chi_{i_4} \in \ddot{\mathbf{B}}^=$ follows from Proposition 17, which means that $y^* - \chi_{i_2} + \chi_{i_4} \in \overline{\mathbf{B}}_C$[3]. However, this implies a shortcut of P. Therefore, if $i_4 \in N^=$, then $y^* - \chi_{i_3} + \chi_{i_4} \notin \overline{\mathbf{B}}_C$, and hence $x^* - \chi_{i_3} + \chi_{i_4} \in \ddot{\mathbf{B}}_M$. By the same argument, we have $x^* - \chi_{i_\ell} + \chi_{i_{\ell+1}} \in \ddot{\mathbf{B}}_M$ if ℓ is an odd number and $i_{\ell+1} \in N^=$, and $y^* - \chi_{i_\ell} + \chi_{i_{\ell+1}} \in \overline{\mathbf{B}}_C$ if ℓ is an even number. Note that k must be an even number, because $y^* - \chi_{i_{k-1}} + \chi_{i_k} \notin \overline{\mathbf{B}}_C$ follows from Lemma 20 and $z^*_{i_k} < z^*_{i_{k-1}}$. Moreover, for any integers ℓ and h with $\ell \geq h+2$, we have $x^* - \chi_{i_\ell} + \chi_{i_h} \notin \ddot{\mathbf{B}}_M$ and $y^* - \chi_{i_\ell} + \chi_{i_h} \notin \overline{\mathbf{B}}_C$. Let $x' = x^* - \sum_{\ell:\,\mathrm{odd}}(\chi_{i_\ell} - \chi_{i_{\ell+1}})$, $y' = y^* - \sum_{\ell:\,\mathrm{even}}(\chi_{i_\ell} - \chi_{i_{\ell+1}})$, and $z' = x' + y'$. By Proposition 18, we have $x' \in \ddot{\mathbf{B}}_M$, $y' \in \overline{\mathbf{B}}_C$, and $z' \in \mathbf{B}_E$. Note that $z' = z^* - \chi_{i_1} + \chi_{i_k}$, $y'(N^>) = y^*(N^>)$ $(= f_C(N^>))$ and $y'(N^> \cup N^=) = y^*(N^> \cup N^=)$ $(= f_C(N^> \cup N^=))$ by the construction. For notational convenience, we denote $i^* = i_1$ and $j^* = i_k$ in the following.

If $z^*_{i^*} > z^*_{j^*} + 1$, then $\Phi(z') < \Phi(z^*)$. Thus, suppose that $z^*_{i^*} \leq z^*_{j^*} + 1$ $(< \beta + 1)$. In this case, $\beta - 1 < z'_{i^*} < \beta$ and $\beta < z'_{j^*} < \beta + 1$. By Proposition 19, y' can be represented by a convex combination of its integral neighbors in $\overline{\mathbf{B}}_C$. Let $y' = \sum_{t=1}^r \lambda^{(t)} \cdot y^{(t)}$, where $y^{(t)} \in \ddot{\mathbf{B}}_C \cap \{y \in \mathbb{Z}^N : \|y - y'\| < 1\}$ $(\forall t \in [r])$, $\sum_{t=1}^r \lambda^{(t)} = 1$, and $\lambda^{(t)} \geq 0$ $(\forall t \in [r])$. Define $z^{(t)} = x' + y^{(t)}$ for each t. Thus, we also obtain $z' = \sum_{t=1}^r \lambda^{(t)} \cdot z^{(t)}$. Note that $z^{(t)}_{i^*} \in \{\beta - 1, \beta\}$ and $z^{(t)}_{j^*} \in \{\beta, \beta + 1\}$ for each t. In addition, for each t, it holds that $y^{(t)}(N^>) = f_C(N^>)$ because $\sum_{t=1}^r \lambda^{(t)} \cdot y^{(t)}(N^>) = y'(N^>) = y^*(N^>) = f_C(N^>)$ and $y^{(t)}(N^>) \geq f_C(N^>)$. Similarly, we can see that $y^{(t)}(N^> \cup N^=) = f_C(N^> \cup N^=)$ for each t.

Let us choose an arbitrary t with $z^{(t)}_{i^*} = \beta - 1$. Let

$$\overline{\mathbf{B}}^> = \{\tilde{y} \in \mathbb{R}^{N^>} : \tilde{y}(N^>) = f_C(N^>) \text{ and } \tilde{y}(X) \geq f_C(X)\ (\forall X \subseteq N^>)\}$$

(the restriction of $\overline{\mathbf{B}}_C$ to $N^>$), and $\ddot{\mathbf{B}}^>$ be the M-convex set induced from $\overline{\mathbf{B}}^>$. Then it holds that $y^{(t)}_{N^>} \in \ddot{\mathbf{B}}^>$, $y'_{N^>} \in \overline{\mathbf{B}}^>$ and $i^* \in \mathrm{supp}^-(y^{(t)}_{N^>} - y'_{N^>})$. We apply Proposition 15 to them. Then we can choose an index $i^{(t)} \in \mathrm{supp}^+(y^{(t)}_{N^>} - y'_{N^>})$ such that $y^{(t)}_{N^>} + \chi_{i^*} - \chi_{i^{(t)}} \in \overline{\mathbf{B}}^>$. We show that this implies $\hat{y}^{(t)} := y^{(t)} + \chi_{i^*} - \chi_{i^{(t)}} \in \overline{\mathbf{B}}_C$. Indeed, for any X with $i^* \notin X$ and $i^{(t)} \in X$ (the other cases are trivial), since $y^{(t)}(X \cap N^>) - 1 \geq f_C(X \cap N^>)$, we have

$$\begin{aligned} y^{(t)}(X) &= y^{(t)}(X \cap N^>) + y^{(t)}(X \cup N^>) - y^{(t)}(N^>) \\ &> f_C(X \cap N^>) + f_C(X \cup N^>) - f_C(N^>) \geq f_C(X), \end{aligned}$$

which implies that $\hat{y}^{(t)}(X) \geq f_C(X)$. In addition, we observe that $z^{(t)}_{i^{(t)}} = y^{(t)}_{i^{(t)}} + x'_{i^{(t)}} > y'_{i^{(t)}} + x'_{i^{(t)}} = z'_{i^{(t)}} = z_{i^{(t)}} > \beta$. Thus, since $z^{(t)}_{i^{(t)}}$ is an integer,

$$\hat{y}^{(t)}_{i^{(t)}} + x'_{i^{(t)}} = z^{(t)}_{i^{(t)}} - 1 \geq \beta. \tag{4}$$

On the other hand, for each t with $z^{(t)}_{i^*} = \beta$, we denote $\hat{y}^{(t)} = y^{(t)}$. We also remark that $\hat{y}^{(t)}(N^> \cup N^=) = y^{(t)}(N^> \cup N^=)$ and $\hat{y}^{(t)}_i = y^{(t)}_i$ for each t and $i \in N \setminus N^>$.

We will do similar operations for indices in $N^<$. Let us choose an arbitrary t with $z^{(t)}_{j^*} = \beta_1 + 1$. We show that we can choose $j^{(t)} \in N^<$ with $z^{(t)}_{j^{(t)}} \leq \beta_1 - 1$ such that $\hat{y}^{(t)} - \chi_{j^*} + \chi_{j^{(t)}} \in \overline{\mathbf{B}}_C$ by applying Proposition 15. We denote $N^\geq = N^> \cup N^=$. Let also

$$\overline{\mathbf{B}}^< = \left\{ \tilde{y} \in \mathbb{R}^{N^<} : \begin{array}{l} \tilde{y}(N^<) = f_C(N) - f_C(N^\geq), \\ \tilde{y}(X) \geq f_C(X \cup N^\geq) - f_C(N^\geq)\ (\forall X \subseteq N^<) \end{array} \right\}.$$

[3] We need to check for each X such that $i_2 \in X$ but $i_4 \notin X$. The case when $X \supseteq \{i_2, i_3\}$ follows by $y^* - \chi_{i_3} + \chi_{i_4} \in \overline{\mathbf{B}}_C$, and the case when $i_2 \in X$ but $i_3 \notin X$ follows by $y^* - \chi_{i_2} + \chi_{i_3} \in \overline{\mathbf{B}}_C$.

(the contraction of $\overline{\mathbf{B}}_C$ by $N^{\geq}$), and $\ddot{\mathbf{B}}^{<}$ be the M-convex set induced from $\overline{\mathbf{B}}^{<}$. Then it holds that $\hat{y}^{(t)}_{N^{<}} \in \ddot{\mathbf{B}}^{<}$, $y'_{N^{<}} \in \overline{\mathbf{B}}^{<}$ and $j^* \in \mathrm{supp}^+(\hat{y}^{(t)}_{N^{<}} - y'_{N^{<}})$. We apply Proposition 15 to them. Then we can choose an index $j^{(t)} \in \mathrm{supp}^-(\hat{y}^{(t)}_{N^{<}} - y'_{N^{<}})$ such that $\hat{y}^{(t)}_{N^{<}} - \chi_{j^*} + \chi_{j^{(t)}} \in \overline{\mathbf{B}}^{<}$. Then we can observe that $\hat{y}^{(t)} - \chi_{j^*} + \chi_{j^{(t)}} \in \overline{\mathbf{B}}_C$. Indeed, for any X with $j^* \in X$ and $j^{(t)} \notin X$, since $y^{(t)}(X \cap N^{<}) - 1 \geq f_C((X \cap N^{<}) \cup N^{\geq}) - f_C(N^{\geq})$ and $y^{(t)}(X \cap N^{\geq}) \geq f_C(X \cap N^{\geq})$, we have

$$\begin{aligned} y^{(t)}(X) &= y^{(t)}(X \cap N^{<}) + y^{(t)}(X \cap N^{\geq}) \\ &> f_C(X \cap N^{<}) - f_C(N^{\geq}) + f_C(X \cup N^{\geq}) \geq f_C(X). \end{aligned}$$

Moreover, $z^{(t)}_{j^{(t)}} < y'_{j^{(t)}} + x'_{j^{(t)}} = z'_{j^{(t)}} = z_{j^{(t)}} < \beta$, which implies that

$$\hat{y}^{(t)}_{j^{(t)}} + 1 + x'_{j^{(t)}} = z^{(t)}_{j^{(t)}} + 1 \leq \beta. \tag{5}$$

For simplicity, let $j^{(t)} = j^*$ for each t with $z^{(t)}_{j^*} = \beta$. Then, $y'' \coloneqq \sum_{t=1}^{r} \lambda^{(t)} \cdot (\hat{y}^{(t)} - \chi_{j^*} + \chi_{j^{(t)}}) \in \overline{\mathbf{B}}_C$.

Let $z'' = x' + y''$. Note that this operation to produce z'' first reduces the value of elements more than β while keeping them at least β by (4), and then increases the value of elements less than β while keeping them at most β by (5). In other words, $\beta_1 \leq z''_i \leq z^*_i$ for $i \in N^{>}$, $z''_i = z^*_i$ for $i \in N^{=}$, and $z^*_i \leq z''_i \leq \beta_1$ for $i \in N^{<}$. Therefore, $\Phi(z'') < \Phi(z^*)$ holds. This contradicts to the optimality of z^*. ◀

To prove $\beta_1 - 1 \leq z^*_i \leq \beta_1$, we find a path on the graph by supposing the contrary. While it is easy for the continuous or discrete case, elaborate analysis is required in the hybrid case. Recall that $\beta_1 = \max\{\lceil f_E(X)/|X| \rceil : \emptyset \neq X \subseteq N\}$ by (3).

▶ **Lemma 22.** *$z^*_i \leq \beta_1$ for all $i \in N$.*

Proof. Define the sets $N^{>} = \{i \in N : z^*_i > \beta_1\}$, $N^{=} = \{i \in N : z^*_i = \beta_1\}$, and $N^{<} = \{i \in N : z^*_i < \beta_1\}$. By Lemma 20, $y^*(N^{>}) = f_C(N^{>})$ and $y^*(N^{>} \cup N^{=}) = f_C(N^{>} \cup N^{=})$.

Suppose to the contrary that $N^{>}$ is nonempty. We construct a graph

$$G = \big(N, \big\{(i,j) \in N^2 : x^* - \chi_i + \chi_j \in \ddot{\mathbf{B}}_M \text{ or } y^* - \chi_i + \chi_j \in \overline{\mathbf{B}}_C\big\}\big).$$

We observe that for any Z with $N^{>} \subseteq Z \subseteq N^{>} \cup N^{=}$, it holds that

$$z^*(Z) > f_E(Z) \tag{6}$$

because $f_M(Z) + f_C(Z) = x^*(Z) + y^*(Z) = z^*(Z) > \beta_1 \cdot |Z| \geq f_E(Z) = f_M(Z) + f_C(Z)$, where the last inequality holds by the definition of β_1. This implies the following claim.

▷ Claim 23. For any Z satisfying (6), there exists an edge $(i,j) \in Z \times (N \setminus Z)$.

▷ Claim 24. There exist paths in G from some vertex in $N^{>}$ to some vertex in $N^{<}$.

By the above claim and Lemma 21 with $N' = N$ and $\beta = \beta_1$, there exists a vector z with $\Phi(z) < \phi(z^*)$, which contradicts to the optimality of z^*. This completes the proof of Lemma 22. ◀

We then prove that z^*_i is at least $\beta_1 - 1$ for all $i \in N_1$ by a similar technique to Lemma 22. Recall that N_1 is the smallest subset of N maximizing $f_E(X) - (\beta_1 - 1)|X|$.

▶ **Lemma 25.** *$z^*_i \geq \beta_1 - 1$ for all $i \in N_1$.*

We then show that we cannot decrease the values of elements in N_1 (which have high values) anymore. In the words of fair allocation, this means that goods not required to be assigned to N_1 are not assigned to N_1.

▶ **Lemma 26.** *$x^*(N_1) = f_M(N_1)$, $y^*(N_1) = f_C(N_1)$, and $z^*(N_1) = f_E(N_1)$.*

Proof. It is sufficient to prove that $z^*(N_1) = f_E(N_1)$. Let $N^> := \{i \in N : z^*_i > \beta_1 - 1\}$, $N^= := \{i \in N : z^*_i = \beta_1 - 1\}$, and $N^< := \{i \in N : z^*_i < \beta_1 - 1\}$.

Suppose to the contrary that $z^*(N_1) > f_E(N_1)$. We construct a graph

$$G = \big(N, \{(i,j) \in N \times N : x^* - \chi_i + \chi_j \in \ddot{\mathbf{B}}_M \text{ or } y^* - \chi_i + \chi_j \in \overline{\mathbf{B}}_C\}\big).$$

Since $N_1 \in \arg\max_{S \subseteq N} f_E(S) - (\beta_1 - 1)|S|$, we have $f_E(N_1) - (\beta_1 - 1)|N_1| \geq f_E(X) - (\beta_1 - 1)|X|$ for any $X \subseteq N$. For any X with $N^> \subseteq X \subseteq N^> \cup N^=$, it holds that

$$\begin{aligned}
z^*(X) &= z^*(N^>) + z^*(X \setminus N^>) = z^*(N^>) + (\beta_1 - 1) \cdot (|X| - |N^>|) \\
&\geq z^*(N^> \cap N_1) + (\beta_1 - 1) \cdot (|X| - |N^> \cap N_1|) \\
&= z^*(N_1) - (\beta_1 - 1) \cdot (|N_1| - |N^> \cap N_1|) + (\beta_1 - 1) \cdot (|X| - |N^> \cap N_1|) \\
&\qquad\qquad \text{(by Lemma 25)} \\
&= z^*(N_1) - (\beta_1 - 1)|N_1| + (\beta_1 - 1)|X| \\
&> f_E(N_1) - (\beta_1 - 1)|N_1| + (\beta_1 - 1)|X| \qquad \text{(by assumption)} \\
&\geq f_E(X) - (\beta_1 - 1)|X| + (\beta_1 - 1)|X| = f_E(X).
\end{aligned}$$

Hence, by the same proofs of Claims 23 and 24, there exist paths in G from an agent in $N^>$ to an agent in $N^<$. Then, by applying Lemma 21 with $N' = N$ and $\beta = \beta_1 - 1$, we can decrease the value of Φ, which is a contradiction. Hence, we obtain $z^*(N_1) = f_E(N_1)$. This implies that $x^*(N_1) = f_M(N_1)$ and $y^*(N_1) = f_C(N_1)$. ◀

4.2 Decomposition

We describe that we can derive a similar result for $N_2, \ldots, N_q$.

Let $N'_1 = N \setminus N_1$. For a supermodular function f, we denote $f^{(1)} \colon 2^{N'_1} \to \mathbb{Z}$ to be the supermodular function obtained from f by contracting N_1, i.e., $f^{(1)}(X) = f(X \cup N_1) - f(N_1)$ for each $X \subseteq N'_1$. We consider the M-convex set $\ddot{\mathbf{B}}^{(1)}_M$ of $f^{(1)}_M$, integral base-polyhedra $\overline{\mathbf{B}}^{(1)}_C$ of $f^{(1)}_C$, and $\mathbf{B}^{(1)}_E = \ddot{\mathbf{B}}^{(1)}_M + \overline{\mathbf{B}}^{(1)}_C$.

By Lemma 26, we have $z^*_{N'_1} \in \mathbf{B}^{(1)}_E$. In addition, for any $z_{N'_1} \in \mathbf{B}^{(1)}_E$, an extended vector $z = (z^*_{N_1}, z_{N'_1})$ is contained in $\mathbf{B}_E$ because $z(X) \geq f_E(X \cap N_1) + f^{(1)}_E(X \cap N'_1) = f_E(X \cap N_1) + f_E(X \cup N_1) - f(N_1) \geq f_E(X)$ for all $X \subseteq N$ by the supermodularity of f_E. Hence, we obtain the following lemma. Let $\Phi' \colon \mathbb{R}^{N'_1} \to \mathbb{R}$ be the symmetric strictly convex function such that $\Phi'(z_{N'_1}) = \Phi(z^*_{N_1}, z_{N'_1})$.

▶ **Lemma 27.** *For any $z_{N'_1} \in \mathbf{B}^{(1)}_E$, a vector $z = (z^*_{N_1}, z_{N'_1})$ is a Φ-minimizer of $\mathbf{B}_E$ if and only if $z_{N'_1}$ is a Φ'-minimizer of $\mathbf{B}^{(1)}_E$.*

Therefore, we can apply the results in Section 4.1 to N'_1, $f^{(1)}_M$, $f^{(1)}_C$ and $f^{(1)}_E$, and repeat the same procedure. By the definition of the canonical partition and the essential values, we obtain the following lemma.

▶ **Lemma 28.** *For each $j = 1, \ldots, q$, it holds that $\beta_j - 1 \leq z^*_i \leq \beta_j$ for every $i \in N_j$.*

We can prove Theorem 2 by using Theorem 9 and Lemma 28, and basic results for the integral base-polyhedra.

Proof of Theorem 2. Recall that the partition $N_1, \dots, N_q$ is the canonical partition of $\ddot{\mathbf{B}}_E$, and $\beta_1, \dots, \beta_q$ are the corresponding essential values. Let $\hat{N}_1, \dots, \hat{N}_r$ be the principal partition of $\overline{\mathbf{B}}_E$, and let $\lambda_1, \dots, \lambda_r$ be the critical values. We fix $i \in N$ arbitrarily. Let $j \in [q]$ and $k \in [r]$ be the unique indices such that $i \in N_j$ and $i \in \hat{N}_k$. By invoking Theorem 9, we have $\beta_j = \lceil \lambda_k \rceil$. Consequently, by Theorem 8 and Lemma 28, we obtain $z^*_i \leq \beta_j = \lceil \lambda_k \rceil = \lceil \bar{z}_i \rceil$.

To show the other inequality, let $\Phi' \colon \mathbb{R}^N \to \mathbb{R}$ be a symmetric strictly convex function such that $\Phi'(z) = \Phi(-z)$ for $z \in \mathbb{R}^N$. Let $\mathbf{B}'_E = -\mathbf{B}_E$ and $\overline{\mathbf{B}}'_E = -\overline{\mathbf{B}}_E$. Then, $-z^*$ and $-\bar{z}$ are Φ'-minimizers of $\mathbf{B}'_E$ and $\overline{\mathbf{B}}'_E$, respectively. By applying the same argument as above, we obtain $-z^*_i \leq \lceil -\bar{z}_i \rceil$, which is equivalent to $z^*_i \geq \lfloor \bar{z}_i \rfloor$. ◀

4.3 Structures in Fair Allocation

We establish the structure in the case of fair allocations. We partition the goods according to the canonical partition as follows. Let M_1 and C_1 denote the subset of indivisible goods M and divisible goods C, respectively, that must be allocated to agents in N_1. We iteratively define M_j and C_j as the subset of $M \setminus \bigcup_{j'=1}^{j-1} M_{j'}$ and $C \setminus \bigcup_{j'=1}^{j-1} C_{j'}$, respectively, that must be allocated to agents in $\bigcup_{j'=1}^{j} N_{j'}$. In other words, M_j and C_j $(j = 1, \dots, q)$ is defined as

$$M_j = \left\{ g \in M \setminus \textstyle\bigcup_{j'=1}^{j-1} M_{j'} : v_{ig} = 0 \ (\forall i \in N \setminus \bigcup_{j'=1}^{j} N_{j'}) \right\}, \tag{7}$$

$$C_j = \left\{ c \in C \setminus \textstyle\bigcup_{j'=1}^{j-1} C_{j'} : v_{ic} = 0 \ (\forall i \in N \setminus \bigcup_{j'=1}^{j} N_{j'}) \right\}. \tag{8}$$

We refer $M_1, \dots, M_q$ and $C_1, \dots, C_q$ as the canonical partitions of the indivisible goods and the divisible goods, respectively. By Lemmas 26 and 27, we show the following.

▶ **Theorem 29.** *For any allocation π^* whose utility vector is a Φ-minimizer over $\mathbf{B}_E$, it holds that $\sum_{i \in N_j} \pi^*_{ie} = 1$ for every good $e \in M_j \cup C_j$ and $j = 1, 2, \dots, q$.*

Proof. Let z^* be the utility vector of π^*. Let x^* and y^* be the utility vectors for indivisible and divisible goods in π^*, respectively. Thus, $z^* = x^* + y^*$.

First, since $f_M(N_1) = |M_1|$ and $f_C(N_1) = |C_1|$, we have $z^*(N_1) = |M_1 \cup C_1|$ by Lemma 26. Next, let $N'_j = N \setminus \bigcup_{j'=1}^{j} N_{j'}$ for $j = 1, \dots, q-1$. By Lemma 27, $z^*_{N'_1}$ is a Φ'-minimizer, where $\Phi'(z) = \phi(z^*_{N_1}, z)$. Thus by the definition of N_2 and Lemma 26 again, $x^*_{N'_1}(N_2) = f^{(1)}_M(N_2) = f_M(N_1 \cup N_2) - f_M(N_1) = |M_2|$ and $y^*_{N'_1}(N_2) = f^{(1)}_C(N_2) = |C_2|$. By iteratively applying this argument, we observe that $z^*(N_j) = |M_j \cup C_j|$ for $j = 2, \dots, q$.

Because agents in N_q want only the goods in $M_q \cup C_q$, these goods are allocated to the agents in N_q. Then, for each $j = q-1, \dots, 1$, since agents in N_j want only the goods in $\bigcup_{j'=j}^{q} (M_{j'} \cup C_{j'})$ but the goods in $\bigcup_{j'=j+1}^{q} (M_{j'} \cup C_{j'})$ are allocated to agents in $N_{j+1} \cup \cdots \cup N_q$, the goods in $M_j \cup C_j$ are allocated to agents in N_j. Therefore, the theorem holds. ◀

5 Tractability for Identical Divisible Goods

In this section, we focus on the setting where all the divisible goods are identical, i.e., $v_{ic} = v_{ic'}$ for any $c, c' \in C$ and $i \in N$. Let Φ be a symmetric strictly convex function. The main result of this section is Theorem 4, i.e., a polynomial-time algorithm to find a Φ-fair allocation. As a corollary, an MNW allocation for mixed goods can be found in polynomial time when the divisible goods are identical.

Our algorithm utilizes structures discussed in the previous section. Namely, we seek allocations whose utility vectors satisfy the statements in Lemma 28 and Theorem 29, which are necessary conditions to be optimal. The high-level idea of our algorithm is described as follows.

First, we find a discrete Φ-minimizer $\ddot{z}^* \in \arg\min_{z \in \ddot{\mathbf{B}}_E} \Phi(z)$, the canonical partition $N_1, \ldots, N_q$, and the essential values $\beta_1, \ldots, \beta_q$ of $\ddot{\mathbf{B}}_E$. These can be found in polynomial time by Theorem 9. Note that $\ddot{z}^*$ is an optimal utility vector if every good is assumed to be indivisible. We also compute the canonical partition of the indivisible goods $M_1, \ldots, M_q$ and that of the divisible ones $C_1, \ldots, C_q$ by (7) and (8). Since all divisible goods are identical, only one of $C_1, \ldots, C_q$ can be non-empty. Let j^* be the index such that $C_{j^*} = C$.

Next, we decide on an allocation for agents in N_j for each $j \neq j^*$. Let us fix $j \neq j^*$. Theorem 29 implies that in an optimal allocation, the agents in N_j receive only the indivisible goods in M_j. Moreover, by Lemma 28, some agents in N_j must receive β_j goods and the others must receive $\beta_j - 1$. Because $\beta_j - 1 \leq \ddot{z}^*_i \leq \beta_j$ holds by Theorem 9, we allocate goods in M_j so that each agent $i \in N \setminus N_{j^*}$ receives $\ddot{z}^*_i$ goods. Such an allocation can be computed by solving a bipartite matching problem.[4] For agents in N_j, the utility vector of this allocation is the same as an optimal one up to arrangement of elements. Thus, we have found an optimal allocation for agents in N_j.

The remaining task is to determine the allocation of $M_{j^*} \cup C$ to agents N_{j^*}. Since the optimal allocation of $M_{j^*} \cup C$ depends on Φ, we conduct an enumeration-based approach rather than performing a full characterization.

Let π^* be an optimal allocation. Let $N^+_{j^*}$ be the set of agents in N_{j^*} who desire the divisible goods, i.e., $N^+_{j^*} = \{i \in N_{j^*} : v_i(c) = 1 \ (\forall c \in C)\}$. Let $N^-_{j^*} = N_{j^*} \setminus N^+_{j^*}$. The following lemma indicates that there are a finite number of candidates for a Φ-minimizer.

▶ **Lemma 30.** *All the agents receiving divisible goods (i.e., $\pi^*_i(C) > 0$) have the same utility.*

Let k be the number of agents in $N^+_{j^*}$ who receive β_{j^*} indivisible goods and let ℓ be the total number of indivisible goods received by agents in $N^+_{j^*}$. Note that $k < |N^+_{j^*}|$. The key observation is the following lemma.

▶ **Lemma 31.** *The following properties hold:*

1. $|N^+_{j^*}| \cdot (\beta_{j^*} - 1) + k \leq \ell + |C| \leq |N^+_{j^*}| \cdot \beta_{j^*}$;
2. *there exist $X \subseteq N^+_{j^*}$ such that $|X| = k$ and*
   a. *for each $i \in X$: $\pi^*_i(M_{j^*}) = \beta_{j^*}$ and $\pi^*_i(C) = 0$;*
   b. *for each $i \in N^+_{j^*} \setminus X$: $\pi^*_i(M_{j^*}) \leq \beta_{j^*} - 1$ and $\pi^*_i(E) = \beta_{j^*} - (|N^+_{j^*}| \cdot \beta_{j^*} - \ell - |C|)/(|N^+_{j^*}| - k)$;*
3. *$\pi^*_i(M_{j^*}) \in \{\beta_{j^*}, \beta_{j^*} - 1\}$ and $\pi^*_i(C) = 0$ for each $i \in N^-_{j^*}$.*

Since Lemma 31 specifies the utility vector (i.e., $\pi^*_i(E)$ for each $i \in N_{j^*}$) of an optimal allocation up to arrangement, it suffices to find an allocation whose utility vector satisfies the statement in Lemma 31. In fact, if we are given ℓ, an optimal allocation can be computed as follows. For each $k = 0, \ldots, |N^+_{j^*}|$ such that property 1 in Lemma 31 is satisfied,

1. find an allocation $\pi^{k,\ell} \in \{0,1\}^{N_{j^*} \times M_{j^*}}$ of indivisible goods in M_{j^*} such that (a) $|\{i \in N^+_{j^*} : \pi_i(M_{j^*}) = \beta_{j^*}\}| \leq k$, (b) $\pi_i(M_{j^*}) \leq \beta_{j^*}$ for each $i \in N^+_{j^*}$, (c) $\sum_{i \in N^+_{j^*}} \pi_i(M_{j^*}) = \ell$, (d) $\pi_i(M_{j^*}) \in \{\beta_{j^*}, \beta_{j^*} - 1\}$ for each $i \in N^-_{j^*}$;

[4] It can also be calculated directly with a method of Harvey et al. [26].

2. if $\pi^{k,\ell}$ exists, let $\hat{\pi}^{k,\ell}$ be the allocation by allocating indivisible goods according to $\pi^{k,\ell}$, and allocating divisible goods by a water-filling policy so that

$$\hat{\pi}_i^{k,\ell}(c) = \frac{1}{|C|} \cdot \left(\beta_{j^*} - \frac{|N_{j^*}^+| \cdot \beta_{j^*} - \ell - |C|}{|N_{j^*}^+| - k} - \pi_i^{k,\ell}(M) \right)$$

for each $i \in N_{j^*}^+$ such that $\pi_i^{k,\ell}(M) < \beta_{j^*}$ and $c \in C$.

Let us see that this indeed works. Since $\Phi(\hat{\pi}^{k,\ell}) \leq \Phi(\hat{\pi}^{k+1,\ell})$ holds (if $\pi^{k,\ell}$ exists), the smallest k such that $\pi^{k,\ell}$ exists is the number in Lemma 31. For such an integer k, we have $|\{i \in N_{j^*}^+ : \pi_i(M_{j^*}) = \beta_{j^*}\}| = k$, and the properties in Lemma 31 are satisfied except the allocation of divisible goods. Once we have decided on an allocation of indivisible goods, an optimal allocation of divisible goods is found by the water-filling policy.

Now, we explain how to find an allocation $\pi^{k,\ell}$ at step 1 in polynomial time. We reduce this problem to the submodular flow problem. Let $G = (V, A)$ be a directed graph constructed as follows. The set of vertices V is $M_{j^*} \cup N_{j^*} \cup N'_{j^*}$ where N'_{j^*} is a set of copy i' of each $i \in N_{j^*}$. The set of edges A is $A_1 \cup A_2 \cup A_3$ where $A_1 = \{(g, i') \in M_{j^*} \times N'_{j^*} : v_i(g) = 1\}$, $A_2 = \{(i', i) : i \in N_{j^*}^+\}$, and $A_3 = \{(i', i) : i \in N_{j^*}^-\}$. We define $\underline{c}, \bar{c} \colon A \to \mathbb{Z}$ as $\underline{c}(a) = 0$ and $\bar{c}(a) = 1$ for each $a \in A_1$; $\underline{c}(a) = 0$ and $\bar{c}(a) = \beta_{j^*}$ for each $a \in A_2$; $\underline{c}(a) = \beta_{j^*} - 1$ and $\bar{c}(a) = \beta_{j^*}$ for each $a \in A_3$. In addition, let $f_{k,\ell} \colon 2^V \to \mathbb{Z}$ be a function such that

$$f_{k,\ell}(X) = \varphi_{k,\ell}(|X \cap N_{j^*}^+|) + (|M_{j^*}| - \ell)\mathbf{1}_{X \cap N_{j^*}^- \neq \emptyset} - |X \cap M_{j^*}| \quad (\forall X \subseteq V), \tag{9}$$

where $\varphi_{k,\ell}(h) = \min\{\beta_{j^*} h,\ (\beta_{j^*} - 1)h + k,\ \ell\}$, and $\mathbf{1}_{X \cap N_{j^*}^- \neq \emptyset}$ takes the value 1 if $X \cap N_{j^*}^- \neq \emptyset$ and 0 otherwise. We remark that $f_{k,\ell}$ is a submodular function, and $f_{k,\ell}(V) = 0$ since $\ell \leq \sum_{i \in N_{j^*}^+} \pi_i^*(M_{j^*}) \leq (\beta_{j^*} - 1)|N_{j^*}^+| + k \leq \beta_{j^*}|N_{j^*}^+|$.

▶ **Lemma 32.** *There exists an allocation $\pi \in \{0,1\}^{N_{j^*} \times M_{j^*}}$ satisfying (a)–(d) if and only if there exists an integral flow $\xi \colon A \to \mathbb{Z}$ satisfying $\underline{c}(a) \leq \xi(a) \leq \bar{c}(a)$ (capacity constraints) and a constraint (called supply specification) that the boundary $\partial\xi \in \mathbb{Z}^V$ of the flow ξ, which is defined by $\partial\xi(v) = \sum_{a=(v,u) \in A} \xi(a) - \sum_{a=(u,v) \in A} \xi(a)$, is in the M-convex set $\ddot{\mathbf{B}}$ of $f_{k,\ell}$.*

Since the feasibility of the submodular flow problem can be determined in polynomial time [19], the existence of an allocation satisfying conditions (a)–(d) can be determined in polynomial time by Lemma 32. Moreover, if such an allocation exists, we can find one of such allocations in polynomial time.

Finally, because we do not know ℓ in advance, we enumerate all possibilities. That is, find a best allocation $\pi^{k,\ell}$ for each $\ell = 0, 1, \ldots, |M_{j^*}|$ by the above procedure, and choose the best one. Then the resulting allocation is as good as an optimal allocation π^*.

We give the formal description of our algorithm in Algorithm 1. By summarizing the discussions thus far, we can prove Theorem 4.

6 Hardness for Identical Indivisible Goods

In this section, we show a hardness result on the fair allocation setting when divisible goods are non-identical but indivisible goods are identical. By using Theorem 2, we prove the NP-hardness of finding a Φ-fair allocation by using the 3-dimensional matching (3DM) problem, which is known to be NP-hard [21].

▶ **Theorem 33** (restatement of Theorem 3). *For any fixed symmetric strictly convex function Φ, the problem (1) is NP-hard even in the fair allocation setting with identical indivisible goods. Hence, finding a Φ-fair allocation is NP-hard.*

Algorithm 1 Allocation algorithm when the divisible goods are identical.

input : A fair allocation instance (N, M, C, v) and a symmetric strictly convex function Φ
output : A Φ-fair allocation

Compute the canonical partition $N_1, \dots, N_q$, the essential values $\beta_1, \dots, \beta_q$, the canonical partition of the indivisible goods $M_1, \dots, M_q$, and the canonical partition of the divisible goods $C_1, \dots, C_q$;
Let j^* be the index such that $C_{j^*} = C$;
for $j \leftarrow 1, \dots, j^* - 1, j^* + 1, \dots, q$ **do**
    Allocate M_j to N_j so that each agent receives β_j or $\beta_j - 1$;
Let $N_{j^*}^+ \leftarrow \{j \in N_{j^*} : v_i(c) = 1 \ (\forall c \in C)\}$ and $N_{j^*}^- \leftarrow \{j \in N_{j^*} : v_i(c) = 0 \ (\forall c \in C)\}$;
Let $\Pi \leftarrow \emptyset$ be a set of candidate allocations;
for $k \leftarrow 0, 1, \dots, |N_{j^*}^+|$ *and* $\ell \leftarrow 0, 1, \dots, |M_{j^*}|$ **do**
    if $|N_{j^*}^+| \cdot (\beta_{j^*} - 1) + k \leq \ell + |C| \leq |N_{j^*}^+| \cdot \beta_{j^*}$ **then**
        Determine the existence an allocation $\pi^{k,\ell} \in \{0,1\}^{N_{j^*} \times M_{j^*}}$ satisfying the following conditions via the submodular flow problem:
        $|\{i \in N_{j^*}^+ : \pi_i(M_{j^*}) = \beta_{j^*}\}| \leq k$, $\pi_i(M_{j^*}) \leq \beta_{j^*}$ for each $i \in N_{j^*}^+$,
        $\sum_{i \in N_{j^*}^+} \pi_i(M_{j^*}) = \ell$, $\pi_i(M_{j^*}) \in \{\beta_{j^*}, \beta_{j^*} - 1\}$ for each $i \in N_{j^*}^-$;
        if *Such an allocation* $\pi^{k,\ell}$ *exists* **then**
            Let π be an allocation such that indivisible goods are allocated according to Algorithm 1 and $\pi^{k,\ell}$, and the divisible goods are allocated to agents in $N_{j^*}^+$ by a water-filling policy;
            $\Pi \leftarrow \Pi \cup \{\pi\}$;
return $\pi^* \in \arg\min_{\pi \in \Pi} \Phi(\pi(E))$;

We can also prove the following from the same proof of this theorem.

▶ **Corollary 34.** *The problems of finding an MNW allocation and an optimal egalitarian allocation are both NP-hard, even when indivisible goods are identical.*

References

1 Nima Anari, Tung Mai, Shayan Oveis Gharan, and Vijay V. Vazirani. Nash social welfare for indivisible items under separable, piecewise-linear concave utilities. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2274–2290, 2018.

2 Moshe Babaioff, Tomer Ezra, and Uriel Feige. Fair and truthful mechanisms for dichotomous valuations. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, pages 5119–5126, 2021.

3 S Barman, A Krishna, Y Narahari, and S Sadhukan. Achieving envy-freeness with limited subsidies under dichotomous valuations. In *Proceedings of the 31st International Joint Conference on Artificial Intelligence*, pages 60–66, 2022.

4 Siddharth Barman, Sanath Kumar Krishnamurthy, and Rohit Vaish. Greedy algorithms for maximizing Nash social welfare. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 7–13, 2018.

5 Xiaohui Bei, Zihao Li, Jinyan Liu, Shengxin Liu, and Xinhang Lu. Fair division of mixed divisible and indivisible goods. *Artificial Intelligence*, 293(103436):1–17, 2021.

6 Nawal Benabbou, Mithun Chakraborty, Ayumi Igarashi, and Yair Zick. Finding fair and efficient allocations for matroid rank valuations. *ACM Transactions on Economics and Computation*, 9(4):21:1–21:41, 2021.

7 Umang Bhaskar, AR Sricharan, and Rohit Vaish. On approximate envy-freeness for indivisible chores and mixed resources. In *Proceedings of Approx*, 2021.

8 Johannes Brustle, Jack Dippel, Vishnu V Narayan, Mashbat Suzuki, and Adrian Vetta. One dollar each eliminates envy. In *Proceedings of the 21st ACM Conference on Economics and Computation*, pages 23–39, 2020.

9 Ioannis Caragiannis and Stavros D. Ioannidis. Computing envy-freeable allocations with limited subsidies. In *Proceedings of the 17th Conference on Web and Internet Economics*, pages 522–539, 2022.

10 Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D. Procaccia, Nisarg Shah, and Junxing Wang. The unreasonable fairness of maximum Nash welfare. *ACM Transactions on Economics and Computation*, 7(3):12:1–12:32, 2019.

11 Richard Cole, Nikhil Devanur, Vasilis Gkatzelis, Kamal Jain, Tung Mai, Vijay V. Vazirani, and Sadra Yazdanbod. Convex program duality, Fisher markets, and Nash social welfare. In *Proceedings of the 18th ACM Conference on Economics and Computation*, pages 459–460, 2017.

12 Richard Cole and Vasilis Gkatzelis. Approximating the Nash social welfare with indivisible items. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing*, pages 371–380, 2015.

13 Richard Cole and Vasilis Gkatzelis. Approximating the Nash social welfare with indivisible items. *SIAM Journal on Computing*, 47(3):1211–1236, 2018.

14 Andreas Darmann and Joachim Schauer. Maximizing Nash product social welfare in allocating indivisible goods. *European Journal of Operational Research*, 247(2):548–559, 2015.

15 András Frank and Kazuo Murota. Decreasing minimization on M-convex sets: background and structures. *Mathematical Programming*, 195(1–2):977–1025, 2022.

16 András Frank and Kazuo Murota. Decreasing minimization on M-convex sets: algorithms and applications. *Mathematical Programming*, 195(1–2):1027–1068, 2022.

17 András Frank and Kazuo Murota. Decreasing minimization on base-polyhedra: Relation between discrete and continuous cases. *Japan Journal of Industrial and Applied Mathematics*, 40(1):183–221, 2023.

18 Satoru Fujishige. Lexicographically optimal base of a polymatroid with respect to a weight vector. *Mathematics of Operations Research*, 5(2):186–196, 1980.

19 Satoru Fujishige. *Submodular Functions and Optimization*. Elsevier, 2nd edition, 2005.

20 Satoru Fujishige. Theory of principal partitions revisited. *Research Trends in Combinatorial Optimization: Bonn 2008*, pages 127–162, 2009.

21 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman New York, 1979.

22 Jugal Garg, Martin Hoefer, and Kurt Mehlhorn. Approximating the Nash social welfare with budget-additive valuations. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2326–2340, 2018.

23 Hiromichi Goko, Ayumi Igarashi, Yasushi Kawase, Kazuhisa Makino, Hanna Sumita, Akihisa Tamura, Yu Yokoi, and Makoto Yokoo. A fair and truthful mechanism with limited subsidy. *Games and Economic Behavior*, 144:49–70, 2024.

24 Daniel Halpern, Ariel D. Procaccia, Alexandros Psomas, and Nisarg Shah. Fair division with binary valuations: One rule to rule them all. In *Proceedings of the 16th International Conference on Web and Internet Economics*, pages 370–383, 2020.

25 Daniel Halpern and Nisarg Shah. Fair division with subsidy. In *Proceedings of the 12th International Symposium on Algorithmic Game Theory*, pages 374–389, 2019.

26 Nicholas JA Harvey, Richard E Ladner, László Lovász, and Tami Tamir. Semi-matchings for bipartite graphs and load balancing. *Journal of Algorithms*, 59(1):53–78, 2006.

27 Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM*, 48(4):761–777, 2001.

28 Yasushi Kawase, Kazuhisa Makino, Hanna Sumita, Akihisa Tamura, and Makoto Yokoo. Towards optimal subsidy bounds for envy-freeable allocations. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence*, pages 9824–9831, 2024.

29 Yasushi Kawase, Koichi Nishimura, and Hanna Sumita. Fair allocation with binary valuations for mixed divisible and indivisible goods, 2023. `arXiv:2306.05986`.

30 Euiwoong Lee. APX-hardness of maximizing Nash social welfare with indivisible items. *Information Processing Letters*, 122:17–20, 2017.

31 Zihao Li, Shengxin Liu, Xinhang Lu, and Biaoshuai Tao. Truthful fair mechanisms for allocating mixed divisible and indivisible goods. In *Proceedings of the 32nd International Joint Conference on Artificial Intelligence*, pages 2808–2816, 2023.

32 Shengxin Liu, Xinhang Lu, Mashbat Suzuki, and Toby Walsh. Mixed fair division: A survey. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence*, volume 38, pages 22641–22649, 2024.

33 Xinhang Lu, Jannik Peters, Haris Aziz, Xiaohui Bei, and Warut Suksompong. Approval-based voting with mixed goods. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence*, pages 5781–5788, 2023.

34 F. Maruyama. A unified study on problems in information theory via polymatroids (in Japanese), 1978. Graduation Thesis, University of Tokyo.

35 Satoko Moriguchi, Shinji Hara, and Kazuo Murota. On continuous/discrete hybrid $M^\natural$-convex functions (in Japanese). *Transactions of the Institute of Systems, Control and Information Engineers*, 20:84–86, 2007.

36 Kazuo Murota. Discrete convex analysis. *Mathematical Programming*, 83(1-3):313–371, 1998.

37 Kazuo Murota. *Discrete Convex Analysis*. Society for Industrial and Applied Mathematics, 2003.

38 Kiyohito Nagano. On convex minimization over base polytopes. In *Proceedings of the 12th International Conference on Integer Programming and Combinatorial Optimization*, pages 252–266. Springer, 2007.

39 Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

40 Koichi Nishimura and Hanna Sumita. Envy-freeness and maximum Nash welfare for mixed divisible and indivisible goods. *arXiv:2302.13342*, 2023. `arXiv:2302.13342`.

41 James B. Orlin. Improved algorithms for computing Fisher's market clearing prices: Computing Fisher's market clearing prices. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, pages 291–300, 2010.

42 Alexander Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80(2):346–355, 2000.

43 Erel Segal-Halevi and Balázs R. Sziklai. Monotonicity and competitive equilibrium in cake-cutting. *Economic Theory*, 68(2):363–401, 2019.

44 Yoshiro Takamatsu, Shinji Hgara, and Kazuo Murota. Continuous/discrete hybrid convex optimization and its optimality criterion (in Japanese). *Transactions of the Institute of Systems, Control and Information Engineers*, 17:409–411, 2004.

45 Hal R. Varian. Equity, envy and efficiency. *Journal of Economic Theory*, 9:63–91, 1974.

46 László A. Végh. A strongly polynomial algorithm for a class of minimum-cost flow problems with separable convex objectives. *SIAM Journal on Computing*, 45(5):1729–1761, 2016.

Cut Sparsification and Succinct Representation of Submodular Hypergraphs

Yotam Kenneth
Weizmann Institute of Science, Rehovot, Israel

Robert Krauthgamer
Weizmann Institute of Science, Rehovot, Israel

Abstract

In cut sparsification, all cuts of a hypergraph $H = (V, E, w)$ are approximated within $1 \pm \epsilon$ factor by a small hypergraph H'. This widely applied method was generalized recently to a setting where the cost of cutting each hyperedge e is provided by a splitting function $g_e : 2^e \to \mathbb{R}_+$. This generalization is called a submodular hypergraph when the functions $\{g_e\}_{e \in E}$ are submodular, and it arises in machine learning, combinatorial optimization, and algorithmic game theory.

Previous work studied the setting where H' is a reweighted sub-hypergraph of H, and measured the size of H' by the number of hyperedges in it. In this setting, we present two results: (i) all submodular hypergraphs admit sparsifiers of size polynomial in $n = |V|$ and ϵ^{-1}; (ii) we propose a new parameter, called spread, and use it to obtain smaller sparsifiers in some cases.

We also show that for a natural family of splitting functions, relaxing the requirement that H' be a reweighted sub-hypergraph of H yields a substantially smaller encoding of the cuts of H (almost a factor n in the number of bits). This is in contrast to graphs, where the most succinct representation is attained by reweighted subgraphs. A new tool in our construction of succinct representation is the notion of deformation, where a splitting function g_e is decomposed into a sum of functions of small description, and we provide upper and lower bounds for deformation of common splitting functions.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners; Theory of computation → Submodular optimization and polymatroids; Mathematics of computing → Hypergraphs; Theory of computation → Lower bounds and information complexity

Keywords and phrases Cut Sparsification, Submodular Hypergraphs, Succinct Representation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.97

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2307.09110

Funding This research was partially supported by the Israel Science Foundation grant #1336/23, by a Weizmann-UK Making Connections Grant, by a Minerva Foundation grant, by the Israeli Council for Higher Education (CHE) via the Weizmann Data Science Research Center, and by a research grant from the Estate of Harry Schutzman.

1 Introduction

A powerful tool for many graph problems is sparsification, where an input graph is replaced by a small graph that preserves (perhaps approximately) certain properties, for example all the input graph's cuts [7] or its spectrum [42, 6, 26]. Downstream applications can then be executed on the small graph, which improves the overall running time, and the small graph can also be stored (or sent to another site) instead of the input graph, which improves the memory (or communication) requirements. The extensive research on cut sparsification has started with the seminal work of Benczúr and Karger on cuts in graphs [7], and was later extended to hypergraphs [29, 5, 10] and to directed hypergraphs [40, 9, 27, 36]. In recent


months the study of sparsification has been extended to even more general objects such as semi-norms [25], matroid quotients [38], and linear codes [28]. We focus on sparsifying a generalized form of hypergraphs, as explained next.

In recent years, the notion of cuts in a weighted hypergraph $H = (V, E, w)$ has been generalized to a setting where each hyperedge $e \in E$ has a *splitting function* $g_e : 2^e \to \mathbb{R}_+$, such that $g_e(\emptyset) = 0$, and the *value* of a cut $S \subseteq V$ is defined as

$$\mathrm{cut}_H(S) \coloneqq \sum_{e \in E} g_e(S \cap e). \tag{1}$$

Associating every $e \in E$ with the *all-or-nothing* splitting function, given by $g_e^{\mathrm{aon}} : S \mapsto w_e \cdot \mathbb{1}_{\{S \neq \emptyset, e\}}$, clearly models an ordinary hypergraph $H = (V, E, w)$, where the value of a cut is the total weight of hyperedges that intersect both sides; in fact, a simple extension can model a directed hypergraph. Such a generalized hypergraph $H = (V, E, g)$, where $g = \{g_e\}_{e \in E}$, is called a *submodular hypergraph* if all its splitting functions g_e are submodular. Recall that a set function $g : 2^e \to \mathbb{R}_+$ is *submodular* if

$$\forall S, T \subseteq e, \qquad g(S \cup T) + g(S \cap T) \leq g(S) + g(T).$$

Submodular hypergraphs are useful in clustering data with higher-order relations that are not captured by ordinary hyperedges [31, 32, 44, 34, 45, 48]. For example, the *small-side splitting* function, given by $g_e^{\mathrm{sml}} : S \mapsto \min(|S|, |e \setminus S|)$, is employed when unbalanced cuts are preferable. Cut functions of submodular hypergraphs were studied also under a different name of *decomposable submodular functions*. A submodular function $f : 2^V \to \mathbb{R}_+$ is called *decomposable* if it can be written as $f = \sum_i f_i$, where each $f_i : 2^V \to \mathbb{R}_+$ is submodular. This notion is widely applied in data summarization [23, 33, 43], where each f_i is a submodular similarity function, and the task of summarizing the data under a given budget k is modeled by maximizing $f(S)$ over all $S \subset V$ of size $|S| \leq k$. Decomposable submodular functions arise also in welfare maximization, where each agent has a submodular utility function, for instance in approximation algorithms [16, 17] and in truthful mechanisms [15, 4].

We study how to succinctly represent all the cuts of a submodular hypergraph H up to $1 \pm \epsilon$ factor. We examine two complementary approaches: (1) *sparsification*, which reduces the number of hyperedges, i.e., H is represented using a sparse H'; and (2) *deformation*, which replaces large hyperedges or complicated splitting functions by new ones of low space complexity, i.e., H is represented using H' whose hyperedges can be stored succinctly. These approaches can yield (separately and/or together) a sparsifier H' that can be encoded using a small number of bits. More generally, we may consider a general encoding that *need not* rely on a sparsifier H', e.g., an explicit list of all the $2^{|V|}$ cut values.

Let us introduce some basic notation to make the discussion more precise. Throughout, let $n \coloneqq |V|$; we write $\tilde{O}(t)$ or $\tilde{\Omega}(t)$ to suppress a polylogarithmic factor in t, and $O_\alpha(t)$ or $\Omega_\alpha(t)$ to hide a factor that depends only on α.

▶ **Definition 1.1** (Sparsifier). *A* cut sparsifier *of* quality $1 + \epsilon$ *for* $H = (V, E, g)$*, or in short a* $(1 + \epsilon)$-sparsifier*, is a submodular hypergraph* $H' = (V, E', g')$ *such that*

$$\forall S \subseteq V, \qquad \mathrm{cut}_{H'}(S) \in (1 \pm \epsilon) \cdot \mathrm{cut}_H(S). \tag{2}$$

The size *of the sparsifier is* $|E'|$*. We call* H' *a* reweighted subgraph *of* H *if* $E' \subseteq E$ *and each function* g'_e *for* $e \in E'$ *is a scaling of* g_e *(i.e.,* $g'_e \equiv s_e g_e$ *for some* $s_e > 0$*).*

▶ **Question 1.2** (Sparsification). *Do all submodular hypergraphs admit a reweighted-subgraph sparsifier with few hyperedges, say* $\mathrm{poly}(\epsilon^{-1} n)$*? And which families of splitting functions admit even smaller sparsifiers, like* $\tilde{O}_\epsilon(n^2)$ *or even* $\tilde{O}_\epsilon(n)$*?*

The first question (about a polynomial bound) was previously answered for several families of splitting functions (see Section 1.1 for a detailed account), but despite this significant progress, the case of general submodular splitting was left open in [39], where the bound on the sparsifier size depends on g and is exponential in n in the worst case. We answer this first question in the affirmative, and also address the second question by showing families of splitting functions that admit even smaller sparsifiers.

We further ask about a more general notion, of encoding an approximation of all the cuts of H, which can potentially be more succinct than a sparsifier.

▶ **Question 1.3** (Succinct Representation). *What is the smallest encoding (in bits of space) that stores a submodular hypergraph H so as to report $(1+\epsilon)$-approximation to every cut value? In particular, what is the smallest number of bits $s = s(\epsilon, n)$ that suffices to store a sparsifier for H?*

For simplicity, we ask above only about the existence of a sparsifier or an encoding, but we are of course interested also in fast algorithms to build them. Fortunately, an algorithmic solution follows from the existential ones because our proofs are constructive. Furthermore, the running times are polynomial under the assumption that every g_e takes integral values and $\max_{S \subseteq e} g_e(S) \leq \text{poly}(n)$.[1]

1.1 Sparsification: All Submodular Hypergraphs

We start with addressing Question 1.2. Our first result (proved in Section 2) provides the first polynomial (in n) bound for all submodular splitting functions; the previous bound, due to [39], was $O_\epsilon(n^2 B_H)$, where $B_H := \max_{e \in E} |\mathcal{B}(g_e)|$ and $\mathcal{B}(g_e)$ is the set of extreme points in the polytope of g_e.[2] In general, B_H can be exponential in n, for example small-side splitting g_e^{sml} has $|\mathcal{B}(g_e^{\text{sml}})| = 2^{\Theta(|e|)}$.

▶ **Theorem 1.4.** *Every submodular hypergraph admits a $(1+\epsilon)$-sparsifier of size $O(\epsilon^{-2} n^3)$, which is in fact a reweighted sub-hypergraph.*

This bound is within factor $O_\epsilon(n)$ of the $\Omega(n^2/\epsilon)$ lower bound known for cut sparsification of directed hypergraphs [36]. We also show that if all the splitting functions are monotone (i.e., $g_e(S) \leq g_e(T)$ for all $S \subseteq T$), then the sparsifier size can be improved to $O_\epsilon(n^2)$. Monotone submodular functions arise in many applications, however no sparsification bound was previously known for this family.[3] The formal statement and its proof appear in Section 2.

Related Work. Previous work on sparsification focused mostly on specific splitting functions. The study of this problem began with sparsifiers for undirected graph cut; the current size bound is $O(\epsilon^{-2} n)$ edges [6], which improves over [7] and is known to be tight [2, 8]. Furthermore, sparsifiers of size $\tilde{O}_\epsilon(n)$ are known for all-or-nothing splitting g_e^{aon} [10] (see also [38]) and for *product splitting*, given by $g_e^{\text{prd}} : S \mapsto |S| \cdot |e \setminus S|$ [13]. In contrast, for the splitting that models cuts in a directed hypergraph, the best construction known has size $\tilde{O}_\epsilon(n^2)$ [36], which is near-tight with an $\Omega(n^2/\epsilon)$ lower bound [36]; this function, called

[1] The running times of Theorem 1.4 and Theorem 1.9 are polynomial in general. Theorem 1.6 is polynomial under the stated assumption.

[2] A recent manuscript [30] claims that the proof in [39] has a flaw and holds only for monotone submodular hypergraphs.

[3] The running time of [39] was improved in [30], where a sparsifier of size $O(\epsilon^{-2} n^2 B)$ for monotone functions with low curvature is constructed in polynomial time.

directed all-or-nothing splitting, is given by $g_e^{\text{d-aon}} : S \mapsto \mathbb{1}_{\{e_T \cap S \neq \emptyset \ \wedge \ e_H \not\subseteq S\}}$, where $e_H, e_T \subseteq e$ are the hyperedge's head and tail, respectively. A recent result is more general and shows that the entire family of symmetric splitting functions admits sparsifiers of size $\tilde{O}_\epsilon(n)$ [25].

Figure 1 depicts several families of splitting functions and the sparsification bounds known for them, including our results from above and from Section 1.2.

Techniques. Our sparsification method follows the importance-sampling approach, which has been used extensively in the literature. Every hyperedge $e \in E$ is assigned an importance σ_e, and sampled with probability p_e that is (at least) proportional to σ_e, and the splitting function of every sampled e is scaled by $1/p_e$. The expected sparsifier size is clearly proportional to $\sum_{e \in E} \sigma_e$.

A standard method to set the importance of a hyperedge $e \in E$, is to consider all its possible cuts, namely, $\sigma_e \coloneqq \max_{S \subseteq V} g_e(S \cap e)/\text{cut}_H(S)$, and this method was indeed used in [39]. Bounding $\sum_{e \in E} \sigma_e$ naively by replacing the maximization over $S \subseteq V$ by summation yields an exponential size bound. An improved bound was given in [39] based on a quantity B_H related to the polytopes of the splitting functions. Unfortunately, this improved bound is still exponential for many families of splitting functions.

Our main contribution is to identify a set of "basic" quantities for each hyperedge e that can serve as coarse approximations of its splitting function g_e. These approximations allow us to define new sampling probabilities and achieve an improved size bound: Given $e \in E$, define the minimum directed cut between $u, v \in V$ to be $g_e^{u \to v} \coloneqq \min_{S \subseteq V : u \in S, v \notin S} g_e(S \cap e)$;[4] then our main technical lemma bounds $g_e(\cdot)$ from below and from above by

$$\forall S \subseteq V, \qquad \max_{u \in S, v \in V \setminus S} g_e^{u \to v} \leq g_e(S \cap e) \leq \sum_{u \in S, v \in V \setminus S} g_e^{u \to v}. \tag{3}$$

The lower bound holds by definition, and the upper bound is analogous to bounding the value of a graph cut by the sum of the maximum flows between all pairs of vertices across the cut. It is well-known that importance sampling will produce a sparsifier even if σ_e is replaced with an over-estimate for it. We replace σ_e with $\rho_e \coloneqq \sum_{(u,v) \in V \times V} g_e^{u \to v} / \sum_{f \in E} g_f^{u \to v}$, which we can easily see is an over-estimate, i.e., $\rho_e \geq \sigma_e$, by using the two bounds from (3) to verify that

$$\forall S \subseteq V, \qquad \frac{g_e(S \cap e)}{\text{cut}_H(S)} = \frac{g_e(S \cap e)}{\sum_{f \in E} g_f(S \cap f)} \leq \sum_{u \in S, v \in V \setminus S} \frac{g_e^{u \to v}}{\sum_{f \in E} g_f^{u \to v}} \leq \rho_e.$$

The expected number of hyperedges in the sparsifier H' equals to $\sum_{e \in E} \rho_e$ times an amplification factor M, where $M = O(\epsilon^{-2} n)$ is sufficient by standard arguments (combining a concentration bound and a union bound). The crux here is that it is easy to bound $\sum_{e \in E} \rho_e \leq O(n^2)$, basically swapping the order of a double summation. Another advantage of ρ_e is that it can be computed in polynomial time, while computing σ_e requires maximizing the ratio of two submodular functions, which is NP-hard in general.

In the monotone case, we follow the same approach but employ a simpler over-estimate $\rho'_e \coloneqq \sum_{v \in e} g_e(\{v\})/\text{cut}_H(\{v\})$. The proof is similar to the general case, except that instead of (3) we use the straightforward bound

$$\forall S \subseteq V, \quad \max_{v \in S} g_e(\{v\} \cap e) \leq g_e(S \cap e) \leq \sum_{v \in S} g_e(\{v\} \cap e).$$

[4] The most natural case is $u, v \in e$, but considering all $u, v \in V$ streamlines the presentation.

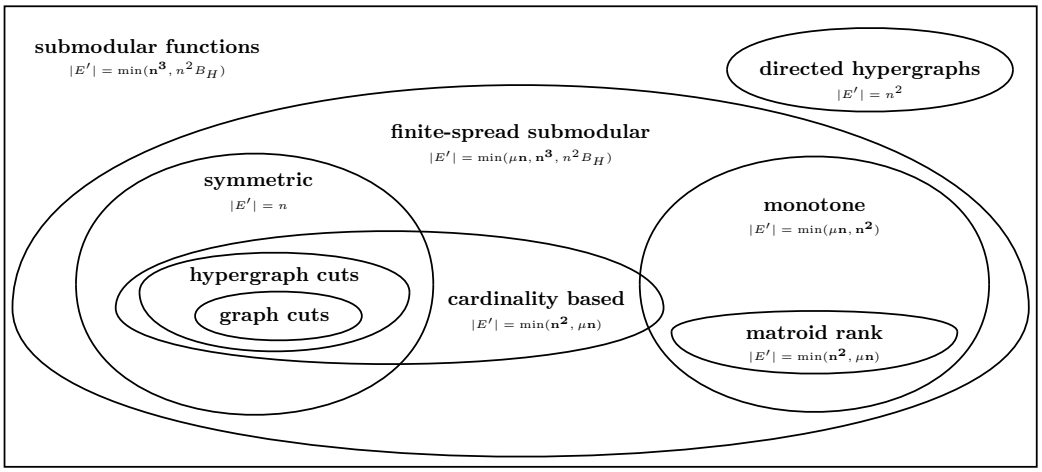


Figure 1 Sparsification bounds for various families of submodular functions, omitting for simplicity $\text{poly}(\epsilon^{-1} \log n)$ factors.

1.2 Sparsification: Parameterized by Spread

We already know that submodular splitting functions can have very different optimal sparsification bounds, see e.g. the bounds $\tilde{\Theta}_\epsilon(n)$ and $\tilde{\Theta}_\epsilon(n^2)$ mentioned above. However, there are too many submodular functions to analyze each one separately, and we thus seek a parameter that can control the sparsifier size. Our approach is inspired by the notion of imbalance in a directed graph $G = (V, E, w)$, defined as the worst ratio between antiparallel edge weights, i.e., $\beta_G := \max\{w(i,j)/w(j,i) : \ i, j \in V\}$. This parameter can be used to show that every directed graph admits a sparsifier of size $\tilde{O}_\epsilon(\beta_G n)$.[5] For submodular hypergraphs, we propose an analogous parameter, which is basically the ratio between the maximum and minimum values of the splitting function, excluding certain trivial cuts.

▶ **Definition 1.5** (Spread). *For hyperedge $e \in E$ with splitting function g_e, let $W_e := \{\emptyset\}$, unless $g_e(e) = 0$ in which case $W_e := \{\emptyset, e\}$. The* spread *of e is*

$$\mu_e := \frac{\max_{T \subseteq e} g_e(T)}{\min_{S \subseteq e : S \notin W_e} g_e(S)}. \tag{4}$$

Our third result (proved in the full version) constructs a sparsifier whose size depends on the spread of the input H, defined as $\mu_H := \max_{e \in E} \mu_e$. By convention, the spread μ_e is called *finite* if it is well-defined (the denominator in (4) is non-zero), and similarly μ_H is called finite if it is well-defined (all the terms μ_e are finite).

▶ **Theorem 1.6** (Sparsifier Parameterized by Spread). *Every submodular hypergraph $H = (V, E, g)$ with finite spread admits a $(1+\epsilon)$-sparsifier of size $\tilde{O}(\epsilon^{-2} \mu_H n)$, which is a sub reweighted-subgraph.*

Many natural submodular functions have finite spread, and in many common cases even $\mu_H \leq n$. This can be seen, for example, in an easy application of Theorem 1.6 to approximation of coverage functions, see the full version for details. Another example is the

[5] This condition can actually be relaxed significantly to $\beta_G := \max\{\text{cut}_G(S)/\text{cut}_G(\bar{S}) : \ S \subset V\}$, and the same sparsification bound still holds [9].

sparsification of the capped version of small-side splitting, given by $g_e : S \mapsto \min(|S|, |e \setminus S|, K)$ for $K > 0$, which clearly has spread $\mu_e \leq K$. This function is part of a much larger family, cardinality-based splitting functions, a notion formalized in [46] as follows: A submodular function $g_e : 2^e \to \mathbb{R}_+$ is called *cardinality-based* if there exists a function $f_e : [|e|] \to \mathbb{R}_+$ such that $g_e : S \mapsto f_e(|S|)$. Cardinality-based functions, which are commonly used in submodular hypergraph clustering, all have spread $\mu_e \leq n$, which is an easy consequence of the symmetry and subadditivity of g_e. By Theorem 1.6, these splitting function admit a $(1+\epsilon)$-sparsifier of size $\tilde{O}(\epsilon^{-2}n^2)$, which is the first bound for this family.

It is easily verified that for monotone splitting functions, the spread is approximately equal to the imbalance, when we generalize the imbalance from above to hyperedges by $\beta_e \coloneqq \max\{g_e(S)/g_e(e \setminus S) : S \subset V\}$.[6] Hence, we immediately obtain the following.

▶ **Corollary 1.7.** *Every finite-spread monotone splitting function admits a $(1+\epsilon)$-sparsisfier of size $\tilde{O}(\epsilon^{-2}\beta_H n)$.*

Two other examples of commonly used monotone functions with finite spread are set-coverage functions (defined in the full version) and the *matroid-rank functions*,[7] which have $\mu_e = r$ where r is the rank of the matroid.

We remark that spread does not fully characterize the sparsifier size. Indeed, symmetric functions can have a large spread μ_e but still admit $\tilde{O}_\epsilon(n)$ sparsifier due to [25], consider e.g. product splitting g_e^{prd} which has $\mu_e = O(n)$. Furthermore, directed all-or-nothing splitting $g_e^{\text{d-aon}}$ does not have finite spread, and nevertheless admits a sparsifier of size $\tilde{O}_\epsilon(n^2)$ [36]. Figure 1 depicts different families of splitting functions including that of finite spread, and the sparsification bounds known for them.

Techniques. Our technique is based on approximate H as an undirected hypergraph and use the sampling probabilities of [10] but amplified by μ_e for each hyperedge. This is a known technique in generalizing sampling mechanisms. Our main contribution is to identify the spread as a relevant and useful parameter. We remark that the generalization of balance, which is known to control the size of sparsifier in directed graphs, to submodular hypergraphs does not suffice for sparsification. Furthermore, we prove that the spread also characterizes other traits of splitting function, such as the deformation lower bound.

1.3 Succinct Representation

We provide the first example of submodular splitting functions for which sparsifiers that are not subgraphs are provably (much) more succinct than sparsifiers that are reweighted subgraphs.[8] To be more precise, we exhibit a natural family of splitting functions, where the former $(1+\epsilon)$-sparsifiers take only $\tilde{O}_\epsilon(n)$ bits (Corollary 1.10), while the latter $(1+\epsilon)$-sparsifiers require $\tilde{\Omega}_\epsilon(n^2)$ bits (Theorem 1.11). It follows that a reweighted subgraph *need not be* the smallest encoding that stores a $(1+\epsilon)$-approximation of the cuts values, and by a wide margin!

---

[6] For a monotone g_e, the spread is $\mu_e = g_e(V)/\min_{v\in V} g_e(\{v\})$ and the imbalance is $\beta_e = \max_{v\in V} g_e(V \setminus \{v\})/g_e(\{v\})$, and they differ by at most a constant factor by the subadditivity of g_e.

[7] For a matroid with ground set e and independent sets $\mathcal{I}$, the rank function is given by $g_e : S \mapsto \max_{T\subseteq S : T\in\mathcal{I}} |T|$. This rank function is submodular and monotone.

[8] Previously, a non-subgraph sparsifier was shown in [1] for small-side splitting, however it optimizes the number of hyperedges and not the encoding size.

Our plan for constructing a succinct representation has two stages. The first stage creates a $(1+\epsilon)$-sparsifier H', by deforming each $e \in E$ into multiple small hyperedges. The second stage computes for this H' a $(1+\epsilon)$-sparsifier H'' that is a reweighted subgraph. It then follows that H'' is a $(1+\epsilon)^2$-sparsifier, and has a few hyperedges that are all small.

▶ **Definition 1.8.** *A splitting function $g_e : 2^e \to \mathbb{R}_+$ on hyperedge e is called $(1+\epsilon)$-approximable with support size p if there are submodular functions $g_{e_i} : 2^{e_i} \to \mathbb{R}_+$ for $i = 1, \ldots, r$, each on a hyperedge $e_i \subseteq e$ of size $|e_i| \leq p$, such that*

$$\forall S \subseteq e, \qquad \sum_{i=1}^{r} g_{e_i}(S \cap e_i) \in (1 \pm \epsilon) g_e(S).$$

Our example is the family of *additive* splitting functions, defined as functions g_e that can be written as either $g_e : S \mapsto \min(|S|, K)$ or $g_e : S \mapsto \min(|S|, |e \setminus S|, K)$ for some $K > 0$. The next theorem (proved in the full version) achieves the first stage in our plan above; it shows that additive functions can be $(1+\epsilon)$-approximated by creating several copies of e and sampling the vertices.

▶ **Theorem 1.9** (Deformation of Additive Functions)**.** *Let g_e be an additive splitting function on hyperedge e. Then g_e can be $(1+\epsilon)$-approximated with support size $O(\epsilon^{-2}(|e|/K)\log|e|)$.*

Following our plan, suppose that given an input H, we first apply Theorem 1.9 to obtain a sparsifier H' with small support size. The construction of H' also implies that it has small spread, $\mu_{H'} \leq O(\epsilon^{-2}\log n)$. Applying Theorem 1.6 on H' we obtain a succinct representation H''. A straightforward encoding of H'' then proves the following corollary (see the full version for details).

▶ **Corollary 1.10** (Additive Functions admit Small Representation)**.** *Let $H = (V, E, \{g_e\})$ be a submodular hypergraph such that every g_e is additive with parameter $K_e > 0$, and let $\hat{K} := \min_{e \in E} K_e/|e|$ be a normalized bound on K_e over all hyperedges. Then H admits a $(1+\epsilon)$-sparsifier with encoding size $O(\epsilon^{-6}\hat{K}^{-1} n \log^4 n)$ bits.*

The next theorem (proved in the full version) shows that reweighted-subgraph sparsifiers of additive functions require $\Omega(n^2)$ bits in the worst-case. Putting this together with our succinct representation from Corollary 1.10, we conclude that relaxing the (natural) restriction to reweighted subgraphs improves the space complexity by a factor of $\tilde{\Omega}_\epsilon(n\hat{K})$, observe that this can be $\tilde{\Omega}_\epsilon(n)$ when $\hat{K} \in \Omega(1)$.

▶ **Theorem 1.11** (Reweighted Sparsifiers Require $\Omega(n^2)$ Bits)**.** *There exists a family $\mathcal{H}$ of hypergraphs with additive splitting functions with parameter $1 \leq K \leq n/3$, such that encoding a reweighted-subgraph $(1+\epsilon)$-sparsifier for an input $H \in \mathcal{H}$ requires $\Omega(n^2)$ bits.*

This lower bound is surprising because in the case of undirected graphs, the best encoding size is achieved by a reweighted-subgraph sparsifier [6, 2, 8]. Our proof is based on a technical lemma that can be applied to many cardinality-based splitting functions. Furthermore, Theorem 1.11 can be extended to the directed all-or-nothing splitting function $g_e^{\text{d-aon}}$, to show a lower bound of $\Omega(n^3/\epsilon)$ bits. For details see the full version.

Finally, we can also prove a space lower bound for an arbitrary encoding of cuts in a directed hypergraph (arbitrary means that it need not represent a reweighted-subgraph sparsifier, see the full version for details). This proof provides an ϵ^{-1} factor improvement over the trivial lower bound of $\Omega\left(n^2\right)$ bits. The proof combines the techniques from Theorem 1.11 with a lower bound from [36] on the number of edges in a reweighted-subgraph sparsifier.

▶ **Theorem 1.12.** *There exists a family of directed hypergraphs $\mathcal{H}$ such that encoding a $(1+\epsilon)$-approximation of their cuts requires $\Omega(n^2/\epsilon)$ bits.*

Techniques. Our lower bound for the encoding size of reweighted-subgraph sparsifiers (Theorem 1.11) boils down to a counting argument on a large family of hypergraphs $\mathcal{H}$, that have sufficiently different cut values and thus require distinct encodings. We construct hypergraphs in this family $\mathcal{H}$ by partitioning the vertices into three parts V, U, W, and adding hyperedges that contain vertices from all three parts. We first create hyperedges consisting of a large random subset of vertices from V; this adds entropy that will differentiate between hypergraphs in $\mathcal{H}$. We then augment each hyperedge with vertices from U, where each hyperedge is defined by a word in the Hadamard code. We use the structure of this code to show that by making cut queries to a hypergraph $H \in \mathcal{H}$, one can recover the random bits encoded in the adjacency matrix of H induced on V. We use W to create an unsparsifiable hypergraph, i.e., one where removing any hyperedge will violate the approximation guarantee. Finally, every hyperedge on $V \cup U$ is combined with a hyperedge on W.

1.4 Deformation Lower Bounds

Our success in finding a small succinct representation for additive functions motivates searching for deformations of other splitting functions.

A similar problem, of approximating a submodular function by functions of small support but over the uniform distribution (i.e., in average-case rather than worst-case), has received significant attention [19, 11, 24, 18, 20], and it is known that every submodular function $f : 2^V \to [0, 1]$ can be approximated within additive error ϵ using support size $O(\epsilon^{-2} \log \epsilon^{-1})$ [20]. We show (see the full version) that a similar result is unfortunately not possible in our setting (multiplicative error for worst-case approximation).

▶ **Theorem 1.13** (Approximation Requires Large Support Size). *Let g_e be an additive splitting function on a hyperedge e. Then every 1.1-approximation of g_e must have support size $p \geq \Omega(|e|/K)$.*

Techniques. The proof of Theorem 1.13 is based on a technical lemma that can be applied to many splitting functions. The main idea is to examine a certain quantity δ_t, which is related the notion of curvature (of a submodular function). The curvature is often used to parameterize approximation guarantees in maximization of submodular optimization [12, 47]. Intuitively, both the curvature and δ_t characterize the locality of the function, i.e., how much error is introduced by decomposing the function into smaller parts and summing them. The main difference between the two quantities is that the curvature looks at the marginal contributions and δ_t characterizes the curvature of the union of two sets of size t. Furthermore, in the approximation setting, a low worst-case curvature is desirable while for our proof it suffices that δ_t is high for many sets of size t. Specifically, we show that if a constant fraction of pairs of subsets of size t have constant positive δ_t, then g_e cannot be approximated with support size smaller than $O(\delta_t^2 n/t)$.

By applying the technical lemma, we obtain lower bounds on the support size required to approximate several natural splitting functions, as presented in Table 1.

1.5 Related Work

Submodular functions appear in many applications, and have been studied extensively in the literature. In particular, the problem of finding a simple representation for submodular functions has been studied in several works. An $O(\sqrt{n} \log n)$-approximation for monotone submodular functions by functions of the form $f(S) = \sqrt{\sum_{v \in S} c_v}$, where $c_v > 0$ are weights for all $v \in V$, was obtained in [22]. A later result [14] showed the same approximation using

Table 1 Our lower bounds on the support size for several families of splitting functions. They are all obtained by applying the technical lemma, stated for simplicity for sufficiently small fixed $\epsilon > 0$ and $|e| = n$.

Function Family	Example	Support Size	See
additive functions	$g_e(S) = \min(\lvert S\rvert, K)$	$\Omega(n/K)$	Lemma 1.13
polynomial	$g_e(S) = \lvert S\rvert^\alpha$ for $\alpha \in (0,1)$	$\Omega(n)$	The full version
logarithmic	$g_e(S) = \log(\lvert S\rvert + 1)$	$\Omega(n)$	The full version
cardinality based	$g_e(S) = f(\lvert S\rvert)$ for concave f	$\Omega\left(n/\mu_e^{1.5}\right)$	The full version
unweighted	$g_e(v) = 1$ for all $v \in V$	$\Omega\left(n/\mu_e^{3}\right)$	The full version

coverage and budget-additive functions. The same paper also provided a lower bound of $\Omega(n^{1/3}\log^{-2} n)$ for approximating monotone submodular functions by coverage and budget additive. Approximating the all-or-nothing splitting function on n vertices using hyperedges with the all-or-nothing function and with support size r must incur approximation factor $\Omega(n/r)$ [37, Section 2.3].

It was previously shown that every symmetric cardinality-based splitting functions can be deformed into a sum of $|e|/2$ hyperedges with capped small-side splitting function, while preserving the value of g_e exactly [46]. Subsequent work by the same authors [45], achieves a similar deformation but with $(1+\epsilon)$-approximation and using only $O(\epsilon^{-1}\log|e|)$ hyperedges. Notice the difference from our work, which focuses on an approximation with small support size.

1.6 Concluding Remarks

Our work provides several promising directions for future work. We prove that all submodular hypergraph admit sparsifiers of polynomial size (Theorem 1.4), leaving a gap of $\tilde{\Omega}_\epsilon(n)$ between the upper and lower bounds. We conjecture that submodular hypergraphs admit the same sparsification bounds as (the special case of) directed hypergraphs.

▶ **Conjecture 1.14.** *Every submodular hypergraph admits a $(1+\epsilon)$-sparsifier of size $O(\epsilon^{-2}n^2)$, which is in fact a reweighted sub-hypergraph.*

Notice that the known lower bound of $\Omega(n^2/\epsilon)$ is not tight with this conjecture, and improving it is an interesting open problem. The main challenge in bridging the gap between our upper bound in Theorem 1.4 and the conjecture is the use of a union bound over all 2^n cuts. This challenge was overcome in graph and hypergraph sparsification by different methods, such as cut counting [7, 21, 10, 28], a matrix Chernoff bound [41], and chaining which uses progressively finer discretizations [5, 27, 36, 25]. Unfortunately, the matrix Chernoff bound is based on linear-algebra tools that are clearly inapplicable to hypergraphs. The cut-counting methods partition the cuts so that a union bound can be applied separately on each part; however these partitions rely on the binary nature of the all-or-nothing splitting function, which seems challenging in the submodular hypergraph setting, because the same g_e can contribute very different values to different cuts $S \subseteq V$. The chaining methods seem more promising, especially the recent one [25] for all symmetric submodular functions, in which the contribution of a single g_e is not binary, although it seems to rely on the splitting functions being symmetric.

In the sparsification setting, we obtain smaller sparsifiers for several families (monotone and finite-spread), however characterizing the optimal sparsifier size for each family remains open. In the succinct-representation setting, we found a useful deformation only for additive splitting functions (Theorem 1.9), and it would be desirable to find deformations for more families.

Another interesting avenue is to find applications or connections to other problems. For example, we show that Theorem 1.6 can be used to approximate a set-coverage function using a small ground set. Another potential application is constructing succinct representations for terminal cuts in a graph, see the full version for details on both applications.

2 Polynomial-Size Sparsifiers for Submodular Hypergraphs

This section proves Theorem 1.4 and its improvement in the monotone case. Our sparsification method is based on importance sampling, where hyperedges are sampled with probability that is (at least) proportional to their maximum relative contribution to any cut. A standard choice, that was indeed used in [39], is to sample every $e \in E$ with probability exactly proportional to its importance, defined as

$$\sigma_e \coloneqq \max_{S \subseteq V} \frac{g_e(S \cap e)}{\sum_{f \in E} g_f(S \cap f)}.$$

The expected size of this sparsifier is proportional to the total importance $\sum_{e \in E} \sigma_e$, which is non-trivial to bound (e.g., naively replacing the maximization over $S \subseteq V$ by summation yields an exponential size bound). An improved bound on the size of a sparsifier constructed in this manner is given in [39], based on a quantity B_H related to the polytopes of the splitting functions. Unfortunately, this improved bound is still exponential for many families of splitting functions.

Our approach achieves a polynomial bound by using a different set of sampling probabilities and a different analysis. Our main insight is that it suffices to consider only a few cuts. Formally, define the minimum directed cut of g_e between $(u, v) \in V \times V$ as

$$g_e^{u \to v} \coloneqq \min_{S \subseteq V : u \in S, v \notin S} g_e(S \cap e). \tag{5}$$

Notice that we do not require $u, v \in e$; clearly, $g_e^{u \to v} = 0$ if $u \notin e$, but $g_e^{u \to v}$ can be positive if $v \notin e$. Our sampling probabilities are proportional to

$$\rho_e \coloneqq \sum_{(u,v) \in V \times V} \frac{g_e^{u \to v}}{\sum_{f \in E} g_f^{u \to v}},$$

where by convention the fraction is equal to zero if the denominator (and thus also the numerator) is zero. The proof follows by showing that $\rho_e \geq \sigma_e$, hence sampling every $e \in E$ with probability proportional to ρ_e suffices to approximate the cuts, and that the expected number of hyperedges in the sparsifier $O(\epsilon^{-2} n^3)$. Since $\rho_e \geq \sigma_e$, our analysis implies that the same size bound holds also for sampling with probabilities proportional to σ_e, i.e., for the sparsifier of [39] but with our amplification factor $M = O(\epsilon^{-2} n)$.

Finally, observe that the directed minimum cuts $g_e^{u \to v}$ can be computed in polynomial time using standard submodular minimization techniques [35].[9] In contrast, calculating σ_e requires maximizing the ratio of two submodular functions, which is NP-hard. In the monotone case, previous work had achieved a polynomial running time [39, 30].

Proof of Theorem 1.4. Our construction of a quality $(1 + \epsilon)$-sparsifier for H utilizes the importance sampling method, where each hyperedge is sampled independently with probability p_e that is defined below, and the splitting functions of every sampled hyperedge d is scaled by factor $1/p_e$.

[9] In fact, computing an $O(1)$-approximation to ρ_e would suffice, and this may be used to speed up the computation, at the cost of increasing the sparsifier size only by a constant factor.

We will use the following claim to bound cuts of H by minimum directed cuts. Throughout, we denote $\bar{S} = V \setminus S$.

▷ Claim 2.1. For every $e \in E$ and $S \subset V$,

$$\max_{u \in S, v \in \bar{S}} g_e^{u \to v} \leq g_e(S \cap e) \leq \sum_{u \in S} \sum_{v \in \bar{S}} g_e^{u \to v}.$$

The proof of Claim 2.1 appears later. Intuitively, it is similar to bounding the capacity of a cut in a graph by the sum of maximum flows between each vertex from S and each vertex from $\bar{S}$. We proceed assuming this claim, to show that $\rho_e \geq \sigma_e$.

▶ **Corollary 2.2.** *For every $e \in E$ and $S \subseteq V$, we have $\rho_e \geq g_e(S \cap e)/\mathrm{cut}_H(S)$.*

Proof. By Claim 2.1, using both the upper bound and the lower bound on $g_e(\cdot)$,

$$\frac{g_e(S \cap e)}{\mathrm{cut}_H(S)} = \frac{g_e(S \cap e)}{\sum_{f \in E} g_f(S \cap f)} \leq \sum_{u \in S, v \in \bar{S}} \frac{g_e^{u \to v}}{\sum_{f \in E} g_f^{u \to v}} \leq \rho_e.$$

Note that the first inequality holds even if $\mathrm{cut}_H(S) = 0$, by our convention that if the denominator (and thus also numerator) is zero then the fraction is zero. ◀

For every hyperedge $e \in E$, set $\rho'_e := g_e(e)/\sum_{f \in E} g_f(f)$ as the importance of the cuts that contain the entire hyperedge (the case $S = V$), and let $p_e := \min(1, M(\rho_e + \rho'_e))$ for a suitable parameter $M = O(\epsilon^{-2} n)$. Now sample every hyperedge $e \in E$ independently with probability p_e and rescale the splitting functions of every sampled hyperedge by factor $1/p_e$. Let H' be the resulting hypergraph.

We first prove that the number of hyperedges in the sparsifier H' is $O(Mn^2)$, which satisfies the claimed size bound by our choice of $M = O(\epsilon^{-2} n)$. Let I_e be an indicator for the event that the hyperedge e is sampled into H'. The expected number of sampled hyperedges is

$$\begin{aligned}\mathbb{E}\left[\sum_{e \in E} I_e\right] = \sum_{e \in E} p_e &\leq M \sum_{e \in E} \left(\frac{g_e(e)}{\sum_{f \in E} g_f(f)} + \sum_{(u,v) \in V \times V} \frac{g_e^{u \to v}}{\sum_{f \in E} g_f^{u \to v}} \right) \\ &\leq M \left(1 + \sum_{(u,v) \in V \times V} \frac{\sum_{e \in E} g_e^{u \to v}}{\sum_{f \in E} g_f^{u \to v}} \right) \leq Mn^2,\end{aligned}$$

where the second inequality follows by changing the order of summation and the last one is because $|V \times V| = n^2$, but we can exclude from the summation the case $u = v$ (as it contributes 0 by our convention). By Markov's inequality, with high constant probability the sparsifier has at most $O(Mn^2)$ hyperedges.

Let us prove that the sparsifier H' indeed approximates the cuts of H. Fix some $S \subseteq V$ and notice that

$$\begin{aligned}\mathbb{E}\left[\mathrm{cut}_{H'}(S)\right] &= \mathbb{E}\left[\sum_{e \in E} I_e \cdot \frac{1}{p_e} g_e(S \cap e)\right] = \sum_{e \in E} \frac{g_e(S \cap e)}{p_e} \cdot \mathbb{E}\left[I_e\right] \\ &= \sum_{e \in E} g_e(S \cap e) = \mathrm{cut}_H(S).\end{aligned}$$

Hence, the cut is preserved in expectation. We shall now prove that the value of the cut is concentrated around its expectation. Let $Q_S = \{e \in E : p_e \in (0,1) \wedge g_e(S \cap e) > 0\}$ be the set of all hyperedges whose contribution to $\mathrm{cut}_{H'}(S)$ is random. Furthermore, denote the maximum contribution of any such hyperedge to $\mathrm{cut}_{H'}(S)$ by $b \coloneqq \max_{e \in Q_S} p_e^{-1} g_e(S \cap e)$. By the Chernoff bound for bounded variables (Theorem A.1),

$$\Pr\left[\mathrm{cut}_{H'}(S) \notin (1 \pm \epsilon) \cdot \mathrm{cut}_H(S)\right] \leq 2 \cdot \exp\left(-\frac{\epsilon^2 \cdot \mathrm{cut}_H(S)}{b}\right). \tag{6}$$

We first analyze the special case $S = V$. Observe that if $\mathrm{cut}_H(V) = 0$ then the cut is preserved trivially. Otherwise, note that $p_e \geq M\rho'_e = \frac{M g_e(e)}{\sum_{f \in E} g_f(f)}$ and hence

$$b = \max_{e \in Q_V} \frac{g_e(e)}{p_e} \leq \max_{e \in Q_V} g_e(e) \frac{\sum_{f \in E} g_f(f)}{M g_e(e)} = \frac{\mathrm{cut}_H(V)}{M}.$$

Plugging this into Equation (6), we find $\Pr\left[\mathrm{cut}_{H'}(V) \notin (1 \pm \epsilon) \cdot \mathrm{cut}_H(V)\right] \leq 2 \cdot \exp\left(-\epsilon^2 M\right)$. Now turning to the general case $S \subset V$, observe that by Corollary 2.2, $p_e \geq M g_e(S \cap e)/\mathrm{cut}_H(S)$. Hence, we again obtain that

$$b \leq \max_{e \in E} g_e(S \cap e) \cdot \frac{\mathrm{cut}_H(S)}{M g_e(S \cap e)} = \frac{\mathrm{cut}_H(S)}{M}. \tag{7}$$

Plugging this back into our concentration bound, Equation (6), we get

$$\Pr\left[\mathrm{cut}_{H'}(S) \notin (1 \pm \epsilon) \cdot \mathrm{cut}_H(S)\right] \leq 2 \cdot \exp\left(-\epsilon^2 M\right).$$

Notice that this is the same probability as the case $S = V$. Setting $M \coloneqq c \cdot \epsilon^{-2} n$ for large enough but fixed $c > 0$, we get that $\mathrm{cut}_{H'}(S)$ approximates $\mathrm{cut}_H(S)$ up to a $1 \pm \epsilon$ factor with probability at least $1 - 2\exp(-cn)$. Applying a union bound over all $S \subseteq V$ we get that the sparsifier approximates all cuts simultaneously with probability at least $1 - 2\exp(-cn) \cdot 2^n \geq 1 - 2\exp(-n)$. This completes the construction of a quality $1 + \epsilon$ sparsifier for H with $O(\epsilon^{-2} n^3)$ hyperedges.

We now turn back to proving Claim 2.1.

Proof of Claim 2.1. Fix some $e \in E$ and $S \subset V$. For each directed minimum cut, let $P_e^{u \to v} \coloneqq \arg\min_{S \subseteq V : S \cap \{u,v\} = \{u\}} g_e(S)$ be some set $S \subseteq V$ attaining the minimum cut value (breaking ties arbitrarily). We need to show that

$$\max_{u \in S, v \in \overline{S}} g_e(P_e^{u \to v}) \leq g_e(S \cap e) \leq \sum_{u \in S} \sum_{v \in \overline{S}} g_e(P_e^{u \to v}). \tag{8}$$

The lower bound is immediate because $g_e(P_e^{u \to v})$ is a minimizer over the cuts separating u from v. For the upper bound, since g_e is submodular and non-negative,

$$\forall A, B \subseteq e, \quad g_e(A) + g_e(B) \geq g_e(A \cap B) + g_e(A \cup B) \geq g_e(A \cap B),$$

and similarly, $g_e(A) + g_e(B) \geq g_e(A \cup B)$. Using these two inequalities and summing over all $v \in \overline{S}$ and $u \in S$, we get

$$\sum_{u \in S} \sum_{v \in \overline{S}} g_e(P_e^{u \to v}) \geq \sum_{u \in S} g_e\left(\bigcap_{v \in \overline{S}} P_e^{u \to v}\right) \geq g_e\left(\bigcup_{u \in S} \bigcap_{v \in \overline{S}} P_e^{u \to v}\right).$$

To conclude the proof we show that $S \cap e = \bigcup_{u \in S} \bigcap_{v \in \overline{S}} P_e^{u \to v}$. For all $u \in S \cap e$ we have $\{u\} \subseteq \bigcap_{v \in \overline{S}} P_e^{u \to v}$, therefore $S \cap e \subseteq \bigcup_{u \in S} \bigcap_{v \in \overline{S}} P_e^{u \to v}$. In addition, for all $u \in S$ we have $\bigcap_{v \in \overline{S}} P_e^{u \to v} \subseteq S \cap e$ if $u \in e$ and $P_e^{u \to v} = \emptyset$ otherwise, therefore $S \cap e = \bigcup_{u \in S} \bigcap_{v \in \overline{S}} P_e^{u \to v}$. We conclude that Equation (8) holds. ◁

This completes the proof of Theorem 1.4. ◀

2.1 Monotone Submodular Hypergraphs

This section proves that every monotone submodular hypergraph admits a quality $(1+\epsilon)$-sparsifier of size $O(\epsilon^{-2}n^2)$.

▶ **Theorem 2.3.** *Every hypergraph with monotone splitting functions admits a quality* $(1+\epsilon)$*-sparsifier of size* $O(\epsilon^{-2}n^2)$*, which is a reweighted sub-hypergraph.*

The proof for the monotone case is similar to the general case. However, since monotone splitting functions are more structured it suffices to examine the importance of all the singleton cuts for each hyperedge. This results in smaller sampling probabilities and a better bound on the number of hyperedges in the sparsifier. The proof utilizes the following well known property of monotone submodular functions.

▷ Claim 2.4. Let $g_e : 2^e \to \mathbb{R}_+$ be a monotone submodular splitting function. Then

$$\forall S \subseteq V, \qquad \max_{v\in S} g_e(\{v\}\cap e) \le g_e(S\cap e) \le \sum_{v\in S} g_e(\{v\}\cap e).$$

Proof. The lower bound holds as g_e is monotone. For the upper bound, since g_e is submodular and non-negative,

$$\sum_{v\in S} g_e(\{v\}\cap e) \ge g_e\left(\bigcup_{v\in S}\{v\}\cap e\right) = g_e(S\cap e). \qquad \blacktriangleleft$$

Similarly to the general case, our over sampling probabilities are proportional to

$$\rho_e = \sum_{v\in V} \frac{g_e(\{v\}\cap e)}{\sum_{f\in E} g_f(\{v\}\cap f)}.$$

The following corollary shows that $\rho_e \ge \sigma_e$. This implies that sampling every $e \in E$ with probability proportional to ρ_e suffices to approximate the cuts of H, in the same manner as in the general case.

▶ **Corollary 2.5.** *For every* $e \in E$ *and* $S \subseteq V$*, we have* $\rho_e \ge g_e(S\cap e)/\mathrm{cut}_H(S)$.

Proof. Observe that by Claim 2.4,

$$\frac{g_e(S\cap e)}{\mathrm{cut}_H(S)} = \frac{g_e(S\cap e)}{\sum_{f\in E} g_f(S\cap f)} \le \sum_{v\in S} \frac{g_e(\{v\}\cap e)}{\sum_{f\in E} g_f(\{v\}\cap f)} \le \rho_e.$$

Notice that the first inequality is well-defined by the convention that if the denominator (and thus also the numerator) is zero then the fraction is zero. ◀

We now turn to proving Theorem 2.3

Proof of Theorem 2.3. To construct H', sample each hyperedge with probability $p_e = \min(1, M\cdot\rho_e)$ for a suitable parameter $M = O(\epsilon^{-2}n)$. Then, reweigh every sampled hyperedge by factor p_e^{-1}. The proof that H' is with high probability a $(1+\epsilon)$-sparsifier is similar to the general case because $\rho_e \ge \sigma_e$, and we omit it.

To bound the number of hyperedges in the sparsifier, let I_e be an indicator for the event that the hyperedge e is sampled into H'. Then the expected number of sampled hyperedges is,

$$\begin{aligned}\mathbb{E}\left[\sum_{e\in E} I_e\right] &= \sum_{e\in E} p_e \le M \sum_{e\in E}\sum_{v\in V} \frac{g_e(\{v\}\cap e)}{\sum_{f\in E} g_f(\{v\}\cap f)} \\ &\le M \sum_{v\in V}\sum_{e\in E} \frac{g_e(\{v\}\cap e)}{\sum_{f\in E} g_f(\{v\}\cap f)} \le Mn,\end{aligned}$$

where the second inequality is from changing the order of summation. Hence, by Markov's inequality we find that with high constant probability the size of the sparsifier is at most $O(Mn) = O(\epsilon^{-2}n^2)$. This concludes the proof. ◀

References

1 Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS*, pages 335–344. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.44.

2 Alexandr Andoni, Jiecao Chen, Robert Krauthgamer, Bo Qin, David P. Woodruff, and Qin Zhang. On sketching quadratic forms. In *Innovations in Theoretical Computer Science*, ITCS'16, pages 311–319. ACM, 2016. doi:10.1145/2840728.2840753.

3 Alexandr Andoni, Anupam Gupta, and Robert Krauthgamer. Towards $(1 + \epsilon)$-approximate flow sparsifiers. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 279–293. SIAM, 2014.

4 Sepehr Assadi and Sahil Singla. Improved truthful mechanisms for combinatorial auctions with submodular bidders. *SIGecom Exch.*, 18(1):19–27, 2020. doi:10.1145/3440959.3440964.

5 Nikhil Bansal, Ola Svensson, and Luca Trevisan. New notions and constructions of sparsification for graphs and hypergraphs. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 910–928. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00059.

6 Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-Ramanujan sparsifiers. *SIAM Rev.*, 56(2):315–334, 2014. doi:10.1137/130949117.

7 András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 47–55. ACM, 1996. doi:10.1145/237814.237827.

8 Charles Carlson, Alexandra Kolla, Nikhil Srivastava, and Luca Trevisan. Optimal lower bounds for sketching graph cuts. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2565–2569, 2019. doi:10.1137/1.9781611975482.158.

9 Ruoxu Cen, Yu Cheng, Debmalya Panigrahi, and Kevin Sun. Sparsification of directed graphs via cut balance. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 45:1–45:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.45.

10 Yu Chen, Sanjeev Khanna, and Ansh Nagda. Near-linear size hypergraph cut sparsifiers. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 61–72. IEEE, 2020. doi:10.1109/FOCS46700.2020.00015.

11 Mahdi Cheraghchi, Adam R. Klivans, Pravesh Kothari, and Homin K. Lee. Submodular functions are noise stable. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012*, pages 1586–1592. SIAM, 2012. doi:10.1137/1.9781611973099.126.

12 Michele Conforti and Gérard Cornuéjols. Submodular set functions, matroids and the greedy algorithm: Tight worst-case bounds and some generalizations of the Rado-Edmonds theorem. *Discret. Appl. Math.*, 7(3):251–274, 1984. doi:10.1016/0166-218X(84)90003-9.

13 Marcel Kenji de Carli Silva, Nicholas J. A. Harvey, and Cristiane M. Sato. Sparse sums of positive semidefinite matrices. *ACM Trans. Algorithms*, 12(1):9:1–9:17, 2016. doi:10.1145/2746241.

14 Nikhil R. Devanur, Shaddin Dughmi, Roy Schwartz, Ankit Sharma, and Mohit Singh. On the approximation of submodular functions. *CoRR*, abs/1304.4948, 2013. arXiv:1304.4948.

15 Shahar Dobzinski and Michael Schapira. An improved approximation algorithm for combinatorial auctions with submodular bidders. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006*, pages 1064–1073. ACM Press, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109675.

16 Uriel Feige. On maximizing welfare when utility functions are subadditive. *SIAM J. Comput.*, 39(1):122–142, 2009. doi:10.1137/070680977.

17 Uriel Feige and Jan Vondrák. Approximation algorithms for allocation problems: Improving the factor of 1 - 1/e. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006)*, pages 667–676. IEEE Computer Society, 2006. doi:10.1109/FOCS.2006.14.

18 Vitaly Feldman and Pravesh Kothari. Learning coverage functions and private release of marginals. In *Proceedings of The 27th Conference on Learning Theory, COLT 2014*, volume 35 of *JMLR Workshop and Conference Proceedings*, pages 679–702. JMLR.org, 2014. URL: http://proceedings.mlr.press/v35/feldman14a.html.

19 Vitaly Feldman, Pravesh Kothari, and Jan Vondrák. Representation, approximation and learning of submodular functions using low-rank decision trees. In *COLT 2013 - The 26th Annual Conference on Learning Theory*, volume 30 of *JMLR Workshop and Conference Proceedings*, pages 711–740. JMLR.org, 2013. URL: http://proceedings.mlr.press/v30/Feldman13.html.

20 Vitaly Feldman and Jan Vondrák. Optimal bounds on approximation of submodular and XOS functions by juntas. *SIAM J. Comput.*, 45(3):1129–1170, 2016. doi:10.1137/140958207.

21 Wai-Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. *SIAM J. Comput.*, 48(4):1196–1223, 2019. doi:10.1137/16M1091666.

22 Michel X. Goemans, Nicholas J. A. Harvey, Satoru Iwata, and Vahab S. Mirrokni. Approximating submodular functions everywhere. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009*, pages 535–544. SIAM, 2009. doi:10.1137/1.9781611973068.59.

23 Ryan Gomes and Andreas Krause. Budgeted nonparametric learning from data streams. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 391–398. Omnipress, 2010. URL: https://icml.cc/Conferences/2010/papers/433.pdf.

24 Anupam Gupta, Moritz Hardt, Aaron Roth, and Jonathan R. Ullman. Privately releasing conjunctions and the statistical query barrier. *SIAM J. Comput.*, 42(4):1494–1520, 2013. doi:10.1137/110857714.

25 Arun Jambulapati, James R. Lee, Yang P. Liu, and Aaron Sidford. Sparsifying sums of norms. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*, pages 1953–1962. IEEE, 2023. doi:10.1109/FOCS57990.2023.00119.

26 Arun Jambulapati, Victor Reis, and Kevin Tian. Linear-sized sparsifiers via near-linear time discrepancy theory. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 5169–5208. SIAM, 2024. doi:10.1137/1.9781611977912.186.

27 Michael Kapralov, Robert Krauthgamer, Jakab Tardos, and Yuichi Yoshida. Towards tight bounds for spectral sparsification of hypergraphs. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 598–611. ACM, 2021. doi:10.1145/3406325.3451061.

28 Sanjeev Khanna, Aaron Putterman, and Madhu Sudan. Code sparsification and its applications. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 5145–5168. SIAM, 2024. doi:10.1137/1.9781611977912.185.

29 Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015*, pages 367–376. ACM, 2015. doi:10.1145/2688073.2688093.

30 Jannik Kudla and Stanislav Zivný. Sparsification of monotone k-submodular functions of low curvature. *CoRR*, abs/2302.03143, 2023. doi:10.48550/arXiv.2302.03143.

31 Pan Li and Olgica Milenkovic. Inhomogeneous hypergraph clustering with applications. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pages 2308–2318, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/a50abba8132a77191791390c3eb19fe7-Abstract.html.

32 Pan Li and Olgica Milenkovic. Submodular hypergraphs: p-laplacians, cheeger inequalities and spectral clustering. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 3020–3029. PMLR, 2018. URL: http://proceedings.mlr.press/v80/li18e.html.

33 Hui Lin and Jeff A. Bilmes. A class of submodular functions for document summarization. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference 2011*, pages 510–520. The Association for Computer Linguistics, 2011. URL: https://aclanthology.org/P11-1052/.

34 Meng Liu, Nate Veldt, Haoyu Song, Pan Li, and David F. Gleich. Strongly local hypergraph diffusions for clustering and semi-supervised learning. In *WWW '21: The Web Conference 2021*, pages 2092–2103. ACM / IW3C2, 2021. doi:10.1145/3442381.3449887.

35 S Thomas McCormick. Submodular function minimization. *Handbooks in operations research and management science*, 12:321–391, 2005.

36 Kazusato Oko, Shinsaku Sakaue, and Shin-ichi Tanigawa. Nearly tight spectral sparsification of directed hypergraphs. In *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023*, volume 261 of *LIPIcs*, pages 94:1–94:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.94.

37 Yosef Pogrow. Solving symmetric diagonally dominant linear systems in sublinear time (and some observations on graph sparsification). Master's thesis, Weizmann Institute of Science, 2017. URL: https://www.wisdom.weizmann.ac.il/~robi/files/YosefPogrow-MScThesis-2017_12.pdf.

38 Kent Quanrud. Quotient sparsification for submodular functions. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 5209–5248. SIAM, 2024. doi:10.1137/1.9781611977912.187.

39 Akbar Rafiey and Yuichi Yoshida. Sparsification of decomposable submodular functions. In *Thirty-Sixth AAAI Conference on Artificial Intelligence*, pages 10336–10344. AAAI Press, 2022. doi:10.1609/aaai.v36i9.21275.

40 Tasuku Soma and Yuichi Yoshida. Spectral sparsification of hypergraphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 2570–2581. SIAM, 2019. doi:10.1137/1.9781611975482.159.

41 Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM J. Comput.*, 40(6):1913–1926, 2011. doi:10.1137/080734029.

42 Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM J. Comput.*, 40(4):981–1025, 2011. doi:10.1137/08074489X.

43 Sebastian Tschiatschek, Rishabh K. Iyer, Haochen Wei, and Jeff A. Bilmes. Learning mixtures of submodular functions for image collection summarization. In *Advances in Neural Information Processing Systems 27 (NeurIPS 2014)*, pages 1413–1421, 2014. URL: https://proceedings.neurips.cc/paper/2014/hash/a8e864d04c95572d1aece099af852d0a-Abstract.html.

44 Nate Veldt, Austin R. Benson, and Jon M. Kleinberg. Minimizing localized ratio cut objectives in hypergraphs. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1708–1718. ACM, 2020. doi:10.1145/3394486.3403222.

45 Nate Veldt, Austin R. Benson, and Jon M. Kleinberg. Approximate decomposable submodular function minimization for cardinality-based components. In *Advances in Neural Information Processing Systems 34 (NeurIPS 2021)*, pages 3744–3756, 2021. URL: https://proceedings.neurips.cc/paper/2021/hash/1e8a19426224ca89e83cef47f1e7f53b-Abstract.html.

46 Nate Veldt, Austin R. Benson, and Jon M. Kleinberg. Hypergraph cuts with general splitting functions. *SIAM Rev.*, 64(3):650–685, 2022. doi:10.1137/20m1321048.

47 Jan Vondrák. Submodularity and curvature: The optimal algorithm (combinatorial optimization and discrete algorithms). *RIMS Kokyuroku Bessatsu*, 23:253–266, 2010. URL: http://hdl.handle.net/2433/177046.

48 Yu Zhu, Boning Li, and Santiago Segarra. Hypergraph 1-spectral clustering with general submodular weights. In *56th Asilomar Conference on Signals, Systems, and Computers, ACSSC 2022*, pages 935–939. IEEE, 2022. doi:10.1109/IEEECONF56349.2022.10052065.

A Chernoff Bounds

We use the following version of the Chernoff bound throughout the paper.

▶ **Theorem A.1** (Chernoff bound for bounded random variables, Theorem 6.1 in [3]). *Let $X_1, \ldots, X_m \geq 0$ be independent random variables such that either X_i is deterministic or $X_i \in [0, b]$. Let X denote their sum and $\mu = \mathbb{E}[X]$, then,*

$$\forall \delta > 0, \quad \Pr[X - \mu \geq \delta\mu] \leq 2 \cdot \exp\left(-\frac{\delta^2 \mu}{(2+\delta)b}\right).$$

Additionally,

$$\forall \delta \in [0, 1], \quad \Pr[|X - \mu| \geq \delta\mu] \leq 2 \cdot \exp\left(-\frac{\delta^2 \mu}{3b}\right).$$

Almost-Tight Bounds on Preserving Cuts in Classes of Submodular Hypergraphs

Sanjeev Khanna
School of Engineering and Applied Sciences, University of Pennsylvania, Philadelphia, PA, USA

Aaron (Louie) Putterman
School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA

Madhu Sudan
School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA

Abstract

Recently, a number of variants of the notion of cut-preserving hypergraph sparsification have been studied in the literature. These variants include directed hypergraph sparsification, submodular hypergraph sparsification, general notions of approximation including spectral approximations, and more general notions like sketching that can answer cut queries using more general data structures than just sparsifiers. In this work, we provide reductions between these different variants of hypergraph sparsification and establish new upper and lower bounds on the space complexity of preserving their cuts. Specifically, we show that:

1. $(1 \pm \epsilon)$ directed hypergraph spectral (respectively cut) sparsification on n vertices efficiently reduces to $(1 \pm \epsilon)$ undirected hypergraph spectral (respectively cut) sparsification on n^2+1 vertices. Using the work of Lee and Jambulapati, Liu, and Sidford (STOC 2023) this gives us directed hypergraph spectral sparsifiers with $O(n^2 \log^2(n)/\epsilon^2)$ hyperedges and directed hypergraph cut sparsifiers with $O(n^2 \log(n)/\epsilon^2)$ hyperedges by using the work of Chen, Khanna, and Nagda (FOCS 2020), both of which improve upon the work of Oko, Sakaue, and Tanigawa (ICALP 2023).
2. Any cut sketching scheme which preserves all cuts in any directed hypergraph on n vertices to a $(1 \pm \epsilon)$ factor (for $\epsilon = \frac{1}{2^{O(\sqrt{\log(n)})}}$) must have worst-case bit complexity $n^{3-o(1)}$. Because directed hypergraphs are a subclass of submodular hypergraphs, this also shows a worst-case sketching lower bound of $n^{3-o(1)}$ bits for sketching cuts in general submodular hypergraphs.
3. $(1 \pm \epsilon)$ monotone submodular hypergraph cut sparsification on n vertices efficiently reduces to $(1 \pm \epsilon)$ symmetric submodular hypergraph sparsification on $n+1$ vertices. Using the work of Jambulapati et. al. (FOCS 2023) this gives us monotone submodular hypergraph sparsifiers with $\widetilde{O}(n/\epsilon^2)$ hyperedges, improving on the $O(n^3/\epsilon^2)$ hyperedge bound of Kenneth and Krauthgamer (arxiv 2023).

At a high level, our results use the same general principle, namely, by showing that cuts in one class of hypergraphs can be simulated by cuts in a simpler class of hypergraphs, we can leverage sparsification results for the simpler class of hypergraphs.

2012 ACM Subject Classification Theory of computation → Sketching and sampling

Keywords and phrases Sparsification, sketching, hypergraphs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.98

Category Track A: Algorithms, Complexity and Games

Funding *Sanjeev Khanna*: Supported in part by NSF awards CCF-1934876 and CCF-2008305.
Aaron (Louie) Putterman: Supported in part by the Simons Investigator Awards of Madhu Sudan and Salil Vadhan, NSF Award CCF 2152413 and a Hudson River Trading PhD Research Scholarship.
Madhu Sudan: Supported in part by a Simons Investigator Award and NSF Award CCF 2152413.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 98; pp. 98:1–98:17

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Sparsification deals with the following natural question: given a large object, how much can we compress it while still retaining some of its key properties? In the realm of graphs, this has been a well-studied notion spanning decades of research. Starting with the work of Karger [9], the question of how sparse we can make a graph while still preserving the approximate sizes of every cut has been a central topic of research. Since then, numerous works by many authors have resolved this question (starting with the work of Benczúr and Karger [2]) and pushed the boundaries of this research beyond just graph cuts [1, 21, 12, 3].

More rigorously, for a weighted graph $G = (V, E)$ on n vertices, we can define a cut in the graph corresponding to each set $S \subseteq V$. For such a set S, we define the vector $\mathbf{1}_S \in \{0,1\}^{|V|}$ as the indicator vector of whether the ith vertex is in S. Using this vector, we say that $\text{cut}_G(S) = \sum_{(u,v)\in E} w_{(u,v)}(\mathbf{1}_{Su} - \mathbf{1}_{Sv})^2$, i.e., the weight of the edges crossing between S and $V - S$. A cut-sparsifier asks for a reweighted subset of edges $\hat{E} \subseteq E$ such that in the graph $G = (V, \hat{E})$, with the corresponding new weights $\hat{w}$, for every $S \subseteq V$

$$(1-\varepsilon)\text{cut}_G(S) \leq \text{cut}_{\hat{G}}(S) \leq (1+\varepsilon)\text{cut}_G(S).$$

The seminal work of [2] was the first to show the existence of such sparsifiers $\hat{G}$ for any graph G such that $|\hat{E}| = \widetilde{O}(n/\varepsilon^2)$. Subsequent work in the *spectral* regime asked whether such sparsifiers still exist when we consider real-valued vectors as opposed to cut-vectors. In this setting, we define a Laplacian L_G for our graph G. We say that for $x \in \mathbb{R}^{|V|}$

$$x^T L_G x = \sum_{(u,v)\in E} w_{(u,v)}(x_u - x_v)^2.$$

The goal in this regime instead becomes finding a reweighted subgraph $\hat{G}$ such that for every $x \in \mathbb{R}^{|V|}$,

$$(1-\varepsilon)x^T L_{\hat{G}} x \leq x^T L_G x \leq (1+\varepsilon)x^T L_{\hat{G}} x.$$

Work by Batson, Spielman, and Srivastava, and Spielman and Teng [1, 21] settled the size complexity of spectral sparsifiers for ordinary graphs by showing the existence of such sparsifiers of size $O(n/\varepsilon^2)$.

Recently, starting with the work of Kogan and Krauthgamer [12], a natural extension to the study of graph sparsification has been the study of sparsifying *hypergraphs*. In this setting, one is given a hypergraph $H = (V, E)$, and asked to preserve to a $(1 \pm \varepsilon)$ factor the weight of all hyperedges crossing a particular cut. A cut is given by a bichromatic coloring of the vertices and a hyperedge is considered cut if it is not monochromatic. Work by Chen, Khanna, and Nagda [3] was the first to completely characterize the cut-sparsifiability of hypergraphs, which showed that there exist $(1 \pm \varepsilon)$-cut-sparsifiers for any hypergraph on n vertices of size $O(n \log(n)/\varepsilon^2)$. As in the graph setting, where the natural next step from cut-sparsifiers was spectral-sparsifiers, Soma and Yoshida [20] later introduced this notion of spectral *hypergraph* sparsification. More explicitly, the "energy function" (also called the Laplacian) of an undirected hypergraph $H = (V, E)$ is as follows:

$$L_H(x) = \text{cut}_H(x) = \sum_{e\in E} w_e \max_{u,v\in e}(x_u - x_v)^2.$$

A $(1 \pm \varepsilon)$-spectral sparsifier for an undirected hypergraph then corresponds to a reweighted subhypergraph of H, denoted by $\hat{H}$ such that for any $x \in \mathbb{R}^{|V|}$,

$$(1-\varepsilon)L_H(x) \leq L_{\hat{H}}(x) \leq (1+\varepsilon)L_H(x).$$

This question of whether one could preserve the Laplacian of undirected hypergraphs with only a near-linear number of hyperedges was then resolved by Kapralov et. al. [7], Jambulapati, Liu, and Sidford [6], and Lee [14] in the affirmative.

More recently however, work has sought to generalize hypergraph sparsification even further. Indeed, given a hypergraph $H = (V, E)$, instead of viewing edge-cuts in the traditional way (i.e., for a bichromatic coloring of the vertices counting how many hyperedges are not one color), a more general *splitting* function is assigned to each hyperedge $e \subseteq V$. This splitting function is a set function $g_e : 2^e \to \mathbb{R}^{\geq 0}$. One natural extension to the case of ordinary hypergraphs that has received particular attention is the case in which these splitting functions g_e are also required to be submodular [10, 13] (though there has also been work on the regime where these functions are not submodular, for instance with parity functions in [11]). Such submodular hypergraphs appear in numerous contexts, for instance in clustering data points with complex relationships [15, 16, 22, 23] and summarizing data [17]. For such a submodular hypergraph $H = (V, E)$, the value on any cut $S \subset V$ is

$$\text{cut}_H(S) = \sum_{e \in E} g_e(S \cap e).$$

Recall that a function $g : 2^V \to \mathbb{R}^{\geq 0}$ is said to be *submodular* if it has the property of diminishing returns. That is, for any $S \subset T \subset V$, and any element $x \in V, x \notin T$,

$$g(S \cup \{x\}) - g(S) \geq g(T \cup \{x\}) - g(T).$$

Under this definition, one type of submodular hypergraph is a *directed hypergraph*. In a directed hypergraph, one can view each directed hyperedge instead as a tuple $(e_{\text{tail}}, e_{\text{head}})$ of subsets of V. The cut function of a directed hyperedge e on cut S is 1 if and only if an element from S is in e_{tail} and an element from $V - S$ is in e_{head}. More explicitly, for a directed hypergraph $G = (V, E, w)$ on n vertices, and a vector $x \in \mathbb{R}^n$, we can define the Laplacian for G as

$$L_G(x) = \sum_{e \in E} \max_{u \in L(e), v \in R(e)} (x_u - x_v)_+^2.$$

In this context, $(x_u - x_v)_+ = \max((x_u - x_v), 0)$, and directed hypergraph cuts are simply the restriction of the vector x to be in $\{0, 1\}^{|V|}$ (seen as the indicator vector for a set $S \subseteq V$). A non-zero contribution from a hyperedge occurs only if a tail vertex of the hyperedge has a larger value than a head vertex of the hyperedge.

One can check that in the cut regime (i.e. $x \in \{0, 1\}^n$), each directed hyperedge cut yields a submodular function $g_e : 2^{e_{\text{head}} \cup e_{\text{tail}}} \to \mathbb{R}^{\geq 0}$. In what follows, we describe our contributions to various problems in this area.

1.1 Improved Bounds for Directed Hypergraph Sparsification

In the graph case, it is known that directed graph cut-sparsifiers for graphs with n vertices can require as many as $\Omega(n^2)$ edges to preserve cuts to a $(1 \pm \varepsilon)$ factor. In this sense, directed graph cut-sparsification is a trivial task, as any graph has at most $O(n^2)$ edges to begin with. Contrary to this however, directed *hypergraph* sparsification is non-trivial. While the same $\Omega(n^2)$ lower bound exists, a directed hypergraph can have as many as 4^n directed hyperedges to start with, so a sparsifier with $O(n^2)$ directed hyperedges is a vast improvement. This observation has led to a rich line of research studying the feasibility of sparsifying directed hypergraphs. The first work on this front was the work of [20] which

showed the existence of directed hypergraph sparsifiers with $O(n^3/\varepsilon^2)$ directed hyperedges and gave a polynomial time algorithm for computing them. Later work by [7] presented a proof of sparsifiers with $\widetilde{O}(nr/\varepsilon^2)$ (where r is the maximum size of any hyperedge) hyperedges for undirected hypergraph spectral sparsification, and with $\widetilde{O}(n^2r^3/\varepsilon^2)$ directed hyperedges for directed hypergraph spectral sparsification by tuning their algorithm and performing a different analysis. In particular, this improved upon the result of [20] in the regime where r is constant. Note that as with graphs, spectral sparsification is a *stronger* notion than cut sparsification, so in particular, these proofs imply the existence of cut-sparsifiers of the same complexity.

Ultimately however, the complexity of directed spectral hypergraph sparsification was nearly settled by the work of Oko, Sakaue, and Tanigawa [18], who showed $(1 \pm \varepsilon)$ spectral-sparsifiers with $O(n^2 \log^3(n/\varepsilon)/\varepsilon^2)$ directed hyperedges exist for directed hypergraphs on n vertices.

Continuing this line of research, we show that fundamentally, the task of directed hypergraph sparsification can be reduced in a black-box manner to undirected hypergraph spectral sparsification.

More specifically, we show there is a lifting from a directed hypergraph on n vertices to an undirected hypergraph on $n^2 + 1$ vertices such that the Laplacian of every individual hyperedge is simultaneously preserved. That is, we show the following theorem:

▶ **Theorem 1.** *For $H = (V, E)$ an a directed hypergraph on n vertices, one can compute an undirected hypergraph $\psi(H)$ on $n^2 + 1$ vertices in time $O(mr^2)$ (where m is the number of hyperedges in H, and r is the maximum size of any hyperedge in H), such that for any $x \in \mathbb{R}^n$, one can also compute $\vartheta(x) \in \mathbb{R}^{n^2+1}$ in time $O(n^2)$ such that*

$$L_H(x) = L_{\psi(H)}(\vartheta(x)).$$

Moreover, for any hyperedge $e \in H$, there is a single corresponding hyperedge $\psi(e)$ in $\psi(H)$ such that

$$L_e(x) = L_{\psi(e)}(\vartheta(x)).$$

The size of $\psi(e)$ is at most $|e|^2$. Further, for $x \in \{0,1\}^n$, i.e. corresponding to a cut, $\vartheta(x)$ will be in $\{0,1\}^{n^2+1}$, i.e. also corresponding to a cut.

We can then use the existing state of the art literature of undirected spectral hypergraph sparsification [6, 14] to conclude the existence of directed spectral hypergraph sparsifiers with only $O(n^2 \log(n) \log(r)/\varepsilon^2)$ hyperedges which can be found in time $\widetilde{O}(mr^2)$, where m is the original number of hyperedges and r is the maximum size of any hyperedge. Note that this bound on the size of sparsifiers improves on the result of [18], and in particular, makes the dependence on ε exactly $O(1/\varepsilon^2)$, which now matches the literature for undirected sparsification. That is, we show the following:

▶ **Theorem 2.** *For any directed hypergraph $H = (V, E)$ on n vertices, and any $0 < \varepsilon < 1$ there exists a weighted sub-hypergraph $\hat{H}$ such that for all $x \in \mathbb{R}^n$:*

$$(1 - \varepsilon)L_H(x) \leq L_{\hat{H}}(x) \leq (1 + \varepsilon)L_H(x),$$

and $\hat{H}$ only has $O(n^2 \log(n) \log(r)/\varepsilon^2)$ hyperedges, where r is the maximum size of any hyperedge of H.

As an additional benefit, because the reduction of Theorem 1 preserves cut vectors, we can also invoke the result of [3] to conclude the existence of directed hypergraph *cut*-sparsifiers with $O(n^2 \log(n)/\varepsilon^2)$ hyperedges.

1.2 Lower Bounds for Sketching Cuts in Directed Hypergraphs

We next focus on the bit complexity of creating cut-sparsifiers for directed hypergraphs. This is done in hopes of answering an open question from [10] regarding the bit-complexity of arbitrary sketching schemes for submodular hypergraphs. In prior work [18, 10], a lower bound of size $\Omega(n^3)$ (ignoring ε) was established for the bit complexity of any directed hypergraph cut-*sparsifier*. However, lower bounds for sparsifiers explicitly take advantage of the sparsifier structure by starting with known examples of sparsifiers that require $\Omega(n^2)$ hyperedges, and then padding these hyperedges with random vertices in their tail such that the bit complexity of each hyperedge becomes $\Omega(n)$. One can trivially show that this padding does not change the requirement of preserving $\Omega(n^2)$ hyperedges. Because sparsifiers are limited to storing only hyperedges that were originally present, this then forces a bit complexity lower bound of $\Omega(n^3)$. However, this same technique is not amenable to a sketching lower bound as the padding procedure only adds complexity to each hyperedge, and not necessarily to the cut function as a whole. Thus, the difficulty is in showing that the cut function itself requires a large description size, regardless of how we choose to represent it. This marks a fundamental difference.

Addressing this, we show the following theorem:

▶ **Theorem 3.** *Any $(1 \pm \varepsilon)$ cut-sketching scheme for directed hypergraphs on n vertices must have worst-case space $\frac{n^3}{2^{O(\sqrt{\log(n)})}}$ bits (for $\varepsilon = \frac{1}{2^{O(\sqrt{\log(n)})}}$).*

At a high level, our proof takes advantage of a result of Kapralov et. al. [8]. In this work, the authors show that there exists a family of undirected hypergraphs on n vertices, each with at most n hyperedges, such that any sketching scheme which can sketch cuts in any of the hypergraphs in this family to an additive error of εn (for $\varepsilon = \frac{1}{2^{O(\sqrt{\log(n)})}}$) must have worst-case size at least $\frac{n^2}{2^{O(\sqrt{\log(n)})}}$. We show that by using a specific construction of a directed hypergraph, along with a specific reconstruction procedure, we can actually store an additive cut-approximation to n distinct undirected hypergraphs in a single cut-sketch of a directed hypergraph. That is, we show the following theorem:

▶ **Theorem 4.** *For any undirected hypergraphs $H_1, \dots H_n$, each on vertex set V, with $|V| = n$, there exists a directed hypergraph G on $2n$ vertices, such that given a $(1 \pm \varepsilon)$ cut-sketch for G, for any of the undirected hypergraphs $H_i = (V, E_i)$, one can recover $cut_{H_i}(S)$ to within additive error $3\varepsilon|E_i|$.*

Now, by sampling these undirected hypergraphs $H_1, \dots H_n$ from a specific family of hypergraphs, we can argue that simultaneously preserving the cut-values in all of these hypergraphs (even to an additive error) requires a data structure of size $\frac{n^2}{2^{O(\sqrt{\log(n)})}} \cdot n = \frac{n^3}{2^{O(\sqrt{\log(n)})}}$. In particular, by the previous reduction, any general scheme for sketching directed hypergraphs or submodular hypergraphs would be such a scheme, and therefore must have worst-case size at least $\Omega(n^{3-o(1)})$ (for $\varepsilon = \frac{1}{2^{O(\sqrt{\log(n)})}}$).

Prior to our work, there was no known super-quadratic (in n) lower bound on the sketching complexity of cuts in directed hypergraphs. In conjunction with our positive results on the sparsifiability of directed hypergraphs, this shows that directed hypergraph sparsification is almost-optimal even among all possible sketches for preserving cut values. That is, from the previous section, we know that directed hypergraph sparsifiers approximately preserve the sizes of all cuts in a directed hypergraph to a factor $(1 \pm \varepsilon)$ using $\widetilde{O}(n^3/\varepsilon^2)$ bits. In conjunction with our lower bound, we can conclude that this is almost the best possible (among *any* sketching scheme) in the regime where $\varepsilon = \frac{1}{2^{O(\sqrt{\log(n)})}}$. Thus, we show that for approximately storing cuts in directed hypergraphs using as few bits as possible, using a sparsifier is almost optimal. We view this as an important contribution of our work.

1.3 Cut Sparsifiers for Monotone Submodular Hypergraphs

Finally, we show that one can simulate cuts in monotone submodular hypergraphs with cuts in symmetric submodular hypergraphs. Recall that a set function is monotone if $f(S \cup \{t\}) \geq f(S)$, and we say that a submodular hypergraph is monotone if *every* splitting function is also monotone. This model of hypergraphs was specifically studied in the work of [10], where their sparsifiers ultimately achieved a complexity of $O(n^3/\varepsilon^2)$ hyperedges. In particular, monotone submodular functions capture a wide variety of natural and common functions such as matroid rank and entropy of random variables.

With respect to this, we show the following theorem:

▶ **Theorem 5.** *Suppose $f : 2^V \to \mathbb{R}^{\geq 0}$ is a monotone, submodular function. Then, $f' : 2^{V \cup \{*\}} \to \mathbb{R}^{\geq 0}$ defined as $\forall S \subseteq V$*

$$f'(S) = f(S) = f'(V - S \cup \{*\})$$

is submodular and symmetric.

Next, we show that given an arbitrary monotone, submodular hypergraph on n vertices, we can lift this to a symmetric submodular hypergraph on $n + 1$ vertices, where the single extra vertex is the $\{*\}$ vertex from the preceding theorem. Next, for each individual splitting function $g_e : 2^e \to \mathbb{R}^+$ in the monotone, submodular hypergraph, we replace g_e with g'_e, again using the preceding theorem.

Note that for each monotone submodular function, we *re-use* the same $\{*\}$ vertex. Thus, the increase in the size of the vertex set is only 1. Finally, we can then invoke a result from [5], which states that for any submodular hypergraph H where each splitting function is symmetric, one can calculate a sparsifier for H with only $\widetilde{O}(n/\varepsilon^2)$ hyperedges.

We then get the following:

▶ **Theorem 6.** *Let $H = (V, E)$ be a hypergraph, such that $\forall e \in E$, the corresponding splitting function $g_e : 2^e \to \mathbb{R}^{\geq 0}$ is submodular and monotone. Then there exists a $(1 \pm \varepsilon)$ cut-sparsifier for H retaining only $\widetilde{O}(n/\varepsilon^2)$ hyperedges.*

Prior to this work, the best known upper bound for the size complexity (in hyperedges) for $(1 \pm \varepsilon)$-sparsifying any monotone submodular hypergraph was $O(n^3/\varepsilon^2)$ hyperedges, proved in the work of [10]. Our result essentially improves this to the best possible, where we now only have a near-linear dependence on the size of the vertex set. We view it as an interesting open question if one can extend our proof method used here to general submodular functions (although this case will necessarily require a blow-up of at least quadratic size).

1.4 Overview

At a high level, all of our results use the same general principle, namely, by showing that cuts in one class of hypergraphs can be simulated by cuts in a simpler class of hypergraphs, we can leverage sparsification results for the simpler class of hypergraphs. This leads to our proofs being quite simple despite the fact that the results improve upon the state-of-the-art knowledge in hypergraph sparsification.

In Section 2 we introduce formal definitions and other preliminaries. In Section 3 we present the algorithms for sparsifying directed hypergraphs by reducing to undirected hypergraph sparsification. Next, in Section 4, we show how to simultaneously simulate cuts in many different undirected graphs thereby leading to new lower bounds for the worst case size of sketching cuts in directed hypergraphs. Finally, in Section 5, we show how to sparsify arbitrary monotone, submodular hypergraphs to near-optimal size.

2 Preliminaries

First, we introduce the definitions of undirected and directed hypergraphs.

▶ **Definition 7.** *An **undirected hypergraph** $G = (V, E)$ is a collection of vertices V, with associated hyperedges $e \in E$, where $e \subseteq V$ can be of arbitrary size.*

▶ **Definition 8.** *A **directed hypergraph** $H = (V, E)$ is a collection of vertices V along with directed hyperedges $e \in E$. Each directed hyperedge is of the form $e = (e_{tail}, e_{head})$, where $e_{head}, e_{tail} \subseteq V$. We will use $L(e) = e_{tail}, R(e) = e_{head}$. Note that e_{head}, e_{tail} are not necessarily disjoint.*

Next, we introduce the definition of spectral sparsifiers for both undirected and directed hypergraphs.

▶ **Definition 9.** *For an undirected hypergraph $G = (V, E, w)$ on n vertices, and a vector $x \in \mathbb{R}^n$, the **quadratic form of the Laplacian of G** is*

$$L_G(x) = \sum_{e \in E} \max_{u,v \in e} (x_u - x_v)^2.$$

▶ **Definition 10.** *For a directed hypergraph $G = (V, E, w)$ on n vertices, and a vector $x \in \mathbb{R}^n$, the **directed quadratic form of the Laplacian of G** is*

$$L_G(x) = \sum_{e \in E} \max_{u \in L(e), v \in R(e)} (x_u - x_v)_+^2.$$

In this context, $(x_u - x_v)_+ = \max((x_u - x_v), 0)$. A non-zero contribution from a hyperedge occurs only if a tail vertex of the hyperedge has a larger value than a head vertex of the hyperedge. Note that the head set and tail set of a directed hyperedge are not necessarily disjoint.

▶ **Definition 11.** *For a (directed or undirected) hypergraph $G = (V, E)$ on n vertices, a **$(1 \pm \varepsilon)$-spectral sparsifier** for G is a weighted (directed or undirected) sub-hypergraph H such that for every $x \in \mathbb{R}^n$,*

$$(1 - \varepsilon) L_G(x) \leq L_H(x) \leq (1 + \varepsilon) L_G(x).$$

Further, we require that the hyperedges of H are a subset of the hyperedges of G.

▶ Remark 12. For all the above definitions, if a reweighted sub-hypergraph H of G preserves the quadratic form for vectors $x \in \{0, 1\}^n$ to $(1 \pm \varepsilon)$ multiplicative error, we say that H is a **cut-sparsifiers**. Note that all spectral sparsifiers are cut-sparsifiers, while the converse is not necessarily true.

We also refer to cut-sizes in hypergraphs. A cut is specified by a set $S \subseteq V$, and we say the size of the cut S in G (denoted $|\text{cut}_G(S)|$) is $L_G(\mathbf{1}_S)^T$, where $\mathbf{1}_S$ is the indicator vector in $\{0, 1\}^n$ for the set S. Combinatorially, this refers to the weight of the hyperedges that are "leaving" the set S.

Next we define submodular functions and submodular hypergraphs.

▶ **Definition 13.** *A function $g : 2^V \to \mathbb{R}^{\geq 0}$ is said to be submodular if for any $S \subset T \subset V$, and any $x \in V - T$,*

$$g(S \cup \{x\}) - g(S) \geq g(T \cup \{x\}) - g(T).$$

Using this, we can define a submodular hypergraph.

▶ **Definition 14.** *A submodular hypergraph $H = (V, E)$ is a set of n vertices along with a set of hyperedges E. For each hyperedge $e \in E$, there is a corresponding submodular splitting function $g_e : 2^e \to \mathbb{R}^{\geq 0}$. For any subset $S \subseteq V$, the corresponding cut of the submodular hypergraph is*

$$cut_H(S) = \sum_{e \in E} g_e(S \cap e).$$

▶ **Definition 15.** *We say that a data structure G is a $(1 \pm \varepsilon)$-cut* sketch *of a submodular hypergraph $H = (V, E)$, if for any $S \subseteq V$ one can deterministically recover $cut_H(S)$ to within a $(1 \pm \varepsilon)$ factor using only the data structure G, and the set S.*

We will use the following result from [5] regarding the sparsifiability of symmetric, submodular hypergraphs. Note that a submodular function $f : 2^V \to \mathbb{R}^+$ is said to be symmetric if $\forall S \subseteq V, f(S) = f(V - S)$.

▶ **Theorem 16** (Corollary 1.2 of [5])**.** *For any symmetric submodular hypergraph H on n vertices, there is a $(1 \pm \varepsilon)$-sparsifier for H with $\widetilde{O}(n/\varepsilon^2)$ hyperedges.*

3 Directed to Undirected Hypergraph Sparsification

In this section, we will show that any algorithm that produces an undirected spectral hypergraph sparsifier with $f(n, r)$ hyperedges (for a vertex set of size n, and maximum hyperedge size r), can be used in a *black-box manner* to create a spectral sparsifier with $f(n^2 + 1, r^2)$ hyperedges for any n-vertex directed hypergraph.

To this end, we first have to define the "lifting" operation from a directed hypergraph on n vertices to an undirected hypergraph on $n^2 + 1$ vertices.

▶ **Definition 17.** *For a directed hypergraph $H = (V, E)$ on n vertices, let $\psi(H)$ be an undirected hypergraph on $n^2 + 1$ vertices defined as follows. For the first n^2 vertices of $\psi(H)$, associate these vertices with tuples of vertices from H, that is, each of these vertices is associated with an element from the set $V \times V$. The final vertex in $\psi(H)$ will be a special vertex we denote by $*$. Now, for each hyperedge $e \in E$ of H, define a corresponding hyperedge $\varphi(e)$ in $\psi(H)$ as follows: let the vertices in $L(e)$ be $u_1, \ldots u_\ell$, and let the vertices in $R(e)$ be $v_1, \ldots v_r$. Let $\varphi(e)$ contain*

$$L(e) \times R(e) \cup \{*\} = \{(u_1, v_1), (u_1, v_2), \ldots (u_1, v_r), (u_2, v_1), \ldots (u_2, v_r), \ldots (u_3, v_1), \ldots (u_\ell, v_r), *\}.$$

Note that this transformation is invertible. If we are given an undirected hyperedge of the form $\varphi(e) = L(e) \times R(e) \cup \{*\}$, we can invert this transformation to recover the directed hyperedge $e = (L(e), R(e))$. Additionally, note that this transformation and its inverse are efficiently computable (running in time $O(r^2)$, where r is the size of the undirected hyperedge).

Next, we define the lifting of a test vector.

▶ **Definition 18.** *For a vector $x \in \mathbb{R}^n$, we define the lifting of x denoted as $\vartheta(x)$. $\vartheta(x)$ is in $\mathbb{R}^{n^2+1}$, and in particular, for the first n^2 entries, we associate these with the set $[n] \times [n]$. We say that $(\vartheta(x))_{u,v} = \max(x_u - x_v, 0)$. For the final entry, which we associate with the special vertex $*$ in the lifted H, we let $\vartheta(x)_* = 0$.*

Note again that $\vartheta(x)$ is efficiently computable in time $O(n^2)$ where n is the dimension of x.

▶ **Theorem 19.** *Let $H = (V, E)$ be a directed hypergraph on n vertices. Then, for any $x \in \mathbb{R}^n$,*

$$L_H(x) = L_{\psi(H)}(\vartheta(x)).$$

Proof. It suffices to show that for a single hyperedge $e \in E$,

$$\max_{u \in L(e), v \in R(e)} (x_u - x_v)_+^2 = \max_{(y,z) \in \varphi(e)} (\vartheta(x)_y - \vartheta(x)_z)^2.$$

The reason this suffices is that there is one $\varphi(e)$ for each corresponding hyperedge $e \in E$. So, we are in effect showing that every term in the sum of the quadratic form of the Laplacians is the same.

To see why this equality is true, let some $\widehat{u} \in L(e), \widehat{v} \in R(e)$ be the maximizers for the expression on the left. Then, note that the corresponding entry $\vartheta(x)_{\widehat{u},\widehat{v}}$ is exactly $(x_{\widehat{u}} - x_{\widehat{v}})_+$. Now, because $\widehat{u} \in L(e)$ and $\widehat{v} \in R(e)$, it follows that $(\widehat{u}, \widehat{v}) \in \varphi(e)$. Because the special vertex $* \in \varphi(e)$, it follows that in the above expression

$$\max_{(y,z) \in \varphi(e)} (\vartheta(x)_y - \vartheta(x)_z)^2 \geq (\vartheta(x)_{(\widehat{u},\widehat{v})} - \vartheta(x)_*)^2 = (x_{\widehat{u}} - x_{\widehat{v}})_+^2 = \max_{u \in L(e), v \in R(e)} (x_u - x_v)_+^2.$$

Now, we will show the opposite direction. Indeed, suppose that some elements $\widehat{y}, \widehat{z}$ are maximizers for $\max_{(y,z) \in \varphi(e)} (\vartheta(x)_y - \vartheta(x)_z)^2$. Note that by construction, every entry in $\vartheta(x)$ is ≥ 0. This means that without loss of generality, we can assume that $\widehat{z} = *$ (the special vertex), as this vertex attains the smallest possible value 0. This means that the maximizing value of the expression is exactly $\vartheta(x)_{\widehat{y}}^2$, where $\widehat{y}$ is one of the first n^2 vertices in $\psi(H)$. So, let us write $\widehat{y} = (\widehat{a}, \widehat{b})$, where $\widehat{a}, \widehat{b}$ are both vertices in G. By construction, because $\widehat{y} \in \widehat{e}$, it follows that $\widehat{a} \in L(e)$, and $\widehat{b} \in R(e)$. As such it follows that

$$\max_{u \in L(e), v \in R(e)} (x_u - x_v)_+^2 \geq (x_{\widehat{a}} - x_{\widehat{b}})_+^2 = \vartheta(x)_{\widehat{y}}^2 = \max_{(y,z) \in \widehat{e}} (\vartheta(x)_y - \vartheta(x)_z)^2.$$

Thus, it follows that

$$\max_{u \in L(e), v \in R(e)} (x_u - x_v)_+^2 = \max_{(y,z) \in \varphi(e)} (\vartheta(x)_y - \vartheta(x)_z)^2,$$

as claimed. ◀

▶ **Corollary 20.** *Let H be a directed hypergraph on n vertices. Suppose that $\widehat{\psi(H)}$ is a $(1 \pm \varepsilon)$ undirected hypergraph spectral sparsifier to $\psi(H)$. Then, it follows that the unlifted graph $\widehat{H}$ which is calculated by applying φ^{-1} to each hyperedge in $\widehat{\psi(H)}$, is a $(1 \pm \varepsilon)$ directed hypergraph spectral sparsifier to H.*

Proof. Indeed, suppose $H, \psi(H), \widehat{H}, \widehat{\psi(H)}$ are as specified above, and let $x \in \mathbb{R}^n$. It follows that

$$(1 - \varepsilon) L_H(x) = (1 - \varepsilon) L_{\psi(H)}(\vartheta(x)) \leq L_{\widehat{\psi(H)}}(\vartheta(x)) = L_{\widehat{H}}(x)$$

$$= L_{\widehat{\psi(H)}}(\vartheta(x)) \leq (1 + \varepsilon) L_{\psi(H)}(\vartheta(x)) = (1 + \varepsilon) L_H(x).$$

To conclude, this implies that for $\widehat{H}, H$ as above,

$$(1 - \varepsilon) L_H(x) \leq L_{\widehat{H}}(x) \leq (1 + \varepsilon) L_H(x).$$ ◀

▶ **Theorem 21.** *For a directed hypergraph H on n vertices, one can find a directed hypergraph spectral sparsifier $\widehat{H}$ of H, with $O(n^2 \log(n) \log(r)/\varepsilon^2)$ hyperedges in time $\widetilde{O}(mr^2)$, where m is the number of directed hyperedges in H and r is the maximum size of a hyperedge in H.*

Proof. If the number of hyperedges in H is less than n^2, simply return H. Otherwise, lift H to $\psi(H)$, and spectrally sparsify $\psi(H)$ using [6]. This will result in a $(1 \pm \varepsilon)$ spectral sparsifier $\widehat{\psi(H)}$ to $\psi(H)$, with at most $O(n^2 \log(n^2) \log(r^2)/\varepsilon^2)$ hyperedges, as we desire. Here, we have used that the maximum rank of a hyperedge in $\widehat{G}$ is at most the squared rank of a hyperedge in G. Further, the running time of this algorithm is $\widetilde{O}(mr^2)$, as the number of hyperedges in $\widehat{G}$ is the same as the number of hyperedges in G, and the rank, again, is at most r^2. Now, we can unlift $\widehat{\psi(H)}$ to $\widehat{H}$ by applying φ^{-1} to each hyperedge, and use the previous corollary to conclude our theorem. ◀

▶ Remark 22. Note that if we restrict our original vector x to be in $\{0,1\}^n$, it follows that $\vartheta(x) \in \{0,1\}^{n^2+1}$. By repeating the exact same steps above, this means that we can use the same reduction from above to get directed hypergraph cut sparsifiers, by only using algorithms from undirected hypergraph cut sparsifiers.

▶ **Corollary 23.** *For a directed hypergraph H on n vertices, one can find a directed hypergraph cut sparsifier $\widehat{H}$ of H, with $O(n^2 \log(n)/\varepsilon^2)$ hyperedges in time $\widetilde{O}(mr^2/\varepsilon^2)$, where m is the number of directed hyperedges in H, and r the maximum size of any hyperedge.*

Proof. Simply perform the reduction from above, and invoke the algorithm for undirected hypergraph cut-sparsification from Theorem 1.3 of [19]. Specifically, for an undirected hypergraph G, Quanrud presents an algorithm running in time $\widetilde{O}(p(G))$ which constructs $(1 \pm \varepsilon)$ k-cut sparsifiers (a stronger requirement than the cut-sparsifiers required here), where $p(G) = \sum_{e \in G} |e|$. In our setting, for a directed hypergraph H, when we construct the undirected lifting $\psi(H)$, it will be the case that $p(\psi(H)) = \sum_{e \in \psi(H)} |e| \leq \sum_{e \in H} |e|^2 \leq mr^2$, yielding the desired runtime. ◀

4 Space Lower-bounds for Sketching Cuts in Directed Hypergraphs

In this section, we will establish an $\Omega(n^{3-o(1)})$ lower-bound for worst-case sketching of the cuts in a directed hypergraph on n vertices to a $(1 \pm \varepsilon)$ factor for ε being $\frac{1}{2^{O(\sqrt{\log(n)})}}$. As mentioned in the introduction, this improves upon a result of [10] who showed a lower bound of size $\Omega(n^3)$ for the bit complexity of any *sparsifier*. However, their lower bound explicitly takes advantage of the sparsifier structure by starting with known examples of sparsifiers that require $\Omega(n^2)$ hyperedges, and then padding these hyperedges with random vertices in their tail such that the bit complexity is $\Omega(n)$. One can trivially show that this padding does not change the requirement of preserving $\Omega(n^2)$ hyperedges. Because sparsifiers are limited to storing only hyperedges that were originally present, this then forces a bit complexity lower bound of $\Omega(n^3)$. However, this same technique is not amenable to a sketching lower bound as the padding procedure only adds complexity to each hyperedge, and not necessarily to the cut function as a whole.

To overcome this, we take advantage of a result of [8] who showed that, in general, any $(1 \pm \varepsilon)$ cut-sketching scheme for undirected hypergraphs on n vertices, with $\varepsilon = \frac{1}{2^{O(\sqrt{\log(n)})}}$ must have worst case bit complexity $\frac{n^2}{2^{O(\sqrt{\log(n)})}}$. This result uses encodings of Rusza-Szemerédi graphs into undirected hypergraphs, along with a reconstruction argument to show that general $(1 \pm \varepsilon)$ cut-sketching schemes in undirected hypergraphs give very non-trivial string

compression schemes. Then, by invoking known results on size lower bounds for string compression schemes, they are able to conclude worst-case lower bounds of $\frac{n^2}{2^{O(\sqrt{\log(n)})}}$ for the bit complexity of sketching cuts in undirected hypergraphs. To this end, we first reintroduce their notion of a string compression scheme:

▶ **Definition 24** ([4]). *Let ℓ, k be positive integers, and let $\varepsilon, g > 0$. We say that a pair of functions* $\text{Encode} : \{0,1\}^\ell \to \{0,1\}^k$ *and* $\text{Decode} : \{0,1\}^k \times 2^{[\ell]} \to \mathbb{N}$ *is an $(\ell, k, \varepsilon, g)$ string compression scheme (SCS) if there exists a set of strings $\mathcal{G} \subseteq \{0,1\}^\ell$ such that:*

1. $|\mathcal{G}| \geq g \cdot 2^\ell$.
2. *For every string $s \in \mathcal{G}$, and every query $q \in 2^{[\ell]}$,*

$$|\text{Decode}(\text{Encode}(s), q) - |s \cap q|| \leq \varepsilon\ell/2.$$

The work of [8] takes advantage of the following theorem, which is proved in [4]:

▶ **Theorem 25** ([4]). *Suppose* $(\text{Encode}, \text{Decode})$ *is an $(\ell, k, \varepsilon, g)$-SCS, where $\varepsilon \leq 1/10$. Then,*

$$k \geq \frac{\log(g) + 3\ell/50}{\log 2} - 1.$$

Qualitatively, [8] shows that for a specific family of *undirected* hypergraphs with n vertices, for some $\varepsilon = \frac{1}{2^{O(\sqrt{\log(n)})}}$ any $(1 \pm \varepsilon)$ cut-sketching scheme for these hypergraphs using $\leq k$ bits implicitly gives an $\left(\frac{n^2}{2^{O(\sqrt{\log(n)})}}, k, 1/10, 1/2\right)$-SCS. Thus, by invoking the previous theorem, these sparsifiers must have bit complexity $\frac{n^2}{2^{O(\sqrt{\log(n)})}}$. However, their proof actually provides a stronger result than stated. Although the sparsifiers they use give $(1 \pm \varepsilon)$ multiplicative approximations to cut-sizes, their argument makes uses of an *additive* error bound of $\varepsilon \cdot (\#$ of hyperedges). We take advantage of this in our method by showing that a $(1 \pm \varepsilon)$ cut-sketch for a directed hypergraph can be used to retrieve cut sizes in n distinct undirected hypergraphs with only additive error ε (with respect to each of these undirected hypergraphs). We first state the result of [8] more succinctly, and then describe our construction in more detail.

▶ **Theorem 26** ([8]). *For any n, and some $\ell = \frac{n^2}{2^{O(\sqrt{\log(n)})}}$, for at least $2^\ell/2$ strings $s \in \{0,1\}^\ell$, there exists an undirected hypergraph $H_s = (V, E_s)$ on n vertices, with $\leq n$ hyperedges, such that any data structure which can approximate cuts in H_s to within additive error $|E_s|/2^{O(\sqrt{\log(n)})}$ can for any query $q \subseteq [\ell]$, answer the subset sum $|q \cap s|$ to within additive error $\ell/20$.*

Now, we will prove our theorem regarding capability of directed hypergraphs to simulate cuts in undirected hypergraphs with only additive error.

▶ **Theorem 27.** *Given any undirected hypergraphs $H_1, \ldots H_n$, each on vertex set V, with $|V| = n$, there exists a directed hypergraph G on $2n$ vertices, such that given a $(1 \pm \varepsilon)$ cut-sketch for G, for any of the undirected hypergraphs $H_i = (V, E_i)$ and any set $S \subseteq V$, one can recover $|cut_{H_i}(S)|$ to within additive error $3\varepsilon|E_i|$.*

Proof. As stated, each of the undirected hypergraphs $H_1, \ldots H_n$ are on a vertex set of size n, which we denote by V. We also create a vertex set W of size n, which we associate with $w_1, \ldots w_n$. Now, we create the directed hypergraph G, which lives on the vertex set $V \cup W$ as follows: for each undirected hypergraph H_i for $i = 1, \ldots n$, and for each undirected hyperedge e in H_i, we add the corresponding directed hyperedge (e, w_i). That is, the head of the directed hyperedge has the vertices from V corresponding to e, and the tail of the directed hyperedge has only vertex w_i.

Clearly, G has $2n$ vertices, so now it suffices to argue that for any $H_i = (V, E_i)$, and for any cut $S \subseteq V$, we can recover $\text{cut}_{H_i}(S)$ within additive error $\varepsilon|E_i|$. Indeed, let any such H_i be given, and let $S \subseteq V$ be given as well. Then, suppose we have a $(1 \pm \varepsilon)$ cut-sketch for G, which we denote by $\widetilde{G}$. Let us consider the query to $\widetilde{G}$ with the set $S \cup W - \{w_i\}$. A directed hyperedge $e \in G$ is crossing this cut if and only if $e_{\text{head}} \cap (S \cup W - \{w_i\}) \neq \emptyset$ and $e_{\text{tail}} \cap ((V \cup W) - (S \cup W - \{w_i\})) \neq \emptyset$. In particular, note that by construction, e_{head} is a subset of V and e_{tail} is a subset of W. This means that a directed hyperedge e is crossing the cut if and only if $e_{\text{head}} \cap S \neq \emptyset$ and $e_{\text{tail}} \cap \{w_i\} \neq \emptyset$. The only directed hyperedges in G which satisfy this second condition are exactly those directed hyperedges in G which correspond to H_i. By construction, this means that the number of directed hyperedges crossing this cut $S \cup W - \{w_i\}$ in G is exactly the number of undirected hyperedges $e \in E_i$ such that $e \cap S \neq \emptyset$. Thus this query to $\widetilde{G}$ returns a $(1 \pm \varepsilon)$ approximation to $|\{e \in E_i | S \cap e \neq \emptyset\}|$. Note that the actual size of the cut S in H_i is $|\{e \in E_i | S \cap e \neq \emptyset \wedge S \cap e \neq e\}|$.

However, note that by symmetry, we can also query $\widetilde{G}$ with $(V - S) \cup (W - \{w_i\})$. By symmetry, this query to $\widetilde{G}$ returns a $(1 \pm \varepsilon)$ approximation to $|\{e \in E_i | (V - S) \cap e \neq \emptyset\}|$, which is the same as $|\{e \in E_i | S \cap e \neq e\}|$. Lastly, we can query $\widetilde{G}$ with $V \cup (W - \{w_i\})$. This query to $\widetilde{G}$ returns a $(1 \pm \varepsilon)$ approximation to $|\{e \in E_i | V \cap e \neq \emptyset\}|$, which is exactly $|\{e \in E_i\}|$.

Now, we operate by the principle of inclusion-exclusion (PIE). Let A be the event that a hyperedge $e \in E_i$ satisfies $e \cap S \neq \emptyset$, and let B be the event that e satisfies $e \cap S \neq e$. By PIE,

$$\begin{aligned}|\{e \in E_i | e \text{ satisfies } A \wedge \text{ satisfies } B\}| =& |\{e \in E_i | e \text{ satisfies } A\}| + |\{e \in E_i | e \text{ satisfies } B\}| \\ & - |\{e \in E_i | e \text{ satisfies } A \vee \text{ satisfies } B\}|.\end{aligned}$$

Note that this final expression is trivially satisfied, i.e. $|\{e \in E_i | e \text{ satisfies } A \vee \text{ satisfies } B\}| = |\{e \in E_i\}|$ as a hyperedge cannot simultaneously have an empty and a non-trivial intersection. Thus, we get that

$$\begin{aligned}\text{cut}_{H_i}(S) =& |\{e \in E_i | e \text{ satisfies } A \wedge \text{ satisfies } B\}| \\ =& |\{e \in E_i | e \text{ satisfies } A\}| + |\{e \in E_i | e \text{ satisfies } B\}| - |\{e \in E_i\}|.\end{aligned}$$

Now, note that our query to $\widetilde{G}$ with the set $S \cup W - \{w_i\}$ gave us a $(1 \pm \varepsilon)$ approximation to $|\{e \in E_i | e \text{ satisfies } A\}|$, our query with $(V - S) \cup W - \{w_i\}$ gave us a $(1 \pm \varepsilon)$ approximation to $|\{e \in E_i | e \text{ satisfies } B\}|$, and our query with $V \cup W - \{w_i\}$ gave us a $(1 \pm \varepsilon)$ approximation to $|\{e \in E_i\}|$. Because each of these has additive error at most $\varepsilon|E_i|$ (as the error from $\widetilde{G}$ is a multiplicative guarantee), in total, the expression

$$\text{cut}_{\widetilde{G}}(S \cup W - \{w_i\}) + \text{cut}_{\widetilde{G}}((V - S) \cup W - \{w_i\}) - \text{cut}_{\widetilde{G}}(V \cup W - \{w_i\})$$

gives us a $(3\varepsilon|E_i|)$-additive approximation to $\text{cut}_{H_i}(S)$, as we desire. ◀

Now, we will show how we can use the above construction to argue a lower bound of size $\frac{n^3}{2^{O(\sqrt{\log(n)})}}$ on the bit complexity of directed hypergraph cut-sketching. We will do this by showing that we can use a directed hypergraph cut-sketch of size k to create a $(\ell, k, 1/10, 2^{-n})$-SCS, for $\ell = \Omega\left(\frac{n^3}{2^{O(\sqrt{\log(n)})}}\right)$.

▶ **Theorem 28.** *A general unweighted directed hypergraph $(1 \pm \frac{1}{2^{O(\sqrt{\log(n)})}})$ cut-sketching scheme on n vertices with maximum sketch size of k bits yields an $(n \cdot \ell, k, 1/10, 2^{-n})$-SCS for $\ell = \frac{n^2}{2^{O(\sqrt{\log(n)})}}$.*

Proof. First, we will define the set $\mathcal{G}$ of size $\frac{2^{n\cdot\ell}}{2^n}$. Indeed, from Theorem 26, let L be the strings of length ℓ which are able to be compressed and still allow for estimating subset sum queries. Now, let $\mathcal{G} = L \circ L \circ L \circ \cdots \circ L$ (n times), where the $S_1 \circ S_2$ operation takes every string in S_1 and prepends it to every string in S_2 (resulting in a new set of size $|S_1| \cdot |S_2|$). Note that this means that strings in $\mathcal{G}$ will be of length $n \cdot \frac{n^2}{2^{O(\sqrt{\log(n)})}} = \frac{n^3}{2^{O(\sqrt{\log(n)})}}$ and $\mathcal{G}$ obtains the stated size bound.

Now we describe our string compression scheme. Indeed, for any string $s \in \mathcal{G}$, decompose s into $s_1, \ldots s_n$ such that each $s_i \in L$. Now, because each $s_i \in L$, we know there exists a corresponding undirected hypergraph $H_{s_i} = (V, E_{s_i})$ on n vertices such that preserving cuts in H_{s_i} to within additive error $|E_{s_i}|/2^{O(\sqrt{\log(n)})}$ allows us to answer subset sum queries in H_{s_i} to within additive error $\ell/20$. Now let G be the directed hypergraph on $2n$ vertices, built with hypergraphs $H_{s_1}, H_{s_2}, \ldots H_{s_n}$ as guaranteed by Theorem 27. It follows that G is an unweighted directed hypergraph on $2n$ vertices.

Now, suppose there exists a general, unweighted, directed hypergraph cut-sketching scheme on n vertices with maximum sketch size of k bits which preserves cuts to a $(1 \pm \frac{1}{2^{O(\sqrt{\log(n)})}})$ multiplicative factor. Then, we can invoke such a scheme on the directed hypergraph G as specified by Theorem 27 to conclude that such a scheme allows us to recover $\text{cut}_{H_{s_i}}(S)$ for any $S \subseteq V$ to within additive error $|E_{s_i}/2^{O(\sqrt{\log(n)})}|$. As a result, this means that for any s_i, and any query to s_i, denoted by $q_i \in [\ell]$, we can recover $|q_i \cap s_i|$ to within additive error $\ell/20$.

Finally, suppose we are given any subset query $q \subseteq [n \cdot \ell]$. We want to show that we can compute the size of $|s \cap q|$ (i.e. the sum of the bits of s on the positions indicated by q) to within additive error $\frac{n\ell}{20}$. For convenience, we view q as a bit string of length $n \cdot \ell$, where a bit is 1 if and only if the corresponding element of $[n \cdot \ell]$ was in the subset. Then, we break q into $q_1, \ldots q_n$ such that each q_i is of length ℓ. Now, we use the aforementioned sketch to compute $|s_i \cap q_i|$ to within additive error $\ell/20$ for every i. Adding these together, we get an estimate to $|s \cap q|$ with additive error at most $n\ell/20$. Thus, a general directed hypergraph cut-sketching scheme of size k bits to multiplicative error $(1 \pm \frac{1}{2^{O(\sqrt{\log(n)})}})$ yields a $(n \cdot \ell, k, 1/10, 2^{-n})$-SCS. ◀

▶ **Theorem 29.** *Any cut-sketching scheme for directed hypergraphs on $2n$ vertices which preserves cuts to a $(1 \pm \varepsilon)$ factor, for $\varepsilon = \frac{1}{2^{O(\sqrt{\log(n)})}}$ must have worst case bit complexity $\frac{n^3}{2^{O(\sqrt{\log(n)})}}$.*

Proof. Indeed, by the preceding theorem (Theorem 28), any such scheme for $\varepsilon = \frac{1}{2^{O(\sqrt{\log(n)})}}$, with bit complexity k implies a $(n \cdot \ell, k, 1/10, 2^{-n})$-SCS, for $\ell = \frac{n^2}{2^{O(\sqrt{\log(n)})}}$. By Theorem 25 [4], this means that

$$k \geq \frac{\log(2^{-n}) + 3n \cdot \ell}{\log 2} - 1 \geq \frac{n^3}{2^{O(\sqrt{\log(n)})}}.$$ ◀

▶ **Corollary 30.** *Any cut-sketching scheme for submodular hypergraphs on $2n$ vertices which preserves cuts to a $(1 \pm \varepsilon)$ factor, for $\varepsilon = \frac{1}{2^{O(\sqrt{\log(n)})}}$ must have bit complexity $\frac{n^3}{2^{O(\sqrt{\log(n)})}}$.*

Proof. This follows simply by noting that directed hypergraphs are a subclass of submodular hypergraphs, so in particular the lower bound from Theorem 29 must extend to this case. ◀

5 Monotone Hypergraph Sparsifiers

In this section, we show how to reduce sparsifying monotone submodular hypergraphs to sparsifying symmetric submodular hypergraphs. At this point, we then invoke the result of [5] to conclude. First, we detail the reduction:

▶ **Theorem 31.** *Suppose* $f : 2^V \to \mathbb{R}^{\geq 0}$ *is a monotone, submodular function. Then,* $f' : 2^{V \cup \{*\}} \to \mathbb{R}^{\geq 0}$ *defined as* $\forall S \subseteq V$

$$f'(S) = f(S) = f'(V - S \cup \{*\})$$

is submodular and symmetric.

Proof. First, the symmetry of f' is easy to see. Indeed, for any set $S \subseteq V \cup \{*\}$, it follows that $f'(S) = f'(V \cup \{*\} - S)$. So, all that remains to be shown is that f' is submodular. To do this, we will show that f' has decreasing marginals. So consider any $T \subset U \subset V \cup \{*\}$. We will show that for any $x \notin U$ that

$$f'(T \cup \{x\}) - f'(T) \geq f'(U \cup \{x\}) - f'(U).$$

We will do this by cases.

1. Suppose that $x = *$. Then, it must be the case that $x \notin U, T$. So, $f'(U) = f(U), f'(T) = f(T)$. Because $T \subset U$, it must also therefore be the case that $f'(T) \leq f'(U)$ (by the monotonicity of f). Next, we note that because $x = *$, $f'(T \cup \{x\}) = f(V - T), f'(U \cup \{x\}) = f(V - U)$. Because $T \subset U$ and f is monotone, it must be the case that $f'(T \cup \{x\}) = f(V - T) \geq f'(U \cup \{x\}) = f(V - U)$. Putting this together, we get that it must be the case that

   $$f'(T \cup \{x\}) - f'(T) \geq f'(U \cup \{x\}) - f'(U),$$

   as we desire.
2. Suppose that $x \neq *$, and that neither U, T contain $*$. Then the submodularity of f' follows by the submodularity of f.
3. Suppose that $x \neq *$, and that both U, T contain $*$. Then, let $\hat{U}, \hat{T}$ be $U - \{*\}, T - \{*\}$ respectively. It follows that $\hat{T} \subset \hat{U}$. Further, $f'(T \cup \{x\}) = f(V - (\hat{T} \cup \{x\})), f'(U \cup \{x\}) = f(V - (\hat{U} \cup \{x\}))$, and likewise $f'(T) = f(V - \hat{T}), f'(U) = f(V - \hat{U})$. It follows that

   $$\begin{aligned} f'(T \cup \{x\}) - f'(T) &= f(V - (\hat{T} \cup \{x\})) - f(V - \hat{T}) \\ &= f(V - (\hat{T} \cup \{x\})) - f(V - (\hat{T} \cup \{x\}) \cup \{x\}) \\ &\geq f(V - (\hat{U} \cup \{x\})) - f(V - (\hat{U} \cup \{x\}) \cup \{x\}) \\ &= f'(U \cup \{x\}) - f'(U). \end{aligned}$$

   The inequality in the middle holds because $V - (\hat{U} \cup \{x\}) \subset V - (\hat{T} \cup \{x\})$. Thus, the marginal gain from adding x to $V - (\hat{U} \cup \{x\})$ is larger than the marginal gain from adding x to $V - (\hat{T} \cup \{x\})$ by the submodularity of f.
4. Suppose that $x \neq *$, but that $* \notin T, * \in U$. Then, by the monotonicity of f, $f'(T \cup \{x\}) - f'(T) = f(T \cup \{x\}) - f(T) \geq 0$. Likewise,

   $$f'(U \cup \{x\}) - f'(U) = f(V - (\hat{U} \cup \{x\})) - f(V - \hat{U}) \leq 0,$$

   again using the monotonicity of f. Therefore, it must be the case that

   $$f'(T \cup \{x\}) - f'(T) \geq f'(U \cup \{x\}) - f'(U),$$

   as we desire. ◀

Next, we show how to use this reduction to create sparsifiers.

▶ **Corollary 32.** *Let $H = (V, E)$ be a hypergraph, such that $\forall e \in E$, the corresponding splitting function $g_e : 2^e \to \mathbb{R}^{\geq 0}$ is submodular and monotone. Then there exists a $(1 \pm \varepsilon)$ cut-sparsifier for H with $\widetilde{O}(|V|/\varepsilon^2)$ hyperedges.*

Proof. We first define the lifting of a monotone, submodular hypergraph into a symmetric submodular hypergraph.

▶ **Definition 33.** *Let $H = (V, E)$ be a monotone submodular hypergraph. Then, define H' to be the corresponding hypergraph defined on vertex set $V \cup \{*\}$, where for each edge $e \in E$, we replace it with a hyperedge $e' = e \cup \{*\}$, and replace the function g_e with the symmetric, submodular splitting function $g'_e : 2^{e'} \to \mathbb{R}^{\geq 0}$ defined in accordance with Theorem 31.*

Now, we construct this hypergraph H'. Because each g'_e is symmetric and submodular, we can invoke Theorem 16 to conclude the existence of a hypergraph $\hat{H}'$ such that $\forall S \subseteq V \cup \{*\}$

$$(1-\varepsilon)\text{cut}_{H'}(S) \leq \text{cut}_{\hat{H}'}(S) \leq (1+\varepsilon)\text{cut}_{H'}(S),$$

and $\hat{H}'$ only has $\widetilde{O}(|V|/\varepsilon^2)$ hyperedges remaining.

It follows that because $\forall S \subseteq V$, $g'_e(S) = g_e(S)$, the corresponding hyperedges chosen to create a $(1 \pm \varepsilon)$ cut-sparsifier for H' also create a $(1 \pm \varepsilon)$ cut-sparsifier for H. That is, if we create the hypergraph $\hat{H}$ by replacing $e' \in \hat{H}'$ with $e \in H$ (but keeping the same corresponding weights that $\hat{H}'$ assigns), it will be the case that $\forall S \subseteq V$

$$(1-\varepsilon)\text{cut}_{H'}(S) = (1-\varepsilon)\text{cut}_{H}(S) \leq \text{cut}_{\hat{H}'}(S) = \text{cut}_{\hat{H}}(S) \leq (1+\varepsilon)\text{cut}_{H'}(S) = (1+\varepsilon)\text{cut}_{H}(S).$$

Thus, $\hat{H}$ will be a $(1\pm\varepsilon)$-sparsifier for H, and $\hat{H}$ will only keep $\widetilde{O}(|V|/\varepsilon^2)$ hyperedges. ◀

References

1. Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 255–262. ACM, 2009. doi:10.1145/1536414.1536451.
2. András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 47–55. ACM, 1996. doi:10.1145/237814.237827.
3. Yu Chen, Sanjeev Khanna, and Ansh Nagda. Near-linear size hypergraph cut sparsifiers. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 61–72. IEEE, 2020. doi:10.1109/FOCS46700.2020.00015.
4. Irit Dinur and Kobbi Nissim. Revealing information while preserving privacy. In Frank Neven, Catriel Beeri, and Tova Milo, editors, *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, pages 202–210. ACM, 2003. doi:10.1145/773153.773173.
5. Arun Jambulapati, James R. Lee, Yang P. Liu, and Aaron Sidford. Sparsifying sums of norms. *CoRR*, abs/2305.09049, 2023. doi:10.48550/arXiv.2305.09049.
6. Arun Jambulapati, Yang P. Liu, and Aaron Sidford. Chaining, group leverage score overestimates, and fast spectral hypergraph sparsification. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 196–206. ACM, 2023. doi:10.1145/3564246.3585136.

7 Michael Kapralov, Robert Krauthgamer, Jakab Tardos, and Yuichi Yoshida. Spectral hypergraph sparsifiers of nearly linear size. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1159–1170. IEEE, 2021. doi:10.1109/FOCS52979.2021.00114.

8 Michael Kapralov, Robert Krauthgamer, Jakab Tardos, and Yuichi Yoshida. Towards tight bounds for spectral sparsification of hypergraphs. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 598–611. ACM, 2021. doi:10.1145/3406325.3451061.

9 David R. Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 21–30. ACM/SIAM, 1993. URL: http://dl.acm.org/citation.cfm?id=313559.313605.

10 Yotam Kenneth and Robert Krauthgamer. Cut sparsification and succinct representation of submodular hypergraphs. *CoRR*, abs/2307.09110, 2023. arXiv:2307.09110.

11 Sanjeev Khanna, Aaron Putterman, and Madhu Sudan. Code sparsification and its applications. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 5145–5168. SIAM, 2024.

12 Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In Tim Roughgarden, editor, *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 367–376. ACM, 2015. doi:10.1145/2688073.2688093.

13 Jannik Kudla and Stanislav Zivný. Sparsification of monotone k-submodular functions of low curvature. *CoRR*, abs/2302.03143, 2023. doi:10.48550/arXiv.2302.03143.

14 James R. Lee. Spectral hypergraph sparsification via chaining. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 207–218. ACM, 2023. doi:10.1145/3564246.3585165.

15 Pan Li and Olgica Milenkovic. Inhomogeneous hypergraph clustering with applications. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 2308–2318, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/a50abba8132a77191791390c3eb19fe7-Abstract.html.

16 Pan Li and Olgica Milenkovic. Submodular hypergraphs: p-laplacians, cheeger inequalities and spectral clustering. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 3020–3029. PMLR, 2018. URL: http://proceedings.mlr.press/v80/li18e.html.

17 Hui Lin and Jeff A. Bilmes. A class of submodular functions for document summarization. In Dekang Lin, Yuji Matsumoto, and Rada Mihalcea, editors, *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 510–520. The Association for Computer Linguistics, 2011. URL: https://aclanthology.org/P11-1052/.

18 Kazusato Oko, Shinsaku Sakaue, and Shin-ichi Tanigawa. Nearly tight spectral sparsification of directed hypergraphs. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 94:1–94:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.94.

19 Kent Quanrud. *Quotient sparsification for submodular functions*, pages 5209–5248. SIAM, 2024. doi:10.1137/1.9781611977912.187.

20 Tasuku Soma and Yuichi Yoshida. Spectral sparsification of hypergraphs. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2570–2581. SIAM, 2019. doi:10.1137/1.9781611975482.159.

21 Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM J. Comput.*, 40(4):981–1025, 2011. doi:10.1137/08074489X.

22 Nate Veldt, Austin R. Benson, and Jon M. Kleinberg. Minimizing localized ratio cut objectives in hypergraphs. In Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash, editors, *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 1708–1718. ACM, 2020. doi:10.1145/3394486.3403222.

23 Nate Veldt, Austin R. Benson, and Jon M. Kleinberg. Approximate decomposable submodular function minimization for cardinality-based components. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 3744–3756, 2021. URL: https://proceedings.neurips.cc/paper/2021/hash/1e8a19426224ca89e83cef47f1e7f53b-Abstract.html.

Constrained Level Planarity Is FPT with Respect to the Vertex Cover Number

Boris Klemz
Universität Würzburg, Germany

Marie Diana Sieper
Universität Würzburg, Germany

Abstract

The problem Level Planarity asks for a crossing-free drawing of a graph in the plane such that vertices are placed at prescribed y-coordinates (called *levels*) and such that every edge is realized as a y-monotone curve. In the variant Constrained Level Planarity, each level y is equipped with a partial order $\prec_y$ on its vertices and in the desired drawing the left-to-right order of vertices on level y has to be a linear extension of $\prec_y$. Constrained Level Planarity is known to be a remarkably difficult problem: previous results by Klemz and Rote [ACM Trans. Alg.'19] and by Brückner and Rutter [SODA'17] imply that it remains NP-hard even when restricted to graphs whose tree-depth and feedback vertex set number are bounded by a constant and even when the instances are additionally required to be either *proper*, meaning that each edge spans two consecutive levels, or *ordered*, meaning that all given partial orders are total orders. In particular, these results rule out the existence of FPT-time (even XP-time) algorithms with respect to these and related graph parameters (unless P=NP). However, the parameterized complexity of Constrained Level Planarity with respect to the vertex cover number of the input graph remained open.

In this paper, we show that Constrained Level Planarity can be solved in FPT-time when parameterized by the vertex cover number. In view of the previous intractability statements, our result is best-possible in several regards: a speed-up to polynomial time or a generalization to the aforementioned smaller graph parameters is not possible, even if restricting to proper or ordered instances.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Human-centered computing → Graph drawings; Mathematics of computing → Graphs and surfaces; Theory of computation → Computational geometry; Mathematics of computing → Paths and connectivity problems

Keywords and phrases Parameterized Complexity, Graph Drawing, Planar Poset Diagram, Level Planarity, Constrained Level Planarity, Vertex Cover, FPT, Computational Geometry

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.99

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2404.16723` [28]

1 Introduction

A large body of literature related to graph drawing is dedicated to so-called upward planar drawings, which provide a natural way of visualizing a partial order on a set of items. An *upward planar drawing* of a directed graph is a crossing-free drawing in the plane where every edge $e = (u, v)$ is realized as a y-monotone curve that goes upwards from u to v, i.e., the y-coordinate strictly increases when traversing e from u towards v. The most classical computational problem in this context is Upward Planarity: given a directed graph, decide whether it admits an upward planar drawing. The standard version of this problem is NP-hard [18], but, if the y-coordinate of each vertex is prescribed, it can be solved in polynomial time [13, 21, 25], which suggests that a large part of the challenge of Upward

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 99; pp. 99:1–99:17


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Planarity comes from choosing an appropriate y-coordinate for each vertex. However, when both the y-coordinate and the x-coordinate of each vertex are prescribed, the problem is yet again NP-hard [27], indicating that another part of the challenge comes from drawing the edges in a y-monotone non-crossing fashion while respecting the given or chosen coordinates of their endpoints. The paper at hand is concerned with the parameterized complexity of a generalization of the latter of these two variants of Upward Planarity, which is known as Constrained Level Planarity. It is expressed in terms of so-called level graphs, which are defined next; we adopt the notation and terminology used in [27].

Level Planarity. A *level graph* $\mathcal{G} = (G, \gamma)$ is a directed graph $G = (V, E)$ together with a *level assignment*, which is a function[1] $\gamma\colon V \to \mathbb{R}$ where $\gamma(u) < \gamma(v)$ for every edge $(u, v) \in E$. For every $i \in \mathbb{R}$ where $V_i = \{v \in V \mid \gamma(v) = i\}$ is non-empty, the set V_i is called a (the i-th) *level* of $\mathcal{G}$. The *width* of level V_i is $|V_i|$. The *level-width* of $\mathcal{G}$ is the maximum width of any level in $\mathcal{G}$ and the *height* of $\mathcal{G}$ is the number of (non-empty) levels. A *level planar drawing* of $\mathcal{G}$ is an upward planar drawing of G where the y-coordinate of each vertex v is $\gamma(v)$. We use L_i to denote the horizontal line with y-coordinate i. The level graph $\mathcal{G}$ is called *proper* if every edge spans two consecutive levels, that is, for every edge $(u, v) \in E$ there is no level V_j with $\gamma(u) < j < \gamma(v)$. The problem Level Planarity asks whether a given level graph admits a level planar drawing. In a series of papers [13, 21, 24, 25], it was shown that Level Planarity can be solved in linear[1] time; we refer to [16] for a more detailed discussion of the history of the corresponding algorithm and of alternative approaches to solve Level Planarity.

Constrained Level Planarity. In 2017, Brückner and Rutter [8] and Klemz and Rote [27] independently introduced and studied two closely related variants of Level Planarity, which are defined in the following. A *constrained level graph* $\mathcal{G} = (G, \gamma, (\prec_i)_i)$ is a triple corresponding to a level graph (G, γ) equipped with a family $(\prec_i)_i$ containing, for each level V_i, a partial order $\prec_i$ on the vertices V_i. A *constrained* level planar drawing of $\mathcal{G}$ is a level planar drawing of (G, γ) where, for each level V_i, the left-to-right order of the vertices V_i corresponds to a linear extension of $\prec_i$. For a pair of vertices $u, v \in V_i$ with $u \prec_i v$, we refer to $u \prec_i v$ as a *constraint* on u and v. The problem Constrained Level Planarity (CLP) asks whether a given constrained level graph admits a constrained level planar drawing. Ordered Level Planarity (OLP) corresponds to the special case of CLP where the given partial orders are total orders, which is polynomial time equivalent to prescribing the x-coordinate (in addition to the y-coordinate) of each vertex.

Klemz and Rote [27] established a complexity dichotomy for OLP with respect to both the maximum degree and the level-width. In particular, they showed that OLP is NP-hard even when restricted to the case where $\mathcal{G}$ has a level-width of 2 and the underlying undirected graph of G is a disjoint union of paths, i.e., a graph of maximum degree 2, path-width (and tree-width) 1, and feedback vertex/edge set number 0. In fact, with a simple modification[2]

[1] Traditionally, in the literature, the level assignment γ is defined as a surjective function that maps to an integer interval $\{1, 2, \ldots, h\}$; it merely acts as a convenient way to encode a total preorder on V. It is well known that the traditional and our (more general) definition are polynomial-time equivalent: algorithms designed assuming the classical definition can also be applied in the more general context: one simple has to first sort the vertices by y-coordinates and then apply the traditional algorithm using the sorting-ranks as y-coordinates. We are using the given general definition as it eases the description of our algorithms; though, specific polynomial runtimes obtained in previous work that are stated in our introduction assume the classical definition.

[2] In the variable gadget of every variable u_j, one can remove the subdivision vertices of the tunnels with index larger than j. This modification does not influence the realizability of the instance since the left-to-right order of all tunnels is already fixed due to the subdivision vertices on level ℓ_0.

to their construction, the underlying undirected graph produced by the reduction becomes a disjoint union of paths with constant length, implying that even the tree-depth is bounded. (The definitions of all these classical graph parameters can be found, e.g., in [10].) It follows that CLP is NP-hard in the same scenario and when, additionally, each of the prescribed partial orders $\prec_i$ is a total order. OLP is (trivially) solvable in linear[1] time when restricted to proper instances [27]. In contrast, an instance of CLP can always be turned into an equivalent proper instance by subdividing each edge on each level it passes through without introducing any constraints on the resulting subdivision vertices [8]. Hence, CLP is NP-hard even in the proper case. Independently, Brückner and Rutter [8] also presented a proof for the NP-hardness of CLP, which relies on a very different strategy. It is not obvious whether the graphs produced by their construction have bounded tree-width, however, it is not difficult to see[3] that the socket/plug gadget used in their reduction can be utilized in the context of a reduction from 3-Partition to show that CLP remains NP-hard for proper instances whose underlying undirected graph is a single (rooted) tree of constant depth. In fact, the unpublished full version of [8] features such a construction [29].

On the positive side, Brückner and Rutter [8] presented a polynomial time algorithm for the special case of CLP where the input graph G has a single source. They further improved the runtime of this algorithm in [9]. Very recently, Blažej, Klemz, Klesen, Sieper, Wolff, and Zink studied the parameterized complexity of CLP and OLP with respect to the height of the input graph [7]. They showed that OLP parameterized by height is XNLP-complete (implying that it is in XP, but $W[t]$-hard for every $t \geq 1$). In contrast, CLP is NP-hard even if restricted to instances of height 4, but it can be solved in polynomial time if restricted to instances of height at most 3.

Other related work. Several other restricted variants of Level Planarity have been studied, e.g., Clustered Level Planarity [15, 2, 27], T-Level Planarity [30, 2, 27], and Partial Level Planarity [8]. In particular, in Partial Level Planarity, a given level planar drawing of a subgraph of the input graph $\mathcal{G}$ has to be extended to a full drawing of $\mathcal{G}$, which can be seen as a generalization of OLP and, in the proper case, a specialization of CLP. Level Planarity has been extended to surfaces different from the plane [4, 1, 5]. There are also related problems with a more geometric flavor, e.g., finding a level planar straight-line drawing where each face is bounded by a convex polygon [23, 26], and problems where the input is an undirected graph without a level assignment and the task is to find a crossing-free drawing with y-monotone edges that, if interpreted as a level planar drawing, satisfies or optimizes certain criteria, e.g., being proper or having minimum height [6, 14, 22].

Contribution. As discussed above, the previous results of Brückner and Rutter [8] and Klemz and Rote [27] rule out the existence of FPT-time (even XP-time) algorithms for CLP when considering the tree-width, path-width, tree-depth, or feedback vertex set number as a parameter, even when restricted to OLP or proper CLP instances (unless P=NP). As all of these parameters are bounded[4] by the vertex cover number, it is natural to study the parameterized complexity of CLP with respect to this parameter. We prove the following main result:

[3] In the strongly NP-hard 3-Partition problem [17], one has to partition $3n$ positive integers $B/4 < a_1, a_2, \ldots, a_{3n} < B/2$ of total sum nB into n triples (or *buckets*) of sum (or *size*) B. To reduce to CLP, one can simulate a bucket of size B as a sequence of B consecutive sockets and a number a_i as a_i plugs that are connected in a star-like fashion to a common ancestor v_i located above all these plugs. Finally, all ancestors v_i and all sockets are connected in a star-like fashion to a common root vertex.

[4] More precisely, $\mathrm{tw}(G) \leq \mathrm{pw}(G) \leq \mathrm{td}(G) - 1 \leq \mathrm{vc}(G)$ and $\mathrm{fvs}(G) \leq \mathrm{vc}(G)$.

▶ **Theorem 1.** *CLP parameterized by the vertex cover number is FPT.*

In view of the previous intractability statements, Theorem 1 is best-possible in several regards: a speed-up to polynomial time or a generalization to the aforementioned smaller graph parameters is not possible, even if restricting to OLP or proper CLP instances.

Organization. The proof of Theorem 1 and the remainder of this paper are organized as follows. We begin by introducing some basic notation, terminology, and other preliminaries in Section 2. In particular, we describe a partition of the vertex set of a given constrained level graph $\mathcal{G}$ into different categories with respect to a given vertex cover C and we show that the vertices of two of these categories are, in some sense, easy to handle. In Section 3, we introduce cores and (refined) visibility extensions of level planar drawings with respect to a fixed vertex cover C. Intuitively, the core-induced subdrawing of a (refined) visibility extension of a constrained level planar drawing Γ^* of $\mathcal{G}$ with respect to C is a drawing Λ_{core} that captures crucial structural properties of Γ^* and whose total complexity is bounded in $|C|$. The latter allows us to efficiently obtain such a core-induced subdrawing via the process of exhaustive enumeration. This is the first main step of the algorithm corresponding to the proof of Theorem 1, which is described in Section 4. Due to the properties of the core-induced subdrawing, it is then possible to place the remaining vertices in the subsequent main steps of the algorithm, each of which is concerned with the placement of the vertices of a particular vertex category. We conclude with a discussion of an open problem in Section 5. Proofs of statements marked with a $\star$ can be found in the appendix of the full preprint version [28].

2 Preliminaries

Conventions. Recall that in a level graph $\mathcal{G} = (G = (V, E), \gamma)$, the graph G is directed by definition. However, when it comes to vertex-adjacencies, we always refer to the underlying undirected graph of G, that is, the neighborhood of $v \in V$ is $\mathrm{N}_G(v) = \{u \in V \mid (u, v) \in E \vee (v, u) \in E\}$, the degree of v is $|\mathrm{N}_G(v)|$, and "a vertex cover of G" refers to a vertex cover[5] of the underlying undirected graph of G. The *level planar embedding* of a level planar drawing of $\mathcal{G}$ lists, for each level V_i, the left-to-right sequence of vertices and edges intersected by the line L_i in the drawing. Note that this corresponds to an equivalence class of drawings from which an actual drawing is easily derived, which is why algorithms for constructing level planar drawings (including our algorithms) usually just determine a level planar embedding. For brevity, we often use the term "drawing" as a synonym for "embedding of a drawing".

Vertex categories & notation. For $m \in \mathbb{N}$, we use $[m]$ to denote the set $\{1, 2, \ldots, m\}$. Let $\mathcal{G} = (G = (V, E), \gamma)$ be a (constrained) level graph and let C be a vertex cover of G. An *ear* of $\mathcal{G}$ with respect to C is a degree-2 vertex of $V \setminus C$ that is a source or sink. For a subset $X \subseteq C$, we define $V_X(C) = \{v \in V \setminus C \mid \mathrm{N}_G(v) = X\}$. We partition the vertices $V \setminus C$ of the graph G into four sets $V_{=0}(C), V_{=1}(C), V_{=2}(C), V_{\geq 3}(C)$ where $V_{=0}(C) = \{v \in V \setminus C \mid \deg(v) = 0\}$, $V_{=1}(C) = \{v \in V \setminus C \mid \deg(v) = 1\}$ (the *leaves*), $V_{=2}(C) = \{v \in V \setminus C \mid \deg(v) = 2\}$, and $V_{\geq 3}(C) = \{v \in V \setminus C \mid \deg(v) \geq 3\}$. The set $V_{=2}(C)$ is further partitioned into two sets $V^{\mathrm{e}}_{=2}(C), V^{\mathrm{t}}_{=2}(C)$ where $V^{\mathrm{e}}_{=2}(C)$ contains the ears and $V^{\mathrm{t}}_{=2}(C)$ the non-ears, called *transition*

[5] A *vertex cover* of an undirected graph $G = (V, E)$ is a vertex set $C \subseteq V$ such that every edge in E is incident to at least one vertex in C. The *vertex cover number* of G is the size of a smallest vertex cover of G.

vertices. Let $v \in V_{=1}(C)$ and let $c \in C$ denote its (unique) neighbor. We say that v is a *leaf* of c. Similarly, let $u \in V_{=2}(C)$ and let $c_a, c_b \in C$ denote its (unique) two neighbors. We say that u is a *transition vertex* (*ear*) of c_a and c_b if $u \in V^{\mathrm{t}}_{=2}(C)$ (if $u \in V^{\mathrm{e}}_{=2}(C)$). We often omit C if it is clear from the context.

Let $\mathcal{G} = (G = (V, E), \gamma, (\prec_i)_i)$ be a constrained level graph and let C be a vertex cover of G. The main challenge when constructing a constrained level planar drawing of $\mathcal{G}$ is the placement of the leaves, ears, and transition vertices (along with their incident edges). Indeed, it is not difficult to insert the isolated vertices (which include $V_{=0}$) in a post-processing step (performing a topological sort on each level), see Lemma 4. Moreover, since we may assume G to be planar, the size of $V_{\geq 3}(C)$ is linear in $|C|$. This well known bound can be derived, e.g., by combining the fact that the complement of a vertex cover is an independent set with the following statement (setting $A = C$).

▶ **Lemma 2** ([10, Corollary 9.25]). *Let $G = (V, E)$ be a planar graph and $A \subseteq V$. Then there are at most $2|A|$ connected components in the subgraph of G induced by $V \setminus A$ that are adjacent to more than two vertices of A.*

▶ **Corollary 3** (Folklore). *Let G be a planar graph and let C be a vertex cover of G. Then $|V_{\geq 3}(C)| \leq 2k$, where $k = |C|$.*

▶ **Lemma 4** ($\star$). *Let $\mathcal{G} = (G, \gamma, (\prec_i)_i)$ be a constrained level graph, let G' be the subgraph of G induced by the non-isolated vertices V', and let γ' and $(\prec'_i)_i$ be the restrictions of γ and $(\prec_i)_i$ to V', respectively. There is an algorithm that, given $\mathcal{G}$ and a constrained level planar drawing Γ' of $\mathcal{G}' = (G', \gamma', (\prec'_i)_i)$, constructs a constrained level-planar drawing of $\mathcal{G}$ in polynomial time.*

Our main algorithm will exploit the fact that only few ears may share a common level:

▶ **Lemma 5** ($\star$). *Let $\mathcal{G} = (G, \gamma)$ be a level graph, let Γ be a level planar drawing of $\mathcal{G}$, let C be a vertex cover of G. Then there are at most $2|C|$ ears with respect to C per level.*

Compatible edge orderings. Let Γ be a level planar drawing of a (possibly constrained) level graph $\mathcal{G} = (G = (V, E), \gamma)$ without isolated vertices. We will now define a useful (not necessarily unique) linear order $\prec^{\mathrm{e}}$ on the edges E with respect to Γ. We refer to $\prec^{\mathrm{e}}$ as an edge ordering of E that is *compatible* with Γ. Compatible edge orderings can be seen as a generalization of a linear order described in [27, Proof of Lemma 4.4] for a set of pairwise disjoint y-monotone paths, which in turn follows considerations about horizontal separability of y-monotone sets by translations [12, 3, 19, 20]. Intuitively, $\prec^{\mathrm{e}}$ is a linear extension of a partial order in which $e \in E$ precedes $f \in E$ if it is possible to shoot a horizontal rightwards ray from e to f in Γ without crossing other edges before reaching f. Formally, we say that a vertex v is *visible from the left* in Γ if the horizontal ray r_v emanating from v to the left intersects Γ only in v. We say that an edge $e = (u, v)$ is *visible from the left* in Γ if the closed (unbounded) region that is to the left of e and whose boundary is described by e, r_u, r_v intersects Γ only in e. The order $\prec^{\mathrm{e}}$ is now constructed as follows: the minimum of $\prec^{\mathrm{e}}$ is an edge e_1 of E that is visible from the left in Γ. Such an edge always exists [27, 19, 20]: among the edges whose lower endpoint is visible from the left, the edge with the topmost lower endpoint is visible from the left. Let Γ' denote the drawing derived from Γ by removing e_1 and any isolated vertices created by the removal of e_1. The restriction of $\prec^{\mathrm{e}}$ to the remaining edges $E \setminus e_1$ corresponds to an edge ordering compatible with Γ', which is constructed recursively. Note that $\mathcal{G}$ and $\prec^{\mathrm{e}}$ uniquely describe the drawing Γ and, given $\mathcal{G}$ and $\prec^{\mathrm{e}}$, it is possible to construct Γ in polynomial time (by traversing $\prec^{\mathrm{e}}$ in reverse).

3 Visibility extensions and cores

In this section, we introduce and study (refined) visibility extensions and cores of level planar drawings. We will see that the core-induced subdrawing of a (refined) visibility extension of a level planar drawing Γ with respect to some vertex cover C captures crucial structural properties of Γ while having a size that is bounded in $|C|$.

Visibility extensions. Let Γ be a level planar drawing of a level graph $\mathcal{G} = (G = (V, E), \gamma)$. A *visibility edge* e for Γ is (1) a y-monotone curve that joins two vertices of Γ and can be inserted into Γ without creating any crossings (but possibly a pair of parallel edges); or (2) a horizontal segment that joins two consecutive vertices on a common level of Γ and can be inserted into Γ without creating any crossings. A *visibility extension* of Γ with respect to a vertex set $V' \subseteq V$ is a drawing Λ derived from Γ by inserting a maximal set of pairwise non-crossing visibility edges incident only to vertices of V' such that for each pair e, e' of parallel edges in Λ there is at least one vertex of V' in the interior of the simple closed curve formed by e and e'; for an illustration see Figure 1(a). We remark that if $V' = V$, then Λ is essentially an interior triangulation containing Γ. However, we will always choose V' to be a (small) vertex cover, resulting in a much sparser yet still connected augmentation:

▶ **Lemma 6 (⋆).** *Let $\mathcal{G} = (G = (V, E), \gamma)$ be a level graph without isolated vertices, let C be a vertex cover of G, let Γ be a level planar drawing of $\mathcal{G}$, let Λ be a visibility extension of Γ with respect to C, and let Λ_C be the subdrawing of Λ that is induced by C. Then Λ_C is connected and has $\mathcal{O}(k)$ edges, where $k = |C|$.*

Cores and refined visibility extensions. Intuitively, the core of a level planar drawing is a subset of the vertex set with certain crucial properties. To define it formally, we will first classify the ears of the drawing according to several categories. The concepts introduced in this paragraph are illustrated in Figure 1(b). Let $\mathcal{G} = (G = (V, E), \gamma)$ be a level graph, let C be a vertex cover of G, and let Γ be a level planar drawing of $\mathcal{G}$. Consider an ear $v \in V^{\mathrm{e}}_{=2}(C)$ with neighbors c_a, c_b where $\gamma(c_a) \geq \gamma(c_b)$. If $\gamma(v) > \gamma(c_a)$, we say that v is a *top* ear. Otherwise (if $\gamma(v) < \gamma(c_b)$), we say that v is a *bottom* ear. Assume that $\gamma(c_a) > \gamma(c_b)$ and that v is a top ear. If in Γ the edge $c_b v$ is drawn to the left (right) of c_a, we say that v is a *left* (*right*) ear in Γ. The terms "left" and "right" are defined analogously for bottom ears. If $\gamma(c_a) = \gamma(c_b)$, we consider v to be a left ear if it is a top ear; otherwise it is a right ear. Consider a pair $c_a, c_b \in C$ with at least one left ear in Γ and let Γ' denote the subdrawing of Γ induced by the set of edges that are incident to at least one left ear of c_a, c_b in Γ. Note that either all these ears are top ears or all these ears are bottom ears in Γ, and they are arranged in a nested fashion. In case of top (bottom) ears, we refer to the unique one with the largest (smallest) y-coordinate as the *outermost* left ear of c_a, c_b. The *innermost* left ear of c_a, c_b is defined symmetrically. If Γ' has an interior (i.e., bounded) face f such that the open region enclosed by the boundary of f contains a vertex of C in Γ, then we say that the two ears v_1, v_2 on the boundary of f are *bounding ears* of c_a, c_b in Γ with respect to C. Moreover, we say that v_1, v_2 are a pair of *matching* bounding ears whose *region* corresponds to f. The terms "outermost", "innermost" and "(region of matching pair of) bounding ears" are analogously defined for the right ears of c_a, c_b. Every vertex of Γ that is an outermost, innermost, or bounding ear (with respect to some pair $c_a, c_b \in C$) is called *crucial* with respect to C.

The *core* of Γ with respect to C is the (unique) subset of V that contains C, $V_{\geq 3}(C)$, as well as all crucial ears of Γ with respect to C. The subdrawing Λ_{core} of a visibility extension Λ of Γ with respect C that is induced by the core of Λ with respect to C captures crucial

structural properties of Γ, which we will exploit in our main algorithm for constructing constrained level planar drawings in FPT-time. Due to the fact that Λ_{core} has only $\mathcal{O}(|C|)$ vertices and edges, it is not difficult to "guess" Λ_{core} in XP-time (via the process of exhaustive enumeration) when given $\mathcal{G}$ and C. The main bottleneck is the enumeration of all possible sets of crucial ears. To improve the runtime of this step to FPT-time, we will now describe a variant of visibility extensions that contains some additional ears, which take over the role of the original crucial vertices. Loosely speaking, one can create such a drawing by placing one or two new ears near each crucial ear in a visibility extension. The resulting augmentation retains the helpful structural properties of its underlying visibility extension and we will see that the (positions of the) crucial vertices of some such augmentation can be guessed more efficiently since we can restrict the possible levels of these new vertices to a small set. Formally, a *refined visibilty extension* Λ' of Γ is a crossing-free drawing of a level graph $\mathcal{G}' = (G' = (V', E'), \gamma')$ such that G is a subgraph of G', C is a vertex cover of G', every vertex in $V' \setminus V$ is an ear with respect to C and its incident edges are drawn as y-monotone curves, the subdrawing of Λ' induced by V is a visibility extension Λ of Γ, the crucial ears of Λ' are precisely the vertices in $V' \setminus V$, and $|V' \setminus V| \in \mathcal{O}(|C|)$.

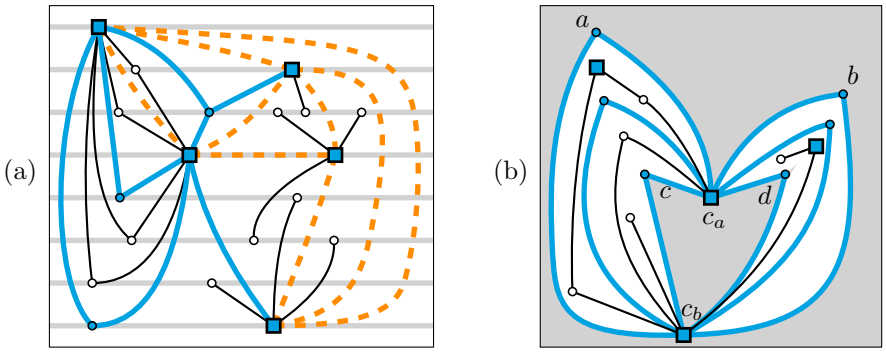


Figure 1 In this (and all other) figure(s), filled square vertices belong to a vertex cover C of the depicted graph and filled (round or square) vertices belong to the core of the shown drawing with respect to C. (a) A drawing Γ (non-dashed edges) is augmented with visibility edges (dashed) to obtain a visbility extension Λ with respect to C (note that this augmentation is not unique). The thick (non-dashed or dashed) edges and filled vertices represent Λ_{core}. (b) All filled round vertices are top crucial ears of c_a and c_b. All of them are bounding ears except for b and c. The vertex a / b / c / d is an outermost left / outermost right / innermost left / innermost right ear.

▶ **Lemma 7** (⋆). *Let $\mathcal{G} = (G = (V, E), \gamma)$ be a level graph without isolated vertices, let C be a vertex cover of G, let Γ be a level planar drawing of $\mathcal{G}$, let Λ be a refined or non-refined visibility extension of Γ with respect to C, and let Λ_{core} the subdrawing of Λ induced by the core of Λ with respect to C. Then Λ_{core} is connected and has $\mathcal{O}(k)$ vertices and $\mathcal{O}(k)$ edges, where $k = |C|$.*

4 Algorithm

In this section, we describe the algorithm corresponding to the proof of Theorem 1. Let $\mathcal{G} = (G = (V, E), \gamma, (\prec_i)_i)$ be a constrained level graph and let C be a vertex cover of G. Our goal is to construct a constrained level planar drawing of $\mathcal{G}$ or correctly report that such a drawing does not exist. In view of Lemma 4, we may assume that $\mathcal{G}$ has no isolated vertices. To construct the desired drawing, we proceed in three main steps. In Step 1, we "guess" a

core-induced subdrawing of a refined visibility extension of a constrained level planar drawing of $\mathcal{G}$ with respect to C (via the process of exhaustive enumeration). In Step 2, we augment our drawing by inserting the transition vertices of $\mathcal{G}$ with respect to C. In Step 3, we finalize our drawing by inserting the leaves and ears of $\mathcal{G}$ with respect to C.

Step 1: Guessing a core-induced subdrawing. Assume that there is a constrained level planar drawing Γ^* of $\mathcal{G}$, let Λ^* be a refined visibility extension of Γ^* with respect to C, and let Λ_{core} be the subdrawing of Λ^* induced by the core of Λ^* with respect to C. The procedures corresponding to Steps 2 and 3 of our algorithm are guaranteed to produce a constrained level planar drawing (not necessarily Γ^*) of $\mathcal{G}$ when given Λ_{core}. Hence, the goal of Step 1 is to determine (or, rather, guess) Λ_{core}, given $\mathcal{G}$ and C. More precisely, we will construct a set $\mathcal{F}$ of drawings such that $\Lambda_{\text{core}} \in \mathcal{F}$ and the number $|\mathcal{F}|$ of drawings in $\mathcal{F}$ is sufficiently small. For each drawing in $\mathcal{F}$, we then apply Steps 2 and 3 of the algorithm (incurring a factor of $|\mathcal{F}|$ in the total running time). Given that $\Lambda_{\text{core}} \in \mathcal{F}$, one of the iterations is guaranteed to terminate with a constrained level planar drawing of $\mathcal{G}$.

▶ **Lemma 8 ($\star$).** *Let $\mathcal{G} = (G = (V, E), \gamma, (\prec_i)_i)$ be a constrained level graph without isolated vertices, let C be a vertex cover of G, and let Γ^* be a constrained level planar drawing of $\mathcal{G}$. There is an algorithm that, given $\mathcal{G}$ and C, constructs a set $\mathcal{F}$ of $2^{\mathcal{O}(k \log k)}$ drawings in $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ time, where $n = |V|$ and $k = |C|$, such that all drawings in $\mathcal{F}$ have size $\mathcal{O}(k)$ and are level planar drawings of subgraphs of G induced by C and $V_{\geq 3}(C)$ that respect γ and the orderings $\prec_i$ and are augmented by some visibility edges and additional ears (with respect to C). Further, there exists a refined visibility extension Λ^* of Γ^* such that the subdrawing Λ_{core} of Λ^* induced by the core of Λ^* with respect to C is contained in $\mathcal{F}$.*

Proof sketch. We introduce the following terminology: let x be a vertex in a level planar drawing (possibly augmented by some horizontal edges). Let ℓ be the y-coordinate of x and let ℓ' be the largest y-coordinate of a vertex below x (if there no such vertex, we set $\ell' = \ell - 1$). We say that the line $L_{(\ell+\ell')/2}$ is *directly below* x. The line *directly above* x is defined symmetrically.

We proceed in two main steps. In the first main step, we show that there *exists* a refined visibility extension Λ^* of Γ^*. To this end, we start with a visibility extension Λ of Γ^* and describe an incremental strategy that performs a total of $\mathcal{O}(k)$ augmentation steps, in each of which a new ear is added that takes over the role of a crucial ear in Γ^*. (The description of this first main step is deferred to the appendix of the full preprint version [28].) In the second main step, we discuss the construction of the desired family $\mathcal{F}$. To this end, let Λ_{core} be the subdrawing of Λ^* induced by the core of Λ^* with respect to C. The drawing Λ_{core} is uniquely described by $\mathcal{G}$, C, the set of visibility edges of Λ_{core} (and Λ^*), the set of crucial ears of Λ_{core} (and Λ^*) together with their level assignments and their incident edges, and a compatible edge ordering of the nonhorizontal edges of Λ_{core}. The graph $\mathcal{G}$, as well as the vertex cover C are given, so it suffices to enumerate all possible options for the remaining elements.

There are $m_{\text{vis}} \in \mathcal{O}(k)$ visibility edges by Lemma 7 and each of these visibility edges joins a pair of vertices in C. Hence, there are at most $\binom{k}{2}^{m_{\text{vis}}} \subseteq k^{\mathcal{O}(k)} \subseteq 2^{\mathcal{O}(k \log k)}$ possible options for choosing the set of visibility edges. To enumerate the set of crucial ears along with their level assignments, we mimic the aforementioned incremental strategy for constructing Λ^*: we first enumerate all options to pick the pair of neighbors of the first new vertex along with its level, then, for each of these options, we enumerate all options to pick the pair of neighbors of the the second vertex along with its level, etc., until we have obtained all options to pick

the desired $\mathcal{O}(k)$ vertices together with their levels. More precisely, suppose we have already enumerated all options to pick the first i vertices together with their neighbors and levels. For each of these options, to enumerate all options to pick the next vertex v', we go through all ways to pick its two neighbors $u, w \in C$ and through all ways to pick the level of v'. There are $\mathcal{O}(k)$ pairs of vertices in C with ears (by Lemma 7). To bound the number of ways to pick the level of v', we make use of the fact that whenever the incremental strategy for constructing Λ^* places a new vertex v', it is assigned to a new level directly above or below a level of one of the following categories:

- a level with a vertex in C ($\mathcal{O}(k)$ possibilities),
- a level with a vertex in $V_{\geq 3}(C)$ ($\mathcal{O}(k)$ possibilities by Corollary 3),
- a level of a vertex that does not belong to $\mathcal{G}$, i.e., a level used for one of the already placed vertices ($\mathcal{O}(i) \subseteq \mathcal{O}(k)$ possibilities),
- a level with a top-most or bottom-most vertex of $\mathcal{G}$ ($\mathcal{O}(1)$ possibilities),
- a level with a top-most top ear, a top-most bottom ear, a bottom-most top ear, or a bottom-most bottom ear of some pair of vertices in C ($\mathcal{O}(k)$ possibilities by Lemma 7),
- a level with a top-most or bottom-most vertex of a connected component that contains a vertex of C in the graph obtained by removing u and w from the current graph ($\mathcal{G}$ together with the visbility edges and the already added vertices) ($\mathcal{O}(k)$ possibilities).

In total, for a fixed pair of neighbors u, w, there are thus $\mathcal{O}(k)$ options to pick a level for v'. We immediately discard level assignments for which v' is no ear. By multiplying with the number of ways to choose the neighbors, we obtain $\mathcal{O}(k^2)$ options to choose v' and its level. Multiplying the number of options for all $\mathcal{O}(k)$ steps together, we obtain a total number of $k^{\mathcal{O}(k)} \subseteq 2^{\mathcal{O}(k \log k)}$ ways to create the set of crucial ears along with their levels. By multiplying with the number of ways to choose the visibility edges, we obtain a total of $2^{\mathcal{O}(k \log k)}$ options to choose the graph that corresponds to Λ_{core}. For each of these options we enumerate all $k^{\mathcal{O}(k)} \subseteq 2^{\mathcal{O}(k \log k)}$ permutations of the set of non-horizontal edges and, interpreting the permutation as a compatible edge order, try to construct a level planar drawing for which this order is compatible (cf. Section 2). If we succeed, we check whether the drawing is conform with $(\prec_i)_i$ and can be augmented with the horizontal visibility edges. If so, we include the drawing in the set $\mathcal{F}$ of reported drawings. The size of the thereby constructed set $\mathcal{F}$ is bounded by $2^{\mathcal{O}(k \log k)}$ and it is guaranteed to contain Λ_{core} by construction. ◀

Step 2: Inserting transition vertices. We now describe how to insert the transition vertices into the core-induced subdrawing Λ_{core} of the (refined) visibility extension Λ^*. Our plan is to first show that in Λ^*, every transition vertex is placed "very close to" some visibility edge. Intuitively, this means that the visibility edges of Λ_{core} act as placeholders near which the transition vertices have to be placed. We will describe a procedure that does so while carefully taking into account the given partial orderings $\prec_i$ and prove its correctness by means of an (somewhat technical) exchange argument. To formalize the notion of "very close to", let e be an edge of a level planar drawing joining two vertices a, b such that there is a degree-2 vertex t with neighbors a, b and $\gamma'(b) < \gamma'(t) < \gamma'(a)$ where γ' is the level assignment. We say that t is drawn in the *vicinity* of e with respect to a vertex set V' if the simple closed curve formed by e and the two edges incident to t does not contain a vertex of V' in its interior.

▶ **Lemma 9** ($\star$). *Let $\mathcal{G} = (G = (V, E), \gamma, (\prec_i)_i)$ be a constrained level graph without isolated vertices, let C be a vertex cover of G, let Γ^* be a constrained level planar drawing of $\mathcal{G}$, let Λ^* be a refined or non-refined visibility extension of Γ^* with respect to C, and let Λ_{core} the subdrawing of Λ^* induced by the core of Λ^* with respect to C. There is an algorithm*

that, given $\mathcal{G}$, C and Λ_{core}, inserts all transition vertices $V^{\mathrm{t}}_{=2}(C)$ (and their incident edges) into vicinities (with respect to C) of visibility edges in Λ_{core} in polynomial time such that the resulting drawing $\Lambda^{\mathrm{t}}_{\mathrm{core}}$ can be extended to a drawing whose restriction to G is a constrained level planar drawing of $\mathcal{G}$.

Step 3: Inserting leaves and ears. In this step, we start with the output of Step 2 and finalize our drawing by placing all the vertices that are still missing.

▶ **Lemma 10.** *Let $\mathcal{G} = (G = (V, E), \gamma, (\prec_i)_i)$ be a constrained level graph without isolated vertices, let C be a vertex cover of G, let Γ^* be a constrained level planar drawing of $\mathcal{G}$, let Λ^* be a refined or non-refined visibility extension of Γ^* with respect to C in which each transition vertex is placed in the vicinity of some visibility edge with respect to C, and let Λ_{core} ($\Lambda^{\mathrm{t}}_{\mathrm{core}}$) be the subdrawing of Λ^* induced by the core (and the transition vertices) of Λ^* with respect to C. There is an algorithm that, given $\mathcal{G}$, C, Λ_{core} and $\Lambda^{\mathrm{t}}_{\mathrm{core}}$, extends Λ_{core} to a drawing Γ whose restriction to G is a constrained level planar drawing of $\mathcal{G}$ in $2^{\mathcal{O}(k^2 \log k)} n^{\mathcal{O}(1)}$ time, where $n = |V|, k = |C|$.*

Proof. The only vertices of G that are missing in $\Lambda^{\mathrm{t}}_{\mathrm{core}}$ are the leaves and the non-crucial ears with respect to C (in case Λ^* is a *refined* visibility extension, the non-crucial ears are exactly the ears of $\mathcal{G}$). Our plan to insert them into our drawing is as follows. We begin by introducing more structure in $\Lambda^{\mathrm{t}}_{\mathrm{core}}$ and Λ^* by adding some additional visibility edges and making some normalizing assumptions, which will simplify the description of the upcoming steps. In particular, this step will ensure that for each missing ear, there are essentially only (up to) two possible placements, which will allow us to enumerate all possible ear placements (so-called ear orientations) on a given level in FPT-time. We then describe a partition of the plane into so-called cells in a way that is very reminiscent of the well-known trapezoidal decomposition from the field of computational geometry, cf. [11]. We merge some cells into so-called channels, which correspond to connected y-monotone regions in which the missing leaves along with their incident edges will be drawn (a region is *y-monotone* if its intersection with every horizontal line is connected). We then introduce (and describe an enumerative process that constructs in FPT-time) a so-called traversal sequence that is compatible with Λ^*, which is a sequence of sets of channels with several useful structural properties related to Λ^*. In particular, this sequence, in some sense, sweeps the plane from left to right in a way where for each edge incident to a leaf in Λ^*, at some point there is a channel that contains it. Exploiting the properties of the traversal sequence, we then describe how to construct a so-called insertion sequence for the leaves on a given level with respect to a given placement of the ears of that level in polynomial time. Such an insertion sequence does not necessarily exist for every placement of ears, but we are guaranteed to find one by enumerating all possible ear placements of the level. This computation is performed independently for each level. Finally, we show how to construct in polynomial time the desired drawing Γ when given an insertion sequence along with its ear placement for each level. Notably, the final step can be executed even if some of the ear placements are different from the ones used in Λ^*. Let us proceed to formalize these ideas.

Augmenting and normalizing Λ_{core}, $\Lambda^{\mathrm{t}}_{\mathrm{core}}$, and Λ^*. Let $e = (u, v)$ be a visibility edge of Λ^* (and $\Lambda^{\mathrm{t}}_{\mathrm{core}}$) that has at least one transition vertex in its vicinity. In both $\Lambda^{\mathrm{t}}_{\mathrm{core}}$ and Λ^*, we add two copies of e; one directly to the left of the leftmost transition vertex and the other directly to the right of the rightmost transition vertex in the vicinity of e, which is possible to do in a y-monotone fashion and without introducing any crossings; see Figure 2(a). Note that the region enclosed by these two edges only contains transitions vertices of u and v, as well as leaves of u or v. We repeat this operation for all visibility edges e.

The following steps are illustrated in Figure 2(b). Let f be a face of Λ^t_{core} that is bounded by four vertices v_1, v_2, u_1, u_2 where $v_1, v_2 \in C$ and $u_1, u_2 \in V_{\{v_1,v_2\}}$ are either both left ears or both right ears. Without loss of generality, assume they are both left ears with $\gamma(v_1) \leq \gamma(v_2) < \gamma(u_1) < \gamma(u_2)$. We add copies of the edges (v_1, u_2) and (v_2, u_2) in f in both Λ^t_{core} and Λ^*, which can be done without introducing crossings. These edges partition f into three regions. Note that in Λ^* these regions only contain leaves and non-crucial ears. Without loss of generality, we will assume that each leaf in f is either placed in the region bounded by (v_1, u_2) and its copy or the region bounded by (v_2, u_2) and its copy (note that a leaf v that is adjacent to v_1 cannot have a constraint of the form $w \prec_i v$, where w is a non-crucial ear in f; the situation for leaves adjacent to v_2 is symmetric). Thus, the remaining (central) third region only contains non-crucial ears and is henceforth called an *ear-face* of Λ^t_{core}. We repeat this modification for all faces such as f.

For the remainder of the proof, Λ^t_{core} and Λ^* are used to refer to the thusly augmented and normalized drawings. We also add all the new edges to Λ_{core} and use Λ_{core} to refer to this augmentation. Note that this implies that it now suffices to search for a drawing of $\mathcal{G}$ in which every non-crucial ear is placed in an ear-face, whereas no leaf is placed in an ear-face.

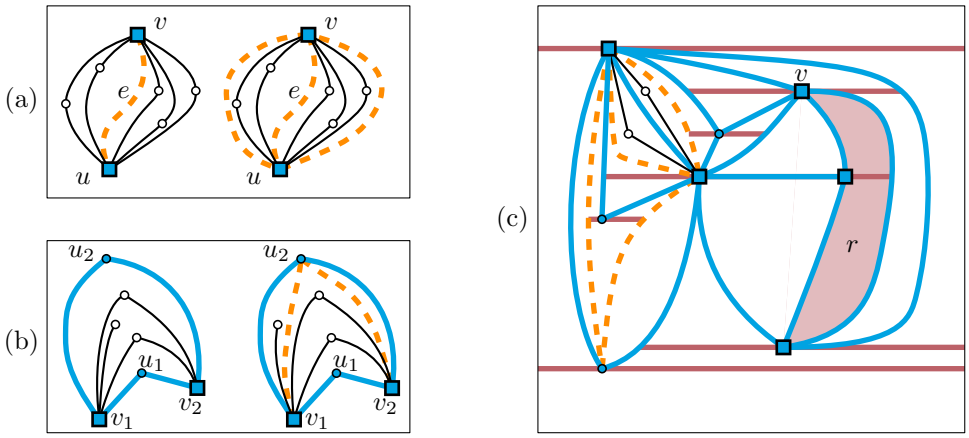


Figure 2 Like in all figures, filled square vertices belong to a vertex cover C of the depicted graph and filled (round or square) vertices belong to the core of the shown drawing with respect to C. Figures (a) and (b) visualize how Λ_{core}, Λ^t_{core}, and Λ^* are augmented by (a) enclosing transition vertices and (b) creating ear-faces with visibility edges (dashed). Moreover, (b) illustrates our normalizing assumption, i.e., the leaf can be moved to the exterior of the ear-face without violating any constraints. (c) The drawing Λ^t_{core} with its additional visibility edges (dashed) from the augmentation step and the horizontal rays and segments (thick, red) from the cell decomposition. The shaded region corresponds to a channel (v, r, R) from $v \in C$ to the cell r with $|R| = 2$.

Decomposition into cells. We will now describe a partition of the plane that essentially corresponds to a trapezoidal decomposition (cf. [11]) of Λ_{core}; for illustrations refer to Figure 2(c): for each vertex v in Λ_{core}, shoot a horizontal ray from v to the left until hitting an edge or vertex of Λ_{core}, then add the corresponding segment to Λ_{core}. In case the ray does not intersect any part of Λ_{core}, add the ray itself to Λ_{core}. Perform a symmetric augmentation by shooting a horizontal ray from v to the right. The maximal connected regions of the resulting partition of the plane are henceforth called *cells*. We consider the cells to be closed. Note that each cell is y-monotone and bounded by up to two horizontal segments or rays and up to two y-monotone curves. By Lemma 7, Λ_{core} has $\mathcal{O}(k)$ vertices and edges (note that the augmentation step copies each edge at most twice) and, hence, it has $\mathcal{O}(k)$ faces. Consequently, the number of cells is also $\mathcal{O}(k)$ since the insertion of a single segment or ray can only increase the number of faces (or, rather, maximal connected regions) by one.

Channels. Let $v \in C$, let R be a set of cells that do not belong to ear-faces, and let $r \in R$. Further, assume that R contains a cell that is incident to v. The triple $c = (v, r, R)$ is a *channel from v to r* if it is possible to draw a y-monotone curve in $\Lambda^{\mathrm{t}}_{\mathrm{core}}$ in the interior of the union of R that intersects each cell in R and does not cross any edge of $\Lambda^{\mathrm{t}}_{\mathrm{core}}$; as illustrated in Figure 2(c). We say that c *can be used* by a leaf $w \in V_{\{v\}}$ and that the edge e_w incident to w *can be drawn in* c if e_w can be drawn in $\Lambda^{\mathrm{t}}_{\mathrm{core}}$ in the union of R without any crossings and there is no channel (v, r', R') with this property for which $R' \subset R$. Further, we say that c *is used* by w if it can be used by w and the edge incident to w is drawn in the union of R in Λ^*. We use U to denote the set of all channels and $U_{\mathrm{used}} \subseteq U$ to denote the set of all channels that are used. The connectivity of $\Lambda^{\mathrm{t}}_{\mathrm{core}}$ (cf. Lemma 7) can be used to show:

▷ Claim 11 (⋆). $|U_{\mathrm{used}}| \leq |U| \in \mathcal{O}(k^2)$.

Traversal sequences. Let $\mathcal{U} = (\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_m)$ be a sequence of sets of channels. We say $\mathcal{U}$ is a *traversal sequence* if the following properties are fulfilled (see Figure 3 for illustrations):

(T1) Let $i \in [m]$ and let $(v, r, R) \in \mathcal{U}_i$ and $(v', r', R') \in \mathcal{U}_i$ with $v \neq v'$. Then the intersection of the line $L_{\gamma(v)}$ with the interior of the union of R' is empty.

(T2) Let c be a channel and let $a \leq b$ be two indices such that $c \in \mathcal{U}_a$ and $c \in \mathcal{U}_b$. Then for every $a \leq i \leq b$, $c \in \mathcal{U}_i$.

We say a channel u is *active* in $\mathcal{U}_i$ if it is contained in it, and otherwise it is *inactive* in it. We say a traversal sequence $\mathcal{U} = (\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_m)$ is *compatible* with Λ^* if the following conditions are satisfied (refer again to Figure 3 for illustrations):

(C1) For every channel c, there exists an $i \in [m]$ such that $c \in \mathcal{U}_i$ if and only if c is used.

(C2) There exists a compatible edge ordering $\prec^{\mathrm{e}}$ for the restriction of Λ^* to its nonhorizontal edges (recall that some visibility edges are horizontal) such that:

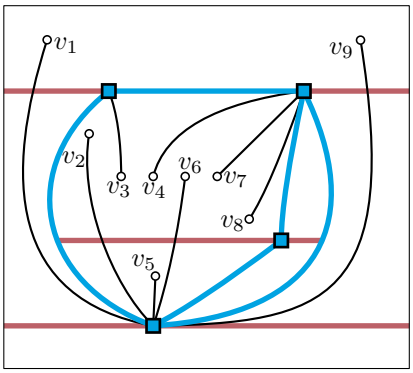
- **a.** Let e, e' be two edges that are incident to leaves and where $e \prec^{\mathrm{e}} e'$, let c be the channel used by e, and let c' be the channel used by e'. Then there exist indices i, i' such that $i \leq i'$ and $c \in \mathcal{U}_i$, $c' \in \mathcal{U}_{i'}$.
- **b.** Let $c_1, c_2 \in U_{\mathrm{used}}$ such that for every edge e_1 using c_1 and for every edge e_2 using c_2 we have $e_1 \prec^{\mathrm{e}} e_2$. Then there is no index $i \in [m]$ such that $\mathcal{U}_i$ contains both c_1 and c_2 (and every index for which c_1 is active is smaller than every index where c_2 is active).
- **c.** For every pair of used channels c_1, c_2 such that c_2 is being used by an edge e that succeeds all edges that use c_1 in $\prec^{\mathrm{e}}$ there exists an index i such that $c_2 \in \mathcal{U}_i$ and $c_1 \notin \mathcal{U}_i$ (and c_1 is active for some index smaller than i).

▷ Claim 12 (⋆). There exists a traversal sequence $\mathcal{U} = (\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_m)$ that is compatible with Λ^* and whose length is $m \in \mathcal{O}(k^2)$. Moreover, there is an algorithm that, given $\mathcal{G}$, C and $\Lambda^{\mathrm{t}}_{\mathrm{core}}$, computes a set of $2^{\mathcal{O}(k^2 \log k)}$ traversal sequences that contains $\mathcal{U}$ in $2^{\mathcal{O}(k^2 \log k)} n^{\mathcal{O}(1)}$ time.

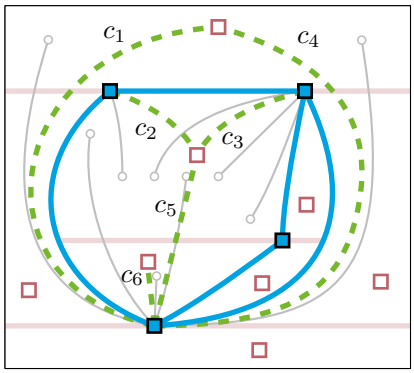
▷ Claim 13 (⋆). Let $\mathcal{U} = (\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_m)$ be a traversal sequence that is compatible with Λ^*, let $i \in [m]$, and let v be a leaf. Then $\mathcal{U}_i$ contains at most channel that can be used by v.

Ear orientations. Let $i \in [h]$ and let $V^{\mathrm{e}}_i \subseteq V_i$ be all non-crucial ears on the ith level. Further, consider a mapping $s \colon V^{\mathrm{e}}_i \to \{\mathrm{left}, \mathrm{right}\}$. We say that s is an *ear orientation* of level i. We say that s is *valid* if it is possible to insert the ears V^{e}_i (on the line L_i) along with their incident edges into $\Lambda^{\mathrm{t}}_{\mathrm{core}}$ such that the resulting drawing Λ is crossing-free, no constraint is violated (i.e., if $u \prec_i v$, then u is placed to the left of v), and for every $v \in V^{\mathrm{e}}_i$, we have that v is a left ear if and only if $s(v) = \mathrm{left}$. We say that Λ is *induced* by s. Note that for any ear $v \in V^{\mathrm{e}}_i$ there is at most one left ear-face and at most one right ear-face into which it can be

(a)



(b)



(c)

	$\mathcal{U}_1$	$\mathcal{U}_2$	$\mathcal{U}_3$	$\mathcal{U}_4$	$\mathcal{U}_5$	$\mathcal{U}_6$	$\mathcal{U}_7$	$\mathcal{U}_8$	$\mathcal{U}_9$
c_1	■								
c_2			■						
c_3				■	■	■	■	■	
c_4									■
c_5		■	■	■	■	■			
c_6					■				

Figure 3 Like in all figures, filled square vertices belong to a vertex cover C of the depicted graph. (a) The drawing Λ^* together with its cell decomposition. (b) The dashed (green) "edges" represent the used channels $c_1 \dots, c_6$ of Λ^* leading from vertices of C to cells in R, which are represented by unfilled (red) squares. (c) A traversal sequence compatible with Λ^* for which Property (C2) is satisfied for any compatible edge ordering that contains the edges of $v_1, v_2, \dots, v_9$ in this order.

inserted without introducing crossings. Hence, a valid ear orientation uniquely describes the ear-face in which each ear is placed. Further, note that no two ears of V_i^{e} can be placed in the same ear-face without introducing crossings. In contrast, whenever an ear orientation assigns only one ear to a given ear-face, it is possible to place the ear without introducing crossings. These properties make it easy to test whether a given ear orientation is valid and, if so, construct the (unique) induced drawing in polynomial time. In view of Lemma 5, this means we can enumerate all valid ear orientations of a given level in $2^{2k}n^{\mathcal{O}(1)}$ time.

Insertion sequences. Let $\mathcal{U} = (\mathcal{U}_1, \mathcal{U}_2, \dots, \mathcal{U}_m)$ be a traversal sequence that is compatible with Λ^*. Further, let $i \in [h]$, let s be a valid ear orientation of level i, and let Λ be its induced drawing. Finally, let $\mathcal{Q} = (Q_1, Q_2, \dots, Q_q)$ be a sequence with $Q_t = (v, j)$, $v \in V_i \cap V_{=1}$, and $1 \leq j \leq m$ for all $1 \leq t \leq q$. We say $\mathcal{Q}$ is an *insertion sequence* for i, s, and $\mathcal{U}$ if the following conditions are fulfilled (for an example, see Figure 4):

- **(I1)** Let $v \in V_i \cap V_{=1}$. Then there exists at most one index t such that $v \in Q_t$.
- **(I2)** Let $Q_x = (v_x, j_x) \in \mathcal{Q}$ and $Q_y = (v_y, j_y) \in \mathcal{Q}$ with $x < y$. Then $j_x \leq j_y$.
- **(I3)** Let $Q_x = (v, j) \in \mathcal{Q}$. Then there exists a channel $c \in \mathcal{U}_j$ that can be used by v.
- **(I4)** Let $Q_x = (v, j) \in \mathcal{Q}$. Then for every $w \in V_i \cap V_{=1}$ with $w \prec_i v$, there exists an index $x' < x$ such that $w \in Q_{x'}$.
- **(I5)** Let $Q_x = (v, j) \in \mathcal{Q}$ and let $(v, r, R) \in \mathcal{U}_j$ be the (unique, by Claim 13) channel usable by v in $\mathcal{U}_j$. Then for every $w \in V_i \setminus V_{=1}$ that is not a transition vertex in r and where $v \prec_i w$, w is to the right of r or on the right boundary of r. Symmetrically, for every $w \in V_i \setminus V_{=1}$ that is not a transition vertex in r and where $w \prec_i v$, w is to the left of r or on the left boundary of r.

Let $\mathcal{Q} = (Q_1, Q_2, \ldots, Q_q)$ be an insertion sequence for i, s, and $\mathcal{U}$. We say a leaf $v \in V_i \cap V_{=1}$ is *choosable* with regard to $\mathcal{Q}$ if (i) there exists an index j, such that $(Q_1, Q_2, \ldots, Q_q, (v, j))$ is an insertion sequence for i, s, and $\mathcal{U}$ as well and (ii) there exists no pair v', j', with $v' \in V_i \cap V_{=1}$ and $j' < j$ such that $\mathcal{Q} = (Q_1, Q_2, \ldots, Q_q, (v', j'))$ is an insertion sequence.

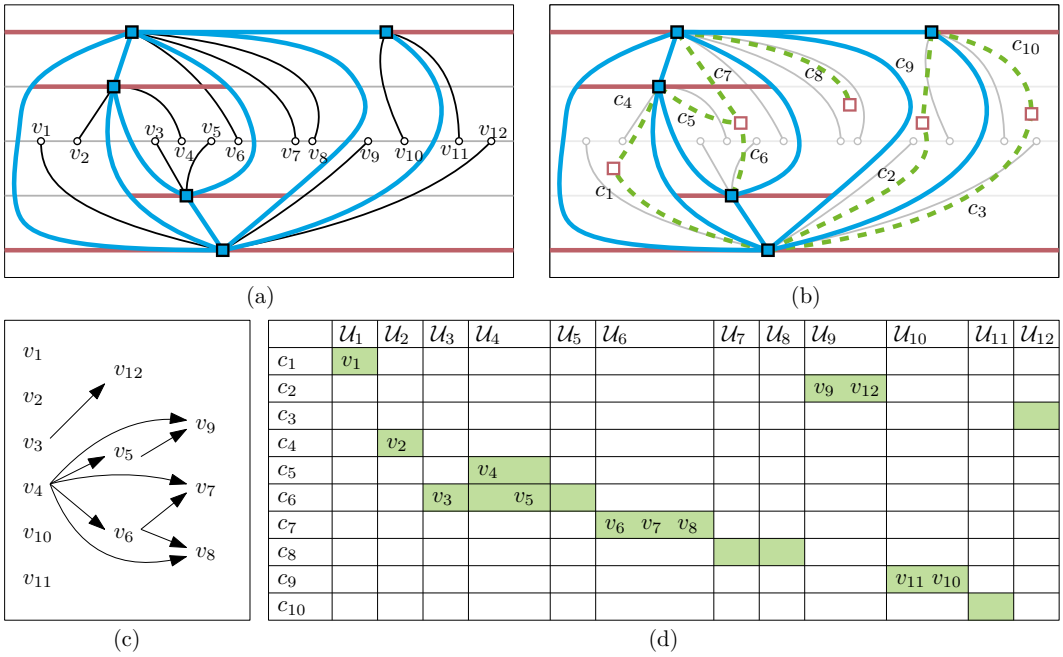


Figure 4 Like in all figures, filled square vertices belong to a vertex cover C of the depicted graph. (a) The drawing Λ^* together with its cell decomposition. (b) The dashed (green) "edges" represent the used channels $c_1, \ldots, c_{10}$ of Λ^* leading from vertices of C to cells in R represented by unfilled (red) squares. Figure (c) shows the constraints between the vertices of the third level depicted in (a) and Figure (d) illustrates the insertion sequences $\mathcal{Q} = ((v_1, 1), (v_2, 2), (v_3, 3), (v_4, 4), (v_5, 4), (v_6, 6), (v_7, 6), (v_8, 6), (v_9, 9), (v_{12}, 9), (v_{11}, 10), (v_{10}, 10))$ for this level and the depicted traversal sequence $\mathcal{U} = (\mathcal{U}_1, \ldots, \mathcal{U}_{12})$. Note that the order of the vertices in $\mathcal{Q}$ is different from the one in Λ^*.

▷ Claim 14 (⋆). Let $\mathcal{U} = (\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_m)$ be a traversal sequence that is compatible with Λ^* and let $\prec^{\mathrm{e}}$ be a compatible edge ordering for the restriction of Λ^* to its nonhorizontal edges, for which Property (C2) is fulfilled for $\mathcal{U}$. Further, let $i \in [h]$ and let s be the valid ear orientation of level i that is used in Λ^*. For every $q \in \{0, 1, \ldots, |V_i \cap V_{=1}|\}$, there exists an insertion sequence $\mathcal{Q}_q = (Q_1, Q_2, \ldots, Q_q)$ for i, s, and $\mathcal{U}$ such that the following two properties are satisfied (for an example, see Figure 4):

Interval property: For every vertex $v \in V_i \cap V_{=1}$ there exists at most one nonempty maximal interval $[a, b]$ where $0 \leq a \leq b \leq q$ such that v is choosable with regard to $\mathcal{Q}_j$ if and only if $a \leq j \leq b$. If such an interval $[a, b]$ exists, then $b = q$ or $v \in Q_{b+1}$. Conversely, if v occurs in some entry of $\mathcal{Q}_q$, then the interval exists.

Dominance property: Let $\prec_i^l$ be the restriction of $\prec^{\mathrm{e}}$ to edges incident to leaves on level i. Further, let $Q_\ell = (v_\ell, j_\ell) \in \mathcal{Q}_q$, let $v_{\ell'} \in V_i \cup V_{=1}$, and let e_ℓ ($e_{\ell'}$) be the edge incident to v_ℓ ($v_{\ell'}$). Then, if $e_\ell \prec_i^l e_{\ell'}$ or $v_\ell = v_{\ell'}$, we have $j_\ell \leq j'$, where j' is the maximum index such that the channel c' used by $v_{\ell'}$ (in Λ^*) is in $\mathcal{U}_{j'}$.

Moreover, $\mathcal{Q}_k$ is a prefix of $\mathcal{Q}_{k+1}$ for all $0 \leq k \leq |V_i \cap V_{=1}| - 1$. Finally, there is an algorithm that, given $\mathcal{G}$, C, $\Lambda^{\mathrm{t}}_{\mathrm{core}}$, $\mathcal{U}$, i and s, computes $\mathcal{Q}_{|V_i \cap V_{=1}|}$ in polynomial time.

Computing the drawing. When given a traversal sequence that is compatible with Λ^*, we can utilize Claim 14 to obtain a valid ear orientation together with an insertion sequence for a given level by simply trying to apply the algorithm corresponding to Claim 14 for all valid ear orientations of that level. We do so for each level and then use the gathered information to construct the desired drawing by means of the following claim. We remark that when the algorithm corresponding to Claim 14 successfully terminates, it is guaranteed to return an insertion sequence for the given valid ear orientation. It might output an insertion sequence even if the given valid ear orientation is not the one used in Λ^*. However, this does not invalidate our strategy as the following claim does *not* require that the given ear orientations are the ones used in Λ^*.

▷ Claim 15 ($\star$). There is an algorithm that, given $\mathcal{G}$, C, Λ_{core}, $\Lambda^{\text{t}}_{\text{core}}$, a traversal sequence $\mathcal{U} = (\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_m)$ that is compatible with Λ^*, and, for each level $i \in [h]$, a valid ear orientation s^i, as well as an insertion sequence $\mathcal{Q}^i = (Q^i_1, Q^i_2, \ldots, Q^i_{q^i})$ for i, s^i, and $\mathcal{U}$ such that $q^i = |V_i \cap V_{=1}|$ (that is, $\mathcal{Q}^i$ contains all leaves of level i), computes an extension of Λ_{core} whose restriction to G is a constrained level planar drawing of $\mathcal{G}$ in polynomial time.

Wrap-up. In the beginning of (and throughout the) proof of Lemma 10, we have already sketched how the individual pieces of the proof fit together. We formally summarize our strategy in the proof of the following claim.

▷ Claim 16 ($\star$). There is an algorithm that, given $\mathcal{G}$, C, Λ_{core} and $\Lambda^{\text{t}}_{\text{core}}$, extends Λ_{core} to a drawing Γ whose restriction to G is a constrained level planar drawing of $\mathcal{G}$ in $2^{\mathcal{O}(k^2 \log k)} n^{\mathcal{O}(1)}$ time.

This concludes the proof of Lemma 10. ◀

Summary. In the beginning of Section 4, we have already sketched how Lemmas 4 and 8–10 can be combined to obtain Theorem 1. We formally summarize:

▶ **Theorem 17** ($\star$). *There is an algorithm that, given a constrained level graph $\mathcal{G} = (G = (V, E), \gamma, (\prec_i)_i)$ and a vertex cover C of G, either constructs a constrained level planar drawing of $\mathcal{G}$ or correctly reports that such a drawing does not exist in time $2^{\mathcal{O}(k^2 \log k)} \cdot n^{\mathcal{O}(1)}$, where $n = |V|$ and $k = |C|$.*

Given that a smallest vertex cover can be obtained in FPT-time with respect to its size [10], our main result (Theorem 1) follows from Theorem 17.

5 Discussion

We have shown that CLP is FPT when parameterized by the vertex cover number. A speed-up to polynomial time or a generalization to the smaller graph parameters (in particular, tree-depth, path-width, tree-width, and feedback vertex set number) is not possible, even if restricting to OLP or proper CLP instances.

Recall from Section 1 that in the Level Planarity variant Partial Level Planarity (PLP), a given level planar drawing of a subgraph of the input graph $\mathcal{G}$ has to be extended to a full drawing of $\mathcal{G}$, which can be seen as a generalization of OLP and, in the proper case, a specialization of CLP. An instance of PLP can always be turned into an equivalent proper instance (and, thus, a CLP instance) by subdividing each edge on each level it passes through. However, in general this operation will (dramatically) increase the vertex cover number of the instance. Hence, our techniques cannot (directly) be applied. It thus is an interesting problem to study the parameterized complexity of PLP with respect to the vertex cover number.

References

1 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Ignaz Rutter. Beyond level planarity. In Yifan Hu and Martin Nöllenburg, editors, *Graph Drawing and Network Visualization - 24th International Symposium, GD 2016, Athens, Greece, September 19-21, 2016, Revised Selected Papers*, volume 9801 of *Lecture Notes in Computer Science*, pages 482–495. Springer, 2016. doi:10.1007/978-3-319-50106-2_37.

2 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, and Vincenzo Roselli. The importance of being proper: (in clustered-level planarity and t-level planarity). *Theor. Comput. Sci.*, 571:1–9, 2015. doi:10.1016/j.tcs.2014.12.019.

3 Andrei Asinowski, Tillmann Miltzow, and Günter Rote. Quasi-parallel segments and characterization of unique bichromatic matchings. *Journal of Computational Geometry*, 6:185–219, 2015. doi:10.20382/jocg.v6i1a8.

4 Christian Bachmaier, Franz-Josef Brandenburg, and Michael Forster. Radial level planarity testing and embedding in linear time. *J. Graph Algorithms Appl.*, 9(1):53–97, 2005. URL: http://jgaa.info/accepted/2005/BachmaierBrandenburgForster2005.9.1.pdf, doi:10.7155/JGAA.00100.

5 Christian Bachmaier and Wolfgang Brunner. Linear time planarity testing and embedding of strongly connected cyclic level graphs. In Dan Halperin and Kurt Mehlhorn, editors, *Algorithms - ESA 2008, 16th Annual European Symposium, Karlsruhe, Germany, September 15-17, 2008. Proceedings*, volume 5193 of *Lecture Notes in Computer Science*, pages 136–147. Springer, 2008. doi:10.1007/978-3-540-87744-8_12.

6 Michael J. Bannister, William E. Devanny, Vida Dujmović, David Eppstein, and David R. Wood. Track layouts, layered path decompositions, and leveled planarity. *Algorithmica*, 81(4):1561–1583, 2019. doi:10.1007/S00453-018-0487-5.

7 Václav Blažej, Boris Klemz, Felix Klesen, Marie Diana Sieper, Alexander Wolff, and Johannes Zink. Constrained and ordered level planarity parameterized by the number of levels. In Wolfgang Mulzer and Jeff M. Phillips, editors, *40th International Symposium on Computational Geometry, SoCG 2024, June 11-14, 2024, Athens, Greece*, volume 293 of *LIPIcs*, pages 20:1–20:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.SOCG.2024.20.

8 Guido Brückner and Ignaz Rutter. Partial and constrained level planarity. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2000–2011. SIAM, 2017. doi:10.1137/1.9781611974782.130.

9 Guido Brückner and Ignaz Rutter. An spqr-tree-like embedding representation for level planarity. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation, ISAAC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 181 of *LIPIcs*, pages 8:1–8:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ISAAC.2020.8.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

11 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. URL: https://www.worldcat.org/oclc/227584184.

12 N. G. de Bruijn. Problems 17 and 18. *Nieuw Archief voor Wiskunde*, 2:67, 1954. Answers in *Wiskundige Opgaven met de Oplossingen* 20:19–20, 1955.

13 Giuseppe Di Battista and Enrico Nardelli. Hierarchies and planarity theory. *IEEE Trans. Systems, Man, and Cybernetics*, 18(6):1035–1046, 1988. doi:10.1109/21.23105.

14 Vida Dujmović, Michael R. Fellows, Matthew Kitching, Giuseppe Liotta, Catherine McCartin, Naomi Nishimura, Prabhakar Ragde, Frances A. Rosamond, Sue Whitesides, and David R.

Wood. On the parameterized complexity of layered graph drawing. *Algorithmica*, 52(2):267–292, 2008. doi:10.1007/S00453-007-9151-1.

15 Michael Forster and Christian Bachmaier. Clustered level planarity. In Peter van Emde Boas, Jaroslav Pokorný, Mária Bieliková, and Julius Stuller, editors, *SOFSEM 2004: Theory and Practice of Computer Science, 30th Conference on Current Trends in Theory and Practice of Computer Science, Merin, Czech Republic, January 24-30, 2004*, volume 2932 of *Lecture Notes in Computer Science*, pages 218–228. Springer, 2004. doi:10.1007/978-3-540-24618-3_18.

16 Radoslav Fulek, Michael J Pelsmajer, Marcus Schaefer, and Daniel Štefankovič. Hanani–Tutte, monotone drawings, and level-planarity. In *Thirty Essays on Geometric Graph Theory*, pages 263–287. Springer, 2013. doi:10.1007/978-1-4614-0110-0_14.

17 Michael R Garey and David S Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

18 Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.*, 31(2):601–625, 2001. doi:10.1137/S0097539794277123.

19 Leonidas J. Guibas and F. Francis. Yao. On translating a set of rectangles. In *Proc. 12th Annual ACM Symposium Theory of Computing (STOC 1980)*, pages 154–160, 1980. doi:10.1145/800141.804663.

20 Leonidas J. Guibas and F. Francis. Yao. On translating a set of rectangles. In F. P. Preparata, editor, *Computational Geometry*, volume 1 of *Advances in Computing Research*, pages 61–77. JAI Press, Greenwich, Conn., 1983.

21 Lenwood S. Heath and Sriram V. Pemmaraju. Recognizing leveled-planar dags in linear time. In Franz-Josef Brandenburg, editor, *Graph Drawing, Symposium on Graph Drawing, GD '95, Passau, Germany, September 20-22, 1995, Proceedings*, volume 1027 of *Lecture Notes in Computer Science*, pages 300–311. Springer, 1995. doi:10.1007/BFb0021813.

22 Lenwood S. Heath and Arnold L. Rosenberg. Laying out graphs using queues. *SIAM J. Comput.*, 21(5):927–958, 1992. doi:10.1137/0221055.

23 Seok-Hee Hong and Hiroshi Nagamochi. Convex drawings of hierarchical planar graphs and clustered planar graphs. *J. Discrete Algorithms*, 8(3):282–295, 2010. doi:10.1016/j.jda.2009.05.003.

24 Michael Jünger, Sebastian Leipert, and Petra Mutzel. Pitfalls of using PQ-trees in automatic graph drawing. In Giuseppe Di Battista, editor, *Graph Drawing, 5th International Symposium, GD '97, Rome, Italy, September 18-20, 1997, Proceedings*, volume 1353 of *Lecture Notes in Computer Science*, pages 193–204. Springer, 1997. doi:10.1007/3-540-63938-1_62.

25 Michael Jünger, Sebastian Leipert, and Petra Mutzel. Level planarity testing in linear time. In Sue Whitesides, editor, *Graph Drawing, 6th International Symposium, GD'98, Montréal, Canada, August 1998, Proceedings*, volume 1547 of *Lecture Notes in Computer Science*, pages 224–237. Springer, 1998. doi:10.1007/3-540-37623-2_17.

26 Boris Klemz. Convex drawings of hierarchical graphs in linear time, with applications to planar graph morphing. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *Proc. 29th Ann. Europ. Symp. Algorithms (ESA'21)*, volume 204 of *LIPIcs*, pages 57:1–57:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.57.

27 Boris Klemz and Günter Rote. Ordered level planarity and its relationship to geodesic planarity, bi-monotonicity, and variations of level planarity. *ACM Trans. Algorithms*, 15(4):53:1–53:25, 2019. Conference version in Proc. GD'17. doi:10.1145/3359587.

28 Boris Klemz and Marie Diana Sieper. Constrained level planarity is FPT with respect to the vertex cover number. arXiv report, 2024. arXiv:2404.16723.

29 Ignaz Rutter. Personal communication, 2022.

30 Andreas Wotzlaw, Ewald Speckenmeyer, and Stefan Porschen. Generalized k-ary tanglegrams on level graphs: A satisfiability-based approach and its evaluation. *Discrete Applied Mathematics*, 160(16-17):2349–2363, 2012. doi:10.1016/j.dam.2012.05.028.

Subquadratic Submodular Maximization with a General Matroid Constraint

Yusuke Kobayashi
Research Institute for Mathematical Sciences, Kyoto University, Japan

Tatsuya Terao
Research Institute for Mathematical Sciences, Kyoto University, Japan

Abstract

We consider fast algorithms for monotone submodular maximization with a general matroid constraint. We present a randomized $(1-1/e-\epsilon)$-approximation algorithm that requires $\tilde{O}_\epsilon(\sqrt{r}n)$ independence oracle and value oracle queries, where n is the number of elements in the matroid and $r \leq n$ is the rank of the matroid. This improves upon the previously best algorithm by Buchbinder-Feldman-Schwartz [Mathematics of Operations Research 2017] that requires $\tilde{O}_\epsilon(r^2 + \sqrt{r}n)$ queries.

Our algorithm is based on continuous relaxation, as with other submodular maximization algorithms in the literature. To achieve subquadratic query complexity, we develop a new rounding algorithm, which is our main technical contribution. The rounding algorithm takes as input a point represented as a convex combination of t bases of a matroid and rounds it to an integral solution. Our rounding algorithm requires $\tilde{O}(r^{3/2}t)$ independence oracle queries, while the previously best rounding algorithm by Chekuri-Vondrák-Zenklusen [FOCS 2010] requires $O(r^2t)$ independence oracle queries. A key idea in our rounding algorithm is to use a directed cycle of arbitrary length in an auxiliary graph, while the algorithm of Chekuri-Vondrák-Zenklusen focused on directed cycles of length two.

2012 ACM Subject Classification Theory of computation → Algorithm design techniques; Theory of computation → Submodular optimization and polymatroids

Keywords and phrases submodular maximization, matroid constraint, approximation algorithm, rounding algorithm, query complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.100

Category Track A: Algorithms, Complexity and Games

Funding This work was partially supported by the joint project of Kyoto University and Toyota Motor Corporation, titled "Advanced Mathematical Science for Mobility Society", by JST ERATO Grant Number JPMJER2310, and by JSPS KAKENHI Grant Numbers JP20K11692, JP22H05001, JP24KJ1494, and JP24K02901.

Acknowledgements The authors thank the three anonymous reviewers for their valuable comments.

1 Introduction

1.1 Submodular Maximization

Submodular maximization is a fundamental and well-studied problem in theoretical computer science and combinatorial optimization. This is because a number of important problems can be regarded as special cases of submodular maximization, including maximum coverage, generalized assignment, and facility location. Furthermore, submodular maximization has many practical applications in machine learning, economics, and many other areas. In the submodular maximization problem, the input consists of a (monotone) submodular set function $f\colon 2^V \to \mathbb{R}_+$ and a feasible region $\mathcal{F} \subseteq 2^V$ specified by some constraints, and the aim is to find a set $S \in \mathcal{F}$ maximizing $f(S)$.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 100; pp. 100:1–100:19


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The study of submodular maximization was initiated by a seminal work of Fisher, Nemhauser, and Wolsey in the 1970's [26,34,35]. They showed that, for monotone submodular maximization, the greedy algorithm achieves $(1 - 1/e)$-approximation for a cardinality constraint and $\frac{1}{2}$-approximation for a matroid constraint. The advantage of their algorithm is that it is very simple and fast, indeed, it runs in quadratic time. It is known that unless P = NP, for any $\varepsilon > 0$, there is no $(1 - 1/e + \varepsilon)$-approximation algorithm for the maximum coverage problem [24], which is a special case of monotone submodular maximization with a cardinality constraint or a matroid constraint. Thus, the factor $1 - 1/e$ is optimal for a cardinality constraint.

To obtain an optimal $(1 - 1/e)$-approximation algorithm for a matroid constraint, Calinescu-Chekuri-Pál-Vondrák [13] developed a framework based on continuous optimization and rounding technique. In their algorithm, they first solve the continuous optimization problem of maximizing the multilinear extension F of f, a natural continuous extension of f. By using a continuous greedy algorithm, they obtain a $(1 - 1/e)$-approximation solution for the continuous optimization problem. In order to round the obtained fractional solution to an integral one, they use a variant of the pipage rounding technique of Ageev-Sviridenko [1]. Consequently, their algorithm achieves the optimal $(1 - 1/e)$-approximation. Note that, although their algorithm runs in polynomial time, its running time is very high.

Since submodular maximization has a number of applications, providing efficient approximation algorithms is a fundamental task both in theory and in practice. Thus, it has received considerable attention to develop fast submodular maximization algorithms that achieve an approximation close to the optimal factor, typically with an approximation factor of $1 - 1/e - \varepsilon$ for any $\varepsilon > 0$.

In the submodular maximization problem with a general matroid constraint, it is standard to suppose that the objective function f is given as a value oracle, and the feasible region $\mathcal{F} \subseteq 2^V$ is given as an independence oracle of a matroid. In such a case, the efficiency of an algorithm is usually measured by the number of value and independence oracle queries used in it.

Badanidiyuru-Vondrák [3] presented a fast algorithm that achieves an almost optimal approximation factor $1-1/e-\varepsilon$, for any $\varepsilon > 0$, for a matroid constraint. Their algorithm uses $O\left(\frac{rn}{\varepsilon^4} \log^2\left(\frac{n}{\varepsilon}\right)\right)$ value oracle queries and $O\left(\frac{n}{\varepsilon^2} \log\left(\frac{n}{\varepsilon}\right) + \frac{r^2}{\varepsilon}\right)$ independence oracle queries, where n is the number of elements in the matroid and r is the rank of the matroid. To achieve this query complexity, they developed a fast implementation of the continuous greedy algorithm that uses $\tilde{O}_\varepsilon(rn)$ value oracle queries and $\tilde{O}_\varepsilon(n)$ independence oracle queries.[1] The output of their continuous greedy algorithm is a fractional solution represented as a convex combination of $1/\varepsilon$ bases. Then, they apply the swap rounding algorithm of Chekuri-Vondrák-Zenklusen [18] to round the obtained fractional solution to an integral solution, which requires $O(r^2/\varepsilon)$ independence oracle queries.

Buchbinder-Feldman-Schwartz [12] presented a $(1-1/e-\varepsilon)$-approximation algorithm that has a trade-off between the number of value oracle queries and the number of independence oracle queries used in the algorithm. In their algorithm, they combine a variant of the residual random greedy algorithm of Buchbinder-Feldman-Naor-Schwartz [11] and the fast continuous greedy algorithm of Badanidiyuru-Vondrák described above. Then, for a parameter $\lambda \in [1, r]$, their algorithm uses $\tilde{O}_\varepsilon(r\lambda + \frac{rn}{\lambda})$ value oracle queries and $\tilde{O}_\varepsilon(\lambda n + r^2)$ independence oracle queries. We note that the $\tilde{O}_\varepsilon(r^2)$ term in the independence query complexity is due to

[1] The $\tilde{O}_\varepsilon$ notation hides polylogarithmic factors in n and polynomial factors in ε^{-1}.

the rounding algorithm in the same way as the algorithm of Badanidiyuru-Vondrák. If we evaluate the algorithm by the total number of queries regardless of their types, then the query complexity is minimized when $\lambda = \Theta(\sqrt{r})$. In this case, their algorithm uses $\tilde{O}_\varepsilon(r^2 + \sqrt{r}n)$ value and independence oracle queries. This query complexity is better than that of Badanidiyuru-Vondrák [3] when $r = o(n)$, but a quadratic number of queries is still required when r is large.

Recently, for several important classes of matroids, faster algorithms for monotone submodular maximization with a matroid constraint have been investigated. Ene-Nguyễn [22] presented a $(1-1/e-\varepsilon)$-approximation algorithm for graphic matroid and partition matroid constraints in time nearly-linear in the size of their representation. Henzinger-Liu-Vondrák-Zheng [27] presented a $(1-1/e-\varepsilon)$-approximation algorithm for laminar matroid and transversal matroid constraints in nearly-linear time. A key ingredient in these algorithms is a fast dynamic data structure for maintaining an (approximate) maximum weight basis of the matroid.

1.2 Our Results

This paper focuses on the monotone submodular maximization problem with a general matroid constraint. In the problem, this input consists of a monotone submodular set function $f\colon 2^V \to \mathbb{R}_+$ given as a value oracle, and a matroid $\mathcal{M} = (V, \mathcal{I})$ given as an independence oracle. The objective is to find an independent set $S \in \mathcal{I}$ that maximizes $f(S)$. For $\alpha \in [0,1]$, a randomized algorithm is said to be an *α-approximation algorithm* if it returns a solution $S \in \mathcal{I}$ with $\mathbb{E}[f(S)] \geq \alpha \cdot \max\{f(T) \mid T \in \mathcal{I}\}$. A randomized algorithm is often called simply an algorithm throughout the paper. Our main result is to give a first $(1-1/e-\varepsilon)$-approximation algorithm for this problem that requires a subquadratic number of queries. Recall that $n = |V|$ and r is the rank of $\mathcal{M}$.

▶ **Theorem 1.** *For any $\varepsilon > 0$, there is a randomized algorithm that achieves $(1-1/e-\varepsilon)$-approximation for maximizing a monotone submodular function subject to a matroid constraint and uses $O(\sqrt{r}n \operatorname{poly}(1/\varepsilon, \log n))$ value and independence oracle queries.*

It is worth mentioning that, for the case of $r = \Theta(n)$, our algorithm uses $\tilde{O}_\varepsilon(n^{3/2})$ oracle queries, whereas the algorithm of Buchbinder-Feldman-Schwartz [12] uses $\tilde{O}_\varepsilon(n^2)$ oracle queries.

Our algorithm is based on continuous relaxation and rounding technique in the same way as previous algorithms [3, 12, 13]. In this framework, currently, the bottleneck of the query complexity comes from the rounding algorithm. Indeed, the swap rounding algorithm by Chekuri-Vondrák-Zenklusen [18] requires $O(r^2 t)$ independence oracle queries if the input point is represented as a convex combination of t bases of the matroid. Then, this rounding algorithm requires a quadratic number of independence oracle queries even when t is small. Therefore, in order to break the quadratic-independence-query barrier in this framework, it is necessary to devise a faster rounding algorithm.

The key technical contribution of this paper is to develop a new rounding algorithm that uses $o(r^2 t)$ independence oracle queries.

▶ **Theorem 2.** *For any $\varepsilon > 0$, there is a randomized algorithm satisfying the following conditions:*

- *the input consists of a matroid $\mathcal{M} = (V, \mathcal{I})$ given as an independence oracle and a point x in the base polytope of $\mathcal{M}$ that is represented as a convex combination of t bases,*
- *the output is a basis S of $\mathcal{M}$ such that $\mathbb{E}[f(S)] \geq (1-\varepsilon)F(x)$ for any submodular function $f\colon 2^V \to \mathbb{R}$ and its multilinear extension F, and*
- *it uses $O(r^{3/2} t \log^{3/2}(\frac{rt}{\varepsilon}))$ independence oracle queries.*

By combining this theorem with the submodular maximization algorithm by Buchbinder-Feldman-Schwartz [12], we obtain Theorem 1; see Section 3 for details.

We also show that if the matroid is given as a rank oracle instead of an independence oracle, then we obtain a $(1 - 1/e - \varepsilon)$-approximation algorithm using $\tilde{O}_\varepsilon(n + r^{3/2})$ value and rank oracle queries.

▶ **Theorem 3.** *For any $\varepsilon > 0$, there is a randomized algorithm that achieves $(1 - 1/e - \varepsilon)$-approximation for maximizing a monotone submodular function subject to a matroid constraint and uses $O((n + r^{3/2})\ \mathrm{poly}(1/\varepsilon, \log n))$ value and rank oracle queries.*

1.3 Overview of Our Rounding Algorithm

In this subsection, we give a technical overview of our new rounding algorithm. Since our rounding algorithm is based on that of Chekuri-Vondrák-Zenklusen [18], we first review their algorithm and then explain the key ideas behind ours.

Swap Rounding Algorithm of Chekuri-Vondrák-Zenklusen. The rounding algorithm by Chekuri-Vondrák-Zenklusen is called the swap rounding algorithm. Their algorithm takes as input a point x represented as a convex combination of t bases of $\mathcal{M}$ and returns an integral solution S such that $\mathbb{E}[f(S)] \geq F(x)$ for any submodular function f and its multilinear extension F. In each phase of the algorithm, we pick up two bases in the representation of x and merge them into a single basis. By applying this procedure $t - 1$ times, we obtain a single basis of $\mathcal{M}$.

In order to merge two bases, say B_1 and B_2, their swap rounding algorithm uses a *strongly exchangeable pair* of elements, that is, a pair of elements $u \in B_1 \setminus B_2$ and $v \in B_2 \setminus B_1$ such that $B_1 + v - u \in \mathcal{I}$ and $B_2 + u - v \in \mathcal{I}$. Since we can find a strongly exchangeable pair using $O(r)$ independence oracle queries and we need to find such a pair $O(rt)$ times in the algorithm, the total number of queries is $O(r^2 t)$. It is still not clear whether we can develop an algorithm for finding a strongly exchangeable pair using $o(r)$ independence oracle queries, and hence their algorithm is now stuck at $\Omega(r^2 t)$ independence oracle queries.

See Section 4 for details of the swap rounding algorithm of Chekuri-Vondrák-Zenklusen.

Our Faster Rounding Algorithm. We develop a new rounding algorithm that requires $\tilde{O}(r^{3/2} t)$ independence oracle queries with high probability. Our rounding algorithm is based on that of Chekuri-Vondrák-Zenklusen in a sense that we update bases by swapping a pair of elements $O(rt)$ times. Therefore, in each step of our algorithm, we need to update some basis by using only $\tilde{O}(\sqrt{r})$ independence oracle queries. To achieve this, we need substantially new ideas.

First, we introduce a digraph that represents exchangeability of the elements in the matroid (see Definition 8), and provide a new interpretation of the swap rounding algorithm of Chekuri-Vondrák-Zenklusen using this auxiliary graph. Indeed, each step of their algorithm can be seen as an update using a directed cycle of length two in the auxiliary graph. This interpretation motivates us to focus on a directed cycle of arbitrary length in the auxiliary graph instead of a directed cycle of length two. By extending the argument of Chekuri-Vondrák-Zenklusen, we show that we can appropriately update bases using a directed cycle of arbitrary length in the auxiliary graph.

Second, we show that we can find a directed cycle in the auxiliary graph using $o(r)$ independence oracle queries with high probability, which is the most technical part in our argument. To achieve this, we combine sampling technique and binary search technique.

In our algorithm for finding a directed cycle in the auxiliary graph, we first sample $\tilde{O}(\sqrt{r})$ vertices, and define D' as the subgraph induced by the sampled vertex set. If every vertex in D' has an incoming edge, then we can easily find a directed cycle in D' by traversing such directed edges in the opposite direction. Otherwise, by using a vertex with no incoming edge, we find a directed cycle of length two using $\tilde{O}(\sqrt{r})$ independence oracle queries with high probability. We can check whether each vertex in D' has an incoming edge or not using $\tilde{O}(1)$ independence oracle queries with the aid of the binary search technique proposed by Nguyễn [36] and Chakrabarty-Lee-Sidford-Singla-Wong [14]; see Lemma 4 for details. Note that this technique was used in recent studies on fast matroid intersection [8,10,14,36,39] and matroid partition [38] algorithms. Therefore, we obtain an algorithm that finds a directed cycle using $\tilde{O}(\sqrt{r})$ independence oracle queries with high probability.

In our rounding algorithm, we update bases using a directed cycle in the auxiliary graph repeatedly. Since we update bases $O(rt)$ times and each update requires $\tilde{O}(\sqrt{r})$ independence oracle queries, the total number of independence oracle queries is $\tilde{O}(r^{3/2}t)$ with high probability.

1.4 Related Work

We have mentioned several recent studies on fast submodular maximization with matroid constraints in Section 1.1. Other than these, there are a lot of studies on fast submodular maximization algorithms in the literature [2, 3, 15, 20, 25, 30, 32].

Badanidiyuru-Vondrák [3] developed a $(1 - 1/e - \varepsilon)$-approximation algorithm using $O(\frac{n}{\varepsilon}\log(\frac{n}{\varepsilon}))$ value oracle queries for the cardinality constraint. Mirzasoleiman-Badanidiyuru-Ashwinkumar-Karbasi-Vondrák-Krause [32] developed a $(1-1/e-\varepsilon)$-approximation algorithm using $O(n\log(1/\varepsilon))$ value oracle queries for the cardinality constraint. Ene-Nguyễn [20] developed a $(1 - 1/e - \varepsilon)$-approximation algorithm using $(1/\varepsilon)^{O(1/\varepsilon^4)}n\log^2 n$ value oracle queries for the knapsack constraint. Filmus-Ward [25] presented a combinatorial $(1 - 1/e)$-approximation algorithm for monotone submodular maximization with a matroid constraint, which uses $O(n^7r^2)$ oracle queries. They also obtain a $(1 - 1/e - O(\varepsilon))$-approximation algorithm that uses $O(\varepsilon^{-3}n^4r)$ value oracle queries and $O(\varepsilon^{-1}n^2r\log n)$ independence oracle queries.

Studies on fast submodular maximization algorithms have developed also in the direction of parallelized settings [4, 5, 16, 21, 23], distributed settings [7, 31], and dynamic settings [6, 19, 29, 33].

Chekuri-Quanrud-Torres [17] developed a fast swap rounding algorithm for graphic matroid constraints to obtain fast approximation algorithms for the Bounded Degree MST problem and the Crossing Spanning Tree problem.

1.5 Paper Organization

The remaining of this paper is organized as follows. In Section 2, we give some preliminaries. In Section 3, we show how to derive Theorem 1 from our fast rounding algorithm in Theorem 2. In Section 4, we describe the swap rounding algorithm by Chekuri-Vondrák-Zenklusen [18] in detail, because it is the basis of our rounding algorithm. In Section 5, we describe our fast rounding algorithm and prove Theorem 2, which is the main technical part of this paper. In Section 6, we discuss the rank oracle setting and prove Theorem 3.

2 Preliminaries

Basic Notation. Let $\mathbb{R}_+$ denote the set of non-negative real numbers. Throughout this paper, let V be a finite set and let n denote its cardinality. For a set $A \subseteq V$ and an element $v \in V$, we will often write $A+v := A \cup \{v\}$ and $A-v := A \setminus \{v\}$. For two sets $A, B \subseteq V$, their symmetric difference is denoted by $A \triangle B := (A \setminus B) \cup (B \setminus A)$. For $A \subseteq V$, the *characteristic vector* of A is defined as the vector $x \in \{0,1\}^V$ with $x_v = 1$ for $v \in A$ and $x_v = 0$ for $v \in V \setminus A$. We will denote by $\mathbf{1}_A$ the characteristic vector of A. For $v \in V$, we will write $\mathbf{1}_v := \mathbf{1}_{\{v\}}$.

Submodular Functions and Multilinear Extension. Let $f\colon 2^V \to \mathbb{R}_+$ be a set function on a finite ground set V of size n. The function is *submodular* if $f(A)+f(B) \geq f(A \cup B)+f(A \cap B)$ for any two subsets $A, B \subseteq V$. The function is *monotone* if $f(A) \leq f(B)$ for any subsets $A \subseteq B \subseteq V$. In this paper, we only consider monotone submodular functions.

For a function $f\colon 2^V \to \mathbb{R}_+$, we define its *multilinear extension* $F\colon [0,1]^V \to \mathbb{R}_+$ by

$$F(x) = \sum_{S \subseteq V} f(S) \prod_{v \in S} x_v \prod_{v \in V \setminus S} (1 - x_v)$$

for $x \in [0,1]^V$. Note that this value is equal to $\mathbb{E}[f(R(x))]$, where $R(x)$ is a random set that contains each element $v \in V$ independently with probability x_v. In particular, $F(\mathbf{1}_S) = f(S)$ for any $S \subseteq V$.

Matroids. A pair $\mathcal{M} = (V, \mathcal{I})$ of a finite set V and a non-empty set family $\mathcal{I} \subseteq 2^V$ is called a *matroid* if the following properties are satisfied.

(Downward closure property) if $S \in \mathcal{I}$ and $S' \subseteq S$, then $S' \in \mathcal{I}$.
(Augmentation property) if $S, S' \in \mathcal{I}$ and $|S'| < |S|$, then there exists $v \in S \setminus S'$ such that $S' + v \in \mathcal{I}$.

A set $S \subseteq V$ is called *independent* if $S \in \mathcal{I}$ and *dependent* otherwise. The *rank* of $\mathcal{M}$ is defined as the size of a largest independent set. In addition, for a subset $S \subseteq V$, the rank of S is defined as the size of a largest independent set contained in S. Inclusionwise maximal independent sets are called *bases*. Note that every basis has the same size. For an independent set $S \in \mathcal{I}$, let $\mathcal{M}/S = (V \setminus S, \mathcal{I}')$ be the matroid obtained by contracting S in $\mathcal{M}$, that is, $S' \in \mathcal{I}'$ if and only if $S' \cup S \in \mathcal{I}$.

Let $\mathcal{B}$ be the set of all bases of a matroid $\mathcal{M} = (V, \mathcal{I})$ and let $B, B' \in \mathcal{B}$ be two bases. It is well-known that, for any $u \in B \setminus B'$, there exists $v \in B' \setminus B$ such that $B - u + v \in \mathcal{B}$ and $B' - v + u \in \mathcal{B}$ (see e.g., [37, Theorem 39.12]). This property is called *strong basis exchange property* of matroids.

Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid whose rank function and basis family are denoted by $r_{\mathcal{M}}$ and $\mathcal{B}$, respectively. The *matroid polytope* $P(\mathcal{M})$ is defined as the convex hull of the characteristic vectors of all the independent sets of $\mathcal{M}$. The *matroid base polytope* $B(\mathcal{M})$ is defined as the convex hull of the characteristic vectors of all the bases of $\mathcal{M}$. It is well-known that $P(\mathcal{M})$ and $B(\mathcal{M})$ are described as follows (see e.g., [37, Section 40.2]):

$$P(\mathcal{M}) := \mathrm{conv}\{\mathbf{1}_I \mid I \in \mathcal{I}\} = \left\{ x \in \mathbb{R}_+^V \;\middle|\; \sum_{v \in S} x_v \leq r_{\mathcal{M}}(S) \text{ for any } S \subseteq V \right\},$$

$$B(\mathcal{M}) := \mathrm{conv}\{\mathbf{1}_B \mid B \in \mathcal{B}\} = \left\{ x \in P(\mathcal{M}) \;\middle|\; \sum_{v \in V} x_v = r_{\mathcal{M}}(V) \right\}.$$

Oracles. When we consider the submodular maximization problem, we assume that the submodular function f is given as a value oracle, which takes as input any subset $S \subseteq V$ and outputs $f(S)$. We also assume that we access a matroid $\mathcal{M}$ through an oracle. Given a subset $S \subseteq V$, an *independence oracle* outputs whether $S \in \mathcal{I}$ or not. Given a subset $S \subseteq V$, a *rank oracle* outputs the rank of S, i.e., the size of a largest independent set contained in S. Note that the rank oracle is more powerful than the independence oracle, since one query of the rank oracle can determine whether a given subset is independent or not.

Binary Search Technique. For a matroid $\mathcal{M} = (V, \mathcal{I})$, an independent set $S \in \mathcal{I}$, an element $u \in V \setminus S$, and $T \subseteq S$, we consider a procedure that finds an element $v \in T$ with $S + u - v \in \mathcal{I}$ if one exists. Chakrabarty et al. [14] and Nguyễn [36] independently proved that this procedure can be implemented efficiently by using the binary search technique in the independence oracle model. Their result is formally described as follows.

▶ **Lemma 4** ([14, 36]). *There is an algorithm* `FindExchangeElement` *which, given a matroid* $\mathcal{M} = (V, \mathcal{I})$*, an independent set* $S \in \mathcal{I}$*, an element* $u \in V \setminus S$*, and* $T \subseteq S$*, finds an element* $v \in T$ *such that* $S + u - v \in \mathcal{I}$ *or otherwise determines that no such element exists, and uses* $O(\log |T|)$ *independence oracle queries.*

3 Submodular Maxmization Algorithm (Proof of Theorem 1)

In this section, we give a proof of Theorem 1 by combining the algorithm of Buchbinder-Feldman-Schwartz [12] and our rounding algorithm in Theorem 2. Note that a proof of Theorem 2 is given in Section 5 later.

For monotone submodular maximization with a matroid constraint, Buchbinder-Feldman-Schwartz presented a $(1 - 1/e - \varepsilon)$-approximation algorithm that has a trade-off between the number of value oracle queries and the number of independence oracle queries used in the algorithm. The main part of their algorithm is to solve the continuous relaxation of the submodular maximization problem efficiently.

Let $\lambda \in [1, r]$ be a parameter that controls the trade-off. In their algorithm for solving the continuous relaxation problem, they first apply a variant of the residual random greedy algorithm of Buchbinder-Feldman-Naor-Schwartz [11]. This residual random greedy algorithm outputs $S \subseteq V$ and uses $\tilde{O}_\varepsilon(r\lambda + n)$ value oracle queries and $\tilde{O}_\varepsilon(\lambda n)$ independence oracle queries; see [12, Lemma 3.3]. Then they apply a variant of the fast continuous greedy algorithm of Badanidiyuru-Vondrák [3]. This continuous greedy algorithm outputs a point x' represented as a convex combination of $1/\varepsilon$ bases of $\mathcal{M}/S$ and uses $\tilde{O}_\varepsilon(\frac{rn}{\lambda})$ value oracle queries and $\tilde{O}_\varepsilon(n)$ independence oracle queries; see [12, Corollary 3.1]. Then $x = \mathbf{1}_S \vee x'$ is an approximate solution for the continuous relaxation problem, which can be represented as a convex combination of $1/\varepsilon$ bases of $\mathcal{M}$. Here, for vectors y and z, let $y \vee z$ denote the vector such that $(y \vee z)_i = \max\{y_i, z_i\}$ for all i.

Overall, Buchbinder-Feldman-Schwartz [12] presented an efficient algorithm for solving the continuous relaxation problem, which is formally stated as follows.

▶ **Theorem 5** (follows from [12, Corollary 3.1] and [12, Lemma 3.3]). *Given a non-negative monotone submodular function* $f\colon 2^V \to \mathbb{R}_+$*, a matroid* $\mathcal{M} = (V, \mathcal{I})$ *of rank* r*, and parameters* $\varepsilon > 0$ *and* $\lambda \in [1, r]$*, there is an algorithm satisfying the following conditions:*

- *the algorithm outputs a point* $x \in B(\mathcal{M})$ *represented as a convex combination of* $1/\varepsilon$ *bases such that* $\mathbb{E}[F(x)] \geq (1 - 1/e - \varepsilon) \cdot \max\{f(T) \mid T \in \mathcal{I}\}$ *holds, where* $F\colon [0,1]^V \to \mathbb{R}_+$ *is the multilinear extension of* f*,*

- *it uses* $O\left(r\lambda + \frac{rn}{\lambda\varepsilon^5}\log^2\left(\frac{n}{\varepsilon}\right)\right)$ *value oracle queries, and*
- *it uses* $O\left(\frac{\lambda n}{\varepsilon^2}\log\left(\frac{n}{\varepsilon}\right)\right)$ *independence oracle queries.*

Suppose that $x \in B(\mathcal{M})$ is a point as in Theorem 5. In the submodular maximization algorithm of Buchbinder-Feldman-Schwartz, they round x to an integral solution with the aid of the swap rounding algorithm of Chekuri-Vondrák-Zenklusen [18] using $O(r^2/\varepsilon)$ independence oracle queries. Therefore, their entire algorithm requires $\tilde{O}_\varepsilon(r\lambda + \frac{rn}{\lambda})$ value oracle queries and $\tilde{O}_\varepsilon(\lambda n + r^2)$ independence oracle queries.

▶ Remark 6. Theorem 5 is not explicitly stated in the paper by Buchbinder-Feldman-Schwartz [12], because they do not separately evaluate the query complexity for solving the continuous relaxation problem and for the rounding algorithm. Indeed, they just state that the entire algorithm requires $O(\frac{r^2}{\varepsilon} + \frac{\lambda n}{\varepsilon^2}\log\left(\frac{n}{\varepsilon}\right))$ independence oracle queries; see [12, Theorem 1.1]. The $O(\frac{r^2}{\varepsilon})$ term in this query complexity comes from [12, Corollary 3.1], which states that the continuous greedy algorithm together with the rounding algorithm requires $O(\frac{n}{\varepsilon^2}\log(\frac{n}{\varepsilon}) + \frac{r^2}{\varepsilon})$ independence oracle queries. This corollary is a direct consequence of [3, Claim 4.4], whose proof shows that the $O(\frac{r^2}{\varepsilon})$ term comes from the rounding algorithm, while the $O(\frac{n}{\varepsilon^2}\log(\frac{n}{\varepsilon}))$ term comes from the continuous greedy algorithm. Therefore, the $O(\frac{r^2}{\varepsilon})$ term is not needed to solve the continuous relaxation problem.

We now show that our submodular maximization algorithm with subquadratic query complexity is derived from Theorems 2 and 5.

Proof of Theorem 1. Let $\lambda = \Theta(\sqrt{r})$ and let $\varepsilon' = \varepsilon/2$. Note that we can compute r using $O(n)$ independence oracle queries by a greedy algorithm. We first run the algorithm in Theorem 5 with parameters λ and ε' to obtain a point $x \in B(\mathcal{M})$, in which we use $O(\sqrt{r}n\ \mathrm{poly}(1/\varepsilon, \log n))$ value and independence oracle queries. For the obtained point x, we apply our fast rounding algorithm in Theorem 2 with an error parameter ε'. Then, we obtain a basis S of $\mathcal{M}$ such that

$$\begin{aligned}\mathbb{E}[f(S)] &\geq (1-\varepsilon')\cdot\mathbb{E}[F(x)]\\ &\geq (1-\varepsilon')\cdot(1-1/e-\varepsilon')\cdot\max\{f(T)\mid T\in\mathcal{I}\}\\ &\geq (1-1/e-\varepsilon)\cdot\max\{f(T)\mid T\in\mathcal{I}\}.\end{aligned}$$

Since x is represented as a convex combination of $1/\varepsilon'$ bases of $\mathcal{M}$ by Theorem 5, our rounding algorithm requires $O(r^{3/2}\ \mathrm{poly}(1/\varepsilon, \log n))$ independence oracle queries by Theorem 2. Therefore, we obtain a $(1-1/e-\varepsilon)$-approximation algorithm that uses $O(\sqrt{r}n\ \mathrm{poly}(1/\varepsilon, \log n))$ value and independence oracle queries, which completes the proof. ◀

4 Swap Rounding Algorithm in Previous Work

In this section, we describe the swap rounding algorithm of Chekuri-Vondrák-Zenklusen [18] for a matroid base polytope, which we denote `SwapRound`. As described in Section 1.3, our new rounding algorithm is based on `SwapRound`. In `SwapRound`, we are given a point $x \in B(\mathcal{M})$ that is represented as a convex combination of the characteristic vectors of t bases of $\mathcal{M}$. The output is a single basis S of $\mathcal{M}$ such that $\mathbb{E}[f(S)] \geq F(x)$ for any submodular function $f\colon 2^V \to \mathbb{R}$ and its multilinear extension F. In each phase of `SwapRound`, we pick up two bases in the representation of x and merge them into a basis. By applying this procedure $t-1$ times, `SwapRound` finally returns a single basis of $\mathcal{M}$; see Algorithm 1.

The procedure for merging two bases is denoted by `MergeBases` (Algorithm 2). The input of `MergeBases` consists of two bases B_1 and B_2 together with their coefficients β_1 and β_2. In the procedure, until B_1 and B_2 coincide, we repeatedly update B_1 and B_2 so that $|B_1 \setminus B_2|$ decreases monotonically. In each update of B_1 and B_2, we need a *strongly exchangeable pair* of elements, that is, a pair of elements $u \in B_1 \setminus B_2$ and $v \in B_2 \setminus B_1$ such that $B_1 + v - u \in \mathcal{I}$ and $B_2 + u - v \in \mathcal{I}$. As described in `UpdateViaStrongBasisExchange` (Algorithm 3), for a strongly exchangeable pair u and v, we apply $B_1 \leftarrow B_1 + v - u$ with probability $\frac{\beta_2}{\beta_1+\beta_2}$ and apply $B_2 \leftarrow B_2 + u - v$ with the remaining probability. Note that, in `UpdateViaStrongBasisExchange`, B_1 and B_2 are updated to B_1' and B_2' so that $\mathbb{E}[\beta_1 \mathbf{1}_{B_1'} + \beta_2 \mathbf{1}_{B_2'}] = \beta_1 \mathbf{1}_{B_1} + \beta_2 \mathbf{1}_{B_2}$, which is a key property to show the validity of the algorithm.

The most time consuming part in `MergeBases` is to find a strongly exchangeable pair. By the strong basis exchange property of matroids, we can find such a pair of elements using $O(r)$ independence oracle queries in the following way: fix an element $u \in B_1 \setminus B_2$ arbitrarily and check the conditions for each element $v \in B_2 \setminus B_1$ one by one. Since we update the bases $|B_1 \setminus B_2| = O(r)$ times, `MergeBases` requires $O(r^2)$ independence oracle queries in total. Hence, `SwapRound` requires $O(r^2 t)$ independence oracle queries.

It is not clear whether we can develop an algorithm that finds a strongly exchangeable pair using $o(r)$ independence oracle queries. Therefore, their implementation of `SwapRound` is now stuck at $\Omega(r^2 t)$ independence oracle queries.

Algorithm 1 `SwapRound`$(x = \sum_{i=1}^{t} \beta_i \mathbf{1}_{B_i})$.

```
C_1 ← B_1
γ_1 ← β_1
for i = 1 to t − 1 do
    C_{i+1} ← MergeBases(γ_i, C_i, β_{i+1}, B_{i+1})
    γ_{i+1} ← γ_i + β_{i+1}
return C_t
```

Algorithm 2 `MergeBases`$(\beta_1, B_1, \beta_2, B_2)$.

```
while B_1 ≠ B_2 do
    Pick arbitrary u ∈ B_1 \ B_2
    Find v ∈ B_2 \ B_1 such that B_1 + v − u ∈ I and B_2 + u − v ∈ I
    UpdateViaStrongBasisExchange(β_1, B_1, β_2, B_2, v, u)
return B_1
```

5 Faster Rounding Algorithm

In this section, we present our fast rounding algorithm. We first show the following theorem, and then prove Theorem 2 using this theorem.

▶ **Theorem 7.** *There is a randomized algorithm satisfying the following conditions:*

- *the input consists of a matroid $\mathcal{M} = (V, \mathcal{I})$ given as an independence oracle and a point $x \in B(\mathcal{M})$ represented as a convex combination of t bases,*
- *the output is a basis S of $\mathcal{M}$ such that $\mathbb{E}[f(S)] \geq F(x)$ for any submodular function $f\colon 2^V \to \mathbb{R}$ and its multilinear extension F, and*
- *it uses $O(r^{3/2} t \log^{3/2}(rt))$ independence oracle queries with probability at least $1 - (rt)^{-1}$.*

Algorithm 3 `UpdateViaStrongBasisExchange`$(\beta_1, B_1, \beta_2, B_2, v, u)$.

Input: $\beta_1, \beta_2 \in \mathbb{R}_+$, two bases B_1, B_2, and elements $v \in B_2 \setminus B_1$ and $u \in B_1 \setminus B_2$ such that $B_1 + v - u \in \mathcal{I}$ and $B_2 + u - v \in \mathcal{I}$

Flip a coin with `Heads` probability $\dfrac{\beta_2}{\beta_1 + \beta_2}$

if *coin flipped* `Heads` **then**

  $B_1 \leftarrow B_1 + v - u$

else

  $B_2 \leftarrow B_2 + u - v$

To show this theorem, we propose an algorithm that merges the bases one by one in the same way as `SwapRound`. Our contribution is to improve `MergeBases`, that is, we give a faster algorithm for merging two bases into a single basis. The following auxiliary graph plays an important role in our algorithm.

▶ **Definition 8.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid, and let B_1, B_2 be two bases of $\mathcal{M}$. Then we define the bipartite directed graph $D_{\mathcal{M}}(B_1, B_2)$ whose vertex set and edge set are $B_1 \triangle B_2$ and $E_1(B_1, B_2) \cup E_2(B_1, B_2)$, respectively, where*

$$E_1(B_1, B_2) = \{(u, v) \mid u \in B_1 \setminus B_2, v \in B_2 \setminus B_1, B_1 + v - u \in \mathcal{I}\},$$
$$E_2(B_1, B_2) = \{(v, u) \mid u \in B_1 \setminus B_2, v \in B_2 \setminus B_1, B_2 + u - v \in \mathcal{I}\}.$$

In terms of this auxiliary graph, each step of `MergeBases` can be interpreted as follows: it finds a directed cycle of length two (or a bidirected edge) in $D_{\mathcal{M}}(B_1, B_2)$ and updates the bases B_1 and B_2 using this directed cycle as in `UpdateViaStrongBasisExchange`. Note that we use $O(r)$ independence oracle queries to find a directed cycle of length two.

A key idea in our algorithm is to focus on a directed cycle of arbitrary length in $D_{\mathcal{M}}(B_1, B_2)$ instead of a directed cycle of length two. More precisely, our contribution consists of the following two technical results.

1. We can find a directed cycle in $D_{\mathcal{M}}(B_1, B_2)$ using $o(r)$ independence oracle queries with high probability.
2. We can appropriately update the bases using a directed cycle of arbitrary length in $D_{\mathcal{M}}(B_1, B_2)$.

We discuss the first and second technical results in Sections 5.1 and 5.2, respectively. Then, we describe the entire algorithm and give proofs for Theorems 2 and 7 in Section 5.3.

5.1 Finding a Directed Cycle

The objective of this subsection is to show the following proposition, which states that we can find a directed cycle in $D_{\mathcal{M}}(B_1, B_2)$ using $o(r)$ independence oracle queries with high probability.

▶ **Proposition 9.** *Suppose we are given two bases B_1 and B_2 of a matroid $\mathcal{M}$ and an integer $t \geq 2$. Then, we can find a directed cycle in $D_{\mathcal{M}}(B_1, B_2)$ using $O(\sqrt{r}\log^{3/2}(rt))$ independence oracle queries with probability at least $1 - (rt)^{-2}$.*

To show this proposition, we first show that a directed cycle of length two can be found efficiently if we have an element whose indegree is small in $D_{\mathcal{M}}(B_1, B_2)$.

▶ **Lemma 10.** *Suppose we are given two bases B_1 and B_2 of a matroid $\mathcal{M}$, and an element $a \in B_1 \triangle B_2$ whose indegree is d in $D_{\mathcal{M}}(B_1, B_2)$. Then, we can find a directed cycle of length two in $D_{\mathcal{M}}(B_1, B_2)$ using $O(d \log r)$ independence oracle queries.*

Proof. By symmetry, it suffices to consider the case when $a \in B_1 \setminus B_2$.

We give an algorithm that finds an element $v \in B_2 \setminus B_1$ such that $B_1 + v - a \in \mathcal{I}$ and $B_2 + a - v \in \mathcal{I}$. In our algorithm, let $A \subseteq B_2 \setminus B_1$ denote the set of elements v such that we have already checked that $B_1 + v - a \notin \mathcal{I}$. We initialize $A = \emptyset$.

In each step of our algorithm, by applying Lemma 4 in which $u = a$, $S = B_2$, and $T = B_2 \setminus (B_1 \cup A)$, we can find an element $v \in T$ such that $B_2 + a - v \in \mathcal{I}$ if it exists. For such v, we check whether $B_1 + v - a \in \mathcal{I}$ holds or not. If $B_1 + v - a \in \mathcal{I}$ holds, then a and v induce a desired directed cycle. Otherwise, we add v to A, and repeat the procedure.

This algorithm finds a directed cycle correctly by the strong basis exchange property. Since we apply Lemma 4 at most d times and $|T| \leq r$, this algorithm uses $O(d \log r)$ independence oracle queries. ◀

We now describe our algorithm for finding a directed cycle in $D_{\mathcal{M}}(B_1, B_2)$. In our algorithm, we first sample $2\sqrt{r \log(rt)}$ elements from $B_1 \setminus B_2$ (resp. $B_2 \setminus B_1$) uniformly at random with replacement, where the base of the logarithm is e, and let L (resp. R) be the sampled vertex set, ignoring the multiplicity. Note that $1 \leq |L| \leq 2\sqrt{r \log(rt)}$ and $1 \leq |R| \leq 2\sqrt{r \log(rt)}$ as we ignore the multiplicity. Let D' be the subgraph of $D_{\mathcal{M}}(B_1, B_2)$ induced by $L \cup R$.

For each vertex u in D', we find a directed edge in D' that enters u or conclude that such a directed edge does not exist. This can be done by calling `FindExchangeElement` exactly once for each u. If every vertex in D' has an incoming edge, then we can easily find a directed cycle in D' by traversing such directed edges in the opposite direction. Otherwise, we pick up a vertex a in $L \cup R$ that has no incoming edge in D', and then apply Lemma 10 with this vertex a to find a directed cycle of length two.

Since the correctness of this algorithm is clear, it suffices to analyze the independence query complexity. We use the following lemma in our analysis.

▶ **Lemma 11.** *Let $u \in B_1 \triangle B_2$ be an element whose indegree in $D_{\mathcal{M}}(B_1, B_2)$ is at least $2\sqrt{r \log(rt)}$. Then, the probability that $D_{\mathcal{M}}(B_1, B_2)$ has no directed edge from $L \cup R$ to u is at most $(rt)^{-4}$.*

Proof. By symmetry, it suffices to consider the case when $u \in B_1 \setminus B_2$. Let $N = \{v \in B_2 \setminus B_1 \mid (v, u) \in E(B_1, B_2)\}$. Since R is obtained by sampling $2\sqrt{r \log(rt)}$ vertices from $B_2 \setminus B_1$ and $r \geq |B_2 \setminus B_1| \geq |N| \geq 2\sqrt{r \log(rt)}$, we have the following:

$$\begin{aligned}
\Pr\left[\{v \in R \mid (v, u) \in E(B_1, B_2)\} = \emptyset\right] &= \Pr\left[N \cap R = \emptyset\right] \\
&= \left(1 - \frac{|N|}{|B_2 \setminus B_1|}\right)^{2\sqrt{r \log(rt)}} \\
&\leq \left(1 - \frac{2\sqrt{r \log(rt)}}{r}\right)^{2\sqrt{r \log(rt)}} \\
&\leq \left(e^{-1}\right)^{4 \log(rt)} \\
&= (rt)^{-4},
\end{aligned}$$

which completes the proof. ◀

We are now ready to prove Proposition 9.

Proof of Proposition 9. We analyze the independence query complexity of the algorithm described above. First, since we call `FindExchangeElement` for each vertex $u \in L \cup R$ exactly once to find an incoming edge in D', the number of calls of `FindExchangeElement` is $|L \cup R| = O(\sqrt{r \log(rt)})$. Hence, by Lemma 4, the number of independence oracle queries used in this part is $O(\sqrt{r \log(rt)} \log r)$.

We next analyze the number of independence oracle queries when there exists a vertex $a \in L \cup R$ that has no incoming edge in D'.

We call a vertex $u \in L \cup R$ *bad* if D' has no directed edge entering u and $D_{\mathcal{M}}(B_1, B_2)$ has at least $2\sqrt{r \log(rt)}$ directed edges entering u. By Lemma 11, for each $u \in L \cup R$, the vertex u is bad with probability at most $(rt)^{-4}$. Thus, by taking the union bound over all vertices in $L \cup R$, we see that there exists a bad vertex in $L \cup R$ with probability at most $(rt)^{-2}$.

We now consider the case where there is no bad vertex in $L \cup R$. Suppose that there exists a vertex $a \in L \cup R$ that has no incoming edge in D'. Then, since a is not bad, the indegree of a is at most $2\sqrt{r \log(rt)}$ in $D_{\mathcal{M}}(B_1, B_2)$. Therefore, we can apply Lemma 10 with a using $O(\sqrt{r \log(rt)} \log r)$ independence oracle queries.

Therefore, the total number of independence oracle queries used in the algorithm is $O(\sqrt{r \log(rt)} \log r)$ with probability at least $1 - (rt)^{-2}$, which completes the proof. ◀

5.2 Update with a Directed Cycle

In this subsection, we describe how to update the bases using a directed cycle in $D_{\mathcal{M}}(B_1, B_2)$. Let C be a directed cycle in $D_{\mathcal{M}}(B_1, B_2)$ that traverses $u_0, v_0, u_1, v_1, \ldots, v_{l-1}$ in this order, where $u_i \in B_1 \setminus B_2$ and $v_i \in B_2 \setminus B_1$ for each i. In our algorithm, we first choose B_1 with probability $\frac{\beta_2}{\beta_1+\beta_2}$ and choose B_2 with the remaining probability. If we choose B_1, then we pick up an index i uniformly at random from $\{0, \ldots, l-1\}$ and update B_1 by $B_1 \leftarrow B_1 + v_i - u_i$. If we choose B_2, then we pick up an index i uniformly at random from $\{0, \ldots, l-1\}$ and update B_2 by $B_2 \leftarrow B_2 + u_{i+1} - v_i$, where we denote $u_l = u_0$. The pseudocode of this algorithm is shown in `UpdateWithCycle` (Algorithm 4). We note that, if the length of the directed cycle is two, then `UpdateWithCycle` coincides with `UpdateViaStrongBasisExchange`.

Algorithm 4 `UpdateWithCycle`$(\beta_1, B_1, \beta_2, B_2, C)$.

Input: $\beta_1, \beta_2 \in \mathbb{R}_+$, two bases B_1, B_2, and a directed cycle C in the bipartite directed graph $D_{\mathcal{M}}(B_1, B_2)$

Denote by $V(C) = \{u_0, v_0, u_1, v_1, \ldots, v_{l-1}\}$ the vertices in C in this order (with $u_i \in B_1 \setminus B_2$ and $v_i \in B_2 \setminus B_1$ for each i)

Flip a coin with `Heads` probability $\dfrac{\beta_2}{\beta_1 + \beta_2}$

if *coin flipped* `Heads` **then**

  Pick an index i uniformly at random from $\{0, \ldots, l-1\}$

  $B_1 \leftarrow B_1 + v_i - u_i$

else

  Pick an index i uniformly at random from $\{0, \ldots, l-1\}$

  $B_2 \leftarrow B_2 + u_{i+1} - v_i$ `// We define` $u_l = u_0$.

In order to show the validity of the algorithm, we use the following two lemmas.

▶ **Lemma 12.** *Given two bases B_1 and B_2 and a directed cycle C in the bipartite directed graph $D_{\mathcal{M}}(B_1, B_2)$, the procedure* `UpdateWithCycle` *updates B_1 and B_2 to B_1' and B_2', respectively, so that $\mathbb{E}[\beta_1 \mathbf{1}_{B_1'} + \beta_2 \mathbf{1}_{B_2'}] = \beta_1 \mathbf{1}_{B_1} + \beta_2 \mathbf{1}_{B_2}$.*

Proof. Recall that C traverses $u_0, v_0, u_1, v_1, \ldots, v_{l-1}$ in this order, where $u_i \in B_1 \setminus B_2$ and $v_i \in B_2 \setminus B_1$ for each i. In the procedure `UpdateWithCycle`, we obtain $B_1' = B_1 + v_i - u_i$ for some $i \in \{0, \ldots, l-1\}$ and $B_2' = B_2$ with probability $\beta_2/(\beta_1 + \beta_2)$, and we obtain $B_1' = B_1$ and $B_2' = B_2 + u_{i+1} - v_i$ for some $i \in \{0, \ldots, l-1\}$ with probability $\beta_1/(\beta_1 + \beta_2)$. Thus, we have the following equation:

$$\begin{aligned}\mathbb{E}[\beta_1 \mathbf{1}_{B_1'} + \beta_2 \mathbf{1}_{B_2'}] =& \frac{\beta_2}{\beta_1 + \beta_2} \left(\beta_1 \left(\mathbf{1}_{B_1} + \frac{1}{l} \sum_{i=0}^{l-1} (\mathbf{1}_{v_i} - \mathbf{1}_{u_i}) \right) + \beta_2 \mathbf{1}_{B_2} \right) \\ &+ \frac{\beta_1}{\beta_1 + \beta_2} \left(\beta_1 \mathbf{1}_{B_1} + \beta_2 \left(\mathbf{1}_{B_2} + \frac{1}{l} \sum_{i=0}^{l-1} (\mathbf{1}_{u_{i+1}} - \mathbf{1}_{v_i}) \right) \right) \\ =& \beta_1 \mathbf{1}_{B_1} + \beta_2 \mathbf{1}_{B_2}.\end{aligned}$$

This completes the proof. ◀

▶ **Lemma 13** ([18, Lemma VI.2]). *Let $x \in \mathbb{R}_+^n$ be a non-negative vector and $\mathbf{X} = (X_1, \ldots, X_n)$ be a non-negative vector-valued random variable satisfying the following properties:*

- *$\mathbb{E}[\mathbf{X}] = x$, and*
- *$\mathbf{X} - x$ has at most one positive coordinate and at most one negative coordinate.*

Then, we have $\mathbb{E}[F(\mathbf{X})] \geq F(x)$ for any function F that is a multilinear extension of some submodular function.

By combining these lemmas, we obtain the following proposition, which shows the validity of `UpdateWithCycle`.

▶ **Proposition 14.** *Let $x = \sum_{i=1}^t \beta_i \mathbf{1}_{B_i}$ be a point represented by a convex combination of the characteristic vectors of t bases of a matroid $\mathcal{M}$. Suppose that the procedure* `UpdateWithCycle` *updates B_1 and B_2 to B_1' and B_2' using a directed cycle in $D_{\mathcal{M}}(B_1, B_2)$. Let $B_i' = B_i$ for $i \in \{3, \ldots, t\}$ and let $x' = \sum_{i=1}^t \beta_i \mathbf{1}_{B_i'}$. Then, we obtain $\mathbb{E}[F(x')] \geq F(x)$ for any function F that is a multilinear extension of some submodular function.*

Proof. It is obvious that $x' - x$ has at most one positive coordinate and at most one negative coordinate, since only two coordinate are involved in `UpdateWithCycle`, and exactly one of them increases and the other decreases. We also see that $\mathbb{E}[x'] = x$ holds by Lemma 12. Therefore, Lemma 13 shows that $\mathbb{E}[F(x')] \geq F(x)$ for any function F that is a multilinear extension of some submodular function. ◀

5.3 Whole Algorithm

We now prove Theorem 7 by giving our fast swap rounding algorithm. See `FastMergeBases` (Algorithm 5) for the pseudocode of our algorithm.

Proof of Theorem 7. Suppose that $x = \sum_{i=1}^t \beta_i \mathbf{1}_{B_i}$ is a point represented by a convex combination of the characteristic vectors of t bases of a matroid $\mathcal{M}$. We pick up two bases, say B_1 and B_2, in the representation and merge them into a single basis in the following way: until B_1 and B_2 coincide, we find a directed cycle C in $D_{\mathcal{M}}(B_1, B_2)$ using Proposition 9, and update B_1 and B_2 by `UpdateWithCycle` using C. Our algorithm repeats this process $t - 1$ times so that all the bases are merged into a single basis.

Since the correctness of this algorithm is shown by Proposition 14, it remains to analyze the independence query complexity of this rounding algorithm.

For merging two bases into a single basis, since we apply Proposition 9 at most r times, we require $O(r^{3/2}\log^{3/2}(rt))$ independence oracle queries with probability at least $1-r^{-1}t^{-2}$. Furthermore, since we apply this procedure $t-1$ times in our swap rounding algorithm, the entire algorithm requires $O(tr^{3/2}\log^{3/2}(rt))$ independence oracle queries with probability at least $1-(rt)^{-1}$. This completes the proof of Theorem 7. ◀

We can remove the condition "with probability at least $1-(rt)^{-1}$" by losing a sufficiently small approximation factor $\varepsilon > 0$. That is, we obtain Theorem 2, which we restate here.

▶ **Theorem 2.** *For any $\varepsilon > 0$, there is a randomized algorithm satisfying the following conditions:*

- *the input consists of a matroid $\mathcal{M} = (V, \mathcal{I})$ given as an independence oracle and a point x in the base polytope of $\mathcal{M}$ that is represented as a convex combination of t bases,*
- *the output is a basis S of $\mathcal{M}$ such that $\mathbb{E}[f(S)] \geq (1-\varepsilon)F(x)$ for any submodular function $f\colon 2^V \to \mathbb{R}$ and its multilinear extension F, and*
- *it uses $O(r^{3/2}t\log^{3/2}(\frac{rt}{\varepsilon}))$ independence oracle queries.*

Proof. Recall that the algorithm in Theorem 7 (Algorithm 5) uses $O(r^{3/2}t\log^{3/2}(rt))$ independence oracle queries with probability at least $1-(rt)^{-1}$. If Algorithm 5 returns a basis using $O(r^{3/2}t\log^{3/2}(rt))$ independence oracle queries, then we say that it *succeeds*. Otherwise, we say that it *fails*. By a slight modification, when the algorithm fails, we suppose that it uses $O(r^{3/2}t\log^{3/2}(rt))$ independence oracle queries and terminates without returning a basis. This modified algorithm is denoted by Algorithm 5'. Note that Algorithm 5' fails with probability at most $(rt)^{-1}$.

Let $q := \lceil \log_{(rt)^{-1}} \varepsilon \rceil = \lceil \frac{\log(1/\varepsilon)}{\log rt} \rceil = O\left(\frac{\log(rt/\varepsilon)}{\log rt}\right)$. In our algorithm, we run Algorithm 5' q times. If at least one execution of Algorithm 5' succeeds, then our algorithm returns a basis that is obtained in the first successful execution of Algorithm 5'. If all the executions of Algorithm 5' fail, then our algorithm returns an arbitrary basis. Then, we use $O(r^{3/2}t\log^{3/2}(\frac{rt}{\varepsilon}))$ independence oracle queries in total. Furthermore, the probability that all the executions of Algorithm 5' fail is at most $(rt)^{-q} \leq \varepsilon$. Therefore, the output S satisfies $\mathbb{E}[f(S)] \geq (1-\varepsilon)F(x)$ for any submodular function f and its multilinear extension F. This completes the proof. ◀

6 Submodular Maximization with Rank Oracle

In this section, we present a fast submodular maximization algorithm in the rank oracle model and prove Theorem 3. In the rank oracle setting, the input consists of a monotone submodular set function $f\colon 2^V \to \mathbb{R}_+$ given as a value oracle, and a matroid $\mathcal{M} = (V, \mathcal{I})$ given as a rank oracle. The objective is to maximize $f(S)$ subject to $S \in \mathcal{I}$. We restate Theorem 3 here.

▶ **Theorem 3.** *For any $\varepsilon > 0$, there is a randomized algorithm that achieves $(1-1/e-\varepsilon)$-approximation for maximizing a monotone submodular function subject to a matroid constraint and uses $O((n+r^{3/2})\ \mathrm{poly}(1/\varepsilon, \log n))$ value and rank oracle queries.*

In the same way as the independence oracle setting, our algorithm is based on continuous relaxation and rounding technique.

Algorithm 5 `FastMergeBases`$(\beta_1, B_1, \beta_2, B_2)$.

while $B_1 \neq B_2$ **do**
  Sample a set L of $2\sqrt{r \log(rt)}$ elements drawn uniformly and independently from $B_1 \setminus B_2$ with replacement.
  Sample a set R of $2\sqrt{r \log(rt)}$ elements drawn uniformly and independently from $B_2 \setminus B_1$ with replacement.
  $a \leftarrow \emptyset$
  $E \leftarrow \emptyset$
  for $u \in L$ **do**
    $v \leftarrow$ `FindExchangeElement`$(\mathcal{M}, B_2, u, R)$
    if $v = \emptyset$ **then**
      $a \leftarrow u$
    else
      $E \leftarrow E \cup \{(u, v)\}$
  for $v \in R$ **do**
    $u \leftarrow$ `FindExchangeElement`$(\mathcal{M}, B_1, v, L)$
    if $u = \emptyset$ **then**
      $a \leftarrow v$
    else
      $E \leftarrow E \cup \{(v, u)\}$
  if $a = \emptyset$ **then**
    Find a directed cycle C in the bipartite directed graph $(L \cup R, E)$
    `UpdateWithCycle`$(\beta_1, B_1, \beta_2, B_2, C)$
  else
    if $a \in B_1 \setminus B_2$ **then**
      $A \leftarrow \emptyset$
      while $v =$ `FindExchangeElement`$(\mathcal{M}, B_2, a, B_2 \setminus (B_1 \cup A))$ *satisfies* $v \neq \emptyset$ **do**
        if $B_1 + v - a \in \mathcal{I}$ **then**
          `UpdateViaStrongBasisExchange`$(\beta_1, B_1, \beta_2, B_2, v, a)$
          break
        $A \leftarrow A + v$
    else
      $A \leftarrow \emptyset$
      while $u =$ `FindExchangeElement`$(\mathcal{M}, B_1, a, B_1 \setminus (B_2 \cup A))$ *satisfies* $u \neq \emptyset$ **do**
        if $B_2 + u - a \in \mathcal{I}$ **then**
          `UpdateViaStrongBasisExchange`$(\beta_1, B_1, \beta_2, B_2, a, u)$
          break
        $A \leftarrow A + u$
return B_1

Algorithm for the Continuous Relaxation Problem. Let F be the multilinear extension of f and let $P(\mathcal{M})$ be the matroid polytope of $\mathcal{M}$. Ene-Nguyễn [22] presented a framework to solve the continuous optimization problem $\max_{x \in P(\mathcal{M})} F(x)$ in near-linear time for several important classes of matroids. In their algorithm, they use a data structure for maintaining a maximum weight basis of the matroid, where each element has a weight and the weights are updated. In each update, the weight of exactly one element decreases, while all the other weights do not change. The data structure supports an operation that decreases the weight of an element and updates the current basis to a maximum weight basis with respect to the updated weights. This operation is called the *maximum weight basis data structure operation*. With this terminology, their result is stated as follows.

▶ **Lemma 15** (follows from Lemmas 8 and 9 in the arXiv version of [22]). *Given a non-negative monotone submodular function $f\colon 2^V \to \mathbb{R}_+$, a matroid $\mathcal{M} = (V, \mathcal{I})$ of rank r, and a parameter $\varepsilon > 0$, there is a randomized algorithm satisfying the following conditions:*

- *the algorithm finds a point $x \in P(\mathcal{M})$ represented as a convex combination of $1/\varepsilon$ bases such that $\mathbb{E}[F(x)] \geq (1 - 1/e - \varepsilon) \cdot \max\{f(T) \mid T \in \mathcal{I}\}$, where $F\colon [0,1]^V \to \mathbb{R}_+$ is the multilinear extension of f,*
- *it uses $O\left(\frac{n}{\varepsilon^5} \log^2\left(\frac{n}{\varepsilon}\right)\right)$ value oracle queries,*
- *it uses $O\left(\frac{n}{\varepsilon} \log\left(\frac{n}{\varepsilon}\right)\right)$ independence oracle queries, and*
- *it uses $O\left(\frac{r}{\varepsilon} \log^2\left(\frac{n}{\varepsilon}\right)\right)$ maximum weight basis data structure operations.*

Maximum Weight Basis Data Structure Operation. To implement a maximum weight basis data structure operation by using rank oracle queries efficiently, we use the following lemma obtained by the binary search technique of Nguyễn [36] and Chakrabarty et al. [14]; see also [39, Lemma 2].

▶ **Lemma 16** ([14, Lemma 10]; see also [39, Lemma 2] and [10]). *There is an algorithm* `FindFreeElement` *which, given a matroid $\mathcal{M} = (V, \mathcal{I})$, a weight function $w\colon V \to \mathbb{R}$, an independent set $S \in \mathcal{I}$, and $T \subseteq V \setminus S$, finds an element $u \in T$ maximizing $w(u)$ such that $S + u \in \mathcal{I}$ or otherwise determines that no such element exists, and uses $O(\log |T|)$ rank oracle queries.*

This lemma shows that the maximum weight basis data structure operation can be easily implemented in the rank oracle model as follows.

▶ **Lemma 17.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid given as a rank oracle and let $w\colon V \to \mathbb{R}$ be a weight function. Let $w'\colon V \to \mathbb{R}$ be a weight function such that $w'(v) < w(v)$ for some $v \in V$ and $w'(u) = w(u)$ for any $u \in V - v$. Given a maximum weight basis B of $\mathcal{M}$ with respect to w, we can compute a maximum weight basis B' of $\mathcal{M}$ with respect to w' using $\tilde{O}(1)$ rank oracle queries.*

Proof. If $v \notin B$, then $B' := B$ is a desired basis, because $w'(v) < w(v)$. Otherwise, we apply `FindFreeElement` with the weight function w' in which $S = B - v$ and $T = (V \setminus B) \cup \{v\}$. Let u be the element found by the procedure (possibly, $u = v$). Then our algorithm returns a basis $B' := B - v + u$, which is a maximum weight basis with respect to w' (see e.g., [28, Lemma 3.1] and Section 6 of the arXiv version of [9]). By Lemma 16, this algorithm requires $\tilde{O}(1)$ rank oracle queries. ◀

Putting Them Together (Proof of Theorem 3). We now prove Theorem 3. Lemma 17 shows that we can execute the maximum weight basis data structure operation using $\tilde{O}(1)$ rank oracle queries without a sophisticated data structure. Hence, by Lemma 15, we can solve the continuous optimization problem $\max_{x \in P(\mathcal{M})} F(x)$ using $\tilde{O}_\varepsilon(n)$ value and rank oracle queries, where we note that the rank oracle is more powerful than the independence oracle.

For the obtained point x, we apply our fast rounding algorithm given in Theorem 2 to obtain an integral solution. Note again that the rank oracle is more powerful than the independence oracle, and hence this rounding algorithm requires $\tilde{O}_\varepsilon(r^{3/2})$ value and rank oracle queries.

By replacing ε with $\varepsilon/2$ in the same way as in the proof of Theorem 1, we obtain a $(1 - 1/e - \varepsilon)$-approximation algorithm that uses $\tilde{O}_\varepsilon(n + r^{3/2})$ value and rank oracle queries, which completes the proof. ◀

References

1. Alexander A Ageev and Maxim I Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8:307–328, 2004. doi:10.1023/B:JOCO.0000038913.96607.c2.
2. Yossi Azar and Iftah Gamzu. Efficient submodular function maximization under linear packing constraints. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming (ICALP 2012)*, pages 38–50, 2012. doi:10.1007/978-3-642-31594-7_4.
3. Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 1497–1514, 2014. doi:10.1137/1.9781611973402.110.
4. Eric Balkanski, Aviad Rubinstein, and Yaron Singer. An optimal approximation for submodular maximization under a matroid constraint in the adaptive complexity model. *Operations Research*, 70(5):2967–2981, 2022. doi:10.1287/opre.2021.2170.
5. Eric Balkanski and Yaron Singer. The adaptive complexity of maximizing a submodular function. In *Proceedings of the 50th annual ACM SIGACT Symposium on Theory of Computing (STOC 2018)*, pages 1138–1151, 2018. doi:10.1145/3188745.3188752.
6. Kiarash Banihashem, Leyla Biabani, Samira Goudarzi, MohammadTaghi Hajiaghayi, Peyman Jabbarzade, and Morteza Monemizadeh. Dynamic algorithms for matroid submodular maximization. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2024)*, pages 3485–3533, 2024. doi:10.1137/1.9781611977912.125.
7. Rafael da Ponte Barbosa, Alina Ene, Huy L Nguyễn, and Justin Ward. A new framework for distributed submodular maximization. In *Proceedings of the 57th Annual Symposium on Foundations of Computer Science (FOCS 2016)*, pages 645–654, 2016. doi:10.1109/FOCS.2016.74.
8. Joakim Blikstad. Breaking $O(nr)$ for matroid intersection. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 198, pages 31:1–31:17, 2021. doi:10.4230/LIPIcs.ICALP.2021.31.
9. Joakim Blikstad, Sagnik Mukhopadhyay, Danupon Nanongkai, and Ta-Wei Tu. Fast algorithms via dynamic-oracle matroids. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC 2023)*, pages 1229–1242, 2023. arXiv version is arXiv:2302.09796. doi:10.1145/3564246.3585219.
10. Joakim Blikstad, Jan van den Brand, Sagnik Mukhopadhyay, and Danupon Nanongkai. Breaking the quadratic barrier for matroid intersection. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2021)*, pages 421–432, 2021. doi:10.1145/3406325.3451092.
11. Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 1433–1452, 2014. doi:10.1137/1.9781611973402.106.

12 Niv Buchbinder, Moran Feldman, and Roy Schwartz. Comparing apples and oranges: Query trade-off in submodular maximization. *Mathematics of Operations Research*, 42(2):308–329, 2017. doi:10.1287/moor.2016.0809.

13 Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011. doi:10.1137/080733991.

14 Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, Sahil Singla, and Sam Chiu-wai Wong. Faster matroid intersection. In *Proceedings of the 60th Annual Symposium on Foundations of Computer Science (FOCS 2019)*, pages 1146–1168, 2019. doi:10.1109/FOCS.2019.00072.

15 Chandra Chekuri, TS Jayram, and Jan Vondrák. On multiplicative weight updates for concave and submodular function maximization. In *Proceedings of the 6th Conference on Innovations in Theoretical Computer Science (ITCS 2015)*, pages 201–210, 2015. doi:10.1145/2688073.2688086.

16 Chandra Chekuri and Kent Quanrud. Parallelizing greedy for submodular set function maximization in matroids and beyond. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC 2019)*, pages 78–89, 2019. doi:10.1145/3313276.3316406.

17 Chandra Chekuri, Kent Quanrud, and Manuel R Torres. Fast approximation algorithms for bounded degree and crossing spanning tree problems. In *Proceedings of the 24th International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2021)*, volume 207, pages 24:1–24:21, 2021. doi:10.4230/LIPIcs.APPROX/RANDOM.2021.24.

18 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science (FOCS 2010)*, pages 575–584, 2010. doi:10.1109/FOCS.2010.60.

19 Xi Chen and Binghui Peng. On the complexity of dynamic submodular maximization. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (FOCS 2022)*, pages 1685–1698, 2022. doi:10.1145/3519935.3519951.

20 Alina Ene and Huy L. Nguyễn. A nearly-linear time algorithm for submodular maximization with a knapsack constraint. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132, pages 53:1–53:12, 2019. doi:10.4230/LIPIcs.ICALP.2019.53.

21 Alina Ene and Huy L Nguyễn. Submodular maximization with nearly-optimal approximation and adaptivity in nearly-linear time. In *Proceedings of the 30-th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 274–282, 2019. doi:10.1137/1.9781611975482.18.

22 Alina Ene and Huy L. Nguyễn. Towards nearly-linear time algorithms for submodular maximization with a matroid constraint. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132, pages 54:1–54:14, 2019. arXiv version is arXiv:1811.07464. doi:10.4230/LIPIcs.ICALP.2019.54.

23 Alina Ene, Huy L Nguyễn, and Adrian Vladu. Submodular maximization with matroid and packing constraints in parallel. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC 2019)*, pages 90–101, 2019. doi:10.1145/3313276.3316389.

24 Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

25 Yuval Filmus and Justin Ward. A tight combinatorial algorithm for submodular maximization subject to a matroid constraint. In *Proceedings of the 53rd Annual Symposium on Foundations of Computer Science (FOCS 2012)*, pages 659–668, 2012. doi:10.1109/FOCS.2012.55.

26 ML Fisher, GL Nemhauser, and LA Wolsey. An analysis of approximations for maximizing submodular set functions–II. *Mathematical Programming Studies*, 8:73–87, 1978. doi:10.1007/BFb0121195.

27 Monika Henzinger, Paul Liu, Jan Vondrák, and Da Wei Zheng. Faster submodular maximization for several classes of matroids. In *Proceedings of the 50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261, pages 74:1–74:18, 2023. doi:10.4230/LIPIcs.ICALP.2023.74.

28 Felix Hommelsheim, Nicole Megow, Komal Muluk, and Britta Peis. Recoverable robust optimization with commitment, 2023. arXiv:2306.08546.

29 Silvio Lattanzi, Slobodan Mitrović, Ashkan Norouzi-Fard, Jakub M Tarnawski, and Morteza Zadimoghaddam. Fully dynamic algorithm for constrained submodular optimization. *Advances in Neural Information Processing Systems 33: Proceedings of the 34th Annual Conference on Neural Information Processing Systems (Neurips 2020)*, 33:12923–12933, 2020.

30 Wenxin Li, Moran Feldman, Ehsan Kazemi, and Amin Karbasi. Submodular maximization in clean linear time. *Advances in Neural Information Processing Systems 35: Proceedings of the 36th Annual Conference on Neural Information Processing Systems (Neurips 2022)*, 35:17473–17487, 2022.

31 Paul Liu and Jan Vondrák. Submodular optimization in the MapReduce model. In *Proceedings of the 2nd Symposium on Simplicity in Algorithms (SOSA 2019)*, volume 69, pages 18:1–18:10, 2019. doi:10.4230/OASIcs.SOSA.2019.18.

32 Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. Lazier than lazy greedy. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI 2015)*, volume 29(1), 2015. doi:10.1609/aaai.v29i1.9486.

33 Morteza Monemizadeh. Dynamic submodular maximization. *Advances in Neural Information Processing Systems 33: Proceedings of the 34th Annual Conference on Neural Information Processing Systems (Neurips 2020)*, 33:9806–9817, 2020.

34 George L Nemhauser and Laurence A Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of operations research*, 3(3):177–188, 1978. doi:10.1287/moor.3.3.177.

35 George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions–I. *Mathematical Programming*, 14:265–294, 1978. doi:10.1007/BF01588971.

36 Huy L Nguyễn. A note on Cunningham's algorithm for matroid intersection. *arXiv preprint arXiv:1904.04129*, 2019. doi:10.48550/arXiv.1904.04129.

37 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

38 Tatsuya Terao. Faster matroid partition algorithms. In *Proceedings of the 50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261, pages 104:1–104:20, 2023. doi:10.4230/LIPIcs.ICALP.2023.104.

39 Ta-Wei Tu. Subquadratic weighted matroid intersection under rank oracles. In *Proceedings of the 33rd International Symposium on Algorithms and Computation (ISAAC 2022)*, volume 248, pages 63:1–63:14, 2022. doi:10.4230/LIPIcs.ISAAC.2022.63.

On the Space Usage of Approximate Distance Oracles with Sub-2 Stretch

Tsvi Kopelowitz
Bar-Ilan University, Ramat-Gan, Israel

Ariel Korin
Bar-Ilan University, Ramat-Gan, Israel

Liam Roditty
Bar-Ilan University, Ramat-Gan, Israel

Abstract

For an undirected unweighted graph $G = (V, E)$ with n vertices and m edges, let $d(u, v)$ denote the distance from $u \in V$ to $v \in V$ in G. An (α, β)-stretch approximate distance oracle (ADO) for G is a data structure that given $u, v \in V$ returns in constant (or near constant) time a value $\hat{d}(u, v)$ such that $d(u, v) \leq \hat{d}(u, v) \leq \alpha \cdot d(u, v) + \beta$, for some reals $\alpha > 1, \beta$.

Thorup and Zwick [34] showed that one cannot beat stretch 3 with subquadratic space (in terms of n) for general graphs. Pătraşcu and Roditty [27] showed that one can obtain stretch 2 using $O(m^{1/3}n^{4/3})$ space, and so if m is subquadratic in n then the space usage is also subquadratic. Moreover, Pătraşcu and Roditty [27] showed that one cannot beat stretch 2 with subquadratic space even for graphs where $m = \tilde{O}(n)$, based on the set-intersection hypothesis.

In this paper we explore the conditions for which an ADO can beat stretch 2 while using subquadratic space. In particular, we show that if the maximum degree in G is $\Delta_G \leq O(n^{1/k-\varepsilon})$ for some $0 < \varepsilon \leq 1/k$, then there exists an ADO for G that uses $\tilde{O}(n^{2-\frac{k\varepsilon}{3}})$ space and has a $(2, 1-k)$-stretch. For $k = 2$ this result implies a subquadratic sub-2 stretch ADO for graphs with $\Delta_G \leq O(n^{1/2-\varepsilon})$.

Moreover, we prove a conditional lower bound, based on the set intersection hypothesis, which states that for any positive integer $k \leq \log n$, obtaining a sub-$\frac{k+2}{k}$ stretch for graphs with $\Delta_G = \Theta(n^{1/k})$ requires $\tilde{\Omega}(n^2)$ space. Thus, for graphs with maximum degree $\Theta(n^{1/2})$, obtaining a sub-2 stretch requires $\tilde{\Omega}(n^2)$ space.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Graph algorithms, Approximate distance oracle, data structures, shortest path

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.101

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2310.12239

Funding This work was supported in part by the Israel Science Foundation (ISF) grant 1926/19 and by the United States - Israel Binational Science Foundation (BSF) grants 2018364, 2020356.

1 Introduction

One of the most fundamental and classic problems in algorithmic research is the task of computing distances in graphs. Formally, given an undirected unweighted graph $G = (V, E)$, $|V| = n$ and $|E| = m$, the distance between two vertices $u, v \in V$, denoted $d(u, v)$, is the length of a shortest path between u and v. A central problem in distance computations is the *all-pairs shortest paths* (APSP) problem [18, 10, 20, 16, 37, 12, 22] where the objective is to compute the distances between every pair of vertices in the graph. A main disadvantage in handling the output of the APSP problem is that storing the distances between every

pair of vertices in the graph requires $\Omega(n^2)$ space. As in many other problems in computer science, the lack of space efficiency in solving the APSP problem has motivated researchers to search for a tradeoff between space and accuracy. As a result, one central form of the APSP problem emerging from this line of research is constructing an *approximate distance oracle* where we sacrifice accuracy for space efficiency.

Approximate Distance Oracles. An approximate distance oracle (ADO) is a space efficient data structure that produces distance estimations between any two vertices in the graph in constant or near-constant time. Formally, given two vertices, $u, v \in V$, an ADO returns an estimation $\hat{d}(u,v)$ for the distance between u and v that satisfies: $d(u,v) \leq \hat{d}(u,v) \leq \alpha \cdot d(u,v)$, for some real $\alpha > 1$ which is called the *stretch* of the ADO. If the estimation of the ADO satisfies $d(u,v) \leq \hat{d}(u,v) \leq \max\{d(u,v), \alpha \cdot d(u,v) + \beta\}$ for some reals $\alpha > 1$ and β (which can be negative), we say that the stretch of the ADO is an (α, β)-stretch.

ADO for general graphs. ADOs were originally presented by Thorup and Zwick [34] who designed a randomized algorithm that for any positive integer k constructs an ADO for *weighted* undirected graphs in $O(kmn^{1/k})$ time that uses $O(kn^{1+1/k})$ space and returns a $2k-1$-stretch in $O(k)$ query time.

Thorup and Zwick [34] showed that the space usage of their ADO construction for their given stretch is optimal for *general graphs* based on the girth conjecture of Erdős. Moreover, for stretch 3 (when $k = 2$), the appropriate case of the girth conjecture is known to be true (due to complete bipartite graphs), and so the quadratic (in n) space lower bound for this case is unconditional. Notice that constructing an exact distance oracle in quadratic space is trivial.

In the case where one allows for an additive error, Pătraşcu and Roditty [27] designed an algorithm which constructs a $(2,1)$-stretch ADO for unweighted graphs using $O(n^{5/3})$ space and $O(1)$ query time. Their result demonstrates that in such a case the multiplicative error can be reduced while still using subquadratic space.

ADO for sparse graphs. The (conditional) lower bound of Thorup and Zwick [34] does not apply to sparser graphs with $m = o(n^{1+1/k})$, and indeed additional results show that it is possible to use subquadratic space and return a sub-3 stretch in such cases. Specifically, Pătraşcu and Roditty [27], designed an algorithm that constructs a 2-stretch ADO using $O(m^{1/3}n^{4/3})$ space, and so for subquadratic m the space usage is subquadratic. Pătraşcu, Roditty and Thorup [28] presented additional tradeoffs for sub-3 stretch using subquadratic space for graphs where $m = \tilde{O}(n)$[1]. Roditty and Tov [30] improved the stretch of the ADO presented by Thorup and Zwick [34] while using the same space for graphs with $m = \tilde{O}(n)$.

Conditional lower bounds and set-intersection. Pătraşcu and Roditty [27] proved a lower bound for the space usage of sub-2 stretch ADOs (i.e., ADOs which satisfy $d(u,v) \leq \hat{d}(u,v) < 2d(u,v)$) that holds (even) for sparse graphs, conditioned on the space usage for data structures that solve the following set intersection problem.

▶ **Problem 1.** *Let $X = \log^c N$ for a large enough constant c. Construct a data structure that preprocesses sets $S_1, \ldots, S_N \subseteq [X]$, and answers queries of the form "does S_i intersect S_j?" in constant time.*

[1] Throughout this paper we use $\sim$ when suppressing poly-logarithmic factors in asymptotic complexities.

The lower bound proof by Pătraşcu and Roditty [27] is based on the following hypothesis.

▶ **Hypothesis 2** ([27, 31, 17]). *A data structure that solves Problem 1 requires $\tilde{\Omega}(N^2)$ space.*

Since understanding the reduction by Pătraşcu and Roditty [27] is useful for our results, we provide an overview of their reduction tailored to our needs. Given an instance of Problem 1, we construct a 3 *layered* graph, where edges are only between adjacent layers, as follows. The first layer is $V_L = \{v_1, \ldots, v_N\}$, the second layer is $V_M = X$, and the third layer is $V_R = \{u_1, \ldots, u_N\}$. Vertices v_i and u_i represent S_i, and so for each set S_i and each $x \in S_i$, we add edges (v_i, x) and (x, u_i). Notice that the graph contains $\Theta(N)$ vertices. It is straightforward to observe that $S_i \cap S_j \neq \emptyset$ if and only if there is a path of length 2 between v_i and u_j. Moreover, since the graph is a 3 layered graph and the representatives of the sets are at the outer layers, there are no paths of length 3 between representatives of sets. Thus, one can solve Problem 1 using a solution to the following problem (for $a = 2$ and $b = 4$).

▶ **Problem 3.** *For positive integers a and b, an (a, b)-*distinguisher oracle *for a graph $G = (V, E)$, is a data structure that, given $u, v \in V$ establishes in constant time whether $d(u, v) \leq a$ or $d(u, v) \geq b$. If $a < d(u, v) < b$ then the data structure can return any arbitrary answer.*

We conclude that a $(2, 4)$-distinguisher oracle that uses $f(n)$ space can be used to solve Problem 1 using $f(N)$ space by applying the oracle onto the 3 layered graph. Finally, since a sub-2 stretch ADO is a $(2, 4)$-distinguisher oracle, any sub-2 stretch ADO must use at least $\Omega(n^2)$ space.

1.1 Main results: When can we beat stretch 2 with subquadratic space?

The line of work by [27, 28, 30] is a natural research path given the observation that the (conditional) lower bounds of [34] apply only to graphs with $m = \Omega(n^{1+1/k})$. Similarly, a natural goal, which we address in this paper, is to understand for which families of graphs can an ADO beat stretch 2 using subquadratic space. In particular, the conditional lower bound proof of Pătraşcu and Roditty [27] does not apply to graphs with maximum degree of $n^{\frac{1}{2}-\Omega(1)}$, since in such graphs the number of paths of length 2 is $n^{2-\Omega(1)}$, and so constructing a subquadratic space $(2, 4)$-distinguisher oracle is straightforward (by explicitly storing all length 2 paths).

Thus, a natural goal, which we investigate in this paper, is to understand the relationship between the maximum degree of G, denoted by Δ_G, and the best possible stretch obtainable for an ADO using subquadratic space. To address this question, we present two main results. The first result is an upper bound for $\Delta_G = n^{\frac{1}{k}-\Omega(1)}$ which is summarized in the following Theorem.

▶ **Theorem 4.** *For any graph G, positive real constant c, and positive integer k for which $\Delta_G \leq cn^{\frac{1}{k}-\varepsilon}$ for some real $0 < \varepsilon \leq 1/k$, there exists an ADO for G that uses $\tilde{O}(c^k n^{2-\frac{k\varepsilon}{3}})$ space and has a $(2, 1-k)$-stretch.*

For $k = 2$, Theorem 4 implies a subquadratic sub-2 stretch ADO for graphs for which $\Delta_G \leq n^{\frac{1}{2}-\Omega(1)}$.

The second result is a conditional lower bound, conditioned on Hypothesis 2, that applies when $\Delta_G = \Theta(n^{1/k})$, for all integers $k \geq 1$ [2]. The conditional lower bound is summarized in the following Theorem.

[2] The case $k = 1$ was proven by Thorup and Zwick [34] to hold unconditionally. Thus, Theorem 5 focuses on $k \geq 2$.

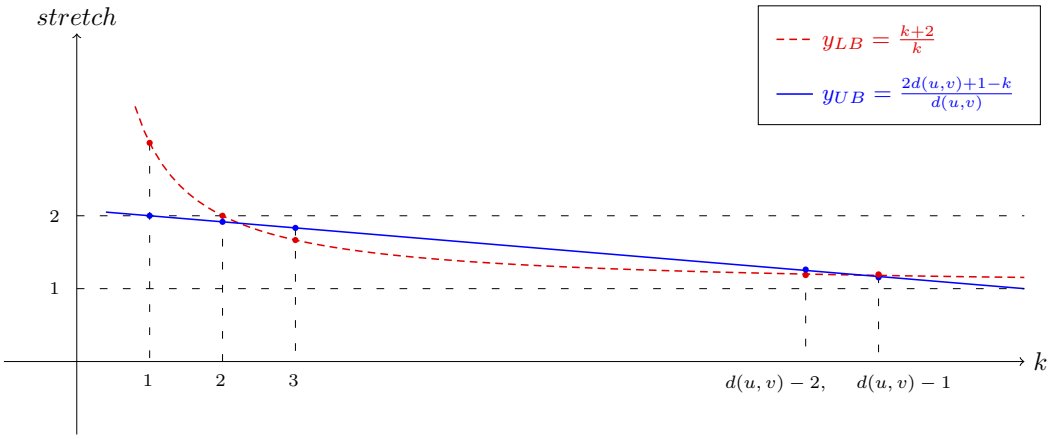


Figure 1 A comparison between the upper bound for graphs with $\Delta_G \leq O(n^{\frac{1}{k}-\varepsilon})$ (Theorem 4) and the lower bound for graphs with $\Delta_G = \Theta(n^{\frac{1}{k}})$ (Theorem 5). The figure demonstrates the stretch as a function of k for a fixed (sufficiently large) value of $d(u,v)$. The curves intersect at $2 < k_1 < 3$ and at $d(u,v) - 2 < k_2 < d(u,v) - 1$ which implies that for $k = 1, 2$ the ADO from Theorem 4 for graphs with $\Delta_G \leq O(n^{\frac{1}{k}-\varepsilon})$ produces a better stretch than the best possible stretch for graphs with $\Delta_G = \Theta(n^{\frac{1}{k}})$. On the other hand, the intersection point k_2 does not provide strong enough results; see the discussion following the statement of Theorem 5.

▶ **Theorem 5.** *Let $2 \leq k \leq \log n$. Assuming Hypothesis 2, a sub-$\frac{k+2}{k}$ stretch ADO for graphs with n vertices and maximum degree $\Theta(n^{1/k})$ must use $\tilde{\Omega}(n^2)$ space.*

When $k = 2$, the lower bound of Theorem 5 implies that Theorem 4 is essentially optimal, in the sense that there is no ADO for graphs with a maximum degree of $\Theta(n^{1/2})$ that still uses subquadratic space and has a sub-2 stretch.

For larger values of k, although the upper and lower bounds are defined only for integer values of k, the values lie on two curves: $y_{UB} = \frac{2d(u,v)+1-k}{d(u,v)}$ for the upper bound and $y_{LB} = \frac{k+2}{k}$ for the lower bound. See Figure 1 for a depiction and comparison of the two curves for a fixed value of $d(u,v)$. When $d(u,v) \leq 6$ (which is not the case shown in Figure 1), the upper bound will always produce a stretch which is equal or smaller to the lower bound. Otherwise, the curves intersect at $2 < k_1 < 3$ and $d(u,v) - 2 < k_2 < d(u,v) - 1$. We emphasize that the bounds on k_1 are independent of $d(u,v)$, and so for $k = 1, 2$, the fact that the upper bound curve is beneath the lower bound curve is relevant for all for all possible distances. On the other hand, the bounds on k_2 depend on $d(u,v)$, and so while it is true that for a fixed value of large enough $d(u,v)$ there are many values of k for which the ADO of Theorem 4 provides an approximation which beats the lower bound, it is also true that for any integer value of $k \geq 3$ there will always exist some distances for which the upper bound is above the lower bound. Thus, the intersection at k_2 is unfortunately not meaningful enough.

1.2 Organization

The rest of this paper is organized as follows. In Section 1.3 we provide an overview of the main ideas used in this paper. In Section 1.4 we survey some additional related work. In Section 2 we provide some definitions that are used in the more technical parts of the paper. In Section 3 we prove some useful lemmas that are used in the proof of Theorem 4, which is described in Section 4 together with the construction of our new ADO. In Section 5 we prove Theorem 5. Finally, in Section 6 we provide some conclusions and describe a natural open problem.

1.3 Overview of Results and Techniques

In this section we describe an overview of the intuition and techniques used to obtain our main results.

1.3.1 Upper Bound: A New ADO

Since our new ADO is based on the ADO of Agarwal and Godfrey [4], that has a $(2,1)$-stretch and uses $\tilde{O}(n^{5/3})$ space (which simplifies the ADO of [27]), we provide an overview of the construction of their ADO.

The ADO constructed by Agarwal and Godfrey [4] uses the concept of *bunches* and *clusters* introduced by Thorup and Zwick [33]. Following the conventions of Thorup and Zwick [33, 34], for a vertex $v \in V$ and set $A \subseteq V$, let $p_A(v)$ be the vertex in A which is closest to v (breaking ties arbitrarily). The *bunch* $B_A(v)$ of v with respect to A is defined as $B_A(v) = \{w \in V \mid d(v,w) < d(v,p_A(v))\}$. The *cluster* $C_A(w)$ of w with respect to A is defined as $C_A(w) = \{v \in V \mid d(w,v) < d(v,p_A(v))\}$. We omit A from the notation when it is clear from context. Thorup and Zwick [33] presented an algorithm that computes a set A of size $\tilde{O}(s)$ such that $|B(v)|, |C(v)| \leq 4n/s$, for every $v \in V$. The ADO of Agarwal and Godfrey [4] uses a hitting set A of size $\tilde{O}(n^{2/3})$ such that for every $v \in V$, $|B(v)|, |C(v)| \leq O(n^{1/3})$.

Given two vertices $u, v \in V$, the ADO first tests whether $B(u) \cap B(v) \neq \emptyset$, and, if so, then the ADO returns the exact distance $d(u,v)$. The method for testing whether the two bunches intersect is based on the observation (which follows from the definitions of bunch and cluster) that $B(u) \cap B(v) \neq \emptyset$ if and only if $u \in C(B(v))$[3]. Thus, each vertex $v \in V$ stores the exact distances to all vertices in $C(B(v))$, and now, the case of $B(u) \cap B(v) \neq \emptyset$ costs constant time and returns an exact distance. To deal with the case of $B(u) \cap B(v) = \emptyset$, the oracle stores the distances of pairs in $V \times A$, and the ADO returns the minimum of either the length of the shortest path between u and v passing through $p(u)$ or the length of the shortest path between u and v passing through $p(v)$. The space usage is $O(n^{5/3})$ for storing $C(B(v))$ for every $v \in V$, and $\tilde{O}(n|A|) = \tilde{O}(n^{5/3})$ for storing the distances for all pairs in $V \times A$.

Intuition for the new ADO. Our new ADO construction is based on the following intuition regarding the ADO construction of Agarwal and Godfrey [4]. If we enlarge $B(u)$ by moving $p(u)$ to a further vertex (from u), then we would increase the likelihood of $B(u) \cap B(v) \neq \emptyset$, and so the ADO would return exact distances for more pairs of vertices. However, in such a case, the quality guarantee on the stretch obtained by approximating $d(u,v)$ with the shortest path from u to v that passes through $p(u)$ becomes worse. Part of the challenge is to balance the size of $B(u)$ which affects the usefulness of the intersections and the role of $p(u)$ when approximating the distances.

Our approach, intuitively, is to separate the definition of $p(u)$ used for the approximations and the set chosen for the intersections. Specifically, the definition of $p(u)$ remains unchanged relative to A (we do however change the size of A), but instead of testing whether $B(u) \cap B(v) \neq \emptyset$, we use a larger set $N(u)$ (which contains $B(u)$), and test whether $N(u) \cap B(v) \neq \emptyset$ (or $B(u) \cap N(v) \neq \emptyset$). Testing whether $N(u) \cap B(v) \neq \emptyset$ is implemented by storing all of the distances between u and vertices in $C(N(u))$. We remark that one may consider the possibility of testing whether $N(u) \cap N(v) \neq \emptyset$ instead of testing whether $N(u) \cap B(v) \neq \emptyset$, however, such an approach seems to require too much space.

[3] For $S \subseteq V$, let $C(S) = \bigcup_{u \in S} C(u)$.

Recall that Agarwal and Godfrey [4] obtained a $(2,1)$-stretch. In our algorithm, we choose $N(u)$ in such a way that when $N(u) \cap B(v) = \emptyset$ then $\min_{x \in B(u), y \in B(v)}\{d(x,y)\} > k$ if $\Delta_G \leq n^{\frac{1}{k}-\Omega(1)}$, which ends up reducing the additive component of the stretch by at least k (see Claim 13). Thus, the approximation of the ADO is always at most $(2, 1-k)$, which is less than stretch 2 for $k \geq 2$.

1.3.2 Conditional Lower bound

In Section 5 we prove the following lemma, which directly implies Theorem 5 since a sub-$\frac{k+2}{k}$ stretch ADO is also a $(k, k+2)$-distinguisher oracle.

▶ **Lemma 6.** *Let $2 \leq k \leq \log n$. Assuming Hypothesis 2, any $(k, k+2)$-distinguisher oracle for graphs with n vertices and maximum degree $\Theta(n^{1/k})$ must use $\tilde{\Omega}(n^2)$ space.*

Notice that it is straightforward to construct a $(k, k+2)$-distinguisher oracle for graphs with n vertices and maximum degree $\Theta(n^{\frac{1}{k}-\varepsilon})$ in $O(n^{2-k\varepsilon})$ space by storing all pairs of vertices at distance exactly k. Thus, Lemma 6 shows that such a construction is essentially optimal.

The challenges. There are two issues that need to be addressed in order to extend the reduction by Pătraşcu and Roditty [27] in a way that proves Lemma 6. The first is to adjust the distances so that $S_i \cap S_j \neq \emptyset$ if and only if $d(v_i, u_j) = k$, and otherwise, $d(v_i, u_j) \geq k+2$. The second issue is that the degrees of vertices in V_M need to be adjusted to be at most $\tilde{O}(N^{1/k})$. In order to simplify our intuitive explanation, we focus our attention to the special case where $X = \{x\}$ has only one element.

One straightforward way of dealing with the first issue is to replace vertex $x \in V_M$ with a path $P_x = (w_1, \ldots, w_{k-1})$ of length $k-2$, and for each S_i that contains x we add edges (v_i, w_1) and (w_{k-1}, u_i). Thus, the constructed graph would be a $k+1$ layered graph. The number of vertices in such a graph is $O(k+N) = O(N)$, and the distance between vertices in the first and last layers are $k+2q$ for some integer $q \geq 0$. However, we still need to address the second issue of bounding the maximum degree, since w_1 and w_{k-1} may have a very high degree corresponding to the number of sets containing x.

On the other hand, one initial idea (that does not work) for dealing with the second issue is to replace $x \in V_M$ (in the original 3 layered graph) with N vertices $y_1, y_2, \ldots, y_N$, and for each S_i that contains x we add edges (v_i, y_i) and (y_i, u_i). Now the maximum degree of each node is constant, however, for $i \neq j$ such that $x \in S_i \cap S_j$, there is no path from v_i to u_j. This idea is missing the functionality of the path P_x which allows us to connect more than one pair of vertices from $V_L \times V_R$.

Combining approaches. Our reduction makes use of an underlying $k+1$ layered *infrastructure graph* $\mathcal{L}$, commonly known as the *butterfly graph* (see [26, 28]), which has the following three properties: (*i*) each layer contains N vertices, (*ii*) there is a path of length k from every vertex in the first layer to every vertex in the last layer, and (*iii*) the degree of every vertex is at most $2N^{1/k}$. The layers of $\mathcal{L}$ are numbered 0 to k. The vertices in each layer are (separately) indexed with integers from 1 to N, and the construction of $\mathcal{L}$ is based on the base $N^{1/k}$ representation of the these indices: for $1 \leq t \leq k$, vertices from layer $t-1$ are connected with vertices from layer t if and only if the base $N^{1/k}$ representation of their corresponding indices are the same, except for possibly the t'th digit. Similar to before, we denote the first layer of $\mathcal{L}$ by $V_L = \{v_1, \ldots, v_N\}$ and the last layer by $V_R = \{u_1, \ldots, u_N\}$.

Finally, we construct a $k+1$ layered graph G_x which is intuitively obtained by removing from $\mathcal{L}$ edges touching either v_i or u_i for every S_i that does not contain x. Thus, in G_x, if $x \in S_i \cap S_j$ then there is a path of length k from v_i to u_j in G_x, and otherwise, there is no path from v_i to u_j in G_x.

We remark that in the general case, where $|X|$ may be larger than 1, we combine G_x for different $x \in X$ in a special way, and so we may introduce paths from v_i to u_j even if $S_i \cap S_j = \emptyset$. However, since the resulting graph is still a $k+1$ layered graph, and we are interested in paths between vertices in the first layer and vertices in the last layer, the lengths of such paths must be at least $k+2$. Thus, a $(k, k+2)$-distinguisher oracle on the combined graph suffices for solving Problem 1. See Section 5 for more details.

1.4 Additional related work

Different aspects of Thorup and Zwick [34] ADOs were studied since they were introduced for the first time. Chechik [14, 13] reduced the query time from $O(k)$ to $O(1)$, while keeping the stretch and the space unchanged. (See also [36, 23].) Roditty, Thorup, and Zwick [29] presented a deterministic algorithm that constructs an ADO in $\tilde{O}(mn^{1/k})$ time while keeping the stretch and the space unchanged. Baswana and Kavitha [9] presented an algorithm with $O(n^2 \log n)$ running time[4]. Baswana, Goyaland and Sen [8] presented an $\tilde{O}(n^2)$ time algorithm that computes a $(2,3)$-distance oracle with $\tilde{O}(n^{5/3})$ space. Sommer [32] presented an $\tilde{O}(n^2)$ time algorithm that computes a $(2,1)$-distance oracle with $\tilde{O}(n^{5/3})$ space. Akav and Roditty [7] presented the first sub-quadratic time algorithm that constructs an ADO with stretch better than 3. They presented an $O(n^{2-\epsilon})$-time algorithm that constructs a ADO with $O(n^{11/6})$ space and $(2+\epsilon, 5)$-stretch. Chechik and Zhang [15] improved the result of Akav and Roditty [7]. Among their results is an $O(m + n^{1.987})$ time algorithm that constructs an ADO with $(2,3)$-stretch and $\tilde{O}(n^{5/3})$ space. Following the work by Pătraşcu and Roditty [27] who constructed an ADO for unweighted graphs that uses $O(n^{5/3})$ space and returns a $(2,1)$-stretch in $O(1)$ time, Abraham and Gavoille [2] extended the ADO by Pătraşcu and Roditty [27] for all even stretch values, by constructing for any integer $k \geq 2$, an ADO of size $\tilde{O}(n^{1+2/(2k-1)})$ with a $(2k-2,1)$-stretch returned in $O(k)$ time. Pătraşcu, Roditty and Thorup [28] focused on analyzing sparse graphs where $m = \tilde{O}(n)$ and noted that both the ADOs by Thorup and Zwick [34], and the ADOs by Abraham and Gavoille [2] use a space complexity that can be described by the curve $S(\alpha, m) = \tilde{O}(m^{1+2/(\alpha+1)})$ where α is the stretch of the ADO and m is the number of edges in the graph. Pătraşcu, Roditty and Thorup [28] extended the curve $S(\alpha, m)$ to work for non integer stretch values $\alpha > 2$. Although our research focuses on constant query time ADOs, another branch of research includes ADOs that have non constant query time [6, 24, 5, 3, 11].

In the lower bound regime, the problem of constructing a $(2,4)$-distinguisher oracle was analyzed from the perspective of time complexity as well. For graphs with degree of at most $n^{1/2}$, the problem of determining for each edge in the graph whether it is in a triangle in $O(n^{2-\varepsilon})$ time for some $\varepsilon > 0$ was shown to be hard under either the 3SUM [25, 21] or APSP [35] hypotheses. Since there exists a standard reduction from the problem of determining for each edge in the graph whether it is in a triangle to the problem of constructing a $(2,4)$-distinguisher oracle (see [1]), a $(2,4)$-distinguisher oracle is also hard to construct in subquadratic time for graphs with degree of at most $n^{1/2}$ under either the 3SUM or APSP hypotheses.

[4] For $k = 2$ the query time is $O(\log n)$. For $k > 2$ the query time is $O(k)$.

The problem of constructing a $(k, k+2)$-distinguisher oracle for a general integer $k \geq 2$ was also studied in the past in terms of time complexity. Dor, Halperin and Zwick [19] showed that if all distances in an undirected n vertex graph can be approximated with an additive error of at most 1 in $O(A(n))$ time, then *Boolean matrix multiplication* on matrices of size $n \times n$ can also be performed in $O(A(n))$ time. Dor, Halperin and Zwick [19] conclude that constructing a $(k, k+2)$-distinguisher oracle for an integer $k \geq 2$ is at least as hard as multiplying two Boolean matrices.

2 Preliminaries

Let $d_G(u, v)$ be the distance between vertices u and v in the graph G. The eccentricity of a vertex $v \in V$ in a graph G, denoted by $ecc_G(v)$, is defined as $ecc_G(v) = \max_{u \in V}\{d_G(v, u)\}$. The diameter of G is defined as $diam_G = \max_{v \in V}\{ecc_G(v)\}$ and the radius of G is defined as $rad_G = \min_{v \in V}\{ecc_G(v)\}$.

The eccentricity of a vertex v can be thought of as the distance between v and the last vertex met during a *Breadth First Search* (BFS) of the graph starting at v. Since our goal is to construct an ADO that uses subquadratic space, we cannot afford to store a separate BFS tree for each vertex. Instead, the construction algorithm of the ADO from Theorem 4 will store only a partial BFS tree for each vertex by truncating the BFS scan after some number of vertices. Motivated by this notion of a truncated scan, we introduce the following generalization of eccentricity which turns out to be useful for our purposes.

Let $N_G(v, s)$ be the first s vertices met during a BFS[5] starting from v in the graph G, i.e., the s closest vertices to v (excluding v). If s is not an integer, then let $N(v, s) = N(v, \lfloor s \rfloor)$. For an integer $r \geq 0$, define $L_G(v, r) = \{u \in V \setminus \{v\} \mid d_G(u, v) = r\}$ and $T_G(v, r) = \{u \in V \mid 0 < d_G(u, v) \leq r\}$. Notice that $T_G(v, r) = \bigcup_{i=1}^{r} L_G(v, i)$. For any real $1 \leq s \leq n-1$, define $ecc_G(v, s)$ to be the maximum integer $k \in [0, ecc_G(v)]$ for which $T_G(v, k) \subseteq N_G(v, s)$. Notice that $ecc_G(v, n-1) = ecc_G(v)$. Define $rad_G(s) = \min_{v \in V}\{ecc_G(v, s)\}$. Notice that $rad_G(n-1) = rad_G$. We omit the subscript G when using the definitions above whenever G is clear from context.

3 Useful Lemmas

In this section we prove several useful properties of the graph attributes defined in Section 2 which will be used throughout the paper.

The following observation and corollary address the relationship between $T(v, r)$ and $N(v, s)$, and follow from the definition of BFS.

▶ **Observation 7.** *Let $G = (V, E)$ be an unweighted undirected graph, with $|V| = n$. For any $v \in V$ and integers s and r such that $1 \leq s < n$ and $1 \leq r \leq diam_G$, either (i) $T(v, r) \subset N(v, s)$, (ii) $N(v, s) \subset T(v, r)$, or (iii) $N(v, s) = T(v, r)$*

▶ **Corollary 8.** *Let $G = (V, E)$ be an unweighted undirected graph, with $|V| = n$. For any $v \in V$ and integers s and r such that $1 \leq s < n$ and $1 \leq r \leq diam_G$, (i) if $|T(v, r)| < |N(v, s)|$ then $T(v, r) \subset N(v, s)$, (ii) if $|N(v, s)| < |T(v, r)|$ then $N(v, s) \subset T(v, r)$, and (iii) if $|N(v, s)| = |T(v, r)|$ then $N(v, s) = T(v, r)$.*

The following useful property addresses the relationship between $T(v, r)$ and $N(v, s)$ for the special cases where either $r = ecc(v, s)$ or $r = ecc(v, s) + 1$.

[5] The traversal order of vertices in the same layer during the BFS execution does not matter as long as the order is consistent.

▶ **Property 9.** *Let $G = (V, E)$ be an unweighted undirected graph, with $|V| = n$. For any $v \in V$ and integer s such that $1 \leq s < n$, we have: (i) $T(v, ecc(v, s)) \subseteq N(v, s)$, and (ii) if $s < n - 1$, then $T(v, ecc(v, s) + 1) \not\subseteq N(v, s)$.*

Proof. By definition, $ecc(v, s)$ is the largest integer $k \in [0, ecc(v)]$ for which $T(v, k) \subseteq N(v, s)$. Thus, (i) $T(v, ecc(v, s)) \subseteq N(v, s)$, and (ii) if $ecc(v, s) < ecc(v)$ then $T(v, ecc(v, s) + 1) \not\subseteq N(v, s)$. If $s < n - 1$, it must be that $ecc(v, s) < ecc(v)$, since if we assume towards a contradiction that $ecc(v, s) = ecc(v)$ for some $s < n - 1$ then $T(v, ecc(v, s)) = T(v, ecc(v)) = V \setminus \{v\} = N(v, n-1)$ but on the other hand, by definition of $ecc(v, s)$, we have $T(v, ecc(v, s)) \subseteq N(v, s) \subset N(v, n - 1)$, which is a contradiction. ◀

The following lemma states that $ecc_G(v, s)$ exhibits a behavior that is similar to the behavior of the distance function which cannot decrease when removing edges and vertices from G.

▶ **Lemma 10.** *Let $G = (V, E)$ be an unweighted undirected graph, $V' \subseteq V$, and let G' be the subgraph of G induced by the vertices in V'. For any vertex $v \in V'$ and for any integer s such that $1 \leq s < |V'|$, it holds that $ecc_G(v, s) \leq ecc_{G'}(v, s)$.*

Proof. Given an integer $1 \leq s < |V'|$, let $r = ecc_G(v, s)$ and $r' = ecc_{G'}(v, s)$. We want to show that $r \leq r'$. By definition of $ecc_{G'}(v, s)$, $r' = ecc_{G'}(v, s)$ is the largest value for which $T_{G'}(v, r') \subseteq N_{G'}(v, s)$. Thus, in order to show that $r \leq r'$, it suffices to show that $T_{G'}(v, r) \subseteq N_{G'}(v, s)$.

For any vertex pair $u, w \in V'$, we have $d_G(u, w) \leq d_{G'}(u, w)$ since G' is a subgraph of G. Thus,

$$\begin{aligned} T_{G'}(v, r) &= \{u \in V' \mid 0 < d_{G'}(v, u) \leq r\} \\ &\subseteq \{u \in V \mid 0 < d_G(v, u) \leq r\} \\ &= T_G(v, r) \\ &\underset{\text{Property 9}}{\subseteq} N_G(v, s). \end{aligned}$$

This implies that $|T_{G'}(v, r)| \leq |N_G(v, s)| = s = |N_{G'}(v, s)|$. By Corollary 8, since $|T_{G'}(v, r)| \leq |N_{G'}(v, s)|$ then $T_{G'}(v, r) \subseteq N_{G'}(v, s)$, as required. ◀

3.1 The Logarithmic-Like Behavior of Eccentricity

In the following lemma, which is an important ingredient in the analysis of our new ADO, we show that $ecc(v, s)$ satisfies a logarithmic-like behavior. Specifically, $\log(xy) = \log x + \log y$. The reason for this behavior is that the number of vertices in each layer of a BFS tree expands in a similar way to an exponential function. For a tree-graph G with minimum degree δ rooted at a vertex v, for integers $0 \leq i < t < ecc(v)$ it holds that $|L(v, t)| \geq \delta^i \cdot |L(v, t - i)|$. Since the number of vertices in every layer of the rooted tree grows exponentially, the eccentricity $ecc(v, s)$ grows logarithmically (in relation to s). Unlike in trees where the expansion of the number of vertices in every layer of a BFS can be analyzed using δ, for general graphs, in order to achieve a lower bound for the expansion rate of the eccentricity of the vertices, we use $rad_G(s)$ instead.

▶ **Lemma 11.** *Let $G = (V, E)$ be an unweighted undirected graph, with $|V| = n$. For any vertex $v \in V$ and integers $s_1, s_2 \geq 1$ such that $s_1(s_2 + 1) < n - 1$, it holds that $ecc_G(v, s_1(s_2 + 1)) \geq ecc_G(v, s_1) + rad_G(s_2)$.*

Proof. Assume towards a contradiction that $ecc_G(v, s_1(s_2+1)) < ecc_G(v,s_1) + rad_G(s_2)$. Thus, $ecc_G(v, s_1(s_2+1)) + 1 \leq ecc_G(v,s_1) + rad_G(s_2)$.

Let T_{BFS} be a BFS tree rooted at v in graph G. Let $\ell = |L(v, ecc_G(v,s_1))|$ and let $u_1, u_2, \ldots, u_\ell$ be the vertices in $L(v, ecc_G(v,s_1))$. For any i, where $1 \leq i \leq \ell$, let V_i be the set of descendant of u_i in T_{BFS} and let G_i be the graph induced by V_i in G[6].

Let $\mu = \min_{i\in[1,\ell]}\{ecc_{G_i}(u_i, s_2)\}$. We will show that:

$$\{u \in V \mid ecc_G(v,s_1) < d_G(v,u) \leq ecc_G(v,s_1) + \mu\} \subseteq \bigcup_{i=1}^{\ell} N_{G_i}(u_i, s_2). \tag{1}$$

Let $w \in \{u \in V \mid ecc_G(v,s_1) < d_G(v,u) \leq ecc_G(v,s_1) + \mu\}$. By the definition of BFS, since $d_G(v,w) > ecc_G(v,s_1)$, w must be a descendant of some vertex u_j, and so $w \in V_j$. Since T_{BFS} is a shortest path tree rooted at v and since $w \in V_j$, it must be that $d_G(v,w) = d_G(v,u_j) + d_G(u_j,w) = ecc_G(v,s_1) + d_G(u_j,w)$. By definition of w, $d_G(v,w) \leq ecc_G(v,s_1) + \mu$. Thus, $ecc_G(v,s_1) + d_G(u_j,w) \leq ecc_G(v,s_1) + \mu$ and so $d_G(u_j,w) \leq \mu$.

By definition, $T_{G_j}(u_j, ecc_{G_j}(u_j,s_2)) = \{x \mid x \in V_j \wedge 0 < d_{G_j}(u_j,x) \leq ecc_{G_j}(u_j,s_2)\}$. Since T_{BFS} is a shortest path tree, for any $y \in V_j$ it must be that $d_{G_j}(u_j,y) = d_G(u_j,y)$. Thus, $T_{G_j}(u_j, ecc_{G_j}(u_j,s_2)) = \{x \mid x \in V_j \wedge 0 < d_G(u_j,x) \leq ecc_{G_j}(u_j,s_2)\}$.

Since $w \in V_j$, and since $d_G(u_j,w) \leq \mu \leq ecc_{G_j}(u_j,s_2)$, then $w \in T_{G_j}(u_j, ecc_{G_j}(u_j,s_2))$. By Property 9, $T_{G_j}(u_j, ecc_{G_j}(u_j,s_2)) \subseteq N_{G_j}(u_j,s_2)$. It follows that every vertex in $\{u \in V \mid ecc_G(v,s_1) < d_G(v,u) \leq ecc_G(v,s_1) + \mu\}$ must be included in $N_{G_i}(u_i,s_2)$ for some i, thus confirming Equation (1).

By Property 9, $\{u \in V \mid 0 < d_G(v,u) \leq ecc_G(v,s_1)\} = T(v, ecc_G(v,s_1)) \subseteq N(v,s_1)$. Combining with Equation (1) we have that $\{u \in V \mid 0 < d_G(v,u) \leq ecc_G(v,s_1)\} \cup \{u \in V \mid ecc_G(v,s_1) < d_G(v,u) \leq ecc_G(v,s_1) + \mu\} \subseteq N(v,s_1) \cup \left(\bigcup_{i=1}^{\ell} N_{G_i}(u_i,s_2)\right)$, and so:

$$\begin{aligned}\{u \in V \mid 0 < d_G(v,u) \leq ecc_G(v,s_1) + \mu\} &= T(v, ecc_G(v,s_1) + \mu) \\ &\subseteq N(v,s_1) \cup \left(\bigcup_{i=1}^{\ell} N_{G_i}(u_i,s_2)\right).\end{aligned}$$

Now,

$$\begin{aligned} ecc_G(v,s_1) + \mu &\underset{\substack{\uparrow \\ \text{definition of } \mu}}{=} ecc_G(v,s_1) + \min_{i\in[1,\ell]}\{ecc_{G_i}(u_i,s_2)\} \\ &\underset{\substack{\uparrow \\ \text{by Lemma 10}}}{\geq} ecc_G(v,s_1) + \min_{i\in[1,\ell]}\{ecc_G(u_i,s_2)\} \\ &\geq ecc_G(v,s_1) + \min_{u\in V}\{ecc_G(u,s_2)\} \\ &\underset{\substack{\uparrow \\ \text{definition of } rad_G(s_2)}}{=} ecc_G(v,s_1) + rad_G(s_2) \\ &\underset{\substack{\uparrow \\ \text{assumption}}}{\geq} ecc_G(v, s_1(s_2+1)) + 1. \end{aligned}$$

[6] It is important to note that for all scans referenced in this proof, which include a BFS procedure of G starting at v and BFS procedures of G_i starting at u_i for $1 \leq i \leq \ell$, we require a consistent order of scanning, i.e., that for a given $1 \leq i \leq \ell$, and vertices $x, x' \in V_i \subseteq V$, if x is scanned before x' in G then x should also be scanned before x' in G_i (and vice versa). This is a valid requirement since for any vertices $y, y' \in V_i \subseteq V$, $d_G(v,y) \leq d_G(v,y')$ if and only if $d_{G_i}(u_i,y) \leq d_{G_i}(u_i,y')$.

Thus, $T(v, ecc_G(v, s_1(s_2+1))+1) \subseteq T(v, ecc_G(v, s_1)+\mu) \subseteq N(v, s_1) \cup \left(\bigcup_{i=1}^{\ell} N_{G_i}(u_i, s_2)\right)$.

Notice that $\ell \leq s_1$, since, by Property 9, $\ell = |L(v, ecc_G(v, s_1))| \leq |T(v, ecc_G(v, s_1))| \leq N(v, s_1) = s_1$. Therefore,

$$\begin{aligned}
|T(v, ecc_G(v, s_1(s_2+1))+1)| &\leq |N(v, s_1) \cup \left(\bigcup_{i=1}^{\ell} N_{G_i}(u_i, s_2)\right)| \\
&\leq |N(v, s_1)| + \sum_{i=1}^{\ell} |N_{G_i}(u_i, s_2)| \\
&\leq s_1 + \ell \cdot s_2 \\
&\leq s_1 + s_1 \cdot s_2 \\
&= s_1(1+s_2) \\
&= |N(v, s_1(s_2+1))|.
\end{aligned}$$

By Corollary 8, it follows that $T(v, ecc_G(v, s_1(s_2+1))+1) \subseteq N(v, s_1(s_2+1))$, which contradicts Property 9. ◀

4 The new ADO

In this section we prove Theorem 4 by introducing a new ADO which uses subquadratic space and produces a $(2, 1-k)$-stretch for graphs for which $\Delta_G \leq O(n^{1/k-\varepsilon})$ for a positive integer k and real constant $0 < \varepsilon \leq 1/k$. The ADO is parameterized by a parameter $0 \leq \alpha < 1/3$ which quantifies the tradeoff between the space and the stretch of the ADO. When $\alpha = 0$ the ADO is very similar to the ADO of Agarwal and Godfrey [4] which uses $\tilde{O}(n^{5/3})$ space and has a $(2,1)$-stretch. For $0 < \alpha < 1/3$, the ADO uses additional space and is able to improve the stretch of the ADO for the family of graphs for which $\Delta_G \leq O(n^{3\alpha/k})$.

4.1 The Construction Algorithm

The description of our construction algorithm follows the notations and definitions described in Section 1.3.1. The construction begins with an algorithm of Thorup and Zwick [33] that computes a set A of size $\tilde{O}(s)$ such that $|B(v)|, |C(v)| \leq 4n/s$, for every $v \in V$. In our case we set $s = n^{2/3+\alpha}$, thus $|A| = \tilde{O}(n^{2/3+\alpha})$ and $|B(v)|, |C(v)| \leq 4n^{1/3-\alpha}$, for every $v \in V$.

For every vertex $v \in V$, the ADO explicitly stores the distances between v and every vertex in $C(N(v, 16\hat{c}\, n^{1/3+2\alpha}))$ for some constant $\hat{c}$ to be decided later. In addition, for every vertex $v \in V$ the ADO stores $p(v)$, $d(v, p(v))$ and the distances between v and every vertex in A.

A distance query between vertices u and v is answered as follows. If one of the following conditions holds (i) $u \in A$ or $v \in A$, (ii) $u \in C(N(v, 16\hat{c}\, n^{1/3+2\alpha}))$ or $v \in C(N(u, 16\hat{c}\, n^{1/3+2\alpha}))$, then the exact distance is returned. Otherwise, the ADO returns $\hat{d}(u, v) = \min\{d(u, p(u)) + d(p(u), v),\ d(u, p(v)) + d(p(v), v)\}$. Notice that the query time is constant.

In Claim 12, we show that the space complexity of the ADO is $\tilde{O}(\hat{c}n^{5/3+\alpha})$ and in Claim 13, we show that the ADO satisfies a $(2, 1 - rad(\hat{c}n^{3\alpha}))$-stretch.

▷ Claim 12. The space complexity of the ADO is $\tilde{O}(\hat{c}n^{5/3+\alpha})$.

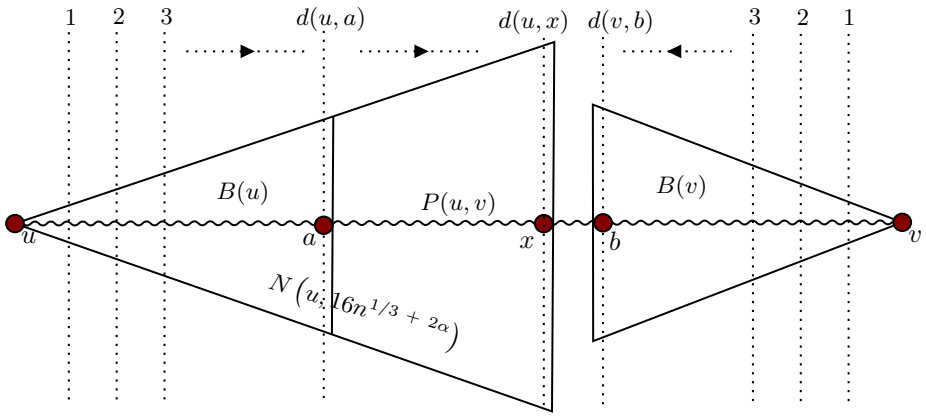


Figure 2 A query for $u, v \in V$ in the case that $N(u, 16\hat{c}\, n^{1/3+2\alpha}) \cap B(v) = \emptyset$. Since $x \in N(u, 16\hat{c}\, n^{1/3+2\alpha})$, $b \in B(v)$ and x and b are both on a shortest path between u and v, it must be that $d(u,x) \leq d(u,b) - 1$.

Proof. Storing $p(v)$, $d(v, p(v))$ and the distances between v and every vertex in A, for all vertices $v \in V$, uses $\tilde{O}(n \cdot n^{2/3+\alpha}) = \tilde{O}(n^{5/3+\alpha})$ space. As mentioned in the construction phase, $|B(v)|, |C(v)| \leq O(n^{1/3-\alpha})$ for every $v \in V$. Thus, storing the distances between every vertex v and $C(N(v, 16\hat{c}\, n^{1/3+2\alpha}))$ requires $O(n \cdot n^{1/3-\alpha} \cdot 16\hat{c}\, n^{1/3+2\alpha}) = \tilde{O}(\hat{c}n^{5/3+\alpha})$ space as well, leading to an overall space complexity of $\tilde{O}(\hat{c}n^{5/3+\alpha})$. ◁

▷ Claim 13. The distance estimation $\hat{d}(u,v)$ returned by the ADO satisfies $d(u,v) \leq \hat{d}(u,v) \leq 2d(u,v) + 1 - rad(\hat{c}n^{3\alpha})$.

Proof. Notice that $d(u,v) \leq \hat{d}(u,v)$ since the ADO always returns a length of some path in the graph between u and v. It is left to show that $\hat{d}(u,v) \leq 2d(u,v) + 1 - rad(\hat{c}n^{3\alpha})$.

If the exact distance is stored in the ADO then $\hat{d}(u,v) = d(u,v)$ and the claim follows. Consider the case that the exact distance is not stored. This implies that $u, v \notin A$ and $v \notin C(N(u, 16\hat{c}\, n^{1/3+2\alpha}))$. Assume towards a contradiction that $N(u, 16\hat{c}\, n^{1/3+2\alpha}) \cap B(v) \neq \emptyset$ and let w be a vertex such that $w \in N(u, 16\hat{c}\, n^{1/3+2\alpha}) \cap B(v)$. From the definitions of bunch and cluster, we have that $w \in B(v)$ if and only if $v \in C(w)$. Thus, $v \in C(w)$, and since $w \in N(u, 16\hat{c}\, n^{1/3+2\alpha})$, it must be that $v \in C(N(u, 16\hat{c}\, n^{1/3+2\alpha}))$ which is a contradiction. Thus, we have that $N(u, 16\hat{c}\, n^{1/3+2\alpha}) \cap B(v) = \emptyset$.

Let $P(u,v)$ be a shortest path between u and v. Let a be the furthest vertex from u in $B(u) \cap P(u,v)$, let x be the furthest vertex from u in $N(u, 16\hat{c}\, n^{1/3+2\alpha}) \cap P(u,v)$ and let b be the furthest vertex from v in $B(v) \cap P(u,v)$ (see Figure 2). Notice that, by definition of x and $ecc(v,s)$, if $T(u, d(u,x)) \subseteq N(u, 16\hat{c}\, n^{1/3+2\alpha})$ then $d(u,x) = ecc(u, 16\hat{c}\, n^{1/3+2\alpha})$ and if $T(u, d(u,x)) \not\subseteq N(u, 16\hat{c}\, n^{1/3+2\alpha})$ then $d(u,x) = ecc(u, 16\hat{c}\, n^{1/3+2\alpha}) + 1$. Thus, we get that $d(u,x) \geq ecc(u, 16\hat{c}\, n^{1/3+2\alpha})$.

Since $N(u, 16\hat{c}\, n^{1/3+2\alpha}) \cap B(v) = \emptyset$, $x \in N(u, 16\hat{c}\, n^{1/3+2\alpha})$, $b \in B(v)$ and x and b are both on a shortest path between u and v, it must be that $d(u,x) \leq d(u,b) - 1$. Since b is on a shortest path between u and v, it holds that $d(u,b) = d(u,v) - d(v,b)$, and so $d(u,x) \leq d(u,v) - d(v,b) - 1$. Since $d(u,x) \geq ecc(u, 16\hat{c}\, n^{1/3+2\alpha})$, it follows that:

$$ecc(u, 16\hat{c}\, n^{1/3+2\alpha}) \leq d(u,v) - d(v,b) - 1. \tag{2}$$

By Lemma 11, $ecc(u, \lceil 4n^{1/3-\alpha} \rceil) + rad(\lceil \hat{c}n^{3\alpha} \rceil) \leq ecc(u, (4n^{1/3-\alpha} + 1)(\hat{c}n^{3\alpha} + 2))$. Since $a \in B(u)$ and $|B(u)| \leq 4n^{1/3-\alpha}$, it follows from the definitions of $ecc(v,s)$ and bunch that $d(u,a) \leq ecc(u, 4n^{1/3-\alpha})$. We have that:

$$\begin{aligned} d(u,a) + rad(\hat{c}n^{3\alpha}) &\leq ecc(u, 4n^{1/3-\alpha}) + rad(\hat{c}n^{3\alpha}) \\ &\leq ecc(u, \left\lceil 4n^{1/3-\alpha} \right\rceil) + rad(\lceil \hat{c}n^{3\alpha} \rceil) \\ &\leq ecc(u, (4n^{1/3-\alpha}+1)(\hat{c}n^{3\alpha}+2)) \\ &\leq ecc(u, 16\hat{c}\, n^{1/3+2\alpha}) \\ &\underset{\text{Equation (2)}}{\leq} d(u,v) - d(v,b) - 1. \end{aligned}$$

Thus, $d(u,a) + d(b,v) \leq d(u,v) - rad(\hat{c}n^{3\alpha}) - 1$. It follows that:

$$2\min\{d(u,a), d(b,v)\} \leq d(u,v) - rad(\hat{c}n^{3\alpha}) - 1. \tag{3}$$

Notice that by the definitions of bunch, a and $p(u)$, it holds that $d(u,p(u)) = d(u,a) + 1$. Similarly, $d(v,p(v)) = d(v,b) + 1$. Thus:

$$\begin{aligned} \hat{d}(u,v) &\leq \min\{d(u,p(u)) + d(p(u),v), d(u,p(v)) + d(p(v),v)\} \\ &\underset{\text{triangle inequallity}}{\leq} \min\{2d(u,p(u)) + d(u,v), 2d(v,p(v)) + d(u,v)\} \\ &\underset{d(u,p(u)) = d(u,a)+1 \text{ and } d(v,p(v)) = d(v,b)+1}{=} \min\{2(d(u,a)+1) + d(u,v), 2(d(v,b)+1) + d(u,v)\} \\ &\leq 2\min\{d(u,a), d(v,b)\} + 2 + d(u,v) \qquad (4) \\ &\underset{\text{Equation (3)}}{\leq} d(u,v) - rad(\hat{c}n^{3\alpha}) + 1 + d(u,v) \\ &= 2d(u,v) + 1 - rad(\hat{c}n^{3\alpha}). \qquad (5) \end{aligned}$$

◁

By combining our ADO construction with Claims 12 and 13 we have proven the following lemma.

▶ **Lemma 14.** *For any graph G with n vertices, real $0 \leq \alpha < \frac{1}{3}$ and constant $\hat{c} \geq 1$, it is possible to construct an ADO that uses $\tilde{O}(\hat{c}n^{\frac{5}{3}+\alpha})$ space and has a $(2, 1 - rad(\hat{c}n^{3\alpha}))$-stretch.*

4.2 Proof of Main Upper Bound Theorem

The following lemma connects Δ_G and $rad(s)$, which is the last ingredient needed for proving Theorem 4.

▶ **Lemma 15.** *Let $G = (V, E)$ be an unweighted undirected graph, with $|V| = n$. For any real s such that $1 \leq s < n$, it holds that $rad_G(s) \geq \lfloor \log_{\Delta_G}(s/2) \rfloor$.*

Proof. For any vertex v and integer $t \geq 1$, $T(v,t)$ cannot include more than $\Delta_G \cdot \sum_{i=0}^{t-1}(\Delta_G - 1)^i$ vertices. Since $\Delta_G, t \geq 1$ we have that $\Delta_G \cdot \sum_{i=0}^{t-1}(\Delta_G - 1)^i \leq 2 \cdot \Delta_G^t$ and so for any integer $t \geq 1$ such that $2 \cdot \Delta_G^t < n$ it must be that $T(v,t) \subseteq N(v, 2 \cdot \Delta_G^t)$. By definition, $ecc(v,s)$ is equal to the largest integer $x \in [0, ecc(v)]$ for which $T(v,x) \subseteq N(v,s)$. Thus, $t \leq ecc(v, 2 \cdot \Delta_G^t)$. Since $t \leq ecc(v, 2 \cdot \Delta_G^t)$ for any vertex v, it follows from the definition of $rad(s) = \min_{v \in V}\{ecc(v,s)\}$ that $t \leq rad(2 \cdot \Delta_G^t)$. Setting $s \geq 2 \cdot \Delta_G^t$, or $t \leq \log_{\Delta_G}(s/2)$, it follows that for any integer t such that $t \leq \log_{\Delta_G}(s/2)$ it must be that $t \leq rad(2 \cdot \Delta_G^t) \leq rad(s)$. Thus, $\lfloor \log_{\Delta_G}(s/2) \rfloor \leq rad(s)$. ◀

Finally, we are ready to prove Theorem 4.

Proof of Theorem 4. It holds that $k = \lfloor \log_{\Delta_G}(\Delta_G^k) \rfloor \leq \lfloor \log_{\Delta_G}(c^k n^{1-k\varepsilon}) \rfloor$, and by Lemma 15, $\lfloor \log_{\Delta_G}(c^k n^{1-k\varepsilon}) \rfloor \leq rad_G(2c^k n^{1-k\varepsilon})$. Thus, the ADO from Lemma 14 constructed for G using $\alpha = \frac{1-k\varepsilon}{3}$ and $\hat{c} = 2c^k$ uses $\tilde{O}(c^k n^{2-\frac{k\varepsilon}{3}})$ space and produces a distance estimation that satisfies $d(u,v) \leq \hat{d}(u,v) \leq \max\{d(u,v), 2d(u,v) + 1 - rad_G(2c^k n^{1-k\varepsilon})\} \leq \max\{d(u,v), 2d(u,v) + 1 - k\}$. ◀

5 Reduction from the Set Intersection Problem

Proof of Lemma 6. Given an instance of Problem 1, we construct a graph G with $n = \tilde{O}(N)$ vertices and $\Delta_G = O(n^{1/k})$, such that a $(k, k+2)$-distinguisher oracle for G solves the instance of Problem 1.

We begin by focusing on a $k+1$ layered graph $\mathcal{L} = (V_{\mathcal{L}}, E_{\mathcal{L}})$, which we call the *infrastructure graph*. The infrastructure graph has three important properties: (*i*) each layer contains N vertices, (*ii*) there is a path of length k from every vertex in the first layer to every vertex in the last layer, and (*iii*) the degree of every vertex is at most $2N^{1/k}$.

We then construct for each $x \in X$ a graph $G_x = (V_x, E_x)$, which is a subgraph of (a copy of) $\mathcal{L}$, by removing some of the edges between the first (last) and second (second to last) layers of $\mathcal{L}$ in a way that expresses which sets contain x and which do not. Finally, we construct the graph G which is *specialized* union of all of the graphs G_x for all $x \in X$, and enables solving the instance of Problem 1 by using a $(k, k+2)$-distinguisher oracle on G.

The infrastructure graph. The infrastructure graph $\mathcal{L}$ is a $k+1$ layered graph where each layer contains N vertices, and each layer of N vertices is locally indexed from 1 to N. The layers are numbered 0 to k.

The edges of $\mathcal{L}$ are defined using the following labels. Assign a *label* $\ell(v)$ to every vertex v in $\mathcal{L}$ which is the k digit representation in base[7] $N^{1/k}$ of the local index (an integer between 1 and N) of v. Then, for every $1 \leq t \leq k$, connect u from layer $t-1$ with v from layer t if and only if the digits of $\ell(u)$ and the digits of $\ell(v)$ all match, except for possibly the t'th digit. It is straightforward to observe (since each digit has $N^{1/k}$ options) that the degree of every vertex in $\mathcal{L}$ is $2N^{1/k}$, except for the vertices in the first and last layers which have degree $N^{1/k}$. The following claim shows that there is a path of length k from every vertex in the first layer and every vertex in the last layer.

▷ Claim 16. Let v be a vertex in the first layer of $\mathcal{L}$ and let u be a vertex in the last layer of $\mathcal{L}$. Then there exists a path of length k from u to v in $\mathcal{L}$.

Proof. We describe the path of length k between u and v. For any $0 \leq t \leq k$, consider the vertex w_t in layer t of $\mathcal{L}$ with the label of the following form: the first t digits are the first t digits of $\ell(u)$, and the last $k-t$ digits are the last $k-t$ digits of $\ell(v)$. Thus, for $0 \leq t \leq k-1$, the edge (w_t, w_{t+1}) is in $\mathcal{L}$ since $\ell(w_t)$ and $\ell(w_{t+1})$ are the same, except for possibly the $(t+1)$-th digit. The set of edges which we described form a path of length k between v and u. ◁

[7] We assume for convenience that $N^{1/k}$ is an integer, since otherwise, one can increase N slightly without affecting the asymptotic complexities.

Constructing G_x. We construct V_x by making copies of each vertex in $V_{\mathcal{L}}$. Denote the first layer of $\mathcal{L}$ by $V_L = \{v_1, \ldots, v_N\}$ and the last layer by $V_R = \{u_1, \ldots, u_N\}$. Let $\hat{E}_x = \{(v_i, w) | x \notin S_i \wedge (v_i, w) \in E_{\mathcal{L}}\} \cup \{(u_i, w) | x \notin S_i \wedge (u_i, w) \in E_{\mathcal{L}}\}$. Thus, $\hat{E}_x$ is the set of edges in $\mathcal{L}$ that touch vertices in the first or last layers whose index corresponds to the index of sets that do not contain x. We construct E_x by making copies of all edges in $E_{\mathcal{L}} \setminus \hat{E}_x$. The reason for removing the edges in $\hat{E}_x$ is so that vertices in the first and last layers of G_x whose edges are in $\hat{E}_x$ are not connected to any other vertex in G_x. Thus, for each v_i (u_j) in the first (last) layer of G_x, $x \in S_i$ if and only if there are edges between v_i (u_j) and the second (second to last) layer in G_x. By Claim 16, if $S_i \cap S_j \neq \emptyset$ then there exists a path of length k between v_i and u_j, and otherwise, there is no path in G_x between v_i and u_j. Finally, since G_x is a partial copy of $\mathcal{L}$, the maximum degree in G_x is $2N^{1/k}$.

Constructing G. We construct the $k+1$ layered graph G by performing the following special union of G_x for all x: for $1 \leq t \leq k-1$ the t'th layer of G is the union of the t'th layer of all of the G_x graphs taken over all $x \in X$. Thus, each of the $k-1$ inner layers (excluding the first and last layer of G) has $|X|N$ vertices. For the first (last) layer G, instead of taking the union of all of the first (last) layers from all of the G_x graphs, we merge them all into one layer of N vertices. So the i'th vertex in the first (last) layer of G is a vertex obtained by merging the i'th vertex in the first (last) layer of every G_x, for all $x \in X$. Thus, the first and last layers of G_x contain N vertices each. Since the vertices in the first and last layer of G correspond directly to the vertices V_L and V_R in $\mathcal{L}$, respectively, we treat the first layer of G as $V_L = \{v_1, \ldots v_n\}$ and the last layer of G by $V_R = \{u_1, \ldots, u_N\}$. Thus, each node in $V_L \cup V_R$ has maximum degree at most $|X|N^{1/k} = \tilde{O}(N^{1/k})$.

Answering a set intersection query. Notice that for a set intersection query between S_i and S_j, if $S_i \cap S_j \neq \emptyset$, then there exists some $x \in S_i \cap S_j$, and since G contains G_x as a subgraph, the distance between v_i and u_j is at most (and actually exactly) k. On the other hand, if there exists a path P of length k between v_i and u_j, then by the construction of G, P must be completely contained within some G_x for some $x \in X$. By the construction of G_x, and specifically E_x, the existence of P in G_x implies that $x \in S_i$ and $x \in S_j$. So, in such a case $S_i \cap S_j \neq \emptyset$.

Notice that, since G is a $k+1$ layered graph, any path between a vertex in the first layer and a vertex in the last layer must be of length $k + 2q$ for some integer $q \geq 0$. Thus, to answer a set intersection query, it suffices to establish whether the distance in G between v_i and u_j is either k or at least $k+2$, which the $(k, k+2)$-distinguisher oracle returns in constant time.

Analysis. We conclude that a $(k, k+2)$-distinguisher oracle for graphs with $n = \tilde{O}(N)$ vertices and maximum degree $\tilde{O}(n^{1/k})$ also solves the instance of Problem 1 (of size N). Thus, according to Hypothesis 2, an ADO for graphs with $n = \tilde{O}(N)$ vertices, for which the maximum degree is $\tilde{O}(n^{1/k})$, must use $\tilde{O}(N^2) = \tilde{O}(n^2)$ space. We note that the maximum degree can be reduced to $O(n^{1/k})$ by artificially adding $\tilde{O}(n) = \tilde{O}(N)$ isolated vertices to G. ◀

6 Conclusions and Open Problems

In this paper we provide an algorithm (Theorem 4) and a conditional lower bound (Theorem 5) for subquadratic space ADOs as a function of the maximum degree. As mentioned in Section 1, the case of $k = 2$ in Theorem 5 essentially matches the upper bound of Theorem 4. Although

the upper bound from Theorem 4 improves the additive approximation of the ADO for larger values of k, a natural remaining open problem is whether it is also possible to reduce the multiplicative approximation of the ADO and design a sub-$\frac{k+2}{k}$ stretch ADO for graphs with maximum degree $\Theta(n^{\frac{1}{k}-\Omega(1)})$ while using subquadratic space for integers $k \geq 3$.

References

1 Amir Abboud, Karl Bringmann, Seri Khoury, and Or Zamir. Hardness of approximation in p via short cycle removal: cycle detection, distance oracles, and beyond. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1487–1500. ACM, 2022. doi:10.1145/3519935.3520066.

2 Ittai Abraham and Cyril Gavoille. On approximate distance labels and routing schemes with affine stretch. In David Peleg, editor, *Distributed Computing - 25th International Symposium, DISC 2011, Rome, Italy, September 20-22, 2011. Proceedings*, volume 6950 of *Lecture Notes in Computer Science*, pages 404–415. Springer, 2011. doi:10.1007/978-3-642-24100-0_39.

3 Rachit Agarwal. The space-stretch-time tradeoff in distance oracles. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 49–60. Springer, 2014. doi:10.1007/978-3-662-44777-2_5.

4 Rachit Agarwal and Philip Brighten Godfrey. Brief announcement: a simple stretch 2 distance oracle. In Panagiota Fatourou and Gadi Taubenfeld, editors, *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 110–112. ACM, 2013. doi:10.1145/2484239.2484277.

5 Rachit Agarwal and Philip Brighten Godfrey. Distance oracles for stretch less than 2. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 526–538. SIAM, 2013. doi:10.1137/1.9781611973105.38.

6 Rachit Agarwal, Philip Brighten Godfrey, and Sariel Har-Peled. Approximate distance queries and compact routing in sparse graphs. In *INFOCOM 2011. 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 10-15 April 2011, Shanghai, China*, pages 1754–1762. IEEE, 2011. doi:10.1109/INFCOM.2011.5934973.

7 Maor Akav and Liam Roditty. An almost 2-approximation for all-pairs of shortest paths in subquadratic time. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1–11. SIAM, 2020. doi:10.1137/1.9781611975994.1.

8 Surender Baswana, Vishrut Goyal, and Sandeep Sen. All-pairs nearly 2-approximate shortest paths in I time. *Theor. Comput. Sci.*, 410(1):84–93, 2009. doi:10.1016/J.TCS.2008.10.018.

9 Surender Baswana and Telikepalli Kavitha. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 591–602. IEEE, 2006.

10 Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

11 Davide Bilò, Shiri Chechik, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, and Martin Schirneck. Improved approximate distance oracles: Bypassing the thorup-zwick bound in dense graphs. *arXiv preprint*, 2023. arXiv:2307.11677.

12 Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.*, 39(5):2075–2089, 2010. doi:10.1137/08071990X.

13 Shiri Chechik. Approximate distance oracles with constant query time. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 654–663. ACM, 2014. doi:10.1145/2591796.2591801.

14 Shiri Chechik. Approximate distance oracles with improved bounds. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 1–10, 2015.

15 Shiri Chechik and Tianyi Zhang. Nearly 2-approximate distance oracles in subquadratic time. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 551–580. SIAM, 2022. doi:10.1137/1.9781611977073.26.

16 Boris V. Cherkassky, Andrew V. Goldberg, and Tomasz Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Math. Program.*, 73:129–174, 1996. doi:10.1007/BF02592101.

17 Hagai Cohen and Ely Porat. On the hardness of distance oracle for sparse graph. *CoRR*, abs/1006.1117, 2010. arXiv:1006.1117.

18 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. doi:10.1007/BF01386390.

19 Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM J. Comput.*, 29(5):1740–1759, 2000. doi:10.1137/S0097539797327908.

20 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. doi:10.1145/28869.28874.

21 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1272–1287. SIAM, 2016. doi:10.1137/1.9781611974331.CH89.

22 Amgad Madkour, Walid G. Aref, Faizan Ur Rehman, Mohamed Abdur Rahman, and Saleh M. Basalamah. A survey of shortest-path algorithms. *CoRR*, abs/1705.02044, 2017. arXiv:1705.02044.

23 Manor Mendel and Assaf Naor. Ramsey partitions and proximity data structures. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 109–118. IEEE Computer Society, 2006. doi:10.1109/FOCS.2006.65.

24 Ely Porat and Liam Roditty. Preprocess, set, query! *Algorithmica*, 67(4):516–528, 2013.

25 Mihai Puatracscu. Towards polynomial lower bounds for dynamic problems. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 603–610. ACM, 2010. doi:10.1145/1806689.1806772.

26 Mihai Puatracscu. Unifying the landscape of cell-probe lower bounds. *SIAM J. Comput.*, 40(3):827–847, 2011. doi:10.1137/09075336X.

27 Mihai Puatracscu and Liam Roditty. Distance oracles beyond the thorup-zwick bound. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 815–823. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.83.

28 Mihai Puatracscu, Liam Roditty, and Mikkel Thorup. A new infinity of distance oracles for sparse graphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 738–747. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.44.

29 Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 261–272. Springer, 2005. doi:10.1007/11523468_22.

30 Liam Roditty and Roei Tov. Approximate distance oracles with improved stretch for sparse graphs. *Theor. Comput. Sci.*, 943:89–101, 2023. doi:10.1016/J.TCS.2022.11.016.

31 Christian Sommer. Shortest-path queries in static networks. *ACM Comput. Surv.*, 46(4):45:1–45:31, 2014. doi:10.1145/2530531.

32 Christian Sommer. All-pairs approximate shortest paths and distance oracle preprocessing. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 55:1–55:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.ICALP.2016.55.

33 Mikkel Thorup and Uri Zwick. Compact routing schemes. In Arnold L. Rosenberg, editor, *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA 2001, Heraklion, Crete Island, Greece, July 4-6, 2001*, pages 1–10. ACM, 2001. doi:10.1145/378580.378581.

34 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.

35 Virginia Vassilevska Williams and Yinzhan Xu. Monochromatic triangles, triangle listing and APSP. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 786–797. IEEE, 2020. doi:10.1109/FOCS46700.2020.00078.

36 Christian Wulff-Nilsen. Approximate distance oracles with improved query time. In *Encyclopedia of Algorithms*, pages 94–97, 2016. doi:10.1007/978-1-4939-2864-4_568.

37 Uri Zwick. Exact and approximate distances in graphs - A survey. In Friedhelm Meyer auf der Heide, editor, *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, volume 2161 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 2001. doi:10.1007/3-540-44676-1_3.

Lipschitz Continuous Allocations for Optimization Games

Soh Kumabe
CyberAgent, Tokyo, Japan

Yuichi Yoshida
National Institute of Informatics, Tokyo, Japan

Abstract

In cooperative game theory, the primary focus is the equitable allocation of payoffs or costs among agents. However, in the practical applications of cooperative games, accurately representing games is challenging. In such cases, using an allocation method sensitive to small perturbations in the game can lead to various problems, including dissatisfaction among agents and the potential for manipulation by agents seeking to maximize their own benefits. Therefore, the allocation method must be robust against game perturbations.

In this study, we explore optimization games, in which the value of the characteristic function is provided as the optimal value of an optimization problem. To assess the robustness of the allocation methods, we use the Lipschitz constant, which quantifies the extent of change in the allocation vector in response to a unit perturbation in the weight vector of the underlying problem. Thereafter, we provide an algorithm for the matching game that returns an allocation belonging to the $\left(\frac{1}{2}-\epsilon\right)$-approximate core with Lipschitz constant $O(\epsilon^{-1})$. Additionally, we provide an algorithm for a minimum spanning tree game that returns an allocation belonging to the 4-approximate core with a constant Lipschitz constant.

The Shapley value is a popular allocation that satisfies several desirable properties. Therefore, we investigate the robustness of the Shapley value. We demonstrate that the Lipschitz constant of the Shapley value for the minimum spanning tree is constant, whereas that for the matching game is $\Omega(\log n)$, where n denotes the number of vertices.

2012 ACM Subject Classification Mathematics of computing → Approximation algorithms

Keywords and phrases Cooperative Games, Lipschitz Continuity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.102

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2405.11889

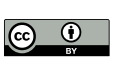
Funding *Soh Kumabe*: Supported by JSPS KAKENHI Grant Number JP21J20547.
Most of this work was done while the author was a student at The University of Tokyo.
Yuichi Yoshida: Supported by JSPS KAKENHI Grant Number JP24K02903.

1 Introduction

1.1 Background and Motivation

Cooperative games model decision-making scenarios in which multiple agents can achieve greater benefits through cooperation. A primary concern in cooperative game theory is the allocation of payoffs or costs provided by the grand coalition in an acceptable manner to each agent. Among the cooperative games, those defined by optimization problems corresponding to the set of agents involved are known as *optimization games* [8].

Consider the following well-known examples formulated by the matching game (MG) [9, 3] and the minimum spanning tree game (MSTG) [6]:



MG: Members of a tennis club form pairs for a doubles tournament. Each pair of players has a predicted value for the prize money they would win if they teamed up. How should the total prize money won by all pairs be distributed among members?

MSTG: Multiple facilities cooperate to construct a power grid to receive electricity from a power plant. Each potential power line has a predetermined installation cost. When constructing a power grid that ensures electricity distribution to all facilities, what cost should each facility bear?

In these examples, the characteristic function of the game often contains errors, or can be manipulated through deliberate misreporting. For instance,

MG: It is challenging to accurately predict the compatibility of pairs who have never teamed up before. Moreover, pairs known to work well may hide this fact.

MSTG: Costs for power line installation might be misestimated owing to unforeseen terrain or geological conditions caused by natural disasters or inadequate surveys, land rights, or landscape regulations. Facilities may conceal these issues.

In such uncertain situations, allocations that drastically change with slight perturbations in the game can lead to problems, such as

MG: If minor estimation errors in predicted prize money significantly change the benefits for each player, players might not be satisfied with the allocation. In addition, when someone falsely reports a substantial increase in their gain, it becomes difficult to prove the intent of the manipulation if the degree of falsehood is small.

MSTG: Facilities might not accept the allocation if trivial issues in power line construction significantly increase their cost burden. Moreover, if such issues are used to significantly reduce their own costs or increase those of competing facilities, several minor construction issues may be concealed, leading to risk management problems.

To avoid these issues, it is desirable to use allocations that are robust against perturbations in real-world cooperative games.

Before proceeding, we define some terms in cooperative game theory. The *cooperative game* (V, ν) is defined as a pair consisting of a set of *agents* V and a *characteristic function* $\nu\colon 2^V \to \mathbb{R}_{\geq 0}$ representing the payoff or cost obtained when subsets of agents cooperate. An *optimization game* is defined by the optimization problem $\mathcal{P}$ in a discrete structure. The *$\mathcal{P}$-game* (G, w) is defined from a pair of structures G consisting of *agent set* V, *edge set* E, and a weight vector $w \in \mathbb{R}^E_{\geq 0}$. In several games, G is a graph (V, E). For each subset $S \subseteq V$, the characteristic function value $\nu(S)$ is defined as the optimum value of $\mathcal{P}$ on the substructure corresponding to S (e.g., the subgraph induced by S) with respect to weight vector w. An optimization game is a *welfare allocation game* (resp., *cost allocation game*) if each agent intends to maximize (resp., minimize) the value allocated to it.

Now, we formally introduce matching games and minimum spanning tree games, as discussed in previous examples.

▶ **Definition 1** (Matching game [3, 8]). *Let $G = (V, E)$ be an undirected graph. For a vertex set $S \subseteq V$ and an edge weight vector $w \in \mathbb{R}^E_{\geq 0}$, let $\mathrm{OPT}(S, w)$ denote the maximum matching weight in $G[S]$ with respect to weight w, where $G[S]$ represents the subgraph of G induced by S. The* matching game *of G with respect to weight w is defined as (V, ν), where $\nu(S) = \mathrm{OPT}(S, w)$. The matching game is a welfare-allocation game.*

▶ **Definition 2** (Minimum spanning tree game [4, 6]). *Let $G = (V \cup \{r\}, E)$ be an undirected graph. For a vertex set $S \subseteq V$ and edge weight vector $w \in \mathbb{R}^E_{\geq 0}$, let $\mathrm{OPT}(S, w)$ denote the minimum weight of a spanning tree in $G[S \cup \{r\}]$ with respect to weight w. Notably, vertex r does not correspond to the agent. To ensure that the characteristic function has finite values, we assume that edge (r, v) exists for all $v \in V$. The* minimum spanning tree game *of G with respect to weight w is defined as (V, ν), where $\nu(S) = \mathrm{OPT}(S, w)$. The minimum spanning tree game is a cost-allocation game.*

Let us return to the discussion of the robustness of the allocation. To measure the robustness of the allocation, we introduce the concept of *Lipschitz continuity* in the allocation of these games, analogous to the introduction of Kumabe and Yoshida [23] for discrete optimization problems under the same name. Algorithm $\mathcal{A}$ that takes a weight vector $w \in \mathbb{R}^E$ and returns an allocation $x \in \mathbb{R}^V_{\geq 0}$ is *L-Lipschitz* or has *Lipschitz constant* L if:

$$\sup_{\substack{w,w' \in \mathbb{R}^E_{\geq 0} \\ w \neq w'}} \frac{\|\mathcal{A}(w) - \mathcal{A}(w')\|_1}{\|w - w'\|_1} \leq L. \tag{1}$$

If the Lipschitz constant of the allocation method is small, the change in the allocation in response to a unit change in the weight vector is guaranteed to be small. Employing such an allocation method can resolve these problems as follows:

MG: Minor mistakes in estimating the compatibility of pairs will not significantly affect the overall distribution of prize money, making it easier for agents to accept the allocation. Additionally, substantial misreporting is necessary to increase benefits significantly through declaration adjustments, making schemes more likely to be exposed.

MSTG: Minor issues related to the installation of power lines will not cause significant fluctuations in the cost burden for each facility, making it easier for them to accept their share of the costs. Additionally, because the loss incurred by reporting such issues is small, the likelihood of these issues being concealed is reduced.

1.1.1 The core

The *core* is the most fundamental solution concept in cooperative game theory. An allocation vector $x \in \mathbb{R}^V$ is in the core of (welfare allocation) cooperative game (V, ν) if:

$$\sum_{v \in S} x_v \geq \nu(S) \quad (S \subsetneq V), \quad \sum_{v \in V} x_v = \nu(V). \tag{2}$$

Similarly, an allocation vector $x \in \mathbb{R}^V$ is in the core of (cost allocation) cooperative game (V, ν) if:

$$\sum_{v \in S} x_v \leq \nu(S) \quad (S \subsetneq V), \quad \sum_{v \in V} x_v = \nu(V). \tag{3}$$

Because the core is one of the most fundamental solution concepts, it is natural to desire a Lipschitz continuous algorithm $\mathcal{A}$ that takes a weight vector as the input and returns an allocation belonging to the core. However, this can only be achieved in a limited number of situations for two reasons. First, a non-empty core does not necessarily exist in all games. Second, even for games in which the core exists in all instances, there is no guarantee that a vector from the core can be selected such that the Lipschitz constant of $\mathcal{A}$ remains small.

To examine the second reason, we consider the *assignment game* introduced by Shapley and Shubik [29], which is a special case of the matching game in which the underlying graph is bipartite and known to have a non-empty core [8, 29].

▶ **Example 3.** Let n be an odd number greater than or equal to 5. Let us consider a path $G = (V, E)$ consisting of n vertices labeled sequentially as $v_1, \ldots, v_n$. The weight vectors $w, w' \in \mathbb{R}^E_{\geq 0}$ are defined as follows:

$$w_{(v_i, v_{i+1})} = 1 \quad (i = 1, \ldots, n-1), \qquad w'_{(v_i, v_{i+1})} = \begin{cases} 1 & \text{if } 2 \leq i \leq n-2 \\ 0 & \text{otherwise} \end{cases}$$

In this case, the allocations x, x' belonging to the core of the assignment game for w and w' are both unique and obtained as follows:

$$x_i = \begin{cases} 1 & \text{if } i \text{ is even} \\ 0 & \text{otherwise} \end{cases} \qquad x'_i = \begin{cases} 1 & \text{if } i \text{ is odd and } i \notin \{1, n\} \\ 0 & \text{otherwise} \end{cases}$$

If $\mathcal{A}$ is an algorithm that takes a weight vector and returns a vector in the core of the game, then it must satisfy:

$$\frac{\|\mathcal{A}(w) - \mathcal{A}(w')\|_1}{\|w - w'\|_1} = \frac{n-2}{2} = \Omega(n)$$

Thus, algorithm $\mathcal{A}$ must have a large Lipschitz constant, $\Omega(n)$.

Therefore, we consider providing a Lipschitz continuous algorithm that outputs allocations that satisfy some looser solution concepts. The least core [28] is one such solution concept that can be defined even for games in which a core does not exist. However, this does not resolve the problem of a large Lipschitz constant (for instance, in the game in Example 3, the core and the least core coincide, resulting in a Lipschitz constant of $\Omega(n)$). Instead, we consider the approximate core, which is a solution concept that multiplicatively relaxes the constraints of partial coalitions.

1.1.2 Approximate Core

The approximate core was introduced by Faigle and Kern [11] as a useful solution concept for games where the core is empty. Intuitively, the approximate core represents the core when the constraints for partial coalitions are relaxed by a factor α. For $\alpha \leq 1$, the allocation vector $x \in \mathbb{R}^V$ is in the *α-(approximate) core* of the (welfare allocation) cooperative game (V, ν) if:

$$\sum_{v \in S} x_v \geq \alpha \nu(S) \quad (S \subsetneq V), \quad \sum_{v \in V} x_v = \nu(V).$$

Similarly, for $\alpha \geq 1$, the allocation vector $x \in \mathbb{R}^V$ is in the *α-(approximate) core* of the (cost allocation) cooperative game (V, ν) if:

$$\sum_{v \in S} x_v \leq \alpha \nu(S) \quad (S \subsetneq V), \quad \sum_{v \in V} x_v = \nu(V).$$

In this study, we provide algorithms with small Lipschitz constants that return an α-approximate core for some constant α in several optimization games. For the matching game, we obtain the following:

▶ **Theorem 4.** *Let $\epsilon \in \left(0, \frac{1}{2}\right]$. For the matching game, there is a polynomial-time algorithm with a Lipschitz constant $O(\epsilon^{-1})$ that returns $\left(\frac{1}{2} - \epsilon\right)$-approximate core allocation.*

Note that Faigle and Kern [11] and Vazirani [32] showed that the $\frac{2}{3}$-approximate core of the matching game is non-empty, and that this is also a tight bound. However, their allocation, which is constructed using the optimal solution of the matching LP, is not Lipschitz continuous. Our allocation compromises the core approximability by $\frac{1}{6} + \epsilon$ to ensure Lipschitz continuity.

Our allocation was inspired by the Lipschitz continuous algorithm for the maximum weight-matching problem proposed by Kumabe and Yoshida [23]. Similarly, the proposed algorithm is based on the greedy method. We emphasize our results because their algorithm

reduces the approximation ratio to $\frac{1}{8} - \epsilon$ to obtain a Lipschitz constant of $O(\epsilon^{-1})$. Although the definitions of Lipschitz continuity and the approximation ratio for discrete algorithms differ from those for allocations, making a simple comparison infeasible, our analysis is simpler and offers better approximation guarantees.

For the minimum spanning tree game, we have the following.

▶ **Theorem 5.** *For the minimum spanning tree game, there is a polynomial-time algorithm with Lipschitz constant $O(1)$ that returns 4-approximate core allocation.*

The core of the minimum spanning tree game is non-empty; an allocation by Bird [4] is known to belong to the core and can be computed in polynomial time. However, it is not Lipschitz continuous. As in the matching game, our allocation compromises the core approximability to ensure Lipschitz continuity.

The original motivation for the approximate core was to provide useful solution concepts for games with non-empty cores. Therefore, no studies have been conducted on the approximate core of the minimum spanning tree game. In this regard, our setting demonstrates the usefulness of considering an approximate core, even for games with a non-empty core.

1.1.3 Shapley Value

Let $\mathfrak{S}_V$ be the set of all permutations over V. For $\sigma \in \mathfrak{S}_V$ and $v \in V$, let $x_{\sigma,v} = \nu(\{\sigma(1), \dots, \sigma(k)\}) - \nu(\{\sigma(1), \dots, \sigma(k-1)\})$, where k is the integer with $\sigma(k) = v$. The *Shapley value* [27] of game (V, ν) is the vector s defined by:

$$s_v = \frac{1}{|V|!} \sum_{\sigma \in \mathfrak{S}_V} x_{\sigma,v}.$$

The Shapley value does not necessarily belong to the core, even if it exists, and the computation is #P-hard in most optimization games. However, they exhibit various desirable properties and have a wide range of applications [16]. Therefore, investigating the Lipschitz continuity of Shapley values is a natural task.

Considering its various properties, it is natural to expect the Shapley value to always have a small Lipschitz constant for general optimization games. Conversely, given the difficulty in computing it, it is natural to anticipate that it may not have a bounded Lipschitz constant. However, neither was accurate. Specifically, we demonstrate that whether the Shapley value in optimization games has a small Lipschitz constant depends on the game.

▶ **Theorem 6.** *There is a graph G such that the Shapley value of the matching game on G has a Lipschitz constant $\Omega(\log n)$.*

▶ **Theorem 7.** *The Shapley value of the minimum spanning tree game has a Lipschitz constant of 2.*

Notably, the computation of the Shapley value for the minimum spanning tree game is #P-hard [1]. The result of Theorem 7 is particularly interesting because it shows a value that is computationally difficult to calculate may still have a small Lipschitz constant.

1.2 Related Work

1.2.1 Optimization Games

The history of optimization games begins with the *assignment game* proposed by Shapley and Shubik [29]. They showed that the core of the assignment game is represented by the optimal solution of dual linear programming and is always non-empty. Deng, Ibaraki, and

Nagamochi [8] defined a class of games in which the characteristic function is represented by integer programming, and discussed the core structures of such games. In particular, for the matching game, they proved that the core is not always non-empty, but that the core non-emptiness problem, core membership problem, and that of generating a vector in the core if it is non-empty are solvable in polynomial time. Aziz and de Keijzer [2] proved that computing the Shapley value of the matching game is #P-hard. Additionally, studies on the generalization of matching games such as the *hypergraph matching game* [7, 8, 20] and *b-matching game* [5, 19, 30, 34], are being conducted. For a more detailed survey of the algorithmic aspects of matching games, see [3].

The minimum spanning tree game was proposed by Claus and Kleitman [6]. Bird [4] later proposed an allocation defined as follows: Regarding the minimum spanning tree of a given graph as a rooted tree rooted at r, each agent corresponding to a vertex v is allocated a cost equal to the weight of the edge from v to its parent. Granot and Huberman [14] proved this allocation is in the core. Unfortunately, it is not (Lipschitz) continuous. Faigle et al. [13] proved that the core membership problem for the minimum spanning tree game is coNP-hard. Ando [1] proved that the Shapley value of the minimum spanning tree game is #P-hard.

The concept of an approximate core was introduced by Faigle and Kern [11] as a useful solution for games where the core is empty. Their work discussed the approximate core allocations for several optimization games, including the matching game. In particular, for the matching game, they constructed a $\frac{2}{3}$-core allocation based on LP relaxation. Subsequently, extensive studies have been conducted to determine the best α for which the α-core is always non-empty in various optimization games in which the core can be empty, such as the traveling salesman game [10, 11] and bin-packing games [11, 12, 17, 18, 26, 33]. Recently, following the rediscovery of Faigle and Kern's results by Vazirani [32], approximate cores have been derived for several optimization games such as the b-matching game [34] and the edge cover game [24].

Notably, in the definition by Faigle and Kern [11] and in subsequent papers, the right-hand side of Equations (2) or (3) being $(1 \pm \epsilon)\nu(S)$ is referred to as the ϵ-core. However, following the conventions in the field of discrete optimization and the study by Vazirani [32] and subsequent studies [24, 34], we adopt the definitions of Equations (2) and (3). An ϵ-core in our definition corresponds to a $|1 - \epsilon|$-core in Faigle and Kern's definition.

1.2.2 Lipschitz Continuity of Discrete Algorithms

Inspired by recent studies on the average sensitivity [15, 21, 25, 31, 35] for *unweighted* discrete optimization problems, Kumabe and Yoshida [23] introduced the Lipschitz continuity of a randomized algorithm $\mathcal{A}$ for *weighted* discrete optimization problems as follows:

$$\sup_{\substack{w,w' \in \mathbb{R}^V_{\geq 0}, \\ w \neq w'}} \frac{\min_{\mathcal{D} \in \Pi(\mathcal{A}(G,w), \mathcal{A}(G,w'))} \mathbf{E}_{(S,S') \sim \mathcal{D}} \left[d_{\mathrm{w}} \left((S, w), (S', w') \right) \right]}{\|w - w'\|_1}, \tag{4}$$

where $\mathcal{A}(G, w)$ represents the output distribution of algorithm $\mathcal{A}$ for input G with weight vector $w \in \mathbb{R}^V_{\geq 0}$, $\Pi(X, X')$ denotes the set of all joint distributions for random variables X and X', and

$$d_{\mathrm{w}}((S, w), (S', w')) = \left\| \sum_{v \in S} w_v \mathbf{1}_v - \sum_{v \in S'} w'_v \mathbf{1}_v \right\|_1 = \sum_{v \in S \cap S'} |w_v - w'_v| + \sum_{v \in S \setminus S'} w_v + \sum_{v \in S' \setminus S} w'_v.$$

They also proposed algorithms with small Lipschitz constants for the minimum spanning tree, shortest path, and maximum weight matching problems. In a subsequent study [22], the authors obtained algorithms with small Lipschitz constants for the minimum weight vertex cover, minimum weight set cover, and feedback vertex set problems.

In discrete optimization, the outputs of the algorithms are discrete sets. Thus, deterministic algorithms cannot be Lipschitz continuous; they consider randomized algorithms and adopt the earth mover's distance in the numerator of (4). In contrast, the outputs of our algorithms are allocations of continuous values, allowing deterministic algorithms to be Lipschitz continuous and randomized algorithms to be derandomized by taking the expectation. Therefore, in our setting, we can use a simple definition of the deterministic algorithms expressed in Equation (1).

1.3 Organization

The remainder of this paper is organized as follows: In Section 2, we provide several useful lemmas to analyze Lipschitz continuity of allocations. In Section 3, we prove Theorem 4 by providing a Lipschitz continuous polynomial-time algorithm that returns an approximate core allocation to the matching game. In Section 4, we prove Theorem 5 by providing a Lipschitz continuous polynomial-time algorithm that returns an approximate core allocation for the minimum spanning tree game. The proofs of Theorem 6 and Theorem 7 are given in the full version.

2 Basic Facts

In this section, we provide useful lemmas to analyze the core approximability and Lipschitz continuity of our allocations. The following lemma is useful for obtaining the Lipschitz constant: We omit this proof because it is similar to Lemma 1.7 of [23].

▶ **Lemma 8.** *Let $(G = (V, E), w)$ be an optimization game and $\mathcal{A}$ be an algorithm that takes a pair (G, w) and outputs an allocation. Suppose that there exist some $c > 0$ and $L > 0$ such that*

$$\|(\mathcal{A}(G, w), \mathcal{A}(G, w + \delta \mathbf{1}_e)\|_1 \leq \delta L$$

holds for any $e \in E$, $w \in \mathbb{R}^E_{\geq 0}$, and $\delta > 0$ with either $\delta \leq c \cdot w_e$ or $w_e = 0$, where $\mathbf{1}_e$ represents the characteristic vector of e. Then, $\mathcal{A}$ has a Lipschitz constant L.

In the analysis of the core approximability and Lipschitz continuity of allocations, we can simplify the discussion by ignoring the constraint $\sum_{v \in V} x_v = \nu(V)$ for a grand coalition. Therefore, we require a method to obtain an allocation with a bounded Lipschitz constant from vectors that satisfy only the constraints for partial coalitions with a bounded Lipschitz constant. To achieve this, we require a mild assumption in the game. An optimization game $(G = (V, E), w)$ is said to be *reasonable* if the inequality $|\nu(V, w) - \nu(V, w')| \leq \|w - w'\|_1$ holds for all $w, w' \in \mathbb{R}_{\geq} 0^E$. Notably, this is a fair assumption. For instance, games defined for optimization problems in the form of

$$\max \text{ or } \min \sum_{e \in X} w_e, \text{ subject to } X \in \mathcal{F}$$

for $\mathcal{F} \subseteq 2^E$ are all reasonable. The next lemma applies to welfare allocation games, such as the matching game. Due to the space limit, the proofs of the next two lemmas are given in the full version.

Algorithm 1 Lipschitz continuous allocation for the matching game.

Procedure MatchingGame(G, w, b, α)
  Input: A graph $G = (V, E)$, where the edge set is indexed with integers in $\{1, 2, \ldots, |E|\}$, a weight vector $w \in \mathbb{R}^E_{\geq 0}$, $b \in [0, 1]$, and $\alpha \in (1, 2]$.
  For each $e \in E$ with $w_e > 0$, let $\widehat{w}_e \leftarrow \alpha^{i_e+1+b}$, where i_e is the unique integer such that $\alpha^{i_e+b} \leq w_e < \alpha^{i_e+1+b}$;
  $z \leftarrow 0^V$, $M \leftarrow \emptyset$;
  for $e \in E$ *in descending order of* w_e*, where ties are broken according to their indices* **do**
    if *none of the two endpoints of* e *are covered by* M **then**
      Add e to M;
      $z_v \leftarrow \widehat{w}_e$ for each endpoint v of e;
  return z;

▶ **Lemma 9.** *Let $D \geq 1$. Let $\mathcal{A}$ be an algorithm that takes a weight vector $w \in \mathbb{R}^E_{\geq 0}$ and returns an allocation $\mathcal{A}(w) \in \mathbb{R}^V_{\geq 0}$ for a reasonable welfare allocation game. Assume $\mathcal{A}$ satisfies $\|\mathcal{A}(w)\|_1 \leq D\nu(V, w)$, $\sum_{v \in S} \mathcal{A}(w)_v \geq \nu(S, w)$ for all weight vector w and $S \subseteq V$, and $\|\mathcal{A}(w) - \mathcal{A}(w')\|_1 \leq L\|w - w'\|_1$ for all two weight vectors w and w'. Then, there is an algorithm that returns $\frac{1}{D}$-approximate core allocation with Lipschitz constant $2L + 1$.*

The next lemma applies to cost allocation games, such as the minimum spanning tree game.

▶ **Lemma 10.** *Let $D \geq 1$. Let $\mathcal{A}$ be an algorithm that takes a weight vector $w \in \mathbb{R}^E_{\geq 0}$ and returns an allocation $\mathcal{A}(w) \in \mathbb{R}^V_{\geq 0}$ for a reasonable cost allocation game. Assume $\mathcal{A}$ satisfies $\|\mathcal{A}(w)\|_1 \geq \nu(V, w)$, $\sum_{v \in S} \mathcal{A}(w)_v \leq D\nu(S, w)$ for all weight vector w, and $\|\mathcal{A}(w) - \mathcal{A}(w')\|_1 \leq L\|w - w'\|_1$ for all two weight vectors w and w'. Then, there is an algorithm that returns D-approximate core allocation with Lipschitz constant $2L + 1$.*

3 Matching Game

In this section, we prove Theorem 4 by giving a Lipschitz continuous algorithm that returns an approximate core allocation of the matching game. We obtain the proof by constructing an algorithm that satisfies the assumptions in Lemma 9. Specifically, we prove that an algorithm that returns a vector represented by

$$\int_0^1 \textsc{MatchingGame}(G, w, b, \alpha) \mathrm{d}b \tag{5}$$

satisfies the assumptions of Lemma 9, where the procedure MatchingGame is provided in Algorithm 1. We give a deterministic algorithm to compute this integral in Section 3.3.

MatchingGame(G, w, b, α) first rounds each edge weight w_e to a value $\widehat{w}_e$ that is proportional to a power of α, where the proportionality constant is determined by b. Thereafter, it sorts the edges in descending order of $\widehat{w}_e$ and greedily selects them to form maximal matching M. Finally, for each edge $e \in M$, the algorithm allocates $\widehat{w}_e$ to both endpoints of the edges in M.

3.1 Core Approximability

The proofs of the following two lemmas for the core approximability analysis of Algorithm 1 are relatively straightforward.

▶ **Lemma 11.** *We have* $\|z\|_1 \leq 2\alpha \mathrm{OPT}(V, w)$.

Proof. Because M is a matching of G, we have $\sum_{e\in M} w_e \leq \mathrm{OPT}(V, w)$. Because the modified weight $\widehat{w}_e$ of each edge e in M contributes twice to $\|z\|_1$, we obtain

$$\|z\|_1 = 2\sum_{e\in M} \widehat{w}_e \leq 2\alpha \sum_{e\in M} w_e \leq 2\alpha \mathrm{OPT}(V, w). \qquad \blacktriangleleft$$

▶ **Lemma 12.** *Let* $S \subseteq V$. *Then, we have* $\sum_{v\in S} z_v \geq \mathrm{OPT}(S, w)$.

Proof. Let $e = (u, v) \in E$. When edge e begins to be examined in the loop starting from Line 4, if at least one of u or v (say, u) is already covered by M, then we have $\widehat{w}_e \leq z_u$. If neither u nor v are covered, then edge e is added to M, resulting in $\widehat{w}_e = z_u = z_v$. Therefore, $\widehat{w}_e \leq \max\{z_u, z_v\} \leq z_u + z_v$. Let M' be the maximum matching of $G[S]$. Then, we have

$$\sum_{v\in S} z_v \geq \sum_{(u,v)\in M'} (z_u + z_v) \geq \sum_{e\in M'} \widehat{w}_e \geq \sum_{e\in M'} w_e = \mathrm{OPT}(S, w). \qquad \blacktriangleleft$$

3.2 Lipschitz Continuity

Let $G = (V, E)$ be a graph, $f \in E$, $\delta > 0$, $b \in [0, 1]$, and $\alpha > 1$. We will bound

$$\frac{1}{\delta}\int_0^1 \left\|\textsc{MatchingGame}(G, w, b, \alpha) - \textsc{MatchingGame}(G, w + \delta\mathbf{1}_f, b, \alpha)\right\|_1 \mathrm{d}b, \tag{6}$$

which is an upper bound on

$$\frac{1}{\delta}\left\|\int_0^1 \textsc{MatchingGame}(G, w, b, \alpha)\mathrm{d}b - \int_0^1 \textsc{MatchingGame}(G, w + \delta\mathbf{1}_f, b, \alpha)\mathrm{d}b\right\|_1.$$

From Lemma 8, bounding (6) for $\delta \leq w_f$ or $w_f = 0$ is sufficient to prove Lipschitz continuity. We denote the value of $\widehat{w}$, M, and z in $\textsc{MatchingGame}(G, w, b, \alpha)$ (resp., $\textsc{MatchingGame}(G, w + \delta\mathbf{1}_f, b, \alpha)$) as $\widehat{w}$, M, and z (resp., $\widehat{w}'$, M', and z').

When $\widehat{w}_f = \widehat{w}'_f$, $\textsc{MatchingGame}(G, w, b, \alpha)$ and $\textsc{MatchingGame}(G, w + \delta\mathbf{1}_f, b, \alpha)$ output the same vector. Assume otherwise. In $\textsc{MatchingGame}(G, w, b)$, edge e_1 coming *before* edge e_2 refers to e_1 being considered before e_2 in the loop starting from Line 4, and is denoted as $e_1 \prec_{\widehat{w}} e_2$. In other words, either $\widehat{w}_{e_1} > \widehat{w}_{e_2}$ or $\widehat{w}_{e_1} = \widehat{w}_{e_2}$ and the index of e_1 comes earlier than that of e_2. For $e_1 \neq f \neq e_2$, the relations $e_1 \prec_{\widehat{w}} e_2$ and $e_1 \prec_{\widehat{w}'} e_2$ are equivalent. Thus, we simply denote this as $e_1 \prec e_2$. The following lemma forms the core of our Lipschitzness analysis:

▶ **Lemma 13.** *Assume* $\widehat{w}_f \neq \widehat{w}'_f$. *Then, we have* $\|z - z'\|_1 \leq 2\widehat{w}'_f$.

Proof. For each edge $e \neq f$ such that $e \in M' \setminus M$ (resp., $e \in M \setminus M'$), edge g is a *witness* of e if it is adjacent to e in M (resp., M') and $g \prec_{\widehat{w}} e$ (resp., $g \prec_{\widehat{w}'} e$). Intuitively, a witness of e is the edge that directly causes e to be excluded from M or M'.

From this definition, the witness of $e \in M' \setminus M$ belongs to $M \setminus M'$ and vice versa. Because $\prec$ is an ordering on $E \setminus \{f\}$, by tracing the witnesses from any edge in $M \triangle M'$, we will consequently arrive at f. This implies that as long as $M \neq M'$, the edges in $M \triangle M'$

form a single path or cycle including f. Moreover, in this case, we have $f \in M' \setminus M$, because if we can add f to M in MatchingGame(G, w, b, α), we could also add f to M' in MatchingGame$(G, w + \delta \mathbf{1}_f, b, \alpha)$.

Let us now complete the proof. When $M = M'$, we have $\|z - z'\|_1 \leq 2(\widehat{w}'_f - \widehat{w}_f) \leq 2\widehat{w}'_f$. Otherwise, we let $f = (u_0, v_0)$. Then, there exists a unique maximal sequence of vertices $(u_0, \dots, u_k)$ such that $(u_i, u_{i+1}) \in M \triangle M'$ for all i and $f \prec_{\widehat{w}'} (u_0, u_1) \prec (u_1, u_2) \prec \cdots \prec (u_{k-1}, u_k)$, and a unique maximal sequence of vertices $(v_0, \dots, v_l)$ such that $(v_i, v_{i+1}) \in M \triangle M'$ for all i and $f \prec_{\widehat{w}'} (v_0, v_1) \prec (v_1, v_2) \prec \cdots \prec (v_{l-1}, v_l)$ (if $M \triangle M'$ forms a cycle, then $(u_{k-1}, u_k) = (v_l, v_{l-1})$). Now, we have

$$\begin{aligned}
\|z - z'\|_1 &\leq \sum_{i=0}^{k} |z_{u_i} - z'_{u_i}| + \sum_{i=0}^{l} |z_{v_i} - z'_{v_i}| \\
&= \left(|\widehat{w}'_f - \widehat{w}_{(u_0,u_1)}| + \sum_{i=1}^{k} |\widehat{w}_{(u_{i-1},u_i)} - \widehat{w}_{(u_i,u_{i+1})}| \right) + \\
&\qquad \left(|\widehat{w}'_f - \widehat{w}_{(v_0,v_1)}| + \sum_{i=1}^{l} |\widehat{w}_{(v_{i-1},v_i)} - \widehat{w}_{(v_i,v_{i+1})}| \right) \\
&\leq 2\widehat{w}'_f,
\end{aligned}$$

where the last inequality is from the fact that the sequences defined by $\left(\widehat{w}'_f, \widehat{w}_{(u_0,u_1)}, \dots, \widehat{w}_{(u_{k-1},u_k)}\right)$ and $\left(\widehat{w}'_f, \widehat{w}_{(v_0,v_1)}, \dots, \widehat{w}_{(v_{l-1},v_l)}\right)$ are both decreasing. ◀

The following lemma analyzes the probability that $\widehat{w}_f \neq \widehat{w}'_f$ happens.

▶ **Lemma 14.** *If b is sampled uniformly from $[0, 1]$, $\widehat{w}_f \neq \widehat{w}'_f$ happens with a probability of at most $\frac{\delta}{w_f \ln \alpha}$.*

Proof. $\widehat{w}_f \neq \widehat{w}'_f$ happens when there exists an integer i with $w_f < \alpha^{i+b} \leq w_f + \delta$, indicating that $\lfloor \log_\alpha w_f - b \rfloor \neq \lfloor \log_\alpha w_f + \delta - b \rfloor$. This happens with probability

$$\log_\alpha(w_f + \delta) - \log_\alpha w_f = \log_\alpha \left(1 + \frac{\delta}{w_f}\right) \leq \frac{1}{\ln \alpha} \cdot \frac{\delta}{w_f} = \frac{\delta}{w_f \ln \alpha}.$$ ◀

Now, we complete our Lipschitzness analysis.

▶ **Lemma 15.** *We have*

$$\int_0^1 \|\mathit{MatchingGame}(G, w, b, \alpha) - \mathit{MatchingGame}(G, w + \delta \mathbf{1}_f, b, \alpha)\|_1 \, \mathrm{d}b \leq \frac{12}{\alpha - 1}\delta.$$

Proof. If $w_f = 0$, we have

$$\begin{aligned}
&\int_0^1 \|\textsc{MatchingGame}(G, w, b, \alpha) - \textsc{MatchingGame}(G, w + \delta \mathbf{1}_f, b, \alpha)\|_1 \, \mathrm{d}b \\
&\leq 2\widehat{w}'_f \leq 2\alpha(w_f + \delta) = 2\alpha\delta \leq \frac{12}{\alpha - 1}\delta,
\end{aligned}$$

where the last inequality is from $\alpha \leq 2$. Otherwise, we have

$$\begin{aligned}
&\int_0^1 \|\textsc{MatchingGame}(G, w, b, \alpha) - \textsc{MatchingGame}(G, w + \delta \mathbf{1}_f, b, \alpha)\|_1 \, \mathrm{d}b \\
&\leq \frac{\delta}{w_f \ln \alpha} \cdot 2\widehat{w}'_f \leq \frac{\delta}{w_f \ln \alpha} \cdot 2\alpha(w_f + \delta) \leq \frac{\delta}{w_f \ln \alpha} \cdot 4\alpha w_f = \frac{4\alpha\delta}{\ln \alpha} \leq \frac{12}{\alpha - 1}\delta,
\end{aligned}$$

where the third inequality is from $\delta \leq w_f$ and the last inequality is from the fact that $\frac{\alpha}{\ln \alpha} \leq \frac{3}{\alpha - 1}$ holds for $\alpha \in (1, 2]$. ◀

3.3 Proof of Theorem 4

Combining Lemmas 11, 12, and 15 and applying Lemma 9 yields the following:

▶ **Lemma 16.** *Let $\epsilon \in \left(0, \frac{1}{2}\right]$. For the matching game, an algorithm that returns $\left(\frac{1}{2} - \epsilon\right)$-approximate core allocation with Lipschitz constant $O(\epsilon^{-1})$ exists.*

Proof. Let $\alpha = 1 + 2\epsilon$. By combining Lemmas 11, 12, 15, and 9, we obtain an algorithm that returns $\frac{1}{2\alpha}$-approximate core allocation for the matching game with Lipschitz constant $\frac{24}{\alpha - 1} + 1$. As $\frac{1}{2(1+2\epsilon)} \geq \frac{1}{2} - \epsilon$ and $\frac{24}{2\epsilon} + 1 \leq O(\epsilon^{-1})$, this algorithm satisfies the claims of the lemma. ◀

Proof of Theorem 4. It is sufficient to prove that the allocation defined by Lemma 16 can be computed in polynomial time. For each edge $e \in E$, let $b_e = \log_\alpha w_e - \lfloor \log_\alpha w_e \rfloor$, and sort the b_e values in ascending order to obtain a sequence $t_1, \ldots, t_{|E|}$. For convenience, we set $t_0 = 0$ and $t_{|E|+1} = 1$. For each $i = 0, \ldots, |E|$, the behavior of Algorithm 1 for any $b \in [t_i, t_{i+1})$ is identical, except for the constant multiplier on $\widehat{w}$. Therefore, by running Algorithm 1 for $b = t_i$ and appropriately scaling the result, and thereafter summing these results for each i, we can compute the integral in Equation (5) in polynomial time. ◀

4 Minimum Spanning Tree Game

In this section, we prove Theorem 5 by giving a Lipschitz continuous algorithm that returns an approximate core allocation for the minimum spanning tree game. The proof is obtained by constructing an algorithm that satisfies the assumption of Lemma 10. Specifically, we prove that an algorithm that returns a vector represented by

$$\int_0^1 \textsc{MSTGame}(G, w, b) \mathrm{d}b$$

satisfies the assumptions of Lemma 10, where the procedure MSTGame is provided in Algorithm 3. We give a deterministic algorithm to compute this integral in Section 4.3.

To derive our allocation, we use an *auxiliary tree* that simulates Kruskal's algorithm, constructed as in Algorithm 2. The auxiliary tree is a rooted tree such that each leaf corresponds to a vertex in $V \cup \{r\}$. We provide an overview of Algorithm 2. Initially, for each vertex $v \in V \cup \{r\}$, the algorithm prepares a vertex $u_{\{v\}}$ and sets its *height* $h_{u_{\{v\}}}$ to 0. The auxiliary tree is constructed by adding the edges of E in ascending order of weight to graph $(V \cup \{r\}, \emptyset)$. Edges of the same weight are added simultaneously. When adding the edges of a certain weight results in merging multiple connected components $C_1, \ldots, C_k$ into a single connected component C, the algorithm creates a vertex u_C corresponding to C in the auxiliary tree. The height of u_C is set as the weight of the edges at that time, and the edges are added to the auxiliary tree from u_C to $u_{C_1}, \ldots, u_{C_k}$.

For $x \in \mathbb{R}_{\geq 0}$ and a weight vector $\widehat{w}$, let $\mathcal{C}_{\widehat{w},<x}$ and $\mathcal{C}_{\widehat{w},\leq x}$ be the families of connected components of graphs whose vertex sets are $V \cup \{r\}$ and edge sets consist of edges $e \in E$ with $\widehat{w}_e < x$ and $\widehat{w}_e \leq x$, respectively. For the auxiliary tree T, we denote the subtree rooted at vertex u by T_u. When the edges of an auxiliary tree are referred to as (u, u'), u is the parent of u'. For an edge $e = (u, u')$, we denote $T_e = T_{u'}$. For simplicity, for $e = (u, u') \in E(T)$, we define $h_e := h_u$.

$\textsc{MSTGame}(G, w, b)$ first rounds each edge weight w_e to a value $\widehat{w}_e$ that is proportional to a power of 2, where the proportionality constant is determined by b. Let T be the auxiliary tree derived from $(G, \widehat{w})$. Then, for each edge e in T such that T_e does not have r as a

Algorithm 2 Construction of the auxiliary tree.

Procedure AuxiliaryTree$(G, \widehat{w})$
  Input: A graph $G = (V \cup \{r\}, E)$ and a weight vector $\widehat{w} \in \mathbb{R}^E_{\geq 0}$.
  $U \leftarrow \{u_{\{v\}} \mid v \in V \cup \{r\}\}$, $F \leftarrow \emptyset$;
  $h_u \leftarrow 0$ for each $u \in U$;
  for $x \in \mathbb{R}_{\geq 0}$ *such that* $\mathcal{C}_{\widehat{w},<x} \neq \mathcal{C}_{\widehat{w},\leq x}$, *in ascending order* **do**
    for $C \in \mathcal{C}_{\widehat{w},\leq x} \setminus \mathcal{C}_{\widehat{w},<x}$ **do**
      Add a new vertex u_C to U;
      $h_{u_C} \leftarrow x$;
      for $C' \in \mathcal{C}_{\widehat{w},<x}$ *such that* $C' \subseteq C$ **do**
        Add a new edge $(u_C, u_{C'})$ to F;
  return (U, F, h);

Algorithm 3 Lipschitz continuous allocation for the minimum spanning tree game.

Procedure MSTGame(G, w, b)
  Input: A graph $G = (V \cup \{r\}, E)$, a weight vector $w \in \mathbb{R}^E_{\geq 0}$, and $b \in [0, 1]$.
  For each $e \in E$ with $w_e > 0$, let $\widehat{w}_e \leftarrow 2^{i_e+1+b}$, where i_e be the unique integer such that $2^{i_e+b} \leq w_e < 2^{i_e+1+b}$;
  $T \leftarrow$ AuxiliaryTree$(G, \widehat{w})$ and identify the vertices of G with the corresponding leaves of T;
  for $e \in E(T)$ *such that* $r \notin T_e$ **do**
    Let X_e be the set of leaves in T_e;
    Let $z_e \leftarrow \frac{h_e}{|X_e|}\mathbf{1}_{X_e}$;
  return $\sum_{e \in E} z_e$;

leaf, the value h_e is evenly distributed among the agents corresponding to the leaves of T_e. At first glance, the total value distributed may seem unrelated to the value of the grand coalition. However, it can be proved in Lemmas 18 and 19 that the total value distributed by this method is at least $\mathrm{OPT}(G, \widehat{w})$ and at most $2\mathrm{OPT}(G, \widehat{w})$.

4.1 Core Approximability

We begin by analyzing the core approximability. Let T be the auxiliary tree for $(G, \widehat{w})$. For $X \subseteq V \cup \{r\}$, the *connector* $\mathrm{conn}(X)$ of X is the minimal connected subgraph of T that contains all leaves of T corresponding to X. The following lemma bounds the value of the characteristic function for a subset S of V using values that can be computed from connector $\mathrm{conn}(S \cup \{r\})$. Due to the space limit, proofs in this section are given in the full version.

▶ **Lemma 17.** *Let $S \subseteq V$. Then, we have*

$$\sum_{\substack{(u,u') \in E(\mathrm{conn}(S \cup \{r\})) \\ r \notin T_{u'}}} (h_u - h_{u'}) \leq \mathrm{OPT}(G[S \cup \{r\}], \widehat{w}).$$

When $V = S$, the equality holds.

Now, we have the following.

▶ **Lemma 18.** *Let $S \subseteq V$. Then, we have*

$$\sum_{v \in S} \sum_{e \in E(T)} z_{e,v} \leq 4\mathrm{OPT}(G[S \cup \{r\}], w).$$

We have the following bound for the grand coalition.

▶ **Lemma 19.** *We have*

$$\sum_{v \in V} \sum_{e \in E(T)} z_{e,v} \geq \mathrm{OPT}(G, w).$$

4.2 Lipschitz Continuity

Let $f \in E(G)$. We bound

$$\frac{1}{\delta} \int_0^1 \left\| \mathrm{MSTGame}(G, w, b) - \mathrm{MSTGame}(G, w + \delta \mathbf{1}_f, b) \right\|_1 \mathrm{d}b,$$

which is an upper bound on

$$\frac{1}{\delta} \left\| \int_0^1 \mathrm{MSTGame}(G, w, b) \mathrm{d}b - \int_0^1 \mathrm{MSTGame}(G, w + \delta \mathbf{1}_f, b) \mathrm{d}b \right\|_1.$$

Without loss of generality, we can assume $\delta \leq w_f$ or $w_f = 0$. Now we fix $b \in [0, 1]$. We denote the value of T, $\widehat{w}$, X, and z in $\mathrm{MSTGame}(G, w, b)$ (resp., $\mathrm{MSTGame}(G, w + \delta \mathbf{1}_f, b)$) as T, $\widehat{w}$, X, and z (resp. T', $\widehat{w}'$, X', and z').

When $\widehat{w}_f = \widehat{w}'_f$, T and T' are the same and $\mathrm{MSTGame}(G, w, b)$ and $\mathrm{MSTGame}(G, w + \delta \mathbf{1}_f, b)$ output the same vector. Otherwise, let C be the connected component of $\mathcal{C}_{\widehat{w}, \leq \widehat{w}_f}$ that contains both the endpoints of f. If C is still connected even after the removal of f, then we have $\mathcal{C}_{\widehat{w}, \leq x} = \mathcal{C}_{\widehat{w}', \leq x}$ for all $x \geq 0$ and thus $T = T'$. Otherwise, let C_1 and C_2 be two connected components of C after the removal of f. Subsequently, T' is obtained from T by the following operation:

(1.1) If no vertex u_{C_1} exists in T, create a vertex u_{C_1}, set $h_{u_{C_1}} = \widehat{w}_f$, and for each child u_X of u_C with $X \subseteq C_1$, replace the edge (u_C, u_X) with (u_{C_1}, u_X). Otherwise, delete the edge (u_C, u_{C_1}).

(1.2) Do exactly the same for C_2.

(2) If u_C has a parent u_Y with $h_{u_Y} = \widehat{w}'_f$, delete the vertex u_C and the edge (u_Y, u_C), and add two new edges (u_Y, u_{C_1}) and (u_Y, u_{C_2}). Otherwise, add two new edges (u_C, u_{C_1}) and (u_C, u_{C_2}) and then change the value of h_{u_C} from $\widehat{w}_f$ to $\widehat{w}'_f$.

We can observe that all edges e of T except for the edges (u_C, u_{C_1}) deleted in (1.1), (u_C, u_{C_2}) deleted in (1.2), and (u_Y, u_C) deleted in (2) naturally correspond to edges in T' that are not added in (2), and if we identify the edges of T with those of T' using that correspondence, it holds that $h_e = h'_e$ and $X_e = X'_e$, which implies $z_e = z'_e$. Therefore, we have the following.

▶ **Lemma 20.** *Assume $\widehat{w}_f \neq \widehat{w}'_f$. Then, we have*

$$\left\| \mathit{MSTGame}(G, w, b) - \mathit{MSTGame}(G, w + \delta \mathbf{1}_f, b) \right\|_1 \leq \widehat{w}_f + 2\widehat{w}'_f.$$

Proof. Let

$$z_1 = \begin{cases} \frac{1}{|C_1|} \mathbf{1}_{C_1} & \text{if } (u_C, u_{C_1}) \text{ is deleted in (1.1) and } r \notin C_1 \\ \mathbf{0} & \text{otherwise} \end{cases},$$

$$z_2 = \begin{cases} \frac{1}{|C_2|}\mathbf{1}_{C_2} & \text{if } (u_C, u_{C_2}) \text{ is deleted in (1.2) and } r \notin C_2 \\ \mathbf{0} & \text{otherwise} \end{cases},$$

$$z_3 = \begin{cases} \frac{1}{|C|}\mathbf{1}_{C} & \text{if } (u_Y, u_C) \text{ is deleted in (1.2) and } r \notin C \\ \mathbf{0} & \text{otherwise} \end{cases},$$

$$z_4 = \begin{cases} \frac{1}{|C_1|}\mathbf{1}_{C_1} & \text{if } r \notin C_1 \\ \mathbf{0} & \text{otherwise} \end{cases},$$

$$z_5 = \begin{cases} \frac{1}{|C_2|}\mathbf{1}_{C_2} & \text{if } r \notin C_2 \\ \mathbf{0} & \text{otherwise} \end{cases}.$$

Then, we have

$$\begin{aligned} &\|\text{MSTGame}(G, w, b) - \text{MSTGame}(G, w + \delta\mathbf{1}_f, b)\|_1 \\ &= \left\|(z_1 + z_2)\,\widehat{w}_f + (z_3 - z_4 - z_5)\,\widehat{w}'_f\right\|_1 \\ &= \left\|\left(z_1\widehat{w}_f + (z_3 \circ \mathbf{1}_{C_1} - z_4)\,\widehat{w}'_f\right)\right\|_1 + \left\|\left(z_2\widehat{w}_f + (z_3 \circ \mathbf{1}_{C_2} - z_5)\,\widehat{w}'_f\right)\right\|_1 \\ &\le \max\left(\widehat{w}_f + \frac{|C_1|}{|C|}\widehat{w}'_f, \widehat{w}'_f\right) + \max\left(\widehat{w}_f + \frac{|C_2|}{|C|}\widehat{w}'_f, \widehat{w}'_f\right) \le \widehat{w}_f + 2\widehat{w}'_f. \end{aligned}$$ ◀

The following lemma analyzes the probability that $\widehat{w}_f \neq \widehat{w}'_f$ happens.

▶ **Lemma 21.** *Assume $w'_f \le 2w_f$. If b is sampled uniformly from $[0, 1]$, $\widehat{w}_f \neq \widehat{w}'_f$ happens with a probability of at most $\frac{\delta}{w_f \log 2}$.*

Proof. $\widehat{w}_f \neq \widehat{w}'_f$ happens when there is an integer i with $w_f < 2^{i+b} \le w_f + \delta$, implying that $\lfloor \log_2 w_f - b \rfloor \neq \lfloor \log_2 w_f + \delta - b \rfloor$. This happens with probability

$$\log_2(w_f + \delta) - \log_2 w_f = \log_2\left(1 + \frac{\delta}{w_f}\right) \le \frac{\delta}{w_f \log 2}.$$ ◀

Now, we have the following:

▶ **Lemma 22.** *We have*

$$\frac{1}{\delta}\int_0^1 \|MSTGame(G, w, b) - MSTGame(G, w + \delta\mathbf{1}_f, b)\|_1 \,\mathrm{d}b \le \frac{10\delta}{\log 2}.$$

Proof. If $w_f = 0$, we have

$$\frac{1}{\delta}\int_0^1 \|\text{MSTGame}(G, w, b) - \text{MSTGame}(G, w + \delta\mathbf{1}_f, b)\|_1 \,\mathrm{d}b \le 2\widehat{w}'_f \le 4w'_f = 4\delta.$$

Otherwise, we have

$$\begin{aligned} &\frac{1}{\delta}\int_0^1 \|\text{MSTGame}(G, w, b) - \text{MSTGame}(G, w + \delta\mathbf{1}_f, b)\|_1 \,\mathrm{d}b \\ &\le \frac{\delta}{w_f \log 2}\left(\widehat{w}_f + 2\widehat{w}'_f\right) = \frac{\delta}{w_f \log 2} \cdot 5\widehat{w}_f \le \frac{\delta}{w_f \log 2} \cdot 10w_f = \frac{10\delta}{\log 2}. \end{aligned}$$ ◀

4.3 Proof of Theorem 5

Combining Lemmas 18, 19, and 22 and applying Lemma 10 yields Theorem 5.

Proof of Theorem 5. Combining Lemmas 18, 19, 22, and 10, we obtain an algorithm that returns 4-approximate core allocation for the matching game with Lipschitz constant $\frac{20}{\log 2} + 1 = O(1)$. The fact that the allocation defined by Lemma 16 can be computed in polynomial time is obtained using the same argument as in the proof of Theorem 4. ◀

References

1 Kazutoshi Ando. Computation of the shapley value of minimum cost spanning tree games: #P-hardness and polynomial cases. *Japan Journal of Industrial and Applied Mathematics*, 29(3):385–400, 2012.

2 Haris Aziz and Bart de Keijzer. Shapley meets shapley. *arXiv preprint*, 2013. `arXiv:1307.0332`.

3 Márton Benedek, Péter Biró, Matthew Johnson, Daniël Paulusma, and Xin Ye. The complexity of matching games: A survey. *Journal of Artificial Intelligence Research*, 77:459–485, 2023.

4 Charles G Bird. On cost allocation for a spanning tree: a game theoretic approach. *Networks*, 6(4):335–350, 1976.

5 Péter Biró, Walter Kern, Daniël Paulusma, and Péter Wojuteczky. The stable fixtures problem with payments. *Games and economic behavior*, 108:245–268, 2018.

6 Armin Claus and Daniel J Kleitman. Cost allocation for a spanning tree. *Networks*, 3(4):289–304, 1973.

7 Vincent Conitzer and Tuomas Sandholm. Complexity of constructing solutions in the core based on synergies among coalitions. *Artificial Intelligence*, 170(6-7):607–619, 2006.

8 Xiaotie Deng, Toshihide Ibaraki, and Hiroshi Nagamochi. Algorithmic aspects of the core of combinatorial optimization games. *Mathematics of Operations Research*, 24(3):751–766, 1999.

9 Kimmo Eriksson and Johan Karlander. Stable outcomes of the roommate game with transferable utility. *International Journal of Game Theory*, 29:555–569, 2001.

10 Ulrich Faigle, Sándor P Fekete, Winfried Hochstättler, and Walter Kern. On approximately fair cost allocation in euclidean tsp games. *Operations-Research-Spektrum*, 20:29–37, 1998.

11 Ulrich Faigle and Walter Kern. On some approximately balanced combinatorial cooperative games. *Zeitschrift für Operations Research*, 38:141–152, 1993.

12 Ulrich Faigle and Walter Kern. Approximate core allocation for binpacking games. *SIAM Journal on Discrete Mathematics*, 11(3):387–399, 1998.

13 Ulrich Faigle, Walter Kern, Sándor P Fekete, and Winfried Hochstättler. On the complexity of testing membership in the core of min-cost spanning tree games. *International Journal of Game Theory*, 26:361–366, 1997.

14 Daniel Granot and Gur Huberman. Minimum cost spanning tree games. *Mathematical programming*, 21:1–18, 1981.

15 Satoshi Hara and Yuichi Yoshida. Average sensitivity of decision tree learning. In *The 11th International Conference on Learning Representations (ICLR)*, 2023.

16 Sergiu Hart. Shapley value. In *Game theory*, pages 210–216. Springer, 1989.

17 Walter Kern and Xian Qiu. Integrality gap analysis for bin packing games. *Operations Research Letters*, 40(5):360–363, 2012.

18 Jeroen Kuipers. Bin packing games. *Mathematical Methods of Operations Research*, 47:499–510, 1998.

19 Soh Kumabe and Takanori Maehara. Convexity of b-matching games. In *International Joint Conference on Artificial Intelligence*, pages 261–267, 2020.

20 Soh Kumabe and Takanori Maehara. Convexity of hypergraph matching game. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 663–671, 2020.

21 Soh Kumabe and Yuichi Yoshida. Average sensitivity of dynamic programming. In *Proceedings of the 33th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1925–1961, 2022.

22 Soh Kumabe and Yuichi Yoshida. Lipschitz continuous algorithms for covering problems. *arXiv preprint*, 2023. `arXiv:2307.08213`.

23 Soh Kumabe and Yuichi Yoshida. Lipschitz continuous algorithms for graph problems. In *Proceedings of the 2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 762–797. IEEE, 2023.

24 Tianhang Lu, Han Xiao, and Qizhi Fang. Approximate core allocations for edge cover games. In *International Workshop on Frontiers in Algorithmics*, pages 105–115. Springer, 2023.

25 Pan Peng and Yuichi Yoshida. Average sensitivity of spectral clustering. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 1132–1140, 2020.

26 Xian Qiu and Walter Kern. Approximate core allocations and integrality gap for the bin packing game. *Theoretical Computer Science*, 627:26–35, 2016.

27 Lloyd S Shapley. *On Balanced Sets and Cores*. RAND Corporation, 1965.

28 Lloyd S Shapley and Martin Shubik. Quasi-cores in a monetary economy with nonconvex preferences. *Econometrica: Journal of the Econometric Society*, pages 805–827, 1966.

29 Lloyd S Shapley and Martin Shubik. The assignment game I: The core. *International Journal of Game Theory*, 1:111–130, 1971.

30 Marilda Sotomayor. The multiple partners game. In *Equilibrium and Dynamics: Essays in Honour of David Gale*, pages 322–354. Springer, 1992.

31 Nithin Varma and Yuichi Yoshida. Average sensitivity of graph algorithms. *SIAM Journal on Computing*, 52(4):1039–1081, 2023.

32 Vijay V Vazirani. The general graph matching game: Approximate core. *Games and Economic Behavior*, 132:478–486, 2022.

33 Gerhard J Woeginger. On the rate of taxation in a cooperative bin packing game. *Zeitschrift für Operations Research*, 42:313–324, 1995.

34 Han Xiao, Tianhang Lu, and Qizhi Fang. Approximate core allocations for multiple partners matching games. *arXiv preprint*, 2021. `arXiv:2107.01442`.

35 Yuichi Yoshida and Shinji Ito. Average sensitivity of Euclidean k-clustering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Towards an Analysis of Quadratic Probing

William Kuszmaul
Harvard University, Cambridge, MA, USA

Zoe Xi
Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

Since 1968, one of the simplest open questions in the theory of hash tables has been to prove *anything* nontrivial about the correctness of quadratic probing. We make the first tangible progress towards this goal, showing that there exists a positive-constant load factor at which quadratic probing is a constant-expected-time hash table. Our analysis applies more generally to any fixed-offset open-addressing hash table, and extends to higher load factors in the case where the hash table examines blocks of some size $B = \omega(1)$.

2012 ACM Subject Classification Theory of computation → Sorting and searching

Keywords and phrases quadratic probing, hashing, open addressing, witness trees

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.103

Category Track A: Algorithms, Complexity and Games

Funding This work was partially sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.
William Kuszmaul: Funded by Harvard Rabin Postdoctural Fellowship.
Zoe Xi: Funded by the Carl W. Hoffman and Elizabeth B. Klerman Fund, and by the John Reed Fund.

1 Introduction

The field of open-addressed hash tables began with the introduction of linear probing in the 1950s [35, 24, 25]. Although early work [35] conjectured that linear probing should scale well to high load factors, with an insertion time of $O(x)$ at load factor $1-1/x$, subsequent analyses by Knuth [24] (unpublished) and by Konheim and Weiss [26] (published in 1966) showed that this is not the case. Due to *clustering effects*, in which elements group together to form long continuous runs of occupied slots, the true expected insertion time is asymptotically larger than researchers had hoped for, evaluating to $\Theta(x^2)$.

In the late 1960s, this prompted researchers to propose alternative hashing algorithms that preserved the simplicity (and in some cases data locality) of linear probing, while mitigating the clustering effects. Two solutions, in particular, emerged as natural alternatives, **double hashing**, which was introduced by de Balbine in his 1968 thesis [4] (and proposed independently by Bell and Kaman in 1970 [7]); and **quadratic probing**, which was introduced by Maurer in 1968 [32] and then refined by other sets of authors through the 1970s [5, 22, 16, 36].

Despite their data-structural simplicity, double hashing and quadratic probing proved far harder to analyze than linear probing. It wasn't until 1976, in a breakthrough paper by Guibas and Szemeredi [19], that double hashing was finally partially analyzed: they proved that, so long as the load factor of the hash table is at most ≈ 0.28, the hash table is

guaranteed to support constant-expected-time operations. This constant was subsequently improved to ≈ 0.31 in 1978 [20], which remained the state of the art until 1988, when Lueker [30, 31] finally extended the analysis to apply to all load factors. In particular, Lueker showed a coupling between double hashing and uniform probing, proving that the expected insertion times are within a $1 + o(1)$ factor of each other.

The coupling techniques [19, 20, 21, 30, 31] that allowed for an analysis of double hashing do not extend to quadratic probing. It has remained an open question for more than five decades to prove *anything* nontrivial about the behavior of quadratic-probing hash tables. It is not even known, for example, whether quadratic-probing is a constant-time data structure when used at a load factor of 0.001.

Linear Probing vs Double Hashing vs Quadratic Probing

Let us take a moment to briefly define the three hash-table designs described above. In each case, elements are stored in an array of some size n, and the **load factor** of the hash table is defined to be the fraction of slots that are occupied. To insert an element x, a sequence $p_1(x), p_2(x), \ldots$ of array positions are examined until an unoccupied spot is found for x. Where the three hash-table designs differ is in the choice of probe sequence: linear probing uses $p_i(x) = h(x) + i \mod n$, where $h(x) \in [n]$ is a random hash; double hashing uses $p_i(x) = h_1(x) + ih_2(x) \mod n$, where $h_1(x), h_2(x) \in [n]$ are both random hashes; and quadratic probing uses $p_i(x) = h(x) + (i-1)^2 \mod n$, where again $h(x) \in [n]$ is a random hash.

The analysis of linear probing [24, 26] hinges on the observation that, if an insertion x takes time k, there must be an array interval of the form $I = [h(x) - j, h(x) + k - 1]$, for some $j \geq 0$, such that the number of elements y with hashes $h(y) \in I$ is at least $|I| = j + k$. Thus, the analysis of linear probing reduces directly to the analysis of how many elements hash into each interval in the hash table. The analysis of double hashing [19, 20, 21, 30, 31] relies on the fact that the probe sequence $p_i(x) = h_1(x) + ih_2(x) \mod n$ is in some formal sense nearly as random as the fully random probe sequence $p_i(x) = h_i(x)$, where $h_1, h_2, \ldots$ are all independent hash functions.

Quadratic probing, on the other hand, sits in an unfortunate middle ground. It lacks the clean interval-based structure of linear probing – if an insertion x collides with another element y in position $p_i(x) = h(x) + (i-1)^2 \mod n$, then there is nothing substantive that we can say about the array interval $[h(x), h(y)]$. But it is not comparable to a fully random probe sequence – indeed, elements x and y with hashes $h(x)$ and $h(y)$ that are close together, are far more likely to interact than are a random pair of elements. The interactions between pairs of elements follow an apparently chaotic combinatorial structure: if an insertion x collides with another element y in position $p_i(x) = h(x) + (i-1)^2 \mod n$, then the same element x would not have interacted with y at all if y's hash had been 1 smaller; and, similarly, y might have been in a different position entirely had any of the elements *it* interacted with had even slightly different hashes, etc.

Yet, empirically, quadratic probing is an excellent hash table design [38, 32, 11, 15, 12, 6, 18, 23, 39, 40]. It preserves much of the data locality that makes linear probing special [38] while also empirically eliminating the asymptotic clustering effects that make linear probing bad [32]. It is recommended in textbooks and courses [11, 15, 12, 6, 18, 23, 39, 40], and is even used as the underlying design (with some modifications we will discuss later) for Google's in-house and open-source hash tables [1]. Thus, the problem would truly seem to be one of algorithm *analysis* rather than one of algorithm *design*.

This Paper: A Partial Analysis of Quadratic Probing

We give the first analysis for quadratic-probing hash tables at low load factors. We show that, at any load factor less than roughly 0.089, the expected time per operation is $O(1)$. In fact, our analysis applies not just to quadratic probing, but to any **fixed-offset** probing scheme, i.e., to any hash table that, like linear and quadratic probing, inserts elements via a probe sequence of the form $p_i(x) = h(x) + f(i)$ for some $f(i)$.

▶ **Theorem 1.** *There exists a positive constant $\alpha \geq 0.089$ such that all fixed-offset open addressing hash tables support constant-time insertions at load factor α and below. Moreover, the insertion time is bounded above by a geometric random variable with mean $O(1)$.*

The proof of Theorem 1 in Section 3 is achieved by a witness argument in which we construct two objects (a witness set S and a witness transcript T) that must exist in order for the insertion time to be large. The witness set S has the property that each individual option for S has only a very small probability of occurring. The witness transcript T, on the other hand, has the property there are only a relatively small number of options for what T can be. Finally, the relationship between the two (and, in particular, the fact T can be used to recover S), allows us to bound the probability of such a pair (S, T) existing at all.

The specific constant, 0.089, that we get from Theorem 1 stems from the careful enumeration of a family of strings that we call **witness strings** (i.e., candidates for the transcript T). Through a mixture of algebraic and combinatorial arguments, we obtain tight bounds on the growth rate for the family – this rate, in turn, dictates the best constant that we can get in our proof of Theorem 1.

Finally, in Section 4, we turn our attention to **chunked** fixed-offset open addressing, in which the probe sequence used is actually of the form

$$p_i(x) = h(x) + B \cdot f(\lfloor i/B \rfloor) + (i - 1)$$

for some **chunk size** B. The use of chunking is quite common in practice, as it reduces cache misses and allows for the use of hardware vectorization. A notable example is the hash table used at Google [1], which uses chunked quadratic probing with chunk size $B = 16$.

Our final result is an analysis of any chunk fixed-offset open-addressing scheme. We show that, when $B = \omega(1)$, such schemes can successfully handle load factors of the form $1 - o(1)$.

▶ **Theorem 2.** *There exists a constant $\alpha \in (0, 1)$ such that the following is true. Consider a chunked fixed-offset open-addressed hash table with chunk size B. Any insertion of an element x at a load factor $\alpha = 1 - 1/q$ satisfying $q \leq \alpha\sqrt{B/\log B}$ takes expected time at most $O(q^2)$. Moreover, the insertion time is bounded above by a geometric random variable with mean $O(q^2)$.*

We remark that, in general, the time/space tradeoff in Theorem 2 is nearly tight in the sense that linear probing (which is a trivial example of chunked open addressing) does, indeed, require $\Theta(q^2)$ time per insertion.

Other related work

In the half century since quadratic probing and double hashing were introduced, there has been a great deal of additional work on hash-table design. Notable examples include Cuckoo hashing [34], which allows for worst-case bounds on query time (and which has the interesting feature that, depending on the parameters with which it is implemented, there are genuine load-factor thresholds above which it cannot be used [34, 17, 14]); Robin-Hood

hashing [2, 10], which reorders elements in ways that reduce query time; graveyard hashing [9], which strategically leaves gaps within a linear probing hash table to reduce clustering; and many others [25]. In addition to these relatively practical designs, there has also been a great deal of progress on the theoretical extremes of how space/time efficient a hash table can be [29, 3, 13, 37]. In fact, recent works by Bender et al. [8] and by Li et al. [27, 28] close off the basic theoretical question of how space-efficient a hash table can be subject to a given set of time bounds. For a detailed but somewhat out-of-date survey, see Knuth's [25] 1998 edition of the Art of Computer Programming, Vol. 3.

2 Preliminaries

Let $h : U \to [n]$ be a random hash function, and let $r_1, r_2, \ldots, r_{n-1} \in \mathbb{N}$ be a permutation of the numbers 1 through $n-1$. Consider an open-addressing hash table with capacity n in which, to insert an element $x \in U$, we place it in the first available position from the sequence $h(x), h(x)+r_1, h(x)+r_2, \ldots$ (the positions wrap around, so position $n+1$ is the same as position 1). Such a hash table is said to perform **fixed-offset open-addressing** with **offset sequence** $r_1, r_2, \ldots$. The hash table is said to support **constant time insertions** at load factor $0 < \alpha \leq 1$ and below if, when the hash table is filled to a α-fraction full, each of the insertions is guaranteed to take constant expected time.

Additionally, a fixed-offset open-addressing scheme is said to be **chunked** (with **chunk size** B) if the $\{r_i\}$'s are broken into consecutive blocks of size B. That is, for each $i \geq 0$, and $j \in \{0, \ldots, B-1\}$, we have $r_{iB+j} = r_{iB} + j$.

One subtlety in the definition of fixed-offset open-addressing is the requirement that $r_1, r_2, \ldots, r_{n-1}$ form a *permutation* of $[n-1]$. We will see that, if our analysis is taking place at load factor $1 - 1/q$, the time per insertion is at most a geometric random variable with mean $O(1+q^2)$ (see Theorems 1 and 2), meaning that with probability $1 - 1/\operatorname{poly}(n)$, the hash table only ever uses the first $O((1+q^2)\log n)$ terms of any probe sequence. Therefore it is not strictly necessary to require that $r_1, r_2, \ldots, r_{n-1}$ are all distinct. It suffices for the probe sequence to satisfy the weaker requirement that $r_1, r_2, \ldots, r_{O((1+q^2)\log n)}$ are distinct, and to assume that the hash table is rebuilt from scratch if any operation ever takes $\omega((1+q^2)\log n)$ time. This distinction is important since the probe sequence used by quadratic probing is not a permutation for all table-sizes n [25], but is trivially guaranteed to have distinct entries for all of its first $\Omega(\sqrt{n})$ terms.

Finally, the reader may wonder whether our analyses can be extended to support deletions in addition to insertions and queries. Here there is a larger issue: quadratic probing does not natively *support* deletions. If one tries to implement deletions by simply removing items, then the query algorithm gets broken (it can no longer terminate when it sees a free slot) [25]. The standard way to implement deletions while preserving the correctness of queries is to use tombstones [9], which formally reduce the problem to the insertion-only setting.

3 Analysis for Sufficiently Small Constant Load Factors

In this section, we will show that there exists a universal load factor $\alpha \geq 0.089$ below which all fixed-offset open-addressing schemes are guaranteed to achieve $O(1)$-time operations.

▶ **Theorem 1.** *There exists a positive constant $\alpha \geq 0.089$ such that all fixed-offset open addressing hash tables support constant-time insertions at load factor α and below. Moreover, the insertion time is bounded above by a geometric random variable with mean $O(1)$.*

▶ **Corollary 3.** *There exists a positive constant $\alpha \geq 0.089$ such that quadratic probing supports constant-time insertions at load factor α and below.*

Let $r_1, r_2, \ldots$ be the offset sequence used by the hash table, and as a convention set $r_0 = 0$. Consider a sequence of up to αn insertions, followed by one additional insertion of an element x. Let D denote the state of the hash table when x is inserted, and for each element x that was inserted in the past, let $\text{index}(x, D)$ be the index i such that x resides in position $h(x) + r_i$.

Given a set $S \subseteq [n]$ we say that an element $x \in D$ **conflicts** with S if $h(x) \notin S$ and if $h(x) + r_i$ is in S for some $i \leq \text{index}(x, D)$. Moreover, if $h(x) + r_k$ is the first element of $h(x), h(x) + r_1, h(x) + r_2, \ldots$ to appear in the set S, then we say that the **pair** (x, k) **conflicts with** S **at position** $j = h(x) + r_k$. Finally, we define $\texttt{Conflicts}(S, j)$, which is referred to as a conflict set, to be the set of pairs that conflict with S at position $j \in S$, i.e.,

$$\texttt{Conflicts}(S, j) = \{(x, k) \mid x \in D, \\ 1 \leq k = \min\{i \mid h(x) + r_i \in S\},\ h(x) + r_k = j,\ k \leq \text{index}(x, D)\}.$$

We want to bound the probability that, when insertion x occurs, all of positions $h(x), h(x) + r_1, \ldots, h(x) + r_{\ell-1}$ are already occupied for some large ℓ, i.e., the insertion takes time greater than ℓ. Using the idea of a conflict set, we design a protocol BuildWitnesses (Algorithm 1) that takes as inputs D, $h(x)$, and ℓ, and returns a **witness set** S along with a **witness transcript** T. The witness set S will be a subset of $[n]$, and the witness transcript T will be a trinary string.

As we shall see, the basic idea is that, if the insertion of $h(x)$ into D takes time at least ℓ, then the $\text{BuildWitnesses}(D, h(x), \ell)$ protocol will return a pair (S, T) such that:

- the set $S \subseteq [n]$ is quite large, satisfying $|S| \geq \ell$;
- there are at least $|S|$ elements $x \in D$ satisfying $h(x) \in S$;
- the set S is fully determined by the triple $(h(x), \ell, T)$;
- and the transcript T is a trinary string of length $O(|S|)$.

We will then be able to argue that the probability of such a pair of objects existing is very small, at most $2^{-\Omega(\ell)}$. Thus, by analyzing BuildWitnesses, we will be able to indirectly arrive at a proof of Theorem 1. We emphasize that, although this approach uses an algorithm (BuildWitnesses) as part of the analysis, it is not an algorithm that actually gets executed by the hash table – it is simply for the sake of analysis.

Before we can dive into the analysis, we must show that Algorithm 1 terminates.

▶ **Lemma 4.** *Algorithm 1 terminates within finite time.*

Proof. First observe that Line 14 only adds elements to S that are not already in S. Since elements are never removed from S, it follows that each $j \in [n]$ can also be added to S (and therefore to `Unprocessed`) at most once. Since each phase (i.e., each iteration of Line 4) removes an item from `Unprocessed`, there can be at most n phases. Furthermore, since each iteration of Lines 11–15 increases the size of S, there can be at most n total iterations of lines 11–15. Therefore, the algorithm completes its construction of the witness objects S and T within $O(n)$ time. ◀

Next, we turn our attention to establishing the properties of S and T, along with their relationship to one another. Specifically, we will need the following three lemmas.

▶ **Lemma 5.** *Suppose the insertion x takes time greater than ℓ (that is, the length of the probe sequence is greater than ℓ). Then the witness set S has size at least ℓ, and at least $|S|$ elements $x \in D$ have hashes $h(x) \in S$.*

Algorithm 1

```
1:  procedure BuildWitnesses(D, h(x), ℓ)
2:      Set Unprocessed = {h(x), h(x) + r_1, h(x) + r_2, ..., h(x) + r_{ℓ−1}}.
3:      Set S = Unprocessed and T = (0)^{ℓ−1} ∘ 1.
4:      while |Unprocessed| > 0 do                    ▷ Each iteration is called a phase.
5:          Let j = max{Unprocessed}.
6:          Remove j from Unprocessed.
7:          Append 2 to T.
8:          while true do
9:              Let Conflicts(S, j) = {(x, k) | x ∈ D, 1 ≤ k = min{i | h(x) + r_i ∈ S},
10:                                                  h(x) + r_k = j, k ≤ index(x, D)}.
11:             if |Conflicts(S, j)| > 0 then
12:                 Let k = max_k{(x, k) ∈ Conflicts(S, j) for some x}.
13:                 Let x be an arbitrary element in {x | (x, k) ∈ Conflicts(S, j)}.
14:                 Add h(x), h(x) + r_1, h(x) + r_2, ..., h(x) + r_{k−1} to Unprocessed and to S.
15:                 Append (0)^{k−1} ∘ 1 to T.
16:             else
17:                 End while loop.
18:     return (S, T)
```

Proof. The fact that S has size at least ℓ is immediate from the initialization of S in BuildWitnesses. To prove that at least $|S|$ elements $x \in D$ have hashes $h(x) \in S$, we will show a stronger claim: that every position in S is occupied by an element x whose hash $h(x)$ is in S.

First, observe that, by design, every position in S is occupied. Indeed, by the assumption that x's insertion takes time greater than ℓ, we know that $h(x), h(x) + r_1, h(x) + r_2, \ldots, h(x) + r_{\ell-1}$ (the positions initially placed in S) are all occupied. And by the definition of $\texttt{Conflicts}(S, j)$, we know that the positions added by Line 14 are also always occupied.

Now suppose for contradiction that some position $s \in S$ contains an element x whose hash $h(x)$ is not in S. Then x must conflict with S, and, in particular, x must be part of a pair (x, k) that conflicts with S at some position $j \in S$. Now consider the phase that processed j, and define S' to be the state of S at the end of the phase. Because the phase ended, we must have had $\texttt{Conflicts}(S', j) = \emptyset$. But, because (x, k) conflicts with S at position j, and since $j \in S' \subseteq S$, it must be that (x, k) also conflicts with S' at position j. This means that $(x, k) \in \texttt{Conflicts}(S', j)$, which is a contradiction. ◀

▶ **Lemma 6.** *Given the witness transcript T, along with $h(x)$ and ℓ, one can reconstruct the witness set S.*

Proof. This is accomplished by the WitnessStringToSet protocol (Algorithm 2). The basic idea is that we can use the witness transcript T to simulate the execution of BuildWitnesses. Namely, we can use 2s in T to determine boundaries between phases; and then we can use runs of 0s in T to determine the value of k that is used in each iteration of the inner while loop. This allows for us to fully reconstruct the witness set S using just $T, h(x), \ell$. ◀

▶ **Lemma 7.** *The witness transcript T is a trinary string of length $2|S|$.*

Proof. Each time that we append a 2 to T, we remove an element from `Unprocessed`. We do this $|S|$ times, so there are $|S|$ 2s in T. Each time we append a string $(0)^{k-1} \circ 1$ to T, we also add k elements to S (and vice-versa). Thus, the total number of 0s and 1s in T is $|S|$. It follows that T is a trinary string of length $2|S|$. ◀

Algorithm 2

```
1: procedure WitnessStringToSet(h(x), ℓ, T)
2:     Set Unprocessed = {h(x), h(x) + r_1, h(x) + r_2, ..., h(x) + r_{ℓ−1}}.
3:     Set S = Unprocessed.
4:     Remove prefix (0)^{ℓ−1} ∘ 1 from T.
5:     while |Unprocessed| > 0 do
6:         Let j = max_{j∈Unprocessed} j.
7:         Remove j from Unprocessed.
8:         while first character of T is not a 2 do
9:             Let k − 1 be the number of 0s at the start of T before the first 1.
10:            Define g = j − r_k.
11:            Add g, g + r_1, g + r_2, ..., g + r_{k−1} to Unprocessed and to S.
12:            Remove first k characters of T, which are (0)^{k−1} ∘ 1.
13:        Remove first character of T, which is a 2.
14:    return S
```

In addition to these structural lemmas, we will need a basic concentration bound on the probability that $|S|$ elements hash to a set S of some size.

▶ **Lemma 8.** *Let $\alpha = 1/e - \Omega(1)$. Consider a set A of αn elements, each with random hashes in $[n]$. Let $B \subseteq [n]$ be a set of some size k. The probability that at least k elements from A have hashes in B is at most $\left((1+o(1))\alpha e^{(1-\alpha)}\right)^k$, where the o-notation is in terms of k.*

Proof. By a standard Poisson approximation (see, e.g., Theorem 5.10 in [33]), and because the event of at least k elements hashing to B is monotone (adding more elements never undoes the event), we have that the probability of at least k elements hashing to B is at most $2\Pr[\text{Poisson}(\alpha k) \geq k]$. This, in turn is

$$
\begin{aligned}
&2\sum_{j\geq k} \Pr[\text{Poisson}(\alpha k) = j] \\
&= 2\sum_{j\geq k} \frac{(\alpha k)^j e^{-\alpha k}}{j!} \\
&\leq 2\sum_{j\geq k} 2^{o(j)} (\alpha k)^j e^{-\alpha k}/(j^j/e^j) && \text{(by Stirling's approximation)} \\
&\leq 2\sum_{j\geq k} 2^{o(j)} \alpha^j e^{-\alpha k + j} \\
&= 2e^{-\alpha k}\sum_{j\geq k} (\alpha e \cdot (1+o(1)))^j \\
&= O\left(e^{-\alpha k}(\alpha e \cdot (1+o(1)))^k\right) && \text{(since } \alpha = 1/e - \Omega(1)\text{)} \\
&= \left((1+o(1))\alpha e^{1-\alpha}\right)^k . && \blacktriangleleft
\end{aligned}
$$

Putting these lemmas together, we can now prove a weak version of Theorem 1 in which we do not seek to optimize the constant α.

▶ **Theorem 9.** *There exists a positive constant α such that all fixed-offset open addressing hash tables support constant-time insertions at load factor α and below. Moreover, the insertion time is bounded above by a geometric random variable with mean $O(1)$.*

Proof. Let us bound the probability that the insertion of x, which takes place at a load factor of at most α, takes time greater than $\ell > 0$. Let S and T be the witness set and witness transcript produced by BuildWitnesses$(D, h(x), \ell)$.

Suppose the insertion takes time greater than ℓ. Then, by Lemma 5, witness set S has some size $q \geq \ell$ and has the property that at least q elements $x \in D$ satisfy $h(x) \in S$.

Define $\mathcal{S}_q$ to be the set of options for what S could be if its size is q, conditioned on $h(x)$ and ℓ. For each $R \in \mathcal{S}_q$, let X_R be the event that $|\{x \in D \mid h(x) \in R\}| \geq q$. Then, in order for the insertion of x to take time greater than ℓ, event X_R must occur for some $R \in \bigcup_{q \geq \ell} \mathcal{S}_q$. By a union bound, the probability of this happening is at most

$$\sum_{q \geq \ell} \sum_{R \in \mathcal{S}_q} \Pr[X_R].$$

By Lemma 8, and assuming that $\alpha \leq e^{-1} - \Omega(1)$, this is at most

$$\sum_{q \geq \ell} \left(|\mathcal{S}_q| \cdot ((1 + o(1)) \cdot \alpha e^{1-\alpha})^q \right). \tag{1}$$

To bound $|\mathcal{S}_q|$, observe that by Lemma 6, each set $R \in \mathcal{S}_q$ corresponds to a unique witness transcript $T \in [3]^{2q}$. Thus $|\mathcal{S}_q| \leq 9^q$, which allows us to bound (1) by

$$\sum_{q \geq \ell} \left(9 \cdot (1 + o(1)) \cdot \alpha e^{1-\alpha} \right)^q. \tag{2}$$

Supposing that $\alpha e^{1-\alpha} < 1/9$, this sum evaluates to $e^{-\Omega(\ell)}$. In other words, the probability of the insertion x taking time greater than ℓ is exponentially small in ℓ. This means that the insertion takes $O(1)$ expected time. ◀

To improve upon the constant α and obtain the full version of Theorem 1, we will need a tighter bound on the number of witness transcripts corresponding to a witness set of size ℓ.

We begin by defining an infinite family of strings that contains all possible witness transcripts. Define a **witness phrase** to be a string of the form

$$P = (0)^{i_1} \circ 1 \circ (0)^{i_2} \circ 1 \circ \cdots \circ (0)^{i_k} \circ 1$$

where $0 \leq i_1 < i_2 < \cdots < i_k$. Define a **witness string** to be a string of the form

$$W = P_1 \circ 2 \circ P_2 \circ 2 \circ P_3 \circ \cdots \circ P_j \circ 2,$$

where each of $P_1, P_2, \ldots, P_j$ are witness phrases. The number j is referred to as W's **phrase count**, and the number of 0s and 1s in W is referred to as W's **zero/one count**.

Let $\mathcal{W}_{a,b}$ be the set of witness strings with zero/one-count a and phrase-count b. Notice, in particular, that $\mathcal{W}_{m,m}$ contains all witness transcripts that correspond to witness sets of size m. In Section 3.1, we will prove the following proposition, which characterizes the exact growth rate of $|\mathcal{W}_{m,m}|$.

Let $u \in (0, 1)$ minimize the quantity

$$f(u) = u^{-1} \prod_{i=1}^{\infty} (1 + u^i).$$

Then,

$$|\mathcal{W}_{m,m}| = f(u)^{(1-o(1))m}.$$

Note that, since $\log f(u) = \log u^{-1} + \sum_i \log(1+u^i)$ is within a constant factor of $\log u^{-1} + \sum_i u^i$, we know that $f(u)$ converges for any $u \in (0,1)$, and that $f(u)$ has a minimum value (since $f(u) \to \infty$ as either $u \to 0$ or $u \to 1$).

A manual calculation shows that $f(u)$ minimizes to slightly smaller than 4.51. It follows that the number of witness strings corresponding to witness sets of size m is at most $O(4.51^m)$. Plugging this into the proof of Theorem 9, (2) becomes

$$\sum_{q \geq \ell} \left(4.51 \cdot (1+o(1)) \cdot \alpha e^{1-\alpha}\right)^q.$$

In order so that, as before, this sum comes out to $e^{-\Omega(\ell)}$, we need that $\alpha e^{1-\alpha} < 1/4.51$. Another manual calculation tells us that it suffices to have $\alpha \leq 0.089$. With this modification to the proof of Theorem 9, we get Theorem 1.

3.1 Proof of Proposition 3

In this subsection, we complete the final component needed to prove Theorem 1, namely, the proof of Proposition 3, restated below.

Let $u \in (0,1)$ minimize the quantity

$$f(u) = u^{-1} \prod_{i=1}^{\infty} (1+u^i).$$

Then,

$$|\mathcal{W}_{m,m}| = f(u)^{(1-o(1))m}.$$

We begin by reinterpreting $|W_{k,m}|$ as the coefficient of x^k in a certain polynomial $G(x)$.

▶ **Lemma 10.** *Consider the formal power series*

$$G(x) = \left(\prod_{i=1}^{\infty}(1+x^i)\right)^m = \sum_i g_i x^i, \tag{3}$$

where the g_i's denote the coefficients of G, and where x is a formal variable. Then, the coefficient g_k is precisely equal to $|\mathcal{W}_{k,m}|$.

Proof. The coefficient of x^k in

$$\prod_{i=1}^{\infty}(1+x^i)$$

is equal to the number of witness phrases with zero/one-count k; and so the coefficient g_k of x^k in

$$\left(\prod_{i=1}^{\infty}(1+x^i)\right)^m$$

is equal to the number of ways to pick m witness phrases with total one-count k. This, in turn, is precisely $|\mathcal{W}_{k,m}|$. ◀

Using a simple argument about what happens when we evaluate the polynomial $G(x)$ at a given value $u \geq 0$, we can obtain the upper-bound side of Proposition 3.

▶ **Lemma 11.** *For any $u \geq 0$, we have*

$$|\mathcal{W}_{m,m}| \leq f(u)^m.$$

Proof. Let G be the formal power series from Lemma 10, and let $f(u)$ be the function defined in Proposition 3. For any $u \geq 0$, we have

$$f(u)^m = G(u)/u^m = \sum_i g_i u^i/u^m \geq g_m u^m/u^m = g_m = |\mathcal{W}_{m,m}|. \quad \blacktriangleleft$$

The lower bound is a bit more tricky. The main step is to re-interpret $G(px)/G(p)$ (where $p \in (0,1)$ and x is a formal variable) as the generating function for a random variable $X^{(p)}$ (this reinterpretation is a trick of the analysis, since $X^{(p)}$ does not appear anywhere in the original problem formulation), and then prove a concentration bound on that random variable.

We begin by establishing some basic conventions for discussing the generating function of a random variable. Supposing that A is a random variable that takes non-negative integer values, we define the **generating function** for A to be the formal power series

$$f(x) = \sum_{i \geq 0} \Pr[A = i] \cdot x^i.$$

We will make extensive use of two standard facts:

- **Fact 1:** Given any formal power series $f(x)$ such that $f(1)$ exists, the polynomial $f(x)/f(1)$ is the generating function for some random variable.
- **Fact 2:** Given two generating functions $f(x)$ and $g(x)$ for random variables A and B, the polynomial $f(x)g(x)$ is the generating function for the random variable C obtained by summing independent copies of A and B.

With these properties in mind, we now argue that, for the formal power series $G(x)$ from Lemma 11, there exists some $p \in \Theta(1) \cap (1 - \Theta(1))$ such that $G(px)/G(p)$ is the generating function for some well-behaved random variable. We will then be able to obtain our lower bound by analyzing the properties of this random variable.

▶ **Lemma 12.** *Consider the formal power series*

$$G(x) = \left(\prod_{i=1}^{\infty} (1 + x^i) \right)^m = \sum_i g_i x^i.$$

Then there exists $p \in \Theta(1) \cap (1 - \Theta(1))$ such that

$$G^{(p)}(x) = \frac{G(px)}{G(p)} = \sum_i g_i^{(p)} x^i$$

is the generating function for a random variable X with mean m and standard deviation $O(\sqrt{m})$.

Proof. By design, for any $p \in (0,1)$, we have that $\sum_i g_i^{(p)} = G^{(p)}(1) = 1$, so by Fact 1, $G^{(p)}(x)$ is the generating function for *some* random variable $X^{(p)}$. We will show that there exists some $p \in (0,1)$, satisfying $p = \Theta(1) \cap (1 - \Theta(1))$, such that $\mathbb{E}[X^{(p)}] = m$; and that for any $p = 1 - \Theta(1)$, the standard deviation of $X^{(p)}$ is $\Theta(\sqrt{m})$.

First observe that

$$G^{(p)}(x) = \frac{G(px)}{G(p)} = \left(\frac{\prod_{i=1}^{\infty}(1+(px)^i)}{\prod_{i=1}^{\infty}(1+p^i)}\right)^m.$$

It follows by Facts 1 and 2 that $X^{(p)}$ can be interpreted as the sum of m independent random variables $X_1^{(p)}, \ldots, X_m^{(p)}$ where each $X_j^{(i)}$ has generating function

$$\frac{\prod_{i=1}^{\infty}(1+(px)^i)}{\prod_{i=1}^{\infty}(1+p^i)}.$$

Observe that, when $p = 0$, we have $\mathbb{E}[X_i^{(p)}] = 0$, and as $p \to 1$, we have $\mathbb{E}[X_i^{(p)}] \to \infty$. Thus, there must exist some $p \in (0,1)$ such that $\mathbb{E}[X_i^{(p)}] = 1$. Since this choice of p is oblivious to m, it must be that $p = \Theta(1) \cap (1 - \Theta(1))$. This value of p gives us $\mathbb{E}[X^{(p)}] = m$, as desired.

Having chosen p, it remains to show that the standard deviation of $X^{(p)}$ is $O(\sqrt{m})$, or equivalently, that the variance of $X^{(p)}$ is $O(m)$. Since $X^{(p)} = \sum_{j=1}^{m} X_j^{(p)}$ is a sum of independent random variables, it suffices to show that each $X_j^{(p)}$ has variance $O(1)$. Notice, however, that the generating function for $X_j^{(p)}$ can be written as

$$\prod_{i=1}^{\infty} \frac{(1+(px)^i)}{(1+p^i)} = \prod_{i=1}^{\infty} \frac{H^{(p,i)}(x)}{H^{(p,i)}(1)},$$

where $H^{(p,i)}(x)$ is the formal power series $1 + (px)^i$. By Facts 1 and 2, it follows that we can interpret $X_j^{(p)}$ as a sum of independent random variables $Y_{j,1}^{(p)}, Y_{j,2}^{(p)}, \ldots$ where each $Y_{j,i}^{(p)}$ has generating function $\frac{(1+(px)^i)}{(1+p^i)}$. This means that

$$\mathrm{var}(X_j^{(p)}) = \sum_i \mathrm{var}(Y_{j,i}^{(p)}) \leq \sum_i \mathbb{E}[(Y_{j,i}^{(p)})^2] = \sum_i \frac{p^i i^2}{(1+p^i)} = O(1),$$

where the final equality uses $p = 1 - \Theta(1)$. As noted above, this implies that $X^{(p)}$ has variance $O(m)$, which completes the proof. ◀

We will also need a straightforward lemma bounding $|\mathcal{W}_{m\pm k,m}|$ in terms of $|\mathcal{W}_{m,m}|$ (multiplicatively) for small k.

▶ **Lemma 13.** *For any $0 \leq k < m$, we have that*

$$|\mathcal{W}_{m-k,m}| \leq |\mathcal{W}_{m,m}|$$

and that

$$|\mathcal{W}_{m+k,m}| \leq 2^{O(k \log m)} |\mathcal{W}_{m,m}|.$$

Proof. Given a string $s \in \mathcal{W}_{m-k,m}$, we can obtain a string $\phi(s) \in \mathcal{W}_{m,m}$ by taking the final run of 1s in s and extending the length of that run by k. This function is injective, implying that $|\mathcal{W}_{m-k,m}| \leq |\mathcal{W}_{m,m}|$.

Given a string $s \in \mathcal{W}_{m+k,m}$, it is possible to remove some set of $O(k)$ characters (zeros and ones) in order to obtain a valid string $s' \in \mathcal{W}_{m,m}$; let $\psi(s)$ be the lexicographically smallest such string s' that one can achieve in this way. For a given $s' \in \mathcal{W}_{m,m}$, we can bound $|\psi^{-1}(s')| \leq 2^{O(k \log m)}$, since there are at most $2^{O(k \log m)}$ ways to *add* $O(k)$ characters to get a string $s \in \mathcal{W}_{m+k,m}$. It follows that $|\mathcal{W}_{m+k,m}| \leq 2^{O(k \log m)} |\mathcal{W}_{m,m}|$. ◀

Finally, we can put the pieces together in order to get the lower-bound side of Proposition 3.

▶ **Lemma 14.** *Let*

$$f(u) = u^{-1} \prod_{i=1}^{\infty} (1 + u^i).$$

Then, there exists $p \in (0, 1)$ *such that*

$$f(p)^m \leq 2^{o(m)} |\mathcal{W}_{m,m}|.$$

Proof. Consider the formal power series

$$G(x) = \left(\prod_{i=1}^{\infty} (1 + x^i) \right)^m = \sum_i g_i x^i.$$

By Lemma 12, there exists $p \in \Theta(1) \cap (1 - \Theta(1))$ such that

$$G^{(p)}(x) = \frac{G(px)}{G(p)} = \sum_i g_i^{(p)} x^i$$

is the generating function for a random variable X with mean m and standard deviation $O(\sqrt{m})$. By Chebyshev's inequality, at least half of X's probability mass must be concentrated between $m - O(\sqrt{m})$ and $m + O(\sqrt{m})$. That is, for some positive constant d,

$$\sum_{k=-d\sqrt{m}}^{d\sqrt{m}} g_{m+k}^{(p)} \geq \frac{1}{2} \sum_{i=0}^{\infty} g_i^{(p)}.$$

Since $g_i^{(p)} = p^i g_i / G(p)$, we can multiply both sides by $G(p)$ to get

$$\sum_{k=-d\sqrt{m}}^{d\sqrt{m}} p^{m+k} g_{m+k} \geq \frac{1}{2} \sum_{i=0}^{\infty} p^i g_i.$$

Since $p = \Theta(1)$ and since $g_{m+k} = |\mathcal{W}_{m+k,m}| \leq 2^{O(|k| \log m)} |\mathcal{W}_{m,m}|$ (by Lemmas 10 and 13), we have that each term $p^{m+k} g_{m+k}$ in the left sum is at most

$$2^{O(\sqrt{m} \log m)} p^{m+k} |\mathcal{W}_{m,m}| = 2^{O(\sqrt{m} \log m)} p^m |\mathcal{W}_{m,m}|.$$

This means that the entire left sum is at most

$$O(\sqrt{m}) 2^{O(\sqrt{m} \log m)} |\mathcal{W}_{m,m}| \leq 2^{o(m)} p^m |\mathcal{W}_{m,m}|.$$

Therefore,

$$\sum_{i=0}^{\infty} p^i g_i \leq 2^{o(m)} p^m |\mathcal{W}_{m,m}|,$$

which implies that

$$G(p)/p^m \leq 2^{o(m)} |\mathcal{W}_{m,m}|.$$

Finally, since $G(p)/p^m = f(p)^m$, the proof is complete. ◀

4 Chunked Fixed-Offset Open Addressing

We now turn our attention to obtaining bounds for chunked fixed-offset open-addressed hash tables. Recall that a fixed-offset open-addressed hash table is said to be chunked with chunk-size B if, for each $i \geq 0$ and $j \in \{0, \ldots, B-1\}$, we have $r_{iB+j} = r_{iB} + j$. As terminology, for a given element x, we will refer to the sequence $h(x) + r_{iB}, \ldots, h(x) + r_{iB+B-1}$ as the ***i*-th block** in x's probe sequence. Our goal will be to prove the following theorem.

▶ **Theorem 2.** *There exists a constant $\alpha \in (0,1)$ such that the following is true. Consider a chunked fixed-offset open-addressed hash table with chunk size B. Any insertion of an element x at a load factor $\alpha = 1 - 1/q$ satisfying $q \leq \alpha\sqrt{B/\log B}$ takes expected time at most $O(q^2)$. Moreover, the insertion time is bounded above by a geometric random variable with mean $O(q^2)$.*

The theorem can be interpreted as saying that, whenever $B = \omega(1)$, it is possible to support load factors of the form $\alpha = 1 - q$ were $q = o(1)$. We remark that the time bound $O(q^2)$ is optimal in general, since linear probing is an example of a chunked open-addressed hash table that has insertion time $\Theta(q^2)$ at load factor $1 - q$.

We will prove Theorem 2 with a slightly more intricate analysis of the BuildWitnesses procedure from Section 3. To do this, we must first define the notion of an *analytical run*.

In lines 2 and 14 of BuildWitnesses, when we add some sequence $Q = h(x), h(x)+r_1, \ldots$ to `Unprocessed`, we can break the positions that we add into **analytical runs**, where the first (up to) B entries of Q form the first analytical run, the next (up to) B entries form the next analytical run, and so on. The execution of the line creates $\lceil |Q|/B \rceil$ analytical runs, and all but the final one have size B.

Say that two analytical runs r_1, r_2 (not necessarily created by the same iteration of Line 14) are **adjacent** if they represent sub-intervals of the form $[i, j]$ and $[j+1, k]$ for some i, j, k. Finally, define the **adjusted size** of an analytical run r to be B if the sequence Q that added r to S satisfied $|Q| \geq B$ or if the analytical run r was created by Line 2 of the algorithm, and define the adjusted size to be $|r|$ otherwise.

A critical step in the proof of Theorem 2 is to observe that, whenever two analytical runs are adjacent, their adjusted sizes must sum to at least B.

▶ **Lemma 15.** *Consider the execution of BuildWitnesses$(D, h(u), \ell)$ for some $\ell > 0$. Consider two adjacent analytical runs r_1, r_2 with adjusted sizes s_1, s_2. Then $s_1 + s_2 \geq B$.*

Proof. Let x_1 and x_2 be the elements that created runs r_1 and r_2. Let t_1 and t_2 be the iterations at which x_1 and x_2 are added to S by BuildWitnesses, let $\texttt{Conflicts}_1$ and $\texttt{Conflicts}_2$ be the conflict sets that are constructed in the while-loop iterations that add x_1 and x_2 to S, and let S_1 and S_2 be the states of S immediately before x_1 and x_2 are added to S, respectively.

If either of r_1 or r_2 have adjusted size B, then the claim is trivial. We can therefore assume without loss of generality that r_1, r_2 are the first runs in the probe sequences for x_1 and x_2 and that $|r_1|, |r_2| < B$. Note that, as immediate consequences, we have that r_1 and r_2 were created by Line 14 of BuildWitnesses (rather than Line 2); that $h(x_1) < h(x_2)$; and that x_2 was added to S before x_1 (i.e., $t_2 < t_1$).

Let ℓ be the right endpoint of r_2. We claim that

$$\ell \geq h(x_1) + B - 1. \tag{4}$$

This would imply that $|r_1| + |r_2| \geq \ell - h(x_1) + 1 \geq B$, completing the proof.

Suppose for contradiction that (4) does not hold. Let $(x_2, k_2) \in \texttt{Conflicts}_2$ be the pair containing x_2 in $\texttt{Conflicts}_2$. Then the final position ℓ of run r_2 satisfies $\ell + 1 = h(x_2) + k_2$, and $\texttt{Conflicts}_2 = \texttt{Conflicts}(S_2, \ell + 1)$ expands to

$$\{(x,k) \mid x \in D, \\ 1 \le k = \min\{i \mid h(x) + r_i \in S_2\},\ h(x) + r_k = \ell + 1,\ k \le \text{index}(x, D)\}.$$

We will show that this forces the pair $(x_1, \ell - h(x_1) + 1)$ to be in $\texttt{Conflicts}_2$. But the fact that $h(x_1) < h(x_2)$ means that BuildWitnesses will rather select $(x_1, \ell - h(x_1) + 1)$ than $(x_2, \ell - h(x_2) + 1)$ from $\texttt{Conflicts}_2$ (see Line 12 of BuildWitnesses). This contradicts the fact that x_2 is added to S at iteration t_2 in the execution of BuildWitnesses.

It remains to prove that $(x_1, \ell - h(x_1) + 1) \in \texttt{Conflicts}_2$. Since $\ell < h(x_1) + B - 1$ (by assumption) and since $h(x_1) < h(x_2)$, we know that $\ell + 1 = h(x_1) + k_1$ for some $k_2 < k_1 < B$. To prove that $(x_1, k_1) \in \texttt{Conflicts}_2$, we must show that

$$k_1 = \min\{i \mid h(x_1) + r_i \in S_2\} \tag{5}$$

and that

$$k_1 \le \text{index}(x_1, D). \tag{6}$$

We begin by showing (5). The fact that $(x_2, k_2) \in \texttt{Conflicts}_2$ tells us that $k_2 = \min\{i \mid h(x_2) + r_i \in S_2\}$, which implies that $h(x_2), h(x_2) + 1, \ldots, \ell \notin S_2$. The fact that $(x_1, h(x_2) - h(x_1)) \in \texttt{Conflicts}_1$ tells us that $h(x_2) - h(x_1) = \min\{i \mid h(x_1) + r_i \in S_1\}$, which implies that $h(x_1), h(x_1) + 1, \ldots, h(x_2) - 1 \notin S_1$. Since BuildWitnesses always processes $\max\{\texttt{Unprocessed}\}$, we know that, between iterations t_2 and t_1, BuildWitnesses processes only values of j satisfying $j \ge h(x_2)$. Thus $S_2 \cap [h(x_2) - 1] \subseteq S_1 \cap [h(x_2) - 1]$. So the fact that $h(x_1), h(x_1) + 1, \ldots, h(x_2) - 1 \notin S_1$ implies that $h(x_1), h(x_1) + 1, \ldots, h(x_2) - 1 \notin S_2$. Therefore, $h(x_1), h(x_1) + 1, \ldots, \ell \notin S_2$, which implies (5).

Finally, we complete the proof by establishing (6). If we had $k_1 = \ell + 1 - h(x_1) > \text{index}(x_1, D)$, then x_1 would have to reside in one of positions $h(x_1), h(x_1) + 1, \ldots, \ell$. If x_1 resides in any of positions $h(x_2), \ldots, \ell - 1$, then at time $t_2 < t_1$, that position must have been vacant, which implies that x_2 could have used it – but this contradicts the fact that x_2 created run r_2. On the other hand, if x_1 resides in any of positions $h(x_1), h(x_1) + 1, \ldots, h(x_2) - 1$, then x_1 does not conflict with position $h(x_2)$, which contradicts the fact that x_1 created run r_1. Thus, x_1 does not reside in any of positions $h(x_1), h(x_1) + 1, \ldots, \ell$, which means that $k_1 = \ell + 1 - h(x_1) \le \text{index}(x_1, D)$. ◀

We can build on Lemma 15 to make a claim about the average size of all of the analytical runs, namely that, if the analytical runs are not *all* adjacent, then their average (non-adjusted!) size must be $\Omega(B)$. It may see odd at first glance that we need to separate out the case where all of the analytical runs are adjacent, but we will see later on that this case actually behaves very differently from the other cases. (Indeed, it is the case that causes the insertion time to be $O(q^2)$ instead of $O(1)$!)

▶ **Lemma 16.** *Consider the execution of* BuildWitnesses$(D, h(u), \ell)$ *for some* $\ell > 0$. *Let* $r_1, r_2, \ldots, r_j$ *be the analytical runs that we add to* `Unprocessed` *during the execution of the algorithm. If* $r_1, r_2, \ldots, r_j$ *are not all adjacent, then* $\sum_{i=1}^{j} |r_i| \ge B$ *and*

$$\frac{1}{j} \sum_{i=1}^{j} |r_i| \ge \Omega(B).$$

Proof. Define $s_1, s_2, \ldots, s_j$ be the sizes of the analytical runs and let $s'_1, s'_2, \ldots, s'_j$ to be the adjusted sizes of the analytical runs. Since $r_1, r_2, \ldots, r_j$ are not all adjacent, we know that at least one r_i satisfies $|s_i| = B$ (namely, the first time we add an r_i that is not adjacent with the other r_js added so far). Therefore, the lemma is immediately true if $j = O(1)$. Suppose for the rest of the proof that $j = \omega(1)$.

With the possible exception of the first analytical run added by Line 2 (recall analytical runs created by Line 2 automatically have adjusted size B), we have that for every analytical run r whose true size s_i is smaller than its adjusted size s'_i, there is some run with true size B. Indeed, such a run r must be the final analytical run added by some iteration of Line 14, and that that iteration of Line 14 must have added at least one analytical run with true size B. It follows that

$$B + \sum_{i=1}^{j} s_i \geq \frac{1}{2} \sum_{i=1}^{j} s'_i.$$

Because $j = \omega(1)$, this implies

$$o(B) + \frac{1}{j} \sum_{i=1}^{j} s_i \geq \Omega\left(\frac{1}{j} \sum_{i=1}^{j} s'_i\right).$$

Thus, to complete the proof, it suffices to show that the right side is $\Omega(B)$.

Now consider a maximal sequence of adjacent analytical runs $r_{a_1}, r_{a_2}, r_{a_3}, \ldots, r_{a_k}$. If $k = 1$, then the adjusted size of the run r_{a_1} is guaranteed to be B (because either its true size is B and it was created by Line 14, or it was created by Line 2 which only creates analytical runs with adjusted sizes of B). If $k > 1$, then Lemma 15 tells us that the runs $r_{a_1}, r_{a_2}, r_{a_3}, \ldots, r_{a_k}$ have adjusted sizes summing to at least $\lfloor k/2 \rfloor \cdot B \geq kB/3$.

Therefore, the average adjusted size over all analytical runs is at least $B/3$. This means that

$$\frac{1}{2j} \sum_{i=1}^{j} s'_i \geq B/6,$$

which completes the proof. ◀

Finally, using the fact that the average analytical run size is $\Omega(B)$, we can obtain a bound on the number of runs of 0s, 1s, and 2s in the witness transcript T. (Here, we are using the term "run" in the string sense, e.g., a run of 0s is a maximal sequence of consecutive zeros in the string.)

▶ **Lemma 17.** *Consider the execution of* BuildWitnesses$(D, h(u), \ell)$ *for some* $\ell > 0$ *and suppose that the analytical runs that are created are not all adjacent. Then the resulting witness transcript* T *is a trinary string that has at most* $O(|T|/B)$ *runs of 0s, 1s, and 2s.*

Proof. Let $s_1, s_2, \ldots, s_j$ be the sizes of the analytical runs that we add to `Unprocessed` during the execution of the algorithm. The total number of 1s and 0s in T is exactly $\sum_i s_j$, and the total number of 2s in T is also exactly $\sum_i s_j$, so $|T| = 2\sum_i s_j$. On the other hand, both the number of 1s in T and the number of *runs* of 0s in T are bounded by j. It follows that the total number of runs of 0s and 1s in T is at most $2j$, which by Lemma 16 is at most $O(\sum_i s_i/B) = O(|T|/B)$. The number of runs of 2s is at most one greater than the number of runs of 0s and 1s (this is true for any trinary string), so the total number of runs overall in T is at most $O(|T|/B)$. ◀

The fact that the witness transcript T has so few runs will lead to a much smaller bound on the number of options for T than we had in the non-chunked setting. This is the key insight that makes it possible to prove Theorem 2. As noted earlier, the proof actually separates into two cases, the case where the analytical runs are all adjacent (this case contributes $O(q^2)$ to the running time) and the case where they are not (this case contributes only $O(1)$ to the running time, and is where we make use of the previous lemmas).

Proof of Theorem 2. Suppose the insertion takes time $t \geq 0$, and let us bound $\Pr[t > \ell]$ for some ℓ. Consider the execution of BuildWitnesses$(D, h(u), \ell)$, producing witness set S and witness transcript T. Let $r_1, r_2, \ldots$ be the analytical runs produced by the algorithm. Let $\mathcal{C}$ be the indicator random variable for the event that $r_1, r_2, \ldots$ are all adjacent, and let $\overline{\mathcal{C}}$ be the indicator random variable for the event that they are not all adjacent.

We begin by considering the event $\mathcal{C} = 1$. In this case, the witness set S is a contiguous interval $[i, j]$ containing $h(u)$. By Lemma 5, if $t > \ell$, then at least $|S| \geq \ell$ elements y in the hash table have hashes $h(y) \in V$. From the standard analysis of linear-probing hash tables (e.g., Lemma 15 of [9]), we know that the probability of any such interval S existing is at most $2^{-\Omega(\ell/q^2)}$. We have therefore shown that

$$\Pr[t\mathcal{C} > \ell] \leq 2^{-\Omega(\ell/q^2)}.$$

So, $\mathbb{E}[t\mathcal{C}] = O(q^2)$ and for any $j \geq 1$, $\Pr[t\mathcal{C} > jq^2] \leq 2^{-\Omega(j)}$.

Now, for the rest of the proof, consider the event $\overline{\mathcal{C}}$. Define $\overline{t} = t\overline{\mathcal{C}}$. We will complete the proof by showing that $\mathbb{E}[\overline{t}] = O(1)$ and that for any $j \geq 1$, $\Pr[\overline{t} > jq^2] \leq 2^{-\Omega(j)}$.

Supposing that $\overline{\mathcal{C}}$ occurs, we have by Lemma 16 that $|T| \geq B$. We also have trivially that $|T| \geq \ell$, so $|T| \geq \max(B, \ell)$. The fact that we only need to consider $|T| \geq \max(B, \ell)$ will be important through the rest of the proof.

By Lemma 17, T has at most $O(|T|/B)$ runs. By standard counting arguments, the number of trinary strings of length ℓ' with $O(\ell'/B)$ runs is at most $B^{O(\ell'/B)}$. It follows, that for a given length $\ell' > \max(B, \ell)$ for T, the number of options for what T could be is at most $B^{O(\ell'/B)}$. Since the witness set S is fully determined by T, and is exactly half of T's size (Lemma 7), the number of options for S of a given size $\ell'' \geq \max(B, \ell)/2$ is at most $B^{O(\ell''/B)}$.

On the other hand, if the insertion took time more than ℓ, then, in order for a given option for S to occur, there would need to be at least $|S|$ elements y in the hash table satisfying $h(y) \in S$ (Lemma 5). For a given set S, the expected number of elements y satisfying $h(y) \in S$ is

$$\alpha|S| \leq (1 - 1/(c\sqrt{B/\log B}))|S| = |S| - \frac{\sqrt{S \log B}}{\sqrt{B}\alpha}\sqrt{|S|}.$$

By a Chernoff bound, the probability of a given candidate witness set S of size $\ell'' \geq \max(B, \ell)/2$ having ℓ'' elements hash to it is at most

$$2^{-\Omega(|S| \log B/(B\alpha^2))} = B^{-\Omega(\ell''/B)/\alpha^2}.$$

Taking a union bound over all $B^{O(\ell''/B)}$ options for S, the probability that any S of size $\ell'' \geq \max(B, \ell)/2$ occurs is at most

$$B^{O(\ell''/B)} B^{-\Omega(\ell''/B)/\alpha^2}.$$

Setting α to be a sufficiently small positive constant, this is at most

$$1/B^{10\ell''/B}.$$

Summing over all $\ell'' \geq \max(B,\ell)/2$, the probability that BuildWitnesses$(D, h(u), \ell)$ produces a saturated witness set S of size at least $\max(B,\ell)/2$ is at most

$$\sum_{\ell'' \geq \max(B,\ell)/2} 1/B^{10\ell''/B} \leq 1/B^{\max(B,\ell)/B}.$$

But, the only way that $\bar{t} \geq \ell$ can occur is if such a witness set S is produced. Thus

$$\Pr[\bar{t} \geq \ell] \leq 1/B^{\max(B,\ell)/B} = B^{B/2+\ell/(2B)} \leq q^{-\Omega(q^2 \log q + \ell/(q^2 \log q))} \leq q^{-\Omega(q^2 \log q)} \cdot 2^{-\Omega(\ell/q^2)}.$$

It follows that $\mathbb{E}[\bar{t}] \leq O(1)$, and that, for $j \geq 1$, $\Pr[\bar{t} > jq^2] \leq 2^{-\Omega(j)}$, as desired. ◀

References

1. Abseil, 2017. Accessed: 2020-11-06. URL: https://abseil.io/.
2. Ole Amble and Donald Ervin Knuth. Ordered hash tables. *The Computer Journal*, 17(2):135–142, January 1974. doi:10.1093/comjnl/17.2.135.
3. Yuriy Arbitman, Moni Naor, and Gil Segev. De-amortized cuckoo hashing: Provable worst-case performance and experimental results. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP 2009)*, volume 5555 of *Lecture Notes in Computer Science*, pages 107–118, 2009. doi:10.1007/978-3-642-02927-1_11.
4. Guy de Balbine. *Computational analysis of the random components induced by a binary equivalence relation*. PhD thesis, California Institute of Technology, 1968.
5. Vladimir Batagelj. The quadratic hash method when the table size is not a prime number. *Communications of the ACM*, 18(4):216–217, 1975.
6. Daniel Bauer. Columbia COMS W3134: Data structures in Java — Lecture 12: Introduction to hashing, October 2015. URL: http://www.cs.columbia.edu/~bauer/cs3134-f15/slides/w3134-1-lecture12.pdf.
7. James R Bell and Charles H Kaman. The linear quotient hash code. *Communications of the ACM*, 13(11):675–676, 1970.
8. Michael A Bender, Martín Farach-Colton, John Kuszmaul, William Kuszmaul, and Mingmou Liu. On the optimal time/space tradeoff for hash tables. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1284–1297, 2022.
9. Michael A Bender, Bradley C Kuszmaul, and William Kuszmaul. Linear probing revisited: Tombstones mark the demise of primary clustering. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1171–1182. IEEE, 2022.
10. Pedro Celis, Per-Åke Larson, and J. Ian Munro. Robin Hood hashing (preliminary report). In *26th Annual Symposium on Foundations of Computer Science (FOCS'85)*, pages 281–288, Portland, Oregon, USA, 21–23 October 1985. doi:10.1109/SFCS.1985.48.
11. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, USA, 3rd edition, 2009.
12. Lilian de Greef. UW CSE 373: Data structures and algorithims — Lecture 7: Hash table collisions, summer 2017. URL: https://courses.cs.washington.edu/courses/cse373/17su/lectures/Lecture%2007%20-%20Hash%20Table%20Collisions.pdf.
13. Erik D. Demaine, Friedhelm Meyer auf der Heide, Rasmus Pagh, and Mihai Pătraşcu. De dictionariis dynamicis pauco spatio utentibus (*lat.* on dynamic dictionaries using little space). In *Proceedings of the 7th Latin American Symposium on Theoretical Informatics (LATIN 2006)*, volume 3887 of *Lecture Notes in Computer Science*, pages 349–361, Valdiva, Chile, 20–24 March 2006. doi:10.1007/11682462_34.
14. Martin Dietzfelbinger, Andreas Goerdt, Michael Mitzenmacher, Andrea Montanari, Rasmus Pagh, and Michael Rink. Tight thresholds for cuckoo hashing via XORSAT. In *37th International Colloquium on Automata, Languages and Programming (ICALP 2010)*, pages 213–225, 2010.

15 Adam Drozdek and Donald L. Simon. *Data Structures in C*. PWS, Boston, Massachusetts, USA, 1995.

16 A Ecker. The period of search for the quadratic and related hash methods. *The Computer Journal*, 17(4):340–343, 1974.

17 Nikolaos Fountoulakis, Megha Khosla, and Konstantinos Panagiotou. The multiple-orientability thresholds for random hypergraphs. *Combinatorics, Probability and Computing*, 25(6):870–908, 2016.

18 David Gries and Doug James. Cornell CS210: Object-oriented programming and data structures – recitation week 8: Hashing, fall 2014. URL: `https://www.cs.cornell.edu/courses/cs2110/2014fa/recitations/recitation08/HashPresentation.pptx`.

19 Leo J Guibas and Endre Szemeredi. The analysis of double hashing. In *Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 187–191, 1976.

20 Leo J Guibas and Endre Szemeredi. The analysis of double hashing. *Journal of Computer and System Sciences*, 16(2):226–274, 1978.

21 Leonidas J Guibas. *The analysis of hashing algorithms*. PhD thesis, Stanford University., 1976.

22 F Robert A Hopgood and J Davenport. The quadratic hash method when the table size is a power of 2. *The Computer Journal*, 15(4):314–315, 1972.

23 Gregory Kesden. CMU 15-310: System-level software development — hashing review, 2007. Accessed 31-May-2021. URL: `https://www.andrew.cmu.edu/course/15-310/applications/ln/hashing-review.html`.

24 Donald E Knuth. Notes on "open" addressing. *Unpublished memorandum*, pages 11–97, 1963.

25 Donald E Knuth. *The Art of Computer Programming: Volume 3: Sorting and Searching*. Addison-Wesley Professional, 1998.

26 Alan G Konheim and Benjamin Weiss. An occupancy discipline and applications. *SIAM Journal on Applied Mathematics*, 14(6):1266–1274, 1966.

27 Tianxiao Li, Jingxun Liang, Huacheng Yu, and Renfei Zhou. Tight cell-probe lower bounds for dynamic succinct dictionaries. *arXiv preprint arXiv:2306.02253*, 2023.

28 Tianxiao Li, Jingxun Liang, Huacheng Yu, and Renfei Zhou. Dynamic dictionary with subconstant wasted bits per key. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 171–207. SIAM, 2024.

29 Mingmou Liu, Yitong Yin, and Huacheng Yu. Succinct filters for sets of unknown sizes. In *Proceedings 47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 79:1–79:19, Saarbrücken, Germany, 8–11 July 2020. `doi:10.4230/LIPIcs.ICALP.2020.79`.

30 George Lueker and Mariko Molodowitch. More analysis of double hashing. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 354–359, 1988.

31 George S Lueker and Mariko Molodowitch. More analysis of double hashing. *Combinatorica*, 13(1):83–96, 1993.

32 Ward Douglas Maurer. Programming technique: An improved hash code for scatter storage. *Communications of the ACM*, 11(1):35–38, 1968.

33 Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

34 Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

35 W Wesley Peterson. Addressing for random-access storage. *IBM journal of Research and Development*, 1(2):130–146, 1957.

36 Charles E Radke. The use of quadratic residue research. *Communications of the ACM*, 13(2):103–105, 1970.

37 Rajeev Raman and Satti Srinivasa Rao. Succinct dynamic dictionaries and trees. In *Automata, Languages and Programming*, pages 357–368, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

38 Stefan Richter, Victor Alvarez, and Jens Dittrich. A seven-dimensional analysis of hashing methods and its implications on query processing. *PVLDB*, 9(3):96–107, 2015.

39 David G. Sullivan. Harvard CS S-111: Intensive introduction to computer science using Java – unit 9, part 4: Hash tables, summer 2021. URL: `https://sites.fas.harvard.edu/~libs111/files/lectures/unit9-4.pdf`.

40 Mark Allen Weiss. *Data Structures and Problem Solving using C++*. Addison-Wesley, Reading, Massachusetts, USA, 2000.

Optimal Non-Adaptive Cell Probe Dictionaries and Hashing

Kasper Green Larsen
Aarhus University, Denmark

Rasmus Pagh
BARC, University of Copenhagen, Denmark

Giuseppe Persiano
Università di Salerno, Italy
Google, New York, NY, USA

Toniann Pitassi
Columbia University, New York, NY, USA

Kevin Yeo
Columbia University, New York, NY, USA
Google, New York, NY, USA

Or Zamir
Tel Aviv University, Israel

Abstract

We present a simple and provably optimal non-adaptive cell probe data structure for the static dictionary problem. Our data structure supports storing a set of n key-value pairs from $[u] \times [u]$ using s words of space and answering key lookup queries in $t = O(\lg(u/n)/\lg(s/n))$ non-adaptive probes. This generalizes a solution to the membership problem (i.e., where no values are associated with keys) due to Buhrman et al. We also present matching lower bounds for the non-adaptive static membership problem in the deterministic setting. Our lower bound implies that both our dictionary algorithm and the preceding membership algorithm are optimal, and in particular that there is an inherent complexity gap in these problems between no adaptivity and one round of adaptivity (with which hashing-based algorithms solve these problems in constant time).

Using the ideas underlying our data structure, we also obtain the first implementation of a n-wise independent family of hash functions with optimal evaluation time in the cell probe model.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases non-adaptive, cell probe, dictionary, hashing

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.104

Category Track A: Algorithms, Complexity and Games

Related Version This paper is a merge and revision of two previous reports.
Previous Version: `https://arxiv.org/abs/2001.05053` [19]
Previous Version: `https://arxiv.org/abs/2308.16042` [14]

Funding *Kasper Green Larsen*: Supported by Independent Research Fund Denmark (DFF) Sapere Aude Research Leader grant No 9064-00068B.
Rasmus Pagh: Supported by grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.
Giuseppe Persiano: Partially supported by a FARB grant, from Università di Salerno.
Toniann Pitassi: Research supported by NSF AF:Medium 2212136
Or Zamir: Supported in part by Len Blavatnik and the Blavatnik Family foundation.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 104; pp. 104:1–104:12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The static membership problem is arguably the simplest and most fundamental data structure problem. In this problem, the input is a set S of n integer keys $x_1, \ldots, x_n \in [u] = \{0, \ldots, u-1\}$ and the goal is to store them in a data structure, such that given a query key $x \in [u]$, the data structure supports reporting whether $x \in S$.

The classic solution to the membership problem is to use *hashing*, suggested as early as by Tarjan-Yao [23]. The textbook hashing-based solution is hashing with chaining, where one draws a random *hash function* $h : [u] \to [m]$ and creates an array A with $m = O(n)$ entries. Each entry $A[i]$ of the array stores a linked list of all keys $x \in S$ such that $h(x) = i$. To answer a membership query for x, we compute $h(x)$ and scan the linked list in entry $A[h(x)]$. If h is drawn from a *universal* family of hash functions, the time to answer queries is $O(1)$ in expectation. The expected query time can be made worst case $O(1)$ using e.g. perfect hashing [11] or (static) Cuckoo hashing [16, 17]. All of the above solutions may also be easily extended to solve the dictionary problem in which the data to be stored is a set of n key-value pairs $\{(x_i, y_i)\}_{i=1}^n$. Upon a query x, the data structure must return the value y_i such that $x_i = x$, or report that no such pair exists.

1.1 Adaptivity and Membership

A common feature of all hashing based solutions to the membership and dictionary problem, is that they are *adaptive*. That is, the memory locations they access depend heavily on the random choice of hash functions. In particular, to answer a query we first need to read the description of the chosen hash function, and only based on that we can compute the next memory cells we should access. A *non-adaptive* data structure has the property that the memory cells to access on a query x are completely determined from x itself. Non-adaptive data structures are studied for several reasons, a common type being computational settings in which interaction with memory is either expensive or limited. Non-adaptive data structures allow retrieving all necessary memory cells in *parallel* when answering a query, circumventing any memory-access related latency. This property also allows simpler implementation of the data structure under cryptographic settings, such as encrypted computation with Fully Homomorphic Encryption (see [25] for more details on the importance of non-adaptive querying in cryptography).

In this work, we present a non-adaptive dictionary algorithm in which a query needs to only access logarithmically many memory cells, and also prove a matching lower bound (which holds even for the static membership problem).

Unlike the textbook solution of hashing with chaining, which requires many rounds of adaptivity due to scanning a linked list, other solutions (e.g., cuckoo hashing) only need one round of adaptivity (i.e., first they read the description of the hash function, and then read memory cells that are determined only by the query and the hash function). Our results imply that a single round of adaptivity is necessary and sufficient to reduce the query time from super-constant to constant.

The Cell Probe Model. The cell probe model by Yao [24] is the de-facto model for proving data structure lower bounds. In this model, a data structure consists of a memory of s cells with integer addresses $0, \ldots, s-1$, each storing w bits. Computation is free of charge in this model and only the number of memory cells accessed/probed when answering a query counts towards the query time. A lower bound in the cell probe model thus applies to any data structure implementable in the classic word-RAM upper bound model.

Previous Work. Buhrman, Miltersen, Radhakrishnan and Venkatesh [6] showed that it is possible to store a data structure of size $O(n \lg u)$ bits such that membership queries can be answered in $O(\lg u)$ non-adaptive *bit* probes (i.e., the cell probe model with $w = 1$). This of course implies a membership data structure with $O(\lg u)$ probes in the cell probe model, but it is not clear how to extend it to solve the dictionary problem with the same time and space complexity. Furthermore, the data structure by Buhrman et al. is non-explicit in the sense that they give a randomized argument showing existence of an efficient data structure. Buhrman et al. also show a lower bound of $t = \Omega(\lg(u/n)/\lg(s/n))$ bit probes. In the setting where n is polynomially smaller than u and s is $O(n)$ this matches the upper bound up to constant factors (but it is possible that a tighter analysis can be made). Alon and Feige [1] as well as Garg and Radhakrishnan [12] studied space lower bounds for dictionary data structures with three non-adaptive probes in the bit probe model. The best lower bound shows that space of $s = \Omega(\sqrt{un})$ is necessary.

Berger et al. [3] study the non-adaptive dictionary problem, but in the I/O model, i.e., a single memory access can retrieve $B \geq 1$ keys or values. In the Word RAM model this corresponds to having word size $B \lg u$. This means that their strongest results for the dictionary problem would require word size $\Omega(\lg(n) \lg(u))$ – as we will see later, our results hold for word size $\lg u$.

Brody et al. [5] present a *dynamic* non-adaptive data structure for the *predecessor search* problem, allowing insertions and deletions of keys while supporting predecessor queries in $O(\lg u)$ probes. A predecessor query for a key x must return the largest $x' \in S$ such that $x' \leq x$. Such a data structure clearly also supports membership queries. However, their data structure critically uses $s = \Theta(2^w) = \Theta(u)$ memory. For the membership problem in this setting, a bit-vector with constant time operations suffices. Brody et al. [5] however prove that for dynamic data structures for predecessor search, this query time is optimal even with $\Theta(u)$ space. Boninger et al. [4] as well as Ramamoorthy and Rao [21] also study lower bounds for the non-adaptive dynamic predecessor problem. Relating their results to the non-adaptive static dictionary problem, the two works show query time must be $t = \Omega(\lg u/\lg w)$ and $t = \Omega(\lg u/(\lg\lg u + \lg w))$ respectively in the cell probe model. To our knowledge, these are the highest known lower bounds for the static, non-adaptive dictionary problem.

This still leaves open the problem of obtaining an optimal static and non-adaptive membership data structure, in both the word-RAM model, and in the cell probe model.

Our Contribution. In this work, we present a simple and optimal non-adaptive cell probe data structure for the dictionary and membership problem:

▶ **Theorem 1.** *For any $s = \Omega(n)$, there is a non-adaptive static cell probe data structure for the dictionary problem, storing n key-value pairs $(x_i, y_i) \in [u] \times [u]$ using s memory cells of $w = \Theta(\lg u)$ bits and answering queries in $t = O(\lg(u/n)/\lg(s/n))$ probes.*

As stated in the theorem, our data structure is implemented in the cell probe model, meaning that we treat computation as free of charge. Implementing the data structure in the more standard upper bound model, the word-RAM, would require the construction of a certain type of explicit bipartite expander graph.

Compared to prior works (such as [6, 3]), our construction shows that we may rely on a significantly weaker expansion argument. Past constructions required an orientability argument to assign memory to keys that required expanders with a strong unique-neighbors property. In contrast, our construction utilizes weaker non-contractive expanders to argue that there is sufficient capacity to accommodate storage of all keys (using Hall's theorem).

This directly translates to a logarithmic improvement in space usage. Namely, we only require the existence of t-left-regular bipartite graphs with expansion factor one; however our bipartite graph is highly imbalanced. Our expansion property corresponds to an imbalanced disperser, and therefore is well-studied and has other applications (e.g., [13]). Such dispersers exist by a counting argument, but it remains an open problem to obtain explicit constructions. A recent work [2] constructs explicit expanders that may be plugged into our construction to obtain an explicit RAM upper bound. However, this incurs a poly-logarithmic blowup and obtaining a tight explicit RAM upper bound would require better explicit expanders.

We also present a matching lower bound for the non-adaptive dictionary and membership problem in the cell probe model:

▶ **Theorem 2.** *For any non-adaptive static cell probe data structure for the dictionary problem storing n key-value pairs $(x_i, y_i) \in [u] \times [u]$ using s memory cells of w bits and answering queries in t probes must satisfy*

$$t = \Omega\left(\min\left\{\frac{n\lg(u/n)}{w}, \frac{\lg(u/n)}{\lg(sw/(n\lg(u/n)))}\right\}\right).$$

Our lower bound shows that adaptivity is crucial to obtain constant query time. In particular, non-adaptive data structures require super-constant query time while well-known constructions with adaptivity (such as cuckoo hashing) can obtain constant query time.

We note that our lower bound peaks higher compared to the prior best lower bounds. For standard parameters of $u = n^{1+O(1)}$ and $w = \Theta(\lg u)$, our lower bound shows that optimal space constructions with $s = O(n)$ require query time $t = \Omega(\lg u)$ in the cell probe model. In contrast, prior works [4, 21] obtain lower bounds of $t = \Omega(\lg u / \lg\lg u)$.

1.2 Hash Functions with High Independence

When using hash functions in the design of data structures and algorithms, it is often assumed for simplicity of analysis that truly random hash functions are available. Such a hash function $h : [u] \to [m]$ maps each key *independently* to a uniform random value in $[m]$. Or said differently, when drawing the random hash function h, we choose a uniform random function in the family of hash functions $\mathcal{H}$ consisting of all (deterministic) functions from $[u]$ to $[m]$. Implementing such a hash function in practice is often infeasible as it requires $u \lg m$ random bits and thus the storage requirement may completely dominate that of any data structure making use of the hash function.

Fortunately, much weaker hash functions suffice in many applications. The simplest property of a family of hash functions $\mathcal{H} \subseteq [u] \to [m]$, is that it is *universal* [7]. A universal family of hash functions has the property that for a uniform random $h \in \mathcal{H}$, it holds for every pair of keys $x \neq y \in [u]$ that $\Pr[h(x) = h(y)] \leq 1/m$. Universal hashing for instance suffices for implementing hashing with chaining with expected constant time membership queries, but is not sufficient for implementing Cuckoo hashing [9]. The next step up from universal hashing is the notion of n-wise independent hashing. A family of hash functions $\mathcal{H}$ is n-wise independent if, for h drawn uniformly from $\mathcal{H}$, it holds for any set of n distinct keys $x_1, \ldots, x_n$ that $h(x_1), \ldots, h(x_n)$ are independent and uniformly random (or nearly uniformly random). The prototypical example of an n-wise independent family of hash function (with nearly uniform hash values) is

$$\mathcal{H} := \left\{ h_{\alpha_0,\ldots,\alpha_{n-1}}(x) = \left(\sum_{i=0}^{n-1} \alpha_i x^i \bmod p\right) \bmod m \mid \alpha_0, \ldots, \alpha_{n-1} \in [p] \right\}$$

where p is any prime greater than or equal to u. That is, to draw a hash function h from $\mathcal{H}$, we sample $\alpha_0, \ldots, \alpha_{n-1}$ uniformly and independently in $[p]$ and let $h(x)$ be the evaluation of the polynomial $(\sum_i \alpha_i x^i \bmod p) \bmod m$.[1] Clearly, the evaluation time of this hash function is $\Theta(n)$. Whether it is possible to implement n-wise independent hash functions with faster evaluation time has been the focus of much research. On the lower bound side, Siegel [22] proved that any implementation of an n-wise independent hash function $h : [u] \to [m]$ using s memory cells of $w = \Theta(\lg u)$ bits, must probe at least $t = \Omega(\min\{\lg(u/n)/\lg(s/n), n\})$ memory cells to evaluate h. The hash function above matches the second term in the minimum. For the first term, the result that comes closest is a recursive form of tabulation hashing by Christiani et al. [8] that gives an n-wise independent family of hash functions that can be implemented using $s = O(nu^{1/c})$ space and evaluation time $t = O(c \lg c)$ for any $c = O(\lg u/\lg n)$. Rewriting the space bound gives $c = \lg u/\lg(s/n)$ and thus $t = O(\lg(u)\lg(\lg(u)/\lg(s/n))/\lg(s/n))$. This is about a $\lg\lg u$ factor away from the lower bound of Siegel in terms of the query time t. This algorithm is adaptive and requires $s \geq n^{1+\Omega(1)}$ as they need $\lg u/\lg(s/n) = O(\lg u/\lg n)$.

Our Contribution. Designing an optimal n-wise independent family of hash functions thus remains open, with or without adaptivity. In this work, we show how to implement such a function in the cell probe model (where computation is free):

▶ **Theorem 3.** *For any $s = \Omega(n)$ and $p = \Omega(u)$, there is a non-adaptive static cell probe data structure for storing an n-wise independent hash function $h : [u] \to \mathbb{F}_p$ using s memory cells of $w = \Theta(\lg p)$ bits and answering evaluation queries in $t = O(\lg(u/n)/\lg(s/n))$ probes.*

We remark that Siegel's lower bound holds in the cell probe model, and thus our data structure is optimal. Furthermore, Siegel's lower bound holds *also* for adaptive data structures, whereas ours is even non-adaptive. Compared to the work of Christiani et al., we have a faster evaluation time and only require $s = \Omega(n)$. The downside is of course that our solution is only implemented in the cell probe model. Implementing our hash function in the word-RAM model would require the same type of explicit expander graph as for implementing our non-adaptive dictionary (and a bit more), further motivating the study of such expanders (see Section 5).

To compare with previous techniques, we note that the majority of prior works (such as [15, 10, 8]) consider adaptive constructions. The original work of Siegel [22] did not directly study non-adaptivity. However, Lemma 2 in [22] can be used to construct a non-adaptive construction in the cell probe model using a suitable expander graph. Our construction leads to a better (and tight) upper bound in addition to being simpler by replacing polynomials with a simple sum of memory cells.

2 Non-Adaptive Dictionaries

We consider the dictionary problem where we are to preprocess a set X of n key-value pairs from $[u] \times [u]$ into a data structure, such that given an $x \in [u]$, we can quickly return the corresponding value y such that $(x, y) \in X$ or conclude that no such y exists. We assume that any for any key x, there is at most one value y such that $(x, y) \in X$.

[1] Technically, this hash function is only approximately n-wise independent, in the sense that the hash values of any n keys *are* independent, but only approximately uniform random.

We focus on non-adaptive data structures in the cell probe model. Non-adaptive means that the memory cells probed on a query depends only on x. We assume $u = \Omega(n)$ and that the cell size w is $\Theta(\lg u)$.

As mentioned in Section 1, we base our data structure on expander graphs. We recall the standard definitions of bipartite expanders in the following:

▶ **Definition 4.** *A (u, s, t)-bipartite graph with u left vertices, s right vertices and left degree t is specified by a function $\Gamma : [u] \times [t] \to [s]$, where $\Gamma(x, y)$ denotes the y^{th} neighbor of x. For a set $S \subseteq [u]$, we write $\Gamma(S)$ to denote its neighbors $\{\Gamma(x, y) : x \in S, y \in [t]\}$.*

▶ **Definition 5.** *A bipartite graph $\Gamma : [u] \times [t] \to [s]$ is a (K, A)-expander if for every set $S \subseteq [u]$ with $|S| = K$, we have $|\Gamma(S)| \geq A \cdot K$. It is a $(\leq K_{\max}, A)$-expander if it is a (K, A)-expander for every $K \leq K_{\max}$.*

The literature on bipartite expanders, see e.g. [13], is focused on graphs with near-optimal expansion $A = (1 - \varepsilon)t$, i.e. very close to the largest possible expansion with degree t. However, for our non-adaptive dictionaries, we need significantly less expansion. We call such expanders *non-contractive* and define them as follows:

▶ **Definition 6.** *A bipartite graph $\Gamma : [u] \times [t] \to [s]$ is a $(\leq K_{\max})$-non-contractive expander if it is a $(\leq K_{\max}, 1)$-expander.*

Said in words, a bipartite is a $(\leq K_{\max})$-non-contractive expander, if every set of at most $K \leq K_{\max}$ left-nodes has at least K neighbors.

Before presenting our dictionary, we present the second ingredient in our dictionary, namely Hall's marriage theorem. For a bipartite graph with left-vertices X, right-vertices Y and edges E, an X-perfect matching is a subset of disjoint edges from E such that every vertex in X has an edge. Hall's theorem then gives the following:

▶ **Theorem 7** (Hall's Marriage Theorem)**.** *A bipartite graph with left-vertices X and right-vertices Y has an X-perfect matching if and only if for every subset $S \subseteq X$, the set of neighbors $\Gamma(S)$ satisfies $|\Gamma(S)| \geq |S|$.*

With these ingredients, we are ready to present our dictionary.

Dictionary from Non-Contractive Expander. Given a set of n key-value pairs $X = \{(x_i, y_i)\}_{i=1}^n \subset [u] \times [u]$ and a space budget of s memory cells, we build a data structure as follows:

Construction. Initialize s memory cells and let $\Gamma : [u] \times [t] \to [s]$ be a $(\leq n)$-non-contractive expander for some t. Construct the bipartite graph G with a left-vertex for each x_i and a right vertex for each of the s memory cells. Add an edge from x_i to each of the nodes $\Gamma(x_i, j)$ for $i = 0, \ldots, t - 1$. Note that this is a subgraph of the bipartite $(\leq n)$-non-contractive expander corresponding to Γ. It follows that for every subset $S \subseteq \{x_i\}_{i=1}^n$, we have $|\Gamma(S)| \geq |S|$. We now invoke Hall's Marriage Theorem (Theorem 7) to conclude the existence of an $\{x_i\}_{i=1}^n$-perfect matching on G. Let $M = \{(x_i, v_i)\}_{i=1}^n$ denote the edges of the matching. For each such edge (x_i, v_i), we store the key-value pair (x_i, y_i) in the memory cell of address v_i. For all remaining $s - n$ memory cells, we store a special *Nil* value.

Querying. Given a query $x \in [u]$, we query the t memory cells of address $\Gamma(x, i)$ for $i = 0, \ldots, t - 1$. If any of them stores a pair (x, y), we return y. Otherwise, we return *Nil* to indicate that no pair (x, y) exists in X.

Analysis. Correctness follows immediately from Hall's Marriage Theorem. The space usage is s memory cells of $w = \Theta(\lg u)$ bits and the query time is t. The required perfect matching M can be computed in $\text{poly}(n, s)$ times after performing $O(nt)$ queries to obtain the edges of the subgraph induced by the left-vertices $\{x_i\}_{i=1}^n$. We thus have the following result:

▶ **Lemma 8.** *Given a bipartite $(\leq n)$-non-contractive expander $\Gamma : [u] \times [t] \to [s]$, there is a non-adaptive dictionary for storing a set of n key-value pairs using s cells of $w = \Theta(\lg u)$ bits and answering queries in t evaluations of Γ and t memory probes. The dictionary can be constructed in $\text{poly}(n, s)$ time plus $O(nt)$ evaluations of Γ.*

Lemma 8 thus gives us a way of obtaining a non-adaptive dictionary from an expander. What remains is to give expanders with good parameters. As mentioned, we do not have optimal explicit constructions of such expanders. However, for the cell probe model where computation is free of charge, we merely need the existence of Γ and not that it is efficiently computable. Concretely, a probabilistic argument gives the following:

▶ **Lemma 9.** *For any $s \geq 2n$ and any $u \geq n$, there exists a (non-explicit) $(\leq n)$-non-contractive expander $\Gamma : [u] \times [t] \to [s]$ with $t = \lg(u/n)/\lg(s/n) + 5$.*

Combining Lemma 8 and Lemma 9 implies our Theorem 1.

Non-Explicit Expander. In the following, we prove Lemma 9. For this, consider drawing $\Gamma : [u] \times [t] \to [s]$ uniformly among all such functions/expanders. That is, we let $\Gamma(x, y)$ be uniform random and independently chosen in $[s]$ for each $x \in [u]$ and $y \in [t]$. For each $S \subseteq [u]$ with $|S| \leq n$ and each $T \subseteq [s]$ with $|T| = |S| - 1$, define an event $E_{S,T}$ that occurs if $\Gamma(S) \subseteq T$. We have that Γ is a $(\leq n)$-non-contractive expander if none of the events $E_{S,T}$ occur. For a fixed $E_{S,T}$, we have $\Pr[E_{S,T}] = (|T|/s)^{t|S|}$ and thus a union bound implies

$$\begin{aligned}
\Pr[\Gamma \text{ is not a } (\leq n)\text{-non-contractive expander}] &\leq \\
\sum_{S,T} \Pr[E_{S,T}] &= \\
\sum_{i=1}^{n} \sum_{S \subseteq [u]:|S|=i} \sum_{T \subseteq [s]:|T|=i-1} \Pr[E_{S,T}] &\leq \\
\sum_{i=1}^{n} \binom{u}{i}\binom{s}{i}(i/s)^{ti} &\leq \\
\sum_{i=1}^{n} (eu/i)^i (es/i)^i (i/s)^{ti} &= \\
\sum_{i=1}^{n} \left(\frac{e^2 u i^{t-2}}{s^{t-1}}\right)^i &\leq \\
\sum_{i=1}^{n} \left(e^2 (u/n)(n/s)^{t-1}\right)^i.
\end{aligned}$$

For $s \geq 2n$ and $t \geq \lg(u/n)/\lg(s/n) + 5$, this is at most $\sum_{i=1}^{n} (e^2/16)^i < 1$ and thus proves Lemma 9.

3 Hashing

In this section, we show how to construct a n-wise independent hash function with fast evaluation in the cell probe model. As a data structure problem, such a data structure has a query $h(x)$ for each $x \in [u]$. Upon construction, the data structure draws a random seed and initializes s memory cells of w bits. The data structure satisfies that the values $h(x)$ are uniform random in $\mathbb{F}_p$ and n-wise independent. Here the randomness is over the choice of random seed.

Similarly to our dictionary, our hashing data structures makes use of a bipartite expander. However, we need a (very) slightly stronger expansion property. Concretely, we assume the availability of a $(\leq n, 2)$-expander $\Gamma : [u] \times [t] \to [s]$ (rather than a $(\leq n, 1)$-expander). The expander Γ thus satisfies that for any $S \subseteq [u]$ with $|S| \leq n$, we have $|\Gamma(S)| \geq 2|S|$.

In addition to the $(\leq n, 2)$-expander Γ, we also need another function assigning *weights* to the edges of Γ. We say that $\Pi : [u] \times [t] \to \mathbb{F}_p$ makes Γ *useful* if the following holds: Construct from (Γ, Π) the $u \times s$ matrix $A_{\Gamma,\Pi}$ such that entry (x, y) equals

$$\sum_{j:\Gamma(x,j)=y} \Pi(x, j) \bmod p$$

We have that (Γ, Π) is useful if every subset of n rows in $A_{\Gamma,\Pi}$ is a linearly independent set of vector over $\mathbb{F}_p^s$. We show later that for any $(\leq n, 2)$-expander Γ, there exists at least one Π making Γ useful:

▶ **Lemma 10.** *If $\Gamma : [u] \times [t] \to [s]$ is a $(\leq n, 2)$-expander, then for $p \geq 2eu$, there exists a $\Pi : [u] \times [t] \to \mathbb{F}_p$ such that (Γ, Π) is useful.*

In the cell probe model, we may assume that Γ and Π are free to evaluate and are known to a data structure since computation is free of charge. With such a pair (Γ, Π) we may now construct our data structure for n-wise independent hashing.

Construction. Initialize the data structure by filling each of the s memory cells by uniformly and independently chosen values in $\mathbb{F}_p$ (the seed). Let $z_0, \ldots, z_{s-1}$ denote the values in the memory cells.

Querying. To evaluate $h(x)$ for an $x \in [u]$, compute and return the value

$$\sum_{j=0}^{t-1} \Pi(x, j) z_{\Gamma(x,j)} \bmod p.$$

Analysis. Observe that the value returned on the query x equals

$$\sum_{j=0}^{t-1} \Pi(x, j) z_{\Gamma(x,j)} \bmod p \equiv \sum_{y=0}^{s-1} \sum_{j:\Gamma(x,j)=y} \Pi(x, j) z_{\Gamma(x,j)} \bmod p.$$

But this is the same as $(A_{\Gamma,\Pi} z)_x$, i.e. the inner product of the x'th row of $A_{\Gamma,\Pi}$ with the randomly drawn vector z. Since the rows of $A_{\Gamma,\Pi}$ are n-wise independent and z is drawn uniformly, we conclude that the query values $h(0), \ldots, h(u-1)$ are n-wise independent as well. The query time is t probes and the space usage is s cells of $\lg p$ bits. We thus conclude

▶ **Lemma 11.** *Given a bipartite $(\leq n, 2)$ expander $\Gamma : [u] \times [t] \to [s]$ and a $p \geq 2eu$, there is a cell probe data structure for evaluating an n-wise independent hash function $h : [u] \to \mathbb{F}_p$ using s cells of $w = \Theta(\lg p)$ bits and answering queries in t cell probes.*

An argument similar to the proof of Lemma 9, we show the existence of the desired expanders:

▶ **Lemma 12.** *For any $s \geq 2n$ and any $u \geq n$, there exists a (non-explicit) $(\leq n, 2)$ expander $\Gamma : [u] \times [t] \to [s]$ with $t = 2\lg(u/n)/\lg(s/n) + 4$.*

Combining Lemma 12, Lemma 10 and Lemma 11 proves Theorem 3.

What remains is to prove Lemma 10 and Lemma 12. We start with Lemma 10.

Proof of Lemma 10. We give a probabilistic argument. Let $\Gamma : [u] \times [t] \to [s]$ be a $(\leq n, 2)$-expander. Draw $\Pi : [u] \times [t] \to \mathbb{F}_p$ by letting $\Pi(x, j)$ be chosen uniformly and independently from $\mathbb{F}_p$. Define an event E_β for every $\beta \in \mathbb{F}_p^u$ with $1 \leq \|\beta\|_0 \leq n$ ($\|\beta\|_0$ gives the number of non-zeros) that occurs if $\beta A_{\Gamma,\Pi} = 0$. We have that (Γ, Π) is useful if none of the events E_β occur.

Consider one of these events E_β. Since Γ is a $(\leq n, 2)$-expander, we have that the set of rows in $A_{\Gamma,\Pi}$ corresponding to non-zero coefficients of β have at least $2\|\beta\|_0$ distinct columns containing an entry that is chosen uniformly at random and independently from $\mathbb{F}_p$. We thus have $\Pr[E_\beta] \leq p^{-2\|\beta\|_0}$. A union bound finally implies:

$$\begin{aligned}
\Pr[(\Gamma, \Pi) \text{ is not useful}] &\leq \\
\sum_{i=1}^{n} \sum_{\beta \in \mathbb{F}_p^u : \|\beta\|_0 = i} \Pr[E_\beta] &\leq \\
\sum_{i=1}^{n} \binom{u}{i} p^i p^{-2i} &\leq \\
\sum_{i=1}^{n} (eu/(ip))^i.
\end{aligned}$$

For $p \geq 2eu$, this is less than 1, which concludes the proof of Lemma 10. ◀

Lastly, we prove Lemma 12.

Proof of Lemma 12. The proof follows that of Lemma 9 uneventfully. Draw Γ randomly, with each $\Gamma(x, y)$ uniform and independently chosen in $[s]$. Again, we define an event $E_{S,T}$ for each $S \subseteq [u]$ with $|S| \leq n$ and each $T \subseteq [s]$ with $|T| = 2|S| - 1$. The event $E_{S,T}$ occurs if $\Gamma(S) \subseteq T$. We have

$$\begin{aligned}
\Pr[\Gamma \text{ is not an } (\leq n, 2)\text{-expander}] &\leq \\
\sum_{S,T} \Pr[E_{S,T}] &\leq \\
\sum_{i=1}^{n} \binom{u}{i} \binom{s}{2i} ((2i)/s)^{ti} &\leq \\
\sum_{i=1}^{n} (eu/i)^i (s/(2i))^{2i} ((2i)/s)^{ti} &= \\
\sum_{i=1}^{n} \left(\frac{eu(2i)^{t-3}}{s^{t-2}} \right)^i &\leq \\
\sum_{i=1}^{n} \left(e(u/n)(2n/s)^{t-2} \right)^i
\end{aligned}$$

For $s \geq 4n$ and $t \geq 2\lg(u/n)/\lg(s/n) + 4 \geq \lg(u/n)/\lg(s/(2n)) + 4$, this is less than 1, completing the proof of Lemma 12. ◀

4 Lower Bound for Non-Adaptive Dictionaries

In this section, we prove cell probe lower bounds for non-adaptive dictionaries supporting membership queries (is x in the input set X?).

We adapt the "cell-sampling" technique from [18]. Roughly speaking, this proof technique shows that there exists a not-too-large subset of cells $C \subseteq [s]$ such that a large number of queries will only probe cells in C (we say such queries are resolved by C) assuming that the query time of the cell probe data structure is impossibly small. For adaptive and static data structures, it can be observed that the subset of cells C will be different for varying choices of the n input key-value pairs as the probed cells during queries can depend on the memory representation.

For our non-adaptive lower bound, we make the critical observation that the subset of sampled cells C need not depend on the n input key-value pairs. In particular, non-adaptive queries must choose the probed cells without any knowledge of the memory representation. As a result, we are able to separate the adaptive and non-adaptive setting for the dictionary problem and successfully prove a matching lower bound to our constructions as follows:

Proof of Theorem 2. Assume the space usage of a data structure is s cells of w bits each. We assume for the proof that $sw \geq 6n \lg(u/n)$. For smaller space usage, we can always pad with dummy memory cells.

For a query $x \in [u]$, let $p(x) \subseteq [s]$ denote the indices of the memory cells probed on query x.

By averaging, for any q with $t \leq q \leq s$, there is a set of q memory cells $C \subseteq [s]$ such that $u\binom{s-t}{q-t}/\binom{s}{q}$ queries x have $p(x) \subseteq C$. Fix such a set C. Assume for the sake of contradiction that

$$t \leq (1/4) \min\left\{q, \frac{\lg(u/n)}{\lg(sw/(n\lg(u/n)))}\right\}.$$

Then we have

$$u \cdot \frac{\binom{s-t}{q-t}}{\binom{s}{q}} = u \cdot \frac{q(q-1)\cdots(q-t+1)}{s(s-1)\cdots(s-t+1)} \geq u \cdot \left(\frac{(3/4)q}{s}\right)^t.$$

Letting $q = (1/4)n\lg(u/n)/w$, this is at least $u \cdot \left(\frac{(3/16)n\lg(u/n)}{sw}\right)^t \geq u \cdot \left(\frac{n\lg(u/n)}{sw}\right)^{2t} \geq u\sqrt{n/u} = \sqrt{un}$.

Let $U \subseteq [u]$ denote the set of queries x with $p(x) \subseteq C$. Notice that the memory cells in C serve as a membership data structure for the universe U and inputs $X \subseteq U$ of size n. Hence the number of bits in C must be at least $\lg\binom{|U|}{n} \geq (1/2)n\lg(u/n)$. But the cells only have $qw = (1/4)n\lg(u/n)$ bits, a contradiction. We thus conclude:

$$t = \Omega\left(\min\left\{\frac{n\lg(u/n)}{w}, \frac{\lg(u/n)}{\lg(sw/(n\lg(u/n)))}\right\}\right). \qquad \blacktriangleleft$$

5 Conclusion and Open Problems

In this work, we presented optimal non-adaptive cell probe dictionaries and data structures for evaluating n-wise independent hash functions. Our upper bounds rely on the existence of bipartite expanders with quite weak expansion properties, namely $(\leq n, 1)$ and $(\leq n, 2)$-bipartite expanders. If efficient explicit constructions of such expanders were to be developed,

they would immediately allow us to implement our dictionary in the standard word-RAM model. They would also go a long way towards a word-RAM implementation of n-wise independent hashing. We thus view our results as strong motivation for further research into such expanders. In personal communication with Bruno Bauwens and Marius Zimand, they have given a preliminary proof that an exciting explicit construction with $s = O(n)$ and $t = (\lg u)^{O(1)}$ exists, thus taking a first step towards an optimal word-RAM implementation.

Next, we remark that our non-explicit constructions of $(\leq n, 1)$ and $(\leq n, 2)$ expanders are essentially optimal. Concretely, a result of Radhakrishnan and Ta-Shma [20] shows that any (u, s, t)-bipartite graph with expansion 1 requires $t = \Omega(\lg(u/n)/\lg(s/n))$. In more detail, Theorem 1.5 (a) of [20] proves that if G is a (u, s, t)-bipartite graph that is an (n, ϵ) disperser (every set of n left-nodes has at least $(1 - \varepsilon)s$ right-nodes), then for $\varepsilon > 1/2$, the left-degree, t, is $\Omega(\lg(u/n)/\lg(1/(1 - \varepsilon)))$. Since a $(\leq n, 1)$-non-contractive expander is also an (n, ϵ)-disperser with $(1 - \epsilon) = n/s$, the lower bound $t = \Omega(\lg(u/n)/\lg(s/n))$ follows.

Finally, we also observe a near-equivalence between non-adaptive data structures for evaluating n-wise independent hash functions and non-constructive bipartite expanders. Concretely, assume we have a word-RAM data structure for evaluating an n-wise independent hash function from $[u]$ to $[u]$ and assume $w = \lg u$ for simplicity. If the data structure uses s space and answers queries in t time (including memory lookups and computation), then we may obtain an explicit expander from the data structure. Concretely, we form a right node for every memory cell, a left node for every query and an edge corresponding to each cell probed on a query. Now observe that if there was a set of n left nodes S with $|\Gamma(S)| < n$, then from those $|\Gamma(S)|$ memory cells, the data structure has to return n independent and uniform random values in $[u]$. But the cells only have $|\Gamma(S)|w < n \lg u$ bits, i.e. a contradiction. Hence the resulting expander is non-contractive. If the query time of the data structure was t, we may obtain the edges incident to a left node simply by running the corresponding query algorithm. Since the query algorithm runs in t time, it clearly accesses at most t right nodes and computing the nodes to access can also be done in t time. A similar connection was observed by [8].

References

1 Noga Alon and Uriel Feige. On the power of two, three and four probes. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 346–354. SIAM, 2009.

2 Bruno Bauwens and Marius Zimand. Hall-type theorems for fast dynamic matching and applications. *arXiv preprint arXiv:2204.01936*, 2022.

3 Mette Berger, Esben Rune Hansen, Rasmus Pagh, Mihai Pătraşcu, Milan Ruzic, and Peter Tiedemann. Deterministic load balancing and dictionaries in the parallel disk model. In Phillip B. Gibbons and Uzi Vishkin, editors, *SPAA 2006: Proceedings of the 18th Annual ACM Symposium on Parallelism in Algorithms and Architectures, Cambridge, Massachusetts, USA, July 30 - August 2, 2006*, pages 299–307. ACM, 2006. doi:10.1145/1148109.1148160.

4 Joe Boninger, Joshua Brody, and Owen Kephart. Non-adaptive data structure bounds for dynamic predecessor search. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017, December 11-15, 2017, Kanpur, India*, volume 93 of *LIPIcs*, pages 20:1–20:12, 2017.

5 Joshua Brody and Kasper Green Larsen. Adapt or die: Polynomial lower bounds for non-adaptive dynamic data structures. *Theory Comput.*, 11:471–489, 2015. doi:10.4086/toc.2015.v011a019.

6 H. Buhrman, P. B. Miltersen, J. Radhakrishnan, and S. Venkatesh. Are bitvectors optimal? *SIAM Journal on Computing*, 31(6):1723–1744, 2002. doi:10.1137/S0097539702405292.

7 J Lawrence Carter and Mark N Wegman. Universal classes of hash functions. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112, 1977.

8 Tobias Christiani, Rasmus Pagh, and Mikkel Thorup. From independence to expansion and back again. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 813–820. ACM, 2015. doi:10.1145/2746539.2746620.

9 Jeffrey Cohen and Daniel M. Kane. Bounds on the independence required for cuckoo hashing. Manuscript, 2009. URL: https://cseweb.ucsd.edu//~dakane/cuchkoohashing.pdf.

10 Martin Dietzfelbinger and Philipp Woelfel. Almost random graphs with simple hash functions. In *Proceedings of the thirty-fifth Annual ACM Symposium on Theory of Computing*, pages 629–638, 2003.

11 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.

12 Mohit Garg and Jaikumar Radhakrishnan. Set membership with non-adaptive bit probes. In *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

13 Venkatesan Guruswami, Christopher Umans, and Salil P. Vadhan. Unbalanced expanders and randomness extractors from parvaresh-vardy codes. *J. ACM*, 56(4):20:1–20:34, 2009. doi:10.1145/1538902.1538904.

14 Kasper Green Larsen, Rasmus Pagh, Toniann Pitassi, and Or Zamir. Optimal non-adaptive cell probe dictionaries and hashing. *arXiv preprint arXiv:2308.16042*, 2023.

15 Anna Pagh and Rasmus Pagh. Uniform hashing in constant time and linear space. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 622–628, 2003.

16 Rasmus Pagh. On the cell probe complexity of membership and perfect hashing. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 425–432. ACM, 2001. doi:10.1145/380752.380836.

17 Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In Friedhelm Meyer auf der Heide, editor, *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, volume 2161 of *Lecture Notes in Computer Science*, pages 121–133. Springer, 2001. doi:10.1007/3-540-44676-1_10.

18 Rina Panigrahy, Kunal Talwar, and Udi Wieder. Lower bounds on near neighbor search via metric expansion. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 805–814. IEEE, 2010.

19 Giuseppe Persiano and Kevin Yeo. Tight static lower bounds for non-adaptive data structures. *CoRR*, abs/2001.05053, 2020. arXiv:2001.05053.

20 Jaikumar Radhakrishnan and Amnon Ta-Shma. Bounds for dispersers, extractors, and depth-two superconcentrators. *SIAM J. Discret. Math.*, 13(1):2–24, 2000. doi:10.1137/S0895480197329508.

21 Sivaramakrishnan Natarajan Ramamoorthy and Anup Rao. Lower bounds on non-adaptive data structures maintaining sets of numbers, from sunflowers. In *33rd Computational Complexity Conference*, 2018.

22 Alan Siegel. On universal classes of fast high performance hash functions, their time-space tradeoff, and their applications (extended abstract). In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 20–25. IEEE Computer Society, 1989. doi:10.1109/SFCS.1989.63450.

23 Robert Endre Tarjan and Andrew Chi-Chih Yao. Storing a sparse table. *Communications of the ACM*, 22(11):606–611, 1979.

24 Andrew Chi-Chih Yao. Should tables be sorted? *J. ACM*, 28(3):615–628, 1981. doi:10.1145/322261.322274.

25 Kevin Yeo. Cuckoo hashing in cryptography: Optimal parameters, robustness and applications. In *Annual International Cryptology Conference*, pages 197–230. Springer, 2023.

An Improved Quantum Max Cut Approximation via Maximum Matching

Eunou Lee
Korea Institute for Advanced Study, Seoul, South Korea

Ojas Parekh
Sandia National Laboratories, Albuquerque, NM, USA

Abstract

Finding a high (or low) energy state of a given quantum Hamiltonian is a potential area to gain a provable and practical quantum advantage. A line of recent studies focuses on Quantum Max Cut, where one is asked to find a high energy state of a given antiferromagnetic Heisenberg Hamiltonian. In this work, we present a classical approximation algorithm for Quantum Max Cut that achieves an approximation ratio of 0.595, outperforming the previous best algorithms of Lee [10] (0.562, generic input graph) and King [8] (0.582, triangle-free input graph). The algorithm is based on finding the maximum weighted matching of an input graph and outputs a product of at most 2-qubit states, which is simpler than the fully entangled output states of the previous best algorithms.

2012 ACM Subject Classification Theory of computation → Rounding techniques

Keywords and phrases approximation, optimization, local Hamiltonian, rounding, SDP, matching

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.105

Category Track A: Algorithms, Complexity and Games

Funding Eunou Lee was supported by Individual Grant No.CG093801 at Korea Institute for Advanced Study.
This work is supported by a collaboration between the US DOE and other Agencies. Ojas Parekh was supported by the U.S. Department of Energy (DOE), Office of Science, National Quantum Information Science Research Centers, Quantum Systems Accelerator. Additional support is acknowledged from DOE, Office of Science, Office of Advanced Scientific Computing Research, Accelerated Research in Quantum Computing, Fundamental Algorithmic Research in Quantum Computing.
This article has been authored by an employee of National Technology & Engineering Solutions of Sandia, LLC under Contract No. DE-NA0003525 with the U.S. Department of Energy (DOE). The employee owns all right, title and interest in and to the article and is solely responsible for its contents. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this article or allow others to do so, for United States Government purposes. The DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan `https://www.energy.gov/downloads/doe-public-access-plan`.

Acknowledgements Eunou Lee appreciates helpful comments on the write-up by Andrus Giraldo.

1 Introduction

A quantum optimization problem seeks to compute the maximum (or minimum) of a function that is defined over the n-qubit Hilbert space. In a restricted case where the function is a sum of k-qubit Hamiltonians, it is well known that the problems are in general QMA-hard, i.e. hard to solve to an inverse polynomial precision even with a quantum computer [4]. One way to cope with the computational hardness is to try to find good approximate solutions.

Quantum Max Cut (QMC) has served as a benchmark problem to develop ideas for quantum Hamiltonian approximation. It has a simple definition, has a good physical motivation, namely the antiferromagnetic Heisenberg model, and extends the well-studied classical Max Cut problem. The task of QMC, given a positive weighted graph $G = (V, E, w)$, is to find a (description of a) maximum energy state for the Hamiltonian

$$H = \sum_{(i,j)\in E} w_{ij}(I - X_iX_j - Y_iY_j - Z_iZ_j)/4,$$

where X_i is the Pauli matrix X on qubit i and identity on the rest. The matrices, Y_i, Z_i are similarly defined.

Most of the existing approximation algorithms for QMC follow the framework of the seminal Goemans-Williamson algorithm [6]. The problem is first relaxed to a semidefinite program (SDP), then the SDP is solved in polynomial time, and finally, the SDP solution is rounded to a feasible solution of the original problem. Since QMC is a maximization problem over the n-qubit Hilbert space, the rounded solution should be an n-qubit quantum state instead of an n-bit string as for Max Cut. Assuming that we follow the Goemans-Williamson framework to approximate QMC, there are still three design choices: 1) which SDP to relax the original problem to, 2) which subset of quantum states (ansatz) to round the SDP solution to, and 3) how to round the SDP solution to an ansatz state. For Max Cut, optimal choices (up to the Unique Games Conjecture) of classical analogues of the above are known; however, these remain unsettled for QMC.

The Quantum Lasserre SDP hierarchy [9, 11] is a sequence of SDPs that upper bounds the maximum energy of a given quantum Hamiltonian. The Quantum Lasserre hierarchy does so by optimizing over pseudo-density operators that are not guaranteed to be positive. The level-k Lasserre SDP includes all valid linear constraints on moments of subsets of at most k qubits. It also includes global constraints characterized by polynomials where each term is tensor product of at most k non-trivial single-qubit Paulis (see e.g., [13]). Hence the SDPs in the hierarchy become tighter as the level increases, eventually converging to the given quantum Hamiltonian problem when $k = n$. The following is a way to view the SDP construction. Fix a quantum state $|\phi\rangle$. For an n-qubit Pauli matrix P, define $v(P) := P|\phi\rangle$. Then the energy of $|\phi\rangle$ for an arbitrary Hamiltonian can be expressed as a sum of inner products of these vectors. For example, $\langle\phi|I - X_iX_j - Y_iY_j - Z_iZ_j|\phi\rangle = \langle v(I), v(I)\rangle - \langle v(X_iX_j), v(I)\rangle - \langle v(Y_iY_j), v(I)\rangle - \langle v(Z_iZ_j), v(I)\rangle$. Additionally, it holds that $\|v(P)\| = 1$, and $\langle v(P_1), v(Q_1)\rangle = \langle v(P_2), v(Q_2)\rangle$ for all n-qubit Pauli matrices P_1, P_2, Q_1, Q_2 such that $P_1Q_1 = P_2Q_2$. Now forget about $|\phi\rangle$ and maximize the energy expression in terms of $v(P)$'s, while satisfying the inner product relations.

In the following definition, $\mathcal{P}_k(n)$ is the set of n-qubit Pauli matrices with non-trivial terms on up to k qubits.

▶ **Definition 1** (Level-k Quantum Lasserre SDP)**.**

$$\textit{Maximize} \sum_{(i,j)\in E} w_{ij} v(I)\cdot(v(I) - v(X_iX_j) - v(Y_iY_j) - v(Z_iZ_j))/4 \tag{S}$$

$$\begin{aligned}
\textit{subject to } & v(P) \in \mathbb{R}^{|\mathcal{P}_k(n)|}, && \forall P \in \mathcal{P}_k(n),\\
& v(P)\cdot v(P) = 1, && \forall P \in \mathcal{P}_k(n),\\
& v(P_1)\cdot v(Q_1) = v(P_2)\cdot v(Q_2), && \forall P_1, P_2, Q_1, Q_2 \in \mathcal{P}_k(n) \text{ s.t. } P_1Q_1 = P_2Q_2,\\
& v(P)\cdot v(Q) = 0, && \forall P, Q \in \mathcal{P}_k(n) \text{ s.t. } PQ + QP = 0.
\end{aligned}$$

Every existing QMC approximation algorithm that follows an SDP rounding framework uses a Quantum Lasserre SDP. The QMC approximation algorithm of Gharibian-Parekh [5] employs the level-1 Lasserre, and Parekh-Thompson [12], Parekh-Thompson [13], Lee [10], and King [8] employ the level-2 Lasserre SDP. More sophisticated SDP hierarchies that are aware of the $SU(2)$ symmetry in the QMC Hamiltonian have also been developed recently [14, 15], and such hierarchies are implicitly used in existing QMC approximation algorithms [13].

Once an SDP relaxation is solved, its solution is rounded to a quantum state. Current algorithms round SDP solutions to a proper subset (ansatz) of the n-qubit Hilbert space. Gharibian-Parekh [5] and Parekh-Thompson [13] round to product states, and Parekh-Thompson [12] rounds to a product of 1- and 2-qubit states, inspired by the non-SDP approximation algorithm of Anshu, Gosset, and Morenz [1]. Lee [10] and King [8] round to n-qubit entangled states.

In this paper, we introduce a simple classical approximation algorithm that solves the level-2 Lasserre SDP and rounds to either a product of 1-qubit states or a product of 1- and 2-qubit states. The former is obtained using the Gharibian-Parekh product state rounding algorithm. The latter is obtained by solving the Maximum Weight Matching problem in the weighted input graph on which the QMC Hamiltonian is defined, and this does not depend on the SDP solution at all. This distinguishes and drastically simplifies our algorithm relative to previous SDP rounding approaches, which crucially use information from a level-2 SDP solution to produce entangled states.

We show that the approximation ratio of our algorithm is 0.595, which improves the previous best algorithms for general graphs (Lee [10] with a ratio of 0.562), and triangle-free graphs (King [8] with a ratio of 0.582).

Quantum optimization

Another issue that we are concerned with is the definition of quantum optimization problems. A common way to define a quantum optimization problem in the literature is to define an energy function on n qubits and then seek a maximum-energy state with respect to the function. What does it mean for a classical algorithm to solve this problem? If we restrict classical algorithms to outputting basis states, then the above quantum optimization reduces to a classical one. A more relaxed and common approach is to only ask for a "description" of a quantum state. This definition accommodates a broader family of algorithms that output a description of an entangled state, such as our and other previous algorithms for approximating QMC. Now the issue is that the meaning of the word "description" is vague: an output state of any quantum algorithm has a classical description, namely the quantum algorithm itself written on paper. Therefore any quantum algorithm is a classical algorithm if we accept this definition. We propose the following definition for a more satisfying notion of a classical algorithm solving a quantum optimization problem.

▶ **Definition 2.** *Given an objective function f that maps an n-qubit state to a real number, a pair of polynomial time quantum or classical algorithms (P, V) maximizes f to a value ν if the following conditions hold:*

1. **a.** *If max $f \geq \nu$: $\exists\ |w\rangle$ of size polynomial in n such that $V(|w\rangle) = 1$ w.p. $\geq 2/3$,*
   b. *If max $f \leq \nu - 1/p(n)$ for some polynomial p: $\forall$ polynomial-sized $|w\rangle$, $V(|w\rangle) = 1$ w.p. $\leq 1/3$*
2. *P outputs $|w'\rangle$ such that $V(|w'\rangle) = 1$ w.p. $\geq 2/3$.*

Each of P and V can be either classical or quantum; when P or V is classical, it is assumed that classical states are employed. Therefore according to our definition, we can have a cc-, qc-, cq-, or qq-optimization algorithm for a quantum optimization problem depending on whether each of P and V is classical or quantum. Only cc-optimization algorithms for QMC are known to us so far.

We argue that finding an optimal qq-optimization algorithm for QMC is a viable path to achieve a *provable and practical quantum advantage*. Suppose the verifier algorithm V is fixed to be classical. Then, from the prover P's side, the task is to find a bit string to convince V that there is a high energy state. Assuming there is a tight NP-hard upper bound for a classical approximation for QMC (for example by the PCP theorem), we cannot find a quantum prover that provably gives a greater ratio than all classical provers unless we prove that NP is a proper subset of QMA. However, to the best of our knowledge, there are no unexpected complexity theoretic consequences of a qq-approximation having a greater ratio than all cc-approximations. Moreover, we already know how to upper bound classical approximation ratios in some cases via the PCP theorem and the Unique Games Conjecture, so we can hope to upper bound the classical ratio for QMC. The current best upper bound for a cc-approximation of QMC is 0.956 up to plausible conjectures [7].

Approximation algorithm

In all previous algorithms outputting entangled states, the following monogamy of entanglement relation on stars is used crucially.

▶ **Definition 3** (SDP solution values). *Let $G = (V = [n], E, w)$ be a weighted graph and let $(v(P))_{P\in\mathcal{P}_2(n)}$ be a feasible solution to the level-2 Lasserre SDP. Define, for $i, j \in V$,*

$$g_{ij} := \frac{1}{4}v(I)\cdot(v(I) - v(X_iX_j) - v(Y_iY_j) - v(Z_iZ_j))$$
$$h_{ij} := \frac{1}{4}v(I)\cdot(v(I) - v(X_iX_j) - v(Y_iY_j) - v(Z_iZ_j)) - \frac{1}{2}.$$

For $x \in \mathbb{R}$, denote $x^+ := max(x, 0)$. In particular for $i, j \in V$,

$$h_{ij}^+ := \max(h_{ij}, 0).$$

The objective function value of the SDP solution is then $\nu := \sum_{(i,j)\in E} w_{ij}g_{ij}$.

▶ **Lemma 4** (Monogamy of entanglement on a star, [12]). *Given a feasible solution to the level-2 Lasserre SDP on a graph $G = (V, E)$, for any vertex $i \in V$ and any $S \subseteq V$,*

$$\sum_{j\in S} h_{ij} \leq \frac{1}{2}.$$

In particular,

$$\sum_{j\in N(i)} h_{ij}^+ \leq \frac{1}{2}, \tag{1}$$

where $N(i) = \{j|(i, j) \in E\}$.

The last statement is obtained by taking the set of edges incident to i with positive h_{ij} values as S.

A matching in G corresponds naturally to a state that earns maximal energy on the Hamiltonian terms corresponding to matched edges. Notice that by Equation (1), $(2h_e^+)_{e\in E}$ forms a fractional matching in the sense that if all these values were either 0 or 1, then we would have a matching that would in turn yield a state. We can round this fractional matching to a true matching, with a loss in objective value. Since a maximum weight matching has weight at least that of any matching we might round to, we may simply use it instead. We solve the Maximum Weight Matching problem and assign the best 2-qubit state, namely the singlet, $(|01\rangle - |10\rangle)/\sqrt{2}$, to matched edges. To each unmatched qubit, we assign the maximally mixed 1-qubit state. The resulting n-qubit state has maximal energy on matched edges but low energy on edges that are not matched. To address this issue, we also run the product state algorithm of Gharibian-Parekh [5]. Below is the complete algorithm.

▶ **Algorithm 5** (Approximation algorithm for Quantum Max Cut). *Given a weighted graph $G = (V, E, w)$ as an input,*

1. *Find a product state as follows:*
   a. *Solve SDP* (S) *for $k = 2$ to get solution vectors $(v(P))_{P\in\mathcal{P}_2(n)}$.*
   b. *Sample a random matrix R with dimension $3\times 3|\mathcal{P}_2(n)|$ with each element independently drawn from $\mathcal{N}(0, 1)$.*
   c. *Perform Gharibian-Parekh rounding on each $v_i := (v(X_i)\|v(Y_i)\|v(Z_i))/\sqrt{3}$ to get $u_i := Rv_i/\|Rv_i\|$.*
   d. *Let $\rho_1 := \prod_{i\in V} \frac{1}{2}(I + u_{i,1}X_i + u_{i,2}Y_i + u_{i,3}Z_i)$.*
2. *Find a matching state as follows:*
   a. *Find the maximum weight matching $M : E \to \{0, 1\}$ of G, for example via Edmonds Algorithm [3].*
   b. *Let $\rho_2 := \prod_{(i,j):M_{(i,j)}=1}(I - X_iX_j - Y_iY_j - Z_iZ_j)/4 \prod_{i\in U} I/2$, where U is the set of qubits unmatched by M.*
3. *Output whichever of ρ_1 and ρ_2 that has greater energy.*

Using matchings to find a good QMC state is not a new idea. Anshu, Gosset, and Morenz [1] introduced the idea of using matchings for QMC approximations, proving that there exists a product of 1- and 2-qubit states with energy at least 0.55 times the maximum QMC energy. Parekh and Thompson use a level-2 Lasserre solution to identify [12, 13] a subgraph on which they find a maximum weight matching; they then output a product state or a product of 1- and 2-qubit states akin to our algorithm, yielding an approximation ratio of 0.533. To obtain our improvement, we relate a level-2 SDP solution to the value of a matching on the whole input graph, whereas Parekh-Thompson do so for a proper subgraph of the input graph that is obtained from the level-2 SDP solution.

Our algorithm is much simpler than previous algorithms producing entangled states. It may be surprising that we can establish that this algorithm offers a better approximation guarantee than previous algorithms, including those outputting states with potential global entanglement. In particular, our algorithm does not even need to solve an SDP to obtain the entangled solution ρ_2. We only use the level-2 Lasserre SDP to argue that a maximum weight matching provides a solution that has reasonably high energy when ρ_1 has low energy. This manifests itself when obtaining the product state ρ_1, which must be done so with respect to the level-2 SDP. Even though ρ_1 requires solving the level-2 SDP, it is obtained by only using the vectors, $v(X_i), v(Y_i), v(Z_i)$, corresponding to single-qubit Paulis (i.e., the level-1 part of a level-2 solution). In fact, our algorithm is well defined if we solve the level-1 SDP relaxation instead of the level-2 SDP in Step (1a); however, we do not expect approximation factors beyond 0.498 using only the level-1 SDP [7].

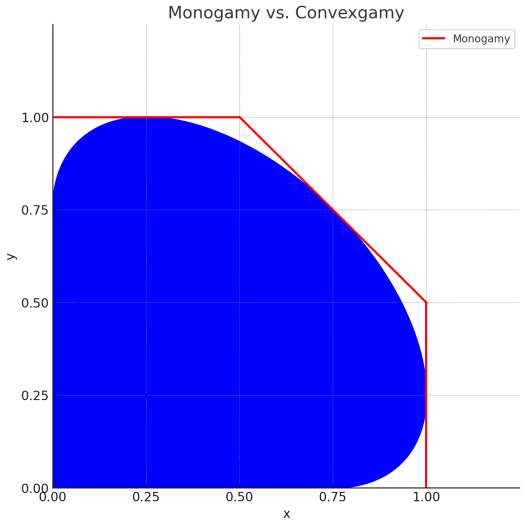


Figure 1 The solid region represents a feasible area characterized by entanglement convexgamy whereas the red line represents the boundary of the feasible region by monogamy of entanglement on a star.

Strengthening monogamy of entanglement on a star

All previous approximation algorithms for QMC outputting entangled states critically rely on monogamy of entanglement on a star (Lemma 4). The previously best-known algorithms by Lee [10] and King [8] both start with a good product state and evolve it to an entangled state while respecting these monogamy of entanglement constraints at each vertex. The key difference in our case is that we directly find a global solution satisfying the monogamy of entanglement constraints. If our algorithm is the optimal way to exploit these constraints, stronger inequalities obtained from the SDP are necessary to deliver a QMC approximation algorithm with a meaningful improvement in approximation ratio.

Even though Monogamy of Entanglement is tight when $h_{ij} = 1/|N(i)|$ for all $j \in N(i)$, it is easy to see that the inequality is far from tight at other points. Suppose $h_{ij} = 1/2$, with (i, j) being maximally entangled. Then on a neighbouring edge (i, k), the energy is $1/4$ and $h_{ik} = -1/4$. In this case, the deviation from the upper bound grows linearly as the number of connected edges grows. Parekh and Thompson derived nonlinear monogamy of entanglement inequalities on a triangle to address this issue in obtaining an optimal approximation for QMC using product states. Their result is captured in Lemma 6.

▶ **Lemma 6** (Monogamy of entanglement on a triangle, Lemma 1 of [13]). *Given a feasible solution to the level-2 Lasserre SDP on a graph $G = (V, E)$, for $i, j, k \in V$,*

$$0 \leq g_{ij} + g_{jk} + g_{ik} \leq \frac{3}{2} \tag{2}$$

$$g_{ij}^2 + g_{jk}^2 + g_{ik}^2 \leq 2g_{ij}g_{jk} + 2g_{ij}g_{ik} + 2g_{jk}g_{ik}. \tag{3}$$

The presentation of the above lemma is equivalent to that of [13] after a change of variables. From these relations, we obtain a tighter bound on star graphs with 2 edges as stated in the lemma below. We denote the relation "convexgamy" to distinguish it from the linear monogamy relation (Lemma 4), and the resulting relation gives a convex region.

▶ **Lemma 7** (Entanglement convexgamy on 2 edges). *Consider a graph $G = (V = \{1, 2, 3\}, E = \{(1, 2), (2, 3)\})$. Let x, y be defined by a feasible level-2 Lasserre SDP solution as $x := g_{12}$ and $y := g_{23}$. Then (x, y) is confined in the region defined by the x-axis, y-axis, and the ellipse touching the x- and y-axis and $x + y = 3/2$ as depicted in Figure 1. More specifically, the ellipse is defined as $3(x + y - 1)^2 + (x - y)^2 = 3/4$.*

Proof. Let $z = g_{13}$. Then $x, y, z \geq 0$. By Lemma 6, $x^2 + y^2 + z^2 \leq 2(xy + yz + zy)$, and $0 \leq x + y + z \leq 3/2$. The first inequality is equivalent to

$$\frac{\sqrt{x^2 + y^2 + z^2}}{x + y + z} \leq \frac{1}{\sqrt{2}}.$$

It means that if $x + y + z = c \geq 0$, then $\sqrt{x^2 + y^2 + z^2} \leq c/\sqrt{2}$. So (x, y, z) lies the intersection of the plane $x + y + z = c$ and the sphere of radius $c/\sqrt{2}$ centered at the origin, which is the incircle of the triangle defined by $x + y + z \geq c$ in the region $x, y, z \geq 0$. When the circle is projected to the xy-plane to give a feasible subset of (x, y), we get the ellipse inscribed in the triangle defined by $(0, 0), (c, 0), (0, c)$. Because $0 \leq c \leq 3/2$, we prove that a feasible point is in the region defined by the x-axis, y-axis, and the ellipse touching the x- and y-axis and $x + y = 3/2$.

The equation of the ellipse follows by solving the inscription condition. ◀

2 Analysis of the algorithm

In the rest of the paper, we introduce necessary concepts regarding matching theory and bound the approximation ratio of our algorithm.

▶ **Theorem 8** (Linear program for Maximum Weight Matching, [3]). *Given a weighted graph $G = (V, E, w)$, the following linear program gives the value of a maximum weight matching in G:*

$$\begin{aligned}
&\textit{maximize} && \sum_{e \in E} w_e x_e && && \text{(M)}\\
&\textit{subject to} && \sum_{j \in N(i)} x_{ij} \leq 1, && \textit{for all } i \in V, && (4)\\
& && \sum_{e \in E(S)} x_e \leq \frac{|S| - 1}{2}, && \textit{for all } S \subseteq V : |S| \textit{ odd}, && (5)\\
& && x_e \geq 0, && \textit{for all } e \in E. && (6)
\end{aligned}$$

where $E(S) := \{(i, j) \in E \mid i, j \in S\}$ for all $S \subseteq V$.

The above linear program (LP) cannot be efficiently solved directly since it has an exponential number of constraints; however, algorithms for Maximum Weight Matching, such as Edmonds Algorithm [3], obtain the optimal value in polynomial time using insights based on the LP and its dual. We will need to show that if we are given a solution $(x)_{e \in E}$ that only satisfies *some* of the constraints, then $(\alpha x)_{e \in E}$ satisfies *all* of the constraints for some $\alpha \in (0, 1)$. This will allow us to relate the objective value of the level-2 SDP to the weight of an optimal matching.

▶ **Lemma 9.** *If $(x)_{e \in E}$ satisfies constraints (4), (6), and (5) for $|S| = 3$, then $(\frac{4}{5}x)_{e \in E}$ is feasible for LP (M).*

Proof. Assume the hypothesis and consider the constraints of (5). We bound the value of the solution x on each such constraint. For $S \subseteq V$, define $\delta(S) := \{(i,j) \in E \mid |\{i,j\} \cap S| = 1\}$. Then,

$$2\sum_{e\in E(S)} x_e \leq 2\sum_{e\in E(S)} x_e + \sum_{e\in\delta(S)} x_e = \sum_{i\in S}\sum_{j\in N(i)} x_{ij} \leq |S|,$$

where the first inequality follows from (6) and the second from (4). So we have

$$\sum_{e\in E(S)} x_e \leq \frac{|S|}{2}, \text{for all } S \subseteq V;$$

however, to satisfy (5), we need a RHS of $\frac{|S|-1}{2}$ instead of $\frac{|S|}{2}$ for sets S of odd size. Since the $|S| = 3$ case is satisfied by assumption, $(\alpha x)_{e\in E}$ is feasible for LP (M), where

$$\alpha := \min_{\{s\in\mathbb{Z} | s \text{ odd}, s>3\}} \frac{s-1}{s} = \frac{4}{5}.$$ ◀

In order to appeal to the above lemma, we will need to show that the energy values arising from the level-2 SDP satisfy the constraints in the hypothesis of the lemma. For this we will rely on monogamy of entanglement on a star and triangle as established in Lemma 4 and the following corollary of Lemma 6, respectively.

▶ **Corollary 10.** *Given a feasible solution to the level-2 Lasserre SDP on a graph $G = (V, E)$, for $i, j, k \in V$,*

$$h^+_{ij} + h^+_{jk} + h^+_{ik} \leq \frac{1}{2}. \tag{7}$$

Proof. Let t be the number of $(u, v) \in \{(i, j), (j, k), (i, k)\}$ with $h^+_{uv} > 0$. If $t \leq 1$ then (7) holds since $h^+_{ij} \leq \frac{1}{2}$ for all $i, j \in V$. If $t \geq 2$ then (7) holds by (2). ◀

We are now in position to prove our main result.

▶ **Theorem 11** (main). *Algorithm 5 gives a 0.595-approximation for any weighted input graph $G = (V, E, w)$.*

Proof. Define

$$H_{ij} = (I - X_iX_j - Y_iY_j - Z_iZ_j)/4,$$
$$H = \sum_{(i,j)\in E} w_{ij}H_{ij}.$$

We bound the expected energy of each case of Steps 1 and 2 of the algorithm. The subroutine of Step 1 is directly from the main algorithm of [5], where in turn the analysis is borrowed from [2]. More precisely, the energy of ρ_1 with respect to H_{ij} is

$$\begin{aligned}\mathrm{Tr}\rho_1 H_{ij} &= \frac{1}{16}\mathrm{Tr}[(I - X\otimes X - Y\otimes Y - Z\otimes Z)\\ &\quad ((I + u_{i,1}X + u_{i,3}Y + u_{i,3}Z)\otimes(I + u_{j,1}X + u_{j,3}Y + u_{j,3}Z))]\\ &= \frac{1}{4}(1 - u_i\cdot u_j),\end{aligned}$$

and its expected value is

$$\mathrm{Tr}\rho_1 H_{ij} = \frac{1 - f_3(v_i\cdot v_j)}{4} \tag{8}$$

where

$$f_3(x) = \frac{2}{3}\left(\frac{\Gamma(2)}{\Gamma(1.5)}\right)^2 x_2F_1\left(\begin{matrix}1/2,1/2\\2\end{matrix}; x^2\right).$$

The estimation is from Lemma 2.1 of [2].

Now we turn to the analysis of the energy of ρ_2 from Step 2. Note that if an edge e is matched, then $\mathrm{Tr}\rho_2 H_e = 1$, and if e is not matched, then $\mathrm{Tr}\rho_2 H_e = 1/4$. More succinctly, $\mathrm{Tr}\rho_2 H_e = 1/4 + 3M_e/4$. Since the SDP values, $(h_e^+)_{e\in E}$ satisfy monogamy of entanglement on a star and triangle (Equations (1) and (7)), we have:

$$\sum_{j\in N(i)} 2h_{ij}^+ \le 1, \text{for all } i \in V,$$
$$2h_{ij}^+ + 2h_{jk}^+ + 2h_{ik}^+ \le 1, \text{for all } i,j,k \in V,$$

where the former correspond to the constraints (4) of the LP (M), and the latter correspond to the constraints (5) with $|S| = 3$. Then by Lemma 9, $(\frac{8}{5}h_e^+)_{e\in E}$ is feasible for the LP. This implies that the optimal solution of the LP, namely a maximum weight matching, has weight at least that of $(\frac{8}{5}h_e^+)_{e\in E}$:

$$\sum_{e\in E} w_e M_e \ge \frac{8}{5}\sum_{e\in E} w_e h_e^+.$$

Therefore,

$$\sum_{e\in E} w_e \mathrm{Tr}\rho_2 H_e = \sum_{e\in E} w_e\left(\frac{1}{4} + \frac{3}{4}M_e\right) \ge \sum_{e\in E} w_e\left(\frac{1}{4} + \frac{6}{5}h_e^+\right). \tag{9}$$

By definition, $v(I)\cdot(v(I) - v(X_iX_j) - v(Y_iY_j) - v(Z_iZ_j))/4 = (1 - 3v_i\cdot v_j)/4 = 1/2 + h_{ij}$. So we have

$$h_{ij} = -\frac{1 + 3v_i\cdot v_j}{4}. \tag{10}$$

Let σ be the density matrix of the output state of the algorithm. By combining the energy estimation of the two cases (8) and (9), we get,

$$\begin{aligned}\sum_{(i,j)\in E} w_{ij}\mathrm{Tr}\sigma H_{ij} &= \max\left\{\sum_{(i,j)\in E} w_{ij}\rho_1 H_{ij}, \sum_{(i,j)\in E} w_{ij}\rho_2 H_{ij}\right\}\\ &\ge \sum_{(i,j)\in E} w_{ij}\left[p\frac{1 - f_3(v_i\cdot v_j)}{4} + (1-p)\left(\frac{1}{4} + \frac{6}{5}\left(-\frac{1 + 3v_i\cdot v_j}{4}\right)^+\right)\right],\end{aligned}$$

for any $p \in [0,1]$. Since $-1 \le v_i\cdot v_j \le 1/3$, it suffices to find

$$\begin{aligned}\alpha &:= \max_{p\in[0,1]}\min_{x\in[-1,1/3]}\left[p\frac{1 - f_3(x)}{4} + (1-p)\left(\frac{1}{4} + \frac{6}{5}\left(-\frac{1+3x}{4}\right)^+\right)\right] \Big/ \frac{1-3x}{4}\\ &= 0.595,\end{aligned}$$

where the maximum occurs at $p = 0.674$. This proves the theorem since

$$\begin{aligned}
&\sum_{(i,j)\in E} w_{ij} \mathrm{Tr}\sigma H_{ij} \\
&\geq \sum_{(i,j)\in E} w_{ij} \left[p_0 \frac{1 - f_3(v_i \cdot v_j)}{4} + (1-p_0)\left(\frac{1}{4} + \frac{6}{5}\left(-\frac{1+3v_i\cdot v_j}{4}\right)^+\right)\right] \\
&\geq 0.595 \sum_{(i,j)\in E} w_{ij} \frac{1 - 3v_i \cdot v_j}{4} \geq 0.595 \lambda_{max}(H).
\end{aligned}$$

◀

▶ Remark 12. The present approach can be likely be improved by deriving analogues of Corollary 10 for larger odd-sized sets of qubits. This would enable a stronger version of Lemma 9 with $\alpha > \frac{4}{5}$. However, the best approximation ratio achievable by such approaches, corresponding to $\alpha = 1$, is 0.606. This is also the approximation ratio of our algorithm on bipartite graphs, since in this case LP (M) gives the value of a maximum weight matching even when constraints (5) are absent from the LP.

3 Open problems

Understanding the approximability of QMC is likely to bring a deeper understanding of the more general local Hamiltonian problem, just as resolving the approximability of Max Cut (up to the Unique Games Conjecture) has had surprising consequences for the theory of classical constraint satisfaction problems. We list relevant research directions below.

- Find a rigorous quantum approximation algorithm for QMC.
- Find a heuristic quantum algorithm (e.g. VQE-based) for QMC that outperforms rigorous classical algorithms.
- The approximability of QMC using product states (i.e. tensor products of 1-qubit states) is well understood [13, 7]. Find the best approximation ratio achievable using a tensor product of 1- and 2-qubit states. Can this be obtained using matchings? Which level of the lassere hierarchy is necessary to achieve this?
- Find an entanglement convexgamy relation (i.e. tighter non-linear inequalities on star graphs) from a valid level-k SDP solution on d edges. Does the optimal such relation (i.e. one precisely describing the feasible space) arise at a constant level k (with respect to d)?

References

1 Anurag Anshu, David Gosset, and Karen Morenz. Beyond Product State Approximations for a Quantum Analogue of Max Cut. In Steven T. Flammia, editor, *15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020)*, volume 158 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.TQC.2020.7.

2 Jop Briët, Fernando Mário de Oliveira Filho, and Frank Vallentin. Grothendieck inequalities for semidefinite programs with rank constraint. *Theory of Computing*, 10(4):77–105, 2014. doi:10.4086/toc.2014.v010a004.

3 Jack Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics*, page 125, 1965. URL: https://api.semanticscholar.org/CorpusID:15379135.

4 Sevag Gharibian, Yichen Huang, Zeph Landau, and Seung Woo Shin. Quantum hamiltonian complexity. *Foundations and Trends® in Theoretical Computer Science*, 10(3):159–282, 2015. doi:10.1561/0400000066.

5 Sevag Gharibian and Ojas Parekh. Almost Optimal Classical Approximation Algorithms for a Quantum Generalization of Max-Cut. In Dimitris Achlioptas and László A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*, volume 145 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:17, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.31.

6 Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, November 1995. doi:10.1145/227683.227684.

7 Yeongwoo Hwang, Joe Neeman, Ojas Parekh, Kevin Thompson, and John Wright. Unique games hardness of quantum max-cut, and a vector-valued borell's inequality, 2021. doi:10.48550/arXiv.2111.01254.

8 Robbie King. An improved approximation algorithm for quantum max-cut on triangle-free graphs. *Quantum*, 7:1180, November 2023. doi:10.22331/q-2023-11-09-1180.

9 Jean B. Lasserre. An explicit equivalent positive semidefinite program for nonlinear 0-1 programs. *SIAM J. on Optimization*, 12(3):756–769, March 2002. doi:10.1137/S1052623400380079.

10 Eunou Lee. Optimizing quantum circuit parameters via SDP. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation, ISAAC 2022, December 19-21, 2022, Seoul, Korea*, volume 248 of *LIPIcs*, pages 48:1–48:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ISAAC.2022.48.

11 Miguel Navascues, Stefano Pironio, and Antonio Acín. Convergent hierarchy of semidefinite programs characterizing the set of quantum correlations. *New Journal of Physics*, 10, July 2008. doi:10.1088/1367-2630/10/7/073013.

12 Ojas Parekh and Kevin Thompson. Application of the Level-2 Quantum Lasserre Hierarchy in Quantum Approximation Algorithms. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 102:1–102:20, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2021.102.

13 Ojas Parekh and Kevin Thompson. An optimal product-state approximation for 2-local quantum hamiltonians with positive terms. *CoRR*, abs/2206.08342, 2022. doi:10.48550/arXiv.2206.08342.

14 Jun Takahashi, Chaithanya Rayudu, Cunlu Zhou, Robbie King, Kevin Thompson, and Ojas Parekh. An su(2)-symmetric semidefinite programming hierarchy for quantum max cut, 2023. arXiv:2307.15688.

15 Adam Bene Watts, Anirban Chowdhury, Aidan Epperly, J. William Helton, and Igor Klep. Relaxations and exact solutions to quantum max cut via the algebraic structure of swap operators, 2023. arXiv:2307.15661.

Polylogarithmic Approximations for Robust s-t Path

Shi Li[1]
Department of Computer Science and Technology, Nanjing University, Jiangsu, China

Chenyang Xu[1]
Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China

Ruilong Zhang[1]
Department of Computer Science, City University of Hong Kong, Hong Kong, China

Abstract

The paper revisits the Robust s-t Path problem, one of the most fundamental problems in robust optimization. In the problem, we are given a directed graph with n vertices and k distinct cost functions (scenarios) defined over edges, and aim to choose an s-t path such that the total cost of the path is always provable no matter which scenario is realized. Viewing each cost function as an agent, our goal is to find a fair s-t path, which minimizes the maximum cost among all agents. The problem is NP-hard to approximate within a factor of $o(\log k)$ unless NP $\subseteq$ DTIME($n^{\text{poly}\log n}$), and the best-known approximation ratio is $\widetilde{O}(\sqrt{n})$, which is based on the natural flow linear program. A longstanding open question is whether we can achieve a polylogarithmic approximation for the problem; it remains open even if a quasi-polynomial running time is allowed.

Our main result is a $O(\log n \log k)$ approximation for the Robust s-t Path problem in quasi-polynomial time, solving the open question in the quasi-polynomial time regime. The algorithm is built on a novel linear program formulation for a decision-tree-type structure, which enables us to overcome the $\Omega(\sqrt{n})$ integrality gap for the natural flow LP. Furthermore, we show that for graphs with bounded treewidth, the quasi-polynomial running time can be improved to a polynomial. We hope our techniques can offer new insights into this problem and other related problems in robust optimization.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Approximation Algorithm, Randomized LP Rounding, Robust s-t Path

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.106

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2305.16439` [25]

Funding Chenyang Xu is supported by the National Key Research Project of China under Grant No. 2023YFA1009402, the National Natural Science Foundation of China (No. 62302166), the Dean's Fund of Shanghai Key Laboratory of Trustworthy Computing, ECNU, and the Key Laboratory of Interdisciplinary Research of Computation and Economics (SUFE), Ministry of Education. Shi Li was supported by the State Key Laboratory for Novel Software Technology, and the New Cornerstone Science Laboratory.

1 Introduction

Robust optimization under uncertainty [5, 6, 17, 26] is one of the most important and challenging computational tasks in the real world. Uncertainty arises in many scenarios. For instance, the travel time for a road segment might be uncertain due to traffic jams. The paper revisits the Robust s-t Path problem [18], a cornerstone problem in the area of robust optimization. In the problem, there are several edge cost functions for a given graph and the

[1] All authors (ordered alphabetically) have equal contributions and are corresponding authors.

goal is to find an s-t path that minimizes the maximum cost across all the cost functions. Apart from serving a model for handling uncertainty, robustness also offers a method to integrate multiple objectives and fairness requirements.

In a routing network, each link (or edge) typically possesses multiple attributes, such as usage cost and delay. By representing each attribute as a cost function, we can formulate the multi-objective routing problem as our model [14, 15, 31]. In the context of fairness computation, diverse edge cost functions can be interpreted as various perspectives of agents on the edges. Our objective is to identify a public path that can accommodate these perspectives under the notion of min-max fairness [1, 27, 30].

The Robust s-t Path problem was initially studied by [18], and since then it has received widespread attention due to its broad applicability. In [18], the authors demonstrated that the problem is strongly NP-Hard even when there are only two scenarios. Later, [2] considers the problem with the constant number of scenarios, and shows that the problem admits a fully polynomial-time approximation scheme (FPTAS). When the number k of scenarios is part of the input, simply computing the shortest path w.r.t the summation of the k cost functions can obtain an approximation ratio of k. Kasperski and Zielinski [20] proved that the problem is hard to approximate within a factor of $o(\log k)$ unless NP $\subseteq$ DTIME($n^{\text{poly}\log n}$). Whether a polylogarithmic approximation can be achieved has been open ever since. A recent breakthrough in the approximation ratio is made by Kasperski and Zielinski [23] in which they gave a flow-LP-based algorithm that is $\widetilde{O}(\sqrt{n})$-approximate, where we use $\widetilde{O}$ to hide a polylogarithmic factor. They further showed that their analysis is nearly tight by proving an integrality gap of $\Omega(\sqrt{n})$ for the flow LP.

It should be noted that Bilò et al. [7] studied the ℓ_q-norm shortest path problem which is a generalized version of robust s-t path, i.e., the problem aims to find a s-t path $\mathcal{P}$ to minimize the value of $\left(\sum_{i\in[k]} c_i(\mathcal{P})^q\right)^{1/q}$, where $c_i(\mathcal{P})$ is agent i's cost for the selected path $\mathcal{P}$. Their algorithm [7, Algorithm 2]) extends the classical Dijkstra algorithm by replacing the distance with the ℓ_q-norm metric. Our directed graph is a DAG, so the Dijkstra-type algorithm becomes a dynamic programming-type algorithm, with nodes processed using a topological order. So, their algorithm just stores the best path in the ℓ_q-norm for every node. It is claimed in [7, Theorem 14] that such an algorithm achieves $O(\min\{q, \log k\})$-approximation for the ℓ_q-norm shortest path problem. However, unfortunately, there exists a crucial error in the analysis. In the full version [25], we give a hard instance on a series-parallel graph for the algorithm and show that the approximation ratio is at least $k^{1-1/q}$. In other words, when $q = O(\log k)$, the proposed algorithm [7, Algorithm 2] is $O(k)$-approximate for the robust s-t problem.

1.1 Our Contributions

This paper makes significant progress in closing the gap between the known upper and the lower bound for Robust s-t Path. We show that for two natural graph classes, a polylogarithmic approximation can be obtained in polynomial time; while for general graphs, there exists a polylogarithmic approximated algorithm running in a quasi-polynomial time. The following formalizes the model and summarizes our main results.

The Robust s-t Path Problem. Consider a directed graph $G(V, E)$ with n vertices and m edges. There are k scenarios (also referred to as "agents" hereafter), where each scenario $i \in [k]$ has an edge cost function $c_i : E \to \mathbb{R}_{\geq 0}$. Given two specified vertices s and t in the graph, the goal is to find an s-t path $\mathcal{P}$ that minimizes $\max_{i\in[k]} c_i(\mathcal{P})$, where $c_i(\mathcal{P}) := \sum_{e\in\mathcal{P}} c_i(e)$.

Main Result 1 (Theorem 1). There is a randomized polynomial-time $O(H \log k)$-approximation algorithm for Robust s-t Path for directed series-parallel graphs, where H is the height of the decomposition tree of the series-parallel graph and k is the number of agents.

Our first result is on the class of *series-parallel graphs* (Section 2), which are used by [20] to demonstrate a lower bound of $\Omega(\log^{1-\epsilon} k)$ (for any $\epsilon > 0$) for the Robust s-t Path problem. We begin by showing that the natural flow linear program (LP) for this class has integrality gaps of $\Omega(k)$ and $\Omega(\sqrt{n})$. The gaps hold even when we integrate the knowledge of the optimum cost to the LP to circumvent some obvious gap instances. This result aligns with the prior findings of [23], but our constructed instance is significantly simpler. It is worth noting that most prior algorithms in the existing literature rely on the flow LP mentioned above, and thus, their approximation ratios cannot be better than $O(\min\{k, \sqrt{n}\})$.

To overcome the gap, we develop a novel linear program based on the *decomposition tree* of series-parallel graphs and demonstrate that a dependent randomized rounding algorithm for the LP obtains an approximation ratio of $O(H \log k)$. Particularly, for the hard instance that leads to a lower bound $\Omega(\log^{1-\epsilon} k)$ (for any $\epsilon > 0$) stated in [20], our algorithm can return a $O(\log \log n \log k)$-approximate solution, which is nearly tight since there is only a $O(\log \log n)$-gap; the detailed discussion can be found in the full version [25].

Main Result 2 (Theorem 8). Given a directed graph with bounded treewidth, there is a randomized algorithm that obtains an approximation ratio of $O(\log n \log k)$ in polynomial time, where n is the number of vertices and k is the number of agents.

We then consider *graphs with bounded treewidth* (Section 3). When the treewidth is 2, it becomes the class of series-parallel graphs. Therefore, combining the above two results gives a $O(\min\{H, \log n\} \cdot \log k)$-approximation for series-parallel graphs. Besides series-parallel graphs, the graph class includes many other common graphs, such as trees, pseudoforests, Cactus graphs, outerplanar graphs, and Halin graphs. In this part, we employ the nice properties provided by the treewidth decomposition of these graphs and obtain a polylogarithmic approximation.

Main Result 3 (Theorem 12). Given any directed graph, there is a randomized algorithm that obtains a $O(\log n \log k)$-approximate solution in quasi-polynomial time, where n is the number of vertices and k is the number of agents. Moreover, any quasi-polynomial time algorithm for Robust s-t Path has an approximation lower bound of $\Omega(\log^{1-\epsilon} k)$ (even on series-parallel graphs) under the assumption that NP $\nsubseteq$ DTIME$(n^{\text{poly} \log n})$.

Finally, we consider general graphs (Section 4). The algorithm is also LP-based, following a similar framework as the algorithm for series-parallel graphs. The main challenge here is that we no longer have a simple tree structure for general graphs. To address this issue, we construct a decision-tree-type tree structure for the given graph and write a linear program based on it. Our algorithm then builds on this new LP to give the first polylogarithmic approximation for general graphs. Additionally, we show that the lower bound of $\Omega(\log^{1-\epsilon} k)$ can be extended to the algorithms running in quasi-polynomial time, i.e., the problem is still hard to approximate within $o(\log k)$ even if we allow quasi-polynomial time algorithms. This part is omitted in this version and can be found in the full version [25].

Main Result 4. For the problems of Robust s-t Path, weighted independent set, and spanning tree under the *maximin criteria*, it is NP-Hard to determine whether their instances have zero-cost optimal solutions or not. This implies that these problems do not admit any polynomial time α-approximate algorithm unless P = NP, where α is an arbitrary function of the input.

The paper also considers the *maximin criteria*, where the goal is to maximize the minimum cost among all agents. By observing that the classic algorithms (e.g., Dijkstra's algorithm) that work for the shortest path problem on DAGs (directed acyclic graphs) also work for the longest path problem on DAGs, one might expect that the maximin criteria is also a candidate objective to investigate the robustness of the s-t path problem, i.e., finding an s-t path $\mathcal{P}$ such that $\min_{i\in[k]} c_i(\mathcal{P})$ is maximized. We demonstrate this is not the case by providing a strong lower bound for the problem under the maximin criteria. Our reduction builds on a variant of the set cover problem. Employing a similar basic idea of the reduction, we also show that the maximin weighted independent set problem on trees or interval graphs is not approximable. This constitutes a strong lower bound for this problem, while the previous works [24, 28] only show the NP-Hardness. Our reduction idea can further be extended to the maximin spanning tree problem, which implies that the robust spanning tree problem is also not approximable under the maximin objective. This part is completely omitted in this version and can be found in the full version [25].

1.2 Other Related Works

Robust Minimax Combinatorial Optimization. Robust minimax optimization under different combinatorial structures has been extensively studied in the past three decades. See [3, 22] for a survey. Many problems that are polynomial-time solvable in the normal setting are shown to be NP-hard in the robust minimax setting: spanning trees, s-t cuts, and perfect matching on bipartite graphs [22]. Besides these fundamental problems, the minimax submodular ranking problem was studied in [10] very recently. For the minimax spanning tree, a $O(\log k/\log\log k)$-approximation algorithm is known [9], which is almost tight by the lower bound of $\Omega(\log^{1-\epsilon} k)$ (for any $\epsilon > 0$) stated in [21]. The problem of minimax perfect matching has a lower bound of $\Omega(\log^{1-\epsilon} k)$ (for any $\epsilon > 0$) [20], while the best upper bound so far is still $O(k)$ which is trivial. In the case where k is a constant, fully polynomial time approximation schemes are known for spanning trees, perfect matching, knapsacks, and s-t paths [3, 4, 22, 29].

Multiobjective s-t Path. Finding an s-t path is a fundamental problem in multi-objective optimization [15]. An Excellent survey of multiobjective combinatorial optimization, including multiobjective s-t path, can be found in [11]. Typically, we are given a directed graph $G := (V, E)$. Each edge $e \in E$ has a positive cost vector $\mathbf{c}(e) := (c_1(e), \ldots, c_k(e))$. For every s-t path $\mathcal{P} \subseteq E$, we have a cost vector $\mathbf{c}(\mathcal{P}) = (c_1(\mathcal{P}), \ldots, c_k(\mathcal{P}))$ with $c_i(\mathcal{P}) = \sum_{e\in\mathcal{P}} c_i(e)$. The goal is to compute an s-t path $\mathcal{P}$ such that $\mathcal{P}$ is *Pareto optimal*. An s-t path is called Pareto optimal if there is no other s-t path that makes one objective better off without making another worse off. Not surprisingly, this problem has been shown to be NP-hard even if the cost vector only has two coordinates [32] in which the problem is called the bi-objective s-t path minimization problem. Bi-objective s-t path minimization has also been studied extensively [14, 31], in which researchers mainly focus on the exact algorithms with exponential running time. In addition, a fully polynomial time approximation scheme is proposed by [29].

Fair Allocation with Public Goods. By observing the minimax objective as a fairness criterion, our problem shares some similarities with the problem of public goods, which was first used to distinguish the previous private goods by Conitzer et al. [12] in the field of fair division. Specifically, there is a multiagent system and different agents hold different opinions about the same goods. And, they aim to select a feasible set of goods to satisfy the various

fairness notions, such as propositional share or its generalization [12, 16]. In [16], they study some constraints of goods, i.e., the selected goods must form a matching or matroid. The minimax criterion is quite different from other fairness measures in the fair division field, which leads to different techniques.

1.3 Roadmap

Section 2 and Section 3 present results on series-parallel graphs and bounded-treewidth graphs, respectively. Subsequently, in Section 4, the general graph case is considered. Section 5 finally concludes the paper. Due to space constraints, all the results on the maximin criteria are deferred to the full version of the paper.Note that we focus on high-level descriptions of our methods in the main body. Some formal descriptions and proofs (including lemma statements) can be found in the corresponding appendices.

2 Series-Parallel Graphs

In this section, we show that there is a randomized algorithm that achieves $O(H \log k)$ approximation for series-parallel graphs, where H is the height of the series-parallel graph's decomposition tree; its meaning will be clear later. The algorithm can be viewed as a warm-up example for the general graph, as the algorithm for the general graph follows a similar algorithmic framework. Formally, we shall show the following theorem (Theorem 1) in this section. We only present the LP formulation and the complete algorithm due to space limits. All proofs can be found in the paper's full version.

▶ **Theorem 1.** *Given any series-parallel graph G, there is a polynomial time algorithm that returns a $O(H \log k)$-approximation solution with probability at least $1 - (\frac{1}{k} + \frac{1}{kH})$ for robust s-t path, where H is the height of G's decomposition tree and k is the number of agents.*

In Section 2.1, we give the basic concepts and properties of the series-parallel graphs, which we will use later to build our linear programming formulation. In Section 2.2, we formally present our LP formulation. We give the complete and rounding algorithm in Section 2.3. Finally, we show the analysis in Section 2.4.

2.1 Basic Concepts

▶ **Definition 2** (Series-Parallel Graph). *A directed graph $G := (V, E, s, t)$ with source s and sink t is called a* series-parallel graph, *if it contains a single edge from s to t, or it can be built inductively using the following* series *and* parallel composition *operations. The series composition of two-terminal graphs $G_1 := (V_1, E_1, s_1, t_1)$ and $G_2 := (V_2, E_2, s_2, t_2)$ is to identify t_1 and s_2, and let s_1 and t_2 be the new source and sink in the resulting graph. The parallel composition of two-terminal graphs $G_1 := (V_1, E_1, s_1, t_1)$ and $G_2 := (V_2, E_2, s_2, t_2)$ is to identify s_1 with s_2 and t_1 with t_2 respectively, and let $s_1 = s_2$ and $t_1 = t_2$ be the new source and sink.*

A series-parallel graph can be represented in a natural way by a tree structure that describes how to assemble some small graphs into a final series-parallel graph through series and parallel composition. Such a tree structure is commonly called the *decomposition tree* of the series-parallel graph in the literature [33]. Formally, a decomposition tree $\mathbf{T} := (\mathbf{V}, \mathbf{E})$ of a series-parallel graph $G := (V, E)$ is a tree such that (i) each leaf node $\mathbf{u} \in \mathbf{V}$ corresponds an edge in E; (ii) each internal node is either a series or parallel node; (iii) the child nodes of a parallel (resp. series) node must be leaf nodes or series (resp. parallel) nodes. The series

(resp. parallel) node corresponds to the series (resp. parallel) composition, and they are used to indicate how to merge the small subgraphs of its child nodes. The subgraph $G_{\mathbf{u}}$ of a node $\mathbf{u}$ is a subgraph of G such that $G_{\mathbf{u}}$ only contains those edges corresponding to the leaf nodes in the subtree rooted at $\mathbf{u}$. Let H be the height of $\mathbf{T}$. Given a series-parallel graph, it is known that its decomposition tree can be built in linear time by the standard series-parallel graph recognition algorithm [33]. An example can be found in Figure 1.

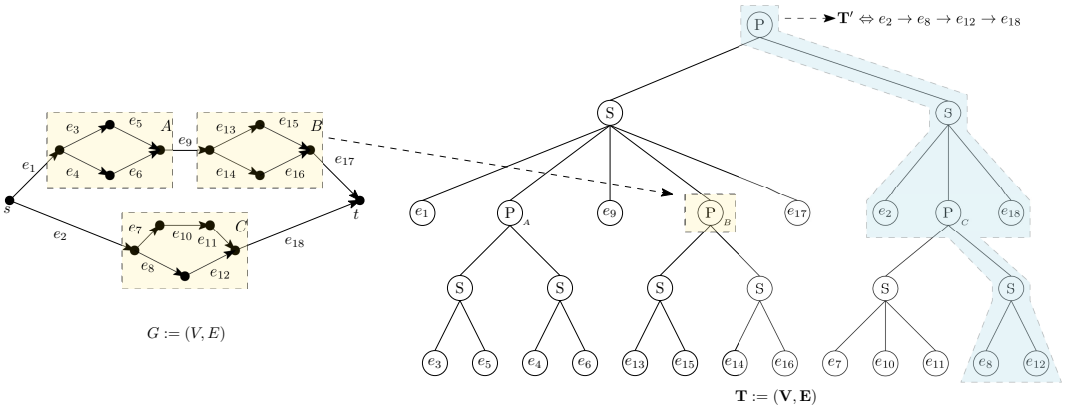


Figure 1 An example of the decomposition tree of a series-parallel graph. The series-parallel graph $G := (V, E)$ is shown on the left and its decomposition tree $\mathbf{T} := (\mathbf{V}, \mathbf{E})$ is shown on the right. Each leaf node in $\mathbf{T}$ corresponds to an edge in E. Each internal node is either a series node S or a parallel node P. And it indicates how to merge the child nodes' subgraph. For example, consider the S node and its two child nodes e_{13} and e_{15}. Then, the subgraph of this S node is $e_{13} \to e_{15}$ which merges its two child nodes' subgraph via the series composition. And also, the subgraph of this S node's parent corresponds to the subgraph B in G, which merges $e_{13} \to e_{15}$ and $e_{14} \to e_{16}$ via the parallel composition. An s-t path corresponds to a feasible subtree (Definition 3). For example, the feasible subtree $\mathbf{T}'$ can be converted to an s-t path $e_2 \to e_8 \to e_{12} \to e_{18}$.

We remark that the children of a parallel node are unordered, and for a series node, the children should be considered as ordered. However, for the s-t path problem, the order is not important, as permuting the children of a series node will lead to an equivalent instance.

We aim to give a linear program based on the decomposition tree $\mathbf{T}$. Clearly, not all subtrees of $\mathbf{T}$ correspond to an s-t path of G. In the following, we introduce the concept of a special subtree of $\mathbf{T}$ called *feasible subtree* (Definition 3), which is able to be converted into an s-t path.

▶ **Definition 3** (Feasible Subtree). *A subtree $\mathbf{T}'$ of $\mathbf{T}$ is called a feasible subtree if and only if (i) $\mathbf{T}'$ includes the root node of $\mathbf{T}$; (ii) for every series node $\mathbf{s}$ in $\mathbf{T}'$, $\mathbf{T}'$ includes all child nodes of $\mathbf{s}$; (iii) for every parallel node $\mathbf{p}$ in $\mathbf{T}'$, $\mathbf{T}'$ includes exactly one child node of $\mathbf{p}$.*

It now remains to define a cost function $f_i : 2^{\mathbf{V}} \to \mathbb{R}_{\geq 0}$ according to the cost function c_i. Since each edge corresponds to a unique leaf node in $\mathbf{T}$, it is easy to define f_i by c_i: for each $\mathbf{v} \in \mathbf{V}$, $f_i(\mathbf{v}) := c_i(e)$ if node $\mathbf{v}$ corresponds to some edge e; otherwise $f_i(\mathbf{v}) := 0$. Formally, we have the following simple observation (Observation 4).

▶ **Observation 4.** *Given any series-parallel graph $G := (V, E)$ and its decomposition tree $\mathbf{T} := (\mathbf{V}, \mathbf{E})$, fix an arbitrary agent $i \in [k]$, any s-t path $\mathcal{P}$ of G with the cost $c_i(\mathcal{P})$ corresponds to a feasible subtree $\mathbf{T}'$ of $\mathbf{T}$ with the cost $f_i(\mathbf{T}')$ such that $f_i(\mathbf{T}') = c_i(\mathcal{P})$ and vice versa.*

2.2 LP Formulation

Given any series-parallel graph $G := (V, E)$, we first employ the standard doubling technique to enhance the linear program. Given a guess of the optimal objective value GS, we discard those edges e such that there exists an agent $i \in [k]$ with $c_i(e) > \mathsf{GS}$. Clearly, these discarded edges cannot belong to the optimal solution. Then, we run the series-parallel graph recognition algorithm [33] to construct a decomposition tree $\mathbf{T} := (\mathbf{V}, \mathbf{E})$ of the series-parallel graph. See Algorithm 1 for the complete description.

The linear program is shown in (Tree-LP). For an internal node $\mathbf{v} \in \mathbf{V}$, use $\mathsf{child}(\mathbf{v})$ and $\Lambda(\mathbf{v})$ to denote its children and descendants in the tree, respectively. Let $P(\mathbf{T})$ and $S(\mathbf{T})$ be the set of parallel and series nodes in $\mathbf{T}$. For each node $\mathbf{v}$, $x_{\mathbf{v}}$ is a relaxed indicator variable denoting whether $\mathbf{v}$ is selected or not. The three constraints (2), (3) and (4) correspond to the three conditions stated in Definition 3 respectively, in order to ensure that the solution is a feasible subtree. The first constraint type (1) is a bit subtle, and it is the key that allows us to surpass the pessimistic $\Omega(k)$ and $\Omega(\sqrt{n})$ integrality gap. In these constraints, $\sum_{\mathbf{u}\in\Lambda(\mathbf{v})} x_{\mathbf{u}} \cdot f_i(\mathbf{u})$ denotes the cost of the selected subtree rooted at $\mathbf{v}$ with respect to agent i. Thus, when $\mathbf{v} = \mathbf{r}$, the constraint implies that for any agent, the total cost of all the selected nodes must be at most $x_{\mathbf{r}} \cdot \mathrm{OPT} = \mathrm{OPT}$. For the cases that $\mathbf{v} \neq \mathbf{r}$, these constraints do not affect the feasible region of integer solutions since $x_{\mathbf{v}}$ is either 1 or 0, but they can reduce the fractional solution's feasible region dramatically by restricting the contribution of each subtree $\Lambda(\mathbf{v})$. A more detailed discussion is given in the full version of the paper.

(Tree-LP)

$$\sum_{\mathbf{u}\in\Lambda(\mathbf{v})} x_{\mathbf{u}} \cdot f_i(\mathbf{u}) \le x_{\mathbf{v}} \cdot \mathsf{GS}, \qquad \forall i \in [k], \forall \mathbf{v} \in \mathbf{V} \tag{1}$$

$$x_{\mathbf{r}} = 1, \tag{2}$$

$$\sum_{\mathbf{u}\in\mathsf{child}(\mathbf{v})} x_{\mathbf{u}} = x_{\mathbf{v}}, \qquad \forall \mathbf{v} \in P(\mathbf{T}) \tag{3}$$

$$x_{\mathbf{u}} = x_{\mathbf{v}}, \qquad \forall \mathbf{v} \in S(\mathbf{T}), \mathbf{u} \in \mathsf{child}(\mathbf{v}) \tag{4}$$

$$x_{\mathbf{v}} \ge 0, \qquad \forall \mathbf{v} \in \mathbf{V} \tag{5}$$

2.3 Algorithms

This section formally describes the complete algorithm (Algorithm 1) for series-parallel graphs. The main algorithm mainly consists of two steps: the doubling step (lines 2-13 of Algorithm 1) and the rounding algorithm (Algorithm 2). After finishing the doubling step, we obtain a fractional solution x^* with the value of GS that is close to the optimal solution OPT (Observation 5). Then, we shall employ a dependent rounding algorithm to obtain a feasible subtree based on x^*. This dependent randomized rounding algorithm selects nodes level by level, starting from the top of $\mathbf{T}$ and proceeding downwards. For parallel nodes, the algorithm selects one of its child nodes with a probability determined by the optimal fractional solution x^*. For series nodes, the algorithm selects all of its child nodes with a probability of 1, ensuring that the resulting subtree is always feasible. A formal description of the algorithm can be found in Algorithm 2.

2.4 Analysis

This section analyzes the performance of our algorithm. We start by describing a simple observation (Observation 5). Let $\mathbf{T}'$ be the subtree returned by Algorithm 2. Recall that H is the height of $\mathbf{T}$.

Algorithm 1 The Complete Algorithm for Series-Parallel Graphs.

Input: A series-parallel graph $G := (V, E)$ with k cost functions $c_i : 2^E \to \mathbb{R}_{\geq 0}, i \in [k]$.
Output: An s-t path $\mathcal{P} \subseteq E$.

1: $\texttt{flag} \leftarrow \texttt{true}$; $\mathsf{GS} \leftarrow \max_{i\in[k]} \sum_{e\in E} c_i(e)$.
2: **while** $\texttt{flag} = \texttt{true}$ **do**
3: $\quad E' \leftarrow \{\, e \in E \mid \exists i \in [k] \text{ s.t. } c_i(e) > \mathsf{GS} \,\}$.
4: $\quad E \leftarrow E \setminus E'$; $G \leftarrow (V, E')$.
5: $\quad$ Compute the tree decomposition $\mathbf{T} := (\mathbf{V}, \mathbf{E})$ of G by [33].
6: $\quad$ Solve the linear program (Tree-LP).
7: $\quad$ **if** (Tree-LP) has a feasible solution **then**
8: $\quad\quad$ Let x^* be a feasible solution to (Tree-LP).
9: $\quad\quad \mathsf{GS} \leftarrow \frac{\mathsf{GS}}{2}$.
10: $\quad$ **else**
11: $\quad\quad \texttt{flag} \leftarrow \texttt{false}$.
12: $\quad$ **end if**
13: **end while**
14: Run Algorithm 2 with the optimal solution x^* to obtain a feasible subtree $\mathbf{T}'$ of $\mathbf{T}$.
15: Convert $\mathbf{T}'$ into an s-t path $\mathcal{P}$.
16: **return** $\mathcal{P}$.

▶ **Observation 5.** *Let GS^* be the guessing value at the beginning of the last round of the while-loop (lines 2-13 of Algorithm 1). Then, we have $\mathsf{GS}^* \leq 2 \cdot \mathrm{OPT}$*[2].

To show that the approximation ratio is $O(H \log k)$ with a constant probability, a natural step is to first bound the expectation of our solution. We first state some intuition. According to the description of the rounding scheme, it is easy to see that for each agent i,

$$\mathbb{E}[f_i(\mathbf{T}')] = \sum_{e\in E} c_i(e) \Pr[e \in \mathbf{T}'] = \mathsf{GS}^*.$$

Then by Markov inequality, we have for each agent i,

$$\Pr[f_i(\mathbf{T}') \geq H \log k \cdot \mathsf{GS}^*] \leq \frac{1}{H \log k}.$$

However, the above inequality is not sufficient because proving Theorem 1 needs to show that $\Pr[\forall i \in [k], f_i(\mathbf{T}') \geq H \log k \cdot \mathsf{GS}^*]$ is at most $\frac{1}{k} + \frac{1}{kH}$. To address this issue, we need to employ an analysis technique called *Moment Method*, which is widely used in the literature [13, 19]. More formally, we aim to show the following key lemma (Lemma 6); a similar proof can also be found in [13].

▶ **Lemma 6.** *For any agent $i \in [k]$, we have $\mathbb{E}[\exp\left(\ln(1 + \frac{1}{2H}) \cdot f_i(\mathbf{T}')\right)] \leq 1 + \frac{1}{H}$.*

Proof. We prove the theorem inductively. First, consider the case that $H = 1$, i.e., the decomposition tree $\mathbf{T}$ only contains a root $\mathbf{r}$. Since $x_{\mathbf{r}} = 1$, there is no randomness in selecting $\mathbf{T}'$. Thus, we have for any $z \geq 1$,

$$\mathbb{E}\left[z^{\frac{f_i(\mathbf{T}')}{\mathsf{GS}^*}}\right] = z^{\frac{x_{\mathbf{r}} \cdot f_i(\mathbf{r})}{\mathsf{GS}^*}} \leq z,$$

where the last inequality uses constraint (1) in (Tree-LP).

[2] One can get a more accurate lower bound of the optimal solution (e.g., $\mathsf{GS}^* \leq (1+\epsilon) \cdot \mathrm{OPT}$ for any $\epsilon > 0$) by the standard binary search technique.

Algorithm 2 Dependent Randomized Rounding.

Input: A tree structure $\mathbf{T}(\mathbf{V}, \mathbf{E})$ rooted at $\mathbf{r}$; a fractional solution $x^* \in [0,1]^{|\mathbf{V}|}$.
Output: A feasible subtree $\mathbf{T}'$.

Initially, set $\mathbf{T}' \leftarrow \emptyset$ and a vertex queue $\mathcal{Q} \leftarrow \{\mathbf{r}\}$.
while $\mathcal{Q} \neq \emptyset$ **do**
  Use $\mathbf{v}$ to represent the front element of $\mathcal{Q}$.
  $\mathbf{T}' \leftarrow \mathbf{T}' \cup \{\mathbf{v}\}$, $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{\mathbf{v}\}$.
  if $\mathbf{v}$ is a parallel node **then**
    Pick one node $\mathbf{u} \in \mathsf{child}(\mathbf{v})$ randomly such that $\mathbf{u}$ is chosen with probability $\frac{x_{\mathbf{u}}}{x_{\mathbf{v}}}$.
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\mathbf{u}\}$.
  end if
  if $\mathbf{v}$ is a series node **then**
    for each $\mathbf{u} \in \mathsf{child}(\mathbf{v})$ **do**
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\mathbf{u}\}$.
    end for
  end if
end while
return $\mathbf{T}'$.

To streamline the analysis, we continue considering the case that $H = 2$. We further distinguish two subcases: (i) root $\mathbf{r}$ is a parallel node; (ii) root $\mathbf{r}$ is a series node. For the first subcase, Algorithm 2 selects exactly one child $\mathbf{v} \in \mathsf{child}(\mathbf{r})$ with probability $x_{\mathbf{v}}/x_{\mathbf{r}} = x_{\mathbf{v}}$. According to the law of total expectation and only leaves in $\mathbf{T}$ have non-zero costs, we have

$$\mathbb{E}\left[z^{\frac{f_i(\mathbf{T}')}{\mathsf{GS}^*}}\right] = \sum_{\mathbf{v}\in\mathsf{child}(\mathbf{r})} \Pr[\mathbf{v} \in \mathbf{T}'] \cdot \mathbb{E}\left[z^{\frac{f_i(\Lambda'(\mathbf{v}))}{\mathsf{GS}^*}} \middle| \mathbf{v} \in \mathbf{T}'\right],$$

where $\Lambda'(\mathbf{v}) := \Lambda(\mathbf{v}) \cap \mathbf{T}'$. Observing that once conditioned on $\mathbf{v} \in \mathbf{T}'$, the conclusion for the $H = 1$ case can be used to bound the expectation, we have

$$\begin{aligned}
\mathbb{E}\left[z^{\frac{f_i(\mathbf{T}')}{\mathsf{GS}^*}}\right] &= \sum_{\mathbf{v}\in\mathsf{child}(\mathbf{r})} \Pr[\mathbf{v} \in \mathbf{T}'] \cdot \mathbb{E}\left[z^{\frac{f_i(\Lambda'(\mathbf{v}))}{\mathsf{GS}^*}} \middle| \mathbf{v} \in \mathbf{T}'\right], \\
&= \sum_{\mathbf{v}\in\mathsf{child}(\mathbf{r})} x_{\mathbf{v}} \cdot z^{\frac{f_i(\mathbf{v})}{\mathsf{GS}^*}} \\
&\leq \sum_{\mathbf{v}\in\mathsf{child}(\mathbf{r})} x_{\mathbf{v}} \cdot \left(1 + (z-1) \cdot \frac{f_i(\mathbf{v})}{\mathsf{GS}^*}\right) \\
&\qquad \text{(Constraint (1) and } z^r \leq 1 + r(z-1) \forall z > 0, r \in [0,1]) \\
&= \left(\sum_{\mathbf{v}\in\mathsf{child}(\mathbf{r})} x_{\mathbf{v}}\right) + (z-1) \cdot \frac{\sum_{\mathbf{v}\in\mathsf{child}(\mathbf{r})} x_{\mathbf{v}} \cdot f_i(\mathbf{v})}{\mathsf{GS}^*} \\
&= 1 + (z-1) \cdot \frac{\sum_{\mathbf{v}\in\Lambda(\mathbf{r})} x_{\mathbf{v}} \cdot f_i(\mathbf{v})}{\mathsf{GS}^*} \qquad \text{(Constraint (3))} \\
&\leq e^{(z-1)\cdot\frac{\sum_{\mathbf{v}\in\Lambda(\mathbf{r})} x_{\mathbf{v}}\cdot f_i(\mathbf{v})}{\mathsf{GS}^*}}. \qquad (1 + x \leq e^x)
\end{aligned}$$

For the second subcase, according to Constraint (4), we have $x_{\mathbf{v}} = x_{\mathbf{r}} = 1$ for each $x_{\mathbf{v}} \in \mathsf{child}(\mathbf{r})$, and therefore,

$$\mathbb{E}\left[z^{\frac{f_i(\mathbf{T}')}{\mathsf{GS}^*}}\right] = \mathbb{E}\left[z^{\frac{\sum_{\mathbf{v}\in\mathsf{child}(\mathbf{r})} f_i(\Lambda'(\mathbf{v}))}{\mathsf{GS}^*}}\right] \leq z^{\frac{\sum_{\mathbf{v}\in\Lambda(\mathbf{r})} x_{\mathbf{v}}\cdot f_i(\mathbf{v})}{\mathsf{GS}^*}}.$$

Since $z \leq e^{z-1}$, combing the two subcases, we have that when $H = 2$,

$$\mathbb{E}\left[z^{\frac{f_i(\mathbf{T}')}{\mathsf{GS}^*}}\right] \leq (e^{(z-1)})^{\frac{\sum_{\mathbf{v}\in\Lambda(\mathbf{r})} x_{\mathbf{v}}\cdot f_i(\mathbf{v})}{\mathsf{GS}^*}}.$$

The above inequality shows that as the height increases by 1, the upper bound of the expectation grows exponentially. Furthermore, when the height increases from 1 to H, we can obtain a sequence $z_1 = z, z_2, \ldots, z_H$, where $z_h = e^{z_{h-1}-1}$ for each $h > 1$, and have

$$\mathbb{E}\left[z^{\frac{f_i(\mathbf{T}')}{\mathsf{GS}^*}}\right] \leq z_h^{\frac{\sum_{\mathbf{v}\in\Lambda(\mathbf{r})} x_{\mathbf{v}}\cdot f_i(\mathbf{v})}{\mathsf{GS}^*}},$$

for any $\mathbf{T}$ with height h by almost the same analysis as above. Due to Constraint (1), we have $\sum_{\mathbf{v}\in\Lambda(\mathbf{r})} x_{\mathbf{v}} \cdot f_i(\mathbf{v}) \leq \mathsf{GS}^*$, and thus, $\mathbb{E}\left[z^{\frac{f_i(\mathbf{T}')}{\mathsf{GS}^*}}\right]$ is at most z_h.

Finally, to obtain the claimed upper bound, we set $z = 1 + \frac{1}{2H}$ such that $z_h \leq 1 + \frac{1}{H}$, and complete the proof. ◀

▶ **Lemma 7.** *Consider an arbitrary agent i, Algorithm 2 returns a feasible subtree $\mathbf{T}'$ such that $f_i(\mathbf{T}') \leq 4H \cdot \log k \cdot \mathsf{GS}$ with high probability, where H is the height of the decomposition tree of the series-parallel graph and GS is a guess of the optimal objective value such that the corresponding* (Tree-LP) *admits a feasible solution.*

Lemma 7 can be proved by Lemma 6 and Markov bound. Theorem 1 can be proved by Lemma 7 and union bound. All proofs are deferred to the full version of the paper.

3 Graphs with Bounded Treewidth

This section considers robust *s-t* path on graphs with bounded treewidth. We mainly show the following theorem. Noting that any series-parallel graph has a treewidth of 2, this result improves upon the above $O(H \log k)$ ratio for series-parallel graphs with large H.

▶ **Theorem 8.** *Given any directed graph G with treewidth $\mathsf{tw}(G) \leq \ell$, there is an algorithm that returns a $O(\log n \log k)$-approximate solution in poly$(n) \cdot n^{O(\ell^2)}$ time with probability at least $1 - (\frac{1}{k} + \frac{1}{k \log n})$ for robust s-t path, where n is the number of vertices and k is the number of agents.*

3.1 Algorithmic Framework

The basic idea of the algorithm is to reduce our problem to *the tree labeling problem* which was proposed by Dinitz et al. [13] very recently. In their paper, they provided a randomized algorithm for the tree label problem. Applying the algorithm to our reduced instance can obtain an *s-t* path whose expected cost with respect to each agent is bounded. Finally, we employ the concentration inequalities to show that with high probability, the returned path is a polylogarithmic approximation solution. To ensure the reduction's correctness, we also need to utilize some other tools. The complete description of our algorithm can be found in

the full version of the paper. Due to space limitations, in the main body, we only focus on the core of our algorithm – the reduction to tree labeling. Before introducing the reduction, it is necessary to restate the definition of tree labeling and the result proved in [13].

The Tree-labeling Problem. Consider a binary tree $\mathbf{T}(\mathbf{V}, \mathbf{E})$ rooted at $\mathbf{r} \in \mathbf{V}$. For each node $\mathbf{v} \in \mathbf{V}$, there is a finite set $L_{\mathbf{v}}$ of labels for $\mathbf{v}$. Let $L := \bigcup_{\mathbf{v}\in\mathbf{V}} L_{\mathbf{v}}$ be the set of all possible labels. The output is a label assignment $\mathcal{L} := (l_{\mathbf{v}} \in L_{\mathbf{v}})_{\mathbf{v}\in\mathbf{V}}$ of the node set $\mathbf{V}$, that satisfies the consistency and cost constraints.

- **(Consistency Constraints)** For every internal node $\mathbf{v}$ of $\mathbf{T}$ with two children $\mathbf{u}$ and $\mathbf{w}$ ($\mathbf{u}$ or $\mathbf{w}$ is possibly empty), we are given a set $\Gamma(\mathbf{v}) \subseteq L_{\mathbf{u}} \times L_{\mathbf{v}} \times L_{\mathbf{w}}$. A valid labeling $\mathcal{L} = (l_{\mathbf{v}} \in L_{\mathbf{v}})_{\mathbf{v}\in\mathbf{V}}$ must satisfy $(l_{\mathbf{u}}, l_{\mathbf{v}}, l_{\mathbf{w}}) \in \Gamma(\mathbf{v})$ for every internal node $\mathbf{v}$.
- **(Cost Constraints)** There are k additive cost functions $f_1, \ldots, f_k$ defined over the labels, i.e., for each $i \in [k]$, $f_i : L \to \mathbb{R}_{\geq 0}$. For each $i \in [k]$, a valid labeling $\mathcal{L}$ needs to satisfy $f_i(\mathcal{L}) := \sum_{\mathbf{v}\in\mathbf{V}} f_i(l_{\mathbf{v}}) \leq 1$.

A label assignment $\mathcal{L} := (l_{\mathbf{v}} \in L_{\mathbf{v}})_{\mathbf{v}\in\mathbf{V}}$ is called *consistent* if it satisfies the consistency constraints; it is *valid* if it satisfies both the consistency and cost constraints. Let H be the height of $\mathbf{T}$ and let $\Delta := \max_{\mathbf{v}\in\mathbf{V}} |L_{\mathbf{v}}|$ be the maximum size of any label set. Let n be the number of nodes in $\mathbf{T}$. In [13], they show the following result.

▶ **Lemma 9** ([13]). *Given a tree labeling instance such that the instance admits a valid label assignment. There is a randomized algorithm that in time poly$(n) \cdot \Delta^{O(H)}$ outputs a consistent label assignment $\mathcal{L}$ such that for every $i \in [k]$, we have* $\mathbb{E}\left[\exp\left(\ln(1 + \frac{1}{2H}) \cdot f_i(\mathcal{L})\right)\right] \leq 1 + \frac{1}{H}$.

3.2 Reduction Intuition

In this section, we give some intuition of our reduction. The formal description and an example can be found in the next section (Section 3.3). Given any directed graph with bounded treewidth, we aim to construct a tree-labeling instance such that the solution to the constructed tree-labeling instance can be converted into an s-t path of the original graph with some cost-preserved property. Note that the treewidth decomposition $\mathbf{T}(\mathbf{V}, \mathbf{E})$ of any input graph G can be computed efficiently [8]. We directly let the treewidth decomposition $\mathbf{T}$ be the binary tree in the reduced tree-labeling instance. The following shows how to construct the labels and the corresponding constraints such that a feasible label assignment can be successfully transformed into an s-t path.

Label Construction. For a graph's treewidth decomposition, each node $\mathbf{v} \in \mathbf{T}$ corresponds to a node subset $X(\mathbf{v})$ of the original graph. Each edge in the original graph is guaranteed to be covered by some $X(\mathbf{v})$, which is the completeness property of a tree decomposition. Use $C(\mathbf{v})$ to denote the edges covered by node $\mathbf{v} \in \mathbf{T}$. Without loss of generality, we can assume that s and t are included in any node $\mathbf{v} \in \mathbf{T}$ and each edge is assigned to a unique $C(\mathbf{v})$. The label of a node $\mathbf{v}$ is a vector of $|C(\mathbf{v})| + |X(\mathbf{v})| \cdot (|X(\mathbf{v})| - 1)$ dimensions, where the first $|C(\mathbf{v})|$ dimensions correspond to the edges covered by it and the last $|X(\mathbf{v})| \cdot (|X(\mathbf{v})| - 1)$ dimensions correspond to all the vertex pairs in $X(\mathbf{v})$. The intuition is the following. To ensure that a feasible label assignment can be translated to an s-t path, we first need to assign a "choosing indicator" to each edge to imply whether the edge is selected or not. However, having the choosing indicators is not enough because we only know some edges have been picked, but cannot determine whether s and t are connected. Thus, we introduce a "connectivity indicator" for each vertex pair (a, b) in $X(\mathbf{v})$ to indicate whether a is connected to b by the selected edges covered in the subtree rooted at node $\mathbf{v}$.

Constraint Construction. The cost constraints are used to bound the total cost of the selected edges. They can be obtained easily by letting the normalized cost of the edges selected by each label assignment be the corresponding cost. For the consistency constraints, the purpose of designing them is to ensure that s is connected to t, and the connectivity indicators can truthfully reflect the connectivity of the subgraph formed by the selected edges. To achieve the former requirement, we define that a label assignment is feasible only if the connectivity indicator of (s,t) in the root $\mathbf{r}$ is $\mathtt{1}$; while for the latter requirement, the constraint construction is still natural, but we note that proving such "local[3]" constraints are able to capture the global connectivity is non-trivial. Consider an arbitrary node $\mathbf{u}$ and its two children $\mathbf{v}, \mathbf{w}$. We can construct a subgraph where the vertex set is $X(\mathbf{u}) \cup X(\mathbf{v}) \cup X(\mathbf{w})$. Add an edge (a,b) in the subgraph if the connectivity indicator of (a,b) is $\mathtt{1}$ in one of the two children or the choosing indicator of edge (a,b) in node $\mathbf{u}$ is $\mathtt{1}$. A label is feasible if the connectivity indicator of node $\mathbf{u}$ is consistent with the connectivity in this subgraph. Note that this subgraph may not contain all nodes that occur in the subtree rooted at $\mathbf{u}$. Thus, to prove the efficiency of these constraints, we further need to show that there does not exist a vertex pair in $\mathbf{u}$ that is connected by $\mathbf{u}$'s subtree but not connected in the above subgraph. We formally show this claim in the paper's full version. The proof heavily relies on the connectivity property of a tree decomposition. Briefly speaking, a graph's tree decomposition can guarantee that all nodes in the tree containing the same node in the original graph form a connected subtree. This nice property allows us to show that the connectivity between vertices can be continuously propagated between nodes in $\mathbf{T}$.

3.3 Tree-labeling Instance Construction

Given an arbitrary node $\mathbf{v} \in \mathbf{V}$, we also use $\mathbf{v}$ to denote the vertices included in node $\mathbf{v}$, i.e., $\mathbf{v} \subseteq V$. Let $E_{\mathbf{v}} \subseteq E$ be the set of edges such that, for any edge (a,b) in $E_{\mathbf{v}}$, node $\mathbf{v}$ is the highest node that contains (a,b). Note that an edge (a,b) may be included in more than one node in $\mathbf{T}$ but the highest node that includes (a,b) is unique. For any node $\mathbf{v} \in \mathbf{V}$, we have two types of labels: choosing label and connectivity label. For each edge (a,b) (or e) in $E_{\mathbf{v}}$, the choosing label $\mathtt{chng}(a,b) = \mathtt{1}$ or (or $\mathtt{chng}(e) = \mathtt{1}$) indicates that the edge is chosen in current label assignment; otherwise, the edge is not chosen. For each pair of vertices (p,q) in $\mathbf{v}$, the connectivity label $\mathtt{conn}(p,q) = \mathtt{1}$ indicates that there is a $\mathcal{P} \subseteq E$ path from p to q such that every edge in $\mathcal{P}$ is chosen in some nodes of the subtree rooted at node $\mathbf{v}$; otherwise, p and q are not connected. Note that $\mathtt{conn}(p,q)$ and $\mathtt{conn}(q,p)$ are two different labels since G is a directed graph. Let $L_{\mathbf{v}}$ be the set of all possible labels and $l_{\mathbf{v}} \in L_{\mathbf{v}}$ be a specific label of node $\mathbf{v}$. We remark that the size of $L_{\mathbf{v}}$ is related to the treewidth of G. Since the treewidth of G is a constant, $|L_{\mathbf{v}}|$ is also constant. See the proof of Theorem 8 for details.

To ensure the feasibility of label assignments for obtaining an s-t path in $\mathbf{T}$, arbitrarily picking labels for each node is not a viable solution. Instead, we define a local constraint that applies to every adjacent set of three nodes $\mathbf{u}, \mathbf{v}, \mathbf{w}$ in $\mathbf{T}$, where $\mathbf{u}$ and $\mathbf{w}$ are the child nodes of $\mathbf{v}$. The purpose of this local constraint is to guarantee that all label assignments are capable of producing an s-t path. For every node $\mathbf{v}$ and its two children $\mathbf{u}$ and $\mathbf{w}$, let $\mathtt{CP}(\mathbf{v}) := L_{\mathbf{u}} \times L_{\mathbf{v}} \times L_{\mathbf{w}}$ be the set of all possible label combinations of these three nodes, i.e., $\mathtt{CP}(\mathbf{v})$ is the Cartesian product of $L_{\mathbf{u}}, L_{\mathbf{v}}, L_{\mathbf{w}}$. Note that $\mathbf{u}$ or $\mathbf{w}$ may not exist. In this case, we refer to $\mathbf{u}$ or $\mathbf{w}$ as empty nodes and $\mathtt{CP}(\mathbf{v})$ is defined as the $L_{\mathbf{v}} \times L_{\mathbf{w}}$ (or $L_{\mathbf{u}} \times L_{\mathbf{v}}$ or $L_{\mathbf{v}}$).

[3] Consistency constraints in the tree-labeling problem are "local" constraints because the feasibility of a node $\mathbf{u}$'s label is only influenced by its child nodes.

▶ **Definition 10** (Feasible Label Assignment). *A label assignment $\mathcal{L} := (l_{\mathbf{v}})_{\mathbf{v}\in\mathbf{V}}$ is a feasible label assignment if, for each $\mathbf{v} \in \mathbf{V}$ and its two children $\mathbf{u}$ and $\mathbf{w}$ ($\mathbf{u}$ and $\mathbf{w}$ maybe empty nodes), $l_{\mathbf{v}} \in \mathtt{CP}(\mathbf{v})$ satisfies the following three constraints:*

(C1) **(Choosing Constraints)** *For each edge $(a, b) \in E_{\mathbf{v}}$, if (a, b) is chosen, then a and b are connected and vice versa. Namely, $\mathtt{chng}(a, b) = \mathtt{1}$ if and only if $\mathtt{conn}(a, b) = \mathtt{1}$.*

(C2) **(Connectivity Constraints)** *For each vertex pair (p, q) in $\mathbf{v}$, vertex p and vertex q are connected (i.e. $\mathtt{conn}(p, q) = \mathtt{1}$) if and only if the following statement is true: there is a vertex sequence $(p, v_1, \ldots, v_d, q)$ such that every two adjacent vertices (a, b) in the sequence are connected in some nodes in $\mathbf{u}, \mathbf{v}, \mathbf{w}$, i.e., $\mathtt{conn}(a, b) = \mathtt{1}$ in some nodes in $\mathbf{u}, \mathbf{v}, \mathbf{w}$ for each pair of adjacent vertices.*

(C3) **(Feasibility Constraints)** *If $\mathbf{v}$ is the root of $\mathbf{T}$, then the source s and sink t are connected, i.e., $\mathtt{conn}(s, t) = \mathtt{1}$ must be true in the root.*

Given a feasible label assignment $\mathcal{L}$, an edge $(p, q) \in E$ is *chosen* by $\mathcal{L}$ if (p, q)'s choosing label is $\mathtt{1}$ in $\mathcal{L}$. (C1) defines the connectivity of each edge in E. If an edge (a, b) is chosen by a label assignment, then vertex a and b are connected. (C2) is the most important constraint which defines the connectivity of each pair of vertices in $\mathbf{v}$. In the case where $\mathbf{v}$ is a leaf node, $\mathbf{u}$ and $\mathbf{w}$ would be empty nodes and thus this constraint is equivalent to (C1). In the case where $\mathbf{v}$ is not a leaf node, an arbitrary vertices pair (a, b) in $\mathbf{v}$ are connected if the connected segments in $\mathbf{u}, \mathbf{v}, \mathbf{w}$ can be merged into a path from a to b. In Lemma 11, we show that such a local constraint is sufficient to describe the connectivity of vertex a and b in the subtree rooted at $\mathbf{v}$. (C3) ensures that a feasible label assignment must contain an s-t path, i.e., source s and sink t are connected.

Consider an arbitrary feasible label assignment $\mathcal{L}$, then $\mathcal{L}$ has the following crucial property (Lemma 11) by our definition. We shall use this property later to show that any feasible label assignment can be converted into an s-t path of the original graph. It is worth noting that an s-t path corresponds to a unique feasible label assignment, but a feasible label assignment may contain multiple s-t paths. An example is shown in Figure 2.

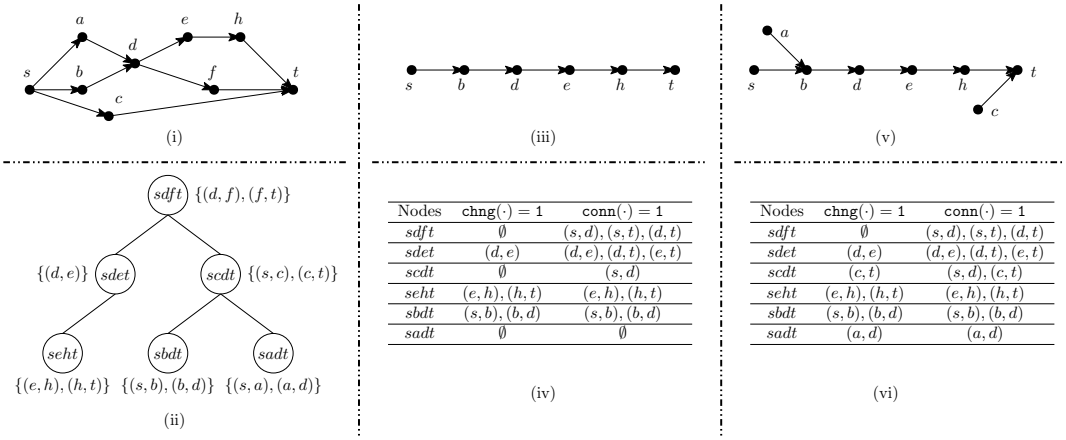
▶ **Lemma 11.** *Given an arbitrary feasible label assignment $\mathcal{L} := (l_{\mathbf{v}})_{\mathbf{v}\in\mathbf{V}}$, consider an arbitrary node $\mathbf{v} \in \mathbf{V}$. For any vertices pair (p, q) in $\mathbf{v}$, there is a path $\mathcal{P} \subseteq E$ from p to q such that every edge in $\mathcal{P}$ is chosen in some nodes in the subtree rooted at $\mathbf{v}$ if and only if (a, b) has a connectivity label of $\mathtt{1}$ in $\mathbf{v}$.*

To complete the instance construction, we also need to define an appropriate label cost function for each node in $\mathbf{T}$. It shall be used to connect the cost of our problem to the tree-labeling problem. This part, together with the proofs of Lemma 11 and Theorem 8, are deferred to the paper's full version.

4 General Graphs

We shall follow the same algorithmic framework stated in Section 2 and show the following result (Theorem 12). Namely, we first construct a tree structure and set up a linear program based on the tree. Then, we employ the same rounding algorithm (Algorithm 2) to obtain a feasible subtree and convert it back to an s-t path in the original graph. The following states some intuition to construct such a tree. We defer the formal descriptions of the construction, the LP formulation, the algorithm, and the analysis to the full version of the paper.

▶ **Theorem 12.** *Given any directed graph G, there is an algorithm that returns a $O(\log n \log k)$-approximate solution in $poly(n) \cdot n^{O(\log n)}$ time with probability at least $1 - (\frac{1}{k} + \frac{1}{k \log n})$ for robust s-t path, where n is the number of vertices and k is the number of agents.*



Nodes	chng(·) = 1	conn(·) = 1
sdft	$\emptyset$	$(s,d),(s,t),(d,t)$
sdet	(d,e)	$(d,e),(d,t),(e,t)$
scdt	$\emptyset$	(s,d)
seht	$(e,h),(h,t)$	$(e,h),(h,t)$
sbdt	$(s,b),(b,d)$	$(s,b),(b,d)$
sadt	$\emptyset$	$\emptyset$

Nodes	chng(·) = 1	conn(·) = 1
sdft	$\emptyset$	$(s,d),(s,t),(d,t)$
sdet	(d,e)	$(d,e),(d,t),(e,t)$
scdt	(c,t)	$(s,d),(c,t)$
seht	$(e,h),(h,t)$	$(e,h),(h,t)$
sbdt	$(s,b),(b,d)$	$(s,b),(b,d)$
sadt	(a,d)	(a,d)

Figure 2 An example of the reduction. The subfigure (i) is the given directed graph. Then, we compute a tree decomposition with the logarithmic depth and add s and t to all nodes, which is shown in subfigure (ii). The edge set next to each node in subfigure (ii) is its corresponding $E_{\mathbf{v}}$. For example, the edge set $E_{\mathbf{r}}$ for root node $\mathbf{r}$ is $\{(d,f),(f,t)\}$ since $\mathbf{r}$ is the highest node that contains edge (d,f) and (f,t). Subfigure (iii) is an s-t path of the given directed graph and subfigure (iv) is the corresponding label assignment of the s-t path in (iii) in which we only list these labels with the value of `1`. The complete label of each node is obtained by merging these single labels, e.g., for the root $\mathbf{r}$, $l_{\mathbf{r}}$ consists of 14 bits (2 choosing labels and 12 connectivity labels). In these 14 bits, only $\texttt{conn}(s,d), \texttt{conn}(s,t), \texttt{conn}(d,t)$ has a value of `1` and all the remaining 11 labels have a value of `0`. Subfigure (v) shows another example and subfigure (vi) is its corresponding label assignment.

In Section 2, we use a tree-based linear program to break through the $\Omega(k)$ and $\Omega(\sqrt{n})$ integrality gap of the flow LP. However, on general graphs, we no longer have such a natural tree structure as in series-parallel graphs.To still obtain a polylogarithmic approximation, we construct a decision-tree-type metatree that maps every s-t path in the graph to a corresponding subtree in the metatree.

Metatree Construction Intuition. The basic idea of the meta tree construction is to iteratively guess the possible middle vertex of an s-t path. There are at most n possibilities for this middle vertex. Once we determine the middle vertex of the s-t path, say it is a, the whole path can be partitioned into two subpaths – path s-a and path a-t. We then recur on the obtained subpaths till level $O(\log n)$; the sufficiency of $O(\log n)$'s levels will be clear later. This process gives us a natural tree structure $\mathbf{T}$ with $O(\log n)$ depth. We define two types of nodes in $\mathbf{T}$. The first node type is referred to as *splitting node*. Each splitting node corresponds to a (sub-)path. It has n children, where each child represents a choice of the (sub-)path's middle vertex. The algorithm needs to ensure that only one of these children can be selected. The second node type is called *merging node*. A merging node has to be a child of a splitting node in $\mathbf{T}$ and represents a scheme for selecting a middle vertex. Further, a merging node has two splitting nodes as its children, corresponding to the two obtained subpaths by this scheme. We can view such a node as being used to merge its two children (subpaths). The algorithm needs to ensure that both children are selected simultaneously. See the paper's full version for more details of the construction.

As one may observe, a splitting (resp. merging) node in our metatree plays the same role as a parallel (resp. series) node in the decomposition tree of the series-parallel graph. Thus, the LP for the general graph is similar to (Tree-LP) and we can still use the same rounding algorithm. Consider an s-t path $\mathcal{P}$ of length n. If we write $\mathcal{P}$ in the form of a binary tree by

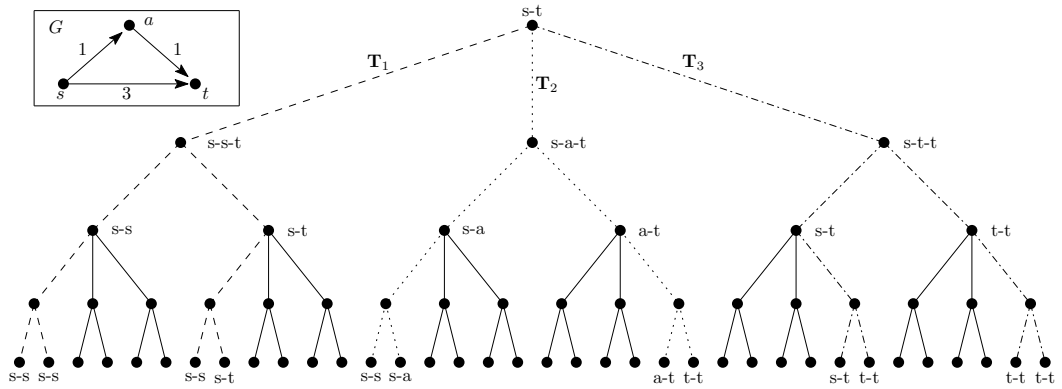


Figure 3 An example for the metatree $\mathbf{T}$ construction. The given directed graph G is shown in the up-left corner. Since there are three vertices in G, $\mathbf{T}$ will consist of five levels because the height of $\mathbf{T}$ is $2\lceil \log n \rceil + 1$. The dashed subtree $\mathbf{T}_1$ corresponds to the path $s \to t$ of G. The dotted subtree $\mathbf{T}_2$ represents the path $s \to a \to t$ of G. The dash-dotted subtree $\mathbf{T}_3$ also corresponds to the path $s \to t$.

guessing the middle vertex of each subpath, it will have at most $\lceil \log n \rceil$ levels. Thus, we can let the recursive tree $\mathbf{T}$ terminate at level $O(\log n)$, and therefore, its size is quasi-polynomial $O(n^{\log n})$ since each node has at most n children. From Section 2, we know that the rounding algorithm is able to find a $O(H \log k)$-approximate solution where H is the height of the tree. This also provides the intuition for the approximation ratio $O(\log n \log k)$ since the height of $\mathbf{T}$ is $O(\log n)$ (specifically, $H = 2\lceil \log n \rceil + 1$).

We now define a *feasible subtree* for $\mathbf{T}$. Recalling the feasible subtrees (Definition 3) on series-parallel graphs, unfortunately, we cannot use the same definition for general graphs. This is because not all leaf nodes in the constructed tree $\mathbf{T}$ correspond to edges in G. We refer to a subtree that satisfies three conditions in Definition 3 as a *consistent subtree*. To ensure that the subtree can be translated to an s-t path, one more condition is needed.

▶ **Definition 13** (Feasible Subtree for General Graphs). *A subtree $\mathbf{T}' \subseteq \mathbf{T}$ is called feasible if and only if (i) $\mathbf{T}'$ is consistent; (ii) each leaf node corresponds to either an edge or a single vertex in G.*

An example can be found in Figure 3. As one might be concerned, using a different definition of the feasible subtree may require a different LP formulation for general graphs, since a natural adaptation of (Tree-LP) can only find a consistent subtree. This issue can be fixed easily by directly disabling the infeasible leaf nodes in the linear program. See the paper's full version for more details of the LP formulation.

5 Conclusion

This paper considers the robust s-t path problem and proposes polylogarithmic approximation algorithms on different graph classes. For graphs with bounded treewidth, we obtain a $O(\log n \log k)$-approximate polynomial algorithm which partially answers the open question in [23]. For general graphs, we prove that there is a quasipolynomial algorithm that is $O(\log n \log k)$-approximate which leaves a logarithmic gap. Our approaches are based on a novel linear program that enables us to get rid of the $\Omega(k)$ and $\Omega(\sqrt{n})$ integrality gap from the natural linear program. We also investigate the robustness of the s-t path, weighted independent set, and spanning tree under the maximin criteria and show some hardness results.

There leave several future works. Closing the gap for robust s-t path still remains open. It is thus interesting to investigate whether there exists a better approximation upper bound or a tighter lower bound. We can also look at other robust optimization problems, e.g., the robust perfect matching problem. The best approximation known to date for robust matching is still $O(k)$ which can be achieved by a trivial algorithm. Since there exists a strong connection between s-t path and min-cost perfect matching, it would be intriguing to explore whether our methods can be applied to improving the upper bound of the robust matching problem.

References

1 Jacob D. Abernethy, Pranjal Awasthi, Matthaus Kleindessner, Jamie Morgenstern, Chris Russell, and Jie Zhang. Active sampling for min-max fairness. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 53–65. PMLR, 2022.

2 Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Approximation of min-max and min-max regret versions of some combinatorial optimization problems. *Eur. J. Oper. Res.*, 179(2):281–290, 2007.

3 Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Min–max and min–max regret versions of combinatorial optimization problems: A survey. *European journal of operational research*, 197(2):427–438, 2009.

4 Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. General approximation schemes for min-max (regret) versions of some (pseudo-)polynomial problems. *Discret. Optim.*, 7(3):136–148, 2010.

5 Yang An and Rui Gao. Generalization bounds for (wasserstein) robust optimization. In *NeurIPS*, pages 10382–10392, 2021.

6 Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust Optimization*, volume 28 of *Princeton Series in Applied Mathematics*. Princeton University Press, 2009.

7 Vittorio Bilò, Ioannis Caragiannis, Angelo Fanelli, Michele Flammini, and Gianpiero Monaco. Simple greedy algorithms for fundamental multidimensional graph problems. In *ICALP*, volume 80 of *LIPIcs*, pages 125:1–125:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

8 Hans L. Bodlaender. Nc-algorithms for graphs with small treewidth. In *WG*, volume 344 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 1988.

9 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *FOCS*, pages 575–584. IEEE Computer Society, 2010.

10 Qingyun Chen, Sungjin Im, Benjamin Moseley, Chenyang Xu, and Ruilong Zhang. Min-max submodular ranking for multiple agents. *AAAI 2023, to appear*, 2023.

11 Altannar Chinchuluun and Panos M. Pardalos. A survey of recent developments in multiobjective optimization. *Ann. Oper. Res.*, 154(1):29–50, 2007.

12 Vincent Conitzer, Rupert Freeman, and Nisarg Shah. Fair public decision making. In *EC*, pages 629–646. ACM, 2017.

13 Michael Dinitz, Guy Kortsarz, and Shi Li. Degrees and network design: New problems and approximations. *CoRR*, abs/2302.11475, 2023. `arXiv:2302.11475`.

14 Daniel Duque, Leonardo Lozano, and Andrés L. Medaglia. An exact method for the biobjective shortest path problem for large-scale road networks. *Eur. J. Oper. Res.*, 242(3):788–797, 2015.

15 Matthias Ehrgott. *Multicriteria Optimization (2. ed.)*. Springer, 2005.

16 Brandon Fain, Kamesh Munagala, and Nisarg Shah. Fair allocation of indivisible public goods. In *EC*, pages 575–592. ACM, 2018.

17 Virginie Gabrel, Cécile Murat, and Aurélie Thiele. Recent advances in robust optimization: An overview. *Eur. J. Oper. Res.*, 235(3):471–483, 2014.

18 Yu Gang and Yang Jian. On the robust shortest path problem. *Comput. Oper. Res.*, 25(6):457–468, 1998.

19 Fabrizio Grandoni, Bundit Laekhanukit, and Shi Li. $O(\log^2 k \,/\, \log\log k)$-approximation algorithm for directed steiner tree: a tight quasi-polynomial-time algorithm. In *STOC*, pages 253–264. ACM, 2019.

20 Adam Kasperski and Pawel Zielinski. On the approximability of minmax (regret) network optimization problems. *Inf. Process. Lett.*, 109(5):262–266, 2009.

21 Adam Kasperski and Pawel Zielinski. On the approximability of robust spanning tree problems. *Theor. Comput. Sci.*, 412(4-5):365–374, 2011.

22 Adam Kasperski and Paweł Zieliński. Robust discrete optimization under discrete and interval uncertainty: A survey. *Robustness analysis in decision aiding, optimization, and analytics*, pages 113–143, 2016.

23 Adam Kasperski and Pawel Zielinski. Approximating some network problems with scenarios. *CoRR*, abs/1806.08936, 2018. `arXiv:1806.08936`.

24 Ana Klobucar and Robert Manger. Solving robust weighted independent set problems on trees and under interval uncertainty. *Symmetry*, 13(12):2259, 2021.

25 Shi Li, Chenyang Xu, and Ruilong Zhang. Polylogarithmic approximation for robust s-t path. *CoRR*, abs/2305.16439, 2023. `arXiv:2305.16439`.

26 Xian Li and Hongyu Gong. Robust optimization for multilingual translation with imbalanced data. In *NeurIPS*, pages 25086–25099, 2021.

27 Natalia Martínez, Martín Bertrán, and Guillermo Sapiro. Minimax pareto fairness: A multi objective perspective. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 6755–6764. PMLR, 2020.

28 Fabrice Talla Nobibon and Roel Leus. Robust maximum weighted independent-set problems on interval graphs. *Optim. Lett.*, 8(1):227–235, 2014.

29 Christos H. Papadimitriou and Mihalis Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *FOCS*, pages 86–92. IEEE Computer Society, 2000.

30 Bozidar Radunovic and Jean-Yves Le Boudec. A unified framework for max-min and min-max fairness with applications. *IEEE/ACM Trans. Netw.*, 15(5):1073–1083, 2007.

31 Antonio Sedeño-Noda and Marcos Colebrook. A biobjective dijkstra algorithm. *Eur. J. Oper. Res.*, 276(1):106–118, 2019.

32 Paolo Serafini. Some considerations about computational complexity for multi objective combinatorial problems. In *Recent Advances and Historical Development of Vector Optimization: Proceedings of an International Conference on Vector Optimization*, pages 222–232. Springer, 1987.

33 Jacobo Valdes, Robert Endre Tarjan, and Eugene L. Lawler. The recognition of series parallel digraphs. *SIAM J. Comput.*, 11(2):298–313, 1982.

Improved Lower Bounds for Approximating Parameterized Nearest Codeword and Related Problems Under ETH

Shuangle Li
State Key Laboratory for Novel Software Technology, Nanjing University, China

Bingkai Lin
State Key Laboratory for Novel Software Technology, Nanjing University, China

Yuwei Liu
BASICS, Shanghai Jiao Tong University, China

Abstract

In this paper we present a new gap-creating randomized self-reduction for the parameterized Maximum Likelihood Decoding problem over $\mathbb{F}_p$ (k-MLD$_p$). The reduction takes a k-MLD$_p$ instance with $k \cdot n$ d-dimensional vectors as input, runs in $O(d2^{O(k)}n^{1.01})$ time for some computable function f, outputs a $(3/2-\varepsilon)$-Gap-k'-MLD$_p$ instance for any $\varepsilon > 0$, where $k' = O(k^2 \log k)$. Using this reduction, we show that assuming the randomized Exponential Time Hypothesis (ETH), no algorithms can approximate k-MLD$_p$ (and therefore its dual problem k-NCP$_p$) within factor $(3/2-\varepsilon)$ in $f(k) \cdot n^{o(\sqrt{k/\log k})}$ time for any $\varepsilon > 0$.

We then use reduction by Bhattacharyya, Ghoshal, Karthik and Manurangsi (ICALP 2018) to amplify the $(3/2-\varepsilon)$-gap to any constant. As a result, we show that assuming ETH, no algorithms can approximate k-NCP$_p$ and k-MDP$_p$ within γ-factor in $f(k) \cdot n^{o(k^{\varepsilon_\gamma})}$ time for some constant $\varepsilon_\gamma > 0$. Combining with the gap-preserving reduction by Bennett, Cheraghchi, Guruswami and Ribeiro (STOC 2023), we also obtain similar lower bounds for k-MDP$_p$, k-CVP$_p$ and k-SVP$_p$.

These results improve upon the previous $f(k) \cdot n^{\Omega(\text{poly} \log k)}$ lower bounds for these problems under ETH using reductions by Bhattacharyya et al. (J.ACM 2021) and Bennett et al. (STOC 2023).

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Error-correcting codes; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Nearest Codeword Problem, Hardness of Approximations, Fine-grained Complexity, Parameterized Complexity, Minimum Distance Problem, Shortest Vector Problem

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.107

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2402.09825

Acknowledgements We thank the anonymous reviewers for their detailed comments.

1 Introduction

The study of linear error correcting codes has drawn attention to two dual fundamental computational problems called Nearest Codeword Problem (NCP) and Maximum Likelihood Decoding (MLD). Given a matrix $A \in \mathbb{F}_p^{m \times n}$ and a vector $\vec{t} \in \mathbb{F}_p^m$, the Nearest Codeword Problem (NCP) asks for a vector $\vec{x} \in \mathbb{F}_p^n$ such that $||A\vec{x} - \vec{t}||_0$ is minimized. Here $|| \cdot ||_0$ denotes the Hamming weight. While in the Maximum Likelihood Decoding (MLD), we are given a matrix $A \in \mathbb{F}_p^{m \times n}$ and a vector $\vec{t} \in \mathbb{F}_p^m$, the goal is to minimize $||\vec{x}||_0$ subject to $A\vec{x} = \vec{t}$. Another fundamental problem related to a linear code is the homogeneous version of NCP, known as Minimum Distance Problem (MDP), where the task is to find a non-zero vector $\vec{x}$ such that $||A\vec{x}||_0$ is minimized.

The computational complexity of MLD, NCP and MDP has been studied with great effort throughout the past several decades. It is known that MLD, NCP and MDP are not only NP-hard [12, 48], but also NP-hard to approximate within any constant ratio [6, 7, 18, 22, 42, 47]. Moreover, the variant of MLD that allows the code being preprocessed by unbounded computational resource is also NP-hard to approximate within a factor of $(3 - \varepsilon)$ [24, 44]. Also it is proven that assuming $\mathsf{NP} \not\subseteq \mathsf{DTIME}(n^{\mathsf{poly}(\log n)})$, no polynomial time algorithm can approximate NCP up to $2^{\log^{1-\epsilon} n}$ factor for any $\epsilon > 0$ [6, 43] and no polynomial time algorithm can approximate MDP up to $2^{\log^{1-\epsilon} n}$ for any $\epsilon > 0$ [7, 18, 22, 42]. For some specific codes, MLD is also shown to be NP-hard, e.g. product code [8], Reed-Solomon code [28], algebraic geometry code [17]. On the algorithmic side, it is known that NCP can be approximate to $O(n/\log n)$ in polynomial time [5].

The lattice version of NCP and MDP are known as Closest Vector Problem (CVP) and Shortest Vector Problem (SVP). In these problems, a lattice $\mathcal{L}$ is given instead of a linear code. For CVP a target $\vec{t}$ is additionally given and the goal is to find a vector $\vec{v} \in \mathcal{L}$ such that $||\vec{v} - \vec{t}||_p$ is minimized, where $||\cdot||_p$ denotes the ℓ_p-norm. And for SVP the goal is to find a non-zero vector $\vec{v} \in \mathcal{L}$ with minimum ℓ_p norm. The study for CVP and SVP also has long history [4, 6, 20, 25, 29, 33, 40–42, 47]. For CVP, it is NP-hard to approximate within factor $n^{c/\log\log n}$ for some constant $c > 0$ [20]. As for SVP, it was shown that no polynomial time algorithm can approximate SVP within any constant factor assuming $\mathsf{NP} \not\subseteq \mathsf{RP}$ [33], and no polynomial time algorithm can approximate SVP up to $2^{\log^{1-\epsilon} n}$ factor assuming $\mathsf{NP} \not\subseteq \mathsf{RTIME}(n^{\mathsf{poly}(\log n)})$ [29]. Lattice problems have many applications in cryptography [45, 46]. Due to their importance, lattice problems are also extensively studied in the fine-grained complexity area, see, e.g., [1–3, 11] and a very recent survey by Bennett [9] for more details on hardness of SVP.

Over the past three decades, parameterized complexity, a new framework to address NP-hard problems, has been rapidly developed and drawing growing attention. The study in the field of parameterized complexity focuses on whether a problem can be solved in $f(k) \cdot n^{O(1)}$ time (FPT time), where k is a parameter given along with the instance. In the parameterized version of k-MLD, k-NCP, k-MDP, k-CVP and k-SVP, an integer k is additionally given and the task is to decide whether the optimal value is no greater than k. Downey, Fellows, Vardy and Whittle [21] showed that k-MLD (and therefore k-NCP) is W[1]-hard and belongs to W[2]. They asked if k-CVP and k-SVP (in ℓ_2 norm) is W[1]-hard. 20 years later in recent breakthroughs [10, 13], the parameterized intractability of k-NCP, k-MDP, k-CVP and k-SVP are settled. Notably they ruled out not only exact FPT algorithms, but also FPT approximation algorithms as well. Specifically, [13] first presented a gap-creating reduction for k-NCP and then showed gap-preserving reductions from k-NCP towards k-MDP, k-CVP and k-SVP. Soon afterwards, Bennett, Cheraghchi, Guruswami and Ribeiro [10] improved the gap-preserving reductions for more general cases (general fields and general ℓ_p norm). These two works jointly showed that it is W[1]-hard to approximate k-NCP and k-MDP within any constant factor over any finite field $\mathbb{F}_p$, and it is W[1]-hard to approximate k-CVP in the ℓ_p norm within any constant factor for any $p \geq 1$. And they showed hardness for k-SVP to approximate within any constant factor in the ℓ_p norm for any $p > 1$, and some constant approaching 2 for $p = 1$.

After obtaining FPT-inapproximability results, it is natural to study fine-grained time lower bounds for parameterized approximability of these problems. Assuming Gap-ETH [19, 39], Manurangsi [38] showed that no $f(k) \cdot n^{o(k)}$ time algorithm can approximate k-NCP and k-CVP to any constant factor. With the gap-preserving reduction in [10], one can further show that no $f(k) \cdot n^{o(k)}$ time algorithm can approximate k-MDP and k-SVP to

any constant under the randomized Gap-ETH. All these results are based on an assumption with a gap. This raises the following open question:

(1) Can we establish similar lower bounds for these problems under the weaker and gap-free assumption of ETH?

We note that the gap-preserving reduction in [10] from Gap-k-NCP (Gap-k-CVP) to Gap-k'-MDP (Gap-k'-SVP) has $k' = O(k)$. So, it suffices to prove constant Gap-k-NCP (Gap-k-CVP) has no $f(k) \cdot n^{o(k)}$-time algorithm assuming ETH [30]. Unfortunately, the gap-creating reduction in [13] causes an exponential growth of the parameter and only gives an $\Omega(n^{(\log k)^{1/(2+\epsilon)}})$-time lower bound for constant Gap-k-NCP under ETH (See the analysis in Section 1.3). Therefore, finding better reductions for Gap-k-NCP and Gap-k-CVP is the crux of improving lower bounds for Gap-k-MDP and Gap-k-SVP.

1.1 Our Contributions

We take a step forward on closing the gap between results under gap-free assumption (ETH) and gap assumption (Gap-ETH). Our main result is a new direct gap-creating self reduction for k-MLD, which is the dual problem of k-NCP, with polynomial growth of the parameter.

▶ **Theorem 1** (informal; See Theorem 20 for a formal statement)**.** *For any constant $1 < \gamma < \frac{3}{2}$ and prime power $p > 1$, there is a reduction runs in $O_k(n^{O(1)})$ that on input a k-MLD$_p$ instance $(V, \vec{t})$, output a Gap-k'-MLD$_p$ instance $(V', \vec{t}')$ satisfies:*

- *(Completeness) If there exists k vectors in V with their sum [1] being $\vec{t}$, then there exists k' vectors in V' with their sum being $\vec{t}'$.*
- *(Soundness) If for any set $S \subseteq V$ with size at most k, $\vec{t} \notin Span(S)$, then for any set $S' \subseteq V'$ with size at most $\gamma k'$, $\vec{t}' \notin Span(S')$.*
- *Polynomial parameter growth $k' = O(k^2 \log k)$. (And $k' = O(k^3)$ if not allowing randomness).*

Combining this gap-creating reduction with the $f(k)n^{\Omega(k)}$-time ETH lower bound for k-MLD in [36, Theorem 11], we obtain improved lower bounds for Gap-k-NCP assuming ETH and randomized ETH.

▶ **Corollary 2.** *Assuming randomized ETH, for any prime power $p > 1$ and real number $\gamma \in (1, \frac{3}{2})$, no $f(k)n^{o(\sqrt{k/\log k})}$ time algorithm can solve γ-Gap-k-NCP$_p$.*

▶ **Corollary 3.** *Assuming ETH, for any prime power $p > 1$ and real number $\gamma \in (1, \frac{3}{2})$, no $f(k)n^{o(k^{1/3})}$ time algorithm can solve γ-Gap-k-NCP$_p$.*

By applying the gap amplification procedure in [14] ($\gamma \to \Omega(\gamma^2), k \to O(k^2)$, see Theorem 22 for a formal statement) sufficiently many (but still constant) times, we obtain a reduction for Gap-k-MLD with any constant gap with still polynomial growth of parameter. Therefore we obtain the following improved ETH lower bound for k-NCP.

▶ **Corollary 4.** *Assuming ETH, for any prime power $p > 1$ and real number $\gamma > 1$, no $f(k)n^{o(k^\epsilon)}$ time algorithm can solve γ-Gap-k-NCP$_p$ where $\epsilon = \frac{1}{\text{polylog}(\gamma)}$ is a constant.*

Combining our results of Gap-k-NCP$_p$ with the gap-preserving reductions in [13] and [10], we obtain improved ETH lower bounds for constant approximating k-NCP, k-CVP, k-MDP and k-SVP. The summarize of corollaries are present in Table 1.

[1] The definition of k-MLD used in our proof is a slightly different variant, where the vectors directly sum to the target in the YES case, but they are essentially equivalent, see Section 2.3 for more details.

Table 1 The $f(k)n^{\Omega(k^\epsilon)}$-time lower bound for k-NCP and k-CVP are based on ETH. The other lower bounds are based on randomized ETH.

Summarize of Corollaries

Problem	Inapprox Factor	Lower Bound	Dependency	Specification
k-NCP	any $\gamma \in (1, \frac{3}{2})$	$f(k)n^{\Omega(\sqrt{k/\log k})}$		any finite field $\mathbb{F}_p$
k-NCP	any $\gamma > 1$	$f(k)n^{\Omega(k^\epsilon)}$	$\epsilon = \frac{1}{\mathsf{polylog}(\gamma)}$	any finite field $\mathbb{F}_p$
k-MDP	any $\gamma > 1$	$f(k)n^{\Omega(k^\epsilon)}$	$\epsilon = \frac{1}{p\log\gamma\cdot\mathsf{polylog}(p)}$	any finite field $\mathbb{F}_p$
k-CVP	any $\gamma > 1$	$f(k)n^{\Omega(k^\epsilon)}$	$\epsilon = \Theta(\frac{1}{\mathsf{polylog}(\gamma)})$	in any ℓ_p norm, $p \geq 1$
k-SVP	any $\gamma > 1$	$f(k)n^{\Omega(k^\epsilon)}$	$\epsilon = \epsilon(p,\gamma)^2$	in any ℓ_p norm, $p > 1$
k-SVP	any $\gamma \in [1, 2)$	$f(k)n^{\Omega(k^\epsilon)}$	$\epsilon = \epsilon(p,\gamma)^3$	in any ℓ_p norm, $p \geq 1$

1.2 Technical Overview of Gap Creation Step

We implicitly use the threshold graph composition method [15, 34, 35, 37] to construct a $(3/2-\varepsilon)$-gap producing reduction for the k-MLD problem. This technique was first introduced in [34] to prove the W[1]-hardness of k-Biclique problem. A threshold graph is a bipartite graph that has a "threshold property", meaning that there is a significant gap in the number of common neighbors between any k vertices and any $k+1$ vertices on the left side. Threshold graph and its variants have been widely used to show hardness of approximation for various parameterized problems, such as k-DominatingSet [16], k-SetCover [32, 35], k-SetIntersection [15] or to create gap for subsequent reductions, e.g. [13].

Let $\dot\cup$ denotes for union set of multiple disjoint sets. In this paper, we implicitly use the strong threshold graphs in [37], which are bipartite graphs $T = (A \dot\cup B, E_T)$ with the following properties:

- **(i)** $A = A_1 \dot\cup A_2 \dot\cup \cdots \dot\cup A_k$.
- **(ii)** $B = B_1 \dot\cup B_2 \dot\cup \cdots \dot\cup B_m$.
- **(iii)** For any $a_1 \in A_1, \ldots, a_k \in A_k$ and $i \in [m]$, $a_1, \ldots, a_k$ have a common neighbor in B_i.
- **(iv)** For any $X \subseteq A$ and $I \subseteq [m]$ with $|I| \geq \varepsilon m$, if for every $i \in I$, there exists $b_i \in B_i$ has $k+1$ neighbors in X, then $|X| > h$.

These strong threshold graphs are constructed from error-correcting codes with large relative distance $(1-o(1))$, and such "threshold" properties essentially come from the following intuition of ECC: *If there is a collection of codewords* (X)*, and a constant fraction of entries of these codewords* $(I \subseteq [m], |I| \geq \varepsilon m)$ *such that, for each entry* $(i \in I)$*, there exists two distinct codewords in the collection having same content in it. Then, the collection must have huge size* (*at least* h). To characterize the aforementioned property, previous works [32, 37] introduced the definition of (ε-)*Collision Number* of an error-correcting code C, $\mathrm{Col}_\varepsilon(C)$, which is the minimum size of X mentioned above.

Diving into coding-based threshold graph. Our construction deeply relies on the collision number of an ECC, so we only use threshold graph as an intuitive illustration for readers, and we directly use the error-correcting codes in our formal analysis.

Below we illustrate the idea of our reduction. For simplicity, here we consider k-MLD problem on d-length vectors over binary field. Given k vectors sets $V_1, \ldots, V_k \subseteq \mathbb{F}_2^d$, a target vector $\vec{t}$ and a strong threshold graph $T = (A \dot\cup B, E_T)$, we first identify V_i with A_i for every

$i \in [k]$. Our goal is to construct a one-to-one mapping $f : A \cup B \to \mathbb{F}_2^D$ and a new target vector $\vec{t}' \in \mathbb{F}_2^D$ for some $D = \mathsf{poly}(d, k)$ such that in order to pick vectors from $f(A \cup B)$[4] with their sum being $\vec{t}'$, one has to pick a set $f(X)$ of vectors from $f(A)$ for some $X \subseteq A$ with $\sum_{\vec{a} \in X} \vec{a} = \vec{t}$ and a set $f(Y)$ of vectors from $f(B)$ for some $Y \subseteq B$ such that for every $i \in [m]$,

(a) either $|Y \cap B_i| \geq 2$,

(b) or $|Y \cap B_i| = 1$ and there exists $b_i \in B_i$ with one of following properties:

- **(b.1)** $|X| = k$ and b_i is the common neighbors of vertices in X.
- **(b.2)** b_i has at least $k + 1$ neighbors in X.

Then we argue that these properties imply a constant gap between the solution sizes in the (YES) and (NO) cases of the k-MLD problem.

(YES) Suppose there are $a_1 \in A_1, \ldots, a_k \in A_k$ such that $\sum_{i \in [k]} a_i = \vec{t}$. By the property (iii) of threshold graphs, $a_1, \ldots, a_k$ have a common neighbor $b_i \in B_i$ for every $i \in [m]$. Then according to (b), the sum of $f(a_1), \ldots, f(a_k)$ and $f(b_1), \ldots, f(b_m)$ is $\vec{t}'$.

(NO) On the other hand, if there are no $a_1 \in A_1, \ldots, a_k \in A_k$ such that $\sum_{i \in [k]} a_i = \vec{t}$, then one should pick either at least $(1 - \varepsilon)2m$ vectors from $f(B)$ and $k + 1$ vectors from $f(A)$, or pick a subset of vectors $f(X)$ from $f(A)$ and a subset of vectors $f(Y)$ from $f(B)$ for some $Y \subseteq B$ with $|\{i \in [m] : |Y \cap B_i| = 1\}| \geq \varepsilon m$. Let $I = \{i \in [m] : |Y \cap B_i| = 1\}$. According to the property (b.2), each vertex in $Y \cap B_i$ ($i \in I$) has $k + 1$ neighbors in X. Since $|I| \geq \varepsilon m$, by the property (iv) of threshold graphs, we have that $|X| > h$. Thus, either $(1 - \varepsilon)2m$ vectors in $f(B)$ and $k + 1$ vectors in $f(A)$ or m vectors in $f(B)$ and h vectors in $f(A)$ must be picked in this case.

To obtain a constant gap, we duplicate each vector in $f(A)$ m/k times and let $h = ck$ where c is some constant to be chosen. In the (yes) case, there are $2m$ vectors with their sum being $\vec{t}'$. In the (no) case, no $\min\{2(1 - \varepsilon)m + m, m + cm\}$ vectors from $f(A \cup B)$ can have sum $\vec{t}'$.

The proof framework above has two problems to be solved.

(P1) How to combine the threshold graph and the k-MLD instance to produce vectors $f(A \cup B)$ with the properties (a) and (b)?

(P2) The smaller parameter blow-up we create in reduction, the tighter running time lower bound we obtain. So how to construct a threshold graph with $h > ck$ and m as small as possible?

Our approach to solve Problem (P1). Problem (P1) is related to the composition step in the threshold graph composition method. For the k-SetCover problem, we can use the hypercube partition system [23] to solve this problem. Unfortunately, this does not apply to the k-MLD problem. To solve problem (P1), we exploit an additional property from the construction of strong threshold graph using error correcting codes. More precisely, we can assume that there is a encoding function $C : A \to \Sigma^m$ and each $b_i \in B_i$ can be written as a k-tuple in $(b_{i,1}, \ldots, b_{i,k}) \in \Sigma^k$ such that b_i is adjacent to $a_j \in A_j$ in the threshold graph if and only if $b_{i,j} = C(a_j)[i]$. Informally speaking, we choose the target vector $\vec{t}'$ and the one-to-one mapping $f : A \cup B \to \mathbb{F}_2^D$ such that any subset of vectors in $f(A \cup B)$ summing up to $\vec{t}'$ must contains, for each $i \in [m]$, at least one vector $f(b_i)$ for some $b_i \in B_i$. And if it contains exactly only one such vector $f(b_i)$, then one need to pick at least k vectors $f(a_1) \in f(A_1), \ldots, f(a_k) \in f(A_k)$ to cancel out the parts corresponding to $b_{i,1}, \ldots, b_{i,k}$ in the vector $f(b_i)$. A careful analysis shows that this construction has the properties (a) and (b).

[4] Here we let $f(X)$ denote the set $\{f(x) : x \in X\}$.

Our approach to solve Problem (P2). The construction of strong threshold graph in [37] was based on the idea of Karthik and Navon [32]. Karthik and Navon [32] observed that the "collision number" of an error-correcting code can be directly used to show the threshold property. Intuitively speaking, a set C of strings with high ε-collision number indicates that if there is some mechanism forces us to choose some strings in C that collides on at least ε fraction of entries, then we must choose at least $\mathrm{Col}_\varepsilon(C)$ strings.

Known analysis of collision number in [10,32] starts from the distance of an error-correcting code. For a code with relative distance δ, previous analysis shows that its ε-collision number is $\mathrm{Col}_\varepsilon(C) = \sqrt{\frac{2\varepsilon}{1-\delta}}$. Note that $\delta = 1 - \Theta(\frac{r}{m})$ for Reed-Solomon codes used in the previous works. To obtain a gap, we require $\mathrm{Col}_\varepsilon(C) \geq \Theta(k)$, which leads to $m = \Omega(k^2)r$. In our reduction, we additionally require $\Sigma^r \geq n$ to fit the input size, which requires $r \geq \frac{\log n}{\log |\Sigma|}$, then we have $m \geq k^2 \log n / \log |\Sigma|$. On the other hand, our reduction needs to enumerate every k-tuples in Σ^k, concerning the running time we require $|\Sigma|^k \leq n^{O(1)}$. Putting all together, we must have $m \geq \Omega(k^3)$. In fact, we showed that the Singleton bound of codes implies such construction must have parameter growth $\Omega(k^3)$.

To obtain a better parameter, we find the analysis by Karthik and Navon [32, Section 3.1] can be modified to show better lower bound for the ε-collision number of a random code. Following their idea, we show a random code $C_R : \Sigma^r \to \Sigma^m$ with superconstant-sized alphabet and $m = \Omega(|\Sigma|^{1/3} \log |\Sigma| r)$ would have ε-collision number $\mathrm{Col}_\varepsilon(C_R) \geq |\Sigma|^{1/3}$, with high probability. Setting $|\Sigma| = \Theta(k^3)$, we have $\mathrm{Col}_\varepsilon(C) \geq \Theta(k)$. But now the parameter $m = \Omega(|\Sigma|^{1/3} \log |\Sigma| r) \geq k \log n$ is too large. Our solution is to consider a new error correcting code with small dimension by increasing the alphabet size and show that this new code has the same collision number. More precisely, we partition the m bits into g blocks, each containing m/g bits and treat the code words as strings in Σ'^g where $\Sigma' = \Sigma^{m/g}$. Since $|\Sigma'^k| \leq n^{O(1)}$, we have $m/g \leq O(\frac{\log n}{k \log |\Sigma|}) = O(\frac{\log n}{3k \log k})$. Thus, $g \geq \Theta(\frac{mk \log k}{\log n}) \geq \Theta(k^2 \log k)$. This reduces the parameter growth from k^3 to $k^2 \log k$, and the (randomized) ETH-based running time lower bound can be improved to $n^{O(\sqrt{k/\log k})}$. We hope to see whether some better construction of threshold graph leads to better lower bound of problems we discuss.

1.3 Previous Work

The parameterized complexity of k-MDP had been open for many years. This problem was first resolved by [13]. Interestingly, the reduction in [13] also ruled out constant FPT-approximation algorithm for k-MDP over binary field. In addition, they also ruled out any constant FPT-approximation algorithm for k-CVP in all ℓ_p norms. Recent work by Bennett, Cheraghchi, Guruswami and Ribeiro [10] proved parameterized inapproximability for k-MDP over all finite fields and k-SVP in all ℓ_p norms and arbitrary constant gap. These results are all based on the W[1]-hardness of constant Gap-k-NCP or Gap-k-CVP in [13].

Unfortunately, the gap-creating reduction from k-Clique to constant Gap-k'-NCP or Gap-k'-CVP in [13] has a long reduction chain and causes a significant increase in the parameter. For example, the reduction from k-Clique to constant Gap-k'-NCP contains the following steps (the reduction for Gap-k'-CVP is similar):

- The first step is to reduce k-Clique to the One-Sided Gap Biclique problem. In this step, the reduction outputs a bipartite graph $H = (L \cup R, E)$ and three integers $s = k(k-1)/2$, $\ell = (k+1)!$ and $h > \ell$ on input a graph G and an integer k such that if G contains a k-clique, then there are s vertices in L with h common neighbors. On the other hand, if G contains no k-clique, then every s-vertex set of L has at most ℓ common neighbors in R.

- The second step is to reduce the One-Sided Gap Biclique problem to Gap-k'-Linear Dependent Set problem (Gap-k'-LDS)[5]. On input the bipartite graph $H = (L \cup R, E)$ and three positive integers $s, \ell, h \in \mathbb{N}$, the reduction outputs a set W of vectors and an integer $k' = hs$ such that, if H contains a $K_{s,h}$-subgraph, then there are k' vectors in W that are linearly dependent. If every s-vertex set in L has at most ℓ common neighbors, then any linearly dependent set of W must have size at least $(h/\ell)^{1/s}$. To create a constant gap, one must choose a large parameter h such that $(h/\ell)^{1/s} \geq \gamma hs$ for some $\gamma > 1$. Hence in [13], the authors have to set $h = (k+6)! \cdot (\gamma k^2)^{k^2}$ and $k' = hs \geq k^{\Omega(k^2)} = 2^{\Omega(k^2 \log k)}$.
- The next step is to reduce the Gap-k'-Linear Dependent Set problem (Gap-k'-LDS) to Gap-k''-Maximum Likelihood Decoding problem (Gap-k''-MLD)[6]. This reduction preserves the parameter i.e., $k'' = k$.
- The remaining step gives a reduction from constant Gap-k''-MLD to constant Gap-k''-NCP.

Combining this with the $f(k) \cdot n^{\Omega(k)}$-time lower bound for the k-Clique problem, we only get a $g(k) \cdot n^{\Omega((\log k)^{1/(2+\epsilon)})}$-time lower bound for Gap-k-NCP using the reduction from [13].

Under a stronger gap assumption (Gap-ETH), Manurangsi [38] showed a tight $n^{\Omega(k)}$ time lower bound for constant approximating problems discussed in this article. His approach is to show an $n^{\Omega(k)}$ time lower bound for constant approximating LaberCover, then reduce it to k-UniqueSetCover, then reduce k-UniqueSetCover to gap problems we discuss using reduction in [6]. The key step in his proof is to establish hardness result for approximating k-UniqueSetCover. To our best knowledge, there is no hardness of approximation result for the parameterized k-UniqueSetCover under gap-free assumptions, e.g. ETH and W[1] $\neq$ FPT.

Very recently, Guruswami, Ren and Sandeep [26] showed constant FPT-inapproximability of k-UniqueSetCover under the assumption that Average Baby PIH holds even for 2CSP instance having rectangular relations. It's interesting whether their result and method can shed some light on showing ETH-based $n^{\Omega(k)}$ time lower bound for k-UniqueSetCover. We remark that the ETH-based $n^{\Omega(k)}$ time lower bound for constant approximating k-UniqueSetCover is still an open problem, and so does its FPT-inapproximability assuming W[1] $\neq$ FPT.

1.4 Paper Organization

In Section 2, we give preliminary of this paper. In Section 3, we give a new analysis on collision number of random code, this section can be skipped if readers wants to see the reduction directly. In Section 4, we present our gap-creating reduction for k-MLD$_p$. In Section 5, we show how to apply our reduction to other results and show inapproximability of other problems. For self-containment, we give a proof of equivalence between k-MLD$_p$ and k-NCP$_p$ in Appendix A of our full version.

2 Preliminaries

For integer $m > 0$, let $[m] = \{1, 2, \cdots, m\}$. For prime power $p > 1$, we let $\mathbb{F}_p = \{0, 1 \cdots, p-1\}$ denote the p-sized finite field. We denote $\mathbb{F}_p^+$ as $\mathbb{F}_p \backslash \{0\}$. For a vector $\vec{v} \in \Sigma^m$ and $i \in [m]$, let $\vec{v}[i] \in \Sigma$ denote the i-th entry of $\vec{v}$. For two vectors $\vec{u}$, $\vec{v}$, let $\vec{u} \circ \vec{v}$ denote their concatenation.

[5] In fact, the reduction in [13] from One-Sided Gap Biclique to Gap-k-LDS goes though an intermediate problem called gap bipartite subgraph with minimum degree (GapBSMD).

[6] Again, they introduced a color-coding technique to Gap-k-LDS (Gap-k-Colored-LDS) and used it as an intermediate problem between Gap-k-LDS and Gap-k-MLD, for details see [13, Lemma 4.8, Theorem 5.4].

The symbol $\dot{\cup}$ denotes for the union set of multiple disjoint sets. As a supplement of big-O notation, we let $f(k,n) = O_k(g(n))$ denote there exists constant $c > 0$ and computable function $h : \mathbb{N} \to \mathbb{N}$ such that for any fixed $k > 0$, $f(k,n) < c \cdot h(k)g(n)$ holds for all sufficiently large n.

For alphabet Σ and vector $\vec{u}, \vec{v} \in \Sigma^m$, the relative distance of them is defined as $\mathsf{dist}(\vec{u}, \vec{v}) = \frac{|\{i \in [m] : \vec{u}[i] \neq \vec{v}[i]\}|}{m}$. In this article, we sometimes use "distance" as shorthand of relative distance. For vector $\vec{v} \in \mathbb{Z}^m$ and $p \geq 1$, let the ℓ_p norm of $\vec{v}$ be $\ell_p(\vec{v}) = (\Sigma_{1 \leq i \leq m} |\vec{v}[i]|^p)^{1/p}$.

2.1 Error-correcting Codes

Error-correcting code plays a fundamental role in computer science and information theory. The problem we mainly discuss in this article and the construction we use are closely related to them. We give a general definition of error-correcting code. A detailed and systematic introduction to coding theory can be found at [27].

▶ **Definition 5** (Error-correcting Codes). *Fix an alphabet Σ, an error-correcting code with length m and relative distance $\delta > 0$ is a subset $\mathcal{C} \subseteq \Sigma^m$ satisfying for all $\vec{x}, \vec{y} \in \mathcal{C}$, if $\vec{x} \neq \vec{y}$, $\mathsf{dist}(\vec{x}, \vec{y}) \geq \delta$.*

▶ **Definition 6** (Linear Codes). *Fix an alphabet Σ such that Σ^r and Σ^m being linear spaces, a linear code is an error-correcting code $\mathcal{C} \subseteq \Sigma^m$ associated with a linear function $f : \Sigma^r \to \Sigma^m$ that for all $x \in \Sigma^r$, $f(x) \in \mathcal{C}$.*

2.2 Hypothesis

We introduce the Exponential Time Hypothesis in this section.

▶ **Definition 7** (3-SAT). *Given a 3-CNF formula (conjunctive formal form, each clause contains exactly 3 literals) φ with n variables and m clauses, decide if there exists a boolean assignment $z \in \{0,1\}^n$ that satisfies φ, i.e., $\varphi(z) = 1$.*

▶ **Hypothesis 8** (Exponential Time Hypothesis [30, 31]). *There exists constant $\delta > 0$ such that 3-SAT with n variable and $O(n)$ clauses cannot be solved in time $O(2^{\delta n})$.*

▶ **Hypothesis 9** (Randomized Exponential Time Hypothesis). *There exists constant $\delta > 0$ such that 3-SAT with n variable and $O(n)$ clauses cannot be solved by randomized algorithm in time $O(2^{\delta n})$.*

2.3 Problems

We first give the definition of general parameterized Maximum Likelihood Decoding problem.

γ-Gap-k-MLD$_p$

Instance:	A vector multi-set $V \subseteq \mathbb{F}_p^d$ with size n and a target vector $\vec{t} \in \mathbb{F}_p^d$.
Parameter:	k.
Problem:	Distinguish between the following two cases:
(YES)	There exists k distinct vectors (with respect to multi-set), $\vec{v}_1, \cdots, \vec{v}_k \in V$ and $\alpha_1, \ldots, \alpha_l \in \mathbb{F}_p^+$ such that $\alpha_1\vec{v}_1 + \cdots + \alpha_k\vec{v}_k = \vec{t}$.
(NO)	Any $\ell \leq \gamma k$, l vectors $\vec{v}_1, \ldots, \vec{v}_l \in V$ and $\alpha_1, \ldots, \alpha_l \in \mathbb{F}_p^+$ satisfies $\alpha_1\vec{v}_1 + \cdots + \alpha_l\vec{v}_l \neq \vec{t}$.

To fit requirements in our reduction, we start from a special restricted type of parameterized Maximum Likelihood Decoding problem that vectors are partitioned into k different sets, and the YES case asks for selecting one vector from each set such that they directly add up to the target vector.

The other problems we study includes parameterized Nearest Codeword Problem, which asks if there is a codeword of the given linear code having distance no more than k to a given target vector; Minimum Distance Problem, which asks if the minimum distance of given linear code does not exceed k; Closest Vector Problem, asking if there is a vector in the given linear lattice having ℓ_p distance no more than k to a given target vector; Shortest Vector Problem, asking if the shortest non-zero vector of given linear lattice does not exceed k. Formal definitions of these problems are referred to the full version.

2.4 Probability Inequality

▶ **Theorem 10** (Chernoff Bound). *Consider independent random variables $X_1, \ldots, X_n \in \{0,1\}$ with $X = \sum_{i=1}^{m} X_i$ and $\mu = \mathbb{E}[X]$. For any $0 < \delta < 1$ we have*

$$\Pr[X \leq (1-\delta)\mu] \leq \exp\left(-\frac{\mu\delta^2}{2}\right).$$

3 Collision Number of Error Correcting Codes

In this section, we introduce the definition of collision number of a code, which is key to our gap-creating reductions. Given a collection of strings $S \subseteq \Sigma^m$, we say that S "collides" on the i-th coordinate if there are distinct $x, y \in S$ such that $x[i] = y[i]$. Following the work of [32, 37], we define the collision number of a set of strings as follows.

▶ **Definition 11** (ε-Collision Number). *For a set $C \subseteq \Sigma^m$ and $0 < \varepsilon < 1$, the ε-collision number of C, denote as $\mathrm{Col}_\varepsilon(C)$, is the smallest integer $s \in \mathbb{N}^+$ such that there exists $S \subseteq C$ with $|S| = s$ and S collides on more than εm coordinates, i.e.,*

$$|\{i \in [m] \mid \exists x, y \in S, x \neq y \text{ s.t. } x[i] = y[i]\}| > \varepsilon m.$$

To create a gap for the k-MLD$_p$ problem, we need to construct codes $C \subseteq \Sigma^m$ with collision number $\mathrm{Col}_\varepsilon(C) \geq \Omega(k)$ and m depends only on k. We sketch two constructions (Theorem 13 and Lemma 17).

▶ **Lemma 12** ([32], See also Theorem 10 in [37]). *For any constant $0 < \varepsilon \leq 1$, an error correcting code $C : \Sigma^r \to \Sigma^m$ with relative distance $0 < \delta < 1$ has $\mathrm{Col}_\varepsilon(C) \geq \sqrt{\frac{2\varepsilon}{1-\delta}}$.*

▶ **Theorem 13** ([32, 37]). *Fix any Reed-Solomon code $\mathcal{C}^{RS} : \Sigma^r \to \Sigma^m$ with $r < m \leq |\Sigma|$. For any $0 < \varepsilon < 1$, $\mathrm{Col}_\varepsilon(\mathcal{C}^{RS}) \geq \sqrt{\frac{2\varepsilon m}{r}}$.*

To fit the requirement in our reduction, i.e., $|\Sigma|^r \geq n$, we choose $|\Sigma| = n^{1/k}$ and $r = \Omega(k)$. To fit the requirement that $\mathrm{Col}_\varepsilon(C) = \Omega(k)$ in Lemma 19, the Reed-Solomon code here must satisfy $m = \Omega(k^2 r) = \Omega(k^3)$. Seeking for a shorter code with high ε-collision number, we turn to randomized construction of codes, and we show the following lemma. The proof is similar to [32, Claim 3.4] by showing each coordinate has collision with low probability in a small set, then apply Chernoff bound and union bound to show a small set can hardly have large collision number. Details see the full version.

▶ **Lemma 14.** *For any constant $0 < \varepsilon < 1$ and any random code $C_R : \Sigma^r \to \Sigma^m$ where each codeword is selected uniformly at random in Σ^m, if $m \geq 16\frac{1}{\varepsilon^2}|\Sigma|^{1/3} \ln |\Sigma| r$ and $|\Sigma| = \omega(1)$, then with high probability, $\mathrm{Col}_\varepsilon(C_R) > |\Sigma|^{1/3}$.*

By instantiating Lemma 14 with appropriate parameter, we have:

▶ **Lemma 15.** *For any constant $c > 0$ and $0 < \varepsilon < 1$, there is a randomized algorithm that given integers $n, k \in \mathbb{N}^+$, constructs a code $C \subseteq \Sigma^m$ with parameters $|C| = n, |\Sigma| = O(k^3)$ and $m = O(k \log n)$ such that with high probability, $\mathrm{Col}_\varepsilon(C) > ck$. Moreover, the running time of this algorithm is $O(nm|\Sigma|)$.*

▶ Remark 16. We remark that using an almost identical argument, Lemma 14 can be extended to the case that for each integer $t \geq 3$, if $m > \Omega(|\Sigma|^{1/t} \log |\Sigma| r)$ and $|\Sigma| = \omega(1)$, then w.h.p., $\mathrm{Col}_\varepsilon(C_R) > |\Sigma|^{1/t}$. For constant $t > 3$, setting $|\Sigma| = \Omega(k^t)$, Lemma 15 can be extended to the case with same parameter but larger code alphabet.

By merging the number of code blocks over small alphabet from Lemma 15, we obtain:

▶ **Lemma 17.** *For any constant $c > 1$ and $0 < \varepsilon < 1$, there is a randomized algorithm that given integers $n, k \in \mathbb{N}^+$, constructs a code $C \subseteq \Sigma^m$ with parameters $|C| = n, |\Sigma| = O(n^{1/k})$ and $m = O(k^2 \log k)$ such that with high probability $\mathrm{Col}_\varepsilon(C) > ck$. Moreover, the running time of this algorithm is $O(k^2 \log k n^{1+1/k})$.*

3.1 Limitation of Collision Analysis in [32, 37]

We have already shown the a direct collision analysis yields $m' = O(k^2 \log k)$. Below, we argue that collision analysis from distance must cause a cubic increase in the parameter.

To obtain a constant gap using Lemma 19 in Section 4, we require the collision number of code C to be $\mathrm{Col}_\varepsilon(C) = \Omega(k)$. Combining with Lemma 12, we immediately have the relative distance of code C must satisfy

$$\delta \geq 1 - 1/\Omega(k^2).$$

On the other hand, we have the following Singleton bound from coding theory, whose proof can be found in [27, Section 4.3].

▶ **Theorem 18** (Singleton Bound)**.** *For every code $C : \Sigma^r \to \Sigma^m$ with relative distance δ must have $r \leq m - \delta m + 1$.*

We apply the bound to parameter we choose and obtain $m - (1 - \frac{1}{\Omega(k^2)}))m + 1 \geq r$, i.e.,

$$m \geq \Omega(k^2) r.$$

Our reduction for MLD associates each input vector with a unique codeword, which requires $|C| = |\Sigma|^r \geq n$, leading to

$$r \geq (\log n)/(\log |\Sigma|).$$

Since our reduction runs in time $\Omega(|\Sigma|^k)$, we need $|\Sigma| \leq n^{O(1/k)}$. Thus $r \geq (\log n)/(\log |\Sigma|) \geq \Omega(k)$ and $m \geq \Omega(k^2) r \geq \Omega(k^3)$. Note that we've shown the Reed-Solomon code already achieves $m = O(k^3)$.

4 Gap-creating Reduction for k-MLD$_p$

In this section we present our gap-creation reduction for k-MLD$_p$. First we present a construction that illustrates our main idea and is the crux of our reduction. This construction produces an "unbalanced gap" k'-MLD$_p$ instance in the sense that the output instance is divided into two parts (with different sizes), any solution must contain an amount of vectors in each part. Further, for the NO case, any solution must contain constant fraction more vectors in at least one part. This construction still needs to be modified later to convert into an actual reduction.

▶ **Lemma 19.** *There is an algorithm which on input k sets of length-d vectors $V_1, \cdots, V_k \subseteq \mathbb{F}_p^d$ each of size n, a target vector $\vec{t} \in \mathbb{F}_p^d$ and a code $C \subseteq |\Sigma|^m$ with $|C| = n$ and $\mathrm{Col}_\varepsilon(C) \geq ck$ outputs $A = A_1 \dot\cup \cdots \dot\cup A_k \subseteq \mathbb{F}_p^D$ and $B = B_1 \dot\cup \ldots \dot\cup B_m \subseteq \mathbb{F}_p^D$ with $D = O(d + km|\Sigma|)$ and a target vector $\vec{t}' \in \mathbb{F}_p^D$ in $O(dm^2k^2|\Sigma|(n + |\Sigma|^k))$-time such that*

- **(i)** *If there exist $\vec{v}_1 \in V_1, \ldots, \vec{v}_k \in V_k$ such that $\sum_{i\in[k]} \vec{v}_i = \vec{t}$, then there exists $\vec{a}'_1 \in A_1, \cdots, \vec{a}'_k \in A_k$ and $\vec{b}'_1 \in B_1, \cdots, \vec{b}'_m \in B_m$ with their sum being $\vec{t}'$.*
- **(ii)** *If for any $\vec{v}_1 \in V_1, \ldots, \vec{v}_k \in V_k$ and $\alpha_1, \ldots \alpha_k \in \mathbb{F}_p^+$ it holds that $\alpha_1\vec{v}_1 + \cdots + \alpha_k\vec{v}_k \neq \vec{t}$, then any $X \subseteq A \dot\cup B$ and $\lambda : X \to \mathbb{F}_p^+$ such that $\sum_{\vec{x}\in X} \lambda(\vec{x})\vec{x} = \vec{t}'$ must satisfy at least one of the following:*
  - *$|X \cap A| \geq ck$ and $|X \cap B| \geq m$,*
  - *$|X \cap A| \geq k$ and $|X \cap B| \geq 2(1-\varepsilon)m$.*

Proof. The resulting dimension is $D = d + mk|\Sigma| + k + m$. We break the resulting dimension into 4 blocks respectively of size $d, mk|\Sigma|, k$ and m. To be precise, for any vector $\vec{x} \in \mathbb{F}_p^D$, let

- $\vec{x}^{(1)} \in \mathbb{F}_p^d$ be the first block,
- $\vec{x}^{(2)} \in \mathbb{F}_p^{mk|\Sigma|}$ be second block,
- $\vec{x}^{(3)} \in \mathbb{F}_p^k$ be the third block,
- $\vec{x}^{(4)} \in \mathbb{F}_p^m$ be the fourth block.

We further break the second block into m sub-blocks each of size $k|\Sigma|$, i.e., $\vec{x}^{(2)} = \vec{x}^{(2,1)} \circ \cdots \circ \vec{x}^{(2,m)}$.

We let $\vec{e}_i$ be the indicator vector of which the i-th entry is 1 and the other entries are 0. To be convenient, the dimension of $\vec{e}_i$ depends on the context. Specially we let $\iota : \Sigma \to [|\Sigma|]$ be an arbitrary bijection, and for every $\sigma \in \Sigma$ we let

$$\vec{e}_\sigma = (\overbrace{0, \ldots, 0}^{\iota(\sigma)-1}, \underbrace{1, 0 \ldots, 0}_{|\Sigma|}).$$

Construction of A. For every V_i, associate each $\vec{v} \in V_i$ a distinct codeword of C, denoted by $C(\vec{v})$. For every $i \in [k]$ and $\vec{v} \in V_i$, introduce a vector $\vec{a}_{i,\vec{v}}$ as

- $\vec{a}^{(1)}_{i,\vec{v}} = \vec{v}$,
- $\vec{a}^{(2,j)}_{i,\vec{v}} = (\underbrace{\overbrace{\vec{0}, \ldots, \vec{0}}^{(i-1)}, \vec{e}_{C(\vec{v})[j]}, \vec{0}, \ldots, \vec{0}}_{k})$, for every $j \in [m]$,
- $\vec{a}^{(3)}_{i,\vec{v}} = \vec{e}_i$,
- $\vec{a}^{(4)}_{i,\vec{v}} = \vec{0}_m$.

And we let $A_i = \{\vec{a}_{i,\vec{v}} \mid \vec{v} \in V_i\}$ and $A = A_1 \cup \cdots \cup A_k$.

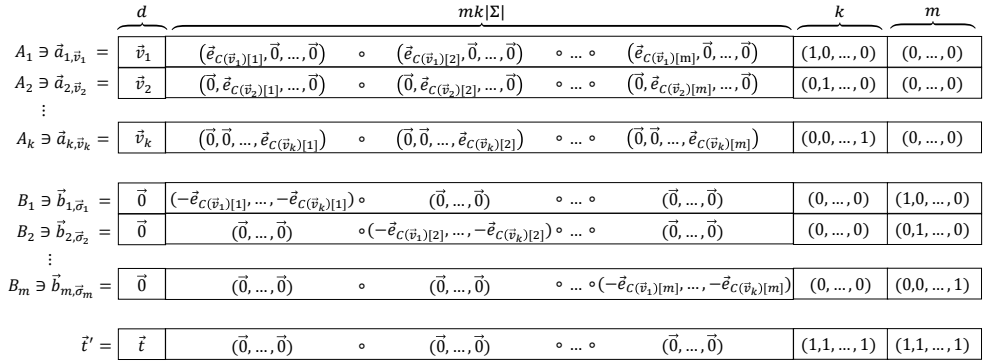


Figure 1 Illustration for the vectors of Lemma 19 in the completeness setting. We can choose each $\vec{b}_{j,\vec{\sigma}_j}$ as $\vec{\sigma}_j = (C(\vec{v}_1)[j], \cdots, C(\vec{v}_k)[j])$.

Construction of B. For every $j \in [m]$ and $\vec{\sigma} = (\sigma_1, \dots, \sigma_k) \in \Sigma^k$, introduce a vector $\vec{b}_{j,\vec{\sigma}}$ as

- $\vec{b}^{(1)}_{j,\vec{\sigma}} = \vec{0}_d$,
- $\vec{b}^{(2,j)}_{j,\vec{\sigma}} = (-\vec{e}_{\sigma_1}, \cdots - \vec{e}_{\sigma_k})$, $\vec{b}^{(2,j')}_{j,\vec{\sigma}} = \vec{0}_k$ for every $j' \in [m] \backslash \{j\}$,
- $\vec{b}^{(3)}_{j,\vec{\sigma}} = \vec{0}_k$,
- $\vec{b}^{(4)}_{j,\vec{\sigma}} = \vec{e}_j$.

We let $B_j = \{\vec{b}_{j,\vec{\sigma}} \mid \vec{\sigma} \in \Sigma^k\}$ and $B = B_1 \cup \cdots \cup B_m$.

Finally we set the target vector $\vec{t}'$ as

- $\vec{t}'^{(1)} = \vec{t}$,
- $\vec{t}'^{(2)} = \vec{0}_{mk|\Sigma|}$,
- $\vec{t}'^{(3)} = \vec{1}_k$,
- $\vec{t}'^{(4)} = \vec{1}_m$.

Time complexity. Producing each vector in A requires $O(d + mk|\Sigma| + km) = O(d + mk|\Sigma|)$ time, so the total time cost producing A is $O(dkn + mk^2 n|\Sigma|)$. Producing each vector in B also requires $O(d + mk|\Sigma|)$ time, and the total time cost producing B is $O(dm|\Sigma|^k + m^2 k|\Sigma|^{k+1})$. So the total time cost of this reduction is $O(dm^2k^2|\Sigma|(n + |\Sigma|^k))$.

Proof of (i). Suppose there exist $\vec{v}_1 \in V_1, \cdots, \vec{v}_k \in V_k$ satisfying $\sum_{i\in[k]} \vec{v}_i = \vec{t}$. For every $i \in [k]$ we choose a vector $\vec{a}_{i,\vec{v}_i} \in A_i$. And for every $j \in [m]$ we choose a vector $\vec{b}_{j,\vec{\sigma}_j} \in B_j$, where $\vec{\sigma}_j = (C(\vec{v}_1)[j], \dots, C(\vec{v}_m)[j]) \in \Sigma^k$. We now examine that $\sum_{i\in[k]} \vec{a}_{i,\vec{v}_i} + \sum_{j\in[m]} \vec{b}_{j,\vec{\sigma}_j} = \vec{t}'$ as:

- For the first block,

$$\sum_{i\in[k]} \vec{a}^{(1)}_{i,\vec{v}_i} + \sum_{j\in[m]} \vec{b}^{(1)}_{j,\vec{\sigma}_j} = \sum_{i\in[k]} \vec{v}_i + \sum_{j\in[m]} \vec{0}_d = \vec{t} = \vec{t}'^{(1)}.$$

- For every $j \in [m]$ the $(2, j)$-th block,

$$\begin{aligned}
\sum_{i\in[k]} \vec{a}^{(2,j)}_{i,\vec{v}_i} + \sum_{j'\in[m]} \vec{b}^{(2,j)}_{j',\vec{\sigma}_{j'}} &= \sum_{i\in[k]} \vec{a}^{(2,j)}_{i,\vec{v}_i} + \vec{b}^{(2,j)}_{j,\vec{\sigma}_j} \\
&= \sum_{i\in[k]} (\overbrace{\vec{0}, \dots, \vec{0}}^{i-1}, \vec{e}_{C(\vec{v}_i)[j]}, \vec{0}, \dots, \vec{0}) + (-\vec{e}_{C(\vec{v}_1)[j]}, \dots, -\vec{e}_{C(\vec{v}_k)[j]}) \\
&= (\vec{e}_{C(\vec{v}_1)[j]}, \dots, \vec{e}_{C(\vec{v}_k)[j]}) + (-\vec{e}_{C(\vec{v}_1)[j]}, \dots, -\vec{e}_{C(\vec{v}_k)[j]}) \\
&= \vec{0}_{k|\Sigma|} = \vec{t}'^{(2,j)}.
\end{aligned}$$

- For the third block,

$$\sum_{i\in[k]} \vec{a}^{(3)}_{i,\vec{v}_i} + \sum_{j\in[m]} \vec{b}^{(3)}_{j,\vec{\sigma}_j} = \sum_{i\in[k]} \vec{e}_i + \sum_{j\in[m]} \vec{0}_k = \vec{1}_k = \vec{t}'^{(3)}.$$

- For the fourth block,

$$\sum_{i\in[k]} \vec{a}^{(4)}_{i,\vec{v}_i} + \sum_{j\in[m]} \vec{b}^{(4)}_{j,\vec{\sigma}_j} = \sum_{i\in[k]} \vec{0}_m + \sum_{j\in[m]} \vec{e}_j = \vec{1}_m = \vec{t}'^{(4)}.$$

Proof of (ii). Suppose $X \subseteq A \dot\cup B$ and $\lambda : X \to \mathbb{F}_p^+$ such that $\sum_{\vec{x}\in X} \lambda(\vec{x})\vec{x} = \vec{t}'$. Observe the third block of the equation:

$$\sum_{\vec{x}\in X} \lambda(\vec{x})\vec{x}^{(3)} = \sum_{i\in[k]} \sum_{\vec{x}\in X\cap A_i} \lambda(\vec{x})\vec{e}_i = \vec{1}_m = \vec{t}'^{(3)}.$$

For every $i \in [k]$, $X \cap A_i$ must not be empty since $\sum_{\vec{x}\in X\cap A_i} \lambda(\vec{x}) = 1$. Also similarly by observing the fourth block it holds that $X \cap B_j$ must not be empty for every $j \in [m]$. Therefore $|X \cap A| \geq k$ and $|X \cap B| \geq m$.

Further suppose that any $\vec{v}_1 \in V_1, \ldots, \vec{v}_k \in V_k$ and $\alpha_1, \ldots \alpha_k \in \mathbb{F}_p^+$ must satisfy $\alpha_1\vec{v}_1 + \cdots + \alpha_k\vec{v}_k \neq \vec{t}$, we show that either $|X \cap A| \geq ck$ or $|X \cap B| \geq 2(1-\varepsilon)m$.

We let $I \subseteq [m]$ be the set of indices j that $X \cap B_j$ contains only one vector, i.e.,

$$I = \{j \in [m] : |X \cap B_j| = 1\}.$$

Since $|X \cap B_j| \geq 1$ for every $j \in [m]$, if $|I| \leq \varepsilon m$ then

$$|X \cap B| \geq \sum_{j\in[m]\setminus I} |X \cap B_j| \geq 2(1-\varepsilon)m$$

as desired. It remains to show that if $|I| > \varepsilon m$ then $|X \cap A| \geq ck$.

First we claim that there must be an $i \in [k]$ such that $X \cap A_i$ contains more than one vector. Otherwise suppose that $|X \cap A_i| = 1$ for every $i \in [k]$, let $\vec{a}_{i,\vec{v}_i} \in X \cap A_i$ be the unique vector in $X \cap A_i$. Recall that in the first block, vectors in $X \cap B$ are all zero, so the sum of vectors in X in the first block is

$$\sum_{\vec{x}\in X} \lambda(\vec{x})\vec{x}^{(1)} = \sum \lambda(\vec{a}_{i,\vec{v}_i})\vec{a}'^{(1)}_{i,\vec{v}_i} = \sum_{i\in[k]} \lambda(\vec{a}_{i,\vec{v}_i})\vec{v}_i = \vec{t} = \vec{t}'^{(1)}$$

This contradicts to our assumption that for all $\vec{v}_1 \in V_1, \ldots, \vec{v}_k \in V_k$ and $\alpha_1, \ldots, \alpha_k \in \mathbb{F}_p^+$, $\sum_{i\in[k]} \alpha_i\vec{v}_i \neq \vec{t}$. Therefore, there must be such an index $i^* \in [k]$ that $|A'_{i^*}| > 1$.

Let $l > 1$ be the size of $X \cap A_{i^*}$, we next show that $l \geq ck$. Suppose that $X \cap A_{i^*} = \{\vec{a}_{i^*,\vec{v}_1}, \ldots, \vec{a}_{i^*,\vec{v}_l}\}$ where $\vec{v}_1, \ldots \vec{v}_l \in V_{i^*}$. We show in the following that the codeword set $\{C(\vec{v}_1), \ldots, C(\vec{v}_l)\}$ must collide on every $j \in I$. Fix any $j \in I$, let $\vec{b}_{j,\vec{\sigma}}$ be the unique vector in $X \cap B_j$, where $\vec{\sigma} = (\sigma_1, \ldots, \sigma_k)$. Recall that the $(2, j)$-th block of the resulting dimension consists of $k|\Sigma|$ coordinates, here we further break it down into k sub-blocks each of size $|\Sigma|$, and we focus on the $(2, j, i^*)$-th sub-block:

$$\begin{aligned}
\sum_{\vec{x}\in X} \lambda(\vec{x})\vec{x}^{(2,j,i^*)} &= \lambda(\vec{a}_{i^*,\vec{v}_1})\vec{a}^{(2,j,i^*)}_{i^*,\vec{v}_1} + \cdots + \lambda(\vec{a}_{i^*,\vec{v}_l})\vec{a}^{(2,j,i^*)}_{i^*,\vec{v}_l} + \lambda(\vec{b}_{j,\vec{\sigma}})\vec{b}^{(2,j,i^*)}_{j,\vec{\sigma}} \\
&= \lambda(\vec{a}_{i^*,\vec{v}_1})\vec{e}_{C(\vec{v}_1)[j]} + \cdots + \lambda(\vec{a}_{i^*,\vec{v}_l})\vec{e}_{C(\vec{v}_l)[j]} - \lambda(\vec{b}_{j,\vec{\sigma}})\vec{e}_{\sigma_{i^*}} \\
&= \vec{0}_{|\Sigma|} = \vec{t}'^{(2,j,i^*)}.
\end{aligned}$$

If $C(\vec{v}_1)[j], \ldots, C(\vec{v}_l)[j]$ are all distinct, the equation $\lambda(\vec{a}_{i^*,\vec{v}_1})\vec{e}_{C(\vec{v}_1)[j]} + \cdots + \lambda(\vec{a}_{i^*,\vec{v}_l})\vec{e}_{C(\vec{v}_l)[j]} - \lambda(\vec{b}_{j,\vec{\sigma}})\vec{e}_{\sigma_{i^*}} = \vec{0}_{|\Sigma|}$ must not be satisfied since $l > 1$ and the λ's are nonzero. Therefore $\{C(\vec{v}_1), \ldots, C(\vec{v}_l)\}$ must collide on the j-th coordinate.

If $|I| > \varepsilon m$ then $\{C(\vec{v}_1), \ldots, C(\vec{v}_l)\}$ collide on more than εm coordinates, by the definition of collision number, it holds that $|\{C(\vec{v}_1), \ldots, C(\vec{v}_l)\}| \geq \mathrm{Col}_\varepsilon(C) \geq ck$. And thus $|X \cap A| \geq |X \cap A_{i^*}| \geq ck$. ◀

Since the codes (with good collision number) we construct has codeword length $m = O(k^2 \log k)$ (Lemma 17) much greater than k, the above construction cannot directly lead to a gap-creating reduction for k-MLD. To settle this, intuitively we further duplicate the vector sets $A_1, \ldots, A_k$ several times into m vector sets. This leads to our gap creating reduction as follows.

▶ **Theorem 20.** *For any $0 < \varepsilon < 1$, there is a randomized reduction which on input k sets of length-d vectors $V_1, \cdots, V_k \subseteq \mathbb{F}_p^d$ each of size n and a target vector $\vec{t} \in \mathbb{F}_p^d$ outputs k' vector sets $U_1, \ldots, U_{k'} \subseteq \mathbb{F}_p^D$ and a target vector $\vec{t}' \in \mathbb{F}_p^D$ with $k' = O(k^2 \log k)$ and $D = O(k'd + k'^2 n^{1/k})$ in $O(d2^{O(k)} n^{1.01})$ time such that*

- **(i)** *If there exist $\vec{v}_1 \in V_1, \ldots, \vec{v}_k \in V_k$ such that $\sum_{i\in[k]} \vec{v}_i = \vec{t}$, then there exists $\vec{u}_1 \in U_1, \ldots, \vec{u}_{k'} \in U_{k'}$ with their sum being $\vec{t}'$.*
- **(ii)** *If any $\vec{v}_1 \in V_1, \ldots, \vec{v}_k \in V_k$ and $\alpha_1, \ldots \alpha_k \in \mathbb{F}_p^+$ must satisfy $\alpha_1\vec{v}_1 + \cdots + \alpha_k\vec{v}_k \neq \vec{t}$, then any $X \subseteq \bigcup_{i\in[k']} U_i$ and $\lambda : X \to \mathbb{F}_p^+$ such that $\sum_{\vec{x}\in X} \lambda(\vec{x})\vec{x} = \vec{t}'$ must satisfy $|X| \geq (\frac{3}{2} - \varepsilon)k'$.*

▶ Remark 21. Consider the k-VectorSum$_q$ problem in [36], whose definition is identical to k-MLD$_q$ except that it requires all the coefficients being 1. A closer look at our reduction shows that it can directly create a gap of almost $(q+1)/2$ for k-VectorSum$_q$ rather than almost $\frac{3}{2}$ in the k-MLD$_q$ case.

5 Lower Bounds for Gap-k-NCP and Other Problems

In this section, we show the reduction described in the previous sections implies improved running time lower bounds for various problems under ETH.

5.1 Maximum Likelihood Decoding and Nearest Codeword Problem

In [14], Bhattacharyya, Ghoshal, Karthik and Manurangsi presented a gap amplification procedure for Gap-k-MLD$_p$. Although they only discussed the procedure on the binary field, it's straightforward to see the procedure also works for Gap-k-MLD$_p$ instances over all $\mathbb{F}_p$. Formally,

▶ **Theorem 22** (Generalization of Lemma 4.5 in [14]). *For integers $k_1, k_2 > 0$, $k' = k_2 + k_1k_2$ and reals $\gamma_1, \gamma_2 > 1$, $\gamma' \geq \gamma_1\gamma_2(1 - \frac{1}{k_1})$, there is a polynomial time algorithm that on input 2 vector sets $U \subseteq \mathbb{F}_p^{m_1}, V \subseteq \mathbb{F}_p^{m_2}$, $|U| = n_1, |V| = n_2$, two target vectors $\vec{t} \in \mathbb{F}_p^{m_1}, \vec{s} \in \mathbb{F}_p^{m_2}$, outputs a vector set $W \subseteq \mathbb{F}_p^{m_2+n_1m_1}$ and a target vector $\vec{t}' \in \mathbb{F}_p^{m_2+n_1m_1}$ satisfies:*

- *If $(U, \vec{t})$ is a YES instance of γ_1-Gap-k_1-MLD$_p$ instance and $(V, \vec{s})$ is a YES instance of γ_2-Gap-k_2-MLD$_p$ instance, then $(W, \vec{t}')$ is a YES instance of γ'-Gap-k'-MLD$_p$.*
- *If $(U, \vec{t})$ is a NO instance of γ_1-Gap-k_1-MLD$_p$ instance and $(V, \vec{s})$ is a NO instance of γ_2-Gap-k_2-MLD$_p$ instance, then $(W, \vec{t}')$ is a NO instance of γ'-Gap-k'-MLD$_p$.*

Readers seeking for a formal proof is referred to [14, Section 4.2].

5.1.1 ETH-based Running Time Lower Bound

Taking a closer look at the reduction from 3-SAT to k-VectorSum in [36, Theorem 11], we observe that by applying a minor modification, their reduction can actually have soundness condition as:

- If ϕ is not satisfiable, then for any $\vec{v}_1 \in V_1, \ldots, \vec{v}_k \in V_k$ and $\alpha_1, \ldots, \alpha_k \in \mathbb{F}_p^+$, $\Sigma_{i=1}^k \alpha_i \vec{v}_i \neq \vec{t}$.

The modification is simply appending a vector $(0^{i-1} \circ 1 \circ 0^{k-i})$ to each vector in V_i, for all $1 \leq i \leq k$. Then, the target vector is changed from a zero vector to $\vec{t} = 0^d \circ 1^k$. Completeness of their reduction is trivially preserved. For soundness we claim, we note that for any $\vec{v}_1 \in V_1, \cdots, \vec{v}_k \in V_k$ and $\alpha_1, \cdots, \alpha_k \in \mathbb{F}_p$, if $\Sigma_{i=1}^k \alpha_i \vec{v}_i = \vec{t}$, then $\alpha_1 = \cdots = \alpha_k = 1$.

By strengthening the soundness condition in [36], we obtain exactly the restricted version of k-MLD$_p$ in the previous sections. Combining with their soundness for k-VectorSum, we obtain the following hardness result for k-MLD$_p$ as:

▶ **Theorem 23** (Theorem 11 in [36]). *Assuming ETH, for any constant integer p, k-MLD$_p$ has no $n^{o(k)}$-time algorithm.*

The parameterized MLD and NCP are equivalent in the sense that there exist reductions preserving the solution size in both direction. Recall that Theorem 20 showed a reduction from k-MLD$_p$ to $(3/2-\varepsilon)$-Gap-k'-MLD$_p$ with $k' = k^2 \log k$ and $\varepsilon > 0$. Combining running time lower bound in Theorem 23, we have:

▶ **Theorem 24.** *Assuming randomized ETH, for any constant integer p, constant $1 < \gamma < \frac{3}{2}$, γ-Gap-k-MLD$_p$ and γ-Gap-k-NCP$_p$ has no $O_k(n^{o(\sqrt{k/\log k})})$-time algorithm.*

By applying Theorem 22 to the gap instance itself $O(\log\log\gamma)$ times, we can obtain the ETH-based time lower bound for approximating parameterized MLD and NCP to any constant factor.

▶ **Corollary 25.** *Assuming ETH, for any constant integer p and constant $\gamma > 1$, γ-Gap-k-MLD$_p$ and γ-Gap-k-NCP$_p$ have no $O_k(n^{o(k^\epsilon)})$ time algorithm, where $\epsilon = \frac{1}{\mathit{polylog}(\gamma)}$ is a constant.*

5.2 Minimum Distance Problem

The reduction from Gap-k-NCP to Gap-k-MDP in [10] is as follows.

▶ **Theorem 26** ([10], Theorem 3.1 and 3.3). *For any prime power $p \geq 2$ there is a randomized reduction from $(4p)$-Gap-k-NCP$_p$ to $\frac{4p}{4p-1}$-Gap-k'-MDP$_p$ runs in polynomial time with $k' = O(k)$.*

Use our gap-creating reduction for Gap-k-MLD, and apply the gap amplification for a constant number of times, then use Theorem 26 to reduce to Gap-k'-MDP and self-tensoring the instance for a constant number of times, we have:

▶ **Corollary 27.** *Assuming randomized ETH, for any prime power $p \geq 2$ and real number $\gamma > 1$, γ-Gap-k-MDP$_p$ has no $O_k(n^{o(k^\epsilon)})$ time algorithm, where $\epsilon = \Theta(\frac{1}{p \log \gamma \cdot \mathsf{polylog}(p)})$.*

5.3 Closest Vector Problem

We need a reduction from (2γ)-Gap-k-MLD$_q$ to γ-Gap-$2k$-CVP$_p$ from [13].

▶ **Theorem 28** ([13], Theorem 7.2). *For any real numbers $\gamma, p \geq 1$ and a prime number $q > 2\gamma$, there is a reduction from (2γ)-Gap-k-MLD$_q$ to γ-Gap-k'-CVP$_p$ runs in polynomial time, where $k' = 2k$.*

Use our gap-creating reduction for Gap-k-MLD, and apply gap amplification for a constant number of times to obtain 2γ-gap, then use Theorem 28 to reduce to Gap-k'-CVP, we have:

▶ **Corollary 29.** *Assuming ETH, there exists constant $c > 0$, for any real numbers $p, \gamma \geq 1$, γ-Gap-k-CVP$_p$ has no $O_k(n^{o(k^\epsilon)})$ time algorithm, where $\epsilon = \Theta(\frac{1}{\gamma^c})$.*

5.4 Shortest Vector Problem

Combining our work with [10], we show two ways of obtaining running time lower bound for γ-Gap-k-SVP$_p$. The first way reduces from Gap-k-CVP$_p$, obtaining lower bound for only a fixed constant ratio and all l_p norms where $p \geq 1$. The second way reduces from Gap-k-NCP$_q$, obtaining lower bound for all constant ratio and all l_p norms except for l_1.

5.4.1 Reduction From Gap-k-CVP$_p$

▶ **Theorem 30** ([10], Theorem 4.1 and 4.3, modified). *For any real numbers $p \geq 1$ and $\gamma' \in [1, 2)$ there exist a real number $\gamma \geq 1$ [7] and a reduction from γ-Gap-k-CVP$_p$ to γ'-Gap-k'-SVP$_p$ runs in polynomial time, where $k' \leq \gamma k$.*

Use Corollary 29 to obtain a γ_0-gap CVP instance, where γ_0 fits requirement in Theorem 30, then use Theorem 30 we have:

▶ **Corollary 31.** *Assuming randomized ETH, for any real numbers $p \geq 1$ and $\gamma \in [1, 2)$, γ-Gap-k-SVP$_p$ has no $O_k(n^{o(k^\epsilon)})$ time algorithm,where $0 < \epsilon < 1$ is some constant that depends on p and γ.*

5.4.2 Reduction From Gap-k-NCP$_2$

▶ **Theorem 32** ([10], Lemma 5.1 and Theorem 5.2, modified). *There exists a constant real $\mu \geq 1$ such that, for any real numbers $p > 1$ and $\gamma' \geq 1$, there exists a reduction from μ-Gap-k-NCP$_2$ to γ'-Gap-k'-SVP$_p$ runs in polynomial time, where $k' = O(k^c)$, $c > 1$ is a constant only depends on p and γ'*[8].

The reduction in Theorem 32 in fact proceeds in two steps: first reduces μ-Gap-k-NCP$_2$ to γ'-Gap-k'-SVP$_p$ for some fixed $\gamma' > 1$ with $k' < \mu k$ (while having some additional properties for the second step), then use a tensor technique to amplify the gap to any constant.

▶ **Corollary 33.** *Assuming randomized ETH, for any real numbers $p > 1$ and $\gamma \geq 1$, γ-Gap-k-SVP$_p$ has no $O_k(n^{o(k^\epsilon)})$ time algorithm, where $0 < \epsilon < 1$ is some constant that depends on p and γ.*

---

[7] $\gamma = (\max\left(12/\varepsilon, \frac{1}{(1+\varepsilon/2)^{1/p}-1}\right))^p$ where $\varepsilon = (\gamma')^{-1} - 1/2 > 0$.

[8] There are two problems here about the parameter blow-up, one is that $k' \leq (\mu k)^{O(1)}$ due to the Haviv-Regev "tensoring" step of SVP, the other is that to achieve final gap γ', the gap μ of NCP needs to satisfy $\frac{\mu}{2^p+1+\alpha\mu} > \gamma'$ for some $1/2 + 2^{-p} < \alpha < 1$, causing a polynomial blow-up of parameter to achieve such μ.

6 Conclusion

We have presented new ETH-based lower bounds for approximating parameterized nearest codeword problem and its related problems, improving upon the previous results from [10,13]. Our reduction technique is also simpler and more straight forward than the one used in [13]. However, our results still do not match the lower bound for constant Gap-k-NCP based on Gap-ETH [38]. A natural open problem is to close this gap by proving a stronger lower bound under an assumption that is weaker than Gap-ETH, such as constant Gap-k-Clique has no $n^{o(k)}$-time algorithm. This would be a key step towards understanding the fine-grained complexity of parameterized nearest codeword problem and its variants.

▶ **Open Problem 34.** *Prove $n^{o(k)}$ time lower bound of approximating k-NCP$_p$ or its related problems to any constant factor under assumptions weaker than Gap-ETH.*

To show such a result, as pointed out in [38], one might need to come up with a better "one-shot proof" that gives arbitrary constant factors without tensoring, and with linear parameter growth.

In this paper, we give a new method of composing *threshold graph* with vector problems to yield hardness of approximation results. We showed the limitation of analyzing collision number of a code from its relative distance in [32,37], and improved the analysis to bypass the limitation above. It might be interesting to consider whether this result can be further improved to yield threshold graph with better parameters, or some limitations of our method can be discovered, formally:

▶ **Open Problem 35.** *Give a better construction of strong threshold graph in Section 1.2 with $h = \Omega(k)$ and $m = O(k)$, or show that such graphs do not exist.*

References

1 Divesh Aggarwal, Huck Bennett, Zvika Brakerski, Alexander Golovnev, Rajendra Kumar, Zeyong Li, Spencer Peters, Noah Stephens-Davidowitz, and Vinod Vaikuntanathan. Lattice problems beyond polynomial time. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1516–1526. ACM, 2023. doi:10.1145/3564246.3585227.

2 Divesh Aggarwal, Huck Bennett, Alexander Golovnev, and Noah Stephens-Davidowitz. Fine-grained hardness of CVP(P) - everything that we can prove (and nothing else). In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 1816–1835. SIAM, 2021. doi:10.1137/1.9781611976465.109.

3 Divesh Aggarwal and Noah Stephens-Davidowitz. (gap/s)eth hardness of SVP. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 228–238. ACM, 2018. doi:10.1145/3188745.3188840.

4 Miklós Ajtai. The shortest vector problem in L_2 is *NP*-hard for randomized reductions (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 10–19. ACM, 1998. doi:10.1145/276698.276705.

5 Noga Alon, Rina Panigrahy, and Sergey Yekhanin. Deterministic approximation algorithms for the nearest codeword problem. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009. Proceedings*, volume 5687 of *Lecture Notes in Computer Science*, pages 339–351. Springer, 2009. doi:10.1007/978-3-642-03685-9_26.

6 Sanjeev Arora, László Babai, Jacques Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes, and systems of linear equations. *J. Comput. Syst. Sci.*, 54(2):317–331, 1997. doi:10.1006/jcss.1997.1472.

7 Per Austrin and Subhash Khot. A simple deterministic reduction for the gap minimum distance of code problem. *IEEE Trans. Inf. Theory*, 60(10):6636–6645, 2014. doi:10.1109/TIT.2014.2340869.

8 S Barg. Some new np-complete coding problems. *Problemy Peredachi Informatsii*, 30(3):23–28, 1994.

9 Huck Bennett. The complexity of the shortest vector problem. *SIGACT News*, 54(1):37–61, 2023. doi:10.1145/3586165.3586172.

10 Huck Bennett, Mahdi Cheraghchi, Venkatesan Guruswami, and João Ribeiro. Parameterized inapproximability of the minimum distance problem over all fields and the shortest vector problem in all ℓ_p norms. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023*, pages 553–566. ACM, 2023. doi:10.1145/3564246.3585214.

11 Huck Bennett, Chris Peikert, and Yi Tang. Improved hardness of BDD and SVP under gap-(s)eth. In *13th Innovations in Theoretical Computer Science Conference, ITCS 2022*, volume 215 of *LIPIcs*, pages 19:1–19:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.19.

12 Elwyn R. Berlekamp, Robert J. McEliece, and Henk C. A. van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Trans. Inf. Theory*, 24(3):384–386, 1978. doi:10.1109/TIT.1978.1055873.

13 Arnab Bhattacharyya, Édouard Bonnet, László Egri, Suprovat Ghoshal, Karthik C. S., Bingkai Lin, Pasin Manurangsi, and Dániel Marx. Parameterized intractability of even set and shortest vector problem. *J. ACM*, 68(3):16:1–16:40, 2021. doi:10.1145/3444942.

14 Arnab Bhattacharyya, Suprovat Ghoshal, Karthik C. S., and Pasin Manurangsi. Parameterized intractability of even set and shortest vector problem from gap-eth. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 17:1–17:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.17.

15 Boris Bukh, Karthik C. S., and Bhargav Narayanan. Applications of random algebraic constructions to hardness of approximation. In *62nd IEEE Annual Symposium on Foundations of Computer Science*, pages 237–244. IEEE, 2021. doi:10.1109/FOCS52979.2021.00032.

16 Yijia Chen and Bingkai Lin. The constant inapproximability of the parameterized dominating set problem. *SIAM J. Comput.*, 48(2):513–533, 2019. doi:10.1137/17M1127211.

17 Qi Cheng. Hard problems of algebraic geometry codes. *IEEE Trans. Inf. Theory*, 54(1):402–406, 2008. doi:10.1109/TIT.2007.911213.

18 Qi Cheng and Daqing Wan. A deterministic reduction for the gap minimum distance problem. *IEEE Trans. Inf. Theory*, 58(11):6935–6941, 2012. doi:10.1109/TIT.2012.2209198.

19 Irit Dinur. Mildly exponential reduction from gap-3sat to polynomial-gap label-cover. In *Electronic colloquium on computational complexity ECCC; research reports, surveys and books in computational complexity*, 2016.

20 Irit Dinur, Guy Kindler, Ran Raz, and Shmuel Safra. Approximating CVP to within almost-polynomial factors is np-hard. *Comb.*, 23(2):205–243, 2003. doi:10.1007/s00493-003-0019-y.

21 Rodney G. Downey, Michael R. Fellows, Alexander Vardy, and Geoff Whittle. The parametrized complexity of some fundamental problems in coding theory. *SIAM J. Comput.*, 29(2):545–570, 1999. doi:10.1137/S0097539797323571.

22 Ilya Dumer, Daniele Micciancio, and Madhu Sudan. Hardness of approximating the minimum distance of a linear code. *IEEE Trans. Inf. Theory*, 49(1):22–37, 2003. doi:10.1109/TIT.2002.806118.

23 Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

24 Uriel Feige and Daniele Micciancio. The inapproximability of lattice and coding problems with preprocessing. *J. Comput. Syst. Sci.*, 69(1):45–67, 2004. doi:10.1016/j.jcss.2004.01.002.

25 Oded Goldreich, Daniele Micciancio, Shmuel Safra, and Jean-Pierre Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Inf. Process. Lett.*, 71(2):55–61, 1999. doi:10.1016/S0020-0190(99)00083-6.

26 Venkatesan Guruswami, Xuandi Ren, and Sai Sandeep. Baby pih: Parameterized inapproximability of min csp. *Electron. Colloquium Comput. Complex.*, TR23-155, 2023. arXiv:TR23-155.

27 Venkatesan Guruswami, Atri Rudra, and Madhu Sudan. Essential coding theory. *Draft available at* https://cse.buffalo.edu/faculty/atri/courses/coding-theory/book/, 2(1), 2023.

28 Venkatesan Guruswami and Alexander Vardy. Maximum-likelihood decoding of reed-solomon codes is np-hard. *IEEE Trans. Inf. Theory*, 51(7):2249–2256, 2005. doi:10.1109/TIT.2005.850102.

29 Ishay Haviv and Oded Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. *Theory Comput.*, 8(1):513–531, 2012. doi:10.4086/toc.2012.v008a023.

30 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

31 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

32 Karthik C. S. and Inbal Livni Navon. On hardness of approximation of parameterized set cover and label cover: Threshold graphs from error correcting codes. In *4th Symposium on Simplicity in Algorithms*, pages 210–223. SIAM, 2021. doi:10.1137/1.9781611976496.24.

33 Subhash Khot. Hardness of approximating the shortest vector problem in lattices. *J. ACM*, 52(5):789–808, 2005. doi:10.1145/1089023.1089027.

34 Bingkai Lin. The parameterized complexity of the k-biclique problem. *J. ACM*, 65(5):34:1–34:23, 2018. doi:10.1145/3212622.

35 Bingkai Lin. A simple gap-producing reduction for the parameterized set cover problem. In *46th International Colloquium on Automata, Languages, and Programming*, volume 132 of *LIPIcs*, pages 81:1–81:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.81.

36 Bingkai Lin, Xuandi Ren, Yican Sun, and Xiuhan Wang. On lower bounds of approximating parameterized k-clique. In *49th International Colloquium on Automata, Languages, and Programming*, volume 229 of *LIPIcs*, pages 90:1–90:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.90.

37 Bingkai Lin, Xuandi Ren, Yican Sun, and Xiuhan Wang. Constant approximating parameterized k-setcover is w[2]-hard. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms*, pages 3305–3316. SIAM, 2023. doi:10.1137/1.9781611977554.ch126.

38 Pasin Manurangsi. Tight running time lower bounds for strong inapproximability of maximum k-coverage, unique set cover and related problems (via t-wise agreement testing theorem). In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 62–81. SIAM, 2020. doi:10.1137/1.9781611975994.5.

39 Pasin Manurangsi and Prasad Raghavendra. A birthday repetition theorem and complexity of approximating dense csps. *arXiv preprint arXiv:1607.02986*, 2016.

40 Daniele Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. *SIAM J. Comput.*, 30(6):2008–2035, 2000. doi:10.1137/S0097539700373039.

41 Daniele Micciancio. The hardness of the closest vector problem with preprocessing. *IEEE Trans. Inf. Theory*, 47(3):1212–1215, 2001. doi:10.1109/18.915688.

42 Daniele Micciancio. Locally dense codes. In *IEEE 29th Conference on Computational Complexity*, pages 90–97. IEEE Computer Society, 2014. doi:10.1109/CCC.2014.17.

43 Ran Raz. A parallel repetition theorem. *SIAM J. Comput.*, 27(3):763–803, 1998. doi:10.1137/S0097539795280895.

44 Oded Regev. Improved inapproximability of lattice and coding problems with preprocessing. *IEEE Trans. Inf. Theory*, 50(9):2031–2037, 2004. doi:10.1109/TIT.2004.833350.

45 Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009. doi:10.1145/1568318.1568324.

46 Oded Regev. The learning with errors problem (invited survey). In *Proceedings of the 25th Annual IEEE Conference on Computational Complexity, CCC 2010*, pages 191–204. IEEE Computer Society, 2010. doi:10.1109/CCC.2010.26.

47 Jacques Stern. Approximating the number of error locations within a constant ratio is np-complete. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 10th International Symposium, AAECC-10, 1993, Proceedings*, volume 673 of *Lecture Notes in Computer Science*, pages 325–331. Springer, 1993. doi:10.1007/3-540-56686-4_54.

48 Alexander Vardy. The intractability of computing the minimum distance of a code. *IEEE Trans. Inf. Theory*, 43(6):1757–1766, 1997. doi:10.1109/18.641542.

Two-Source and Affine Non-Malleable Extractors for Small Entropy

Xin Li
Johns Hopkins University, Baltimore, MD, USA

Yan Zhong
Johns Hopkins University, Baltimore, MD, USA

Abstract

Non-malleable extractors are generalizations and strengthening of standard randomness extractors, that are resilient to adversarial tampering. Such extractors have wide applications in cryptography and have become important cornerstones in recent breakthroughs of explicit constructions of two-source extractors and affine extractors for small entropy. However, explicit constructions of non-malleable extractors appear to be much harder than standard extractors. Indeed, in the well-studied models of two-source and affine non-malleable extractors, the previous best constructions only work for entropy rate $> 2/3$ and $1 - \gamma$ for some small constant $\gamma > 0$ respectively by Li (FOCS' 23).

In this paper, we present explicit constructions of two-source and affine non-malleable extractors that match the state-of-the-art constructions of standard ones for small entropy. Our main results include:

- Two-source and affine non-malleable extractors (over F_2) for sources on n bits with min-entropy $k \geq \log^C n$ and polynomially small error, matching the parameters of standard extractors by Chattopadhyay and Zuckerman (STOC' 16, Annals of Mathematics' 19) and Li (FOCS' 16).
- Two-source and affine non-malleable extractors (over F_2) for sources on n bits with min-entropy $k = O(\log n)$ and constant error, matching the parameters of standard extractors by Li (FOCS' 23).

Our constructions significantly improve previous results, and the parameters (entropy requirement and error) are the best possible without first improving the constructions of standard extractors. In addition, our improved affine non-malleable extractors give strong lower bounds for a certain kind of read-once linear branching programs, recently introduced by Gryaznov, Pudlák, and Talebanfard (CCC' 22) as a generalization of several well studied computational models. These bounds match the previously best-known average-case hardness results given by Chattopadhyay and Liao (CCC' 23) and Li (FOCS' 23), where the branching program size lower bounds are close to optimal, but the explicit functions we use here are different. Our results also suggest a possible deeper connection between non-malleable extractors and standard ones.

2012 ACM Subject Classification Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Randomness Extractors, Non-malleable, Two-source, Affine

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.108

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2404.17013

Funding *Xin Li*: Supported by NSF CAREER Award CCF-1845349 and NSF Award CCF-2127575. *Yan Zhong*: Supported by NSF CAREER Award CCF-1845349.

1 Introduction

Randomness extractors are fundamental objects in the broad area of pseudorandomness. These objects have been studied extensively and found applications in diverse areas such as cryptography, complexity theory, combinatorics and graph theory, and many more.

Informally, randomness extractors are functions that transform imperfect randomness called weak random sources into almost uniform random bits. Originally, the motivation for studying these objects comes from the gap between the requirement of high-quality random bits in various computational and cryptographic applications, and the severe biases in natural random sources. In practice, weak random sources can arise in several different situations. For example, the random bits can become biased and correlated due to the natural process that generates them, or because of the fact that an adversary learns some partial information about a random string in cryptographic applications.

To measure the amount of randomness in a weak random source (a random variable) X, we use the standard definition of *min-entropy*: $H_\infty(X) = \min_{x \in \mathsf{supp}(X)} \log_2(1/\mathbf{Pr}[X = x])$. If $X \in \{0,1\}^n$, we say X is an $(n, H_\infty(X))$-source, or simply an $H_\infty(X)$-source if n is clear from context. We also say X has *entropy rate* $H_\infty(X)/n$. Ideally, one would like to construct deterministic extractors for all (n, k) sources when k is not too small. However, this is well known to be impossible, even if one only desires to extract one bit and k is as large as $n - 1$. Thus, to allow randomness extraction one has to put additional restrictions on the source.

Historically, many different models of randomness extractors have been studied. For example, if one gives the extractor an additional independent short uniform random seed, then there exist extractors that work for any (n, k) source. Such extractors, first introduced by Nisan and Zuckerman [64], are known as *seeded extractors*. These extractors have found wide applications, and by now we have almost optimal constructions (e.g., [62, 42, 35, 34]) after a long line of research.

However, seeded extractors may not be applicable in situations where the short uniform random seed is either not available (e.g., in cryptography) or cannot be simulated by cycling over all possible choices. For these applications, one needs *deterministic extractors* or *seedless extractors*, and many different models have also been studied in this setting. These include for example extractors for independent sources [20, 2, 3, 67, 7, 65, 4, 49, 52, 54, 53, 55, 22, 26, 18, 56, 30, 12, 23, 5, 27, 28, 57, 59, 47, 60], bit fixing sources [21, 46, 39, 66], affine sources [38, 8, 66, 72, 6, 69, 50, 56, 10, 60], samplable sources [70, 71], interleaved sources [68, 18], and small-space sources [45]. We define deterministic extractors below.

▶ **Definition 1.** *Let $\mathcal{X}$ be a family of distribution over $\{0,1\}^n$. A function $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^m$ is a deterministic extractor for $\mathcal{X}$ with error ε if for every distribution $X \in \mathcal{X}$, we have*

$$\mathsf{Ext}(X) \approx_\varepsilon U_m,$$

where U_m stands for the uniform distribution over $\{0,1\}^m$, and $\approx_\varepsilon$ means ε-close in statistical distance. We say Ext is explicit if it is computable by a polynomial-time algorithm.

Among these models, two of the most well-studied are extractors for independent sources and affine sources. This is in part due to their connections to several other areas of interest. For example, extractors for independent sources are useful in distributed computing and cryptography with imperfect randomness [44, 43], and give explicit constructions of Ramsey graphs; while affine sources generalize bit-fixing sources, and extractors for affine sources have applications in exposure-resilient cryptography [21, 46] as well as Boolean circuit lower bounds [31, 37, 48].

Using simple probabilistic arguments, one can show that there exist extractors for two independent (n, k) sources with $k = \log n + O(1)$, which is optimal up to the constant $O(1)$. The first explicit construction of two-source extractors was given by Chor and Goldreich [20], which achieves $k > n/2$. Following a long line of research and several recent breakthroughs,

we now have explicit constructions of two-source extractors for entropy $k \approx 4n/9$ with error $\varepsilon = 2^{-\Omega(n)}$ [47], for entropy $k = \text{polylog}(n)$ with error $\varepsilon = 1/\mathsf{poly}(n)$ [18], and for entropy $k = O(\log n)$ with constant error [60]. Similarly, for affine sources which are uniform distributions over some unknown affine subspace over the vector space F_2^n, [1] one can show the existence of extractors for entropy $k = O(\log n)$, which is also optimal up to the constant $O(1)$. Regarding explicit constructions, we have affine extractors for entropy $k = \delta n$ with error $\varepsilon = 2^{-\Omega(n)}$ for any constant $\delta > 0$ [8, 72, 50], for entropy $k = \text{polylog}(n)$ with error $\varepsilon = 1/\mathsf{poly}(n)$ [56], and for entropy $k = O(\log n)$ with constant error [60].

In the past decade or so, a new kind of extractors, known as *non-malleable extractors*, has gained a lot of attention. These extractors are motivated from cryptographic applications. Informally, the setting is that an adversary can tamper with the inputs to an extractor in some way, and the non-malleable extractor guarantees that the output of the extractor is close to uniform even conditioned on the output of the extractor on the tampered inputs. The most well-studied non-malleable extractors include seeded non-malleable extractors [33], two-source non-malleable extractors [19], and affine non-malleable extractors [15]. These non-malleable extractors have wide applications in cryptography, such as privacy amplification with an active adversary [33] and non-malleable codes [36]. Furthermore, they turn out to have surprising connections to the constructions of standard extractors. Indeed, starting from the work of Li [52] which showed a connection between seeded non-malleable extractors and two-source extractors, these non-malleable extractors have played key roles, and now become important cornerstones in the recent series of breakthroughs that eventually lead to explicit constructions of two-source and affine extractors for asymptotically optimal entropy. In a more recent line of work [41, 17, 61], a special case of affine non-malleable extractors known as *directional affine extractors* is also shown to give strong lower bounds for certain read-once branching programs with linear queries, which generalize both standard read-once branching programs and parity decision trees. Given these applications, non-malleable extractors have become important objects that deserve to be studied on their own. We now define tampering functions and two kinds of non-malleable extractors below.

▶ **Definition 2** (Tampering Function). *For any function $f : S \to S$, We say f has no fixed points if $f(s) \neq s$ for all $s \in S$. For any $n > 0$, let $\mathcal{F}_n$ denote the set of all functions $f : \{0,1\}^n \to \{0,1\}^n$. Any subset of $\mathcal{F}_n$ is a family of tampering functions.*

▶ **Definition 3** ([19]). *A function $\mathsf{2nmExt} : (\{0,1\}^n)^2 \to \{0,1\}^m$ is a (k_1, k_2, ε) two source non-malleable extractor, if it satisfies the following property: Let X, Y be two independent, (n, k_1) and (n, k_2) sources, and $f, g : \{0,1\}^n \to \{0,1\}^n$ be two arbitrary tampering functions such that at least one of them has no fixed point,[2] then*

$$|\mathsf{2nmExt}(X, Y) \circ \mathsf{2nmExt}(f(X), g(Y)) - U_m \circ \mathsf{2nmExt}(f(X), g(Y))| < \varepsilon.$$

▶ **Definition 4** ([15]). *A function $\mathsf{anmExt} : \{0,1\}^n \to \{0,1\}^m$ is a (k, ε) affine non-malleable extractor if for any affine source X with entropy at least k and any affine function $f : \{0,1\}^n \to \{0,1\}^n$ with no fixed point, we have*

$$|\mathsf{anmExt}(X) \circ \mathsf{anmExt}(f(X)) - U_m \circ \mathsf{anmExt}(f(X))| \leq \varepsilon.$$

[1] In this paper we focus on the case where the field is F_2, for larger fields there are affine extractors with better parameters.

[2] We say that x is a fixed point of a function f if $f(x) = x$.

Using the probabilistic method, one can also prove the existence of these non-malleable extractors with excellent parameters. For example, [19] showed that two-source non-malleable extractors exist for (n, k) sources when $k \geq m + \frac{3}{2}\log(1/\varepsilon) + O(1)$ and $k \geq \log n + O(1)$. Similarly, it can be also shown that affine non-malleable extractors exist for entropy $k \geq 2m + 2\log(1/\varepsilon) + \log n + O(1)$.

However, constructing explicit non-malleable extractors appears to be significantly harder than constructing standard extractors, despite considerable effort. Indeed, even for seeded non-malleable extractors, the initial explicit constructions [32, 29, 51] only work for sources with entropy rate $> 1/2$, and it was not until [11] that explicit seeded non-malleable extractors for sources with poly-logarithmic entropy are constructed. After a long line of research [32, 29, 51, 52, 11, 24, 25, 12, 15, 23, 27, 28, 57, 59, 60], an asymptotically optimal seeded non-malleable extractor is finally constructed in [60]. On the other hand, the situation for two-source non-malleable extractors and affine non-malleable extractors is much worse, where the best-known constructions in [60] only achieve entropy $k > 2n/3$ and $k \geq (1-\gamma)n$ for a small constant $\gamma > 0$. This is in sharp contrast to the constructions of standard two-source and affine extractors, where explicit constructions can work for entropy $k = \text{polylog}(n)$ with polynomially small error [18, 56], and for entropy $k = O(\log n)$ with constant error [60].

1.1 Our Results

In this paper, we study two-source and affine non-malleable extractors for small entropy. Our main results give explicit constructions of such non-malleable extractors that essentially match their standard counterparts in the small entropy regime. Specifically, we give explicit two-source and affine non-malleable extractors for $\text{polylog}(n)$ entropy with polynomially small error and for $O(\log n)$ entropy with constant error. We have the following theorems.

▶ **Theorem 5.** *There exists a constant $C > 1$ such that for any $k \geq \log^C n$, there exists an explicit construction of a $(k, k, n^{-\Omega(1)})$ two-source non-malleable extractor with output length $\Omega(k)$.*

▶ **Theorem 6.** *There exists a constant $C > 1$ such that for any $k \geq \log^C n$, there exists an explicit construction of a $(k, n^{-\Omega(1)})$ affine non-malleable extractor with output length $k^{\Omega(1)}$.*

▶ **Theorem 7.** *There exists a constant $c > 1$ such that for any $k \geq c\log n$, there exists an explicit construction of a $(k, k, O(1))$ two-source non-malleable extractor with output length 1.*

▶ **Theorem 8.** *There exists a constant $c > 1$ such that for any $k \geq c\log n$, there exists an explicit construction of a $(k, O(1))$ affine non-malleable extractor with output length 1.*

▶ Remark 9. The output length in the two theorems for entropy $k \geq c\log n$ can be extended to a constant number by using the standard XOR lemma and previous techniques (e.g., those in [56]). Furthermore, our constructions can also be extended to handle multiple tampering functions as in [11]. For simplicity, we omit the details here.

The following tables summarize our results compared to some of the best previous constructions.

Table 1 Prior and current results on two-source non-malleable extractors.

Two-source Non-malleable Extractor	Entropy k_1	Entropy k_2	Output m	Error ε
[11]	$n - n^{\gamma}$	$n - n^{\gamma}$	$n^{\Omega(1)}$	$2^{-n^{\Omega(1)}}$
[59]	$(1-\gamma)n$	$(1-\gamma)n$	$\Omega(n)$	$2^{-\Omega(\frac{n \log\log n}{\log n})}$
[1]	$(\frac{4}{5}+\delta)n$	$\log^C n$	$\Omega(\min\{k_1,k_2\})$	$2^{-\min\{k_1,k_2\}^{\Omega(1)}}$
[60]	$(\frac{2}{3}+\gamma)n$	$k = O(\log n)$	$\Omega(k)$	$2^{-\Omega(k)}$
This work (Theorem 5)	$k \geq \text{polylog}(n)$	$k \geq \text{polylog}(n)$	$\Omega(k)$	$n^{-\Omega(1)}$
This work (Theorem 7)	$O(\log n)$	$O(\log n)$	1	$O(1)$

Table 2 Prior and current results on affine non-malleable extractors.

Affine Non-malleable Extractor	Entropy k	Output m	Error ε
[15]	$n - n^{\delta}$ for some constant $\delta \in (0,1)$	$n^{\Omega(1)}$	$2^{-n^{\Omega(1)}}$
[60]	$(1-\gamma)n$, $\gamma < 1/1000$	$\Omega(n)$	$2^{-\Omega(n)}$
This work (Theorem 6)	$\text{polylog}(n)$	$k^{\Omega(1)}$	$n^{-\Omega(1)}$
This work (Theorem 8)	$O(\log n)$	1	$O(1)$

Our results thus significantly improve the entropy requirement of previous non-malleable extractors. As a comparison, we list below the best-known explicit two-source extractors and affine extractors for small entropy.

Table 3 Best-known results on two-source extractors.

Two-source Extractor	Entropy k	Output m	Error ε
[18]	$\text{polylog}(n)$	1	$n^{-\Omega(1)}$
[63, 56, 13]	$\text{polylog}(n)$	$k^{\Omega(1)}$	$n^{-\Omega(1)}$
[5]	$O(\log n 2^{O(\sqrt{\log\log n})})$	1	$O(1)$
[27]	$O(\log n (\log\log n)^{O(1)})$	1	$O(1)$
[58]	$O(\log n \log\log n)$	1	$O(1)$
[59]	$O(\log n \frac{\log\log n}{\log\log\log n})$	1	$O(1)$
[60]	$O(\log n)$	1	$O(1)$

Table 4 Best-known results on affine extractors.

Affine Extractor	Entropy k	Output m	Error ε
[56]	$\text{polylog}(n)$	$k^{\Omega(1)}$	$n^{-\Omega(1)}$
[10]	$O(\log n \log\log n \log\log\log^6 n)$	1	$O(1)$
[16]	$O(\log n \log\log n \log\log\log^3 n)$	1	$O(1)$
[60]	$O(\log n)$	1	$O(1)$

It can be seen that the parameters of our two-source and affine non-malleable extractors essentially match those of standard two-source and affine extractors for small entropy. We also point out that the error of our non-malleable extractors is the best one can hope for without first improving the error of standard two-source and affine extractors for small entropy, since the non-malleable extractors are stronger versions of extractors, and in particular, they are themselves two-source and affine extractors. Finally, given that our constructions use many of the key components in the constructions of standard extractors, we believe that any future

techniques that improve the error of standard two-source and affine extractors for small entropy (e.g., to negligible error) are also likely applicable to our constructions to get the same improvement on the error of two-source and affine non-malleable extractors.

1.2 Applications to Lower bounds for Read-Once Linear Branching Programs

Our affine non-malleable extractors have applications in proving average-case hardness against read-once linear branching programs (ROLBPs). This computational model was recently introduced by Gryaznov, Pudlák, and Talebanfard [41] as a generalization of several important and well-studied computational models such as decision trees, parity decision trees, and standard read-once branching programs. Roughly, a read-once linear branching program is a branching program that can make linear queries to the input string, while these queries are linearly independent along any path. Formally, we have the following definition.

▶ **Definition 10** (Linear branching program [41]). *A linear branching program on F_2^n is a directed acyclic graph P with the following properties:*

- *There is only one source s in P.*
- *There are two sinks in P, labeled with 0 and 1 respectively.*
- *Every non-sink node v is labeled with a linear function $\ell_v : \mathsf{F}_2^n \to \mathsf{F}_2$. Moreover, there are exactly two outgoing edges from v, one is labeled with 1 and the other is labeled with 0.*

The size of P is the number of non-sink nodes in P. P computes a Boolean function $f : \{0,1\}^n \to \{0,1\}$ in the following way. For every input $x \in \mathsf{F}_2^n$, P follows the computation path by starting from s, and when on a non-sink node v, moves to the next node following the edge with label $\ell_v(x) \in \{0,1\}$. The computation ends when the path ends at a sink, and $f(x)$ is defined to be the label on this sink.

[41] defines two kinds of read-once linear branching programs (ROLBP for short).

▶ **Definition 11** ([41]). *Given any linear branching program P and any node v in P, let Pre_v denote the span of all linear queries that appear on any path from the source to v, excluding the query ℓ_v. Let Post_v denote the span of all linear queries in the subprogram starting at v.*

- *A linear branching program P is weakly read-once if for every inner node v of P, it holds that $\ell_v \notin \mathsf{Pre}_v$.*
- *A linear branching program P is strongly read-once if for every inner node v of P, it holds that $\mathsf{Pre}_v \cap \mathsf{Post}_v = \{0\}$.*

Both kinds of ROLBPs generalize the aforementioned computational models, but weakly read-once linear branching programs (WROLBPs) are more flexible than strongly read-once linear branching programs (SROLBPs). As a result, proving lower bounds for WROLBPs turns out to be much harder than for SROLBPs. Indeed, so far we only have non-trivial lower bounds for SROLBPs. To state our results, we use the following definition.

▶ **Definition 12** ([17]). *For a Boolean function $f : \{0,1\}^n \to \{0,1\}$, let $\mathsf{SROLBP}(f)$ denote the smallest possible size of a strongly read-once linear branching program that computes f, and $\mathsf{SROLBP}_\varepsilon(f)$ denote the smallest possible size of a strongly read-once linear branching program P such that*

$$\mathbf{Pr}_{x \leftarrow_U \mathsf{F}_2^n}[P(x) = f(X)] \geq \frac{1}{2} + \varepsilon.$$

The definition can be adapted to ROBPs naturally.

[41] shows that a stronger version of affine extractors known as *directional affine extractors* give strong average case lower bounds for SROLBPs. They give an explicit construction of directional affine extractors for entropy $k \geq \frac{2n}{3} + c$ with error $\varepsilon \leq 2^{-c}$ for any constant $c > 1$, which also implies exponential average-case hardness for SROLBPs of size up to $2^{\frac{n}{3} - o(n)}$. In a follow-up work, Chattopadhyay and Liao [17] used another kind of extractors known as *sumset extractors* [14] to give an alternative average-case hardness for SROLBPs. In particular, they gave an explicit function Ext such that $\mathsf{SROLBP}_{n^{-\Omega(1)}}(\mathsf{Ext}) \geq 2^{n - \log^{O(1)} n}$. More recently, Li [60] gave an improved sumset extractor which in turn yields an explicit function Ext such that $\mathsf{SROLBP}_{2^{-\Omega(1)}}(\mathsf{Ext}) \geq 2^{n - O(\log n)}$. In these two constructions, the branching program size lower bounds become quite close to optimal (the result of [60] is optimal up to the constant in $O(\cdot)$), while the correlation becomes polynomially large or a constant. Another recent work by Li and Zhong [61] gave explicit directional affine extractor for entropy $k \geq cn(\log\log\log n)^2/\log\log n$ with error $\varepsilon = 2^{-n^{\Omega(1)}}$ for some constant $c > 1$, which implies exponential average-case hardness for SROLBPs of size up to $2^{n - o(n)}$.

For simplicity, we do not define directional affine extractors here, but just mention that directional affine extractors are a special case of affine non-malleable extractors. Hence, our new constructions of affine non-malleable extractors directly imply improved directional affine extractors, which in turn also give average-case hardness for SROLBPs. Specifically, we have the following theorem.

▶ **Theorem 13.** *There exist explicit functions* anmExt_1, anmExt_2 *such that* $\mathsf{SROLBP}_{n^{-\Omega(1)}}(\mathsf{anmExt}_1) \geq 2^{n - \log^{O(1)} n}$ *and* $\mathsf{SROLBP}_{2^{-\Omega(1)}}(\mathsf{anmExt}_2) \geq 2^{n - O(\log n)}$.

These bounds match the previously best-known average-case hardness results for SROLBPs given in [17] and [60], where the branching program size lower bounds are close to optimal, but the explicit functions we use here are different. Specifically, here we use affine non-malleable extractors while [17] and [60] use sumset extractors.

2 Technical Overview

Here we outline the main techniques used in this paper, opting for an informal approach at times for clarity while omitting certain technical details.

We use the standard notation in the literature where a letter with $'$ represents a tampered version. Let f and g denote the tampering functions on X and Y in two-source non-malleable extractors, respectively, and $\mathcal{A}$ be the affine tampering function in affine non-malleable extractors.

Since two-source and affine non-malleable extractors are themselves two-source and affine extractors, our high-level idea is to adapt the constructions of standard extractors for polylogarithmic or logarithmic entropy into the stronger, non-malleable version. Clearly, a direct naive application of standard extractors may not work, since the output on the tampered inputs may be correlated to the output on the original inputs. Below we start with two-source extractors to illustrate our main ideas. Let us first briefly review the constructions of two-source extractors for small entropy. Generally, these extractors are double-layered: the outer layer is a suitable resilient function, which is designed to be an extractor for non-oblivious bit-fixing (NOBF) sources with t-wise independent property for some parameter t. That is, most of the bits are t-wise independently uniform, while the rest of the bits can depend arbitrarily on these bits. Here, the extractor uses a crucial property that bounded independence suffices to work for several resilient functions (or equivalently these functions are *fooled* by bounded independence), such as the derandomized Ajtai-Linial function in [18]

or the Majority function. The inner layer is a transformation that transforms two independent sources into a single NOBF source with the t-wise independent property. This step itself utilizes techniques from seeded non-malleable extractors or correlation breakers, which are functions designed to break correlations between random variables.

To adapt the construction to two-source non-malleable extractors, our first observation is that there is an easy case. Intuitively, this is the case where one input source has large entropy conditioned on the tampered version. For instance, say the source X has high entropy conditioned on every fixing of $X' = f(X) = x'$. Then in the analysis, we can first fix X', and then further fix the tampered output of the extractor, which is now a deterministic function of Y and can be chosen to have a relatively small size. Conditioned on these fixings, we have X and Y are still independent and have good entropy, hence any two-source extractor will give an output that is close to uniform conditioned on the tampered output.

However, it is certainly possible that the above does not hold. For example, the tampering function f can be an injective function, so that conditioned on any fixing of $X' = f(X) = x'$, we have that X is also fixed. In this case, our observation is that X' itself must also have large entropy (since f injective), therefore we can possibly create structures in the distribution of the tampered version as well. Specifically, our strategy is to modify the inner layer of the two-source extractor while essentially using the same outer layer. For simplicity, let us consider extractors with just one bit of output. A standard approach to show the output bit is close to uniform conditioned on the tampered output, is to show that the parity of these two bits is close to uniform. Since the outer extractor is a resilient function, this suggests to look at the parity of two copies of resilient functions on two correlated distributions.

Now another crucial observation behind our construction is that just like in the construction of standard two-source extractors, for certain resilient functions, the parity of two copies of such functions is still fooled by bounded independence. Thus, if in the inner layer, we can create structures such that the *joint distribution* of the NOBF source and the tampered version has the t-wise independent property, then we will be able to show that the extractor is non-malleable. Note that we are now in the case where the tampered sources also have high entropy, which works in our favor since achieving t-wise independence requires a certain amount of entropy. However, we cannot simply use previous techniques since the tampered sources are correlated with the original sources. Therefore, we appropriately modify previous constructions of correlation breakers to ensure the t-wise independent property in the joint distribution.

Finally, in the actual analysis, we are not guaranteed to be in either case; and it may happen that for some x', conditioned on $X' = x'$ we have that X has large entropy, while for others conditioned on $X' = x'$ we have that X has small entropy. The analysis thus needs a careful interpolation between different cases in terms of a convex combination of subsources. We now elaborate with more details on each of these aspects below.

First we give some notation that will help with our presentation.

▶ **Definition 14** (t-non-malleable (k, ε) seeded extractor)**.** *A function* $\mathsf{nmExt} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *is a t-non-malleable (k, ε) extractor if it satisfies the following property: if X is a (n, k)-source and Y is uniform on $\{0,1\}^d$, and $f_1, \cdots, f_t$ are arbitrary functions from d bits to d bits with no fixed point, then*

$$(\mathsf{nmExt}(X, Y), \mathsf{nmExt}(X, f_1(Y)), \cdots, \mathsf{nmExt}(X, f_t(Y)), Y) \approx_\varepsilon (U_m, \mathsf{nmExt}(X, f_1(Y)), \cdots, Y).$$

We say a distribution or a source X on n bits is (q, t, γ) independent if there exists a subset $S \subseteq [n]$ with $|S| \leq q$ such that if we consider the bits of X in $[n] \setminus S$, then every t bits are γ-close to uniform.

A t-non-malleable (k, ε) seeded extractor nmExt with seed length d can be used to generate a (q, t, γ) source from a source X with entropy at least k in the following way: cycle over all possible seeds i, and for each one output a bit $\mathsf{nmExt}(X, i)$. The output is now $(\sqrt{\varepsilon}D, t+1, t\sqrt{\varepsilon})$ independent with $D = 2^d$.

2.1 Taking the parity of two resilient functions

A Boolean function on n variables is a resilient function if it is nearly balanced, and no small coalition can have a significant influence on the output of the function. Such functions are equivalent to extractors for NOBF sources. The resilient functions that have played a key role in the recent advancement of extractors are the derandomized Ajtai-Linial function in [18] and the Majority function. The former is a monotone AC^0 function, that is fooled by polylog(n)-wise independence, while the latter is a threshold function that can be fooled by constant-wise independence.

It is not hard to show that the parity of two independent copies of resilient functions is still a resilient function. What is left to show is that such a parity can also be fooled by bounded independence. When the resilient function is in AC^0, we observe that the parity of two such functions is also in AC^0, because the parity of two bits can be written as a constant size AC^0 circuit. Therefore, the parity of two derandomized Ajtai-Linial functions is still in AC^0, and can be fooled by polylog(n)-wise independence by Braverman's celebrated result [9] on bounded independence fooling AC^0 circuits, together with some standard techniques.

To show that constant-wise independence fools the parity of two Majority functions, we use the work of Gopalan, O'Donnell, Wu, and Zuckerman [40], which shows that constant-wise independence fools any function of halfspaces under product distributions, as long as the function can be implemented as a constant size circuit. In our case, this clearly holds since we are just taking the parity of two Majority functions. Using the XOR lemma and previous techniques (e.g., those in [56], our construction can also be extended to output a constant number of bits.

2.2 Generating NOBF sources from the inputs and its tampered counterparts

We want to construct a function such that when the tampered sources have sufficient entropy, the joint distribution of the generated bits from the input sources and the tampered sources is (q, t, γ) independent for some suitable parameters q, t, and γ.

The standard approach for two-source extractors, as introduced in [18], is to first apply a seeded non-malleable extractor to one source, say Y, and then use another source X to sample a small number of bits from the output. However, in our case, this black-box approach does not work since the tampered sources are correlated with the original inputs. Therefore, we have to create some kind of difference between the tampered sources and the original sources, which will enable us to get the desired (q, t, γ) independent property.

To achieve this, we dig into the constructions of seeded non-malleable extractors and existing two-source non-malleable extractors, which roughly go as follows. First, one uses an *advice generator* to create a short string that is different from the tampered version with high probability. Then, conditioned on the fixing of the advice strings, one can argue that the two sources are still independent and have sufficient entropy. At this point one uses a *correlation breaker with advice*, together with the advice strings to compute the output, which is guaranteed to have the non-malleable property. However, the steps of generating advice and subsequent application of correlation breakers require the sources to have very large entropy (e.g., at least 2/3), which is the main reason that previous two-source non-malleable extractors can only work for large entropy.

To get around this barrier, our approach is to first apply a standard seeded extractor to one source Y, and output say $\Omega(k)$ bits where k is the entropy. By cycling over all possible seeds, we potentially get a matrix with $D = 2^d$ rows where d is the seed length of the seeded extractor. We then use the other source X to sample a small number ($\mathsf{poly}(n)$) of rows from the output. Now, again a standard argument implies that most of the rows are close to uniform. Since we are in the case where the tampered sources X', Y' have sufficient entropy, this is also true for the tampered version. Note that we haven't achieved the (q, t, γ) independent property yet. Our next step is to generate advice from the original sources and the tampered sources. However, in the low-entropy regime, it is hard to generate a single advice for the input sources and the tampered version – the advice generator requires generating uniform seeds to sample from an encoding of the inputs and it is hard to do so from a slice of the sources which could have zero entropy. Therefore, we generate advice from each row. We can then append the index of this row to the advice. This ensures that the advice strings are both different from the tampered version, and also different between different rows. Now, we can apply existing constructions of correlation breakers with advice, which will ensure that for any t rows in the combined matrix from the original sources and the tampered sources, as long as all these rows have high entropy initially, the joint distribution of the final outputs from the correlation breaker is γ-close to uniform.

However, there are additional tricky issues with this approach. First, the correlation breaker requires two independent sources to work, while in our case the outputs in the matrices are already functions of both X and Y. Second, the analysis of the correlation breaker usually requires fixing the advice strings first and arguing that the sources still have sufficient entropy, but now since the matrices have $\mathsf{poly}(n)$ rows and the entropy of the sources is just $k = \mathrm{polylog}(n)$, or even $k = O(\log n)$, if we fix all the advice strings then conditioned on the fixing the sources may not have any entropy left. Finally, the set of "good" rows (the rows that are close to uniform after the sampling using X) in the matrices depends on the source X and Y, and after we fix the advice strings in the analysis, X and Y may have become different, and this could potentially change the set of "good" rows in the first place.

To solve these issues, we use an argument similar to that in [55]. The idea is that since eventually we only need (q, t, γ) independence, in the analysis we can just focus on every subset of t rows from the good rows. In particular, we can set t and the entropy k appropriately, i.e., t is relatively small compared to k. This is because we only need $t = \mathrm{polylog}(n)$ to apply the derandomized Ajtai-Linial in [18] and $t = O(1)$ to apply the Majority function. Now in the analysis, notice that the process of sampling using the source X basically corresponds to $\mathsf{Ext}(Y, \mathsf{Ext}'(X, i))$ where $\mathsf{Ext}, \mathsf{Ext}'$ are two seeded extractors. Thus when t is small, for any subset T with $|T| = t$, we can first fix all $\mathsf{Ext}'(X, i)$ with $i \in T$. By restricting the size of $\mathsf{Ext}'(X, i)$, X still has sufficient entropy conditioned on these fixings, and now the t rows of the outputs $\mathsf{Ext}(Y, \mathsf{Ext}'(X, i))$ are deterministic functions of Y, while X and Y are still independent. By restricting the size of the advice strings, we can preserve the above properties when the analysis fixes the advice strings and goes into the correlation breaker. Finally, as in the analysis in [55], the final error pays a price of a $\mathsf{poly}(n)^t$ factor from a union bound on all possible subsets of size t, which is still fine as long as we set $k \gg t \log n$ and use seeded extractors with error $2^{-\Omega(k)}$ in the correlation breaker.

2.3 Convex combination of subsources

In the above two subsections, we dealt with the case where the tampered sources have sufficient entropy. We now sketch the analysis for the general case.

Let 2nmExt be the two-source non-malleable extractor which works if both X and X' have entropy at least k_x, and both Y and Y' have entropy at least k_y, with error $\varepsilon/2$ and output length m. Now assume X has min-entropy $2k_x + \log(2/\varepsilon)$, and Y has min-entropy $2k_y + \log(2/\varepsilon)$. Further assume without loss of generality that both X and Y are flat sources, i.e., uniform distributions over some unknown subset. The analysis goes by considering the "heavy" elements in the tampered sources X' and Y'. Specifically, for any $x' \in \{0,1\}^n$ and $y' \in \{0,1\}^n$, we consider the pre-image size of $X' = x'$ and $Y' = y'$. If one of them is large, say without loss of generality that the pre-image size of $X' = x'$ is at least 2^{k_x}, then $H_\infty(X|X' = x) \geq k_x$. We can first fix $X' = x'$ and then $\mathsf{2nmExt}(x', Y')$, which is a deterministic function of Y now conditioned on the fixing of $X' = x'$. Since $\mathsf{2nmExt}(x', Y')$ is short compared to $H_\infty(Y)$, conditioned on these fixings we have that X and Y are still independent and have sufficient entropy, so $\mathsf{2nmExt}(X, Y)$ is close to uniform because 2nmExt is itself a two-source extractor. Note that we have already fixed $\mathsf{2nmExt}(x', Y')$, and thus 2nmExt is indeed non-malleable in this case.

Next, consider the set of all the x' whose pre-image size under f is at most 2^{k_x}, and call it BAD_X. If the total probability mass of these x' is at most $\varepsilon/2$, then we can just ignore them (and the corresponding x in the support of X) since this only adds an extra error of $\varepsilon/2$. Similarly, we can also ignore the set of all the y' whose pre-image size under g is at most 2^{k_y} (call it BAD_Y), if the total probability mass of these y' is at most $\varepsilon/2$. In either case, we are done. Otherwise, the subsource of X' formed by all the $x' \in \mathrm{BAD}_X$ has min-entropy at least $-\log(2^{k_x}/(\varepsilon 2^{2k_x}/\varepsilon)) = k_x$, and the corresponding subsource of X has min-entropy at least $2k_x$. Similarly, the subsource of Y' formed by all the $y' \in \mathrm{BAD}_Y$ has min-entropy at least k_y, and the corresponding subsource of Y has min-entropy at least $2k_y$. In this case, both sources and their tampered versions have sufficient entropy, thus by the analysis before, 2nmExt is also a non-malleable extractor.

Since X is just a convex combination of subsources $(X \mid X' = x' \in \mathrm{BAD}_x)$ and $\{(X \mid X' = x' \notin \mathrm{BAD}_x)\}$, and the same is true for Y, the correctness of 2nmExt follows.

Finally, we note that we can modify the two-source non-malleable extractor to output $k^{\Omega(1)}$ bits, by using a similar approach based on the XOR lemma as in [56]. Then, since the two-source non-malleable extractor is strong, we can further apply a standard seeded extractor to increase the output length to $\Omega(k)$.

2.4 Affine non-malleable extractors

Our construction of affine non-malleable extractors roughly follows the same ideas. The difference is that now we do not have access to two independent sources, but the affine source itself has nice structures and the tampering function is affine. Thus, by using appropriate linear seeded extractors as in previous works, certain parts of the affine source behave like independent sources. Therefore, we can suitably adapt our construction of two-source non-malleable extractors to affine non-malleable extractors. One particularly nice property of affine sources is that when applying a strong linear seeded extractor on an affine source, the output on most seeds *is uniform*. This implies that we can generate from an affine source a somewhere random source with no error. In the two-source case, we cannot analyze the generation of NOBF source directly from the definition of the correlation breaker (and have to resort to additional techniques as mentioned in previous paragraph 2.2) due to the error of the somewhere random source. In the affine case, there is no such concern. Therefore, we can argue that we obtain a NOBF source directly from the definition of affine correlation breaker, as in prior works on affine extractors for small entropy (e.g., [10]).

3 Conclusion and Open Problems

In this paper, we significantly improved constructions of two-source and affine non-malleable extractors, and our constructions essentially match standard extractors in the regime of small entropy. We note that any future improvement of extractors for NOBF sources (e.g., improvement in the error) can also translate into improvements of our two-source and affine non-malleable extractors. Furthermore, our results suggest that there may be a deeper connection between standard extractors and their non-malleable counterparts, since their constructions and parameters appear quite similar. In particular, previous works have extensively used non-malleable extractors to construct standard extractors, but is it possible that the reverse direction may also be true? That is, can one also use standard extractors to construct non-malleable extractors?

References

1 Divesh Aggarwal, Eldon Chung, and Maciej Obremski. Extractors: Low entropy requirements colliding with non-malleability. In *Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part II*, pages 580–610, Berlin, Heidelberg, 2023. Springer-Verlag.

2 Boaz Barak, R. Impagliazzo, and Avi Wigderson. Extracting randomness using few independent sources. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 384–393, 2004.

3 Boaz Barak, Guy Kindler, Ronen Shaltiel, Benny Sudakov, and Avi Wigderson. Simulating independence: New constructions of condensers, Ramsey graphs, dispersers, and extractors. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 1–10, 2005.

4 Boaz Barak, Anup Rao, Ronen Shaltiel, and Avi Wigderson. 2 source dispersers for $n^{o(1)}$ entropy and Ramsey graphs beating the Frankl-Wilson construction. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, 2006.

5 Avraham Ben-Aroya, Dean Doron, and Amnon Ta-Shma. An efficient reduction from two-source to non-malleable extractors: Achieving near-logarithmic min-entropy. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 1185–1194, New York, NY, USA, 2017. Association for Computing Machinery.

6 Eli Ben-Sasson and Swastik Kopparty. Affine dispersers from subspace polynomials. *SIAM J. Comput.*, 41(4):880–914, 2012. doi:10.1137/110826254.

7 Jean Bourgain. More on the sum-product phenomenon in prime fields and its applications. *International Journal of Number Theory*, 1:1–32, 2005.

8 Jean Bourgain. On the construction of affine extractors. *GAFA Geometric And Functional Analysis*, 17:33–57, January 2007. doi:10.1007/s00039-007-0593-z.

9 Mark Braverman. Polylogarithmic independence fools ac0 circuits. *Journal of the ACM*, 57(5), 2010.

10 Eshan Chattopadhyay, Jesse Goodman, and Jyun-Jie Liao. Affine extractors for almost logarithmic entropy. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 622–633. IEEE, 2021.

11 Eshan Chattopadhyay, Vipul Goyal, and Xin Li. Non-malleable extractors and codes, with their many tampered extensions. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 285–298, New York, NY, USA, 2016. Association for Computing Machinery.

12 Eshan Chattopadhyay and Xin Li. Explicit non-malleable extractors, multi-source extractors and almost optimal privacy amplification protocols. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science*, 2016.

13 Eshan Chattopadhyay and Xin Li. Explicit non-malleable extractors, multi-source extractors, and almost optimal privacy amplification protocols. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 158–167, 2016. doi:10.1109/FOCS.2016.25.

14 Eshan Chattopadhyay and Xin Li. Extractors for sumset sources. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC, Cambridge, MA, USA, June 18-21, 2016*, pages 299–311. ACM, 2016. doi:10.1145/2897518.2897643.

15 Eshan Chattopadhyay and Xin Li. Non-malleable codes and extractors for small-depth circuits, and affine functions. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1171–1184. ACM, 2017.

16 Eshan Chattopadhyay and Jyun-Jie Liao. Extractors for sum of two sources. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1584–1597. ACM, 2022. doi:10.1145/3519935.3519963.

17 Eshan Chattopadhyay and Jyun-Jie Liao. Hardness against linear branching programs and more. In *Proceedings of the Conference on Proceedings of the 38th Computational Complexity Conference*, CCC '23, Dagstuhl, DEU, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

18 Eshan Chattopadhyay and David Zuckerman. Explicit two-source extractors and resilient functions. *Annals of Mathematics*, 189(3):653–705, 2019.

19 Mahdi Cheraghchi and Venkatesan Guruswami. Non-malleable coding against bit-wise and split-state tampering. In *TCC*, pages 440–464, 2014.

20 Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, 17(2):230–261, 1988.

21 Benny Chor, Oded Goldreich, Johan Hastad, Joel Friedman, Steven Rudich, and Roman Smolensky. The bit extraction problem of t-resilient functions (preliminary version). In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 396–407, 1985.

22 Gil Cohen. Local correlation breakers and applications to three-source extractors and mergers. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science*, 2015.

23 Gil Cohen. Making the most of advice: New correlation breakers and their applications. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science*, 2016.

24 Gil Cohen. Non-malleable extractors - new tools and improved constructions. In *Proceedings of the 31st Annual IEEE Conference on Computational Complexity*, 2016.

25 Gil Cohen. Non-malleable extractors with logarithmic seeds. Technical Report TR16-030, ECCC, 2016.

26 Gil Cohen. Two-source dispersers for polylogarithmic entropy and improved ramsey graphs. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 278–284. ACM, 2016. doi:10.1145/2897518.2897530.

27 Gil Cohen. Two-source extractors for quasi-logarithmic min-entropy and improved privacy amplification protocols. Technical Report TR16-114, ECCC: Electronic Colloquium on Computational Complexity, 2016.

28 Gil Cohen. Towards optimal two-source extractors and ramsey graphs. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1157–1170. ACM, 2017. doi:10.1145/3055399.3055429.

29 Gil Cohen, Ran Raz, and Gil Segev. Non-malleable extractors with short seeds and applications to privacy amplification. *SIAM Journal on Computing*, 43(2):450–476, 2014.

30 Gil Cohen and Leonard Schulman. Extractors for near logarithmic min-entropy. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science*, 2016.

31 Evgeny Demenkov and Alexander Kulikov. An elementary proof of 3n-o(n) lower bound on the circuit complexity of affine dispersers. In *Proceedings of the 36th international conference on Mathematical foundations of computer science*, pages 256–265, 2011.

32 Yevgeniy Dodis, Xin Li, Trevor D. Wooley, and David Zuckerman. Privacy amplification and nonmalleable extractors via character sums. *SIAM Journal on Computing*, 43(2):800–830, 2014.

33 Yevgeniy Dodis and Daniel Wichs. Non-malleable extractors and symmetric key cryptography from weak secrets. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 601–610, 2009.

34 Zeev Dvir, Swastik Kopparty, Shubhangi Saraf, and Madhu Sudan. Extensions to the method of multiplicities, with applications to kakeya sets and mergers. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, 2009.

35 Zeev Dvir and Avi Wigderson. Kakeya sets, new mergers and old extractors. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, 2008.

36 Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. In *ICS*, pages 434–452, 2010.

37 Magnus Gausdal Find, Alexander Golovnev, Edward A. Hirsch, and Alexander S. Kulikov. A better-than-3n lower bound for the circuit complexity of an explicit function. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 89–98, 2016. doi:10.1109/FOCS.2016.19.

38 Ariel Gabizon and Ran Raz. Deterministic extractors for affine sources over large fields. *Combinatorica*, 28(4):415–440, 2008. doi:10.1007/s00493-008-2259-3.

39 Ariel Gabizon, Ran Raz, and Ronen Shaltiel. Deterministic extractors for bit-fixing sources by obtaining an independent seed. *SIAM J. Comput.*, 36(4):1072–1094, 2006. doi:10.1137/S0097539705447049.

40 Parikshit Gopalan, Ryan O'Donnell, Yi Wu, and David Zuckerman. Fooling functions of halfspaces under product distributions. In *2010 IEEE 25th Annual Conference on Computational Complexity*, pages 223–234, 2010. doi:10.1109/CCC.2010.29.

41 Svyatoslav Gryaznov, Pavel Pudlák, and Navid Talebanfard. Linear Branching Programs and Directional Affine Extractors. In *37th Computational Complexity Conference (CCC 2022)*, volume 234, pages 4:1–4:16, 2022.

42 V. Guruswami, C. Umans, and S. Vadhan. Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. *Journal of the ACM*, 56:1–34, 2009.

43 Yael Kalai, Xin Li, and Anup Rao. 2-source extractors under computational assumptions and cryptography with defective randomness. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 617–628, 2009.

44 Yael Tauman Kalai, Xin Li, Anup Rao, and David Zuckerman. Network extractor protocols. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 654–663, 2008.

45 Jesse Kamp, Anup Rao, Salil P. Vadhan, and David Zuckerman. Deterministic extractors for small-space sources. *Journal of Computer and System Sciences*, 77:191–220, 2011. doi:10.1016/j.jcss.2010.06.014.

46 Jesse Kamp and David Zuckerman. Deterministic Extractors for Bit-Fixing Sources and Exposure-Resilient Cryptography. *Siam Journal on Computing*, 36:1231–1247, 2007. doi:10.1137/S0097539705446846.

47 Mark Lewko. An explicit two-source extractor with min-entropy rate near 4/9. *Mathematika*, 65(4):950–957, 2019. doi:10.1112/S0025579319000238.

48 Jiatu Li and Tianqi Yang. 3.1n - o(n) circuit lower bounds for explicit functions. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 1180–1193, New York, NY, USA, 2022. Association for Computing Machinery.

49 Xin Li. Improved constructions of three source extractors. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity*, pages 126–136, 2011.

50 Xin Li. A new approach to affine extractors and dispersers. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity, CCC*, 2011.

51 Xin Li. Design extractors, non-malleable condensers and privacy amplification. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing*, pages 837–854, 2012.

52 Xin Li. Non-malleable extractors, two-source extractors and privacy amplification. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, 2012.

53 Xin Li. Extractors for a constant number of independent sources with polylogarithmic min-entropy. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science*, pages 100–109, 2013.

54 Xin Li. New independent source extractors with exponential improvement. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*, pages 783–792, 2013.

55 Xin Li. Three-source extractors for polylogarithmic min-entropy. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 863–882, Los Alamitos, CA, USA, October 2015. IEEE Computer Society.

56 Xin Li. Improved two-source extractors, and affine extractors for polylogarithmic entropy. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 168–177. IEEE Computer Society, 2016.

57 Xin Li. Improved non-malleable extractors, non-malleable codes and independent source extractors. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing*, 2017.

58 Xin Li. Improved non-malleable extractors, non-malleable codes and independent source extractors. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 1144–1156, New York, NY, USA, 2017. Association for Computing Machinery.

59 Xin Li. Non-malleable extractors and non-malleable codes: Partially optimal constructions. In Amir Shpilka, editor, *34th Computational Complexity Conference, CCC 2019, July 18-20, 2019, New Brunswick, NJ, USA*, volume 137 of *LIPIcs*, pages 28:1–28:49. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CCC.2019.28.

60 Xin Li. Two source extractors for asymptotically optimal entropy, and (many) more. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, 2023.

61 Xin Li and Yan Zhong. Explicit directional affine extractors and improved hardness for linear branching programs. Technical report, Arxiv, 2023. arXiv:2304.11495.

62 C. J. Lu, Omer Reingold, Salil Vadhan, and Avi Wigderson. Extractors: Optimal up to constant factors. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 602–611, 2003.

63 Raghu Meka. Explicit resilient functions matching ajtai-linial. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 1132–1148, USA, 2017. Society for Industrial and Applied Mathematics.

64 Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996.

65 Anup Rao. Extractors for a constant number of polynomially small min-entropy independent sources. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, 2006.

66 Anup Rao. Extractors for low-weight affine sources. In *Proc. of the 24th CCC*, 2009.

67 Ran Raz. Extractors with weak random seeds. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 11–20, 2005.

68 Ran Raz and Amir Yehudayoff. Multilinear formulas, maximal-partition discrepancy and mixed-sources extractors. *Journal of Computer and System Sciences*, 77:167–190, 2011. doi:10.1016/j.jcss.2010.06.013.

69 Ronen Shaltiel. Dispersers for affine sources with sub-polynomial entropy. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science*, 2011.

70 Luca Trevisan and Salil P. Vadhan. Extracting Randomness from Samplable Distributions. In *IEEE Symposium on Foundations of Computer Science*, pages 32–42, 2000. doi:10.1109/SFCS.2000.892063.

71 Emanuele Viola. Extractors for circuit sources. *SIAM Journal on Computing*, 43(2):655–672, 2014.

72 Amir Yehudayoff. Affine extractors over prime fields. *Combinatorica*, 31(2):245–256, 2011.

Better Decremental and Fully Dynamic Sensitivity Oracles for Subgraph Connectivity

Yaowei Long
University of Michigan, Ann Arbor, MI, USA

Yunfan Wang
Tsinghua University, Beijing, China

Abstract

We study the *sensitivity oracles problem for subgraph connectivity* in the *decremental* and *fully dynamic* settings. In the fully dynamic setting, we preprocess an n-vertices m-edges undirected graph G with n_{off} deactivated vertices initially and the others are activated. Then we receive a single update $D \subseteq V(G)$ of size $|D| = d \leq d_\star$, representing vertices whose states will be switched. Finally, we get a sequence of queries, each of which asks the connectivity of two given vertices u and v in the activated subgraph. The decremental setting is a special case when there is no deactivated vertex initially, and it is also known as the *vertex-failure connectivity oracles* problem.

We present a better deterministic vertex-failure connectivity oracle with $\widehat{O}(d_\star m)$ preprocessing time, $\widetilde{O}(m)$ space, $\widetilde{O}(d^2)$ update time and $O(d)$ query time, which improves the update time of the previous almost-optimal oracle [14] from $\widehat{O}(d^2)$ to $\widetilde{O}(d^2)$.

We also present a better deterministic fully dynamic sensitivity oracle for subgraph connectivity with $\widehat{O}(\min\{m(n_{\text{off}} + d_\star), n^\omega\})$ preprocessing time, $\widetilde{O}(\min\{m(n_{\text{off}} + d_\star), n^2\})$ space, $\widetilde{O}(d^2)$ update time and $O(d)$ query time, which significantly improves the update time of the state of the art [9] from $\widetilde{O}(d^4)$ to $\widetilde{O}(d^2)$. Furthermore, our solution is even almost-optimal assuming popular fine-grained complexity conjectures.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases connectivity, sensitivity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.109

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2402.09150

Acknowledgements We thank Thatchaphol Saranurak for helpful discussions.

1 Introduction

We study the *sensitivity oracles problem for subgraph connectivity* in the *decremental* and *fully dynamic* settings, which is one of the fundamental dynamic graph problems in undirected graphs. In the fully dynamic setting, this problem has three phases. In the preprocessing phase, given an integer $d_\star$, we preprocess an n-vertices m-edges undirected graph $G = (V, E)$ in which some vertices are *activated*, called *on-vertices* and denoted by V_{on}, while the others are *deactivated*, called *off-vertices* and denoted by V_{off}. We let $n_{\text{on}} = |V_{\text{on}}|$ and $n_{\text{off}} = |V_{\text{off}}|$ denote the number of initial on-vertices and off-vertices respectively. In the update phase, we will receive a set $D \subseteq V$ with $|D| = d \leq d_\star$, representing vertices whose states will be switched, and we update the oracle. In the subsequent query phase, let $V_{\text{new}} = (V_{\text{on}} \setminus D) \cup (V_{\text{off}} \cap D)$ denote the activated vertices after the update. Each query will give a pair of vertices $u, v \in V_{\text{new}}$ and ask the connectivity of u and v in the new activated subgraph $G[V_{\text{new}}]$. The decremental setting is a special case where there is no off-vertices initially, i.e. V_{off} is empty.

The decremental version of this problem is also called the *vertex-failure connectivity oracles* problem, which has been studied extensively, e.g. [4, 5, 18, 17, 14, 13], and its complexity was well-understood up to subpolynomial factors. Specifically, a line of works by

Duan and Pettie [4, 5] started the study on vertex-failure connectivity oracles for general $d_\star$, and they finally gave a deterministic oracle with $\widetilde{O}(mn)$ preprocessing time, $\widetilde{O}(md_\star)$ space, $\widetilde{O}(d^3)$ update time and $O(d)$ query time. Following [4, 5], Long and Saranurak [14] presented an improved solution with $\widehat{O}(m) + \widetilde{O}(md_\star)$ preprocessing time, $\widetilde{O}(m)$ space, $\widehat{O}(d^2)$ update time and $O(d)$ query time[1], which is optimal up to subpolynomial factors because matching (conditional) lower bounds for all the four complexity measurements were shown in [7, 5, 14]. We refer to Table 1 for more solutions to this problem (for example, Kosinas [13] proposed a simple and practical algorithm using a conceptually different approach). However, there are still relatively large subpolynomial overheads on the current almost-optimal upper bounds (i.e., on the preprocessing time and update time of the LS-oracle), so a natural question is whether we can improve them:

Can we design an almost-optimal deterministic vertex-failure connectivity oracle with only polylogarithmic overheads on the update time, or even on all complexity bounds?

The fully dynamic sensitivity oracles problem for subgraph connectivity was studied by [8, 9]. Henzinger and Neumann [8] showed a black-box reduction from the decremental setting. Plugging in the almost-optimal decremental algorithm of [14], this reduction leads to a fully dynamic sensitivity oracle with $\widehat{O}(n_{\text{off}}^2 m) + \widetilde{O}(d_\star n_{\text{off}}^2 m)$ preprocessing time, $\widetilde{O}(n_{\text{off}}^2 m)$ space, $\widehat{O}(d^4)$ update time and $O(d^2)$. Hu, Kosinas and Polak [9] studied this problem from an equivalent but different perspective called *connectivity oracles for predictable vertex failures*, which gave a solution with $\widetilde{O}((n_{\text{off}} + d_\star)m)$ preprocessing time, $\widetilde{O}((n_{\text{off}} + d_\star)m)$ space, $\widetilde{O}(d^4)$ update time and $O(d)$ query time[2]. Despite the efforts, there are still gaps between the upper bounds of the fully dynamic setting and the decremental setting (except the query time). Naturally, one may have the following question:

Can we match the upper bounds in the fully dynamic and decremental settings or show separations between them for all the four measurements?

Notably, [9] showed a conditional lower bound $\widehat{\Omega}((n_{\text{off}} + d_\star)m)$ on the preprocessing time[3], which separated two settings at the preprocessing time aspect. However, the right complexity bounds are still not clear for the space and the update time. In particular, given that two different approaches [8, 9] both showed upper bounds around d^4 for update time, it is interesting to identify if this is indeed a barrier or it just happened accidentally. Furthermore, we note that it seems hard to improve the update time following either of these two approaches, because the black-box reduction by [8] has been plugged in the almost-optimal decremental oracle, and the fully dynamic oracle by [9] generalizes the decremental oracle by [13], where the latter already has $\widetilde{O}(d^4)$ update time.

1.1 Our Results

We give a partially affirmative answer to the first question and answer the second question affirmatively by the following results.

---

[1] Throughout the paper, we use $\widetilde{O}(\cdot)$ to hide a polylog(n) factor and use $\widehat{O}(\cdot)$ to hide a $n^{o(1)}$ factor.

[2] In [9], they modeled the problem using slightly different parameters, and here we describe their bounds using our parameters. Basically, they defined a parameter d' (named d in their paper) in the preprocessing phase and η in the update phase. Their η is equivalent to our d. Besides, fixing $d_\star$ and n_{off}, our input instances can be reduced to theirs with d' at most $n_{\text{off}} + d_\star$. In the other direction, fixing d', their input instances can be reduced to ours with $d_\star + n_{\text{off}}$ at most $3d'$. Furthermore, their space complexity was not specified, but to our best knowledge, it should be roughly proportional to their preprocessing time.

[3] This lower bound is obtained from input instances with $n_{\text{off}} = \Theta(n)$ and m is roughly linear to n.

The Decremental Setting

We show a better vertex-failure connectivity oracle by improving the $\widehat{O}(d^2)$ update time of the current almost-optimal solution [14] to $\widetilde{O}(d^2)$. See Corollary 8 for a detailed version of Theorem 1.

▶ **Theorem 1.** *There exists a deterministic vertex-failure connectivity oracle with $\widehat{O}(m) + \widetilde{O}(d_\star m)$ preprocessing time, $\widetilde{O}(m)$ space, $\widetilde{O}(d^2)$ update time and $O(d)$ query time.*

Same as all the previous vertex-failure connectivity oracles, we can substitute all the m factors in Theorem 1 with $\bar{m} = \min\{m, n(d_\star + 1)\}$ using a standard sparsification by Nagamochi and Ibaraki [15] at a cost of an additional $O(m)$ preprocessing time.

Table 1 Complexity of known vertex-failure connectivity oracles. All the m factors can be replaced by $\bar{m} = \min\{m, n(d_\star + 1)\}$ at a cost of an additional $O(m)$ preprocessing time. The randomized algorithms are all Monte Carlo. The notation $\bar{O}(\cdot)$ hides a polyloglog(n) factor.

	Det./ Rand.	Space	Preprocessing	Update	Query
Block trees, SQRT trees, and [10] only when $d_\star \leq 3$	Det.	$O(n)$	$\tilde{O}(m)$	$O(1)$	$O(1)$
Duan & Pettie [4] for $c \geq 1$	Det.	linear in preprocessing time	$\tilde{O}(md_\star^{1-\frac{2}{c}} n^{\frac{1}{c}-\frac{1}{c\log(2d_\star)}})$	$\tilde{O}(d^{2c+4})$	$O(d)$
Duan & Pettie [5]	Det.	$O(md_\star \log n)$	$O(mn \log n)$	$O(d^3 \log^3 n)$	$O(d)$
	Rand.	$O(m \log^6 n)$	$O(mn \log n)$	$\bar{O}(d^2 \log^3 n)$ w.h.p.	$O(d)$
Brand & Saranurak [18]	Rand.	$O(n^2)$	$O(n^\omega)$	$O(d^\omega)$	$O(d^2)$
Pilipczuk et al. [17]	Det.	$m2^{2^{O(d_\star)}}$	$mn^2 2^{2^{O(d_\star)}}$	$2^{2^{O(d_\star)}}$	$2^{2^{O(d_\star)}}$
	Det.	$n^2 \text{poly}(d_\star)$	$\text{poly}(n)2^{O(d_\star \log d_\star)}$	$\text{poly}(d_\star)$	$\text{poly}(d_\star)$
Long & Saranurak [14]	Det.	$O(m \log^3 n)$	$O(mn \log n)$	$\bar{O}(d^2 \log^3 n \log^4 d)$	$O(d)$
	Det.	$O(m \log^* n)$	$\hat{O}(m) + \tilde{O}(d_\star m)$	$\hat{O}(d^2)$	$O(d)$
Kosinas [13]	Det.	$O(d_\star m \log n)$	$O(d_\star m \log n)$	$O(d^4 \log n)$	$O(d)$
This paper	Det.	$O(m \log^3 n)$	$\widehat{O}(m) + O(d_\star m \log^3 n)$	$O(d^2(\log^7 n + \log^5 n \log^4 d))$	$O(d)$

We emphasize that our result is a strict improvement on [14]. In addition to the improvement on update time, our algorithm also improves the hidden subpolynomial overheads on the preprocessing time. We achieve this by giving a new construction algorithm of the *low degree hierarchy*, a graph decomposition technique widely used in this area [5, 14, 16]. Roughly speaking, the previous almost-linear-time construction [14] relies on modern graph techniques including vertex expander decomposition and approximate vertex capacitated maxflow algorithm, which are highly complicated and will bring relatively large subpolynomial overheads. Our new construction bypasses the vertex expander decomposition to obtain improvement on both efficiency and quality, which is also considerably simpler. Finally, we point out that the subpolynomial factors in our preprocessing time still comes from the construction of the low degree hierarchy, which can be traced back to the subpolynomial overheads of the current approximate vertex capacitated maxflow algorithm [2].

The Fully Dynamic Setting

We also show a better fully dynamic sensitivity oracle for subgraph connectivity, with update time and query time matching the decremental bounds up to polylogarithmic factors. See Theorem 17 for a detailed version of Theorem 2. The first upper bound $\widehat{O}(m)+\widetilde{O}(m(n_{\text{off}}+d_\star))$ is obtained from a combinatorial algorithm[4].

[4] *Combinatorial* algorithms [1] are algorithms that do not use fast matrix multiplication.

▶ **Theorem 2.** *There exists a deterministic fully dynamic sensitivity oracle for subgraph connectivity with $\widehat{O}(m) + \widetilde{O}(\min\{m(n_{\text{off}} + d_\star), n^\omega\})$ preprocessing time, $\widetilde{O}(\min\{m(n_{\text{off}} + d_\star), n^2\})$ space, $\widetilde{O}(d^2)$ update time and $O(d)$ query time, where ω is the exponent of matrix multiplication.*

Table 2 Complexity of known fully dynamic sensitivity oracles for subgraph connectivity.

	Det./ Rand.	Space	Preprocessing	Update	Query
Henzinger & Neumann [8]	Det.	$\widetilde{O}(n_{\text{off}}^2 m)$	$\widehat{O}(n_{\text{off}}^2 m) + \widetilde{O}(d_\star n_{\text{off}}^2 m)$	$\widehat{O}(d^4)$	$O(d^2)$
Hu, Kosinas & Polak [9]	Det.	$\widetilde{O}((n_{\text{off}} + d_\star)m)$	$\widetilde{O}((n_{\text{off}} + d_\star)m)$	$\widetilde{O}(d^4)$	$O(d)$
This paper	Det.	$O(\min\{(n_{\text{off}} + d_\star)m \log^2 n, n^2\})$	$\widehat{O}(m)+$ $O(\min\{(n_{\text{off}} + d_\star)m, n^\omega\} \log^2 n)$	$O(d^2 \log^7 n)$	$O(d)$

We also show conditional lower bounds on the preprocessing time and the space, which separate the fully dynamic and decremental settings. Furthermore, combining our new lower bounds and the existing ones, our solution in Theorem 2 is optimal up to subpolynomial factors.

▶ **Theorem 3.** *Let $\mathcal{A}$ be a fully dynamic sensitivity oracle for subgraph connectivity with S space, t_p preprocessing time, t_u update time and t_q query time upper bounds. Assuming popular conjectures, we have the following:*

1. *If $t_u + t_p = f(d) \cdot n^{o(1)}$, then $S = \widehat{\Omega}(n^2)$. (See Lemma 7.10 in the full version)*
2. *If $t_u + t_p = f(d) \cdot n^{o(1)}$, then $t_u = \widehat{\Omega}((n_{\text{off}} + d)m)$ (See [9])*
3. *If $t_u + t_p = f(d) \cdot n^{o(1)}$, then $t_u = \widehat{\Omega}(n^{\omega_{\text{bool}}})$ (See Lemma 7.3 in the full version)*
4. *If $t_p = \text{poly}(n)$, then $t_u + t_q = \widehat{\Omega}(d^2)$. (See [14])*
5. *If $t_p = \text{poly}(n)$ and $t_u = \text{poly}(dn^{o(1)})$, then $t_q = \widehat{\Omega}(d)$. (See [7])*

The $f(d)$ above can be an arbitrary growing function, and ω_{bool} is the exponent of Boolean matrix multiplication.

We make some additional remarks here. When discussing lower bounds, we assume $d = d_\star$ for each update. The lower bound on the space (item 1) holds even when the input graphs are sparse, so it naturally holds for input graphs with general density. The lower bounds on the preprocessing time (items 2 and 3) are not contradictory, because item 2 is obtained from sparse graphs while item 3 is obtained from dense graphs. The lower bounds on the update time and query time (items 4 and 5) are from those in the decremental setting. See Section 7 in the full version for the omitted proofs and more discussions.

1.2 Organization

In Section 2, we give an overview of our techniques. In Section 3, we give the preliminaries. In Section 4, we introduce our new construction of the low degree hierarchy and obtain a better vertex-failure connectivity oracle as a corollary. In Sections 5 and 6, we describe the preprocessing, update and query algorithms of our fully dynamic sensitivity oracle for subgraph connectivity. Due to space constraints, some proofs are omitted and can be found in the full version. In particular

2 Technical Overview

Better Vertex-Failure Connectivity Oracles

Our main contribution is a new construction of the *low degree hierarchy*. Then we obtain a better vertex-failure connectivity oracle as a corollary by combining the new construction of the low degree hierarchy and the remaining part in [14].

It is known that the construction of the low degree hierarchy can be reduced to $O(\log n)$ calls to the *low-degree Steiner forest decomposition* [5, 14]. Basically, for an input graph G with terminal set $U \subseteq V(G)$, we say a forest $F \subseteq E(G)$ is a *Steiner forest* of U in G if F spans the whole U (may also span some additional non-U vertices) and for each $u, v \in U$, u, v are connected in F if and only if they are connected in G. We propose a new almost-linear time low-degree Steiner forest decomposition algorithm as shown in Lemma 4, which improves the degree parameter Δ from $n^{o(1)}$ to $O(\log^2 n)$ compared to the previous one by [14]. This will leads to an improvement to the quality of the low degree hierarchy, and finally reflects on the update time.

▶ **Lemma 4** (Lemma 12, Informal). *Let G be an undirected graph with terminals $U \subseteq V(G)$. There is an almost-linear-time algorithm that computes a* separator *$|X| \subseteq V(G)$ of size $|X| \leq |U|/2$, and a low-degree Steiner forest of $U \setminus X$ in $G \setminus X$ with maximum degree $\Delta = O(\log^2 n)$.*

In the following discussion, we assume $U = V(G)$ for simplicity (hence spanning trees/-forests and Steiner trees/forests are now interchangable). To obtain Lemma 4, our starting point is that it is *not* necessary to perform a vertex expander decomposition (which will bring large $n^{o(1)}$ overheads to Δ) to get a low-degree Steiner forest decomposition. Basically, in [14], they obtain a fast low-degree Steiner forest decomposition by first proving that any vertex expander admits a low-degree spanning tree, so then it suffices to perform the stronger vertex expander decomposition. The way they prove the former is to argue that for any vertex expander H, one can embed another expander W into H with low vertex-congestion, which implies that H has a low-degree subgraph including all vertices in $V(H)$.

The key observation is that, to make the above argument work, W does not need to be an expander and W can be an arbitrary *connected graph*. This inspires us to design the following subroutine Lemma 5. Then Lemma 4 can be shown by invoking Lemma 10 using a standard divide-and-conquer framework.

▶ **Lemma 5** (Lemma 10, Informal). *Let G be an undirected graph. There is an almost-linear-time algorithm that computes either*

- *a balanced sparse vertex cut (L, S, R) with $|R| \geq |L| \geq |V(G)|/12$ and $|S| \leq 1/(100 \log n) \cdot |L|$.*
- *a large subset $V' \subseteq V(G)$ with $|V'| \geq 3|U|/4$ s.t. we can embed a connected graph W' with $V(W') = V'$ into $G[V']$ with vertex congestion $O(\log^2 n)$, which implies a spanning tree in $G[V']$ with maximum degree $O(\log^2 n)$.*

We design the algorithm in Lemma 5 using a simplified *cut-matching game*. The original cut-matching game [12, 11] can be used to embed an expander into a graph with low congestion (or produce a balanced sparse cut). To embed a connected graph, consider the following procedure. Assume a standard matching player (i.e. Lemma 9) which, given a graph G and a balanced partition (A, B) of $V(G)$, either embeds a large matching between A and B into G with low vertex-congestion or outputs a balanced sparse vertex cut in almost-linear time. Start with a graph W with $V(W) = V(G)$ but no edge and perform several rounds. At each round, we (as the cut player) partition the connected components of W into two parts with balanced sizes, and feed the partition to the matching player. If

the matching player gives a matching, we add it to W and go to the next round. The game stops once a giant connected component (of size at least $3|V(G)|/4$) appears in W, which will roughly serve as W'. Roughly speaking, the game will stop in $O(\log n)$ rounds because at each round, there exists a large fraction of vertices, s.t. for each of them (say vertex v), the component containing v has its size doubled.

Fully Dynamic Sensitivity Oracles for Subgraph Connectivity

Our fully dynamic oracle is actually a generalization of a simplified version of the decremental oracle in [14].

Initially, we construct a low degree hierarchy on the activated subgraph $G_{\text{on}} := G[V_{\text{on}}]$. As mentioned in [14], the hierarchy will roughly reduce G_{on} to the following *semi-bipartite* form. First, V_{on} can be partitioned into L_{on} and R_{on}, called *left on-vertices* and *right on-vertices* respectively, s.t. there is no edge connecting two vertices in R_{on}. Second, L_{on} is spanned by a known path τ. Therefore, we assume the original graph G has a semi-bipartite G_{on} from now.

When there is no off-vertices initially (i.e. the decremental setting), the properties of a semi-bipartite G_{on} naturally support the following update and query strategy. In the update phase, removing vertices in D will break the path τ into at most $d+1$ *intervals*, and we will somehow (we will not explain this in the overview) recompute the connectivity of these intervals in the graph $G_{\text{on}} \setminus D$. Then, for each query of $u, v \in V_{\text{on}} \setminus D$, it suffices to find two intervals I_u, I_v connecting with u, v in $G_{\text{on}} \setminus D$ respectively. When u, v are left on-vertices, I_u, I_v can be found trivially. When u, v are right on-vertices, we just need to scan at most $d+1$ neighbors of each of u, v, which takes $O(d)$ time. Note that removing D will generate at most $d+1$ intervals is a crucial point to achieve fast update time.

Back to the fully dynamic setting, for an update D, in addition to removing vertices $D_{\text{on}} := D \cap V_{\text{on}}$ from G_{on}, we will also add vertices $D_{\text{off}} := D \cap V_{\text{off}}$ and their incident edges. The key observation is that $G[V_{\text{on}} \cup D_{\text{off}}]$ is still roughly a semi-bipartite graph. The first property will still hold if we put the newly activated vertices D_{off} into the left side. The second property may not hold because we do not have a path spanning the new left vertices $L_{\text{on}} \cup D_{\text{off}}$. However, this will not hurt because we can still partition $L_{\text{on}} \cup D_{\text{off}}$ into $O(d)$ connected parts after removing D_{on} from $G[V_{\text{on}} \cup D_{\text{off}}]$, i.e. at most $d+1$ intervals covering $L_{\text{on}} \setminus D_{\text{on}}$, and at most d vertices in D_{off}.

Giving this key observation, it is quite natural to adapt the decremental algorithm to the fully dynamic setting. Using the ideas of *adding artificial edges* (intuitively, substituting each right vertex and its incident edges with an artificial clique on its left neighbors) and applying *2D range counting structure*, we can design an update algorithm with $\widetilde{O}(d^3)$ update time [5]. To improve the update time to $\widetilde{O}(d^2)$, we can use a *Borůvka's styled* update algorithm and implement it by considering *batched adjacency queries* on intervals [14].

3 Preliminaries

Throughout the paper, we use the standard graph theoretic notation. For any graph, we use $V(\cdot)$ and $E(\cdot)$ to denote its vertex set and edge set respectively. If there is no other specification, we use G to denote the original graph on which we will build the oracle, and we let $n = |V(G)|$ and $m = |E(G)|$. Initially, the vertices $V(G)$ in the original graph are partitioned into *on-vertices* V_{on} and *off-vertices* V_{off}, and we let $n_{\text{off}} = |V_{\text{off}}|$. For a graph H and any $S \subseteq V(H)$, we let $H[S]$ denote the subgraph induced by vertices S. Also, for any $S \subseteq V(H)$, we use $H \setminus S$ to denote the graph after removing vertices in S and edges incident to them. Similarly, for any $F \subseteq E(H)$, $H \setminus F$ denote the graph after removing edges in F.

We also use the notion of *multigraphs*. For a multigraph H, its edge set $E(H)$ is a *multiset*. We use $+$ and $\sum$ to denote the union operation and use $-$ to denote the subtraction operation on multiset. We let ω denote the exponent of matrix multiplication and ω_{bool} denote the exponent of Boolean matrix multiplication. To our best knowledge, currently ω and ω_{bool} have the same upper bound.

4 The Low Degree Hierarchy

The *low degree hierarchy* was first introduced in [5] to design efficient vertex-failure connectivity oracles. The construction of this hierarchy in [5] is based on the approximate minimum degree Steiner forest algorithm of [6], which gives $\tilde{O}(mn)$ construction time. Later, an alternative construction algorithm was shown in [14] by exploiting vertex expander decomposition, which improves the construction time to $m^{1+o(1)}$, at a cost of a small quality loss.

In this section, we will show a new construction algorithm, which still runs in almost-linear time and gives a hierarchy with quality better than the one in [14] (but still worse than the one in [5]). To obtain the quality improvement, we basically simplify the construction in [14] and bypass the vertex expander decomposition.

We define the low degree hierarchy in Definition 6, and the main result of this section is Theorem 7. It was known that constructing a low degree hierarchy reduces to several rounds of *low-degree Steiner forest decomposition*. In Section 4.1, we introduce our key subroutine Lemma 10, which given an input graph, either computes a balanced sparse vertex cut or a low-degree Steiner tree covering a large fraction of terminals. In Section 4.2, we show the low-degree Steiner forest decomposition algorithm Lemma 12 using Lemma 10 in a standard divide and conquer framework, and then complete the proof of Theorem 7.

▶ **Definition 6** (Low Degree Hierarchy [5], Definition 5.1 in [14]). *Let G be a connected undirected graph. A (p, Δ)-low degree hierarchy with height p and degree parameter Δ on G is a pair $(\mathcal{C}, \mathcal{T})$ of sets, where $\mathcal{C}$ is a set of vertex-induced connected subgraphs called components, and $\mathcal{T}$ is a set of Steiner trees with maximum vertex degree at most Δ.*

The set $\mathcal{C}$ of components is a laminar set. Concretely, it satisfies the following properties.

(1) *Components in $\mathcal{C}$ belong to p levels and we denote by $\mathcal{C}_i$ the set of components at level i. In particular, at the top level p, $\mathcal{C}_p = \{G\}$ is a singleton set with the whole G as the unique component. Furthermore, for each level $i \in [1, p]$, components in $\mathcal{C}_i$ are vertex-disjoint and there is no edge in $E(G)$ connecting two components in $\mathcal{C}_i$.*

(2) *For each level $i \in [1, p-1]$ and each component $\gamma \in \mathcal{C}_i$, there is a unique component $\gamma' \in \mathcal{C}_{i+1}$ such that $V(\gamma) \subseteq V(\gamma')$, where we say that γ' is the parent-component of γ and that γ is a child-component of γ'.*

(3) *For each component $\gamma \in \mathcal{C}$, the terminals of γ, denoted by $U(\gamma)$, are vertices in γ but not in any of γ's child-components. Note that $U(\gamma)$ can be empty. In particular, for each $\gamma \in \mathcal{C}_1$, $U(\gamma) = V(\gamma)$.*

Generally, for each level $i \in [1, p]$, we define the terminals at level i be terminals in all components in $\mathcal{C}_i$, denoted by $U_i = \bigcup_{\gamma \in \mathcal{C}_i} U(\gamma)$.

The set $\mathcal{T}$ of low-degree Steiner trees has the following properties.

(4) *$\mathcal{T}$ can also be partitioned into subsets $\mathcal{T}_1, ..., \mathcal{T}_p$, where $\mathcal{T}_i$ denote trees at level i and trees in $\mathcal{T}_i$ are vertex-disjoint.*

(5) *For each level $i \in [1, p]$ and tree $\tau \in \mathcal{T}_i$, the terminals of τ is defined by $U(\tau) = U_i \cap V(\tau)$.*

(6) *For each level $i \in [1, p]$ and each component $\gamma \in \mathcal{C}_i$ with $U(\gamma) \neq \emptyset$, there is a tree $\tau \in \mathcal{T}_i$ such that $U(\gamma) \subseteq U(\tau)$, denoted by $\tau(\gamma)$. We emphasize that two different components γ and $\gamma' \in \mathcal{C}_i$ may correspond to the same tree $\tau \in \mathcal{T}_i$.*

For better understanding, we note that the terminal sets of component $\{U(\gamma) \mid \gamma \in \mathcal{C}\}$, levels $\{U_i \mid 1 \leq i \leq p\}$, and Steiner trees $\{U(\tau) \mid \tau \in \mathcal{T}\}$ are all partitions of $V(G)$. One may also get the picture of the hierarchy from the perspective of construction. See the construction described in Algorithm 2, which invokes Lemma 12 in a black-box way.

▶ **Theorem 7.** *Let G be an undirected graph. There is a deterministic algorithm that computes a (p, Δ)-low degree hierarchy with $p = O(\log n)$ and $\Delta = O(\log^2 n)$. The running time is $m^{1+o(1)}$.*

Corollary 8 is obtained by substituting the construction of low degree hierarchy in [14] with ours. Formally speaking, it is a corollary of Theorem 7, and Lemma 6.14, Theorem 7.2 and Section 7.3 in [14].

▶ **Corollary 8.** *There is a deterministic vertex-failure connectivity oracle with $\widehat{O}(m) + O(d_\star m \log^3 n)$ preprocessing time, $O(m \log^3 n)$ space, $O(d^2(\log^7 n + \log^5 n \cdot \log^4 d))$ update time and $O(d)$ query time.*

4.1 A Balanced Sparse Vertex Cut or a Low-Degree Steiner Tree

The goal of this subsection is to show Lemma 10, a subroutine which given a graph with terminals, outputs either a balanced sparse vertex cut or a low-degree Steiner tree covering a large fraction of terminals. In fact, some expander decomposition algorithms (e.g. [3]) exploit a similar subroutine which either computes a balanced sparse cut or certifies that a large part of the graph is an expander. Our subroutine can be viewed as a weaker and simplified version, because similar to the notion of expanders, a low-degree Steiner tree is also an object that certifies some kind of (weaker) well-linkedness.

At a high level, our algorithm uses a simplified *cut-matching-game* framework. A cut-matching game is an interactive process between a cut player and a matching player with several rounds. Start from a graph with no edge. In each round, the cut player will select a cut and then the matching player is required to add a perfect matching on this cut. It is known that there exists cut-player strategies against an arbitrary matching player that guarantees the final graph is an expander after $\widetilde{O}(1)$ rounds [12, 11]. In the proof of Lemma 10, we show a cut-player strategy that only guarantees the final graph is a *connected graph*. Combining a classic matching player as shown in Lemma 9, we can either find a balanced sparse vertex cut or embed a connected graph covering most of the terminals into the original graph with low vertex congestion. In the latter case, the embedding leads to a low-degree subgraph covering most of the terminals. Finally, picking an arbitrary spanning tree in this subgraph suffices.

Given a cut w.r.t. terminals, Lemma 9 will either output a balanced sparse vertex cut or a large matching between terminals that is embeddable into the original graph with low vertex congestion. In fact, it is a simplified version of the matching player in [14], and the proof can be found in Appendix A.1 of the full version.

▶ **Lemma 9.** *Let G be an undirected graph with a terminal set U. Given a parameter ϕ and a partition (A, B) of U, there is a deterministic algorithm that computes either*

- *a vertex cut (L, S, R) with $|R \cap U| \geq |L \cap U| \geq \min\{|A|, |B|\}/3$ and $|S| \leq \phi \cdot |L \cap U|$, or*
- *a matching M between A and B with size $|M| \geq \min\{|A|, |B|\}/3$ s.t. there is an embedding $\Pi_{M \to G}$ of M into G with vertex congestion at most $\lceil 1/\phi \rceil$.*

The running time is $m^{1+o(1)}$. If the output is a matching M, the algorithm can further output the edge set $E(\Pi_{M \to G})$ of the embedding $\Pi_{M \to G}$.

▶ **Lemma 10.** *Let G be an undirected graph with a terminal set U. Given parameters $0 < \epsilon, \phi \leq 1/4$, there is a deterministic algorithm that computes either*

- *a vertex cut (L, S, R) with $|R \cap U| \geq |L \cap U| \geq \epsilon|U|/3$ and $|S| \leq \phi \cdot |L \cap U|$, or*
- *a subset $U_{\text{drop}} \subseteq U$ of terminals with $|U_{\text{drop}}| \leq \epsilon|U|$ and a Steiner tree τ on $G \setminus U_{\text{drop}}$ of terminal set $U \setminus U_{\text{drop}}$ with maximum degree $O(\log |U|/\phi)$.*

Proof. The algorithm is made up of an iteration phase and a postprocessing phase. The iteration phase will maintain an incremental graph W with $V(W) = U$, called the *witness graph*, and its embedding $\Pi_{W \to G}$ into G. Precisely, instead of storing the embedding $\Pi_{W \to G}$ explicitly, the algorithm will only store its edge set $E(\Pi_{W \to G})$. Initially, the witness graph $W^{(0)}$ has no edge and $E(\Pi_{W^{(0)} \to G})$ is empty. We use $W^{(i)}$ and $E(\Pi_{W^{(i)} \to G})$ to denote the witness graph and the edge set of the embedding right after the i-th round.

In the iteration phase, we do the following steps in the i-th round.

1. We compute all the connected components of $W^{(i-1)}$, which forms a partition $\mathcal{Q}^{(i-1)}$ of U s.t. each $Q \in \mathcal{Q}^{(i-1)}$ is a subset of $|U|$, called a *cluster*. If there is a cluster $Q^\star \in \mathcal{Q}^{(i-1)}$ has $|Q^\star| \geq (1-\epsilon)|U|$, then we terminate the iteration phase and go to the postprocessing phase, otherwise we proceed to the next step.
2. Because step 1 guarantees that all clusters in $\mathcal{Q}^{(i-1)}$ have size at most $(1-\epsilon)|U|$, we will partition $\mathcal{Q}^{(i-1)}$ into two groups $\mathcal{Q}_A$ and $\mathcal{Q}_B$ depending on the following two cases.
   - **(a)** If all clusters in $\mathcal{Q}^{(i-1)}$ have size at most $|U|/2$, then we partition $\mathcal{Q}^{(i-1)}$ into $\mathcal{Q}_A$ and $\mathcal{Q}_B$ s.t. $\sum_{Q \in \mathcal{Q}_A} |Q| \geq |U|/4$ and $\sum_{Q \in \mathcal{Q}_B} |Q| \geq |U|/4$.
   - **(b)** Otherwise, there is a unique cluster $Q^\star$ s.t. $|U|/2 < |Q^\star| \leq (1-\epsilon)|U|$, and we let $\mathcal{Q}_A = \mathcal{Q} \setminus \{Q^\star\}$ and $\mathcal{Q}_B = \{Q^\star\}$.

   Let $A_i = \bigcup_{Q \in \mathcal{Q}_A} Q$ and $B_i = \bigcup_{Q \in \mathcal{Q}_B} Q$. Note that by definition, (A_i, B_i) forms a partition of U. We have $|A_i|, |B_i| \geq |U|/4$ in case (a) and $|A_i|, |B_i| \geq \epsilon|U|$ in case (b).
3. We apply Lemma 9 on graph G and terminal U with parameter ϕ and the partition (A_i, B_i) of U. If we get a vertex cut (L, S, R), it will satisfy $|R \cap U| \geq |L \cap U| \geq \min\{|A_i|, |B_i|\}/3 \geq \epsilon|U|/3$ and $|S| \leq \phi \cdot |L \cap U|$ as desired, so we can terminate the whole algorithm with (L, S, R) as the output. Otherwise, we get a matching M_i between A_i and B_i with size $|M_i| \geq |A_i|/3$, and the edge set $E(\Pi_{M_i \to G})$ of some embedding $\Pi_{M_i \to G}$ that has vertex congestion $O(1/\phi)$. Then we let $W^{(i)} = W^{(i-1)} \cup M_i$ and $E(\Pi_{W^{(i)} \to G}) = E(\Pi_{W^{(i-1)} \to G}) \cup E(\Pi_{M_i \to G})$, and proceed to the next round.

If the algorithm does not end at step 3, it exits the iteration phase at step 1, and then we perform the following postprocessing phase. Let W denote the final witness graph with connected components $\mathcal{Q}$ and a cluster $Q^\star \in \mathcal{Q}$ s.t. $|Q^\star| \geq (1-\epsilon)|U|$. Note that $Q^\star \subseteq U$. Let G' be the subgraph of G induced by $E(\Pi_{W \to G})$. By the definition of embedding, vertices in $Q^\star$ are also connected in G'. In other words, $Q^\star$ is contained by a connected component of G'. We can take an arbitrary spanning tree τ of this component as a Steiner tree of $V(\tau) \cap U$, and define $U_{\text{drop}} = U \setminus V(\tau)$ be the uncovered terminals.

We now show that U_{drop} and τ have the desire property. The number of uncovered terminals is bounded by $|U_{\text{drop}}| \leq |U| - |V(\tau) \cap U| \leq |U| - |Q^\star| \leq \epsilon|U|$, and τ is a Steiner tree of $U \setminus U_{\text{drop}}$ with maximum degree $O(\log |U|/\phi)$ because G' has maximum degree at most the vertex congestion of $\Pi_{W \to G}$, which is at most $O(\log |U|/\phi)$ by Claim 11.

▷ Claim 11. The number of rounds in the iteration phase is at most $O(\log |U|)$, and the vertex congestion of the final embedding $\Pi_{W \to G}$ is at most $O(\log |U|/\phi)$.

Proof. Note that the early rounds will go into case (a) in step 2, while the late rounds will go into case (b). We bound the number of case-(a) rounds and case-(b) rounds separately.

The number of case-(a) rounds is at most $O(\log |U|)$ by the following reason. We define a potential function $\Phi(W)$ of the witness graph by

$$\Phi(W) = \sum_{Q \in \mathcal{Q}} \sum_{v \in Q} \log |Q| = \sum_{Q \in \mathcal{Q}} |Q| \cdot \log |Q|.$$

In particular, for each cluster $Q \in \mathcal{Q}$ and each vertex $v \in Q$, we say the potential at v is $\log |Q|$.

Because initially $\Phi(W^{(0)}) = 0$ and we always have $\Phi(W) \leq |U| \log |U|$, it is sufficient to show that each case-(a) round increases the potential by at least $\Omega(|U|)$. To see this, consider the i-th case-(a) round. For each matching edge $\{u, v\} \in M_i$, let $Q_v^{(i-1)}$ (resp. $Q_u^{(i-1)}$) be the connected component of $W^{(i-1)}$ that contains v (resp. u), and assume without loss of generality that $|Q_v^{(i-1)}| \leq |Q_u^{(i-1)}|$. Then the connected component $Q_v^{(i)}$ of $W^{(i)}$ that contains v will have $|Q_v^{(i)}| \geq 2|Q_v^{(i-1)}|$, because $Q_v^{(i)} \supseteq Q_u^{(i-1)} \cup Q_v^{(i-1)}$. In other words, this round will increase the potential at v (from $\log |Q_v^{(i-1)}|$ to $\log |Q_v^{(i)}|$) by at least 1. Summing over $|M_i|$ matching edges, the total potential $\Phi(W)$ will be increased by at least $|M_i| \geq |U|/12$ as desired, because the potential at any $v \in V(W)$ will never drop.

It remains to show that the number of case-(b) rounds is at most $O(\log |U|)$. This is simple because in each round, the matching M_i will merge at least $|A_i|/3$ terminals in $|A_i|$ into the giant cluster $Q^\star$, which means $|Q^\star|$ will reach the threshold $(1 - \epsilon)|U|$ in $O(\log |U|)$ many case-(b) rounds and then the iteration phase ends.

The final embedding $\Pi_{W \to G}$ has vertex congestion $O(\log |U|/\phi)$ because there are $O(\log |U|)$ rounds and the embedding $\Pi_{M_i \to G}$ has vertex congestion $O(1/\phi)$ each round. ◁

◀

4.2 The Low-Degree Steiner Forest Decomposition

Lemma 12 describes the low-degree Steiner forest decomposition algorithm, which invokes Lemma 10 in a divide-and-conquer fashion. For simplicity, the readers can always assume $\epsilon = 1/2$, which is the value we will choose when constructing the low degree hierarchy. We introduce this tradeoff parameter ϵ just to show that our algorithm has the same flexibility as those in [5, 14].

▶ **Lemma 12.** *Let G be an undirected graph with a terminal set U. Given a parameter $0 < \epsilon \leq 1/2$, there is a deterministic algorithm that computes*

- *a vertex set $X \subseteq V(G)$, called the separator, s.t. $|X| \leq \epsilon |U|$, and*
- *for each connected component Y of $G \setminus X$ s.t. U intersects $V(Y)$, a Steiner tree τ_Y spanning $U \cap V(Y)$ on Y with maximum degree $O(\log^2 |U|/\epsilon)$.*

The running time is $m^{1+o(1)}/\epsilon$.

We include the algorithm of Lemma 12 in Algorithm 1, and the complete proof can be founded in the full version.

Algorithm 1 The low-degree Steiner forest decomposition $\text{SFDecomp}(G, U)$.

Input: An undirected graph G with terminals U.
Output: A separator X and a collection $\mathcal{T}$ of Steiner trees $\{\tau_Y\}$.

Let $\epsilon' = \epsilon/2$ and $\phi = \epsilon'/\log|U|$
Apply Lemma 10 on G and U with parameters ϵ' and ϕ.
if Lemma 10 outputs a vertex cut (L, S, R) **then**
    $(X_L, \mathcal{T}_L) \leftarrow \text{SFDecomp}(G[L], L \cap U)$
    $(X_R, \mathcal{T}_R) \leftarrow \text{SFDecomp}(G[R], R \cap U)$
    Return $X = X_L \cup X_R \cup S$ and $\mathcal{T} = \mathcal{T}_L \cup \mathcal{T}_R$.
else
    Otherwise Lemma 10 outputs $U_{\text{drop}} \subseteq U$ and a Steiner tree τ of $U \setminus U_{\text{drop}}$ on $G \setminus U_{\text{drop}}$.
    Return $X = U_{\text{drop}}$ and $\mathcal{T} = \{\tau\}$.
end if

As shown in [5], to construct a low-degree hierarchy $(\mathcal{C}, \mathcal{T})$, it suffices to invoke the low-degree Steiner forest decomposition (with $\epsilon = 1/2$) $O(\log n)$ times. The algorithm is shown in Algorithm 2, and the proof of correctness is included in Appendix A.2 in the full version.

Algorithm 2 The construction of the low-degree hierarchy.

Input: An undirected graph G.
Output: A low-degree hierarchy $(\mathcal{C}, \mathcal{T})$.

Initialize $i = 1$, $X_1 = V(G)$.
while X_i is not empty **do**
    $(X_{i+1}, \mathcal{T}_i) \leftarrow \text{SFDecomp}(G, X_i)$ with $\epsilon = 1/2$.
    $i \leftarrow i + 1$.
end while
$p \leftarrow i - 1$, which denotes the number of levels.
for each level i **do**
    $U'_i \leftarrow X_i \cup ... \cup X_p$.
    $\mathcal{C}_i \leftarrow$ the connected component of $G \setminus U'_{i+1}$ (particularly, $U'_{p+1} = \emptyset$).
    $U_i \leftarrow U'_i \setminus U'_{i+1}$, which denotes the terminals of level i.
end for

5 The Preprocessing Algorithm

In this section, we will describe the preprocessing algorithm, which basically first computes the low degree hierarchy on $G_{\text{on}} := G[V_{\text{on}}]$, and then constructs some affiliated data structures on top of the hierarchy.

The low degree hierarchy $(\mathcal{C}, \mathcal{T})$ is computed by applying Theorem 7 on G_{on}, if G_{on} is a connected graph. In the case that G_{on} is not connected, we simply apply Theorem 7 on each of the connected components of G_{on}. To simplify the notations, we use $(\mathcal{C}, \mathcal{T})$ to denote the union of hierarchies of connected components of G_{on}, and still say $(\mathcal{C}, \mathcal{T})$ is the low degree hierarchy of G_{on}. Note that $(\mathcal{C}, \mathcal{T})$ has all properties in Definition 6, except that the top level $\mathcal{C}_1$ are now made up of connected components of G_{on}.

In Section 5.1, we introduce the notions of artificial edges and the artificial graph $\hat{G}$. In Section 5.2, we define a global order π based on Euler tour orders of Steiner trees in $\mathcal{T}$, and then construct a 2D range counting structure which can answer the number of edges in $E(\hat{G})$ between two intervals on π. Finally, in Section 5.3, we summarize what we will store, and analyse the preprocessing time and the space complexity.

5.1 Artificial Edges and the Artificial Graph $\hat{G}$

The artificial graph $\hat{G}$ is a multi-graph constructed by adding some *artificial edges* into the original graph G in the following way. For each component $\gamma \in \mathcal{C}$, let A_γ collect the neighbors of $V(\gamma)$ in G, formally defined by $A_\gamma = \{v \mid v \in V(G) \setminus V(\gamma) \text{ s.t. } \exists\{u, v\} \in E(G) \text{ with } u \in V(\gamma)\}$. We call A_γ the *adjacency list* of γ. Let $A_{\gamma,\text{on}} = A_\gamma \cap V_{\text{on}}$ and $A_{\gamma,\text{off}} = A_\gamma \cap V_{\text{off}}$. Next, we let $B_{\gamma,\text{off}} = A_{\gamma,\text{off}}$ and let $B_{\gamma,\text{on}}$ be an arbitrary subset of $A_{\gamma,\text{on}}$ with size $\min\{d_\star + 1, |A_{\gamma,\text{on}}|\}$. Then define $B_\gamma = B_{\gamma,\text{on}} \cup B_{\gamma,\text{off}}$.

The artificial edges added by the component γ is then $\hat{E}_\gamma = \{\{u, v\} \mid u \in A_\gamma, v \in B_\gamma, u \neq v\}$. Namely, $\hat{E}_\gamma$ consists of a clique on B_γ and a biclique between B_γ and $A_\gamma \setminus B_\gamma$. Finally, the artificial graph $\hat{G}$ is defined by $\hat{G} = G + \sum_{\gamma \in \mathcal{C}} \hat{E}_\gamma$. We emphasize that $\hat{G}$ is a multi-graph, and those edges connecting the same endpoints will have different identifiers.

We show some useful properties in Proposition 13. Item 3 of Proposition 13 basically says that, if A_γ has an on-vertex after update, then B_γ also has one.

▶ **Proposition 13.** *We have the following.*

1. $\sum_{\gamma \in \mathcal{C}} |A_\gamma| \leq O(pm)$.
2. $|E(\hat{G})| \leq O(pm(n_{\text{off}} + d_\star))$.
3. *Given any update $D \subseteq V$ with $|D| \leq d_\star$, if $(A_{\gamma,\text{on}} \setminus D) \cup (A_{\gamma,\text{off}} \cap D) \neq \emptyset$, then we have $(B_{\gamma,\text{on}} \setminus D) \cup (B_{\gamma,\text{off}} \cap D) \neq \emptyset$.*

Proof.

Part 1. For each $\gamma \in \mathcal{C}$, observe that $|A_\gamma| \leq \sum_{v \in V(\gamma)} \deg_G(v)$. Hence $\sum_{\gamma \in \mathcal{C}} |A_\gamma| \leq O(pm)$ because each vertex can appear in at most p components (at most one at each level).

Part 2. By definition, $|E(\hat{G})| \leq m + \sum_{\gamma \in C} |\hat{E}_\gamma| \leq m + \sum_{\gamma \in \mathcal{C}} |A_\gamma| \cdot |B_\gamma|$. Note that $|B_\gamma| \leq n_{\text{off}} + d_\star + 1$ for all $\gamma \in \mathcal{C}$ by construction. Combining part 1, we have $|E(\hat{G})| \leq O(pm(n_{\text{off}} + d_\star))$.

Part 3. If $|A_{\gamma,\text{on}}| \leq d_\star + 1$, we have $A_\gamma = B_\gamma$ by construction and the proposition trivially holds. Otherwise, B_γ will include $d_\star + 1$ vertices in $A_{\gamma,\text{on}}$. Because $|D| \leq d_\star$, at least one of them will survive in $B_{\gamma,\text{on}} \setminus D$, which implies $(B_{\gamma,\text{on}} \setminus D) \cup (B_{\gamma,\text{off}} \cap D) \neq \emptyset$. ◀

5.2 The Global Order and Range Counting Structures

Next, we define an order, called the *global order* and denoted by π, over the whole vertex set $V(G)$, based on the Euler Tour orders of Steiner trees in $\mathcal{T}$.

For each $\tau \in \mathcal{T}$, we define its Euler tour order $\text{ET}(\tau)$ as an ordered list of vertices in $V(\tau)$ ordered by the time stamps of their first appearances in an Euler tour of τ (starting from an arbitrary root). Intuitively, the Euler tour order $\text{ET}(\tau)$ can be interpreted as a linearization of τ, i.e. after the removal of failed vertices in τ, the remaining subtrees will corresponding to intervals on $\text{ET}(\tau)$, as shown in Lemma 14.

▶ **Lemma 14** (Lemma 6.3 in [14], Rephrased). *Let τ be an undirected tree with maximum vertex degree Δ. A removal of d failed vertices from τ will split τ into at most $O(\Delta d)$ subtrees $\hat{\tau}_1, \hat{\tau}_2, ..., \hat{\tau}_{O(\Delta d)}$, and there exists a set $\mathcal{I}_\tau$ of at most $O(\Delta d)$ disjoint intervals on $ET(\tau)$, such that each interval is owned by a unique subtree and for each subtree τ_i, $V(\tau_i)$ is equal to the union of intervals it owns.*

Furthermore, by preprocessing τ in $O(|V(\tau)|)$ time, we can store $\mathrm{ET}(\tau)$ and some additional information in $O(|V(\tau)|)$ space, which supports the following operations.

- *Given a set D_τ of d failed vertices, the intervals $\mathcal{I}_t$ can be computed in $O(\Delta d \log(\Delta d))$ update time.*
- *Given a vertex $v \in V(\tau) \setminus D_\tau$, it takes $O(\log d)$ query time to find an interval $I \in \mathcal{I}_t$ s.t. vertices in I are connected to v in $\tau \setminus D_\tau$.*

Given the Euler tour orders of all $\tau \in \mathcal{T}$, we define the global order π as follows. We first concatenate $\mathrm{ET}(\tau) \cap U(\tau)$ (i.e. the restriction of $\mathrm{ET}(\tau)$ on the terminals of τ) of all $\tau \in \mathcal{T}$ in an arbitrary order, and then append all vertices in V_{off} to the end. Recall that $\{U(\tau) \mid \tau \in \mathcal{T}\}$ partitions V_{on}, so π is well-defined.

With the global order π, we will construct a 2D-range counting structure Table, which can answer the number of edges in $E(\hat{G})$ that connect two disjoint intervals on π. We first initialize $\mathsf{Table}_{\text{init}}$ to be an ordinary 2D array on range $\pi \times \pi$. For each $u, v \in \pi$, we store a non-negative integer in the entry $\mathsf{Table}_{\text{init}}(u, v)$ representing the number of edges in $E(\hat{G})$ connecting vertices u and v.

▶ **Lemma 15.** *Suppose that we can access the lists A_γ and B_γ for all $\gamma \in \mathcal{C}$. There is a combinatorial algorithm that computes $\mathsf{Table}_{\text{init}}$ in $O(|E(\hat{G})|)$ time, or $\mathsf{Table}_{\text{init}}$ can be computed in $O(p \cdot n^\omega)$ time using fast matrix multiplication.*

Proof. A trivial construction of $\mathsf{Table}_{\text{init}}$ is to construct the edge sets $E(\hat{G})$ explicitly, and then scan the edges one by one. Obviously, this takes $O(|E(\hat{G})|)$ time.

When $|E(\hat{G})|$ is large, we can use fast matrix multiplication (FMM) to speed up the construction of $\mathsf{Table}_{\text{init}}$. Recall that $E(\hat{G}) = E(G) + \sum_{\gamma \in \mathcal{C}} \hat{E}_\gamma$. We first add the contribution of $E(G)$ into $\mathsf{Table}_{\text{init}}$ using the trivial algorithm, which takes $O(m)$ time. Next, we compute the contribution of artificial edges, i.e. $\sum_{\gamma \in \mathcal{C}} \hat{E}_\gamma$, using FMM. We construct a matrix X with n rows and $|\mathcal{C}|$ columns, where rows are indexed by the global order π and columns are indexed by components (in an arbitrary order). For each vertex $u \in \pi$ and component $\gamma \in \mathcal{C}$, the entry $X(u, \gamma) = 1$ if and only if $u \in A_\gamma$. Similarly, we define an n-row $|\mathcal{C}|$-column matrix Y, in which each entry $Y(u, \gamma) = 1$ if and only if $u \in A_\gamma \setminus B_\gamma$. Let $Z = X \cdot X^\intercal - Y \cdot Y^\intercal$. Observe that, for each pair of distinct vertices $u, v \in \pi$,

$$\begin{aligned} Z(u, v) &= \sum_{\gamma \in \mathcal{C}} (X(u, \gamma) \cdot X(v, \gamma) - Y(u, \gamma) \cdot Y(v, \gamma)) \\ &= \sum_{\gamma \in \mathcal{C}} \mathbb{1}[u, v \in A_\gamma] - \mathbb{1}[u, v \in A_\gamma \setminus B_\gamma] \\ &= \sum_{\gamma \in \mathcal{C}} \mathbb{1}[\{u, v\} \in \hat{E}_\gamma]. \end{aligned}$$

Therefore, the matrix Z count the contribution of $\sum_\gamma \hat{E}_\gamma$ correctly and the last step is to add Z to $\mathsf{Table}_{\text{init}}$. The construction time is dominated by the computation of Z, which takes $O(p \cdot n^\omega)$ time because it involves multiplying an $n \times |\mathcal{C}|$ matrix and a $|\mathcal{C}| \times n$ matrix, and $|\mathcal{C}| = O(pn)$. ◀

▶ **Lemma 16.** *With access to the positive entries of* $\mathsf{Table}_{\text{init}}$*, we can construct a data structure* Table *that given any disjoint intervals* I_1 *and* I_2 *on* π*, answers in* $O(\log n)$ *time the number of edges in* $E(\hat{G})$ *with one endpoint in* I_1 *and the other one in* I_2*. The structure* Table *can be constructed in* $O(N \log n)$ *time and takes space* $O(N \log n)$*, where* N *denotes the number of positive entries in* $\mathsf{Table}_{\text{init}}$*.*

Proof. We simply construct Table as a standard weighted 2D range counting structure of $\mathsf{Table}_{\text{init}}$, By using textbook algorithms such as range trees and persistent segment trees, we can construct Table in $O(N \log n)$ time and it takes space $O(N \log n)$. The correctness of Table follows the definition of $\mathsf{Table}_{\text{init}}$. ◀

5.3 Preprocessing Time and Space Analysis

In conclusion, we will compute and store the following in the preprocessing phase.

- First, we store the low degree hierarchy $(\mathcal{C}, \mathcal{T})$. Constructing the low degree hierarchy takes $\hat{O}(m)$ time by Theorem 7. Storing the low degree hierarchy explicitly takes $O(pn)$ space, because for each level i, the components in $\mathcal{C}_i$ are vertex disjoint, also Steiner trees in $\mathcal{T}_i$.
- Next, for each $\gamma \in \mathcal{C}$, we store the lists A_γ and B_γ after ordering them by π. Computing the lists A_γ and B_γ takes $O(pm)$ time by checking the incident edges of each vertex in each component. Storing the lists A_γ and B_γ takes $O(pm)$ space by Item 1 in Proposition 13. Additionally, for each $\gamma \in \mathcal{C}$, store the list $A_{\gamma,\text{on}}$.
  For each $v \in V_{\text{off}}$ and $\gamma \in \mathcal{C}$, store a binary indicator to indicate whether $v \in A_{\gamma,\text{off}}$ or not. Computing the indicators takes $O(pm)$ time by scanning all the lists A_γ. Storing the indicators explicitly takes $O(|V_{\text{off}}| \cdot |\mathcal{C}|) = O(pn \cdot |V_{\text{off}}|)$ space.
- We also store the global order π, which takes $O(n)$ space. For each $\tau \in \mathcal{T}$, we store $\mathrm{ET}(\tau)$ and the additional information stated in Lemma 14 in $O(|V(\tau)|)$ space. Computing the things above takes totally $\sum_{\tau \in \mathcal{T}} |V(\tau)| = O(pn)$ time by Lemma 14.
- Finally, we store the data structure Table. Combining Item 2 in Proposition 13 and Lemmas 15 and 16, we can compute Table in $O(pm(n_{\text{off}} + d) \log n)$ time using an combinatorial algorithm, or in $O(p \cdot n^\omega \log n)$ time using fast matrix multiplication. The space to store Table is $\min\{pm(n_{\text{off}} + d) \log n, n^2\}$.

In conclusion, the total preprocessing time can be upper bounded by $\widehat{O}(m) + O(pm(n_{\text{off}} + d) \log n)$ using an combinatorial algorithm, then $t_p = \hat{\Omega}(md)$, or $\hat{O}(m) + O(p \cdot n^\omega \log n)$ using fast matrix multiplication. The space complexity is $O(\min\{pm(n_{\text{off}} + d) \log n, n^2\})$. Because the low degree hierarchy has $p = O(\log n)$ levels, the preprocessing time is $\widehat{O}(m) + O(\min\{m(n_{\text{off}} + d) \log^2 n, n^\omega \log^2 n\})$, and the space is $O(\min\{m(n_{\text{off}} + d) \log^2 n, n^2\})$.

6 The Update and Query Algorithms

Let $D \subseteq V(G)$ be a given update. We use $D_{\text{on}} = D \cap V_{\text{on}}$ to denote the vertices that will be turned off in this update and $D_{\text{off}} = D \cap V_{\text{off}}$ to denote the vertices that will be turned on. Let $V_{\text{new}} = (V_{\text{on}} \setminus D_{\text{on}}) \cup D_{\text{off}}$ be the on-vertices after updates.

Our update strategy is to recompute the connectivity of a subset of *affected vertices* $Q^\star \subseteq V_{\text{new}}$ on some affected graph $G^\star$. In Section 6.1, we will define $Q^\star$ and $G^\star$, and prove that $Q^\star$ has the same connectivity on the affected graph $G^\star$ and the updated original graph $G[V_{\text{new}}]$. In Section 6.2, we will partition $G^\star$ into a small number of sets s.t. each set forms an *interval* on the global order π and it is certified to be connected by some Steiner tree in $\mathcal{T}$. Thus, it suffices to solve the connectivity of intervals on $Q^\star$, which is formalized in Lemma 19.

▶ **Theorem 17.** *There exists a deterministic fully dynamic sensitivity oracle for subgraph connectivity with $O(\min\{m(n_{\text{off}}+d_\star)\log^2 n, n^2\})$ space, $O(d^2\log^7 n)$ update time and $O(d)$ query time. The preprocessing time is $\widehat{O}(m)+O(m(n_{\text{off}}+d_\star)\log^2 n)$ by a combinatorial algorithm, and $\widehat{O}(m)+O(n^\omega\log^2 n)$ using fast matrix multiplication.*

We first conclude our fully dynamic sensitivity oracle for subgraph connectivity in Theorem 17. The bounds on preprocessing time and space are shown in Section 5.3. The update time is given by Lemma 19. The query algorithm and the query time analysis are omitted here and they can be founded in the full version.

6.1 Affected Vertices $Q^\star$ and the Affected Graph $G^\star$

For each component $\gamma\in\mathcal{C}$, we call γ an *affected component* if $V(\gamma)$ intersects D_{on}, otherwise it is *unaffected*. Let $\mathcal{C}_{\text{aff}}$ denote the set of affected components. Let $\mathcal{T}_{\text{aff}}=\{\tau(\gamma)\mid\gamma\in\mathcal{C}_{\text{aff}}\}$ denote the Steiner trees corresponding to affected components.

We then define the *affected vertices* to be $Q^\star=D_{\text{off}}\cup\bigcup_{\tau\in\mathcal{T}_{\text{aff}}}U(\tau)\setminus D_{\text{on}}$. Namely, $Q^\star$ collect the newly opened vertices and the open terminals of affected components. Note that $Q^\star\subseteq V_{\text{new}}$. The *affected graph* $G^\star$ is $G^\star=\hat{G}[Q^\star]-\sum_{\gamma\in\mathcal{C}_{\text{aff}}}\hat{E}_\gamma$. In other words, $G^\star$ is the subgraph of the artificial graph $\hat{G}$ induced by the affected vertices $Q^\star$, with the artificial edges from affected components removed.

▶ **Lemma 18.** *For any two vertices $u,v\in Q^\star$, u and v are connected in $G[V_{\text{new}}]$ if and only if u and v are connected in $G^\star$.*

The proof of Lemma 18 can be founded in the full version. Intuitively, this lemma holds because those maximal unaffected components in $\mathcal{C}$ partition $V_{\text{new}}\setminus Q^\star$, and the artificial edges will capture the connectivity contributed by these maximal unaffected components.

6.2 Solving Connectivity of Intervals

Although the primary goal of our update algorithm is to compute the connectivity of $Q^\star$ on $G[V_{\text{new}}]$, Lemma 18 tells that it is equivalent to compute the connectivity of $Q^\star$ on $G^\star$.

▶ **Lemma 19.** *There is a deterministic algorithm that computes a partition $\mathcal{I}$ of $Q^\star$ s.t. each set $I\in\mathcal{I}$ forms an interval on π and all vertices in $\mathcal{I}$ are connected in $G^\star$, and then computes a partition $\mathcal{R}$ of $\mathcal{I}$ s.t. for each group $R\in\mathcal{R}$, the union of intervals in R forms a (maximal) connected component of $G^\star$. The running time is $O(p^2d^2\Delta^2\log n)$.*

Intervals

We first describe how to compute the partition $\mathcal{I}$ of $Q^\star$. Because we require each set $I\in\mathcal{I}$ forms an *interval* on the global order π, we can represent I by the positions of its endpoints on π. Recall that $Q^\star=(\bigcup_{\gamma\in\mathcal{T}_{\text{aff}}}U(\gamma)\setminus D_{\text{on}})\cup D_{\text{off}}$.

- We first construct the intervals of $\bigcup_{\tau\in\mathcal{T}_{\text{aff}}}U(\tau)\setminus D_{\text{on}}$ by exploiting the Steiner trees. For each $\tau\in\mathcal{T}_{\text{aff}}$, by invoking Lemma 14 on τ with failed vertices D_{on}, we will obtain a partition $\mathcal{I}'_\tau$ of $V(\tau)\setminus D_{\text{on}}$ s.t. each $I'\in\mathcal{I}'_\tau$ is an interval on $\text{ET}(\tau)$ and it is contained by a subtree of $\tau\setminus D_{\text{on}}$. We construct a set $\mathcal{I}_t$ of intervals on $\text{ET}(\tau)\cap U(\tau)$ by taking the restriction of intervals $\mathcal{I}'_t$ on $U(\tau)$. Therefore, intervals in $\mathcal{I}_t$ are indeed intervals on π because $\text{ET}(\tau)\cap U(\tau)$ is a consecutive sublist of π. Also, for each interval $I\in\mathcal{I}_\tau$, vertices in I are connected in $G[V_{\text{new}}]$ (because $\tau\setminus D_{\text{on}}$ is a subgraph of $G[V_{\text{new}}]$), which implies vertices in I are connected in $G^\star$ by Lemma 18.
- For each vertex $v\in D_{\text{off}}\subseteq Q^\star$, we construct a singleton interval $I_v=\{v\}$.

Finally, the whole set of intervals is $\mathcal{I} = \bigcup_{\tau \in \mathcal{T}_{\text{aff}}} \mathcal{I}_\tau \cup \{I_v \mid v \in D_{\text{off}}\}$.

▶ **Proposition 20.** *The total number of intervals is $|\mathcal{I}| = O(pd\Delta)$, and computing all intervals takes $O(pd\Delta \log(d\Delta))$ time.*

Proof. By Lemma 14, the number of intervals generated by a tree $\tau \in \mathcal{T}_{\text{aff}}$ is at most $O(|V(\tau) \cap D_{\text{on}}| \cdot \Delta)$, and it takes $O(|V(\tau) \cap D_{\text{on}}| \cdot \Delta \cdot \log(|V(\tau) \cap D_{\text{on}}| \cdot \Delta))$ time to generate them. Observe that $\sum_{\tau \in \mathcal{T}_{\text{aff}}} |V(\tau) \cap D_{\text{on}}| = O(p \cdot |D_{\text{on}}|)$ because each vertex in D_{on} can appear in at most p trees in $\mathcal{T}$ (at most one at each level). Furthermore, the trivial intervals generated by vertices in D_{off} is obviously $|D_{\text{off}}|$. Therefore, the total number of intervals in $O(p|D_{\text{on}}|\Delta) + |D_{\text{off}}| = O(pd\Delta)$, and computing all intervals takes $O(pd\Delta(d\Delta))$ time. ◀

Borůvka's Algorithm

We now discuss how to compute the partition $\mathcal{R}$ of $\mathcal{I}$. We will merge the intervals by a Borůvka's styled algorithm. The algorithm has several *phases*, and each phase j receives a partition $\mathcal{R}^{(j)}$ of $\mathcal{I}$ as input. each group $R \in \mathcal{R}^{(j)}$ is either *active* or *inactive*. Initially, $\mathcal{R}^{(1)} = \{\{I\} \mid I \in \mathcal{I}\}$ is the trivial partition of $\mathcal{I}$ and all groups in $\mathcal{R}^{(1)}$ are active. For each phase j, we do the following to update $\mathcal{R}^{(j)}$ to $\mathcal{R}^{(j+1)}$.

1. For each active group R in $\mathcal{R}^{(j)}$, we will ask the following *adjacency query*.
   - **(Q1)** Given an active group $R \in \mathcal{R}^{(j)}$, find another active group $R' \in \mathcal{R}^{(j)}$ s.t. there exists an edge $e = \{u, v\} \in E(G^\star)$ with $u \in I_u \in R$ and $v \in I_v \in R'$, or claim that there is no such R'.

   After asking (Q1) for all active groups, for each active group R, if (Q1) tells that no such R' exists, we mark R as an inactive group, otherwise we find an *adjacent group-pair* $\{R, R'\}$.
2. Given the adjacent group-pairs in step 1, we construct a graph K with vertices corresponding to active groups and edges corresponding to adjacent group-pairs. Note that for each adjacent group-pair $\{R, R'\}$, R and R' must still be active. Then, for each connected component of K, we merge the groups inside it into a new active group.

The algorithm terminates once it reaches a phase $\bar{j}$ s.t. all groups in $\mathcal{R}^{(\bar{j})}$ are inactive, and we let $\mathcal{R} = \mathcal{R}^{(\bar{j})}$ be the final output. Obviously, $\mathcal{R}$ satisfies the output requirement of Lemma 19. Furthermore, the number of phases is bounded in Proposition 21. Let $\mathcal{R}^{(j)}_{\text{act}} \subseteq \mathcal{R}^{(j)}$ denote the active groups in $\mathcal{R}^{(j)}$ at the moment when phase j starts and let $\bar{k}^{(j)} = |\mathcal{R}^{(j)}_{\text{act}}|$.

▶ **Proposition 21.** *For each $j \geq 2$, $\bar{k}^{(j)} \leq \bar{k}^{(j-1)}/2$. The number of phases is $O(\log |\mathcal{I}|)$.*

Proof. At each phase, the number of active groups is halved because we mark all old active groups without adjacent group inactive in step 1, and each connected component of the graph K in step 2 contains at least two old active groups. Because initially $\bar{k}^{(0)} = |\mathcal{R}^{(0)}| = |\mathcal{I}|$, the number of phases is $O(\log |\mathcal{I}|)$. ◀

Next, we will discuss the implementation of step 1. Basically, for each phase j, we need an algorithm that answers the *adjacency query* (Q1) efficiently. Instead of answering (Q1) directly, we will reduce (Q1) to the following *batched adjacency query* (Q2). We give an arbitrary order to the groups in $\mathcal{R}^{(j)}_{\text{act}}$, denoted by $\mathcal{R}^{(j)}_{\text{act}} = \{R^{(j)}_1, R^{(j)}_2, ..., R^{(j)}_{\bar{k}^{(j)}}\}$.

- **(Q2)** Given a group $R^{(j)}_k \in \mathcal{R}^{(j)}_{\text{act}}$ and a batch of consecutive groups $R^{(j)}_\ell, R^{(j)}_{\ell+1}, ..., R^{(j)}_r \in \mathcal{R}^{(j)}_{\text{act}}$ s.t. $k \notin [\ell, r]$, decide if there exists $R^{(j)}_{k'}$ s.t. $k' \in [\ell, r]$ and $R^{(j)}_{k'}$ is adjacent to $R^{(j)}_k$.

▶ **Lemma 22.** *At phase j, one adjacency query can be reduced to $O(\log \bar{k}^{(j)})$ batched adjacency queries.*

Proof. Consider an adjacency query for some $R_k^{(j)} \in \mathcal{R}_{\text{act}}^{(j)}$. We can either find some $R_{k'}^{(j)} \in \mathcal{R}_{\text{act}}^{(j)}$ s.t. $k+1 \le k' \le \bar{k}^{(j)}$ and $R_{k'}^{(j)}$ is adjacent to $R_k^{(j)}$ or claim there is no such $R_{k'}^{(j)}$ in the following way: first fix $\ell = k+1$, and then perform a binary search on r in range $[k+1, \bar{k}^{(j)}]$, in which each binary search step is guided by a batched adjacency query with parameters k, ℓ, r. Similarly, we can try to find an adjacent group $R_{k'}^{(j)}$ to the left of $R_k^{(j)}$ by fixing $r = k-1$ and performing a binary search on ℓ in range $[1, k-1]$. The total number of calls to (Q2) is obviously $O(\log \bar{k}^{(j)})$ in these two binary searches. ◀

To answer batched adjacency queries in each phase, we will first introduce some *additional structures*, and then use them to design the algorithm answering (Q2), which is formalized in Lemma 23.

▶ **Lemma 23.** *There is a deterministic algorithm that computes some additional structures in $O(p^2d^2\Delta^2 \log n)$ time to support any batched adjacency query in $O(pd)$ time.*

We are now ready to analyse the running time of the Borůvka's algorithm, which completes the proof of Lemma 19. At each phase j, the number of adjacency queries is at most $\bar{k}^{(j)}$ (one for each active group in $\mathcal{R}^{(j)}$), so the number of batched adjacency queries is $O(\bar{k}^{(j)} \log \bar{k}^{(j)})$ by Lemma 22. Thus the total number of batched adjacency queries is $\sum_{j \ge 1} O(\bar{k}^{(j)} \log \bar{k}^{(j)}) = O(|\mathcal{I}| \log |\mathcal{I}|)$ by Proposition 21. By Lemma 23, the total running time of step 1 is $O(p^2d^2\Delta^2 \log n) + O(pd|\mathcal{I}| \log |\mathcal{I}|) = O(p^2d^2\Delta^2 \log n)$. The total running time of the Borůvka's algorithm is asymptotically the same because step 2 takes little time.

In what follows, we prove Lemma 23.

The Additional Structures

We start with introducing some notations. For a group $R \subseteq \mathcal{I}$, we use $V(R) = \bigcup_{I \in R} I$ to denote its vertex set. For two disjoint groups $R_1, R_2 \subseteq \mathcal{I}$ and a (multi) set E of undirected edges, let $\delta_E(R_1, R_2)$ denote the number of edges in E with one endpoint in $V(R_1)$ and the other one in $V(R_2)$. Also, recall that we gave an order to groups in $\mathcal{R}_{\text{act}}^{(j)}$, denoted by $\mathcal{R}_{\text{act}}^{(j)} = \{R_1^{(j)}, ..., R_{\bar{k}^{(j)}}^{(j)}\}$.

For each phase j, we will construct the following data structures.

- First, we construct a two-dimensional $(\bar{k}^{(j)} \times \bar{k}^{(j)})$-array $\mathsf{CountAll}^{(j)}$, where for each $1 \le x, y \le \bar{k}^{(j)}$, the entry $\mathsf{CountAll}^{(j)}(x, y) = \delta_{E(\hat{G})}(R_x^{(j)}, R_y^{(j)})$. Furthermore, we store the 2D-prefix sum of $\mathsf{CountAll}^{(j)}$.
- For each affected component γ, we prepare a one-dimensional array $\mathsf{CountA}_\gamma^{(j)}$ with length $\bar{k}^{(j)}$, where for each $1 \le x \le \bar{k}^{(j)}$, the entry $\mathsf{CountA}_\gamma^{(j)}(x) = |A_\gamma \cap V(R_x^{(j)})|$. Similarly, we construct an one-dimensional array $\mathsf{CountB}_\gamma^{(j)}$ with length $\bar{k}^{(j)}$ in which the entry $\mathsf{CountB}_\gamma^{(j)}(x) = |B_\gamma \cap V(R_x^{(j)})|$. Furthermore, we store the prefix sum of $\mathsf{CountA}_\gamma^{(j)}$ and $\mathsf{CountB}_\gamma^{(j)}$.

▶ **Lemma 24.** *The total construction time of arrays $\mathsf{CountAll}^{(j)}$, $\mathsf{CountA}_\gamma^{(j)}$ and $\mathsf{CountB}_\gamma^{(j)}$ summing over all phases j and all affected components γ is $O(p^2d^2\Delta^2 \log d)$.*

Proof. We first initialize $\mathsf{CountAll}^{(1)}, \mathsf{CountA}_\gamma^{(1)}, \mathsf{CountB}_\gamma^{(1)}$ for phase 1. For each entry $\mathsf{CountAll}^{(1)}(x, y)$ of $\mathsf{CountAll}^{(1)}$, note that $R_x^{(1)}$ and $R_y^{(1)}$ are both singleton groups. Let I_x and I_y be the intervals in $R_x^{(1)}$ and $R_y^{(1)}$. Then $\mathsf{CountAll}^{(1)}(x, y)$ is exactly the number of $E(\hat{G})$-edges that connect I_x and I_y, which can be answered by querying Table in $O(\log n)$ time by Lemma 16 because I_x and I_y are intervals on the global order π. For an entry $\mathsf{CountA}_\gamma^{(1)}(x)$ of $\mathsf{CountA}_\gamma^{(1)}$, let I_x be the single interval in $R_x^{(1)}$, and we can easily compute

$|A_\gamma \cap I_x|$ by binary search in $O(\log n)$ time because I_x is an interval on π and A_γ is ordered consistently with π. Similarly, we can compute the array $\mathsf{CountB}_\gamma^{(1)}$. The construction time of additional structures at phase 1 is $O(((\bar{k}^{(1)})^2 + |\mathcal{C}_{\text{aff}}| \cdot \bar{k}^{(1)}) \log n)$.

For each phase $j \geq 2$, we will compute $\mathsf{CountAll}^{(j)}, \mathsf{CountA}_\gamma^{(j)}, \mathsf{CountB}_\gamma^{(j)}$ based on the arrays of phase $j-1$. For an entry $\mathsf{CountAll}^{(j)}(x,y)$ of $\mathsf{CountAll}^{(j)}$, recall that $R_x^{(j)}$ is the union of several groups $R_{x_1}^{(j-1)}, R_{x_2}^{(j-1)}, ...$ inside $\mathcal{R}_{\text{act}}^{(j-1)}$, and $R_y^{(j)} = R_{y_1}^{(j-1)} \cup R_{y_2}^{(j-1)} \cup ....$ Furthermore, $x_1, x_2, ..., y_1, y_2, ...$ are distinct indexes in $[1, \bar{k}^{(j-1)}]$. Therefore,

$$\mathsf{CountAll}^{(j)}(x,y) = \sum_{x'=x_1,x_2,...} \sum_{y'=y_1,y_2,..} \mathsf{CountAll}^{(j-1)}(x',y').$$

We can compute $\mathsf{CountA}_\gamma^{(j)}$ and $\mathsf{CountB}_\gamma^{(j)}$ in a similar way. The construction time of additional structures at phase j is proportional to the total size of additional structures at phase $j-1$, i.e. $O((\bar{k}^{(j-1)})^2 + |\mathcal{C}_{\text{aff}}| \cdot \bar{k}^{(j-1)})$.

The overall construction time is

$$O(((\bar{k}^{(1)})^2 + |\mathcal{C}_{\text{aff}}| \cdot \bar{k}^{(1)}) \log n) + \sum_{j \geq 2} O((\bar{k}^{(j-1)})^2 + |\mathcal{C}_{\text{aff}}| \cdot \bar{k}^{(j-1)}) = O(p^2 d^2 \Delta^2 \log n),$$

because $\bar{k}^{(1)} = |\mathcal{I}| = O(pd\Delta)$, $|\mathcal{C}_{\text{aff}}| = O(pd)$ and $\bar{k}^{(j)} \leq \bar{k}^{(j-1)}/2$ for each phase j. ◀

Answering Batched Adjacency Queries

Consider a batched adjacency query at phase j with parameters k, ℓ, r. It is equivalent to decide whether the number of $G^\star$-edges connecting $R_k^{(j)}$ and some $R_{k'}^{(j)}$ where $k' \in [\ell, r]$ is greater than zero or not. Namely, it suffices to decide whether

$$\sum_{\ell \leq k' \leq r} \delta_{E(G^\star)}(R_k^{(j)}, R_{k'}^{(j)}) > 0. \tag{1}$$

▶ **Lemma 25.** *For any two disjoint groups $R_1, R_2 \subseteq \mathcal{I}$,*

$$\delta_{E(G^\star)}(R_1, R_2) = \delta_{E(\hat{G})}(R_1, R_2) - \sum_{\gamma \in \mathcal{C}_{\text{aff}}} \delta_{\hat{E}_\gamma}(R_1, R_2).$$

Proof. First the RHS is equal to $\delta_{E(\hat{G}) - \sum_{\gamma \in \mathcal{C}_{\text{aff}}} \hat{E}_\gamma}(R_1, R_2)$ because $\hat{E}_\gamma$ of all $\gamma \in \mathcal{C}_{\text{aff}}$ are disjoint subsets of $E(\hat{G})$ (note that $E(\hat{G})$ is defined to be a multiset).

Recall that $G^\star = \hat{G}[Q^\star] - \sum_{\gamma \in \mathcal{C}_{\text{aff}}} \hat{E}_\gamma$. The LHS is at most the RHS because $E(G^\star) \subseteq E(\hat{G}) - \sum_{\gamma \in \mathcal{C}_{\text{aff}}} \hat{E}_\gamma$. On the other direction, each edge in $E(\hat{G})$ connecting $V(R_1)$ and $V(R_2)$ is inside $\hat{G}[Q^\star]$ since $V(R_1), V(R_2) \subseteq Q^\star$, so the RHS is at most the LHS. ◀

▶ **Lemma 26.** *For each $\gamma \in \mathcal{C}_{\text{aff}}$ and two disjoint groups $R_1, R_2 \subseteq \mathcal{I}$,*

$$\delta_{\hat{E}_\gamma}(R_1, R_2) = |A_\gamma \cap V(R_1)| \cdot |A_\gamma \cap V(R_2)| - |(A_\gamma \setminus B_\gamma) \cap V(R_1)| \cdot |(A_\gamma \setminus B_\gamma) \cap V(R_2)|.$$

Proof. Recall that $\hat{E}_\gamma$ is the union of a clique on B_γ and a biclique between $A_\gamma \setminus B_\gamma$ and B_γ. In other words, $\hat{E}_\gamma$ is a clique on A_γ with the clique on $A_\gamma \setminus B_\gamma$ removed. Because $V(R_1)$ and $V(R_2)$ are disjoint, the equation follows. ◀

Using Lemma 25 and Lemma 26, we can rewrite the LHS of inequality 1 as follows.

$$\begin{aligned}\sum_{\ell\le k'\le r}\delta_{E(G^\star)}(R_k^{(j)},R_{k'}^{(j)}) &= \sum_{\ell\le k'\le r}\delta_{E(\hat{G})}(R_k^{(j)},R_{k'}^{(j)}) - \sum_{\ell\le k'\le r}\sum_{\gamma\in\mathcal{C}_{\text{aff}}}\delta_{\hat{E}_\gamma}(R_k^{(j)},R_{k'}^{(j)})\\ &= \sum_{\ell\le k'\le r}\delta_{E(\hat{G})}(R_k^{(j)},R_{k'}^{(j)})\\ &- \sum_{\gamma\in\mathcal{C}_{\text{aff}}}\sum_{\ell\le k'\le r}|A_\gamma\cap V(R_k^{(j)})|\cdot|A_\gamma\cap V(R_{k'}^{(j)})|\\ &+ \sum_{\gamma\in\mathcal{C}_{\text{aff}}}\sum_{\ell\le k'\le r}|(A_\gamma\setminus B_\gamma)\cap V(R_k^{(j)})|\cdot|(A_\gamma\setminus B_\gamma)\cap V(R_{k'}^{(j)})|.\end{aligned}$$

For convenience, we denote $\alpha_\gamma^{(j)}(k)=\mathsf{CountA}_\gamma^{(j)}(k)$, $\beta_\gamma^{(j)}(k)=\mathsf{CountB}_\gamma^{(j)}(k)$
Combining the definition of the additional structures, we further have

$$\begin{aligned}\sum_{\ell\le k'\le r}\delta_{E(G^\star)}(R_k^{(j)},R_{k'}^{(j)}) &= \sum_{\ell\le k'\le r}\mathsf{CountAll}^{(j)}(k,k') - \sum_{\gamma\in\mathcal{C}_{\text{aff}}}\left(\alpha_\gamma^{(j)}(k)\cdot\sum_{\ell\le k'\le r}\alpha_\gamma^{(j)}(k')\right)\\ &+ \sum_{\gamma\in\mathcal{C}_{\text{aff}}}\left((\alpha_\gamma^{(j)}(k)-\beta_\gamma^{(j)}(k))\cdot\sum_{\ell\le k'\le r}(\alpha_\gamma^{(j)}(k')-\beta_\gamma^{(j)}(k'))\right)\end{aligned}$$

Because we have stored the prefix sum of the arrays $\mathsf{CountAll}^{(j)}$, $\mathsf{CountA}_\gamma^{(j)}$, $\mathsf{CountB}_\gamma^{(j)}$, computing the value of the above expression takes $O(|\mathcal{C}_{\text{aff}}|)=O(pd)$ time.

References

1. Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.
2. Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental SSSP and approximate min-cost flow in almost-linear time. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1000–1008. IEEE, 2021. doi:10.1109/FOCS52979.2021.00100.
3. Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1158–1167. IEEE, 2020. doi:10.1109/FOCS46700.2020.00111.
4. Ran Duan and Seth Pettie. Connectivity oracles for failure prone graphs. In *Proceedings of the forty-second ACM Symposium on Theory of Computing*, pages 465–474, 2010.
5. Ran Duan and Seth Pettie. Connectivity oracles for graphs subject to vertex failures. *SIAM Journal on Computing*, 49(6):1363–1396, 2020.
6. Martin Furer and Balaji Raghavachari. Approximating the minimum-degree steiner tree to within one of optimal. *Journal of Algorithms*, 17(3):409–423, 1994.
7. Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.

8 Monika Henzinger and Stefan Neumann. Incremental and fully dynamic subgraph connectivity for emergency planning. In *24th Annual European Symposium on Algorithms (ESA 2016)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.

9 Bingbing Hu, Evangelos Kosinas, and Adam Polak. Connectivity oracles for predictable vertex failures. *CoRR*, abs/2312.08489, 2023. doi:10.48550/arXiv.2312.08489.

10 Arkady Kanevsky, Roberto Tamassia, Giuseppe Di Battista, and Jianer Chen. On-line maintenance of the four-connected components of a graph. In *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 793–801. IEEE Computer Society, 1991.

11 Rohit Khandekar, Subhash Khot, Lorenzo Orecchia, and Nisheeth K Vishnoi. On a cut-matching game for the sparsest cut problem. *Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2007-177*, 6(7):12, 2007.

12 Rohit Khandekar, Satish Rao, and Umesh V. Vazirani. Graph partitioning using single commodity flows. *J. ACM*, 56(4):19:1–19:15, 2009. doi:10.1145/1538902.1538903.

13 Evangelos Kosinas. Connectivity queries under vertex failures: Not optimal, but practical. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 75:1–75:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.75.

14 Yaowei Long and Thatchaphol Saranurak. Near-optimal deterministic vertex-failure connectivity oracles. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1002–1010. IEEE, 2022.

15 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992. doi:10.1007/BF01758778.

16 Merav Parter, Asaf Petruschka, and Seth Pettie. Connectivity labeling and routing with multiple vertex failures. In *Proceedings 56th ACM Symposium on Theory of Computing (STOC)*, 2024.

17 Michal Pilipczuk, Nicole Schirrmacher, Sebastian Siebertz, Szymon Torunczyk, and Alexandre Vigny. Algorithms and data structures for first-order logic with connectivity under vertex failures. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 102:1–102:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICALP.2022.102.

18 Jan van den Brand and Thatchaphol Saranurak. Sensitive distance and reachability oracles for large batch updates. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 424–435. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00034.

Impagliazzo's Worlds Through the Lens of Conditional Kolmogorov Complexity

Zhenjian Lu
University of Warwick, UK

Rahul Santhanam
University of Oxford, UK

Abstract

We develop new characterizations of Impagliazzo's worlds Algorithmica, Heuristica and Pessiland by the intractability of conditional Kolmogorov complexity K and conditional probabilistic time-bounded Kolmogorov complexity pK^t.

In our first set of results, we show that $\mathsf{NP} \subseteq \mathsf{BPP}$ iff $\mathsf{pK}^t(x \mid y)$ can be computed efficiently in the worst case when t is sublinear in $|x| + |y|$; $\mathsf{DistNP} \subseteq \mathsf{HeurBPP}$ iff $\mathsf{pK}^t(x \mid y)$ can be computed efficiently over all polynomial-time samplable distributions when t is sublinear in $|x| + |y|$; and infinitely-often one-way functions fail to exist iff $\mathsf{pK}^t(x \mid y)$ can be computed efficiently over all polynomial-time samplable distributions for t a sufficiently large polynomial in $|x| + |y|$. These results characterize Impagliazzo's worlds Algorithmica, Heuristica and Pessiland purely in terms of the tractability of conditional pK^t. Notably, the results imply that Pessiland fails to exist iff the average-case intractability of conditional pK^t is insensitive to the difference between sublinear and polynomially bounded t. As a corollary, while we prove conditional pK^t to be NP-hard for sublinear t, showing NP-hardness for large enough polynomially bounded t would eliminate Pessiland as a possible world of average-case complexity.

In our second set of results, we characterize Impagliazzo's worlds Algorithmica, Heuristica and Pessiland by the distributional tractability of a natural problem, i.e., approximating the conditional Kolmogorov complexity, that is provably intractable in the worst case. We show that $\mathsf{NP} \subseteq \mathsf{BPP}$ iff conditional Kolmogorov complexity can be approximated in the *semi-worst case*; and $\mathsf{DistNP} \subseteq \mathsf{HeurBPP}$ iff conditional Kolmogorov complexity can be approximated on average over all *independent polynomial-time samplable distributions*. It follows from a result by Ilango, Ren, and Santhanam (STOC 2022) that infinitely-often one-way functions fail to exist iff conditional Kolmogorov complexity can be approximated on average over all *polynomial-time samplable distributions*. Together, these results yield the claimed characterizations. Our techniques, combined with previous work, also yield a characterization of auxiliary-input one-way functions and equivalences between different average-case tractability assumptions for conditional Kolmogorov complexity and its variants. Our results suggest that novel average-case tractability assumptions such as tractability in the semi-worst case and over independent polynomial-time samplable distributions might be worthy of further study.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases meta-complexity, Kolmogorov complexity, one-way functions, average-case complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.110

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2024/085/` [20]

Acknowledgements We thank Shuichi Hirahara, Yanyi Liu, Igor C. Oliveira, and Hanlin Ren for useful discussions.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 110; pp. 110:1–110:17

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In his influential survey on average-case complexity [12], Impagliazzo described five possible computational worlds: Algorithmica, Heuristica, Pessiland, Minicrypt and Cryptomania. Algorithmica is a world where NP is easy in the worst case; Heuristica a world where NP is hard in the worst case but easy on average; Pessiland a world where NP is hard on average but one-way functions do not exist; Minicrypt a world where one-way functions exist but public-key cryptography does not; and Cryptomania a world where public-key cryptography exists. The general belief among complexity theorists and cryptographers is that we live in Cryptomania, but we are very far from a proof, as even ruling out Algorithmica would involve showing $\mathsf{NP} \neq \mathsf{P}$.

There is the possibility, however, that we might be able to unconditionally rule out some of the intermediate worlds, such as Heuristica, Pessiland and Minicrypt. Until recently, there was little progress on ruling out these intermediate worlds. All that was known was that there are various black-box and relativization barriers to ruling out these worlds.

The study of *meta-complexity*, i.e., the complexity of computational problems that are themselves about complexity, has enabled new attacks on these questions. Examples of meta-complexity problems are the Minimum Circuit Size Problem (MCSP), which asks whether a Boolean function represented by its truth table has circuits of a given size, and the problem of computing Kolmogorov complexity and its resource-bounded variants such as Levin's time-bounded Kolmogorov complexity. The average-case complexity of meta-complexity problems is of particular interest [9]. Hirahara [5] gave an approach via meta-complexity to ruling out the analogue of Heuristica for the Polynomial Hierarchy. More recently, the Polynomial Hierarchy analogue of Pessiland has been ruled out [10], again using meta-complexity techniques.

There have been several successful efforts to characterize the existence of one-way functions via meta-complexity. In [23], a conditional characterization was given, based on a believable but seemingly hard-to-establish conjecture. Liu and Pass [14] unconditionally characterized one-way functions by the average-case hardness of polynomial-time-bounded Kolmogorov complexity over the uniform distribution. This characterization was extended to other meta-complexity problems and notions of one-way function in [15, 21, 1]. A different characterization of one-way functions via the hardness of approximating Kolmogorov complexity over samplable distributions was given in [11]. More recently, Hirahara [7] introduced a meta-complexity problem whose NP-hardness and the worst-case hardness of NP characterize the existence of one-way functions.

These connections between meta-complexity, average-case complexity and one-way functions raise the following question: Can we characterize Impagliazzo's worlds Algorithmica, Heuristica and Pessiland by different notions of hardness for a single computational problem? A positive answer to this question is implicit in [16], who study the problem of conditional polynomial-time-bounded Kolmogorov complexity. They show that the worst-case hardness of conditional polynomial-time-bounded Kolmogorov complexity captures worst-case hardness of NP, and the average-case hardness of conditional polynomial-time-bounded Kolmogorov complexity over the uniform distribution captures the existence of one-way functions. Their result on worst-case hardness immediately implies that the average-case hardness of NP is equivalent to the hardness of conditional polynomial-time-bounded Kolmogorov complexity over some samplable distribution.

In this work, we give two new characterizations of Impagliazzo's worlds by different notions of hardness for a single problem - first for conditional probabilistic time-bounded Kolmogorov complexity pK^t [3], and second for the standard notion of conditional Kolmogorov complexity.

These new characterizations have some interesting features. The first characterization implies that ruling out Pessiland corresponds to *robustness* of the average-case tractability of conditional pK^t over time regimes t that vary from sublinear to polynomial. As a consequence, while we are able to prove (by building on [6]) that pK^t is NP-hard to compute exactly when t is sublinear, Pessiland would fail to exist if pK^t were NP-hard to compute for *arbitrary* polynomial t. This could be a promising route to ruling out Pessiland, since pK^t is a fairly powerful complexity measure with nice properties such as the coding theorem which could potentially be exploited when showing hardness, and the computational version is in (promise) AM but is not known to be in NP.

The second characterization is for a fundamental problem that is *provably intractable in the worst case*, i.e., the problem of approximating conditional Kolmogorov complexity. A somewhat surprising aspect of our results (which is also present in the main result of [11] on which we build) is that conditional Kolmogorov complexity is uncomputable, yet natural average-case hardness assumptions on conditional Kolmogorov complexity capture complexity worlds related to average-case hardness of NP. What this indicates is that the distinctions between Impagliazzo's worlds can be encoded in a natural way into the *distributional assumptions* that are made, while considering a single well-understood problem.

As a corollary of our second set of results together with those in [16], we get new equivalences between hardness assumptions for conditional Kolmogorov complexity and hardness assumptions for conditional time-bounded Kolmogorov complexity. The proofs of these equivalences crucially use the various characterizations of Impagliazzo's worlds, and it seems tricky to show such equivalences directly.

1.1 Results

We state our results formally in this subsection.

1.1.1 Characterizing Both $\mathsf{DistNP} \subseteq \mathsf{HeurBPP}$ and Non-Existence of One-Way Functions by Average-Case Easiness of Conditional pK^t

We present a meta-complexity problem whose average-case tractability over polynomial-time samplable distributions can be used to characterize both the non-existence of one-way functions and $\mathsf{DistNP} \subseteq \mathsf{HeurBPP}$, while considering different time regimes in the measure of time-bounded Kolmogorov complexity. Specifically, we consider the problem of computing conditional probabilistic t-time-bounded Kolmogorov complexity.

As defined in [3], we let $\mathsf{pK}^t_\lambda(x \mid y)$ be the smallest integer k such that, with probability at least λ over the choice of a random string $w \sim \{0,1\}^t$, there is a (deterministic) program of size k that, when running on w and given oracle access to y, prints x within t steps (see [20, Definition 16] for the formal definition).

For $\tau\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, let $\mathsf{Cond\text{-}pK}[\tau]$ be the following promise problem $(\mathsf{YES}, \mathsf{NO})$:

$$\mathsf{YES} := \left\{ (x, y, 1^s) \mid \mathsf{pK}^{\tau(|x|,|y|)}_{2/3}(x \mid y) \leq s \right\},$$
$$\mathsf{NO} := \left\{ (x, y, 1^s) \mid \mathsf{pK}^{\tau(|x|,|y|)}_{1/3}(x \mid y) > s \right\}.$$

We will refer to this problem as "computing conditional pK^t".

We will consider two specific settings for the time bound function τ. For the purpose of illustration, let us consider the following simplified problem. For $\tau\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, we are given x, y and s, and the task is to decide whether $\mathsf{K}^{\tau(|x|,|y|)}(x \mid y) \leq s$, i.e., whether there is a program of size at most s such that given *oracle access* to y, the program outputs x within time $\tau(|x|, |y|)$.

A typical setting of τ is $\tau(n,m) := n^c \cdot m^c$, where $c > 1$ is some constant. For this τ, we want to decide if there is a program of size at most s that, given *oracle access* to y, outputs x within time $\tau(|x|,|y|)$, and such a program has enough time to read the entire string y.

Now consider another setting of τ where $\tau(n,m) := n^c \cdot m^{1-1/c}$ for a constant $c > 1$. In this case, for a string $y \in \{0,1\}^m$, where $m := n^{2c^2}$, we have

$$\tau(n,m) = n^c \cdot m^{1-1/c} = n^{2c^2-c} \ll m.$$

Again, we want to decide if there is a program of size at most s that, given *oracle access* to y, outputs x within time $\tau(|x|,|y|)$. However, in this case any such program does not have time to read the entire string y

We will show that the non-existence of one-way functions corresponds to the average-case tractability of $\mathsf{Cond\text{-}pK}[\tau]$ over polynomial-time samplable distributions for the "polynomial-time regime" of τ, and that $\mathsf{DistNP} \subseteq \mathsf{HeurBPP}$ corresponds to that of the "sublinear-time regime".[1] We state our results formally next.

For an algorithm A, $x, y \in \{0,1\}^*$, and $s \in \mathbb{N}$, we say that A *decides* $\mathsf{Cond\text{-}pK}[\tau]$ *on* $(x, y, 1^s)$ if the following holds:

$$A(x,y,1^s) = \begin{cases} 1 & \text{if } \mathsf{pK}^{\tau(|x|,|y|)}_{2/3}(x \mid y) \le s, \\ 0 & \text{if } \mathsf{pK}^{\tau(|x|,|y|)}_{1/3}(x \mid y) > s, \\ \textbf{either } 0 \textbf{ or } 1 & \text{otherwise.} \end{cases}$$

▶ **Theorem 1.** *The following are equivalent.*

1. *Infinitely-often one-way functions do not exist.*
2. **(Computing conditional pK^t in the polynomial-time regime is easy-on-average over samplable distributions.)**
   For every polynomial-time samplable distribution family $\{\mathcal{D}_{\langle n,m\rangle}\}_{n,m}$ supported over $\{0,1\}^n \times \{0,1\}^m$, every polynomial q, and for all large enough constant c, there exists a probabilistic polynomial-time algorithm A such that for all $n, m, s \in \mathbb{N}$,

$$\Pr_{(x,y)\sim\mathcal{D}_{\langle n,m\rangle}}[A \text{ decides } \mathsf{Cond\text{-}pK}[\tau] \text{ on } (x,y,1^s)] \ge 1 - \frac{1}{q(n,m)},$$

   where $\tau(n,m) := n^c \cdot m^c$.

▶ **Theorem 2.** *The following are equivalent.*

1. $\mathsf{DistNP} \subseteq \mathsf{HeurBPP}$.
2. **(Computing conditional pK^t in the sublinear-time regime is easy-on-average over samplable distributions.)**
   For every polynomial-time samplable distribution family $\{\mathcal{D}_{\langle n,m\rangle}\}_{n,m}$ supported over $\{0,1\}^n \times \{0,1\}^m$, every polynomial q, and for all large enough constant c, there exists a probabilistic polynomial-time algorithm A such that for all $n, m, s \in \mathbb{N}$,

$$\Pr_{(x,y)\sim\mathcal{D}_{\langle n,m\rangle}}[A \text{ decides } \mathsf{Cond\text{-}pK}[\tau] \text{ on } (x,y,1^s)] \ge 1 - \frac{1}{q(n,m)},$$

   where $\tau(n,m) := n^c \cdot m^{1-1/c}$.

[1] Note that even in the "sublinear-time regime" of τ, the program can still run in polynomial time with respect to the the length of x; the word "sublinear-time" refers to the fact that the program runs in sublinear time with respect to the length of y.

In proving Theorem 2, we also show that it is NP-hard to compute conditional pK^t in the sublinear-time regime in the worst case.

▶ **Theorem 3** (Informal). *For any constant $c > 1$, Cond-$\mathsf{pK}[\tau]$ is NP-hard under randomized polynomial-time reductions, where $\tau(n, m) := n^c \cdot m^{1-1/c}$.*

In fact, Theorem 3 holds even if we consider the problem of approximating $\mathsf{pK}^t(x \mid y)$ in the sublinear-time regime within a multiplicative factor of $|x|^{1/\log\log|x|^{O(1)}}$. This also extends a result by Liu and Pass [16] and Hirahara [6], which showed that the problem of computing/approximating conditional K^t in the sublinear-time regime is NP-hard.

Theorem 3, Theorem 1 and Theorem 2 together give characterizations of Impagliazzo's worlds Algorithmica, Heuristica and Pessiland based on different hardness assumptions for the computation of conditional pK^t.

In particular, Theorem 1 and Theorem 2 imply that the task of ruling out Pessiland[2] is equivalent to showing that the problem of computing conditional pK^t on average over polynomial-time samplable distributions is robust with respect to the two different time regimes.

Also, we get that to rule out Pessiland, it suffices to show that it is NP-hard to compute conditional pK^t in the *polynomial-time regime* in the worst case.

▶ **Corollary 4** (Informal. See [20, Corollary 55] for the formal version). *If computing conditional pK^t in the polynomial-time regime is NP-hard, then Pessiland does not exist.*

A proof sketch of Corollary 4 can be found in [20, Section 4.4].

For comparison, it was observed in [7] that if one can show the NP-hardness of *approximating* a certain variant of time-bounded Kolmogorov complexity called q^t, then Pessiland does not exist. It is known that $\mathsf{q}^{\mathsf{poly}}$ and $\mathsf{pK}^{\mathsf{poly}}$ are equivalent to each other up to an additive logarithmic factor. This implies that showing the NP-hardness of *approximating* pK^t will allow us to rule out Pessiland.[3] It can also be shown that the problem of *approximating* pK^t is reducible to that of computing conditional pK^t.[4] On the other hand, Corollary 4 only requires showing the NP-hardness of computing conditional pK^t, which might be easier. Moreover, we note that the barrier of [22] to showing NP-hardness of approximating Kolmogorov complexity and its variants does not seem to apply directly to exact computation.

Equivalences between Average-Case Easiness of Approximating and Computing (Conditional) pK^t

By combining Theorem 1 with existing characterizations of one-way functions, we get that the average-case easiness of approximating and computing different variants of probabilistic (conditional) time-bounded Kolmogorov complexity are in fact equivalent. We state this result more formally below.

We say that "approximating pK^t is easy-on-average over samplable distributions" if the following holds.

---

[2] In this case, we mean basing infinitely-often one-way functions on DistNP $\not\subseteq$ HeurBPP.

[3] Here, we refer to the problem called Gap-MINpKT. For a polynomial τ, Gap-MINpK$\mathsf{T}[\tau]$ is the (promise) problem of deciding, given as input $(x, 1^s, 1^t)$, whether $\mathsf{pK}^t(x) \le s$ or $\mathsf{pK}^{\tau(|x|,t)}(x) > s + \log\tau(|x|, t)$.

[4] More precisely, if we can solve Cond-$\mathsf{pK}[\tau]$ for some polynomial τ, then we can also solve Gap-MINpK$\mathsf{T}[\tau']$ for some polynomial τ'.

> For every polynomial-time samplable distribution family $\{\mathcal{D}_n\}_n$ supported over $\{0,1\}^n$, every polynomial q, and for all large enough polynomial τ, there is a probabilistic polynomial-time algorithm A that can decide, given as input $(x, 1^s, 1^t)$, whether $\mathsf{pK}^t(x) \leq s$ or $\mathsf{pK}^{\tau(|x|,t)}(x) > s + \log \tau(|x|, t)$,[5] with probability at least $1 - 1/q(n)$ over $x \sim \mathcal{D}_n$ and the internal randomness of A.

The above can be naturally generalized to the conditional setting, where we consider any samplable distribution family $\{\mathcal{D}_{\langle n,m\rangle}\}_{n,m}$ supported over $\{0,1\}^n \times \{0,1\}^m$, and for all large enough polynomial τ, we can decide whether $\mathsf{pK}^t(x \mid y) \leq s$ or $\mathsf{pK}^{\tau(|x|,|y|,t)}(x \mid y) > s + \log \tau(|x|, |y|, t)$ with high probability over (x, y) sampled from $\mathcal{D}_{\langle n,m\rangle}$. In this case, we say that "approximating conditional pK^t is easy-on-average over samplable distributions"

Also, we say that "computing pK^t is easy-on-average over samplable distributions" if the following holds.

> For every polynomial-time samplable distribution family $\{\mathcal{D}_n\}_n$ supported over $\{0,1\}^n$, every polynomial q, and for all large enough polynomial τ, there is a probabilistic polynomial-time algorithm A that can decide, given as input $(x, 1^s)$, whether $\mathsf{pK}^{\tau(|x|)}_{2/3}(x) \leq s$ or $\mathsf{pK}^{\tau(|x|)}_{1/3}(x) > s$,[6] with probability at least $1 - 1/q(n)$ over $x \sim \mathcal{D}_n$ and the internal randomness of A.

▶ **Theorem 5** (Informal). *The following are equivalent.*

1. *Infinitely-often one-way functions do not exist.*
2. *Approximating* pK^t *is easy-on-average over samplable distributions.*
3. *Approximating conditional* pK^t *is easy-on-average over samplable distributions.*
4. *Computing* pK^t *is easy-on-average over samplable distributions.*
5. *Computing conditional* pK^t *is easy-on-average over samplable distributions.*

A sketch of the proof of Theorem 5 can be found in [20, Section 3.3].

1.1.2 Characterizing Impagliazzo's Worlds by Tractability of Conditional Time-Unbounded Kolmogorov Complexity

We present a meta-complexity problem, namely approximating conditional Kolmogorov complexity up to an $O(\log n)$ additive term, that is unconditionally hard (even uncomputable) in the worst case, but such that its average-case intractability for different classes of distributions characterize Algorithmica, Heuristica and Pessiland.

Characterizing DistNP $\subseteq$ BPP and DistNP $\subseteq$ HeurBPP by Tractability of Time-Unbounded Kolmogorov Complexity

To begin, we recall a recent result by Ilango, Ren, and Santhanam [11] characterizing the non-existence of one-way functions by the tractability of approximating Kolmogorov complexity over polynomial-time samplable distributions. We consider the following conditional variant from [8].

---

[5] Note that this is the problem Gap-MINpKT mentioned in Footnote 3.
[6] This problem is referred to as $\mathsf{MpK}^\tau\mathsf{P}$ in [17].

▶ **Theorem 6** ([8, Lemma 27], cf. [11]). *The following are equivalent.*

1. *Infinitely-often one-way functions do not exist.*
2. **(Approximating conditional Kolmogorov complexity is easy-on-average over polynomial-time samplable distributions.)**
   For every polynomial-time samplable distribution family $\{\mathcal{D}_n\}_n$, where each $\mathcal{D}_n$ is over $\{0,1\}^n \times \{0,1\}^n$, and every polynomial q, there exist a probabilistic polynomial-time algorithm A and a polynomial p such that for all $n \in \mathbb{N}$,

$$\Pr_{(x,y)\sim\mathcal{D}_n}[\mathsf{K}(x \mid y) \leq A(x,y) \leq \mathsf{K}(x \mid y) + \log p(n)] \geq 1 - \frac{1}{q(n)}.$$

Note that a one-way function is a function that is efficiently computable but hard to invert on average; thus, this notion is based on *average-case* hardness. Theorem 6 characterizes the existence of one-way functions by the average-case hardness of approximating (conditional) Kolmogorov complexity. Then, for $\mathsf{NP} \not\subseteq \mathsf{BPP}$, which is a worst-case hardness notion, one might think that it can be characterized by the worst-case hardness of approximating (conditional) Kolmogorov complexity. However, it is well known that the task of approximating the conditional Kolmogorov complexity is provably intractable in the worst case, so such a characterization would imply $\mathsf{NP} \not\subseteq \mathsf{BPP}$ unconditionally.

Consider a polynomial-time samplable distribution $\mathcal{D}$ over $\{0,1\}^n \times \{0,1\}^n$. Also, let $\mathcal{D}^{(2)}$ be the marginal distribution of $\mathcal{D}$ on the second half, and let $\mathcal{D}(\cdot \mid y)$ denote the conditional distribution of $\mathcal{D}$ on the first half given that the second half is y. Now, observe the following equivalent way of sampling a pair of strings (x,y) from $\mathcal{D}$: We first sample y from $\mathcal{D}^{(2)}$ and then x from $\mathcal{D}(\cdot \mid y)$.

Note that Theorem 6 essentially says that one-way functions do not exist if and only if, for every polynomial-time samplable distribution $\mathcal{D}$, one can approximate $\mathsf{K}(x \mid y)$ on average over (x,y), where we sample y from $\mathcal{D}^{(2)}$ and then x from $\mathcal{D}(\cdot \mid y)$. In order to characterize $\mathsf{NP} \subseteq \mathsf{BPP}$, we consider the tractability of approximating conditional Kolmogorov complexity in the *semi-worst case*, meaning that we can approximate $\mathsf{K}(x \mid y)$ on average over x sampled from $\mathcal{D}(\cdot \mid y)$ for *all* $y \in \{0,1\}^n$ (instead of an average y from $\mathcal{D}^{(2)}$). Our first result is a characterization of $\mathsf{NP} \subseteq \mathsf{BPP}$ by the tractability of approximating conditional Kolmogorov complexity in this semi-worst case. Formally, we show the following.

▶ **Theorem 7.** *The following are equivalent.*

1. $\mathsf{NP} \subseteq \mathsf{BPP}$.
2. **(Approximating conditional Kolmogorov complexity is easy in the semi-worst case.)**
   For every polynomial-time samplable distribution family $\{\mathcal{D}_n\}_n$, where each $\mathcal{D}_n$ is over $\{0,1\}^n \times \{0,1\}^n$, and every polynomial q, there exist a probabilistic polynomial-time algorithm A and a polynomial p such that for all $n \in \mathbb{N}$ and $y \in \{0,1\}^n$,

$$\Pr_{x\sim\mathcal{D}_n(\cdot|y)}[\mathsf{K}(x \mid y) \leq A(x,y) \leq \mathsf{K}(x \mid y) + \log p(n)] \geq 1 - \frac{1}{q(n)}.$$

Theorem 7 shows that $\mathsf{NP} \subseteq \mathsf{BPP}$ if and only if for every polynomial-time samplable distribution $\mathcal{D}$, approximating $\mathsf{K}(x \mid y)$ is easy on average over x sampled from $\mathcal{D}(\cdot \mid y)$ for *every* $y \in \{0,1\}^n$. Now, instead of considering every $y \in \{0,1\}^n$ (a worst-case notion), it is also natural to consider an average y sampled from some polynomial-time samplable distribution $\mathcal{C}$ (an average-case notion). However, the distribution $\mathcal{C}$ here can be independent of $\mathcal{D}$. In particular, it does not necessarily have to be $\mathcal{D}^{(2)}$.

Next, we show that the average-case tractability of approximating conditional Kolmogorov complexity over such *independent polynomial-time samplable distributions*, in fact characterizes the *average-case* easiness of NP (i.e., DistNP $\subseteq$ HeurBPP). We first state formally the definition of independent polynomial-time samplable distributions.

▶ **Definition 8** (Independent Polynomial-Time Samplable [8]). *We say that a distribution family $\{\mathcal{D}_n\}_n$, where each $\mathcal{D}_n$ is over $\{0,1\}^n \times \{0,1\}^n$, is* independent polynomial-time samplable *if there exist two polynomial-time samplable distribution families $\{\mathcal{A}_n\}_n$ and $\{\mathcal{B}_n\}_n$, where each $\mathcal{A}_n$ is over $\{0,1\}^n$ and each $\mathcal{B}_n$ is over $\{0,1\}^n \times \{0,1\}^n$, such that $\mathcal{D}_n$ can be equivalently sampled as follows: sample $y \sim \mathcal{A}_n$, sample $x \sim \mathcal{B}_n(\cdot \mid y)$, and then output (x,y).*

It is easy to see that every polynomial-time samplable distribution is also independent polynomial-time samplable, by letting $\mathcal{A}$ be the marginal distribution of $\mathcal{D}$ on the second half and letting $\mathcal{B}$ be $\mathcal{D}$. However, the converse is not necessarily true. Nevertheless, Theorem 6 and Theorem 9 (which we state below) imply that the task of ruling out Pessiland is equivalent to showing that the hardness of approximating conditional Kolmogorov complexity remains unchanged over these two classes of distributions.

▶ **Theorem 9.** *The following are equivalent.*

1. DistNP $\subseteq$ HeurBPP.
2. **(Approximating conditional Kolmogorov complexity is easy-on-average over independent polynomial-time samplable distributions.)**
   For every independent polynomial-time samplable distribution family $\{\mathcal{D}_n\}_n$ and every polynomial q, there exist a probabilistic polynomial-time algorithm A and a polynomial p such that for all $n \in \mathbb{N}$,

$$\Pr_{(x,y)\sim\mathcal{D}_n}[\mathsf{K}(x \mid y) \le A(x,y) \le \mathsf{K}(x \mid y) + \log p(n)] \ge 1 - \frac{1}{q(n)}.$$

Finally, we extend Theorem 6 to characterize the non-existence of *auxiliary-input one-way functions* by the tractability of approximating conditional Kolmogorov complexity over P/poly-*samplable distributions*.

▶ **Theorem 10.** *The following are equivalent.*

1. *Auxiliary-input one-way functions do not exist.*
2. *For every sequence of strings $\{y_n\}_n$ where each $y_n \in \{0,1\}^n$, every distribution family $\{\mathcal{D}_n\}_n$ samplable in polynomial time using $\{y_n\}_n$ as advice, where each $\mathcal{D}_n$ is over $\{0,1\}^n$, and every polynomial q, there exist a probabilistic polynomial-time algorithm A and a polynomial p such that for all $n \in \mathbb{N}$,*

$$\Pr_{x\sim\mathcal{D}_n}[\mathsf{K}(x \mid y_n) \le A(x,y_n) \le \mathsf{K}(x \mid y_n) + \log p(n)] \ge 1 - \frac{1}{q(n)}.$$

The results above characterize the non-existence of one-way functions, DistNP $\subseteq$ HeurBPP, and NP $\subseteq$ BPP by the *distributional* tractability of approximating the conditional Kolmogorov complexity. They imply that the tasks of ruling out Impagliazzo's certain worlds are equivalent to showing that the hardness of this problem is the same with respect to different classes of distributions. For example, Theorem 6 and Theorem 9 imply that basing one-way functions on DistNP $\not\subseteq$ HeurBPP (a.k.a., ruling out Pessiland) is equivalent to showing that the hardness of approximating conditional Kolmogorov complexity over polynomial-time samplable distributions is the same as the hardness over independent polynomial-time samplable distributions.

Equivalences between Tractability of Time-Unbounded and Time-Bounded Kolmogorov Complexity

We first recall the definition of time-bounded Kolmogorov complexity. For $x, y \in \{0,1\}^*$ and $t \in \mathbb{N}$, we define $\mathsf{K}^t(x \mid y)$ to be the minimum length of a program $p \in \{0,1\}^*$ such that $U^y(p)$ outputs x within t steps. Here, U is a fixed time-optimal universal Turing machine and has oracle access to the string y.

For $\tau\colon \mathbb{N} \to \mathbb{N}$ and $\kappa\colon \mathbb{N} \to \mathbb{N}$, let $\mathsf{McK}^\tau\mathsf{P}[\kappa]$ be the problem where we are given input $x \in \{0,1\}^n$ and $y \in \{0,1\}^{\kappa(n)}$, and we are asked to compute $\mathsf{K}^{\tau(|x|)}(x \mid y)$. Given a polynomial τ and a polynomial κ, we say that:

- $\mathsf{McK}^\tau\mathsf{P}[\kappa]$ is *easy in the worst case* if $\mathsf{McK}^\tau\mathsf{P}[\kappa]$ can be solved in polynomial time.
- $\mathsf{McK}^\tau\mathsf{P}[\kappa]$ is *easy-on-average over polynomial-time samplable distributions* if $\mathsf{McK}^\tau\mathsf{P}[\kappa]$ admits a heuristic scheme. That is for any polynomial-time samplable distribution $\mathcal{D} = \{D_n\}_n$, where each $\mathcal{D}_n$ samples (x, z) with $x \in \{0,1\}^n$ and $y \in \{0,1\}^{\kappa(n)}$, there exists a probabilistic polynomial-time algorithm A such that for all $n, k \in \mathbb{N}$,

$$\Pr_{x,y\sim\mathcal{D}_n}\left[A(x, y; 1^n, 1^k) = \mathsf{K}^{\tau(|x|)}(x \mid y)\right] \geq 1 - \frac{1}{k}.$$

- $\mathsf{McK}^\tau\mathsf{P}[\kappa]$ is *easy-on-average over the uniform distribution* if for every polynomial p, there exists a probabilistic polynomial-time algorithm A such that for all $n \in \mathbb{N}$,

$$\Pr_{x\sim\{0,1\}^n, y\sim\{0,1\}^{\kappa(n)}}\left[A(x, y) = \mathsf{K}^{\tau(|x|)}(x \mid y)\right] \geq 1 - \frac{1}{p(n)}.$$

▶ **Theorem 11** (Implicit in [16]). *The following hold.*

- *For all polynomial $\tau(n) \geq n^2$, there exists a polynomial κ such that $\mathsf{NP} \subseteq \mathsf{BPP}$ if and only if $\mathsf{McK}^\tau\mathsf{P}[\kappa]$ is easy in the worst-case.*
- *For all polynomial $\tau(n) \geq n^2$, there exists a polynomial κ such that $\mathsf{DistNP} \subseteq \mathsf{HeurBPP}$ if and only if $\mathsf{McK}^\tau\mathsf{P}[\kappa]$ is easy-on-average over polynomial-time samplable distributions.*
- *For every polynomial $\tau(n) \geq 1.1n$ and polynomial κ, infinitely-often one-way functions do not exist if and only if $\mathsf{McK}^\tau\mathsf{P}[\kappa]$ is easy-on-average over the uniform distribution.*

As a corollary, we get the following equivalences between the tractability of conditional Kolmogorov complexity and that of conditional time-bounded Kolmogorov complexity.

▶ **Corollary 12** (Informal). *The following hold.*

- *For all polynomial $\tau(n) \geq n^2$, there exists a polynomial κ such that approximating conditional Kolmogorov complexity is easy in the semi-worst case if and only if $\mathsf{McK}^\tau\mathsf{P}[\kappa]$ is easy in the worst-case case.*
- *For all polynomial $\tau(n) \geq n^2$, there exists a polynomial κ such that approximating conditional Kolmogorov complexity is easy-on-average over independent polynomial-time samplable distributions if and only if $\mathsf{McK}^\tau\mathsf{P}[\kappa]$ is easy-on-average over polynomial-time samplable distributions.*
- *For every polynomial $\tau(n) \geq 1.1n$ and polynomial κ, approximating conditional Kolmogorov complexity is easy-on-average over polynomial-time samplable distributions if and only if $\mathsf{McK}^\tau\mathsf{P}[\kappa]$ is easy-on-average over the uniform distribution.*

Proof. This follows directly from Theorem 6, Theorem 7, Theorem 9, and Theorem 11. ◀

1.2 Techniques

In this section, we explain the main ideas behind our proofs.

Characterizing Non-Existence of One-Way Functions by Average-Case Easiness of Conditional pK^t

A recent result by Liu and Pass [17] characterized the non-existence of (infinitely-often) one-way functions by the average-case easiness of computing pK^t over polynomial-time samplable distributions. Here, we describe a proof of this result that is slightly different than the original one and show how to generalize it to *conditional* pK^t.

It will be convenient to think of the pK^t complexity of a string as its K^t complexity *conditioning on a random string* r (see [20, Proposition 17]).

First of all, by employing ideas from [14, 17], one can construct a function, which outputs the string x produced by a randomly selected (time-bounded) program (resp. conditioning on a random string r), and show that if this function can be inverted, then we can obtain a shortest program for x (resp. conditioning on r) "on average". In particular, it can be shown that if infinitely-often one-way functions do not exist, then for every time bound function $\tau(n) = n^{O(1)}$, there exists an efficient algorithm A (for simplicity, think of it as being deterministic) such that with high probability over a uniformly random string r, $A(x;r)$ computes $\mathsf{K}^\tau(x \mid r)$ for an average x sampled from some distribution $\mathcal{E}^\tau_r$, defined as $\mathcal{E}^\tau_r(x) := 2^{-\mathsf{K}^\tau(x|r)}$.

Next, we want to say that, for almost all r, the algorithm $A(-;r)$, which works for the distribution $\mathcal{E}^\tau_r$, also works for a given polynomial-time samplable distribution $\mathcal{D}$ (provided that τ is a sufficiently large polynomial). To get this, it suffices to show that $\mathcal{E}^\tau_r$ *dominates*[7] $\mathcal{D}$, i.e., $2^{-\mathsf{K}^\tau(x|r)} \gtrsim \mathcal{D}(x)$ for *every* x. The observation here is that this follows from the recently discovered coding theorem for $\mathsf{pK}^{\mathsf{poly}}$ [19], which asserts that for *every* string x, $\mathsf{pK}^\tau(x) \lesssim \log(1/\mathcal{D}(x))$ (again, provided that τ is a sufficiently large polynomial). To see this, note that by the definition of pK^t, we have for a uniform random r, $\mathsf{K}^\tau(x \mid r) \leq \mathsf{pK}^\tau(x)$.

Given the above, we have that with high probability over a uniformly random r, $A(x;r) = \mathsf{K}^\tau(x \mid r)$ for an average x sampled from $\mathcal{D}$. By an averaging argument, we get that with high probability over $x \sim \mathcal{D}$, $A(x;r) = \mathsf{K}^\tau(x \mid r)$ with high probability over a uniformly random r. For any such *good* x, if $\mathsf{pK}^\tau_{2/3}(x) \leq s$ (resp. $\mathsf{pK}^\tau_{1/3}(x) > s$), which means $\mathbf{Pr}_r[\mathsf{K}^\tau(x \mid r) \leq s] \geq 2/3$ (resp. $\mathbf{Pr}_r[\mathsf{K}^\tau(x \mid r) > s] \geq 2/3$), then $A(x,r) \leq s$ (resp. $A(x,r) > s$) with high probability over r. This allows us to solve the problem of computing pK^τ on average over the distribution $\mathcal{D}$.

Now we describe how to generalize the above to *conditional* pK^t.

Suppose we want to compute $\mathsf{pK}^\tau(x \mid y)$ over (x,y) sampled from some polynomial-time distribution $\mathcal{D}$. It will be convenient to consider the following equivalent way of sampling $\mathcal{D}$: We first sample $y \sim \mathcal{D}^{(2)}$, where $\mathcal{D}^{(2)}$ is the marginal distribution of $\mathcal{D}$ on the second half, and then sample $x \sim \mathcal{D}(\cdot \mid y)$, where $\mathcal{D}(\cdot \mid y)$ is the conditional distribution of $\mathcal{D}_n$ on the first half given that the second half is y. Finally, we output (x,y).

First of all, by modifying the construction of the candidate one-way function described above (e.g., by incorporating the distribution $\mathcal{D}^{(2)}$ into the construction), we can show that if infinitely-often one-way functions do not exist, then there exists an efficient algorithm A such that with high probability over a uniformly random string r and over y sampled from $\mathcal{D}^{(2)}$, $A(x;y,r)$ computes $\mathsf{K}^\tau(x \mid y,r)$ for an average x sampled from some distribution $\mathcal{E}^\tau_{y,r}$, where $\mathcal{E}^\tau_{y,r}(x) := 2^{-\mathsf{K}^\tau(x|y,r)}$.

[7] Recall that a distribution $\mathcal{D}$ dominates another distribution $\mathcal{D}'$ if $\mathcal{D}(x) \geq \mathcal{D}'(x)/\mathsf{poly}(n)$ for every x.

Now similar to the previous case, we want to say that, with high probability over r and $y \sim \mathcal{D}^{(2)}$, the algorithm $A(-; y, r)$, which works for the distribution $\mathcal{E}^{\tau}_{y,r}$, also works for the distribution $\mathcal{D}(\cdot \mid y)$. Again, it suffices to show that $\mathcal{E}^{\tau}_{y,r}(x) = 2^{-\mathsf{K}^{\tau}(x|y,r)} \gtrsim \mathcal{D}(x \mid y)$ for every x. However, this would require a conditional version of the coding theorem for $\mathsf{pK}^{\mathsf{poly}}$ applying to the distribution $\mathcal{D}(\cdot \mid y)$ (which is not necessarily efficiently samplable given y). Such a coding theorem is not known (in fact, is unlikely to hold).

The key observation is that in order to show that the algorithm $A(-; y, r)$, which works on average over the distribution $\mathcal{E}^{\tau}_{y,r}$, also works for $\mathcal{D}(\cdot \mid y)$, it suffices to have that $\mathcal{E}^{\tau}_{y,r}(x)$ dominates $\mathcal{D}(x \mid y)$ *on almost all* x, instead of every x. Then this weaker condition can be obtained from an *average-case* coding theorem for $\mathsf{pK}^{\mathsf{poly}}$, which has been shown under the assumption that infinitely-often one-way functions do not exist [8] (see [20, Theorem 29]).

More specifically, [8] showed that if infinitely-often one-way functions do not exist, then with high probability over $y \sim \mathcal{D}^{(2)}$ and $x \sim \mathcal{D}(\cdot \mid y)$, it holds that

$$\mathsf{pK}^{\tau}(x \mid y) \lesssim \log \frac{1}{\mathcal{D}(x \mid y)}.$$

Again, by the definition of pK^t and an averaging argument, this yields that with high probability over a uniformly random r and $y \sim \mathcal{D}^{(2)}$,

$$\mathsf{K}^{\tau}(x \mid y, r) \leq \log \frac{1}{\mathcal{D}(x \mid y)}$$

holds for almost all $x \sim \mathcal{D}(\cdot \mid y)$. This allows us to say that with high probability over r and $y \sim \mathcal{D}^{(2)}$, the distribution $\mathcal{E}^{\tau}_{y,r}$ dominates $\mathcal{D}(\cdot \mid y)$ *on average*, so the algorithm $A(-; y, r)$, which works for $\mathcal{E}^{\tau}_{y,r}$, also works for $\mathcal{D}(\cdot \mid y)$.

At this point, we get that with high probability over $(x, y) \sim \mathcal{D}$ and over a uniformly random r, $A(x; y, r) = \mathsf{K}^{\tau}(x \mid y, r)$. By the same argument as described above, this allows us to compute $\mathsf{pK}^{\tau}(x \mid y)$ on average over (x, y) sampled from $\mathcal{D}$.

The converse direction, i.e., that computing conditional pK^t on average allows us to break one-way functions, follows from the standard observation that computing pK^t on average over samplable distributions allows us to distinguish pseudo-random distributions (which are supported on strings of low pK^t complexity) from random strings (which have high pK^t complexity).

Characterizing DistNP $\subseteq$ HeurBPP by Average-Case Easiness of Conditional pK^t in Sublinear-Time Regime

To show that the average-case easiness of computing conditional pK^t (in the sublinear-time regime) implies the average-case easiness of NP (both with respect to polynomial-time samplable distributions), we first show that it is NP-hard to compute conditional pK^t (again, in the sublinear-time regime). Recently, Liu and Pass [16] and Hirahara [6] showed that the problem of computing the conditional K^t in the sublinear-time regime is NP-hard. We generalize this result to pK^t.

At a high level, our proof follows a similar approach but also requires some crucial observations to address the more complex notion of pK^t and to make it applicable to show Theorem 2. In particular, we adapt the proof in [6] which relies on the use of a *secret sharing scheme* (see [6, Section 2.3] for an exposition). More specifically, it reduces the problem of approximating the hamming weight of a *minimum satisfying assignment* of a given monotone formula, which is known to be NP-hard, to that of computing conditional K^t in the sublinear-time regime. That is, for every constant $c > 1$ and time bound function

$\tau(n,m) := n^c \cdot m^{1-1/c}$, there is a randomized reduction R such that if a given monotone formula ψ has a satisfying assignment of hamming weight at most ζ (resp. much larger than ζ), then with high probability, R produces a pair of strings (x,y) and ρ such that $\mathsf{K}^{\tau(|x|,|y|)}(x \mid y) \leq \rho$ (resp. $\mathsf{K}^{\tau(|x|,|y|)}(x \mid y) > \rho$).

Our key observation is that this reduction still works in the presence of any fixed string r. Roughly put, the reason for this is that a secret sharing scheme remains secure even if an adversary has access to some fixed string. More specifically, we can show that with respect to any string r, if a given monotone formula ψ has a satisfying assignment of hamming weight much larger than ζ, then with high probability the algorithm R produces a pair of strings (x,y) and ρ such that $\mathsf{K}^{\tau(|x|,|y|)}(x \mid y,r) > \rho$. This allows us to say that if the minimum weight of ψ is much larger than ζ, then with high probability over a random string r and over the internal randomness of R, $\mathsf{K}^{\tau(|x|,|y|)}(x \mid y,r) > \rho$. By an averaging argument, this gives that with high probability over the internal randomness of R, $\mathsf{K}^{\tau(|x|,|y|)}(x \mid y,r) > \rho$ for more than $2/3$ of the r's, which essentially means $\mathsf{pK}^{\tau(|x|,|y|)}_{1/3}(x \mid y) > \rho$.

Now we have showed that computing conditional pK^t (in the sublinear-time regime) is NP-hard. To solve an NP problem L over a given polynomial-time samplable distribution $\mathcal{D}$, we can compose $\mathcal{D}$ with the reduction R to obtain a new distribution $\mathcal{D}'$. Then we can show that computing conditional pK^t on average over $\mathcal{D}'$ will allow us to solve L on average over $\mathcal{D}$. However, there is an additional subtle issue here, the original reduction R depends on the time bound function (i.e., for every sublinear time bound τ, there is a reduction R that will work). On the other hand, to show Theorem 2 (Item 2 $\implies$ Item 1), it is required that the reduction works for all time bound functions τ of the form $\tau(n,m) = n^c \cdot m^{1-1/c}$. We will then need to further modify the reduction to achieve this. (See [20, Lemma 45] for the details.)

Now we need to show the other direction saying that the average-case easiness of NP implies the average-case easiness of computing conditional pK^t. Unlike the problem of computing (conditional) K^t, computing (conditional) pK^t is not known to be in NP, so we can not get the desired implication directly. However, it is not hard to see that the problem of computing conditional pK^t is in fact in (promise) AM.[8] If we can solve NP, then we can also solve AM (in the randomized setting), by a standard trick that combines the instance of an AM problem with a random string to produce an instance for an NP problem. (See [20, Lemma 53] for the details.)

Characterizing DistNP $\subseteq$ BPP and DistNP $\subseteq$ HeurBPP by Tractability of Time-Unbounded Kolmogorov Complexity.

First, we recap the proof of Theorem 6 as presented in [11]. We will ignore the issue of "infinitely often" in this subsection.

To show that the non-existence of one-way functions implies efficient algorithms for approximating conditional Kolmogorov complexity on average over polynomial-time samplable distributions, we use a powerful result from [13], which asserts that if one-way functions do not exist, then for any polynomial-time samplable distribution $\mathcal{D}$ over $\{0,1\}^n \times \{0,1\}^n$, one can efficiently estimate $\mathcal{D}(x \mid y)$ on average over $(x,y) \sim \mathcal{D}$. In addition, we combine two fundamental properties related to time-unbounded Kolmogorov complexity: The first is called the coding theorem, which roughly says that for every $(x,y) \in \mathsf{Support}(\mathcal{D})$,

[8] Here, we refer to the problem $\mathsf{Cond\text{-}pK}$ instead of the one that asks to decide whether $\mathsf{pK}^{\tau(|x|,|y|)}(x \mid y) \leq s$ for a given input $(x,y,1^s)$ and time bound τ.

$$\mathsf{K}(x \mid y) \lesssim \log \frac{1}{\mathcal{D}(x \mid y)},$$

and the second is the incompressibility property, which states that all $y \in \{0,1\}^n$ and for almost all $x \sim \mathcal{D}(\cdot \mid y)$,

$$\mathsf{K}(x \mid y) \gtrsim \log \frac{1}{\mathcal{D}(x \mid y)}.$$

It follows that for almost all $(x, y) \sim \mathcal{D}$,

$$\mathsf{K}(x \mid y) \approx \log \frac{1}{\mathcal{D}(x \mid y)}.$$

This allows us to approximate $\mathsf{K}(x \mid y)$ by estimating $\mathcal{D}(x \mid y)$, and the latter can be done efficiently if one-way functions do not exist.

For the other direction, the idea is that an efficient algorithm for approximating Kolmogorov complexity on average can be used to construct a function that distinguishes the output distribution of a cryptographic pseudorandom generator from the uniform distribution. Intuitively, this is because the outputs of such a generator have low $\mathsf{K}^{\mathsf{poly}}$ complexity while a random string has high Kolmogorov complexity. Then such an algorithm implies the non-existence of pseudorandom generators and hence of one-way functions [4].

Now, let us try to see if we can adapt the above proof paradigm to show Theorem 9, which characterizes $\mathsf{DistNP} \subseteq \mathsf{HeurBPP}$ by the tractability of approximating conditional Kolmogorov complexity on average over *independent polynomial-time samplable distributions*.

One direction is in fact easy by using tools developed in [8]. In particular, it is observed in [8] that if $\mathsf{DistNP} \subseteq \mathsf{HeurBPP}$, then every independent polynomial-time samplable distribution can be simulated by some polynomial-time samplable distribution (see [20, Lemma 26]). Consequently, if $\mathsf{DistNP} \subseteq \mathsf{HeurBPP}$ (which also implies that one-way functions do not exist), then we can reduce the task of approximating conditional Kolmogorov complexity over independent polynomial-time samplable distributions to that of approximating conditional Kolmogorov complexity over polynomial-time samplable distributions, which is tractable if one-way functions do not exist.

However, for the other direction, it is unclear how we can get $\mathsf{DistNP} \subseteq \mathsf{HeurBPP}$ from the tractability of approximating conditional Kolmogorov complexity over independent polynomial-time samplable distributions, by using ideas from the proof of the characterization for one-way functions. In that scenario, we use the algorithm for approximating conditional Kolmogorov complexity as a distinguisher to break the security of a cryptographic pseudorandom generator.

Here, we will use a different approach. Specifically, we rely on a recently discovered characterization of $\mathsf{DistNP} \subseteq \mathsf{HeurBPP}$ by the validity of a certain property called *conditional coding* for pK^t. More precisely, the authors of [8] showed that $\mathsf{DistNP} \subseteq \mathsf{HeurBPP}$ if and only if conditional coding property for $\mathsf{pK}^{\mathsf{poly}}$ holds on average over pairs of strings drawn from independent polynomial-time samplable distributions, i.e., for any independent polynomial-time samplable distribution $\mathcal{D}$ over $\{0,1\}^n \times \{0,1\}^n$ and for almost all $(x, y) \sim \mathcal{D}$,

$$\mathsf{pK}^{\mathsf{poly}(n)}(x \mid y) \lesssim \log \frac{1}{\mathcal{D}(x \mid y)}$$

(see [20, Theorem 30]).

Now given this characterization of $\mathsf{DistNP} \subseteq \mathsf{HeurBPP}$ using conditional coding, it suffices to show that conditional coding property for $\mathsf{pK}^{\mathsf{poly}}$ over independent polynomial-time samplable distributions follows from the tractability of approximating conditional Kolmogorov complexity over the same class of distributions.

How can we show this? First of all, note that by the coding theorem for *time-unbounded* Kolmogorov complexity, we have that for every $(x, y) \in \mathsf{Support}(\mathcal{D})$,

$$\mathsf{K}(x \mid y) \lesssim \log \frac{1}{\mathcal{D}(x \mid y)}.$$

Then to get the desired conditional coding property for $\mathsf{pK}^{\mathsf{poly}}$, it suffices to show that for almost all $(x, y) \sim \mathcal{D}$,

$$\mathsf{pK}^{\mathsf{poly}(n)}(x \mid y) \leq \mathsf{K}(x \mid y) + O(\log n). \tag{1}$$

Now, let us describe how to show the above, assuming efficient algorithms for approximating conditional Kolmogorov complexity over independent polynomial-time samplable distributions.

The key ingredient here is a pseudorandom generator construction with reconstruction property. Such a generator is instantiated with a target string, it then takes as input a random seed and outputs a string that is longer than the seed. The reconstruction property allows us to say that if there exists a function that can distinguish the output distribution of the generator from the uniform distribution, then it can be used to recover the target string, using an additional advice string. This enables us to say that given a distinguisher, the target string has poly-time-bounded Kolmogorov complexity bounded by the length of the advice string. An algorithm for approximating Kolmogorov complexity can naturally be used as such a distinguisher, since the outputs of the generator have low Kolmogorov complexity while a random string has high Kolmogorov complexity. By appropriately configuring the parameters of the generator, we can ensure that the length of the advice string is comparable to the Kolmogorov complexity of the target string. This allows us to upper bound the poly-time-bounded Kolmogorov complexity of the target string by its Kolmogorov complexity.

Using this approach, the authors of [8] showed that if efficient algorithms exist for approximating conditional Kolmogorov complexity over polynomial-time samplable distributions, then for every polynomial-time samplable distribution $\mathcal{D}$ over $\{0,1\}^n \times \{0,1\}^n$ and almost all $(x, y) \sim \mathcal{D}$,

$$\mathsf{rK}^{\mathsf{poly}(n)}(x \mid y) \leq \mathsf{K}(x \mid y) + O(\log^3 n). \tag{2}$$

Here, rK^t is a certain randomized variant of time-bounded Kolmogorov complexity measure [2, 18].

The $O(\log^3 n)$ additive term in Equation (2) results from the use of a specific pseudorandom generator construction with an rK^t-style reconstruction property (as they need to upper bound $\mathsf{rK}^{\mathsf{poly}}$ by K), and such a generator has sub-optimal "advice complexity" in its reconstruction. In our case, we need to upper bound $\mathsf{pK}^{\mathsf{poly}}$ by K, and we can use a different pseudorandom generator construction with a pK^t-style reconstruction property that is known to have optimal "advice complexity" (see [20, Section 2.7]). This results in only an $O(\log n)$ additive term instead of $O(\log^3 n)$ as in the previous case.

The description provided above does not address an important technical distinction between showing Equation (1) and showing Equation (2) in [8]. In our case, we need to show Equation (1) over *independent polynomial-time samplable distributions*, whereas the

other case involves the simpler class of *polynomial-time samplable samplable distributions*. In fact, in the proof of Equation (2), a crucial fact used is that the uniform mixture of two polynomial-time samplable distributions is also polynomial-time samplable. Intuitively, the reason why this is needed is that we need to obtain a function that can distinguish the output distribution of a pseudorandom generator (induced by a polynomial-time samplable distribution) and the uniform distribution (also combined with a polynomial-time samplable distribution), so we need to apply an algorithm to approximate Kolmogorov complexity over the mixture uniform of those two distributions.

However, in our case, we are dealing with independent polynomial-time samplable distributions, and the uniform mixture of two independent polynomial-time samplable distributions is not necessarily independent polynomial-time samplable. The key insight here is that we don't really need to be concerned with the uniform mixture of two *generic* independently polynomial-time samplable distributions. Instead, the two distributions have the property that they are identical when restricted to the second half. We then show that the uniform mixture of such two distributions remains independently polynomial-time samplable. (See the proofs of [20, Lemma 59] and [20, Lemma 63] for details.)

We now describe the proof of Theorem 7. Again, the direction of showing the tractability of approximating conditional Kolmogorov complexity in the semi-worst case from $\mathsf{NP} \subseteq \mathsf{BPP}$ can be done in a way similar to that of Theorem 6 (as described earlier in this subsection). This is because if $\mathsf{NP} \subseteq \mathsf{BPP}$, then one can estimate $\mathcal{D}(x \mid y)$ for every polynomial-time samplable distribution $\mathcal{D}$ and $(x, y) \in \mathsf{Support}(\mathcal{D})$, a result due to [24] (see also [20, Lemma 27]).

For the other direction, we will employ the same approach as used to to show Theorem 9. In this case, we will use a similar characterization of $\mathsf{NP} \subseteq \mathsf{BPP}$ through conditional coding. Specifically, it has been shown in [8] that $\mathsf{NP} \subseteq \mathsf{BPP}$ if and only if *worst-case* conditional coding for $\mathsf{pK}^{\mathsf{poly}}$ holds, i.e., for every polynomial-time samplable distribution $\mathcal{D}$ over $\{0,1\}^n \times \{0,1\}^n$ and *every* $(x, y) \in \mathsf{Support}(D)$,

$$\mathsf{pK}^{\mathsf{poly}(n)}(x \mid y) \lesssim \log \frac{1}{\mathcal{D}(x \mid y)}. \tag{3}$$

Unfortunately, it is unclear how we can obtain the above worst-case conditional coding property from the tractability of approximating conditional Kolmogorov complexity in the *semi-worst case* by following the same approach. To overcome this, we observe that we can modify the original proof in [8] to obtain a characterization of $\mathsf{NP} \subseteq \mathsf{BPP}$ by *semi-worst-case* conditional coding, which only requires Equation (3) to hold for almost all $x \sim \mathcal{D}(\cdot \mid y)$ and for all $y \in \{0,1\}^n$ (see [20, Lemma 64]).

By using this alternative characterization and addressing a similar issue that arises when transitioning from polynomial-time samplable distributions to semi-worst-case input distributions, as described above in the case of showing Theorem 9, we can now use efficient algorithms for approximating conditional Kolmogorov complexity in the semi-worst case to obtain the desired semi-worst-case conditional coding property, which then yields $\mathsf{NP} \subseteq \mathsf{BPP}$.

1.3 Open Problems

Can we show NP-hardness of computing conditional pK^t in the polynomial-time regime? By Corollary 4, this would imply that Pessiland does not exist. Are there any barriers to showing such an NP-hardness result?

Theorem 9 characterizes the *error-prone* average-case easiness of NP (i.e., $\mathsf{DistNP} \subseteq \mathsf{HeurBPP}$) by the tractability of approximating conditional Kolmogorov complexity over independent polynomial-time samplable distributions. Can we obtain a similar characterization for the *errorless* average-case easiness of NP (i.e., $\mathsf{DistNP} \subseteq \mathsf{AvgBPP}$)?

References

1 Eric Allender, Mahdi Cheraghchi, Dimitrios Myrisiotis, Harsha Tirumala, and Ilya Volkovich. One-way functions and a conditional variant of MKTP. In *Conference on Foundations of Software Technology and Theoretical Computer Science* (FSTTCS), pages 7:1–7:19, 2021. doi:10.4230/LIPIcs.FSTTCS.2021.7.

2 Harry Buhrman, Troy Lee, and Dieter van Melkebeek. Language compression and pseudorandom generators. *Comput. Complex.*, 14(3):228–255, 2005. doi:10.1007/s00037-005-0199-5.

3 Halley Goldberg, Valentine Kabanets, Zhenjian Lu, and Igor C. Oliveira. Probabilistic Kolmogorov complexity with applications to average-case complexity. In *Computational Complexity Conference* (CCC), pages 16:1–16:60, 2022. doi:10.4230/LIPIcs.CCC.2022.16.

4 Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999. doi:10.1137/S0097539793244708.

5 Shuichi Hirahara. Characterizing average-case complexity of PH by worst-case meta-complexity. In *Symposium on Foundations of Computer Science* (FOCS), pages 50–60, 2020. doi:10.1109/FOCS46700.2020.00014.

6 Shuichi Hirahara. Symmetry of information from meta-complexity. In *Computational Complexity Conference* (CCC), pages 26:1–26:41, 2022. doi:10.4230/LIPIcs.CCC.2022.26.

7 Shuichi Hirahara. Capturing one-way functions via NP-hardness of meta-complexity. In *Symposium on Theory of Computing* (STOC), pages 1027–1038, 2023. doi:10.1145/3564246.3585130.

8 Shuichi Hirahara, Rahul Ilango, Zhenjian Lu, Mikito Nanashima, and Igor C. Oliveira. A duality between one-way functions and average-case symmetry of information. In *Symposium on Theory of Computing* (STOC), pages 1039–1050, 2023. doi:10.1145/3564246.3585138.

9 Shuichi Hirahara and Rahul Santhanam. On the average-case complexity of MCSP and its variants. In *Computational Complexity Conference* (CCC), pages 7:1–7:20, 2017. doi:10.4230/LIPIcs.CCC.2017.7.

10 Shuichi Hirahara and Rahul Santhanam. Excluding PH pessiland. In *Innovations in Theoretical Computer Science Conference* (ITCS), pages 85:1–85:25, 2022. doi:10.4230/LIPIcs.ITCS.2022.85.

11 Rahul Ilango, Hanlin Ren, and Rahul Santhanam. Robustness of average-case meta-complexity via pseudorandomness. In *Symposium on Theory of Computing* (STOC), pages 1575–1583, 2022. doi:10.1145/3519935.3520051.

12 Russell Impagliazzo. A personal view of average-case complexity. In *Proceedings of the Tenth Annual Structure in Complexity Theory Conference*, pages 134–147, 1995. doi:10.1109/SCT.1995.514853.

13 Russell Impagliazzo and Leonid A. Levin. No better ways to generate hard NP instances than picking uniformly at random. In *Symposium on Theory of Computing* (STOC), pages 812–821, 1990. doi:10.1109/FSCS.1990.89604.

14 Yanyi Liu and Rafael Pass. On one-way functions and Kolmogorov complexity. In *Symposium on Foundations of Computer Science* (FOCS), pages 1243–1254, 2020. doi:10.1109/FOCS46700.2020.00118.

15 Yanyi Liu and Rafael Pass. On the possibility of basing cryptography on EXP≠BPP. In *International Cryptology Conference* (CRYPTO), pages 11–40, 2021. doi:10.1007/978-3-030-84242-0_2.

16 Yanyi Liu and Rafael Pass. On one-way functions from NP-complete problems. In *Conference on Computational Complexity* (CCC), pages 36:1–36:24, 2022. doi:10.4230/LIPIcs.CCC.2022.36.

17 Yanyi Liu and Rafael Pass. One-way functions and the hardness of (probabilistic) time-bounded Kolmogorov complexity w.r.t. samplable distributions. In *Annual Cryptology Conference* (CRYPTO), pages 645–673, 2023. doi:10.1007/978-3-031-38545-2_21.

18 Zhenjian Lu, Igor C. Oliveira, and Rahul Santhanam. Pseudodeterministic algorithms and the structure of probabilistic time. In *Symposium on Theory of Computing* (STOC), pages 303–316, 2021. doi:10.1145/3406325.3451085.

19 Zhenjian Lu, Igor C. Oliveira, and Marius Zimand. Optimal coding theorems in time-bounded Kolmogorov complexity. In *International Colloquium on Automata, Languages, and Programming* (ICALP), pages 92:1–92:14, 2022. doi:10.4230/LIPICS.ICALP.2022.92.

20 Zhenjian Lu and Rahul Santhanam. Impagliazzo's worlds through the lens of conditional Kolmogorov complexity. *Electronic Colloquium on Computational Complexity* (ECCC), TR24-085, 2024. URL: https://eccc.weizmann.ac.il/report/2024/085.

21 Hanlin Ren and Rahul Santhanam. Hardness of KT characterizes parallel cryptography. In *Computational Complexity Conference* (CCC), pages 35:1–35:58, 2021. doi:10.4230/LIPIcs.CCC.2021.35.

22 Michael E. Saks and Rahul Santhanam. On randomized reductions to the random strings. In *Computational Complexity Conference* (CCC), pages 29:1–29:30, 2022. doi:10.4230/LIPIcs.CCC.2022.29.

23 Rahul Santhanam. Pseudorandomness and the minimum circuit size problem. In *Innovations in Theoretical Computer Science Conference* (ITCS), pages 68:1–68:26, 2020. doi:10.4230/LIPIcs.ITCS.2020.68.

24 Larry J. Stockmeyer. On approximation algorithms for #P. *SIAM J. Comput.*, 14(4):849–861, 1985. doi:10.1137/0214060.

Approximation Algorithms for ℓ_p-Shortest Path and ℓ_p-Group Steiner Tree

Yury Makarychev
Toyota Technological Institute at Chicago, IL, USA

Max Ovsiankin
Toyota Technological Institute at Chicago, IL, USA

Erasmo Tani
University of Chicago, IL, USA

Abstract

We present polylogarithmic approximation algorithms for variants of the Shortest Path, Group Steiner Tree, and Group ATSP problems with vector costs. In these problems, each edge e has a vector cost $c_e \in \mathbb{R}^\ell_{\geq 0}$. For a feasible solution – a path, subtree, or tour (respectively) – we find the total vector cost of all the edges in the solution and then compute the ℓ_p-norm of the obtained cost vector (we assume that $p \geq 1$ is an integer). Our algorithms for series-parallel graphs run in polynomial time and those for arbitrary graphs run in quasi-polynomial time.

To obtain our results, we introduce and use new flow-based Sum-of-Squares relaxations. We also obtain a number of hardness results.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Shortest paths; Theory of computation → Approximation algorithms analysis; Theory of computation → Routing and network design problems

Keywords and phrases Shortest Path, Asymmetric Group Steiner Tree, Sum-of-Squares

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.111

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `http://arxiv.org/abs/2404.17669` [24]

Funding *Yury Makarychev*: Partially supported by NSF Awards CCF-1955173, CCF-1934843, and ECCS-2216899.
Max Ovsiankin: Partially supported by the Institute for Data, Econometrics, Algorithms, and Learning (IDEAL) with NSF Grant ECCS-2216899.
Erasmo Tani: Partially supported by the Institute for Data, Econometrics, Algorithms, and Learning (IDEAL) with NSF Grant ECCS-2216912.

Acknowledgements We would like to thank the anonymous reviewers for providing references to relevant prior work and for comments that improved the presentation of this paper.

1 Introduction

In this work, we study robust versions of network design problems. In the ℓ_p-Shortest Path problem, we are given $p \geq 1$, a graph $G = (V, E)$ with vector-valued edge costs $c_e \in \mathbb{R}^\ell_{\geq 0}$, and two vertices s and t; the goal is to find a path P from s to t in G that minimizes the following cost:

$$\mathrm{cost}_{\ell_p}(P) = \left\| \sum_{e \in P} c_e \right\|_p .$$

This problem is a natural generalization of the classical shortest path problem, but surprisingly has not received much attention till recently. The problem has been studied for $p = \infty$ under the name Robust Shortest Path. Aissi, Bazgan and Vanderpooten [1] used dynamic programming to obtain a fully polynomial-time approximation scheme for the case when the number of coordinates ℓ is a constant and $p = \infty$ (this result generalizes to other p). Kasperski and Zieliński [18] proved that ℓ_∞-Shortest Path is hard to approximate within $\log^{1-\varepsilon} \ell$ for all $\varepsilon > 0$ unless NP $\subseteq$ DTIME($n^{\text{polylog}\, n}$). More recently, the same authors [21] gave an $O(\sqrt{n \log \ell / \log\log \ell})$-approximation to the problem by rounding a flow-based linear programming relaxation and proved that their LP has the integrality gap of $\Omega(\sqrt{n})$. In a recent breakthrough, Li, Xu, and Zhang [23] gave an $O(\log n \log \ell)$-approximation algorithm for ℓ_∞-Shortest Path with running time quasi-polynomial in the size of the input instance. In particular, their algorithm is the first polylogarithmic approximation known to date. They also show that the same approximation guarantees can be obtained in polynomial time for graphs of bounded treewidth, and they give a polynomial time $O(d \log \ell)$ approximation algorithm for series-parallel graphs, where d is the depth (order) of the series-parallel decomposition of the input graph. No results were known for any other exponents $p \in (1, \infty)$. However, it is trivial to get an $\ell^{1-1/p}$-approximation by solving standard shortest path with edge costs $\|c_e\|_1$.

Introducing vector-valued costs to the graph's edges allows this model to capture a number of different applications. First, it allows us to describe a situation in which different parties, each corresponding to a different coordinate of the cost vectors, incur different cost when an edge is added to the solution. In this interpretation, as $p \to \infty$, the problem will increasingly favor paths in which every party simultaneously incurs small cost. Alternatively, each coordinate of the cost vectors may represent the cost incurred in terms of a different resource. This model would then allow one to balance minimizing the total amount of resources spent and ensuring that no single resource is depleted. Furthermore, one can think of this problem as providing an avenue for modeling robustness of a solution in the presence of uncertainty. Each coordinate would then represent the cost incurred by adding an edge in a distinct possible scenario, and the value of the ℓ_p-Shortest Path problem would amount to a trade-off between average and worst-case cost among all scenarios. Finally, this problem generalizes congestion minimization in directed graphs (a fact that we prove in Section 8 of the full version of the paper [24]).

Our results for the ℓ_p-Shortest Path problem. In this paper, we introduce a natural flow-based sum-of-squares (SoS) relaxation for ℓ_p-Shortest Path (Section 3) and present approximation algorithms for all integer $p \geq 1$.

First, we give an $O(pd^{1-1/p})$-approximation algorithm for the problem running in $n^{O(p)}$ time for series-parallel graphs of depth/order d (Section 5). We do this by considering a natural rounding algorithm for the SoS relaxation. We prove the following theorem:

▶ **Theorem 1.** *There exists an approximation algorithm for the ℓ_p-Shortest Path problem in series-parallel graphs that, given a series-parallel graph G of order/depth d and parameters $p \in \mathbb{Z}_{\geq 1}$ and $\varepsilon \in (0, 1)$, finds a $(1 + \varepsilon)\mathcal{B}_d(p)^{1/p} = O(pd^{1-1/p})$ approximation in time $m^{O(p)}/\varepsilon^{O(1)}$ (which is polynomial time when p and ε are fixed). Here, $\mathcal{B}_d(p)$ is the p^{th} d-dimensional Bell number.*[1]

[1] We provide a review of Bell numbers in the preliminaries.

For graphs of small series-parallel order/depth $d \le \log^* p$, the approximation factor is $\mathscr{B}_d^{1/p} \le O(p/\log^{(d)} p)$. Remarkably, in a complementary analysis given in Section 8 of the full version of the paper, we show that the approximation factor $\mathscr{B}_d(p)^{1/p}$ is tight for our rounding scheme. In all algorithms, we assume that ℓ is at most polynomial in n (if not, the running times will also depend on ℓ).

Then, we give a $O(p\log^{1-1/d} n)$-approximation for arbitrary graphs (Section 6 of the full version of the paper), obtaining the following theorem:

▶ **Theorem 2.** *There exists an approximation algorithm for the ℓ_p-Shortest Path problem in arbitrary graphs that, given a graph G and parameters $p \in \mathbb{Z}_{\ge 1}$ and $c \in (0, 1/2)$, finds a $cp\log^{1-1/p} n$ approximation in time $m^{ce^{O(1/c)}\log n} = m^{O_c(\log n)}$.*

Note that when $p = \log \ell$, this result yields an $O(c \log n \log \ell)$ approximation. This is very similar to the approximation guarantee of $O(\log n \log k)$ by Li, Xu, and Zhang, except that in our algorithm c can be an arbitrarily small constant.

▶ Remark 3. As previously discussed, an $\ell^{1-1/p}$-approximation is trivial to achieve in polynomial time by solving standard shortest path with edge costs $\|c_e\|_1$. On the other hand, for each fixed ℓ' there is a polynomial-time approximation scheme (PTAS) for ℓ_p-Shortest Path in ℓ' dimensions that runs in time $n^{O(\ell')}$. We now explain how to combine these two results. Fix $\delta \in (1/\ell, 1/2)$. Divide the coordinates of the cost vectors $c_e \in \mathbb{R}^\ell$ into $\ell' = \lceil 1/\delta \rceil$ groups, each of size at most $k = \lceil \delta \ell \rceil$ and then add up the coordinates in each group. For each cost vector $c_e \in \mathbb{R}^\ell$, we obtain a new vector $c'_e \in \mathbb{R}^{\ell'}$. Costs c'_e approximate costs c_e within a factor of $k^{1-1/p}$ in the following sense: for every path P,

$$\Big\|\sum_{e\in P} c_e\Big\|_p \le \Big\|\sum_{e\in P} c'_e\Big\|_p \le k^{1-1/p}\Big\|\sum_{e\in P} c_e\Big\|_p. \tag{1}$$

Using the PTAS, we solve the problem with costs c'_e and by (1) get a $(1+\varepsilon)k^{1-1/p}$ approximation to the original problem. We conclude that there exists an approximation algorithm that finds an $O((\delta\ell)^{1-1/p})$ approximation in time $n^{O(1/\delta)}$ (for every $\delta \in (1/\ell, 1/2)$).

In the course of analyzing our algorithms, we prove a new majorization inequality for pseudo-expectations (see Section 4) generalizing pseudo-expectation Lyapunov's and Hölder inequality (for the latter see [2, arXiv version]). We believe this result to be of independent interest.

Hardness results. We also complement the analysis above with several hardness results for ℓ_p-Shortest Path. First, in Section 8.1 of the full version of the paper, we give a reduction showing that the problem of congestion minimization can be reduced to the ℓ_∞-Shortest Path problem. This simultaneously speaks to the broad applicability of ℓ_p-Shortest Path and implies hardness for the ℓ_∞-version of the problem, following a result of Chuzhoy and Khanna [9].

▶ **Theorem 4.** *The ℓ_∞-Shortest Path problem is hard to approximate within an $\Omega(\log n/\log\log n)$-factor unless* NP $\subseteq$ ZPTIME$(n^{\log\log n})$.

This theorem slightly strengthens the $\Omega(\log^{1-\varepsilon}\ell)$-hardness of approximation result by Kasperski and Zieliński [18]. Finally, in Section 8.2 of the full version of the paper, we show that allowing the entries of the cost vectors to be negative makes the problem substantially harder. We do this by giving a reduction from the Closest Vector problem in lattices to this (potentially negative costs) version of the ℓ_p-Shortest Path problem. Below, Ω_p hides a constant depending on p.

▶ **Theorem 5.** *For every $p \in [1, \infty]$, it is* NP*-Hard to approximate the ℓ_p-Shortest Path problem allowing negative edge costs within a factor of $n^{\Omega_p(1/\log\log n)}$.*

▶ Remark 6. The requirement that all the coordinates of cost vectors c_e are non-negative can be slightly relaxed when $p = 2$. For our algorithms to work, it is sufficient that all pairwise inner products of cost vectors are non-negative. That is, instead of requiring that the Gram matrix of cost vectors is completely positive, we can only require that it is doubly non-negative.

From shortest path to network design. In network design problems, one is given a graph $G = (V, E)$ with non-negative edge costs $c_e \geq 0$, and wishes to find a subgraph $F = (V_F, E_F)$ that minimizes the total cost $\sum_{e \in E_F} c_e$ subject to some connectivity constraints. By varying the set of allowed subgraphs F, this paradigm encapsulates many central and well-studied network design problems, including the Survivable Network Design, Steiner Forest, Steiner Tree, and Minimum Spanning Tree. In this paper we explore two network design problems, Group Steiner Tree and Asymetric Traveling Salesperson (ATSP). We first recall the Group Steiner Tree problem.

▶ **Problem 7** (Group Steiner Tree)**.** *Given a weighted undirected graph $G = (V, E, c)$, as well as k subsets $R_1, \ldots, R_k$ of V, find a minimum-cost subtree T of G containing at least one vertex from each R_i.*

We then introduce an analogous group variant of ATSP:

▶ **Problem 8** (Group ATSP)**.** *Given a weighted directed graph $G = (V, E, c)$ and a collection of subsets $R_1, \ldots, R_k$ of V, find a minimum-cost tour that visits at least one vertex in each R_i.*

As in the case of Shortest Path, it is natural to ask whether we can approximately solve ℓ_p versions of other network design problems efficiently. Prior work has been done in this area. Hamacher and Ruhe [15] studied ℓ_∞-Minimum Spanning Tree, and proved that it is NP-complete. Following that, the complexity of the problem has been nearly completely settled: Chekuri, Vondrák, and Zenklusen [8] presented an $O(\log \ell / \log\log \ell)$-approximation algorithm, while Kasperski and Zielinski [19] (also see [20], Table 1) proved an $\Omega(\log^{1-\varepsilon} \ell)$-hardness of approximation for every $\varepsilon > 0$, unless all problems in NP can be solved in quasi-polynomial time. Laddha, Singh and Vempala [22] studied the ℓ_∞-version of a subclass of network design problems which encompasses the Generalized Steiner Network problem, and gave a polynomial-time ℓ-approximation algorithm for it.

Our results for ℓ_p-Group Steiner Tree and ℓ_p-Group ATSP. We consider the ℓ_p-version of the Group Steiner Tree and the Group ATSP problems.

In Section 7 of the full version of the paper, we refine the SoS relaxation for ℓ_p-Shortest Path to obtain approximation algorithms for ℓ_p-Group ATSP and ℓ_p-Group Steiner Tree, and thus obtain approximation algorithms for these problems as well. In particular, we prove the following two results:

▶ **Theorem 9.** *There exists an approximation algorithm for ℓ_p-Group ATSP that given graph G, groups R_i, and parameters $p \in \mathbb{Z}_{\geq 0}$ and $c \in (0, 1/2)$ finds a $c^2 p \log^{2-1/p} n \log k$ approximation in time $m^{O(p)+ce^{O(1/c)}\log n} = m^{O_c(\log n)}$. We assume that k is at most polynomial in n.*

▶ **Theorem 10.** *There exists an approximation algorithm for the ℓ_p-Group Steiner Tree problem in undirected graphs that given a graph G, groups R_i, and parameters $p \in \mathbb{Z}_{\geq 1}$ and $c \in (0, 1/2)$ finds a $c^2 p \log^{2-1/p} n \log k$ approximation in time $m^{ce^{O(1/c)}\log n} = m^{O_c(\log n)}$. We assume that k is at most polynomial in n.*

Note that for $p = \lceil \log \ell \rceil$, we get an approximation algorithm for the ℓ_∞ norm.

1.1 Related Works

Previous results on the scalar-cost group Steiner tree problem. Group Steiner Tree with scalar costs was introduced by Reich and Widmayer [29]. Garg, Konjevod, and Ravi [14] gave an $O(\log^2 n \log k)$-approximation to the problem, the first polylogarithmic approximation. Charikar, Chekuri, Goel, and Guha [6] gave the same $O(\log^2 n \log k)$-approximation with a deterministic algorithm. Then Charikar, Chekuri, Cheung, Dai, Goel, Guha, and Li [5] gave an $O(\log^3 k)$ approximation that works even with directed graphs; however, it required quasipolynomial time. Finally, Chekuri and Pál [7] gave an $O(\log^2 k)$ approximation for undirected graphs also in quasipolynomial time. The approximation guarantees of [14] and [6] are presented above with the improvement resulting from using metric embedding by Fakcharoenphol, Rao and Talwar [12]. In terms of the approximation guarantee and running time, out algorithm is most similar to that by Chekuri and Pál [7]: when $k = \Theta(n)$, the approximation guarantees match. In terms of techniques used, our algorithm uses some ideas from that by Garg, Konjevod, and Ravi [14].

Dijkstra-style Algorithm for ℓ_p-Shortest Path. The authors of [3] describe a Dijkstra-style algorithm for the ℓ_p-Shortest Path problem and claim that it achieves an $O(\min\{p, \log \ell\})$-approximation. However, we show that this claim is incorrect and, in fact, the approximation factor of their algorithm is at least $\Omega(n^{1-1/p})$. We discuss this algorithm in Appendix B of the full version of the paper.

Multi-objective combinatorial optimization for shortest path and network design. The work in this paper is closely related to multi-objective combinatorial optimization (MOCO). This area studies combinatorial optimization problems in the presence of multiple competing objective functions. Much of the literature on MOCO is concerned with finding all or some Pareto efficient solutions, that is, solutions that are not dominated in every objective by any other solution, a problem which is often intractable due to the exponential number of these points. In particular, there is prior MOCO work on both shortest path [16, 25, 4] and network design problems [28]. We refer the reader to the paper of Ruzika and Hamacher [30] for a survey on multi-objective spanning tree problems, and the book of Ehrgott [11] for an overview of multi-criteria optimization area as a whole.

1.2 Technical Overview

Let us first discuss the ℓ_p-Shortest Path problem. The most basic variant of this problem is when G is a series-parallel graph (see Section 2 for definitions) and $p = 2$. The first idea is to write an LP flow relaxation with a convex objective: minimize $\| \sum_{e \in E} c_e x_e \|^2$ subject to the constraint that $(x_e)_{e \in E}$ define a unit flow from s to t. However, this LP has an integrality gap of $\sqrt{\ell}$. To deal with this problem, Li, Xu, and Zhang [23] introduced a new constraint for ℓ_∞-Shortest Path: the constraint loosely speaking says that the LP cost of every subgraph/block B in the series-parallel decomposition of G is at most OPT times the probability (according to the LP) that the path visits B. Unfortunately, this new constraint does not help when $p = 2$.

Instead, we consider a sum-of-squares (SoS) strengthening of this LP.[2] The SoS relaxation gives valuations not only to individual edges but also to tuples of edges. Using the standard notation of pseudo-expectations (see Section 2), the SoS relaxation for $p = 2$ gives $\widetilde{\mathbb{E}}[x_e]$ for every edge e and $\widetilde{\mathbb{E}}[x_{e_1}x_{e_2}]$ for every pair of edges e_1 and e_2. The former could be interpreted as the probability that $e \in P$ and the latter as the probability as both $e_1, e_2 \in P$ according to the relaxation. To the best of our knowledge, this is the first flow-based SoS or SDP relaxation studied in the literature.

Our algorithm is very straightforward. We start at $u_0 = s$, then choose one of the edges outgoing from u_1 with probability of choosing e being equal to $\widetilde{\mathbb{E}}[x_e]$. We get to a vertex u_1 and then again sample one of the edges leaving u_1 with probability of choosing e proportional to $\widetilde{\mathbb{E}}[x_e]$. We repeat this step over and over until we reach t. It is clear that the algorithm finds an s-t path P.

Now we need to upper bound the cost of P. We do that recursively using the series-parallel decomposition of G.[3] Assume that G is composed of subgraphs/blocks $B_1, \dots, B_t$ and our algorithm achieves an α approximation for the squared ℓ_2-cost in each of them. For simplicity, assume that $t = 2$ for now. There are two cases: G is a (i) parallel and (ii) series composition of B_1 and B_2. Consider the first case. The SoS relaxation ensures that $\widetilde{\mathbb{E}}[x_{e_1}x_{e_2}] = 0$ for all $e_1 \in B_1$ and $e_2 \in B_2$; this means that the SoS solution is simply a convex combination of solutions for B_1 and B_2 with some weights p_1 and p_2. Also, with probability p_1, the first edge of P will be in B_1 and then the entire path will be in B_1; similarly, with probability p_2, the entire path will be in B_2. Thus running the algorithm reduces to randomly choosing a block B_i with probability p_i and then running the algorithm in B_i. Since the algorithm gets an α approximation in each B_i, it also gets an α approximation in the entire graph. Interestingly, this step would already fail if we used the basic LP relaxation; however, a Sherali–Adams or configuration LP would work in *this* case.

The second case – when G is a series composition of B_1 and B_2 is more challenging and requires the power of an SDP relaxation. Let $P_i = P \cap B_i$. Write the squared objective as follows:

$$\Big\|\sum_{e\in P} c_e\Big\|_2^2 = \underbrace{\Big\|\sum_{e\in P_1} c_e\Big\|_2^2 + \Big\|\sum_{e\in P_2} c_e\Big\|_2^2}_{\leq \alpha \mathsf{OPT}^2 \text{ (in expectation)}} + 2\sum_{\substack{e_1\in P_1\\ e_2\in P_2}} \langle c_{e_1}, c_{e_2}\rangle. \tag{2}$$

The first two terms are squared ℓ_2-costs of paths P_1 and P_2. As we assumed, they are at most α times their SoS costs, and thus their sum is at most $\alpha\mathsf{OPT}^2$ (in expectation). We now analyze the third term. It is not hard to see that our algorithm samples edges in P_1 and P_2 independently (because the last vertex of P_1 and the first vertex of P_2 are fixed). Therefore,

$$\mathbb{E}\Big[\sum_{\substack{e_1\in P_1\\ e_2\in P_2}} \langle c_{e_1}, c_{e_2}\rangle\Big] = \sum_{\substack{e_1\in B_1\\ e_2\in B_2}} \langle c_{e_1}, c_{e_2}\rangle \cdot \Pr\left(e_1, e_2 \in P\right) = \sum_{\substack{e_1\in B_1\\ e_2\in B_2}} \langle c_{e_1}, c_{e_2}\rangle \cdot \Pr\left(e_1 \in P\right) \cdot \Pr\left(e_2 \in P\right). \tag{3}$$

[2] It is sufficient to use a vector-flow SDP with one vector variable per edge in order to approximate the ℓ_2-cost in series-parallel graphs. However, we need higher degree SoS relaxations when $p > 2$ and in general graphs.

[3] Interestingly, neither the relaxation nor the algorithm uses the series-parallel decomposition of G.

It would be natural to upper bound this expression by the corresponding expression in the SoS objective (appropriately scaled):

$$\sum_{\substack{e_1\in B_1\\ e_2\in B_2}} \langle c_{e_1}, c_{e_2}\rangle \cdot \widetilde{\mathbb{E}}[x_{e_1}x_{e_2}].$$

However, this is not possible, since it may happen that $\Pr(e_1 \in P)\cdot\Pr(e_2\in P) > 0$ but $\widetilde{\mathbb{E}}[x_{e_1}x_{e_2}] = 0$. Instead, observing that for every edge e, $\Pr(e\in P) = \widetilde{\mathbb{E}}[x_e]$, we rewrite and upper bound (3):

$$\sum_{\substack{e_1\in B_1\\ e_2\in B_2}} \langle c_{e_1}, c_{e_2}\rangle \widetilde{\mathbb{E}}[x_{e_1}]\widetilde{\mathbb{E}}[x_{e_2}] \le \sum_{e_1,e_2\in E} \langle c_{e_1}, c_{e_2}\rangle \cdot \widetilde{\mathbb{E}}[x_{e_1}]\cdot\widetilde{\mathbb{E}}[x_{e_2}]$$
$$= \Big\|\widetilde{\mathbb{E}}\Big[\sum_{e\in E} c_e x_e\Big]\Big\|_2^2 \le \widetilde{\mathbb{E}}\Big[\Big\|\sum_{e\in E} c_e x_e\Big\|^2\Big] \le \mathsf{OPT}^2.$$

Here, we first expanded the summation, then used the pseudo-expectation Lyapunov's inequality $\|\widetilde{\mathbb{E}}[f]\|_2^2 \le \widetilde{\mathbb{E}}[\|f\|_2^2]$ (see Fact 18), and finally observed that the last pseudo-expectation is the SoS objective for G. We conclude that the expected squared cost of P is at most $(\alpha+2)\mathsf{OPT}$. Applying this argument recursively, we get an $O(d)$-approximation for the squared cost and an $O(\sqrt{d})$-approximation for the cost itself in series-parallel graphs of order/depth d.

When $p > 2$ and blocks in the series-parallel composition of G are formed by $t > 2$ lower-order blocks, the proof becomes more technical. In particular, we need to use a new majorization inequality for pseudo-expectation, which we present in Section 4.

The SoS relaxation for arbitrary graphs is the same as that for series-parallel graphs (except that its degree is higher). However, the rounding algorithm is quite different. Very informally, the algorithm in its simplest form resembles Savitch's algorithm for s-t connectivity in $O(\log^2 n)$ space [31] (see also [7]): (i) we sample the middle edge $e = (u, v)$ of the path using probabilities provided by $\widetilde{\mathbb{E}}[\cdot]$, (ii) condition $\widetilde{\mathbb{E}}[\cdot]$ on e being the middle edge, (iii) then recursively find paths P_1 from s to u and (independently) P_2 from v to t, using the *conditional* pseudo-expectation. To upper bound the cost, as in the analysis of the algorithm for series-parallel graphs, we first use (2) , then bound the third term using a variant of (3), and finally use the majorization inequality for pseudo-expectations.

To solve Group ATSP, we loosely speaking add SoS constraints that require that the tour P visits every group (for technical reasons, we need to require that P visits each group exactly once). Then we run the rounding algorithm for ℓ_p-Shortest Path in arbitrary graphs. It is not guaranteed that P indeed visits every group; however, using the machinery we developed for bounding the cost of P, we show that P visits every group with probability at least $\Omega(1/\log n)$. By sampling sufficiently many tours and concatenating them, we obtain the desired solution with high probability. The Group Steiner Tree problem easily reduces to Group ATSP.

1.3 Paper Organization

The rest of the paper is organized as follows. In Section 2 we define series-parallel graphs, relevant combinatorial quantities and notation used in the rest of the paper, and give some basic facts on Sum-of-Squares relaxations. In Section 3, we describe our Sum-of-Squares relaxation for ℓ_p-Shortest Path in directed acyclic graphs. In Section 4 we show a majorization inequality for pseudo-expectations used in the analysis of our algorithms. In Section 5 we describe and analyze our rounding algorithm for ℓ_p-Shortest Path in series-parallel graphs. In Section 6 we describe our approximation algorithm for ℓ_p-Shortest Path in arbitrary graphs.

The remaining sections appear in the full version of the paper [24]. Section 6 of the full version of the paper contains the proofs of the theorems that have been ommitted in this version. In Section 7 of the full version, we present our algorithms for ℓ_p-Group ATSP and ℓ_p-Group Steiner Tree. In Section 8 of the full version, we give our hardness results: hardness of approximation results for ℓ_p-Shortest Path with potentially negative edge costs and for ℓ_∞-Shortest Path. We also show that our analysis of the ℓ_p-Shortest Path algorithm in series-parallel graphs is tight. In Appendix A of the full version, we prove a recurrence formula and upper bound on multidimensional Bell numbers, which are used in the analyses of our algorithms for ℓ_p-Shortest Path.

2 Preliminaries and Notation

In this paper, all logs are base 2. In this paper, we consider Shortest Path and Group ATSP in directed graphs and Group Steiner Tree in undirected graphs. We assume that graphs may have parallel edges. Let $G = (V, E)$ be a directed graph. We denote $n = |V|$ and $m = |E|$. For $v \in V$, denote the sets of its outgoing and incoming edges by $\delta^+(v)$ and $\delta^-(v)$, respectively. Similarly, define $\delta^+(A)$ and $\delta^-(A)$ for subsets of vertices A. Finally, denote the set of edges from A to B by $\delta(A, B)$. We denote the i-th coordinate of vector edge cost c_e by $c_e(i)$.

2.1 Series-Parallel Graphs

We start with providing a recursive definition of directed series-parallel graphs with source s and sink t. A graph on two vertices s, t and one or more edges from s to t is a series-parallel graph of order (depth) 0. We denote the order of G as $\mathrm{ord}(G)$.

- **Parallel Composition.** Let $B_1, \ldots, B_t$ be series-parallel graphs that share only vertices s and t. Then their union G is a series-parallel graph. Define $\mathrm{ord}(G) = \max_j \mathrm{ord}(B_j)$.
- **Series Composition.** Let $B_1, \ldots, B_t$ be series-parallel graphs. Denote the source and sink of B_i by s_i and t_i (respectively). Assume that $t_i = s_{i+1}$ for all $i \in \{1, \ldots, t-1\}$ and that graphs B_i do not share any other vertices. Then the union G of graphs B_i is a series-parallel graph. Define $\mathrm{ord}(G) = \max_j \mathrm{ord}(B_j) + 1$.

In this definition, we only count *series* compositions when we compute the order of a series-parallel graph. We call vertices s and t *terminals*. We call intermediate graphs that we obtain while constructing G *blocks*. We denote the source and sink of a block B by s_B and t_B, respectively.

2.2 Combinatorics

Unlabeled Partitions. We say that a tuple of integers $\lambda = (\lambda_1, \ldots, \lambda_k)$ is an unlabeled partition of an integer $n \geq 1$ if $n = \sum_{i=1}^k \lambda_i$ and $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k \geq 1$. We will denote this by $\lambda \vdash n$. We will denote the length of $\lambda = (\lambda_1, \ldots, \lambda_k)$ by $|\lambda| = k$.

Given an n and some tuple of non-negative integers α with $\alpha_1 + \ldots + \alpha_k = n$, we use standard notation for the multinomial coefficient

$$\binom{n}{\alpha} \stackrel{\text{def}}{=} \frac{n!}{\prod_{i=1}^k \alpha_i!}$$

Multidimensional Bell Numbers. Recall that the nth Bell number $\mathcal{B}_n$ equals the number of labeled partitions of a set of size n. In this paper, we will need a generalization of Bell numbers, known as multidimensional Bell numbers (see [32, Example 5.2.4] and [10]).

▶ **Definition 11.** *We say that a collection of subsets P is a partition of a set S if all subsets in P are disjoint and their union is S. Consider two partitions P and P' of S. We say that P' is a refinement of P if every $A \in P'$ is a subset of some $B \in P$.*

A d-dimensional partition of p is a tuple $(P_1, \dots, P_d)$ where all P_i are partitions of $[p] \stackrel{\text{def}}{=} \{1, \dots, p\}$ and each P_{i+1} is a refinement of P_i. The d-dimensional Bell number $\mathscr{B}_d(p)$ is the number of d-dimensional partitions of p. If $d = 0$ or $p = 0$, we let $\mathscr{B}_d(p) \stackrel{\text{def}}{=} 1$.

Note that 1-dimensional Bell numbers are simply the standard Bell numbers: $\mathscr{B}_1(i) = \mathscr{B}(i)$. We can also restate the definition of $\mathscr{B}_d(n)$ as follows. $\mathscr{B}_d(n)$ is the number of $(d+2)$-level rooted trees with n labeled leaves: the root must be in level 0, all leaves must be in level $d+1$, and all leaves are labeled with numbers from 1 to n with each number being used exactly once.

We will need the following recurrence formula for d-dimensional Bell numbers, which is proved in Appendix A of the full version of the paper.

▶ **Lemma 12.** *For every $d \geq 1$ and $p \geq 1$, we have*

$$\mathscr{B}_d(p) = \sum_{\lambda \vdash p} \binom{p}{\lambda} \prod_{i=1}^{|\lambda|} \mathscr{B}_{d-1}(\lambda_i) \Big/ \prod_{j=1}^{p} \mathrm{count}(j, \lambda)!$$

where $\mathrm{count}(j, \lambda)$ is the number of times j appears in λ.

Now, we describe the exponential generating function for sequence $(\mathscr{B}_d(i))_i$ when d is fixed.

▶ **Fact 13** ([32, Example 5.2.4])**.** *Let $f_0(x) = \exp(x)$ and $f_{i+1}(x) = \exp(f_i(x) - 1)$. Then the exponential generating function for sequence $(\mathscr{B}_d(i))_{i=0}^{\infty}$ is given by:*

$$\sum_{i=0}^{\infty} \frac{\mathscr{B}_d(i) x^i}{i!} = f_d(x). \tag{4}$$

In this paper, we will present an approximation algorithm for ℓ_p-Shortest Path in depth-d series-parallel graphs with approximation factor $\mathscr{A}_d(p) \stackrel{\text{def}}{=} \mathscr{B}_d(p)^{1/p}$. From Fact 13, we obtain the following upper bound on $\mathscr{A}_d(p)$, proved in Appendix A of the full version of the paper.

▷ Claim 14. For all $p \geq 1$ and $d \geq 1$, we have

$$\mathscr{A}_d(p) = \mathscr{B}_d(p)^{1/p} = O(p d^{1-1/p}).$$

Let $\log^{(j)} p = \underbrace{\log \cdots \log}_{j \text{ times}} p$ and $\log^* p$ be the largest value of j such that $\log^{(j)} p \geq 1$. Then, the following upper bound on $\mathscr{A}_d(p)$ holds for $d \leq \log^* p$:

$$\mathscr{A}_d(p) = \mathscr{B}_d(p)^{1/p} \leq O\Big(\frac{p}{\log^{(d)} p}\Big).$$

2.3 Sum-of-Squares Relaxations

We recall some basics about the sum-of-squares relaxations. Sum-of-squares relaxations can be thought of in terms of moment matrices, pseudo-distributions, and pseudo-expectations. As is common, we will use the pseudo-expectation framework in this paper. We refer the reader to [13] for a detailed description of the sum-of-squares framework.

Consider a set of variables x_i where i belongs to some set of indices $\mathcal{I}$. We denote the entire collection of all variables $(x_i)_{i\in\mathcal{I}}$ by $\mathbf{x}$. Consider the set of multivariate polynomials $\mathbb{R}_{\le d}[\mathbf{x}] \stackrel{\text{def}}{=} \mathbb{R}_{\le d}[\{x_i : i \in \mathcal{I}\}]$ in variables x_i of degree at most d. We say that $f \in \mathbb{R}_{\le d}[\mathbf{x}]$ is a sum of squares (SoS) if $f = \sum_{i=1}^m f_i^2$ for some polynomials $f_1, \dots, f_m$. Note that the product of SoS polynomials is a SoS, and so is any linear combination of SoS polynomials with positive coefficients.

In this paper, we consider SoS relaxations for the Boolean hypercube; that is, all variables x_i take values 0 and 1 in the intended solution. Therefore, we work with the quotient ring $\mathbb{R}_{\le d}/\langle x_i^2 - x_i\rangle_i$, where $\langle x_i^2 - x_i\rangle_i$ is the ideal generated by polynomials $x_i^2 - x_i$. In other words, we identify monomials $x_{i_1}^{a_1}, \dots, x_{i_t}^{a_t}$ and $x_{i_1}, \dots, x_{i_t}$ for all $i_1, \dots, i_t$ and $a_1, \dots, a_t \ge 1$ such that $\sum_{i=1}^t a_i \le d$. In particular, we will write $f = g$ if $f - g \in \langle x_i^2 - x_i\rangle_i$.

▶ **Definition 15.** *A linear map $\widetilde{\mathbb{E}} : \mathbb{R}_{\le d}[\mathbf{x}]/\langle x_i^2 - x_i\rangle_i \to \mathbb{R}$ is a pseudo-expectation of degree d if it satisfies the following properties.*

- $\widetilde{\mathbb{E}}[1] = 1$,
- $\widetilde{\mathbb{E}}[f^2] \ge 0$ *for every polynomial f of degree at most $d/2$,*

We say that a pseudo-expectation $\widetilde{\mathbb{E}}$ satisfies an equality constraint $f = 0$ if $\widetilde{\mathbb{E}}[fg] = 0$ whenever $\deg fg \le d$.

Given an objective function f and sets of equality and inequality constraints, we can find a pseudo-expectation $\widetilde{\mathbb{E}}$ that maximizes $\widetilde{\mathbb{E}}[f]$ and satisfies all the constraints in time polynomial in $N^{O(d)}$, where N is the number variables, as long as it satisfies certain regularity conditions [27]. When we prove any statements about pseudo-expectations $\widetilde{\mathbb{E}}[f]$ below, we will *always* implicitly assume that $d \ge \Omega(\deg f)$ so that all the inequalities appearing in the proofs have degree at most d.

▶ **Definition 16.** *Let $\widetilde{\mathbb{E}}$ be a pseudo-expectation of degree d. Assume that g is a sum of squares and $\widetilde{\mathbb{E}}[g] > 0$. Then the conditional pseudo-expectation $\widetilde{\mathbb{E}}[\cdot \mid g]$ operator is defined as follows: $\widetilde{\mathbb{E}}[f \mid g] \stackrel{\text{def}}{=} \widetilde{\mathbb{E}}[fg]/\widetilde{\mathbb{E}}[g]$.*

▶ **Fact 17.** *A conditional pseudo-expectation $\widetilde{\mathbb{E}}[\cdot \mid g]$ is a pseudo-expectation of degree $d' = d - \deg g$. If $\widetilde{\mathbb{E}}$ satisfies an equality or inequality constraint of degree at most d', then so does $\widetilde{\mathbb{E}}$.*

We will use Lyapunov's inequality for pseudo-expectations (which is also referred to as Jensen's inequality in the literature).

▷ Claim 18. Let g be a sum of squares and f be any polynomial. Assume that $\deg f^2 g \le d$. Then,

$$\widetilde{\mathbb{E}}[fg]^2 \le \widetilde{\mathbb{E}}[f^2 g]\,\widetilde{\mathbb{E}}[g]. \tag{5}$$

If $\widetilde{\mathbb{E}}[g] > 0$, the inequality can be restated as

$$\widetilde{\mathbb{E}}[f \mid g]^2 \le \widetilde{\mathbb{E}}[f^2 \mid g]. \tag{6}$$

▷ Claim 19. Let $f_1, \dots, f_t$ be SoS polynomials. Then, $\widetilde{\mathbb{E}}[\left(\sum_{i=1}^t f_i\right)^p] \ge \sum_{i=1}^t \widetilde{\mathbb{E}}[f_i^p]$.

Proof. We expand $\left(\sum_{i=1}^t f_i\right)^p$ as $\sum_{\substack{\alpha_1,\dots,\alpha_t \ge 0 \\ \alpha_1+\cdots+\alpha_t = p}} \binom{p}{\alpha} f_1^{\alpha_1} \cdots f_t^{\alpha_t}$. All terms in the expansion are SoS polynomials and thus have non-negative pseudo-expectations. The claim follows from the observation that all terms f_i^p are present in the expansion. ◁

3 Sum of Squares Relaxation for ℓ_p-Shortest Path

In this section, we first present our SoS relaxation for ℓ_p-Shortest Path in directed acyclic graphs (DAGs). In Section 5, we will present a rounding algorithm for series-parallel graphs and then, in Section 6 of the full version of the paper, for layered graphs. The latter result will also yield an algorithm for arbitrary graphs. We will also describe a few basic properties that feasible solutions for this relaxation satisfy.

Relaxation. We use a degree $2p$ SoS relaxation with variables $\mathbf{x} = (x_e)_{e\in E}$ for ℓ_p-Shortest Path in series-parallel graphs.

$$\begin{aligned} \min \quad & \widetilde{\mathbb{E}}\Big[\sum_{i=1}^{\ell}\Big(\sum_e c_e(i)x_e\Big)^p\Big] \\ \text{subject to} \quad & (x_e)_{e\in E} \text{ is a unit flow from } s \text{ to } t \end{aligned}$$

The flow constraint says that $\sum_{e\in\delta^+(u)} x_e - \sum_{e\in\delta^-(u)} x_e = 0$ for all u other than s and t (flow conservation) and $\sum_{e\in\delta^+(s)} x_e - 1 = 0$ (x_e sends 1 unit of flow from s to t). It is clear that this is a relaxation for the ℓ_p-Shortest Path problem: $\widetilde{\mathbb{E}}[\sum_{i=1}^{\ell}(\sum_e c_e(i)x_e)^p\Big] \le \mathsf{OPT}^p$, where OPT is the ℓ_p-cost of the optimal s-t path.

Basic properties of the SoS relaxation. We say that two edges e_1 and e_2 are *compatible* if both of them belong to some s-t path; otherwise, we say that e_1 and e_2 are *incompatible*. In a series-parallel graph edges e_1 and e_2 are incompatible if and only if there exist two parallel blocks B_1 and B_2 such that e_1 lies in B_1 and e_2 lies in B_2. For any set of vertices A, let $\mathbf{x}_A^+ = \sum_{e\in\delta^+(A)} x_e$ and $\mathbf{x}_A^- = \sum_{e\in\delta^-(A)} x_e$.

▷ Claim 20. Assume that G is a DAG and $\widetilde{\mathbb{E}}$ is a feasible pseudo-expectation for the relaxation. Let h be a multivariate polynomial. Then

1. If $A \subseteq V$ contains neither of the terminals, then $\widetilde{\mathbb{E}}[(\mathbf{x}_A^+ - \mathbf{x}_A^-)h] = 0$. If A contains s but not t, $\widetilde{\mathbb{E}}[(\mathbf{x}_A^+ - \mathbf{x}_A^-)h] = \widetilde{\mathbb{E}}[h]$.
2. If e_1 and e_2 are not compatible, then $\widetilde{\mathbb{E}}[x_{e_1}x_{e_2}h] = 0$.
3. Assume further that G is a series-parallel graph. Let (L, R) be an s_B-t_B cut in a block B. Let $f_{LR} = \sum_{e\in\delta(L,R)} x_e$. Then

$$\widetilde{\mathbb{E}}\Big[f_{LR}h\Big] = \widetilde{\mathbb{E}}\Big[\Big(\sum_{e\in\delta^+(s_B)\cap B} x_e\Big)h\Big].$$

In particular, $\widetilde{\mathbb{E}}[f_{LR}h]$ does not depend on the cut (L, R) in B.

Proof.

1. The SoS relaxation satisfies the flow conservation constraints and the constraint that the amount of flow being routed equals 1. Therefore, it satisfies any linear combination of them. In particular, it satisfies degree-1 polynomial equations $\mathbf{x}_A^+ - \mathbf{x}_A^- = 0$ when $s, t \notin A$ and $\mathbf{x}_A^+ - \mathbf{x}_A^- = 1$ when $s \in A$ but $t \notin A$. The first item follows.
2. It is sufficient to prove the statement for all monomials (the claim then follows by the linearity of $\widetilde{\mathbb{E}}$). Thus, we will assume that h is a monomial. Recall that an s-t cut is monotone if it cuts exactly one edge on every s-t path. Since e_1 and e_2 are incompatible, there is a monotone s-t cut $(A, \bar{A})$ that cuts both of them. We apply item 1 to polynomial

$x_{e_1}h$ and get $\widetilde{\mathbb{E}}[x_{e_1}h] = \widetilde{\mathbb{E}}[x_{e_1}\mathbf{x}_A^+h] = \widetilde{\mathbb{E}}[x_{e_1}(x_{e_1}+x_{e_2}+\dots)h]$. Here, $\dots$ is a sum of some x_e (the coefficient of each of them is 1). Note that x_eh is a monomial, thus $x_eh = (x_eh)^2$ and therefore $\widetilde{\mathbb{E}}[x_eh] \geq 0$. We conclude that

$$\widetilde{\mathbb{E}}[x_{e_1}h] \geq \widetilde{\mathbb{E}}[x_{e_1}(x_{e_1}+x_{e_2})h] = \widetilde{\mathbb{E}}[(x_{e_1}+x_{e_1}x_{e_2})h]$$

and simplifying, we get $\widetilde{\mathbb{E}}[x_{e_1}x_{e_2}h] \leq 0$. Since $x_{e_1}x_{e_2}h$ is a monomial, $\widetilde{\mathbb{E}}[x_{e_1}x_{e_2}h] \geq 0$, and thus $\widetilde{\mathbb{E}}[x_{e_1}x_{e_2}h] = 0$.

3. Item 3 follows immediately from item 1. ◁

4 Majorization Inequalities for Pseudo-expectations

In this section, we will prove a majorization inequality for pseudo-expectations. This inequality generalizes already known pseudo-expectation Lyapunov's (see Claim 18) and Hölder's (see [2]) inequalities.

▶ **Definition 21.** *Consider two integer sequences $a_1 \geq a_2 \geq \dots \geq a_k \geq 0$ and $b_1 \geq \dots \geq b_k \geq 0$. We say a majorizes b and write $a \succeq b$, if $\sum_{i=1}^k a_i = \sum_{i=1}^k b_i$ and for all $1 \leq i \leq k$ we have*

$$a_1 + \dots + a_i \geq b_1 + \dots + b_i.$$

Sequence majorization is a powerful tool for proving inequalities; it appears in widely used Muirhead's [26] and Karamata's [17] majorization inequalities. We now present a majorization inequality for pseudo-expectations. An analogous inequality for true expectations easily follows both from Karamata's and from Muirhead's majorization inequality.

▶ **Lemma 22.** *Consider a degree d pseudo-expectation $\widetilde{\mathbb{E}}$. Let $a \succeq b$ and f be an SoS polynomial of degree $\deg(f^{a_1}) \leq d$. Then*

$$\prod_{i=1}^k \widetilde{\mathbb{E}}[f^{a_i}] \geq \prod_{i=1}^k \widetilde{\mathbb{E}}[f^{b_i}]. \tag{7}$$

Proof. First, observe that this inequality for sequences $(r+1, r-1) \succeq (r, r)$ follows from Lyapunov's inequality for pseudo-expectations (Claim 18). Indeed, let $g = f^{r-1}$. Then, Lyapunov's inequality states that

$$\widetilde{\mathbb{E}}[f^r]^2 = \widetilde{\mathbb{E}}[fg]^2 \leq \widetilde{\mathbb{E}}[f^2g]\,\widetilde{\mathbb{E}}[g] = \widetilde{\mathbb{E}}[f^{r+1}]\,\widetilde{\mathbb{E}}[f^{r-1}], \tag{8}$$

as required. Now we use this inequality to show the desired inequality (7) for a more general case $(p+1, q-1) \succeq (p, q)$.

▷ Claim 23. Let $p \geq q \geq 1$ be integers. Then

$$\widetilde{\mathbb{E}}[f^{p+1}]\,\widetilde{\mathbb{E}}[f^{q-1}] \geq \widetilde{\mathbb{E}}[f^p]\,\widetilde{\mathbb{E}}[f^q].$$

Proof. We just proved the inequality when $p = q$. So we will assume below that $p > q$. Since f is an SoS polynomial, $\widetilde{\mathbb{E}}[f^r] \geq 0$ for all integers $0 \leq r \leq p$. Let us assume first that the inequality is strict for all r: $\widetilde{\mathbb{E}}[f^r] > 0$. Then, by dividing and multiplying $\widetilde{\mathbb{E}}[f^q]\,\widetilde{\mathbb{E}}[f^p]$ by $\prod_{r=q}^{p-1} \widetilde{\mathbb{E}}[f^r]$, we obtain the following identity.

$$\widetilde{\mathbb{E}}[f^q]\,\widetilde{\mathbb{E}}[f^p] = \frac{\widetilde{\mathbb{E}}[f^q]\,\widetilde{\mathbb{E}}[f^q]\,\widetilde{\mathbb{E}}[f^{q+1}]\dots\widetilde{\mathbb{E}}[f^{p-1}]\,\widetilde{\mathbb{E}}[f^p]}{\widetilde{\mathbb{E}}[f^q]\,\widetilde{\mathbb{E}}[f^{q+1}]\dots\widetilde{\mathbb{E}}[f^{p-1}]}$$

Now we upper bound the numerator of this fraction by iteratively applying (8).

$$\left(\widetilde{\mathbb{E}}[f^q]\,\widetilde{\mathbb{E}}[f^q]\right)\widetilde{\mathbb{E}}[f^{q+1}]\cdots\widetilde{\mathbb{E}}[f^{p-1}]\,\widetilde{\mathbb{E}}[f^p] \le \left(\widetilde{\mathbb{E}}[f^{q-1}]\,\widetilde{\mathbb{E}}[f^{q+1}]\right)\widetilde{\mathbb{E}}[f^{q+1}]\cdots\widetilde{\mathbb{E}}[f^{p-1}]\,\widetilde{\mathbb{E}}[f^p]$$

$$\widetilde{\mathbb{E}}[f^{q-1}]\left(\widetilde{\mathbb{E}}[f^{q+1}]\,\widetilde{\mathbb{E}}[f^{q+1}]\right)\cdots\widetilde{\mathbb{E}}[f^{p-1}]\,\widetilde{\mathbb{E}}[f^p] \le \widetilde{\mathbb{E}}[f^{q-1}]\left(\widetilde{\mathbb{E}}[f^q]\,\widetilde{\mathbb{E}}[f^{q+2}]\right)\cdots\widetilde{\mathbb{E}}[f^{p-1}]\,\widetilde{\mathbb{E}}[f^p]$$

$$\vdots$$

$$\widetilde{\mathbb{E}}[f^{q-1}]\,\widetilde{\mathbb{E}}[f^q]\,\widetilde{\mathbb{E}}[f^{q+1}]\cdots\left(\widetilde{\mathbb{E}}[f^p]\,\widetilde{\mathbb{E}}[f^p]\right) \le \widetilde{\mathbb{E}}[f^{q-1}]\,\widetilde{\mathbb{E}}[f^q]\,\widetilde{\mathbb{E}}[f^{q+1}]\cdots\left(\widetilde{\mathbb{E}}[f^{p-1}]\,\widetilde{\mathbb{E}}[f^{p+1}]\right)$$

We conclude that

$$\widetilde{\mathbb{E}}[f^q]\,\widetilde{\mathbb{E}}[f^p] \le \widetilde{\mathbb{E}}[f^{q-1}]\,\widetilde{\mathbb{E}}[f^{p+1}],$$

as desired. If $\widetilde{\mathbb{E}}[f^r] = 0$ for some r, we apply the inequality to $\hat{f} = f + \varepsilon$ (where $\varepsilon > 0$). Now $\widetilde{\mathbb{E}}[\hat{f}^r] \ge \varepsilon^r > 0$, since f is a SoS polynomial. Therefore, $\widetilde{\mathbb{E}}[\hat{f}^q]\,\widetilde{\mathbb{E}}[\hat{f}^p] \le \widetilde{\mathbb{E}}[\hat{f}^{q-1}]\,\widetilde{\mathbb{E}}[\hat{f}^{p+1}]$. Letting $\varepsilon \to 0$, we obtain the desired inequality in the limit. ◁

Consider an integer sequence $a_1 \ge \ldots a_k \ge 0$ and two indices $1 \le i^* < j^* \le k$ such that $a_{i^*} - a_{j^*} \ge 2$. Define a *transfer* or *T-transform* as follows: we decrease a_{i^*} by 1, increase a_{j^*} by 1, and then sort the obtained sequence in descending order. Claim 23 implies that Lemma 22 holds for sequence a and sequence b obtained from a by a T-transform.

Finally, we use that if $a \succeq b$ then b can be obtained by a sequence of T-transforms [26]: $a = a^{(0)} \mapsto a^{(1)} \mapsto a^{(2)} \mapsto \cdots \mapsto a^{(T)} = b$. As we proved, the value of the product $\prod_{i=1}^k \widetilde{\mathbb{E}}[f^{a_i^{(t)}}]$ may only decrease each time we apply a T-transform. This concludes the proof of the lemma. ◀

Importantly, conditional pseudo-expectations are pseudo-expectations (see Fact 17) and thus Lemma 22 holds for conditional pseudo-expectations as well.

5 Sum-of-Squares Relaxation Rounding

In this section, we describe and analyze a rounding algorithm for series-parallel graphs. It gives an $(1+\varepsilon)A_d(p) = O(pd)$ approximation for ℓ_p-Shortest Path in series-parallel graphs of order d. Later we use a different algorithm with a similar analysis to solve the problem in layered and arbitrary graphs.

5.1 Algorithm

Let us denote $p_e = \widetilde{\mathbb{E}}[x_e]$ and $p_u = \sum_{e\in\delta^+(u)} \widetilde{\mathbb{E}}[x_e]$. Note that the SoS relaxation constraints ensure that p_e is an s-t flow; p_u equals the amount of flow that leaves vertex u. The total amount of flow is 1.

Algorithm 1 Rounding algorithm for SoS.

Input: series-parallel graph G with source s and sink t, a pseudo-expectation $\widetilde{\mathbb{E}}$
Output: an s-t path in G
Let $u = s$ and P be an empty path.
while $u \neq t$ **do**
    Sample $e \in \delta^+(u)$ with probability $\frac{p_e}{p_u} = \frac{\widetilde{\mathbb{E}}[x_e]}{\sum_{e\in\delta^+(u)}\widetilde{\mathbb{E}}[x_e]}$
    Append e to path P
end while
return P

▶ **Lemma 24.** *Let P be the path returned by Algorithm 1. Then $\Pr(e \in P) = p_e$ for every edge e and $\Pr(u \in P) = p_u$ for every vertex u.*

Proof. We consider all vertices in topological order and prove the desired formulas for $\Pr(u \in P)$ and $\Pr((u,v) \in P)$ by induction. For $u = s$, we have $\Pr(s \in P) = 1 = p_s$. Then, $\Pr((s,v) \in P) = p_{(s,v)}/p_s = p_{(s,v)}$, as required. Now assume that we proved the formulas for vertices u' proceeding u in the topological order. We have,

$$\Pr(u \in P) = \sum_{e=(u',u)\in\delta^-(u)} \Pr(e \in P) = \sum_{e=(u,u')\in\delta^-(u)} p_e = p_u.$$

Here, we used the induction hypothesis and the flow conservation condition at vertex u. Now let $e = (u,v)$.

$$\Pr(e \in P) = \Pr(e \in P \mid u \in P)\Pr(u \in P) = (p_e/p_u)\cdot p_u. \qquad \blacktriangleleft$$

Now we will prove an upper bound on the ℓ_p-cost of path P. The proof will be by induction on the series-parallel decomposition of G, going from lower to higher order blocks B. To analyze different blocks B, we first introduce some relevant notation.

Let us say that a path P visits block B if it contains at least one edge from B. Let $P_B = P \cap B$ be the restriction of P to B. If P does not visit B, let $P_B = \varnothing$. Note that if P visits B then it must go through the source s_B and sink t_B of B. However, if B has a parallel block B', a path may go through s_B and t_B but visit B' rather than B itself. It follows from Lemma 24 that the probability that P visits B equals $p_B \stackrel{\text{def}}{=} \sum_{e\in\delta^+(s_B)\cap B} p_e$.

Now, we define conditional expectations and pseudo-expectations restricted to B (that is, conditioned on the event that P visits B). Let $h_B = \sum_{e\in\delta^+(s_B)\cap B} x_e = \sum_{e\in\delta^+(s_B)\cap B} x_e^2$ be a SoS indicator of the event that P visits B. We let

$$\mathbb{E}_B[\cdot] \stackrel{\text{def}}{=} \mathbb{E}[\cdot \mid P \text{ visits } B] \qquad \text{and} \qquad \widetilde{\mathop{\mathbb{E}}_{B}}[\cdot] \stackrel{\text{def}}{=} \widetilde{\mathbb{E}}[\cdot \mid h_B].$$

In the sequel, we shall bound the costs of P coordinate-by-coordinate. Thus, we consider a set of scalar non-negative edge weights $a_e \geq 0$. Define $f_B \stackrel{\text{def}}{=} \sum_{e\in B} a_e \cdot x_e$. For a path P', let $\mathrm{cost}(P') = \sum_{e\in P'} a_e$. Note that $f_B = \sum_{e\in B} a_e \cdot x_e^2$ and thus is a sum of squares.

▷ Claim 25. Let B' be a block inside B (possibly $B' = B$). Assume $p_B > 0$. Then, we have $\mathbb{E}_B[\mathrm{cost}(P_{B'})^r] = \frac{\mathbb{E}[\mathrm{cost}(P_{B'})^r]}{p_B}$ and $\widetilde{\mathbb{E}}_B[f_{B'}^r] = \frac{\widetilde{\mathbb{E}}[f_{B'}^r]}{p_B}$ when $r \in \{1,\dots,p\}$.

Proof. We have

$$\begin{aligned}\mathbb{E}[\mathrm{cost}(P_{B'})^r] &= \mathbb{E}[\mathrm{cost}(P_{B'})^r \mid P_B \neq \varnothing]\cdot\Pr(P_B \neq \varnothing) + \mathbb{E}[\mathrm{cost}(P_{B'})^r \mid P_B = \varnothing]\cdot\Pr(P_B = \varnothing)\\ &= \mathbb{E}_B[\mathrm{cost}(P_{B'})^r]\cdot p_B + 0\cdot(1-p_B) = p_B\cdot\mathbb{E}_B[\mathrm{cost}(P_{B'})^r],\end{aligned}$$

as required. To prove the second identity, consider a monomial g in the expansion of $f_{B'}^r$. We now prove that $\widetilde{\mathbb{E}}[gh_B] = \widetilde{\mathbb{E}}[g]$ and thus $\widetilde{\mathbb{E}}_B[g] \stackrel{\text{def}}{=} \widetilde{\mathbb{E}}[gh_B]/p_B = \widetilde{\mathbb{E}}[g]/p_B$. Note that only x_e with $e \in B'$ appear in g, and $\deg g = r \geq 1$. Choose an arbitrary x_e in g, say $e = (u,v)$ and let g' be such that $g = g'x_e$. Let (L,R) be a monotone s_B-t_B cut in B that cuts e. By Claim 20, item 3,

$$\widetilde{\mathbb{E}}[gh_B] = \widetilde{\mathbb{E}}\Big[g\sum_{e'\in\delta(L,R)} x_{e'}\Big] = \widetilde{\mathbb{E}}\Big[g'\sum_{e'\in\delta(L,R)} x_e x_{e'}\Big].$$

Now all edges $e' \in \delta(L,R)$ other than e are incompatible with e; for them, $\widetilde{\mathbb{E}}[g'x_ex_{e'}] = 0$ by Claim 20, item 2. Also, $\widetilde{\mathbb{E}}[g'x_ex_{e'}] = \widetilde{\mathbb{E}}[g'x_e]$ for $e' = e$. Therefore, $\widetilde{\mathbb{E}}[gh_B] = \widetilde{\mathbb{E}}[g'x_e] = \widetilde{\mathbb{E}}[g]$, as required. ◁

▶ **Lemma 26.** *Let $\widetilde{\mathbb{E}}$ be a feasible solution for the SoS relaxation of ℓ_p-Shortest Path. Let B be a block of order $\hbar = \mathrm{ord}(B)$ with $p_B > 0$. Then for every $r \le p$, we have*

$$\mathbb{E}_B\left[cost(P_B)^r\right] \le \mathscr{B}_\hbar(r)\,\widetilde{\mathop{\mathbb{E}}_B}[f_B^r].$$

Here, $\mathscr{B}_\hbar(r)$ is an $\hbar$-dimensional Bell number (see Section 2.2 for details).

Proof. We will prove the upper bound on $\mathbb{E}\left[\mathrm{cost}(P_B)^r\right]$ by induction on r and on the series-parallel decomposition of B. If $r = 0$ or $\hbar = 0$, the claim trivially holds. We consider two cases: when a block B is a parallel composition and when it is a series composition of lower-level blocks. We start with the former, much simpler case when B is a parallel composition of blocks $B_1, \dots, B_t$ sharing the same source s_B and sink t_B. If P visits B, then it visits exactly one of the blocks $B_1, \dots, B_t$. Therefore,

$$\mathbb{E}_B\left[\mathrm{cost}(P_B)^r\right] = \sum_{i=1}^t \mathbb{E}_{B_i}\left[\mathrm{cost}(P_{B_i})^r\right] \Pr\left(P \text{ visits } B_i | P \text{ visits } B\right) = \sum_{i=1}^t \frac{p_{B_i}}{p_B}\mathbb{E}_{B_i}\left[\mathrm{cost}(P_{B_i})^r\right]. \quad (9)$$

Note that $f_B = \sum_{i=1}^t f_{B_i}$. Applying Claim 19 and then Claim 25 twice, we get

$$\widetilde{\mathop{\mathbb{E}}_B}[f_B^r] \ge \sum_{i=1}^t \widetilde{\mathop{\mathbb{E}}_B}[f_{B_i}^r] = \sum_{i=1}^t \frac{1}{p_B}\,\widetilde{\mathbb{E}}[f_{B_i}^r] = \sum_{i=1}^t \frac{p_{B_i}}{p_B}\,\widetilde{\mathop{\mathbb{E}}_{B_i}}[f_{B_i}^r]. \quad (10)$$

In fact the inequality above is an equality by Claim 20, part 2, but we do not need that here. Comparing (9) and (10) term-by-term, and using the induction hypothesis, we get

$$\mathbb{E}_B\left[\mathrm{cost}(P_B)^r\right] \le \mathscr{B}_\hbar(r)\,\widetilde{\mathop{\mathbb{E}}_B}[f_B^r].$$

This concludes the analysis of this case. Now we assume that B is a series composition of blocks $B_1, \dots, B_t$. In this case, if P visits B then it visits all B_i; if it visits B_i, it visits B and all other B_j. Thus, $p_B = p_{B_1} = \dots = p_{B_t}$. Also, P_B is the concatenation of $P_{B_1}, \dots, P_{B_t}$. Using the multinomial theorem, we get

$$\begin{aligned}\mathbb{E}_B\left[\mathrm{cost}(P_B)^r\right] &= \mathbb{E}_B\left[\left(\sum_{i=1}^t \mathrm{cost}(P_{B_i})\right)^r\right] = \mathbb{E}_B\left[\sum_{\substack{\alpha_1,\dots,\alpha_t \ge 0\\ \alpha_1+\dots+\alpha_t = r}} \binom{r}{\alpha} \prod_{i=1}^t \mathrm{cost}(P_{B_i})^{\alpha_i}\right] \quad (11)\\ &= \sum_{\substack{\alpha_1,\dots,\alpha_t \ge 0\\ \alpha_1+\dots+\alpha_t = r}} \binom{r}{\alpha} \mathbb{E}_B\left[\prod_{i=1}^t \mathrm{cost}(P \cap B_i)^{\alpha_i}\right].\end{aligned}$$

Observe that if P enters B, it necessarily visits $s_{B_1}, \dots, s_{B_t}$ and thus paths $P_{B_1}, \dots, P_{B_t}$ are mutually independent. We then have

$$\mathbb{E}_B\left[\mathrm{cost}(P_B)^r\right] = \sum_{\substack{\alpha_1,\dots,\alpha_t \ge 0\\ \alpha_1+\dots+\alpha_t = r}} \binom{r}{\alpha} \prod_{i=1}^t \mathbb{E}_B\left[\mathrm{cost}(P_{B_i})^{\alpha_i}\right].$$

By Claim 25, $\mathbb{E}_B\left[\mathrm{cost}(P_{B_i})^{\alpha_i}\right] = \mathbb{E}\left[\mathrm{cost}(P_{B_i})^{\alpha_i}\right]/p_B = \mathbb{E}\left[\mathrm{cost}(P_{B_i})^{\alpha_i}\right]/p_{B_i} = \mathbb{E}_{B_i}\left[\mathrm{cost}(P_{B_i})^{\alpha_i}\right]$ and $\widetilde{\mathbb{E}}_B[f_{B_i}^{\alpha_i}] = \widetilde{\mathbb{E}}[f_{B_i}^{\alpha_i}]/p_B = \widetilde{\mathbb{E}}[f_{B_i}^{\alpha_i}]/p_{B_i} = \widetilde{\mathbb{E}}_{B_i}[f_{B_i}^{\alpha_i}]$. Using that $\mathrm{ord}(B_i) \le h - 1$ and applying the induction hypothesis, we get

$$\mathbb{E}_B\left[\mathrm{cost}(P_B)^r\right] = \sum_{\substack{\alpha_1,\dots,\alpha_t\geq 0\\ \alpha_1+\dots+\alpha_t=r}} \binom{r}{\alpha} \prod_{i=1}^{t} \mathbb{E}_B\left[\mathrm{cost}(P_{B_i})^{\alpha_i}\right] = \sum_{\substack{\alpha_1,\dots,\alpha_t\geq 0\\ \alpha_1+\dots+\alpha_t=r}} \binom{r}{\alpha} \prod_{i=1}^{t} \mathbb{E}_{B_i}\left[\mathrm{cost}(P_{B_i})^{\alpha_i}\right]$$

$$\leq \sum_{\substack{\alpha_1,\dots,\alpha_t\geq 0\\ \alpha_1+\dots+\alpha_t=r}} \binom{r}{\alpha} \prod_{i=1}^{t} \mathscr{B}_{\hbar-1}(\alpha_i)\cdot \widetilde{\mathbb{E}}_{B_i} f_{B_i}^{\alpha_i} = \sum_{\substack{\alpha_1,\dots,\alpha_t\geq 0\\ \alpha_1+\dots+\alpha_t=r}} \binom{r}{\alpha} \prod_{i=1}^{t} \mathscr{B}_{\hbar-1}(\alpha_i)\cdot \widetilde{\mathbb{E}}_{B} f_{B_i}^{\alpha_i}.$$

Now, every α in the summation defines an unlabeled partition λ of n: λ is obtained by sorting all non-zero entries α_i of α. Let us denote this $\alpha \to \sigma$. For example, $\alpha = (4,1,0,2,4,0,2,2) \to \lambda = (4,4,2,2,2,1)$. Thus, to go over all α, it is sufficient to go over all $\lambda \vdash r$ and then all α such that $\alpha \to \lambda$. To do the latter, we go over all choices of distinct indices $j_1,\dots,j_{|\lambda|}$ and define α as follows: $\alpha_{j_i} = \lambda_i$ for $i \in [|\lambda|]$ and $\alpha_j = 0$ for all other j. However, if $\lambda_i = \lambda_{i'}$, then indices $j_1,\dots,j_{|\lambda|}$ and those with j_i and $j_{i'}$ swapped define the same α. It is easy to see that the above procedure defines every α exactly $\prod_{j=1}^{r}\mathrm{count}(j,\lambda)$ times. In the example above with $\lambda = (4,4,2,2,2,1)$, this procedure defines every α exactly $2!\cdot 3!$ times. Finally note that $\binom{r}{\alpha} = \binom{r}{\lambda}$. Keeping this discussion in mind, we rewrite the upper bound on $\mathbb{E}_B\left[\mathrm{cost}(P_B)^r\right]$ as follows.

$$\mathbb{E}_B\left[\mathrm{cost}(P_B)^r\right] \leq \sum_{\lambda\vdash r,\ |\lambda|\leq t} \binom{r}{\lambda} \frac{\sum_{j_1,\dots,j_{|\lambda|}\in[t]\ \text{distinct}} \prod_{i=1}^{|\lambda|} \mathscr{B}_{\hbar-1}(\lambda_i)\cdot \widetilde{\mathbb{E}}_B[f_{B_{j_i}}^{\lambda_i}]}{\prod_{j=1}^{r}\mathrm{count}(j,\lambda)}$$

$$\leq \sum_{\lambda\vdash r,\ |\lambda|\leq t} \binom{r}{\lambda} \cdot \frac{\prod_{i=1}^{|\lambda|}\mathscr{B}_{\hbar-1}(\lambda_i)}{\prod_{j=1}^{r}\mathrm{count}(j,\lambda)} \sum_{j_1,\dots,j_{|\lambda|}\in[t]} \prod_{i=1}^{|\lambda|} \widetilde{\mathbb{E}}_B[f_{B_{j_i}}^{\lambda_i}].$$

Note that we removed the requirement that all j_i are distinct in the last inequality (this is valid, since all terms are non-negative). Now we use Claim 19 and then the majorization inequality (see Lemma 22) to upper bound each term in the inner sum.

$$\sum_{j_1,\dots,j_{|\lambda|}\in[t]} \prod_{i=1}^{|\lambda|} \widetilde{\mathbb{E}}_B[f_{B_{j_i}}^{\lambda_i}] = \prod_{i=1}^{|\lambda|} \sum_{j\in[t]} \widetilde{\mathbb{E}}_B\Big[f_{B_j}^{\lambda_i}\Big] = \prod_{i=1}^{|\lambda|} \widetilde{\mathbb{E}}_B\Big[\sum_{j\in[t]} f_{B_j}^{\lambda_i}\Big] \leq \prod_{i=1}^{|\lambda|} \widetilde{\mathbb{E}}_B[f_B^{\lambda_i}] \leq \widetilde{\mathbb{E}}[f_B^r].$$

Using the recurrence relation for multidimensional Bell numbers from Lemma 12, we conclude that

$$\mathbb{E}_B\left[\mathrm{cost}(P_B)^r\right] \leq \sum_{\lambda\vdash r} \frac{\binom{r}{\lambda}\prod_{i=1}^{|\lambda|}\mathscr{B}_{\hbar-1}(\lambda_i)}{\prod_{j=1}^{r}\mathrm{count}(j,\lambda)}\,\widetilde{\mathbb{E}}[f_B^r] = \mathscr{B}_\hbar(r)\,\widetilde{\mathbb{E}}[f_B^r]. \qquad \blacktriangleleft$$

▶ **Theorem 27.** *Algorithm 1 gives an $(1+\varepsilon)\mathscr{A}_d(p) \stackrel{\text{def}}{=} (1+\varepsilon)\mathscr{B}_d(p)^{1/p}$ approximation for the problem in series-parallel graphs of order d in time polynomial in $n^{O(p)}$ and $1/\varepsilon$.*

Proof. We apply Lemma 26 with $a_e = c_e(i)$ to every coordinate $i \in [\ell]$ and add up the obtained upper bounds on $\mathbb{E}\left[\left(\sum_{e\in P} c_e(i)\right)^p\right]$. We get that

$$\mathbb{E}\left[\Big\|\sum_{e\in P} c_e\Big\|_p^p\right] \leq \mathscr{B}_\hbar(p)\,\widetilde{\mathbb{E}}\Big[\sum_{i=1}^{\ell}\Big(\sum_e c_e(i)x_e\Big)^p\Big] \leq \mathscr{B}_\hbar(p)\cdot \mathsf{OPT}^p.$$

By Markov's inequality, $\left\|\sum_{e\in P} c_e\right\|_p^p \leq (1+\varepsilon)\mathscr{B}_\hbar(p)\cdot\mathsf{OPT}^p$ with probability at least $\varepsilon/(1+\varepsilon)$. By running the algorithm $1/\varepsilon$ times, we find a solution of cost at most $\left\|\sum_{e\in P} c_e\right\|_p \leq (1+\varepsilon)^{1/p}\mathscr{B}_\hbar(p)^{1/p}\cdot\mathsf{OPT} \leq (1+\varepsilon)\mathscr{A}_\hbar(p)\cdot\mathsf{OPT}$ with constant probability. (As is standard, we can run this procedure many times and make the failure probability exponentially small.) $\blacktriangleleft$

6 Algorithms for ℓ_p-Shortest Path in Arbitrary Graphs

In this section, we describe an approximation algorithm for ℓ_p-Shortest Path in arbitrary graphs. We note that there is a black-box reduction from the problem in arbitrary graphs to that in series-parallel graphs, which is implicitly used by Li, Xu, and Zhang [23] in their algorithm for ℓ_∞-Shortest Path (which they call Robust s-t Path). This reduction outputs a series-parallel graph with $O(n^{\log n})$ vertices, where n is the number of vertices in the original graph. By using this reduction, we immediately get an $O(p\log^{1-1/p} n)$-approximation algorithm for general graphs with running time $n^{O(p\log n)}$. We describe here how to get an approximation algorithm for general graphs with an improved running time and slightly improved approximation factor; namely we describe how to get a $O(cp\log^{1-1/p} n)$-approximation in time $m^{O(ce^{1/c}\log n)}$ for every $c\in(0,1/2)$. We assume below that ℓ is at most polynomial in n; thus, we may assume $p\le\lceil\log_2\ell\rceil=O(\log n)$ (since all norms $\|\cdot\|_r$ with $r\ge\log_2\ell$ are equivalent within a factor of 2).

Layered graphs and reduction from general graphs to layered graphs. We say that a directed acyclic graph (DAG) $G=(V,E)$ is an s-t layered graph with Δ edge layers if V is the disjoint union of vertex layers $V_0,V_1,\dots,V_\Delta$, E is the disjoint union of layers $E_1,\dots,E_\Delta$, and each edge in E_i goes from V_{i-1} to V_i. Further, we require that $V_0=\{s\}$ and $V_\Delta=\{t\}$.

We transform an arbitrary graph $G=(V_G,E_G)$ with terminals s and t into a layered graph $\hat G=(V_{\hat G},E_{\hat G})$ with $\Delta=n-1$ edge layers. We create vertex layers $V_0,V_1,\dots,V_\Delta$: $V_0=\{s\}$, $V_\Delta=\{t\}$, and each of the other V_is is a disjoint copy of V_G. We connect $\hat u\in V_i$ with $\hat v\in V_{i+1}$ if there is an edge $(u,v)\in E_G$ between the corresponding vertices in G. The vector cost of $(\hat u,\hat v)$ equals that of (u,v). Additionally, we add *padding* edges between copies of t in adjacent layers and assign these edges cost 0.

For every s-t path P with at most Δ edges in G there is a corresponding path $\hat P$ in $\hat G$ and vice versa (P might not be a simple path); if path P has $k<\Delta=n-1$ edges, then $\hat P$ contains k non-padding edges and ends with $\Delta-k$ padding edges. Paths P and $\hat P$ have the same vector costs. Note that the ℓ_p-Shortest Path P^* between s and t is a simple path and thus contains at most $\Delta=n-1$ edges. Therefore, there is a path $\hat P^*$ in $\hat G$ corresponding to P^*. An α-approximation for $\hat P^*$ in $\hat G$ gives an α-approximation for P^* in G. This reduction shows that it is sufficient to consider layered graphs.

An algorithm for layered graphs. Assume that G is a layered graph with Δ edge layers, source s, and sink t. We use the SoS relaxation from Section 3 for G. Let $a=\lceil e^{1/c}\rceil$. We require that $\widetilde{\mathbb{E}}$ be a pseudo-expectation of degree $2d=2(p+(a+1)\lceil\log_{a+1}\Delta\rceil)=\Theta(p+ce^{1/c}\log\Delta)$. For a set of edges A of size at most d, we define polynomial $h_A=\sum_{e\in A}x_a$ and conditional pseudo-expectation $\widetilde{\mathbb{E}}_A[\cdot]\stackrel{\text{def}}{=}\widetilde{\mathbb{E}}[\cdot\,|\,h_A]$. Given A and a set of layer indices $I\subseteq[\Delta]$ so that $|A|+|I|\le d$, we define a sampling procedure that samples an edge from each layer E_i with $i\in I$ using pseudo-expectation $\widetilde{\mathbb{E}}_A$. We assume that the two algorithms below have access to graph G and pseudo-expectation $\widetilde{\mathbb{E}}$.

Algorithm 2 Edge sampling procedure.

Input: a subset of layer indices $I\subseteq[\Delta]$ and a subset A of edges.
Output: one edge from every layer E_i with $i\in I$.
$R=\varnothing$
for all $i\in I$ **do**
    Sample $e\in E_i$ with probability of choosing e equal to $\widetilde{\mathbb{E}}_{A\cup R}[x_e]$
    $R=R\cup\{e\}$
end for
return R

We say that we sample edges $e_1, \ldots, e_k$ in layers $i_1, \ldots, i_k$ conditioning on set A to mean that we run Algorithm 2 with parameters $I = \{i_1, \ldots, i_k\}$ and A.

Algorithm 3 Rounding algorithm for layered graphs.

Input: indices y and z of two edge layers ($1 \le y \le z \le \Delta$) and a subset of edges A.
Output: a path in $\hat{G}$ traversing layers E_y to E_z.
function FindPath(y, z, A)
  if $z - y + 1 \le a$ **then**
    Sample edges $e_0, \ldots, e_{z-y}$ in layers $E_y, E_{y+1}, \ldots, E_z$ conditioning on A.
    return the path formed by $e_0, \ldots, e_{z-y}$.
  end if
  Let $m_i = y + \lceil \frac{z-y}{a+1} \cdot i \rceil$ for $i \in [a]$.
  Sample edges $e_1, \ldots, e_a$ in layers $m_1, \ldots, m_a$ conditioning on A.
  Let $A' = A \cup \{e_i : i \in [a]\}$.
  Let $m_0 = z - 1$ and $m_{a+1} = y + 1$.
  for $i = 0$ **to** a **do**
    $P_i = \text{FindPath}(m_i + 1, m_{i+1} - 1, A')$ unless $m_i + 1 > m_{i+1} - 1$ then $P_i = \varnothing$
  end for
  Let P be the path formed by $P_0, e_1, P_1, e_2, \ldots, e_a, P_a$.
  return P
end function

To solve the problem, we run Algorithm 3 with $y = 1$, $z = \Delta$, and $A = \varnothing$. The analysis of this algorithm is quite similar to that of Algorithm 1 for series-parallel graphs, with the main difference that instead of the series-parallel decomposition of G, we consider the recursion tree whose nodes correspond to invocations of FindPath. The full analysis of this algorithm can be found in the full version of this paper [24], where the following theorems are shown.

▶ **Theorem 28.** *Algorithm 1 gives an $O(cp \log^{1-1/p} \Delta)$ approximation for the ℓ_p-Shortest Path problem in layered graphs with Δ layers in time $m^{O(p+ce^{1/c} \log \Delta)}$ for $c \in (0, 1/2)$.*

As a corollary, we get the following result for arbitrary graphs.

▶ **Theorem 29.** *There is an $O(cp \log^{1-1/p} n)$ approximation algorithm for the ℓ_p-Shortest Path problem in arbitrary graphs that runs in time $m^{O(p+ce^{1/c} \log n)}$ for $c \in (0, 1/2)$.*

References

1 Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Approximation of min–max and min–max regret versions of some combinatorial optimization problems. *European Journal of Operational Research*, 179(2):281–290, 2007. doi:10.1016/j.ejor.2006.03.023.

2 Boaz Barak, Jonathan A. Kelner, and David Steurer. Rounding sum-of-squares relaxations. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 31–40. ACM, 2014. doi:10.1145/2591796.2591886.

3 Vittorio Bilò, Ioannis Caragiannis, Angelo Fanelli, Michele Flammini, and Gianpiero Monaco. Simple greedy algorithms for fundamental multidimensional graph problems. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 125:1–125:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.125.

4 Thomas Breugem, Twan Dollevoet, and Wilco van den Heuvel. Analysis of fptases for the multi-objective shortest path problem. *Comput. Oper. Res.*, 78:44–58, 2017. doi:10.1016/j.cor.2016.06.022.

5 Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. *Journal of Algorithms*, 33(1):73–91, 1999. doi:10.1006/jagm.1999.1042.

6 Moses Charikar, Chandra Chekuri, Ashish Goel, and Sudipto Guha. Rounding via trees: deterministic approximation algorithms for group steiner trees and k-median. In *Proceedings of the Symposium on Theory of Computing*, pages 114–123, 1998. doi:10.1145/276698.276719.

7 Chandra Chekuri and Martin Pál. A recursive greedy algorithm for walks in directed graphs. In *Proceedings of the Symposium on Foundations of Computer Science*, pages 245–253, 2005. doi:10.1109/SFCS.2005.9.

8 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *Proceedings of the Symposium on Foundations of Computer Science*, pages 575–584, 2010. doi:10.1109/FOCS.2010.60.

9 Julia Chuzhoy and Sanjeev Khanna. Hardness of directed routing with congestion. *Electron. Colloquium Comput. Complex.*, TR06-109(109), 2006. arXiv:TR06-109.

10 Jesús de la Cal and Javier Cárcamo. Set partitions and moments of random variables. *Journal of mathematical analysis and applications*, 378(1):16–22, 2011.

11 Matthias Ehrgott. *Multicriteria Optimization (2. ed.)*, volume 491. Springer, 2005. doi:10.1007/3-540-27659-9.

12 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the Symposium on Theory of Computing*, pages 448–455, 2003. doi:10.1145/780542.780608.

13 Noah Fleming, Pravesh Kothari, and Toniann Pitassi. Semialgebraic proofs and efficient algorithm design. *Foundations and Trends in Theoretical Computer Science*, 14(1-2):1–221, 2019. doi:10.1561/0400000086.

14 Naveen Garg, Goran Konjevod, and Ramamoorthi Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. *Journal of Algorithms*, 37(1):66–84, 2000. doi:10.1006/jagm.2000.1096.

15 Horst W Hamacher and Günter Ruhe. On spanning tree problems with multiple objectives. *Annals of Operations Research*, 52(4):209–230, 1994. doi:10.1007/BF02032304.

16 Pierre Hansen. Bicriterion path problems. In *Multiple Criteria Decision Making Theory and Application: Proceedings of the Third Conference Hagen/Königswinter, West Germany, August 20–24, 1979*, pages 109–127. Springer, 1980.

17 Jovan Karamata. Sur une inégalité relative aux fonctions convexes. *Publications de l'Institut mathematique*, 1(1):145–147, 1932.

18 Adam Kasperski and Paweł Zieliński. On the approximability of minmax (regret) network optimization problems. *Information Processing Letters*, 109(5):262–266, 2009. doi:10.1016/j.ipl.2008.10.008.

19 Adam Kasperski and Paweł Zieliński. On the approximability of robust spanning tree problems. *Theoretical Computer Science*, 412(4-5):365–374, 2011. doi:10.1016/j.tcs.2010.10.006.

20 Adam Kasperski and Paweł Zieliński. Robust discrete optimization under discrete and interval uncertainty: A survey. *Robustness analysis in decision aiding, optimization, and analytics*, pages 113–143, 2016.

21 Adam Kasperski and Pawel Zielinski. Approximating some network problems with scenarios. *CoRR*, abs/1806.08936, 2018. doi:10.48550/arXiv.1806.08936.

22 Aditi Laddha, Mohit Singh, and Santosh S Vempala. Socially fair network design via iterative rounding. *Operations Research Letters*, 50(5):536–540, 2022. doi:10.1016/j.orl.2022.07.011.

23 Shi Li, Chenyang Xu, and Ruilong Zhang. Polylogarithmic approximation for robust s-t path. *CoRR*, abs/2305.16439, 2023. doi:10.48550/arXiv.2305.16439.

24 Yury Makarychev, Max Ovsiankin, and Erasmo Tani. Approximation algorithms for ℓ_p-shortest path and ℓ_p-group steiner tree, 2024. arXiv:2404.17669.

25 Ernesto Queiros Vieira Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2):236–245, 1984.

26 Robert Franklin Muirhead. Some methods applicable to identities and inequalities of symmetric algebraic functions of n letters. *Proceedings of the Edinburgh Mathematical Society*, 21:144–162, 1902.

27 Prasad Raghavendra and Benjamin Weitz. On the bit complexity of sum-of-squares proofs. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 80:1–80:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.80.

28 R. Ravi, Madhav V. Marathe, S. S. Ravi, Daniel J. Rosenkrantz, and Harry B. Hunt III. Many birds with one stone: multi-objective approximation algorithms. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 438–447. ACM, 1993. doi:10.1145/167088.167209.

29 Gabriele Reich and Peter Widmayer. Beyond steiner's problem: A vlsi oriented generalization. In *International Workshop on Graph-theoretic Concepts in Computer Science*, pages 196–210. Springer, 1989. doi:10.1007/3-540-52292-1_14.

30 Stefan Ruzika and Horst W Hamacher. A survey on multiple objective minimum spanning tree problems. In *Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation*, pages 104–116. Springer, 2009. doi:10.1007/978-3-642-02094-0_6.

31 Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.

32 Richard Stanley. *Enumerative Combinatorics: Volume 2*. Cambridge University Press, 2023.

Testing Spreading Behavior in Networks with Arbitrary Topologies

Augusto Modanese
Aalto University, Finland

Yuichi Yoshida
National Institute of Informatics, Tokyo, Japan

Abstract

Given the full topology of a network, how hard is it to test if it is evolving according to a local rule or is far from doing so? Inspired by the works of Goldreich and Ron (J. ACM, 2017) and Nakar and Ron (ICALP, 2021), we initiate the study of property testing in dynamic environments with arbitrary topologies. Our focus is on the simplest non-trivial rule that can be tested, which corresponds to the 1-BP rule of bootstrap percolation and models a simple spreading behavior: Every "infected" node stays infected forever, and each "healthy" node becomes infected if and only if it has at least one infected neighbor. Our results are subdivided into two main groups:

- If we are testing a single time step of evolution, then the query complexity is $O(\Delta/\varepsilon)$ or $\tilde{O}(\sqrt{n}/\varepsilon)$ (whichever is smaller), where Δ and n are the maximum degree of a node and the number of vertices in the underlying graph, respectively. We also give lower bounds for both one- and two-sided error testers that match our upper bounds up to $\Delta = o(\sqrt{n})$ and $\Delta = O(n^{1/3})$, respectively. If ε is constant, then the first of these also holds against adaptive testers.
- When testing the environment over T time steps, we have two algorithms that need $O(\Delta^{T-1}/\varepsilon T)$ and $\tilde{O}(|E|/\varepsilon T)$ queries, respectively, where E is the set of edges of the underlying graph.

All of our algorithms are one-sided error, and all of them are also non-adaptive, with the single exception of the more complex $\tilde{O}(\sqrt{n}/\varepsilon)$-query tester for the case $T = 2$.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Distributed computing models

Keywords and phrases Property testing, bootstrap percolation, local phenomena, expander graphs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.112

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2309.05442` [17]

Funding *Augusto Modanese*: Supported by the Helsinki Institute for Information Technology (HIIT). Part of this work done while affiliated with the Karlsruhe Institute of Technology (KIT) and visiting the NII as a JSPS International Research Fellow.
Yuichi Yoshida: Partly supported by JSPS KAKENHI Grant Number 18H05291 and 20H05965.

Acknowledgements We would like to thank Jukka Suomela for interesting discussions.

1 Introduction

Imagine we are observing the state of a network as it evolves over time. The network is static and we have complete knowledge about the connections; it is too large for us to keep track of the state of every single node, though nevertheless we are able to query nodes directly and learn their states. We might hypothesize that the global behavior can be explained by a certain local rule that is applied at every node, and we would like to verify if our hypothesis is correct or not. In this paper, we focus on this question: *How hard is it to test, given a local rule R, if the network is following R or is far from doing so?*

Following previous works [10, 18], we refer to the series of configurations assumed by the network over time as the *environment* ENV that we are observing. The network itself is static and its connections defined by a graph $G = (V, E)$. The local rule R is a map (admitting a finite description) from the states that a node observes in its neighborhood (including the node itself) to the new state it will assume in the next time step. Plausible scenarios that could be modeled in this context include not only rumor dissemination in social networks but also spreading of infectious diseases (where the connections between nodes represent proximity or contact between the organisms that we are observing). As is common in property testing [5], we assume that the bottleneck of this problem is keeping track of the states across the entire network, and hence we consider only the number of queries made by a testing algorithm as its measure of efficiency (and otherwise assume that the algorithm has access to unbounded computational resources).

1.1 Problem Setting

There exist two previous works [10, 18] that study the problem of determining whether ENV evolves according to R in the context of property testing (and, in the case of [10], also in the context of learning theory). In these works, the structure underlying ENV is always a cellular automaton (in the case of [18] one-dimensional, whereas [10] also considers automata of multiple dimensions), and thus ENV corresponds to the time-space diagram of such an automaton. This perspective is certainly meaningful when we are interested in phenomena that take place on a lattice or can be represented in such grid-like structures, for instance the movement of particles on a surface or across three-dimensional space. Nevertheless there are limits as to what can be modeled in this way. A prominent example are social networks, in which the connections hardly fit well into a regular lattice (even with several dimensions).

In this work we cast off these restraints and instead take the radically different approach of making no assumptions about the underlying structure or the space it is embedded in. Our only requirement is that it corresponds to a static graph G that is known to us in advance. This leaves a much broader avenue open when it comes to applications. In addition, the rule that we consider is effectively the simplest rule possible in such a setting that is not trivial. As we will see, despite the rule being very simple, it is rather challenging to fully determine the complexity of the problem. Indeed, compared to the previous works mentioned above, it might seem as if our progress is more modest; however, one should keep in mind that, in our case, the underlying network G has a much more rich structure (whereas in cellular automata we are dealing with a highly regular one).

The rule that we study is the 1-BP rule of *bootstrap percolation* [11, 20, 13, 2]. For $\tau \in \mathbb{N}_0$, the rule τ-BP is defined based on two states, *black* and *white*, as follows: If a node is black, then it always remains black; if a node is white, then it turns black if and only if it has at least τ black neighbors. These rules were originally inspired in the behavior observed in certain materials, and they are very naturally suited for modeling spreading phenomena.

Seen from the lenses of property testing, testing for the 1-BP rule in a sense resembles monotonicity testing [7]. Although we cannot directly apply one strategy to the other, if we view black as 1 and white as 0, then in both cases we have a violation whenever we see a 1 preceding a 0. The difference is that in 1-BP *every* 1 must arise from a preceding 1, whereas in the case of monotonicity we are happy if an isolated 0 spontaneously turns into a 1.

Another way of modeling the 1-BP rule is as a *constraint satisfaction problem* (CSP). CSPs have been studied in the context of property testing to some extent [4, 6]. We can characterize 1-BP by two constraints: A black node in step t implies every one of its neighbors is also black in step $t + 1$; meanwhile, a node is white in step $t + 1$ if and only if every one of

its neighbors in step t was white. Then we can recast testing if ENV follows 1-BP as testing if ENV is a satisfying assignment for these constraints. Nevertheless, although this seems to be a useful rephrasing of the problem, the current methods in CSPs in the context of property testing are not sufficient to tackle it. And, even if we could indeed test either constraint with a sublinear number of queries, ENV being close to satisfying both constraints would not necessarily imply that ENV is close to satisfying their intersection.

1.2 Results and Techniques

We now present our results and the methods used to obtain them. As this is a high-level discussion, formal definitions are postponed to Section 2, which the reader is invited to consult as needed.

The relevant parameters for the results are the number of nodes n in the graph $G = (V, E)$, the number of steps T during which the environment ENV evolves, the maximum degree Δ of G, and the accuracy parameter $\varepsilon > 0$. The size of the environment is nT, which is the baseline for linear complexity in this context (instead of n). We write ENV $\in$ 1-BP to indicate that ENV follows the 1-BP rule and $\mathrm{dist}(\texttt{ENV}, \text{1-BP}) \geq \varepsilon$ when it is ε-far from doing so, that is, one must flip at least εnT colors in ENV in order for 1-BP to be obeyed everywhere. (As already mentioned, see Section 2 for the precise definitions.)

In most cases we will be interested in optimizing the dependency of the query complexity on Δ. This is due to the fact that, intuitively, graphs with small Δ should be easier to verify locally, that is, by looking only at each node's neighborhood. (Indeed, this is the strategy followed by the first algorithm we present below in Theorem 1.)

Another desirable property that we wish our algorithms to have is *non-adaptiveness*; that is, the algorithm first produces a list of queries (without looking at the input), gathers their results, and then decides whether to accept or not. This is in contrast to an *adaptive* algorithm, which may perform later queries based on the answers it has seen so far. A third property "in-between" these is *time-conformability*, meaning that the algorithm does not make queries in step t if it has already queried nodes at some later step $t' > t$. It is easy to see that the existence of a non-adaptive algorithm with query complexity q implies a time-conforming algorithm with the same complexity: Just gather the q queries in a list, sort them according to the time step queried, and then execute the queries in order. The converse is not true in general, however, since a time-conforming algorithm might choose its queries in a later time step based on what it has seen beforehand (or in the same time step, even).

Recalling our motivation of testing the evolution of huge networks, we see that non-adaptive algorithms are the most desirable because the queries may all be performed in parallel (at each time step). In case this cannot be achieved, an adaptive, time-conforming algorithm is still satisfactory, even though it might require "freezing" the network at a specific time step (so that the algorithm has time to gather the results received and decide on the next queries to make on the same time step). Adaptive algorithms that violate time-conformability are not particularly desirable since they require "rewinding" the state of the network back in time. Nevertheless, depending on the nodes' capabilities, there might still be strategies to cope with this; for example, if T is small, it is plausible to have nodes can cache their state in previous steps (and thus answer any of the algorithm's queries, even about previous states).

In this paper, we study two different settings: testing a single time step of evolution ($T = 2$) and testing multiple steps ($T > 2$). In the first case we prove both upper and lower bounds, which also match up to certain values of Δ. In the second we show only upper bounds, but which suffice to demonstrate that the problem admits non-trivial testers, at least for moderate (non-constant) values of Δ.

1.2.1 The Case $T = 2$

Let us first discuss our results for the case where $T = 2$. In this case there is a natural graph-theoretical rephrasing of the problem: For $t \in \{1, 2\}$, let S_t be the set corresponding to $\texttt{ENV}(\cdot, t)$ where we see $\texttt{ENV}(\cdot, t)$ as an indicator function (i.e,. S_t is exactly the set of nodes $v \in V$ for which $\texttt{ENV}(v, t) = 1$). Then $\texttt{ENV} \in$ 1-BP if and only if S_2 is dominated by S_1 (in graph-theoretic terms).[1] From this perspective, the distance from ENV to 1-BP is the (relative) total number of nodes we need to add or remove from either of S_1 or S_2 in order for the domination property to be satisfied.

The hardness of the problem in this case is highly dependent on the maximum degree Δ. Our first result is that there is a very natural and simple algorithm that achieves query complexity $O(\Delta/\varepsilon)$.

▶ **Theorem 1.** *Let $T = 2$ and $\varepsilon > 0$. There is a* non-adaptive, one-sided error *algorithm with query complexity $O(\Delta/\varepsilon)$ that decides whether* $\texttt{ENV} \in$ 1-BP *or* $\mathrm{dist}(\texttt{ENV}, \text{1-BP}) \geq \varepsilon$.

The algorithm simply selects nodes at random and then queries their entire neighborhoods in both time steps. Since we are dealing with a local rule, this is sufficient to detect if ENV contains too many violations of the rule or not. One detail that needs care here is that, in general, our notion of distance does *not* match the number of violations of the rule. Nevertheless, as we show, the cases where it does not are only playing in our favor, and so this strategy always succeeds.

It turns out that this algorithm is optimal when we are in regimes where there is a constant $b \geq 2$ such that $\varepsilon = \Omega(\Delta^b/n)$. We also prove lower bounds for the case where $b \geq 1$, which are especially useful in regimes where Δ is larger than $\sqrt{n}$.

▶ **Theorem 2.** *There is a constant $\varepsilon_0 > 0$ such that the following holds: Let $\varepsilon = \Omega(\Delta^b/n)$ be given where $b \geq 1$ is constant, and let $\varepsilon \leq \varepsilon_0$. Then deciding if* $\texttt{ENV} \in$ 1-BP *or is ε-far from* 1-BP *with a* one-sided error *tester requires at least q queries in general, where:*

1. *If $b > 2$, then $q = \Omega(\Delta/\varepsilon)$ if the tester is non-adaptive or $q = \Omega(1/\varepsilon + \Delta)$ if it is adaptive.*
2. *If $b = 2$, then $q = \Omega(\Delta/\varepsilon \log \Delta)$ if the tester is non-adaptive or $q = \Omega(1/\varepsilon + \Delta/\log \Delta)$ if it is adaptive.*
3. *If $1 \leq b < 2$, then $q = \Omega(\Delta^{b-1}/\varepsilon)$ if the tester is non-adaptive or $q = \Omega(1/\varepsilon + \Delta^{b-1})$ if it is adaptive.*

Note the lower bounds hold even for adaptive testers in general, even for those that do not respect time-conformability.

If we are only interested in the regime where ε is constant, then setting $b = \log_\Delta n$ above we obtain a lower bound of $\Omega(\Delta)$ whenever $\Delta = O(n^{1/2-c})$ for a constant $c > 0$. This is matched by the upper bound of Theorem 1. For $\Delta = \Theta(\sqrt{n})$, the lower bound is $\Omega(\Delta/\log n)$; for larger Δ the lower bound becomes $\Omega(n/\Delta)$ and thus deteriorates as Δ increases.

The lower bound is based on an adequate construction of expander graphs. More specifically, the expanders we construct are bipartite, Δ-regular, and have *distinct* expansion guarantees for sets of nodes on either side. This is needed because the expansion in one direction guarantees ε-farness whereas the one in the other direction yields the actual lower bound on the number of queries that a correct algorithm must make.

[1] Technically the definition of domination is so that A dominates B if and only if $B \subseteq A \cup N(A)$. Using this definition the equivalence is only true if the graph G contains self-loops everywhere. (Nevertheless, adding self-loops everywhere does not impact the maximum degree, which is the relevant parameter here.) The equivalence is certainly true if we change the definition so that A dominates B if $B \subseteq N(A)$.

The hard instances themselves are simple: We color a moderately large set B of randomly chosen nodes black in the second time step and leave the rest colored white. The intuition is that, since B is chosen at random, it will not match nicely with a cover $C = \bigcup_{u \in S} N(u)$ induced by some set of nodes S in the first time step; that is, the symmetric difference between B and C will likely be large, giving us ε-farness. At the same time, since a one-sided error algorithm A cannot reject good instances, it is hard for it to detect that there is something wrong with B without having to "cover" a considerable number of nodes in either step. Indeed, in order to ascertain that a node $v \in B$ is incorrect, A must verify that there is no black node in $N(v)$ in the first step; if the existence of some black $u \in N(v)$ is compatible with its view, then there is no contradiction to v being black, and hence A cannot reject. Querying all of $N(v)$ requires $\Omega(\Delta)$ queries, but it is also possible for A to determine the colors *indirectly* by querying neighbors of nodes in $N(v)$. To obtain the lower bound we show that this other strategy also requires too many queries – although it might not be as inefficient when Δ is large (thus explaining why we get a weaker result in that case).

For two-sided error algorithms, we are able to prove similar, though slightly more modest lower bounds. These are also based on expander graphs but require a more complex set of instances for the argument to go through.

▶ **Theorem 3.** *There are constants $\varepsilon_0, \zeta > 0$ such that, for any $0 < \varepsilon \leq \varepsilon_0$ with $\varepsilon \geq \zeta \Delta^b / n$ where $b \geq 1$ is constant, deciding if* ENV $\in$ 1-BP *or is ε-far from* 1-BP *with a* non-adaptive, two-sided error *tester requires q queries in general, where:*

- *If $b > 3$, then $q = \Omega(\Delta/\varepsilon)$.*
- *If $b = 3$, then $q = \Omega(\Delta/\varepsilon \log \Delta)$.*
- *If $1 \leq b < 3$, then $q = \Omega(\Delta^{(b-1)/2}/\varepsilon)$.*

Again focusing on the regime where ε is constant, we now obtain $\Omega(\Delta)$ as the lower bound for regimes where $\Delta = O(n^{1/3-c})$ for a constant $c > 0$ or also $\tilde{\Omega}(\Delta)$ when $\Delta = \Theta(n^{1/3})$. Hence, given the algorithm of Theorem 1, up to $\Delta = \Theta(n^{1/3})$ there is essentially *no advantage* for two-sided error algorithms compared to one-sided error ones. For larger values of Δ, the lower bound is $\Omega(\sqrt{n/\Delta})$ and again deteriorates as Δ increases.

Since we are dealing with two-sided error algorithms, we apply Yao's minimax principle, and we now generate instances according to two different distributions D_Y and D_N where D_Y follows 1-BP whereas D_N generates instances that are (with high probability) far from doing so. The point is that we can show that it is hard to distinguish between D_Y and D_N without making a considerable number of queries. The distributions are such that, in both cases, we pick a set S of $\Theta(\varepsilon n/\Delta)$ vertices in the first time step uniformly at random. Then we color S and $N(S)$ black in D_Y (and leave the remaining nodes white) while in D_N we color only *a (constant) fraction* of $N(v)$ for $v \in S$ black. (We must also offset the fact that nodes in the second step in D_N are colored black with less probability by using a larger S when generating D_N instances.) By the expansion guarantees, this then gives us ε-farness of the instances in D_N. Observe that in this setting it is meaningless to query nodes in the first step since only a small fraction of them can ever be black; hence we need only deal with a set Q of nodes that are queried in the second step. The indistinguishability of D_Y and D_N follows from using the expansion guarantee from nodes in the second step to those in the first one. The argument is that, unless the set Q of queried nodes is large, almost all neighbors of Q are in fact *unique neighbors* and, moreover, it is impossible to distinguish D_Y from D_N if the set S only intersects the unique neighbors of Q. (That is, one can only distinguish D_Y and D_N if one queries *two* distinct neighbors $u, u' \in N(v)$ of some $v \in S$; due to the expansion guarantees, this requires a large number of queries.)

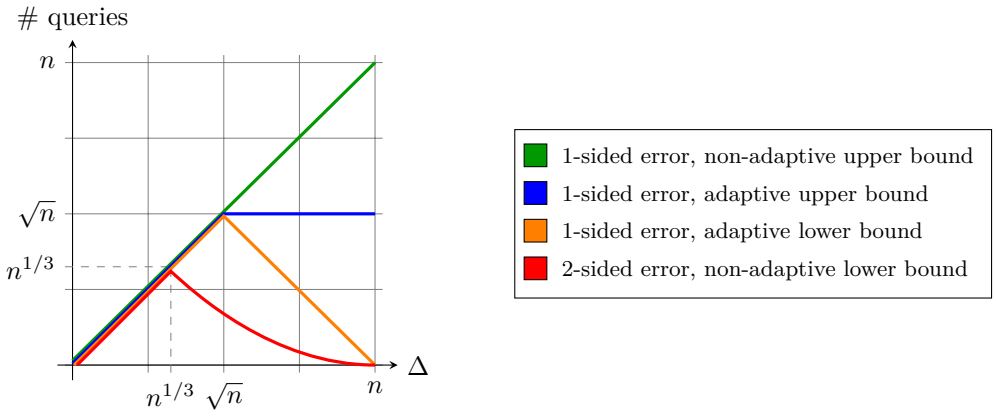


Figure 1 Summary of results for the case $T = 2$ and constant ε, ignoring logarithmic factors.

In light of these lower bounds, looking back at the algorithm of Theorem 1 we realize that its single weakness is that it does not perform well when Δ is large. Unfortunately our lower bounds do not say as much in that case, and thus a wide gap is left between lower and upper bounds in that regime. Nevertheless, we can narrow this gap by using a more complex strategy – if we are prepared to let go of non-adaptiveness and time-conformability (though we can still obtain a one-sided error algorithm). As previously discussed, this is not such a large limitation when taking possible applications into account (as when $T = 2$ it is plausible to, e.g., require nodes to cache their previous state) and indeed it is offset by the significant reduction in the query complexity.

▶ **Theorem 4.** *Let $T = 2$ and let $\varepsilon > 0$ be given. There is an* adaptive, one-sided error *algorithm for testing whether* ENV $\in$ 1-BP *or is ε-far from* 1-BP *with query complexity* $O(\sqrt{n}\log^{3/2}(n)/\varepsilon)$.

The algorithm achieving this is much more complex than that of Theorem 1. Indeed, it must decide whether ENV $\in$ 1-BP or not *without being able to query the entire neighborhood of any node*. To achieve this, we use a "filtering" process in which we first try to infer the color (assuming ENV $\in$ 1-BP) of as many nodes as we can (in either step) by querying some of their neighbors indirectly. Since we are certain of which color these nodes must have, we can verify these separately using a small number of random queries. Making careful observations, we then realize that we can simply ignore these nodes afterwards and thus reduce the degree of most of the remaining nodes to $\tilde{O}(\sqrt{n})$. This allows us to essentially fall back to a strategy as in the algorithm of Theorem 1, though a particular corner case requires special attention.

The results for $T = 2$ and the regime where ε is constant are summarized in Figure 1.

1.2.2 Case of General T

Let us now discuss the case $T > 2$. Here we obtain a couple of upper bounds that show that the problem admits testing algorithms with sublinear query complexity, at least in a few regimes of interest. We present two algorithms that complement each other.

A quick observation shows the problem becomes essentially trivial when $T \geq 2\,\mathrm{diam}(G)/\varepsilon$. (In a nutshell, this is because otherwise ENV $\in$ 1-BP reaches a fixed point well before T, and thus most configurations of ENV must all be this one fixed point.) Hence for this discussion it should be kept in mind that the problem is only interesting when $\mathrm{diam}(G)$ is non-trivial and $T = o(\mathrm{diam}(G)/\varepsilon)$. Furthermore, recall that, since ENV has nT entries, the benchmark for a non-trivial testing algorithm is not $o(n)$ but $o(nT)$.

The first algorithm we present is a direct generalization of the one from Theorem 1.

▶ **Theorem 5.** *Let $\varepsilon > 0$ and $T > 2$. There is a* non-adaptive, one-sided error *algorithm that performs $O(\Delta^{T-1}/\varepsilon T)$ queries and decides if* $\texttt{ENV} \in \textsf{1-BP}$ *or* $\mathrm{dist}(\texttt{ENV}, \textsf{1-BP}) \geq \varepsilon$.

The algorithm is only useful in settings where, say, $T = O(\log_\Delta n)$. Nevertheless, it is relatively simple to obtain and outperforms our more complex algorithm in certain regimes.

▶ **Theorem 6.** *Let $\varepsilon > 0$ and $T \geq 4/\varepsilon$. Then there is a* non-adaptive, one-sided error *algorithm with query complexity $O(|E| \log(n)/\varepsilon T)$ that decides whether* $\texttt{ENV} \in \textsf{1-BP}$ *or* $\mathrm{dist}(\texttt{ENV}, \textsf{1-BP}) \geq \varepsilon$. *In addition, if G excludes a fixed minor H (which includes the case where G is planar or, more generally, G has bounded genus), then $O(|E|/\varepsilon T)$ queries suffice.*

To better judge what this algorithm achieves, let us suppose that the underlying graph is Δ-regular, in which case $|E| = n\Delta$. Then this gives a non-trivial testing algorithm whenever $T = \omega(\sqrt{(\Delta/\varepsilon)\log n})$ (or $T = \omega(\sqrt{\Delta/\varepsilon})$ if we also assume G is planar). Hence, together with Theorem 5, we obtain non-trivial testing algorithms in the regime where $\Delta = o(\log n)$ (or even $\Delta = o(\log^2 n)$ in planar graphs) and *for all values of T*.

The algorithm of Theorem 6 combines some ideas from the work of Nakar and Ron [18] with *graph decompositions*. A graph decomposition is a set C of edges which cuts the graph into components pairwise disjoint components $V_1, \ldots, V_r$ of small diameter. In our case the appropriate choice of diameter will be $d = O(\varepsilon T)$. The basic approach is to query the endpoints of C after d steps have elapsed and then use this view to predict the colors of every node in the graph in the subsequent steps. As we show, the view actually suffices to predict all but at most an $O(\varepsilon)$ fraction of $\texttt{ENV}$ (and hence we need only query the predicted values using $O(1/\varepsilon)$ independent queries to check if $\texttt{ENV}$ is following $\textsf{1-BP}$ or not). We refer to Appendix A.2 for a more in-depth description of the strategy and the ideas involved.

1.3 Open Problems

Since this work is but a first step in an unexplored direction, several questions remain open:

- *The case $T = 2$ and large Δ.* The algorithm of Theorem 1 is essentially optimal up to $\Delta = O(\sqrt{n})$ (if we consider only one-sided error algorithms), but for larger values of Δ the best we have is the $\tilde{O}(\sqrt{n})$-query algorithm of Theorem 4. Can we reduce this, say, to $\tilde{O}(\Delta^{b-1}/\varepsilon)$ for $\varepsilon = \Omega(\Delta^b/n)$ so as to match the lower bound of Theorem 2? Is it really necessary to give up time-conformity in order to do better than $O(\Delta/\varepsilon)$ in this setting? In addition, improving our lower bounds in the case of (both one- and two-sided error) adaptive algorithms seems well within reach.
- *The case $T > 2$.* Our results show that, in this case, we can get non-trivial algorithms for graphs of small degree (e.g., $\Delta = o(\log n)$). Due to the difficulties in the case $T = 2$, larger values of Δ pose additional challenges. In this sense a first step in this direction would be to port the lower bounds from the $T = 2$ case. Nevertheless, it is not immediately clear how to do so since ε-farness there is even harder to achieve given the cascading effects that might occur over multiple time steps (see in particular Lemma 17).
- *Testing other rules.* Finally, from a broader perspective it would also be meaningful to consider other rules than $\textsf{1-BP}$. Of course, by inverting the roles of 0 and 1, all of our results also hold for AND rule (i.e., a node becomes a 1 if and only if all its neighbors are 1; otherwise it becomes a 0). Some very natural rules to consider next are, for instance, τ-$\textsf{BP}$ or the majority rule. There has been extensive study of these rules in other contexts [11, 20, 13, 16, 21, 2, 12, 8, 14], and so there is solid ground to build on there.

1.4 Paper Overview

The rest of the paper is structured as follows: In Section 2 we introduce basic notation, review some standard graph-theoretic results, and formally specify the model and problem we study. For the case $T = 2$, in Section 3 we address the two algorithms (Theorems 1 and 4); the two lower bounds (Theorems 2 and 3) are covered in the full version of the paper [17]. Finally in Appendix A we discuss the two algorithms for the case $T > 2$ (Theorems 5 and 6).

2 Preliminaries

The set of non-negative integers is denoted by $\mathbb{N}_0$ and that of strictly positive integers by $\mathbb{N}_+$. For $n \in \mathbb{N}_+$, we write $[n] = \{i \in \mathbb{N}_+ \mid i \leq n\}$ for the set of the first n positive integers. Without ambiguity, for a statement S, we write $[S]$ for the indicator variable of S (i.e., $[S] = 1$ if S holds; otherwise, $[S] = 0$).

An event is said to occur with high probability if it occurs with probability $1 - o(1)$. For a set X, we write U_X to denote a random variable that takes on values from X following the uniform distribution on X. We assume the reader is familiar with basic notions of discrete probability theory (e.g., Markov's inequality and the union bound). We will use the following version of the Chernoff bound (see, e.g., [9, 19]):

▶ **Theorem 7** (Chernoff bound). *Let $n \in \mathbb{N}_+$ and $\varepsilon > 0$, and let $X_1, \ldots, X_n$ be independent and identically distributed random variables taking values in the interval $[0, 1]$. Then, for $X = (\sum_{i=1}^{n} X_i)/n$ and $\mu = \mathsf{E}[X]$, $\Pr[|X - \mu| > \varepsilon] < 2e^{-n\varepsilon^2/3\mu}$.*

2.1 Graph Theory

We consider only undirected graphs. Except when explicitly written otherwise, we always write just "graph" for a simple graph, though self-loops are allowed.

Let $G = (V, E)$ be a graph. For $S \subseteq V$, $G[S]$ denotes the subgraph of G induced by S. For two nodes $u, v \in V$, $\mathrm{dist}_G(u, v)$ is the length of the shortest path between u and v; we drop the subscript if G is clear from the context. The *diameter* $\mathrm{diam}(G)$ of G is the maximum length among all shortest paths between any pair of vertices $u, v \in V$, that is, $\mathrm{diam}(G) = \max_{u,v \in V} \mathrm{dist}(u, v)$. This notion extends to any $V' \subseteq V$ by considering only pairs of vertices in V', that is, $\mathrm{diam}(V') = \max_{u,v \in V'} \mathrm{dist}(u, v)$.We write $\delta(G)$ for the minimum degree of G and $\Delta(G)$ for the maximum one. If G is clear from the context, we simply write δ and Δ, respectively. If $\delta = \Delta$, then G is Δ-*regular*.

For a node $v \in V$, $N(v) = \{u \in V \mid uv \in E\}$ denotes the set of *neighbors* of v. Generalizing this notation, for a set $S \subseteq V$ we write $N(S)$ for the union $\bigcup_{v \in S} N(v)$. A vertex $u \in V$ is said to be a *unique neighbor* of S if there is a *unique* $s \in S$ such that $us \in E$. When S is clear from the context, we also refer to a unique neighbor of $v \in S$ as a node $u \in V$ for which $u \in N(v')$ if and only if $v' \notin S$ or $v' = v$.

A graph $G = (V, E)$ is *bipartite* if $V = L \cup R$ for disjoint sets L and R and any edge has exactly one endpoint in L and one in R. In this context, we refer to the nodes of L as *left-* and to those of R as *right-vertices*. Additionally, the graph is *balanced* if $|L| = |R|$.

The following is a spin-off of a well-known result on the size of the dominating set of a graph (see, e.g., [1]):

▶ **Lemma 8** (Cover from minimum degree). *Let $G = (V, E)$ be a bipartite graph where each right-vertex has degree at least δ. Then there is a set D of $n \log(n)/\delta$ left-vertices such that every right-vertex has a neighbor in D.*

Proof. We use the probabilistic method. Fix a right-vertex $v \in V$. If we pick a set D of $m = n\log(n)/\delta$ left-vertices uniformly at random, then the probability that $N(v) \cap D$ is empty is at most $(1-\delta/n)^m < e^{-\log n} < 1/n$. Hence, by the union bound, there is a non-zero probability that D is such that $N(v) \cap D$ is non-empty for every right-vertex v. ◀

2.2 Model and Problem Definition

We use the standard query model of property testing [5]. The testing algorithm has unlimited computational power and access to a source of infinitely many random bits that are fully independent from one another. In addition, the model has full knowledge of the underlying topology of the network, which is presented as a graph $G = (V, E)$ with $|V| = n$ nodes. We assume there are no singleton nodes (i.e., every node is such that there is an edge incident to it). The topology remains fixed during the evolution of the network, whose nodes take on different states over a set of discrete time steps. As in the previous works [18, 10], the formal object we are testing is an *environment* $\texttt{ENV}\colon V \times [T] \to Z$ where $T \geq 2$ and Z is the set of states that each node may assume.

The goal is to detect whether $\texttt{ENV}$ is following a certain *local rule* ρ, which is defined as a function that maps every multiset μ over Z to $\rho(\mu) \in Z$. The environment $\texttt{ENV}$ is said to *follow* ρ if, for every time step $t \leq T$ and every node $v \in V$, we have that $\texttt{ENV}(v, t+1) = \rho(\texttt{ENV}(N(v), t))$ (where $\texttt{ENV}(N(v), t)$ here is seen as a multiset, that is, counting multiplicities of the occurrence of each element of Z). Blurring the distinction between ρ and the set of environments that follow it, we write $\texttt{ENV} \in \rho$ if $\texttt{ENV}$ follows ρ.

The *distance* between two environments $\texttt{ENV}, \texttt{ENV}'\colon V \times [T] \to Z$ is the (normalized) number of pairs on which $\texttt{ENV}$ and $\texttt{ENV}'$ differ:

$$\operatorname{dist}(\texttt{ENV}, \texttt{ENV}') = \frac{1}{nT} \sum_{(v,t) \in V \times [T]} [\texttt{ENV}(v,t) \neq \texttt{ENV}'(v,t)].$$

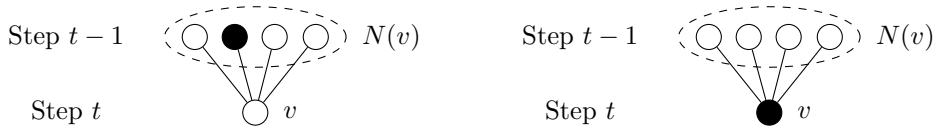
For a set of environments X (all over the same domain $V \times [t]$), we write $\operatorname{dist}(\texttt{ENV}, X) = \min_{\texttt{ENV}' \in X} \operatorname{dist}(\texttt{ENV}, \texttt{ENV}')$ for the minimum distance between $\texttt{ENV}$ and X. Being a bit sloppy, we write $\operatorname{dist}(\texttt{ENV}, \rho)$ for the minimum distance from $\texttt{ENV}$ to the set of environments $\texttt{ENV}'$ for which $\texttt{ENV}' \in \rho$. For $\varepsilon \geq 0$, $\texttt{ENV}$ is *ε-far* from ρ if $\operatorname{dist}(\texttt{ENV}, \rho) \geq \varepsilon$; otherwise it is *$\varepsilon$-near* ρ.

In this work, we focus on $Z = \{0, 1\}$ and on testing the 1-BP rule of bootstrap percolation. The rule is defined by $\rho(\mu) = [1 \in \mu]$ (i.e., $\rho(\mu) = 1$ if $1 \in \mu$ and $\rho(\mu) = 0$ otherwise). Seeing states as colors, we identify state 1 with the color *black* and state 0 with *white*.[2]

For $t \geq 2$, a pair (v, t) is a *successor* of $(u, t-1)$ if there is an edge between v and u; at the same time, $(u, t-1)$ is a *predecessor* of (v, t). If the respective time steps t and $t-1$ are clear from the context, we might also drop any mention of them and simply say that v (as a node) is a successor of u. This is particularly convenient when analyzing the case $T = 2$.

Testing algorithms. Fix $\varepsilon > 0$. A *testing algorithm* A for 1-BP accesses $\texttt{ENV}\colon V \times [T] \to Z$ by means of *queries*, which are pairs $(v, t) \in V \times [T]$. Upon querying the pair (v, t), A receives $\texttt{ENV}(v, t)$ as answer. If the queries are performed in an order where, for every t and t' with $t' > t$, A never makes a $(\cdot, t)$ query after it has queried $(\cdot, t')$, then A is said to be *time-conforming*. As usual in property testing, our interest lies in the *query complexity* of A, that is, the maximum number of queries that A makes, regardless of its randomness.

[2] Being pedantic, the 1-BP rule in the context of bootstrap percolation is such that a black node always remains black. This behavior can be enforced in the model we describe by adding self-loops to all nodes.



(a) Type I **(b)** Type II

Figure 2 Violations can be of two different types. Here we see a node v and its state in time step t (as a color) as well as its neighbors $N(v)$ and their respective states in step $t-1$.

The algorithm A is a *one-sided error tester* for $\texttt{ENV} \in$ 1-BP if the following holds, where the probabilities are taken over the randomness of A:

- If $\texttt{ENV} \in$ 1-BP, then always $A(\texttt{ENV}) = 1$.
- If $\texttt{ENV}$ is ε-far from 1-BP, then $\Pr[A(\texttt{ENV}) = 1] < 1/2$.

In contrast, A is a *two-sided error tester* if it may also err on $\texttt{ENV} \in$ 1-BP:

- If $\texttt{ENV} \in$ 1-BP, then $\Pr[A(\texttt{ENV}) = 1] \geq 2/3$.
- If $\texttt{ENV}$ is ε-far from 1-BP, then $\Pr[A(\texttt{ENV}) = 1] < 1/3$.

Violations. Observe that our notion of distance is *not* the same as counting the number of failures of $\texttt{ENV}$ in following 1-BP. There are two kinds of failures that may occur:

▶ **Definition 9** (Violations). *A pair* $(v,t) \in V \times [T]$ *is* violating *if* $t \geq 2$ *and one of the following conditions hold:*

(I) $\texttt{ENV}(v,t) = 0$ *and* $\exists u \in N(v) : \texttt{ENV}(u, t-1) = 1$

(II) $\texttt{ENV}(v,t) = 1$ *and* $\forall u \in N(v) : \texttt{ENV}(u, t-1) = 0$

We refer to these violations as violations of type I and II*, respectively. We write* viol($\texttt{ENV}$) *for the set of violating pairs in* $\texttt{ENV}$.

Although a larger distance to 1-BP implies a greater number of violations, there is not an exact correspondence between the two. For example, it might be the case that $\texttt{ENV}$ exhibits a great number of violations, but correcting them requires recoloring only a few nodes. We will prove upper and lower bounds between the distance and the number of violations further below (Lemmas 10 and 17).

3 Upper Bounds for the Case $T = 2$

In this section we present our two algorithms for the case where $T = 2$. The first of these (Section 3.1) is quite simple and has query complexity $O(\Delta/\varepsilon)$, which turns out to be optimal for the regimes where $\Delta = o(\sqrt{n})$. The second one (Section 3.2) is much more intricate and gives query complexity $\tilde{O}(\sqrt{n}/\varepsilon)$, which makes it more suitable for the regimes where $\Delta = \omega(\sqrt{n})$. Although both are one-sided error algorithms, the first algorithm is non-adaptive and thus time-conforming whereas the second has neither of these properties (i.e., it is adaptive and also does not respect time-conformity).

3.1 An Upper Bound that Scales with the Maximum Degree

In this section, we prove:

▶ **Theorem 1.** *Let* $T = 2$ *and* $\varepsilon > 0$*. There is a* non-adaptive, one-sided error *algorithm with query complexity* $O(\Delta/\varepsilon)$ *that decides whether* $\texttt{ENV} \in$ 1-BP *or* $\operatorname{dist}(\texttt{ENV}, \text{1-BP}) \geq \varepsilon$.

The claim is that Algorithm 1 satisfies the requirements of Theorem 1. As mentioned above, the strategy followed by Algorithm 1 is quite simple: It chooses a certain subset of nodes U uniformly at random and then queries the states of $u \in U$ and all of $N(u)$ in both time steps. The algorithm then rejects if and only if a violation of either type is detected.

Algorithm 1 Algorithm for the case $T = 2$ with query complexity $O(\Delta/\varepsilon)$.

```
Pick U ⊆ V uniformly at random where |U| = ⌈2/ε⌉;
Query ENV(v, 1) and ENV(u, 2) for every u ∈ U and v ∈ N(u) in a time-conforming
  manner;
for u ∈ U do
    if ENV(u, 2) = 0 and ∃v ∈ N(u) : ENV(v, 1) = 1 then reject;
    if ENV(u, 2) = 1 and ∀v ∈ N(u) : ENV(v, 1) = 0 then reject;
end
accept;
```

At the core of the correctness of Algorithm 1 is the relation between the number of violations and the distance of ENV to 1-BP. With a bit of care, we can relate the two quantities as shown next. (Actually for the correctness of Algorithm 1 we only need one of the two bounds below; the other one comes as a "bonus".)

▶ **Lemma 10.** *Let $T = 2$. Then*

$$\frac{|\mathrm{viol}(\mathtt{ENV})|}{2\Delta n} \leq \mathrm{dist}(\mathtt{ENV}, \text{1-BP}) \leq \frac{|\mathrm{viol}(\mathtt{ENV})|}{2n}.$$

Proof. Every violating pair (u, t) can be corrected by flipping the value of $\mathtt{ENV}(u, t)$, which does not create a new violating pair since $t = T = 2$. In addition, if ENV does not have any violating pair, then $\mathtt{ENV} \in$ 1-BP. This implies $\mathrm{dist}(\mathtt{ENV}, \text{1-BP}) \leq |\mathrm{viol}(\mathtt{ENV})|/2n$. On the other hand, flipping the color of a node can only correct at most Δ violating pairs. Hence we also have $\mathrm{dist}(\mathtt{ENV}, \text{1-BP}) \geq |\mathrm{viol}(\mathtt{ENV})|/2\Delta n$. ◀

The lemma directly implies that, if $\mathrm{dist}(\mathtt{ENV}, \text{1-BP}) \geq \varepsilon$, then $|\mathrm{viol}(\mathtt{ENV})| \geq 2\varepsilon n$. Hence the probability that Algorithm 1 errs in this case is

$$\Pr[(U, 2) \cap \mathrm{viol}(\mathtt{ENV}) = \varnothing] \leq (1 - 2\varepsilon)^{|U|} < \frac{1}{e} < \frac{1}{2}.$$

Since Algorithm 1 only rejects when a violation of either type is detected, Algorithm 1 always accepts if $\mathtt{ENV} \in$ 1-BP. The query complexity and other properties of Algorithm 1 are clear, and hence Theorem 1 follows.

3.2 An Upper Bound Independent of the Maximum Degree

Next we show our second algorithm, which is much more complex than Algorithm 1. Since Algorithm 1 is already optimal for $\Delta = O(\sqrt{n})$, we focus on the regime where $\Delta = \Omega(\sqrt{n})$ and present an algorithm with query complexity that is independent of Δ. The algorithm requires adaptiveness and unfortunately is no longer time-conforming; obtaining a time-conforming or even non-adaptive algorithm with the same query complexity for these large values of Δ (or proving none exists) remains an interesting open question.

▶ **Theorem 4.** *Let $T = 2$ and let $\varepsilon > 0$ be given. There is an* adaptive, one-sided error *algorithm for testing whether $\mathtt{ENV} \in$ 1-BP or is ε-far from 1-BP with query complexity $O(\sqrt{n}\log^{3/2}(n)/\varepsilon)$.*

We claim Algorithm 2 satisfies the requirements of the theorem. Next we give a brief description of the strategy followed by Algorithm 2.

Algorithm 2 Algorithm for the case $T = 2$ with query complexity $\tilde{O}(\sqrt{n}/\varepsilon)$.

1 Select $Q_1, Q_1', Q_2, Q_2' \subseteq V$ with $|Q_i| = |Q_i'| = (24/\varepsilon)\sqrt{n}\log^{3/2} n$ uniformly at random;
2 Query $\texttt{ENV}(Q_1, 1)$ and $\texttt{ENV}(Q_2', 1)$;
3 $B_2 \leftarrow \{v \in V \mid \exists u \in N(v) \cap Q_1 \colon \texttt{ENV}(u, 1) = 1\}$;
4 Query $\texttt{ENV}(Q_1', 2)$ and $\texttt{ENV}(Q_2, 2)$;
5 $W_1 \leftarrow \{v \in V \mid \exists u \in N(v) \cap Q_2 \colon \texttt{ENV}(u, 2) = 0\}$;
6 **if** $\exists u \in Q_1' \cap B_2 : \texttt{ENV}(u, 2) = 0$ *or* $\exists u \in Q_2' \cap W_1 : \texttt{ENV}(u, 1) = 1$ **then reject**;
7 $F \leftarrow \{v \in V \mid |N(v) \setminus W_1| \leq 4\sqrt{n \log n}\}$;
8 Select $Q_3 \subseteq F$ with $|Q_3| = (4/\varepsilon)\log n$ uniformly at random;
9 Query $\texttt{ENV}(v, 2)$ and $\texttt{ENV}(N(v) \setminus W_1, 1)$ for every $v \in Q_3$;
10 **if** $\exists v \in Q_3 : \texttt{ENV}(v, 2) = 0 \wedge \exists u \in N(v) \setminus W_1 : \texttt{ENV}(u, 1) = 1$ *or*
$\exists v \in Q_3 : \texttt{ENV}(v, 2) = 1 \wedge \nexists u \in N(v) \setminus W_1 : \texttt{ENV}(u, 1) = 1$ **then reject**;
11 **accept**;

Approach. The operation of Algorithm 2 can be divided into two parts. The first one is up to line 2. Here we query nodes from the first and second time steps at random (Q_1 and Q_2) and try to ascertain the color of as many nodes as possible using these queries. More specifically, if a node v has a neighbor $u \in N(v)$ which is black in the first step, then we know v must be black in the second step. We gather these nodes in the set B_2. A similar observation holds for the nodes in the set W_1, which must be white since they have a neighbor in the second step that is white. At the same time we query another set of nodes from the first and second step uniformly at random (Q_1' and Q_2') to verify that all but a very small fraction of nodes in W_1 (resp., B_2) are indeed white (resp., black).

The second part of the algorithm starts after line 2. Here we will ignore nodes in W_1 (since we already know they are white) and "filter" nodes that have not too large degree to nodes not in W_1. These nodes are added to the set F. Intuitively we can then test these nodes in the same fashion as Algorithm 1: We select a few nodes $v \in F$ uniformly at random (Q_3) and then query the entire neighborhood of these nodes in the first step, so $\texttt{ENV}(u, 1)$ for $u \in N(v) \setminus W_1$, as well as $\texttt{ENV}(v, 2)$. If any violations are detected here, then we can safely reject. What then remains are only nodes with high degree; as we argue in the analysis below, any set of nodes not in F that are black (which might occur when $\texttt{ENV} \notin$ 1-BP) and which have no white predecessor can actually be covered by recoloring only a small set of nodes (and hence ENV must be close to 1-BP).

Analysis. The query complexity of Algorithm 2 is clear, so we focus on the analysis on its correctness. First we show that Algorithm 2 is indeed a one-sided error algorithm; that is:

▷ Claim 11. If $\texttt{ENV} \in$ 1-BP, then Algorithm 2 always accepts.

Intuitively this is the case because we are only trying to detect violations (and accept unconditionally if we do not manage to find any).

Proof. Since $\texttt{ENV} \in$ 1-BP, we have $\texttt{ENV}(W_1, 1) = 0$ and $\texttt{ENV}(B_2, 2) = 1$. As a result, Algorithm 2 never rejects in line 2. Consider the two possibilities for Algorithm 2 to reject in line 2. The first is that there is a node $v \in Q_3$ with $\texttt{ENV}(v, 2) = 0$ and some $u \in N(v)$

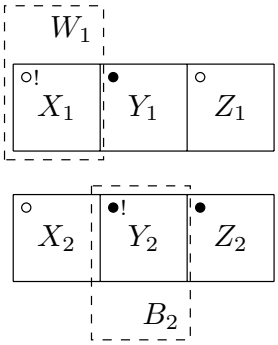


Figure 3 Relation between the sets used in the analysis of Algorithm 2. The sets form a partition of the nodes in the first and second time steps. The small circles indicate the color of the nodes in each set or, in the case of X_1 of Y_2, that the algorithm rejects unless (almost all) nodes in the set have the respective color (denoted with an exclamation mark).

so that $\texttt{ENV}(u,1) = 1$, which contradicts $\texttt{ENV} \in \text{1-BP}$. The second is that $\texttt{ENV}(v,2) = 1$ and $\texttt{ENV}(u,1) = 0$ for every $u \in N(v) \setminus W_1$; however, since $\texttt{ENV}(W_1,1) = 0$, this means $\texttt{ENV}(u,1) = 0$ for every $u \in N(v) \cap W_1$ as well and then $\texttt{ENV}(u,1) = 0$ for every $u \in N(v)$, thus also contradicting $\texttt{ENV} \in \text{1-BP}$. ◁

Now we turn to proving that Algorithm 2 does not have false positives. More specifically we show that Algorithm 2 can only accept with constant probability if $\text{dist}(\texttt{ENV}, \text{1-BP}) < \varepsilon$ is the case (and so, conversely, Algorithm 2 rejects with high probability if $\text{dist}(\texttt{ENV}, \text{1-BP}) \geq \varepsilon$).

For $v \in V$ and $t \in \{1,2\}$, we write $\text{bn}_t(v)$ and $\text{wn}_t(v)$ for the number of black and white neighbors, respectively, of v in step t; formally,

$$\text{bn}_t(v) = |\{u \in N(v) \mid \texttt{ENV}(u,t) = 1\}|, \qquad \text{wn}_t(v) = |\{u \in N(v) \mid \texttt{ENV}(u,t) = 0\}|.$$

Let $\theta = (\varepsilon/4)\sqrt{n/\log n}$ and consider the following sets:

$$\begin{aligned} X_1 &= \{v \in V \mid \text{wn}_2(v) \geq \theta\}, & X_2 &= \{v \in V \mid \text{bn}_1(v) < \theta \wedge \texttt{ENV}(v,2) = 0\}, \\ Y_1 &= \{v \in V \mid \text{wn}_2(v) < \theta \wedge \texttt{ENV}(v,1) = 1\}, & Y_2 &= \{v \in V \mid \text{bn}_1(v) \geq \theta\}, \\ Z_1 &= \{v \in V \mid \text{wn}_2(v) < \theta \wedge \texttt{ENV}(v,1) = 0\}, & Z_2 &= \{v \in V \mid \text{bn}_1(v) < \theta \wedge \texttt{ENV}(v,2) = 1\}. \end{aligned}$$

The sets X_1 and Y_2 contain the nodes for which we can detect that they must be white and black, respectively, by using the query sets Q_1 and Q_2.

▷ Claim 12. With high probability over the choices made by Algorithm 2, $X_1 \subseteq W_1$ and $Y_2 \subseteq B_2$.

Proof. Fix $v \in X_1$. By the Chernoff bound, the probability that no $u \in N(v)$ with $\texttt{ENV}(u,2) = 0$ lands in Q_2 is at most $2e^{-2\log n} < 2/n^2$. Hence, by the union bound, the probability that $X_1 \setminus W_1$ is non-empty is $O(1/n)$. The same applies to Y_2 and B_2. ◁

Next we observe that the queries from Q_1' and Q_2' significantly "reduce" the number of black or white nodes in X_1 or Y_2, respectively; that is, if there is a significant number of such nodes in these sets, then Algorithm 2 will detect them anyway and reject (and thus we can focus the analysis on instances where this is not the case).

▷ Claim 13. If $\texttt{ENV}$ is such that there are $\varepsilon\sqrt{n}$ nodes $v \in X_1$ with $\texttt{ENV}(v,1) = 1$ or $\varepsilon\sqrt{n}$ nodes $v \in Y_2$ with $\texttt{ENV}(v,2) = 0$, then Algorithm 2 rejects $\texttt{ENV}$ with high probability.

Proof. Let $S \subseteq V$ be a subset of $|S| \geq \varepsilon\sqrt{n}$ vertices. Then the probability that $S \cap Q'_i$ is empty is at most

$$\left(1 - \frac{\varepsilon}{\sqrt{n}}\right)^{(24/\varepsilon)\sqrt{n}\log^{3/2} n} < e^{-24\log^{3/2} n} = o\left(\frac{1}{n}\right).$$

Using Claim 12, we have $X_1 \subseteq W_1$ and $Y_2 \subseteq B_2$ with high probability. In this case Algorithm 2 rejects if any node $v \in X_1 \subseteq W_1$ with $\texttt{ENV}(v, 1) = 1$ lands in Q'_2 or any $v \in Y_2 \subseteq B_2$ with $\texttt{ENV}(v, 2) = 0$ lands in Q'_1. Therefore Algorithm 2 rejects with high probability if there are at least $\varepsilon\sqrt{n}$ nodes of either type. ◁

Hence we may now safely assume that all but at most $O(\varepsilon\sqrt{n})$ nodes in X_1 are white in the first time step and that all but at most $O(\varepsilon\sqrt{n})$ nodes in Y_2 are black in the second one. The next observation is that nodes in X_2 are highly connected to X_1. This justifies filtering nodes based on their connections to $W_1 \supseteq X_1$.

▷ Claim 14. On average, a node from X_2 has at most $\theta n/|X_2|$ neighbors not in X_1.

Proof. Every node in Y_1 or Z_1 has at most θ white neighbors by definition, so at most this many neighbors in X_2. Hence there are at most θn edges in total between X_2 and nodes not in X_1. ◁

Finally we show that, if Algorithm 2 accepts ENV with at least constant probability, then we can correct all violations of either type with at most $\varepsilon n/2$ modifications in total for each type. In both cases we must be careful so that these modifications do not create new violations of their own.

▷ Claim 15. If Algorithm 2 accepts ENV with at least constant probability, then there are at most $\varepsilon n/2$ many type I violations in ENV. These violations can be corrected (without creating any new ones) by recoloring $\texttt{ENV}(v, 2)$ black for every violation $(v, 2)$.

Proof. Let R be the set of nodes corresponding to type I violations, that is, $R = \{v \in V \mid \texttt{ENV}(v, 2) = 0 \wedge \exists u \in N(v) : \texttt{ENV}(u, 1) = 1\}$. We prove the claim by proving the contrapositive; that is, if $|R| \geq \varepsilon n/2$, then Algorithm 2 rejects ENV with high probability.

The first observation is that we have $|R \setminus X_2| = o(\varepsilon n)$ (with high probability) due to Claim 13 and then, by the assumption on R, $|X_2| \geq (1 - o(1))\varepsilon n/2$. Hence we focus our analysis on $R \cap X_2$. By Claim 14, on average a node from X_2 has at most $2\theta/\varepsilon = (1/2)\sqrt{n/\log n}$ many neighbors that are outside X_1. By Markov's inequality, this gives us that there are at most $O(n/\log n)$ many nodes $v \in X_2$ for which $|N(v) \setminus X_1| > 4\sqrt{n \log n}$. Using Claim 12, we have $X_1 \subseteq W_1$ and so altogether we have $|R \cap F| \geq \varepsilon n/4$ (with high probability). In this case the probability that $R \cap Q_3$ is empty is at most $(1 - \varepsilon/4)^{4\log(n)/\varepsilon} < e^{-\log(n)} = O(1/n)$, and so Algorithm 2 rejects with high probability. ◁

▷ Claim 16. If Algorithm 2 accepts ENV with at least constant probability, then all type II violations in ENV can be corrected by recoloring at most $\varepsilon n/2$ nodes. In particular, this recoloring is such that we color $\texttt{ENV}(v, 1)$ and $\texttt{ENV}(N(v), 2)$ black for a certain subset of nodes v (and hence does create any new violations).

Proof. Similar to the proof of Claim 15, let $R = \{v \in V \mid \texttt{ENV}(v, 2) = 1 \wedge \forall u \in N(v) : \texttt{ENV}(u, 1) = 0\}$. be the set of type II violations. We show that, if Algorithm 2 accepts with at least constant probability, then we can correct R by recoloring at most $\varepsilon n/2$ nodes black. (Note this does not necessarily mean that $|R| < \varepsilon n/2$ as in the proof of Claim 15. Instead what we prove is an upper bound on the number of recolorings needed to correct R.)

By Claim 13, $|R \setminus Z_2| = o(\varepsilon n)$ and thus $|Z_2| \geq (1 - o(1))\varepsilon n/2$. Arguing as in the proof of Claim 15, if $|R \cap F| \geq \varepsilon n/4$, then Algorithm 2 must reject with high probability. Hence let us focus on the nodes in $R' = R \setminus F$. Consider the bipartite graph where the set of left-vertices is $V \setminus W_1$, that of right-vertices is R', and the edges are as in G. Then the minimum degree of R' in this graph is $4\sqrt{n \log n}$, which means we can apply Lemma 8 and obtain a cover $D \subseteq V \setminus W_1$ of R' with $|D| = (1/4)\sqrt{n \log n}$ nodes. By Claim 12, $D \cap X_1 = \varnothing$ and hence $\mathrm{wn}_2(v) < \theta$ for every $v \in D$ (with high probability). Therefore we can correct R' by coloring $\texttt{ENV}(D, 1)$ and $\texttt{ENV}(N(D), 2)$ all black, which means coloring at most $(\theta/4)\sqrt{n \log n} \leq \varepsilon n/16$ nodes black. Together with $|R \cap F| < \varepsilon n/4$, this means we must color at most $(1/4 + 1/16)\varepsilon n < \varepsilon n/2$ many nodes black in total in order to correct R. ◁

References

1. Noga Alon and Joel H. Spencer. *The Probabilistic Method, Third Edition*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 2008.
2. József Balogh and Boris G. Pittel. Bootstrap percolation on the random regular graph. *Random Struct. Algorithms*, 30(1-2):257–286, 2007. doi:10.1002/RSA.20158.
3. Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 184–193, 1996. doi:10.1109/SFCS.1996.548477.
4. Arnab Bhattacharyya and Yuichi Yoshida. An algebraic characterization of testable boolean CSPs. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 7965, pages 123–134, 2013. doi:10.1007/978-3-642-39206-1_11.
5. Arnab Bhattacharyya and Yuichi Yoshida. *Property Testing - Problems and Techniques*. Springer, 2022. doi:10.1007/978-981-16-8622-1.
6. Hubie Chen, Matt Valeriote, and Yuichi Yoshida. Constant-query testability of assignments to constraint satisfaction problems. *SIAM J. Comput.*, 48(3):1022–1045, 2019. doi:10.1137/18M121085X.
7. Eldar Fischer, Eric Lehman, Ilan Newman, Sofya Raskhodnikova, Ronitt Rubinfeld, and Alex Samorodnitsky. Monotonicity testing over general poset domains. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 474–483, 2002. doi:10.1145/509907.509977.
8. Silvio Frischknecht, Barbara Keller, and Roger Wattenhofer. Convergence in (social) influence networks. In *Proceedings of the 27th International Symposium on Distributed Computing (DISC)*, pages 433–446, 2013. doi:10.1007/978-3-642-41527-2_30.
9. Oded Goldreich. *Computational Complexity: A Conceptional Perspective*. Cambridge University Press, 2008.
10. Oded Goldreich and Dana Ron. On learning and testing dynamic environments. *J. ACM*, 64(3):21:1–21:90, 2017. doi:10.1145/3088509.
11. Paolo De Gregorio, Aonghus Lawlor, and Kenneth A. Dawson. Bootstrap percolation. In Robert A. Meyers, editor, *Encyclopedia of Complexity and Systems Science*, pages 608–626. Springer, 2009. doi:10.1007/978-0-387-30440-3_41.
12. Bernd Gärtner and Ahad N. Zehmakan. Majority model on random regular graphs. In *Proceedings of the 13th Latin American Theoretical Informatics Symposium (LATIN)*, pages 572–583, 2018. doi:10.1007/978-3-319-77404-6_42.
13. Svante Janson, Tomasz Łuczak, Tatyana Turova, and Thomas Vallier. Bootstrap percolation on the random graph $g_{n,p}$. *Ann. Appl. Probab.*, 22(5):1989–2047, 2012. doi:10.1214/11-AAP822.
14. Dominik Kaaser, Frederik Mallmann-Trenn, and Emanuele Natale. On the voting time of the deterministic majority process. In *Proceedings of the 41st International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 55:1–55:15, 2016. doi:10.4230/LIPIcs.MFCS.2016.55.

15 Philip N. Klein, Serge A. Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 682–690, 1993. doi:10.1145/167088.167261.

16 Diego Maldonado, Pedro Montealegre, Martín Ríos Wilson, and Guillaume Theyssier. Local certification of majority dynamics. In Henning Fernau, Serge Gaspers, and Ralf Klasing, editors, *SOFSEM 2024: Theory and Practice of Computer Science - 49th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2024, Cochem, Germany, February 19-23, 2024, Proceedings*, volume 14519 of *Lecture Notes in Computer Science*, pages 369–382. Springer, 2024. doi:10.1007/978-3-031-52113-3_26.

17 Augusto Modanese and Yuichi Yoshida. Testing spreading behavior in networks with arbitrary topologies. *CoRR*, abs/2309.05442, 2023. arXiv:2309.05442.

18 Yonatan Nakar and Dana Ron. Testing dynamic environments: Back to basics. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 98:1–98:20, 2021. doi:10.4230/LIPIcs.ICALP.2021.98.

19 Salil P. Vadhan. Pseudorandomness. *Found. Trends Theor. Comput. Sci.*, 7(1-3):1–336, 2012. doi:10.1561/0400000010.

20 Ahad N. Zehmakan. Tight bounds on the minimum size of a dynamic monopoly. In *Proceedings of the 13th International Conference on Language and Automata Theory and Applications (LATA)*, pages 381–393, 2019. doi:10.1007/978-3-030-13435-8_28.

21 Ahad N. Zehmakan. Majority opinion diffusion in social networks: An adversarial approach. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*, pages 5611–5619, 2021. doi:10.1609/aaai.v35i6.16705.

A Upper Bounds for the Case $T > 2$

In this appendix we consider two different strategies for the case where $T > 2$. An immediate observation to make is that the diameter $\operatorname{diam}(G)$ plays a much more significant role in this setting. For instance, the case where $T \geq (1 + 2/\varepsilon) \operatorname{diam}(G)$ is more or less trivial since then after $\operatorname{diam}(G)$ steps every connected component must be either all-black or all-white and the first $\operatorname{diam}(G)$ steps constitute at most an $\varepsilon/2$ fraction of $\texttt{ENV}$.

A.1 Structure-independent Upper Bound

First we give a generalization of Theorem 1, which is simple to obtain and adequate for settings where Δ and T are not too large. For constant Δ, for instance, it still gives a sublinear query algorithm when $T = O(\log n)$. As the algorithm of Theorem 1, it does not use the graph structure in any way except for determining the neighborhood of each node.

▶ **Theorem 5.** *Let $\varepsilon > 0$ and $T > 2$. There is a* non-adaptive, one-sided error *algorithm that performs $O(\Delta^{T-1}/\varepsilon T)$ queries and decides if $\texttt{ENV} \in \mathsf{1\text{-}BP}$ or $\operatorname{dist}(\texttt{ENV}, \mathsf{1\text{-}BP}) \geq \varepsilon$.*

We adapt Algorithm 1 to obtain Algorithm 3. The analysis does not carry over automatically since we need to consider what happens if we are correcting violations in a time step $t < T$. Unlike in Lemma 10, this kind of correction may now propagate to time steps after t. In addition, we have to assume $\Delta \geq 2$; however, the case $\Delta = 1$ is trivial since then $\operatorname{diam}(G) = 1$ and we need only follow the strategy described at the beginning of this section.

▶ **Lemma 17.** *Let $\Delta \geq 2$. Then*

$$\frac{|\operatorname{viol}(\texttt{ENV})|}{(\Delta+1)nT} \leq \operatorname{dist}(\texttt{ENV}, \mathsf{1\text{-}BP}) \leq \frac{\Delta^{T-1}-1}{(\Delta-1)nT} |\operatorname{viol}(\texttt{ENV})|.$$

Algorithm 3 Structure-independent algorithm for the case of general T with query complexity $O(\Delta^{T-1}/\varepsilon)$.

Pick $Q \subseteq V \times \{t \mid 2 \leq t \leq T\}$ uniformly at random where $|Q| = \lceil 2\Delta^{T-2}/\varepsilon T \rceil$;
Query $\texttt{ENV}(v, t-1)$ and $\texttt{ENV}(u, t)$ for every $(u, t) \in Q$ and $v \in N(u)$ in a time-conforming manner;
for $(u, t) \in Q$ **do**
  if $\texttt{ENV}(u, t) = 0$ *and* $\exists v \in N(u) : \texttt{ENV}(v, t-1) = 1$ **then reject**;
  if $\texttt{ENV}(u, t) = 1$ *and* $\forall v \in N(u) : \texttt{ENV}(v, t-1) = 0$ **then reject**;
end
accept;

See the full version of the paper [17] for the proof.

Now as before with Theorem 1 we have that $\text{dist}(\texttt{ENV}, \text{1-BP}) \geq \varepsilon$ implies

$$\text{viol}(\texttt{ENV}) \geq \frac{\varepsilon(\Delta - 1)nT}{\Delta^{T-1} - 1} > \frac{\varepsilon nT}{2\Delta^{T-2}}.$$

Hence the probability that Algorithm 3 errs is

$$\Pr[Q \cap \text{viol}(\texttt{ENV}) = \varnothing] \leq \left(1 - \frac{\varepsilon T}{2\Delta^{T-2}}\right)^{|Q|} < \frac{1}{e} < \frac{1}{2}.$$

As was the case with Algorithm 1, the query complexity and other properties required in Theorem 5 are clear, and hence Theorem 5 follows.

A.2 Upper Bound Based on Graph Decompositions

The second algorithm is suited for larger values of T and not too dense graphs.

▶ **Theorem 6.** *Let $\varepsilon > 0$ and $T \geq 4/\varepsilon$. Then there is a* non-adaptive, one-sided error *algorithm with query complexity $O(|E| \log(n)/\varepsilon T)$ that decides whether $\texttt{ENV} \in$ 1-BP or $\text{dist}(\texttt{ENV}, \text{1-BP}) \geq \varepsilon$. In addition, if G excludes a fixed minor H (which includes the case where G is planar or, more generally, G has bounded genus), then $O(|E|/\varepsilon T)$ queries suffice.*

The strategy followed by the algorithm relies on graph decompositions. These are partitions induced by sets of edges that cut the graph into components of bounded diameter.

▶ **Definition 18.** *Let $d \in \mathbb{N}_+$ and $\alpha > 0$. A (d, α)-*decomposition *of a graph $G = (V, E)$ is a set of edges $C \subseteq E$ with $|C| \leq \alpha|E|$ and such that there is a partition $V = V_1 + \cdots + V_r$ satisfying the following:*

1. *For $u, v \in V$, $uv \in C$ if and only if there are i and j such that $i \neq j$, $u \in V_i$, and $v \in V_j$.*
2. *For every i, $\text{diam}(V_i) \leq d$.*

The following is a renowned result in graph decompositions:

▶ **Theorem 19** ([3])**.** *For any $d \in \mathbb{N}_+$, every graph G admits a $(d, O(\log(n)/d))$-decomposition.*

This trade-off is optimal for graphs in general. For the special case of graphs excluding a fixed minor (which includes most notably planar graphs or also graphs of bounded genus), we have the following small improvement:

▶ **Theorem 20** ([15])**.** *Let H be a fixed graph. For any $d \in \mathbb{N}_+$, every graph G excluding H as a minor admits a $(d, O(1/d))$-decomposition.*

Algorithm 4 Algorithm for the case of general T based on network decompositions.

$t_1 \leftarrow \lfloor \varepsilon T/4 \rfloor$;
Compute a (t_1, α)-decomposition of G according to Theorem 19 or Theorem 20 and obtain a set of edges C that cuts G into components $V_1, \ldots, V_r$ as in Definition 18;
$B \leftarrow \{v \mid v \text{ is incident to an edge in } C\}$;
Pick $Q \subseteq \{(v,t) \mid v \in V_i \text{ and } t \geq t_1\}$ uniformly at random where $|Q| = \lceil 3/\varepsilon \rceil$;
$Q' \leftarrow \{v \in V \mid \exists t : (v,t) \in Q\}$;
Query $\texttt{ENV}(B, t_1)$, $\texttt{ENV}(Q)$, and $\texttt{ENV}(Q', t_1)$ in a time-conforming fashion;
if $\texttt{ENV}(B, t_1)$ *is not feasible* **then reject**;
for $i \in [r]$ **do**
    $B_i \leftarrow B \cap V_i$;
    $B_i' \leftarrow \{u \in B_i \mid \texttt{ENV}(u, t_1) = 1\}$;
end
for $v \in V$ **do**
    for $i \in [r]$ **do**
        if $B_i' \neq \varnothing$ **then**
            $\alpha_i(v) \leftarrow \min_{u \in B_i' \cup (V_i \setminus B_i)} \operatorname{dist}(u, v)$;
            $\beta_i(v) \leftarrow \min_{u \in B_i'} \operatorname{dist}(u, v)$;
        else
            $\alpha_i(v) \leftarrow \infty$;
            $\beta_i(v) \leftarrow \infty$;
        end
    end
    $\alpha(v) \leftarrow \min_i \alpha_i(v)$;
    $\beta(v) \leftarrow \min_i \beta_i(v)$;
end
for $(v,t) \in Q$ **do**
    Let i be such that $v \in V_i$;
    if $\texttt{ENV}(v, t_1) = 1$ **then**
        if $\texttt{ENV}(v, t) \neq 1$ **then reject**;
    else
        if $t_1 \leq t < t_1 + \alpha(v)$ *and* $\texttt{ENV}(v, t) \neq 0$ **then reject**;
        if $t \geq t_1 + \beta(v)$ *and* $\texttt{ENV}(v, t) \neq 1$ **then reject**;
    end
end
accept;

The claim is that Algorithm 4 satisfies the requirements of Theorem 6. As mentioned in the introduction, the strategy followed by the algorithm is loosely based on a similar testing routine from the paper by Nakar and Ron [18]. In a nutshell, the idea is to split the environment into more manageable components and then use the properties of the local rule to predict how each component must behave.

Approach. Let us recall the relevant details of the strategy of Nakar and Ron [18]. In their paper, the authors studied local rules resembling the majority rule in the restricted setting where G is a path. Their idea involved splitting the path into intervals that intersect at

periodic control points. The first queries yield the state of these control points at a certain time step t_1. If there is no initial configuration leading to what we observe at t_1 (i.e., the configuration is not *feasible*), then we can immediately reject. Otherwise we can use the states of the nodes at the control points (plus some additional queries) to fully predict almost the entirety of $\texttt{ENV}$ after t_1. Hence we only need to test a certain number of times if $\texttt{ENV}(v,t)$ matches our prediction where $(v,t) \in V \times \{t \in \mathbb{N}_+ \mid t \geq t_1\}$ is chosen uniformly at random.

Our approach is more or less the same, though we need to cater for a couple differences between our setting and theirs. We are not in a path, and so in general we cannot split our graph into intervals of the same size; rather we must work with a graph decomposition, which does give us the adequate control points (the vertices incident to the edges of the cut C, which form the set B in Algorithm 4) but only an upper bound on the diameter of each component (which correspond to the intervals in the setting of Nakar and Ron [18]). Fortunately the 1-BP rule is much simpler than majority or the like, and hence the prediction in each component is easier to make. The relevant observation is that the 1-BP rule converges fast to an (all-black) fixed point in graphs of small diameter. (Indeed, the 1-BP rule converges in at most $\operatorname{diam}(G)$ steps.) More specifically, components that started in an all-zero configuration must stay zero until they enter in contact with a black node; meanwhile a component V_i that had at least one black node in it will converge to an all-black configuration in at most $\operatorname{diam}(V_i) \leq t_1$ steps.

Let us now give a more detailed overview of the steps performed by Algorithm 4. For a set $S \subseteq V$ and $t \in [T]$, we say that $\texttt{ENV}(S,t)$ is *feasible* if there is $\texttt{ENV}' \in$ 1-BP such that $\texttt{ENV}'(v,t) = \texttt{ENV}(v,t)$ for every $v \in S$. Theorem 6 first sets t_1 appropriately and determines a graph decomposition of G where the components $V_1, \ldots, V_r$ have diameter at most t_1. We wait for t_1 steps to elapse and then query the states of B, which are the nodes incident to the edge cut C of the graph decomposition, and can immediately reject if what we see is not feasible. At the same time we query a uniformly sampled set Q of pairs corresponding to the states of nodes in time steps after t_1, whose values we shall use later. We then set $B_i = B \cap V_i$ and B_i' to the nodes that are black in B_i in time step t_1. With these we can then compute estimates $\alpha_i(v)$ and $\beta_i(v)$ for each node v and each component V_i. These are only intended to be useful if v is white in time step t_1 and are determined as follows:

- $\alpha_i(v)$ is a *lower bound* on the number of time steps that elapse after t_1 until v turns from white to black. To compute $\alpha_i(v)$, we consider both nodes in B_i' (whose state in t_1 is known to us) and nodes in the inside of V_i (whose state is unknown and which means we must assume that they are black). If there are no nodes in B_i', then we know that V_i was all white at the beginning and we just set $\alpha_i(v) = \infty$.
- $\beta_i(v)$ is an *upper bound* on the number of time steps after t_1 until v turns black at the latest. To compute $\beta_i(v)$ we take into account only nodes which we are sure that are black in t_1, that is, nodes in B_i'. Again, if B_i' is empty, then V_i must have been all white in the first time step; in that case we set $\beta_i(v) = \infty$.

See Figure 4 for an example. Based on these estimates, we can then use the values of Q to make random tests on the state of nodes after t_1 based on what we know from B_i' and $\alpha(v) = \min_i \alpha_i(v)$ and $\beta(v) = \min_i \beta_i(v)$. More precisely, for a pair $(v,t) \in Q$:

- If v was already black in time step t_1, then it must still be black in time step $t \geq t_1$.
- Otherwise v was white in time step t_1 and we can use our estimates $\alpha(v)$ and $\beta(v)$ to verify the predicted state of v in step t, if possible.

The algorithm accepts by default if $\texttt{ENV}$ passes the tests.

The query complexity of Theorem 6 is evident. We refer for the full version of the paper [17] for the proof of its correctness.

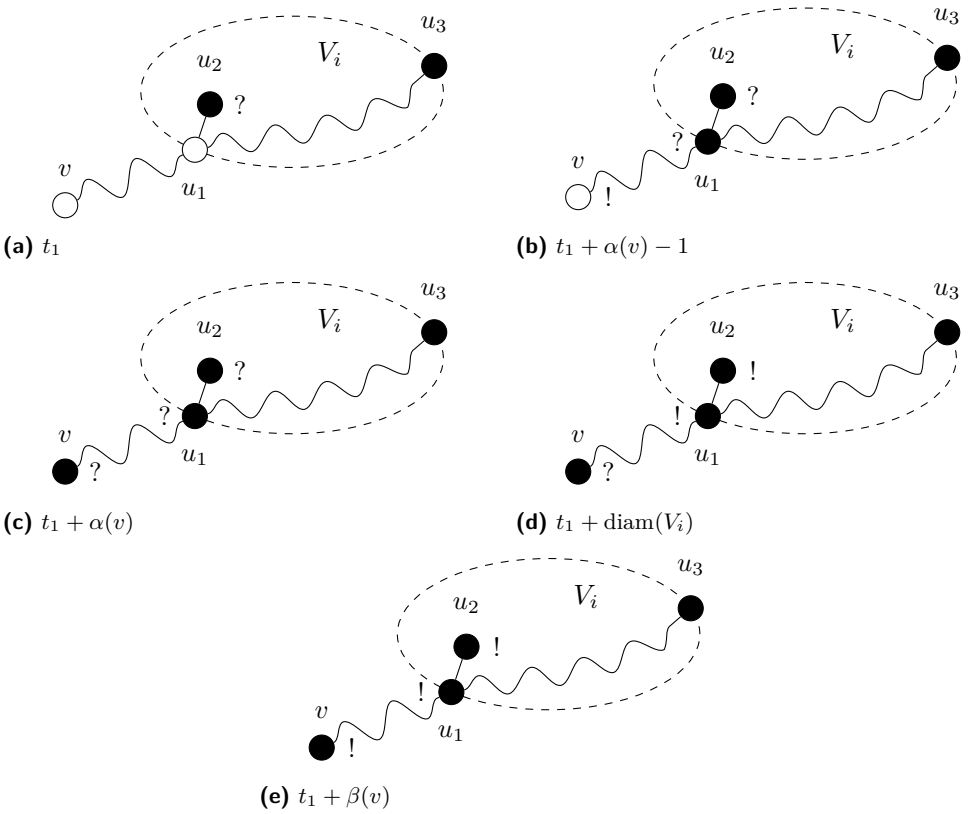


Figure 4 How to predict the color of a node v based on knowledge about the states of nodes in other components. For the sake of illustration, here we are assuming that v belongs to some component which is all white in step t_1 and that the component nearest to v on which a black node appears is V_i. We also suppose that $B_i = \{u_1, u_3\}$ and $\mathrm{dist}(v, u_1) \ll \mathrm{dist}(u_1, u_3) = \mathrm{diam}(V_i)$. In time step t_1 the situation is as in (a). Since $B_i' = \{u_3\}$ is not empty, we must treat V_i as potentially having black nodes since the first time step. We see the states of u_1 and u_3 in t_1 and determine that $\alpha(v) = \mathrm{dist}(v, u_1) + 1 = \mathrm{dist}(v, u_2)$ and $\beta(v) = \mathrm{dist}(v, u_3)$; however, we do not know the color of u_2 since it is inside V_i and we do not query it, so we have to treat it as a potentially black node (denoted by a question mark). After $\alpha(v) - 1$ steps (b) we know that v must still be white (denoted by an exclamation mark) since the closest node to it that is possibly black in step t_1 is the node u_2. After $\alpha(v)$ steps (c) we are no longer certain about the color of v. After $\mathrm{diam}(V_1)$ steps (d) we know that u_1 must be black, but we still cannot say anything about v. Finally after $\beta(v)$ steps (e) we are sure that v has turned black at the latest since u_3 was black in t_1.

Alphabet Reduction for Reconfiguration Problems

Naoto Ohsaka
CyberAgent, Inc., Tokyo, Japan

Abstract

We present a reconfiguration analogue of *alphabet reduction* à la Dinur (J. ACM, 2007) [7] and its applications. Given a binary constraint graph G and its two satisfying assignments ψ^{ini} and ψ^{tar}, the Maxmin 2-CSP Reconfiguration problem requests to transform ψ^{ini} into ψ^{tar} by repeatedly changing the value of a single vertex so that the minimum fraction of satisfied edges is maximized. We demonstrate a polynomial-time reduction from Maxmin 2-CSP Reconfiguration with arbitrarily large alphabet size $W \in \mathbb{N}$ to itself with universal alphabet size $W_0 \in \mathbb{N}$ such that

1. the perfect completeness is preserved, and
2. if any reconfiguration for the former violates ε-fraction of edges, then $\Omega(\varepsilon)$-fraction of edges must be unsatisfied during any reconfiguration for the latter.

The crux of its construction is the *reconfigurability of Hadamard codes*, which enables to reconfigure between a pair of codewords, while avoiding getting too close to the other codewords. Combining this alphabet reduction with gap amplification due to Ohsaka (SODA 2024) [26], we are able to amplify the 1 vs. $1-\varepsilon$ gap for arbitrarily small $\varepsilon \in (0,1)$ up to the 1 vs. $1-\varepsilon_0$ for some universal $\varepsilon_0 \in (0,1)$ *without* blowing up the alphabet size. In particular, a 1 vs. $1-\varepsilon_0$ gap version of Maxmin 2-CSP Reconfiguration with alphabet size W_0 is **PSPACE**-hard given a probabilistically checkable reconfiguration proof system having any soundness error $1-\varepsilon$ due to Hirahara and Ohsaka (STOC 2024) [14] and Karthik C. S. and Manurangsi (2023) [17]. As an immediate corollary, we show that there exists a universal constant $\varepsilon_0 \in (0,1)$ such that many popular reconfiguration problems are **PSPACE**-hard to approximate within a factor of $1-\varepsilon_0$, including those of 3-SAT, Independent Set, Vertex Cover, Clique, Dominating Set, and Set Cover. This may not be achieved only by gap amplification of Ohsaka [26], which makes the alphabet size gigantic depending on ε^{-1}.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Error-correcting codes

Keywords and phrases reconfiguration problems, hardness of approximation, Hadamard codes, alphabet reduction

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.113

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2402.10627` [25]

Acknowledgements I wish to thank Shuichi Hirahara for helpful conversations, and thank the anonymous referees for letting me know a simple construction of an assignment tester due to O'Donnell [22, Theorem 7.16] and for their suggestions which help improve the presentation of this paper.

1 Introduction

1.1 Background

Combinatorial reconfiguration is a brand-new field in theoretical computer science that concerns the reachability and connectivity over the solution space of a combinatorial problem. One canonical **PSPACE**-complete reconfiguration problem is 2-CSP Reconfiguration: given a binary constraint graph G over alphabet Σ and its two satisfying assignments ψ^{ini} and ψ^{tar}, we are requested to transform ψ^{ini} into ψ^{tar} by repeatedly changing the value of a single vertex while the feasibility of intermediate assignments is maintained. Such a sequence of

feasible solutions is referred to as a *reconfiguration sequence*. Since the establishment of the unified framework due to Ito, Demaine, Harvey, Papadimitriou, Sideri, Uehara, and Uno [16], the complexity of many reconfiguration problems has been investigated, including those of Satisfiability, Coloring, Independent Set, Vertex Cover, and Clique. We refer the readers to the survey by Bousquet, Mouawad, Nishimura, and Siebertz [6], Mynhardt and Nasserasr [20], Nishimura [21], and van den Heuvel [31]. One latest trend is to study *approximate reconfigurability* [23, 24, 26], which affords to relax the feasibility of intermediate solutions during reconfiguration. For example, in Maxmin 2-CSP Reconfiguration [23], which is an *optimization version* of 2-CSP Reconfiguration, we can adopt any *non-satisfying assignments*, but are required to maximize the minimum fraction of edges satisfied during reconfiguration. Such optimization versions would be come up with naturally to deal with **PSPACE**-hardness of many reconfiguration problems. See Section 1.5 for other optimization versions of reconfiguration problems.

One of the most important questions concerning approximate reconfigurability was **PSPACE**-*hardness of approximation* for reconfiguration problems, posed by Ito, Demaine, Harvey, Papadimitriou, Sideri, Uehara, and Uno [16, Section 5] as an open problem. Though **NP**-hardness of approximation for reconfiguration problems (e.g., Maxmin SAT Reconfiguration) was shown by [16], their proofs do not imply **PSPACE**-hardness because of relying on the **NP**-hardness of approximating the corresponding optimization problems (e.g., Max SAT). The significance of showing **PSPACE**-hardness compared to **NP**-hardness is that it disproves the existence of a witness (especially a reconfiguration sequence) of polynomial length under **NP** $\neq$ **PSPACE**. Ohsaka [23] showed that a host of (optimization versions of) reconfiguration problems are **PSPACE**-hard to approximate under the *Reconfiguration Inapproximability Hypothesis* (RIH), which postulates that a gap version of Maxmin CSP Reconfiguration is **PSPACE**-hard. Very recently, Hirahara and Ohsaka [14] and Karthik C. S. and Manurangsi [17] independently announced the proof of RIH, thereby affirmatively resolving the open problem of Ito, Demaine, Harvey, Papadimitriou, Sideri, Uehara, and Uno [16]. The proof is based on a construction of *probabilistically checkable reconfiguration proof* (PCRP) systems for **PSPACE**. The present study delves deeper into **PSPACE**-hardness of approximation for reconfiguration problems *given* the resolution of RIH.

The limitation of [23] along with the PCRP theorem [14, 17] is that the degree of inapproximability is not explicitly shown: although the PCRP theorem implies that Maxmin 2-CSP Reconfiguration is **PSPACE**-hard to approximate within a factor of $1-\varepsilon$, its *gap parameter* $\varepsilon \in (0,1)$ is implicit and thus might be very tiny. To circumvent this limitation, Ohsaka [26] successfully developed Dinur's style *gap amplification* [7] for Maxmin 2-CSP Reconfiguration, which amplifies the 1 vs. $1-\varepsilon$ gap for arbitrarily small $\varepsilon \in (0,1)$ up to the 1 vs. 0.9942 gap. This result can be used to show 1.0029-inapproximability for Minmax Set Cover Reconfiguration [26]. Unfortunately, there still remains another issue: *the alphabet size becomes gigantic depending on* ε^{-1}.[1] Consider for example reducing Maxmin 2-CSP Reconfiguration with alphabet size W to Maxmin 3-SAT Reconfiguration in a gap-preserving manner. According to [23], if the former problem has a δ-gap, the latter problem's gap turns out to be $\frac{\delta}{2^{\Omega(W)}}$. This is undesirable if W depends on ε. Our target in this paper is thus a reconfiguration analogue of *alphabet reduction*, i.e., a polynomial-time reduction from Maxmin 2-CSP Reconfiguration to itself that makes a large alphabet into a tiny one without much deteriorating the gap value.

[1] Precisely, the alphabet size becomes $W^{d^{\mathcal{O}(\varepsilon^{-1})}}$ for some $W, d \in \mathbb{N}$ by [26], which is doubly exponential in ε^{-1}.

Table 1 Gap-preserving reductions used in Corollary 1.2. We can reduce $\mathsf{Gap}_{1,1-\varepsilon}$ q-CSP_W Reconfiguration (i.e., the PCRP theorem) to $\mathsf{Gap}_{1,1-\varepsilon_0}$ 2-CSP_{W_0} Reconfiguration regardless of the values of $\varepsilon \in (0,1)$ and $q, W \in \mathbb{N}$.

gap problem	ref.	technique	gap value	alphabet size
q-CSP Reconf	—	—	arbitrarily small ε	arbitrarily large W
2-CSP Reconf	[23]	degree reduction	depends on ε, q, W	universal const.
2-CSP Reconf	[26]	gap amplification	universal const.	depends on ε, q, W
2-CSP Reconf	(this paper)	alphabet reduction	universal const. ε_0	universal const. W_0

1.2 Our Results

We present alphabet reduction for Maxmin 2-CSP Reconfiguration à la Dinur [7] and its applications. Given an instance of Maxmin 2-CSP Reconfiguration with arbitrarily large alphabet, we are able to reduce the alphabet size to a universal constant $W_0 \in \mathbb{N}$ preserving the gap value by up to a constant factor:

▶ **Theorem 1.1** (Alphabet reduction; informal; see Theorem 3.1)**.** *There exist universal constants $W_0 \in \mathbb{N}$ and $\kappa \in (0,1)$ and a polynomial-time reduction from* Maxmin 2-CSP Reconfiguration *with arbitrarily large alphabet size $W \in \mathbb{N}$ to itself with alphabet size W_0 such that*

1. *the perfect completeness is preserved, and*
2. *if any reconfiguration for the former violates ε-fraction of edges, then $\kappa \cdot \varepsilon$-fraction of edges must be unsatisfied during any reconfiguration for the latter.*

Our reduction is independent of ε; namely, ε does not have to be constant, e.g., $\varepsilon = (\#\text{ of edges})^{-1}$. The main ingredient of its construction is the *reconfigurability of Hadamard codes*, which appears later in Section 1.3.

As a corollary of Theorem 3.1 and [23, 26], we are able to amplify the 1 vs. $1-\varepsilon$ gap for arbitrarily small $\varepsilon \in (0,1)$ up to the 1 vs. $1-\varepsilon_0$ gap for some universal $\varepsilon_0 \in (0,1)$ *without* blowing up the alphabet size. Slightly more formally, for any $\varepsilon \in (0,1)$ and $W \in \mathbb{N}$, $\mathsf{Gap}_{1,1-\varepsilon}$ 2-CSP_W Reconfiguration requests to distinguish whether, for a binary constraint graph with alphabet size W and its two satisfying assignments ψ^{ini} and ψ^{tar}, (1) there exists a reconfiguration sequence from ψ^{ini} to ψ^{tar} consisting only of satisfying assignments, or (2) every reconfiguration sequence violates more than ε-fraction of edges.

▶ **Corollary 1.2** (from Theorem 3.1 and [23, 26])**.** *There exist universal constants $\varepsilon_0 \in (0,1)$ and $W_0 \in \mathbb{N}$ such that for arbitrarily small $\varepsilon \in (0,1)$ and large $q, W \in \mathbb{N}$, there exists a gap-preserving reduction from* $\mathsf{Gap}_{1,1-\varepsilon}$ q-CSP_W Reconfiguration *to* $\mathsf{Gap}_{1,1-\varepsilon_0}$ 2-CSP_{W_0} Reconfiguration*. In particular, the latter problem is* **PSPACE***-hard.*

Since *both* ε_0 and W_0 do not depend on either ε, q, or W, Corollary 1.2 makes the degree of inapproximability and alphabet size of Maxmin 2-CSP Reconfiguration *oblivious to* the soundness error, query complexity, and alphabet size of any PCRP system [14, 17]. Concretely, we would have $\varepsilon_0 = \kappa \cdot (1 - 0.9942) > 10^{-18}$ and $W_0 < 2{,}000{,}000$, where number 0.9942 comes from [26]. See also the proof of Theorem 3.1. This may not be achieved only by gap amplification due to Ohsaka [26]. See also Table 1 for a sequence of gap-preserving reductions used in Corollary 1.2.

By Corollary 1.2, we immediately obtain the following gap-preserving reducibility from any PCRP system to many popular reconfiguration problems:

▶ **Theorem 1.3** (from Corollary 1.2 and [23, 26]). *There exists a universal constant $\varepsilon_0 \in (0,1)$ such that for every $\varepsilon \in (0,1)$ and $q, W \in \mathbb{N}$, $\mathsf{Gap}_{1,1-\varepsilon}$ q-CSP_W Reconfiguration is polynomial-time reducible to a 1 vs. $1-\varepsilon_0$ gap version of the following reconfiguration problems*:

2-CSP Reconfiguration, 3-SAT Reconfiguration, Independent Set Reconfiguration, Vertex Cover Reconfiguration, Clique Reconfiguration, Dominating Set Reconfiguration, Set Cover Reconfiguration, *and* Nondeterministic Constraint Logic.

In particular, optimization versions of the above problems are **PSPACE***-hard to approximate within a factor of* $1-\varepsilon_0$.

Once again, Theorem 1.3 is different from a consequence of gap-preserving reductions due to Ohsaka [23] in a sense that it renders ε_0 *independent of* the value of ε.[2] Such **PSPACE**-hardness results seem to be known only for (optimization versions of) 2-CSP Reconfiguration (0.9942-factor) [26], Set Cover Reconfiguration (1.0029-factor) [26], and Clique Reconfiguration ($n^{-\Omega(1)}$-factor) [14] (to the best of our knowledge).

Proofs marked with $*$ are omitted and can be found in the full version of this paper [25].

1.3 Proof Overview

The construction of alphabet reduction for Maxmin 2-CSP Reconfiguration (Theorem 3.1) is based on that for Max 2-CSP due to Dinur [7], which comprises two partial steps: The first step is **robustization**, which replaces each constraint π_e of edge e by a Boolean circuit C_e that accepts $f \circ g$ if and only if $f \circ g = \mathsf{Had}(\alpha) \circ \mathsf{Had}(\beta)$ such that (α, β) satisfies π_e, where Had is the Hadamard code (see Section 2 for the definition).[3] The soundness case ensures that for "many" edges e, the restricted assignment is $\Theta(1)$-far from any satisfying truth assignment to C_e. The second step is **composition**, which composes each circuit C_e with an *assignment tester* [7, 8] (a.k.a. *PCP of proximity* [4]) of constant size to break down C_e into a system of binary constraints over small alphabet while sharing the common variables for different circuits.

The main challenge to achieving alphabet reduction for Maxmin 2-CSP Reconfiguration is its robustization. Simply applying the above robustization procedure to Maxmin 2-CSP Reconfiguration, we are required to reconfigure between a pair of codewords, say, $\mathsf{Had}(\alpha_1)$ and $\mathsf{Had}(\alpha_2)$ for $\alpha_1 \neq \alpha_2$. Such reconfiguration must pass through a function $\gtrapprox \frac{1}{4}$-far from the codeword and thus from any satisfying truth assignment to the above circuit C_e, sacrificing the perfect completeness. There is a dilemma that distinct codewords should be far from each other, yet they need to be reconfigurable with each other. One might thus think of enforcing C_e to accept functions that are $\frac{1}{4}$-close to the codeword. Unfortunately, this modification reduces the robustness to $o(1)$ in the soundness case, as shown in an example below (see also Example 3.7). This explains why robustization for Maxmin 2-CSP Reconfiguration is nontrivial.

▶ **Example 1.4** (Failed attempt). Define a binary constraint $\pi_e \coloneqq \{(\alpha_1, \beta_1), (\alpha_2, \beta_2)\} \subset \Sigma \times \Sigma$ and a Hadamard code $\mathsf{Had}\colon \Sigma \to \mathbb{F}_2^\ell$. Construct a (seemingly promising) circuit $\widetilde{C}_e$ such that $\widetilde{C}_e(f \circ g) = 1$ if and only if

[2] We stress that Theorem 1.3 is essentially different from the following statement (where ε_0 can depend on ε, q, and W), which is immediate from [23]: "For arbitrarily small $\varepsilon \in (0,1)$ and large $q, W \in \mathbb{N}$, there exists $\varepsilon_0 \in (0,1)$ such that $\mathsf{Gap}_{1,1-\varepsilon}$ q-CSP_W Reconfiguration is polynomial-time reducible to a 1 vs. $1-\varepsilon_0$ gap version of the reconfiguration problems listed in Theorem 1.3."

[3] Though Dinur [7] used an error-correcting code $\mathsf{enc}\colon \Sigma \to \mathbb{F}_2^\ell$ having *linear dimension* (i.e., $\ell = \mathcal{O}(\log|\Sigma|)$), we can afford to use the Hadamard code for our purpose because $|\Sigma| = \mathcal{O}(1)$.

1. both f and g are $\frac{1}{4}$-close to some Hadamard codewords;
2. if f and g are $\frac{1}{4}$-close to $\mathsf{Had}(\alpha)$ and $\mathsf{Had}(\beta)$, respectively, then (α, β) must satisfy π_e.

Then, the following issue arises: even if f is closest to $\mathsf{Had}(\alpha)$ and g is closest to $\mathsf{Had}(\beta)$ such that $(\alpha, \beta) \notin \pi$, we cannot exclude the possibility that $f \circ g$ is $o(1)$-close to some satisfying truth assignment to $\widetilde{C}_e$. Suppose f is $\frac{1}{4}$-close to both $\mathsf{Had}(\alpha_1)$ and $\mathsf{Had}(\alpha_2)$ and g is $\frac{1}{4}$-close to both $\mathsf{Had}(\beta_1)$ and $\mathsf{Had}(\beta_2)$. Changing particular two bits of $f \circ g$, we obtain $f^\star \circ g^\star$ that is $\left(\frac{1}{4} - \frac{1}{\ell}\right)$-close to $\mathsf{Had}(\alpha_1) \circ \mathsf{Had}(\beta_1)$. Since $\widetilde{C}(f^\star \circ g^\star) = 1$, $f \circ g$ is $\frac{1}{\ell}$-close to a satisfying truth assignment to $\widetilde{C}_e$. ⌟

The crux of a reconfiguration analogue of robustization is what we call the *reconfigurability of Hadamard codes*:

▶ **Lemma 1.5** (Reconfigurability of Hadamard codes; informal; see Lemma 3.2)**.** *There exists a universal constant $\delta_0 \in (0, 1)$ such that for any $n \geqslant 9$ and $\boldsymbol{\alpha} \neq \boldsymbol{\beta} \in \mathbb{F}_2^n$, there exists a reconfiguration sequence from $\mathsf{Had}(\boldsymbol{\alpha})$ to $\mathsf{Had}(\boldsymbol{\beta})$ such that every function in it is*

- *$\frac{1}{4}$-close to either $\mathsf{Had}(\boldsymbol{\alpha})$ or $\mathsf{Had}(\boldsymbol{\beta})$, and*
- *$\left(\frac{1}{4} + \delta_0\right)$-far from $\mathsf{Had}(\boldsymbol{\gamma})$ for every $\boldsymbol{\gamma} \neq \boldsymbol{\alpha}, \boldsymbol{\beta}$.*

Lemma 3.2 enables us to reconfigure between a pair of codewords, avoiding getting too (say, $\frac{1}{4} + \delta_0$) close to the other codewords. The existence of such a reconfiguration sequence is shown by a simple application of the structural property of a triple of distinct Hadamard codewords and the probabilistic method. Lemma 3.2 is still nontrivial in that it does not hold if $n = 3$ (see Observation 3.3). Using the reconfigurability of Hadamard codes, we implement alphabet reduction of Maxmin 2-CSP Reconfiguration as follows:

- **Robustization** (Lemma 3.6): Convert a binary constraint π_e for edge e into a circuit C_e such that $C_e(f \circ g) = 1$ if and only if
  1. both f and g are $\frac{1}{4}$-close to some Hadamard codewords;
  2. if f and g are $\left(\frac{1}{4} + \frac{\delta_0}{2}\right)$-close to $\mathsf{Had}(\alpha)$ and $\mathsf{Had}(\beta)$, respectively, then (α, β) must satisfy π_e.

  (The difference from $\widetilde{C}_e$ of Example 1.4 is highlighted.) Consider π_e, f, and g appearing in Example 1.4 again for the soundness case. Suppose C_e is constructed from π_e. To make $f \circ g$ to satisfy C_e, we must modify them so that f and g are $\left(\frac{1}{4} + \frac{\delta_0}{2}\right)$-far from $\mathsf{Had}(\alpha_2)$ and $\mathsf{Had}(\beta_2)$ (or $\mathsf{Had}(\alpha_1)$ and $\mathsf{Had}(\beta_1)$), respectively; namely, $f \circ g$ is $\frac{\delta_0}{2}$-far from any satisfying truth assignment to C_e. Even though C_e demands stricter conditions than $\widetilde{C}_e$ of Example 1.4, the perfect completeness can be derived using Lemma 3.2.
- **Composition** (Proposition 3.10): Just feeding each circuit C_e to an assignment tester $\mathcal{P}$ of [7] breaks the perfect completeness; instead, we apply $\mathcal{P}$ to C_e *twice* to create twins of binary constraint systems sharing the input variables to C_e. Our 4-query verifier then picks a pair of edges from each of the twins uniformly at random, and accepts if either of them is satisfied, which may be thought of as *rectangular PCPs* [5]. This kind of redundancy is crucial for ensuring the perfect completeness of reconfiguration problems. On the other hand, if δ-fraction of the edges are unsatisfied in both of the twins, the verifier rejects with probability δ^2 owing to its rectangularity.

In the language of probabilistic proofs, the above alphabet reduction may be thought of as a composition of (PCRPs) due to Hirahara and Ohsaka [14], where the outer PCRP is a 2-query PCRP verifier and the inner PC(R)P is an assignment tester. To make the outer PCRP enjoy a reconfiguration analogue of the robustness as in Lemma 3.6, we replace each variable by a block of bits and modify the original circuit associated with each edge e (i.e., binary constraint π_e) appropriately so as to reflect the reconfigurability of Hadamard codes.

1.4 Towards Dinur's Style Proof of RIH?

Given degree reduction [23], gap amplification [26], and alphabet reduction (this paper) for Maxmin 2-CSP Reconfiguration, one might think of proving RIH imitating Dinur's proof of the PCP theorem [7]. Though the resolution of RIH is not the main motivation for developing alphabet reduction and RIH has already been proven by Hirahara and Ohsaka [14] and Karthik C. S. and Manurangsi [17], such a different proof is still useful in a sense that it would rely only on (slightly) simpler tools. Unfortunately, merely putting them together does not work as expected because some of the reductions are only *gap-preserving*, which requires that there is already a constant gap $\varepsilon \in (0,1)$ between completeness and soundness, and thus weaker than those of Dinur [7]. Consider for example degree reduction of Maxmin 2-CSP Reconfiguration. Unlike Papadimitriou–Yannakakis's degree reduction for Max 2-CSP [28], Ohsaka's degree reduction [23] uses near-Ramanujan graphs [2, 19] of degree $\Theta(\varepsilon^{-2})$. Since we need to begin gap amplification with $\varepsilon = (\#\text{ of edges})^{-1} = o(1)$, applying the degree reduction step of [23] results in a superconstant degree, failing to reduce the degree of Maxmin 2-CSP Reconfiguration. Gap amplification of Ohsaka [26] also relies on the assumption that the gap value is a constant, see [26, Claim 3.7]. Note that alphabet reduction in the present study works for any subconstant gap.

1.5 Additional Related Work

In [16], **NP**-hardness of approximation is shown for optimization versions of Clique Reconfiguration and SAT Reconfiguration using **NP**-hardness of approximating Max Clique [12] and Max SAT [13], respectively. Other reconfiguration problems whose approximability was investigated include Subset Sum Reconfiguration, which admits a PTAS [15] and Submodular Reconfiguration, which admits a constant-factor approximation [27]. It is known that a naive parallel repetition for Maxmin 2-CSP Reconfiguration fails to decrease the soundness error [24] unlike the parallel repetition theorem due to Raz [30]; in fact, Maxmin 2-CSP Reconfiguration is approximable within a factor of nearly $\frac{1}{4}$ [24] while **NP**-hard to approximate within a factor better than $\frac{3}{4}$ [26]. Karthik C. S. and Manurangsi [17] demonstrate matching lower and upper bounds, i.e., **NP**-hardness of $\left(\frac{1}{2}+\varepsilon\right)$-factor approximation and a $\left(\frac{1}{2}-\varepsilon\right)$-factor approximation algorithm for every $\varepsilon \in \left(0, \frac{1}{2}\right)$.

The *overlap gap property* [10, 1, 18, 11, 32] refers to the separation phenomena of the overlaps between near-optimal solutions on random instance, which implies approximate reconfigurability; see also [26].

2 Preliminaries

2.1 Notations

For a nonnegative integer $n \in \mathbb{N}$, let $[n] \coloneqq \{1, 2, \ldots, n\}$. Denote by $\mathfrak{S}_n$ the set of all permutations over $[n]$. A *sequence* $\mathscr{S}$ of a finite number of objects $S^{(1)}, \ldots, S^{(T)}$ is denoted by $(S^{(1)}, \ldots, S^{(T)})$, and we write $S^{(t)} \in \mathscr{S}$ to indicate that $S^{(t)}$ appears in $\mathscr{S}$. The symbol $\circ$ stands for a concatenation of two strings, $\langle \cdot, \cdot \rangle$ for the inner product, $\mathbb{F}_2 = \{0,1\}$ for the finite field with two elements. We use $\uplus$ to emphasize that the union is taken over disjoint sets. Let Σ be a finite set called *alphabet*. For a length-n string $f \in \Sigma^n$ and index set $I \subseteq [n]$, we use $f|_I$ to denote the restriction of f to I. The *relative distance* between two strings $f, g \in \Sigma^n$, denoted $\Delta(f,g)$, is defined as the fraction of positions on which f and g differ; namely,

$$\Delta(f,g) \coloneqq \mathop{\mathbb{P}}_{i\sim[n]}\Big[f_i \neq g_i\Big] = \frac{|\{i\in[n] \mid f_i \neq g_i\}|}{n}. \tag{2.1}$$

We say that f is *ε-close* to g if $\Delta(f,g) \leqslant \varepsilon$ and *ε-far* from g if $\Delta(f,g) > \varepsilon$. For a set of strings $S \subseteq \Sigma^n$, analogous notions are used; e.g., $\Delta(f,S) \coloneqq \min_{g\in S}\Delta(f,g)$ and f is ε-close to S if $\Delta(f,S) \leqslant \varepsilon$. For a string $\boldsymbol{\alpha} \in \mathbb{F}_2^n$, its *Hadamard code* is defined as a function $\mathsf{Had}(\boldsymbol{\alpha})\colon \mathbb{F}_2^n \to \mathbb{F}_2$ such that $\mathsf{Had}(\boldsymbol{\alpha})(\mathbf{x}) = \langle \boldsymbol{\alpha}, \mathbf{x}\rangle$ for all $\mathbf{x} \in \mathbb{F}_2^n$. We call $\mathsf{Had}(\boldsymbol{\alpha})$ for each $\boldsymbol{\alpha}$ a *codeword* of the Hadamard code, and write $\mathsf{Had}(\cdot)$ for the set of all codewords. Note that the relative distance between any pair of distinct codewords of $\mathsf{Had}(\cdot)$ is $\frac{1}{2}$; i.e., $\Delta(\mathsf{Had}(\boldsymbol{\alpha}), \mathsf{Had}(\boldsymbol{\beta})) = \frac{1}{2}$ for all $\boldsymbol{\alpha} \neq \boldsymbol{\beta} \in \mathbb{F}_2^n$.

2.2 Constraint Satisfaction Problem and Reconfigurability

We introduce reconfiguration problems on constraint satisfaction. The notion of constraint graphs is first introduced.

▶ **Definition 2.1.** A *q-ary constraint graph* is defined as a tuple $G = (V, E, \Sigma, \Pi)$ such that (V, E) is a q-uniform hypergraph called the *underlying graph*, Σ is a finite set called the *alphabet*, and $\Pi = (\pi_e)_{e\in E}$ is a collection of q-ary *constraints*, where each constraint $\pi_e \subseteq \Sigma^e$ is a set of q-tuples of acceptable values that q vertices in e can take. ⌟

For an *assignment* $\psi\colon V \to \Sigma$, we say that ψ *satisfies* hyperedge $e = \{v_1, \dots, v_q\} \in E$ (or constraint π_e) if $\psi(e) \coloneqq (\psi(v_1), \dots, \psi(v_q)) \in \pi_e$, and ψ *satisfies* G if it satisfies all hyperedges of G. For two satisfying assignments ψ^{ini} and ψ^{tar} for G, a *reconfiguration sequence from ψ^{ini} to ψ^{tar}* over Σ^V is any sequence $(\psi^{(1)}, \dots \psi^{(T)})$ such that $\psi^{(1)} = \psi^{\mathsf{ini}}$, $\psi^{(T)} = \psi^{\mathsf{tar}}$, and every two neighboring assignments $\psi^{(t)}$ and $\psi^{(t+1)}$ differ in at most one vertex. In the q-CSP Reconfiguration problem, for a q-ary constraint graph G and its two satisfying assignments ψ^{ini} and ψ^{tar}, we are asked to decide if there is a reconfiguration sequence of satisfying assignments for G from ψ^{ini} to ψ^{tar}. Hereafter, the suffix "$_W$" designates the restricted case that the alphabet size $|\Sigma|$ is integer $W \in \mathbb{N}$.

Subsequently, we formulate an optimization version of q-CSP Reconfiguration [16, 23], which allows going through non-satisfying assignments. For a constraint graph $G = (V, E, \Sigma, \Pi)$ and an assignment $\psi\colon V \to \Sigma$, its *value* is defined as the fraction of edges of G satisfied by ψ; namely,

$$\mathsf{val}_G(\psi) \coloneqq \frac{1}{|E|} \cdot \left|\left\{e \in E \;\middle|\; \psi \text{ satisfies } e\right\}\right|. \tag{2.2}$$

For a reconfiguration sequence $\boldsymbol{\psi} = (\psi^{(1)}, \dots, \psi^{(T)})$ of assignments, let $\mathsf{val}_G(\boldsymbol{\psi})$ denote the *minimum fraction* of satisfied edges over all $\psi^{(t)}$'s in $\boldsymbol{\psi}$; namely,

$$\mathsf{val}_G(\boldsymbol{\psi}) \coloneqq \min_{\psi^{(t)}\in\boldsymbol{\psi}} \mathsf{val}_G(\psi^{(t)}). \tag{2.3}$$

In Maxmin q-CSP Reconfiguration, we wish to maximize $\mathsf{val}_G(\boldsymbol{\psi})$ subject to $\boldsymbol{\psi} = (\psi^{\mathsf{ini}}, \dots, \psi^{\mathsf{tar}})$. For two assignments $\psi^{\mathsf{ini}}, \psi^{\mathsf{tar}}\colon V \to \Sigma$ for G, let $\mathsf{val}_G(\psi^{\mathsf{ini}} \leftrightsquigarrow \psi^{\mathsf{tar}})$ denote the maximum value of $\mathsf{val}_G(\boldsymbol{\psi})$ over all possible reconfiguration sequences $\boldsymbol{\psi}$ from ψ^{ini} to ψ^{tar}; namely,

$$\mathsf{val}_G(\psi^{\mathsf{ini}} \leftrightsquigarrow \psi^{\mathsf{tar}}) \coloneqq \max_{\boldsymbol{\psi}=(\psi^{\mathsf{ini}},\dots,\psi^{\mathsf{tar}})} \mathsf{val}_G(\boldsymbol{\psi}). \tag{2.4}$$

The gap version of Maxmin q-CSP Reconfiguration is defined as follows.

▶ **Problem 2.2.** For every numbers $0 \leqslant s \leqslant c \leqslant 1$ and integer $q \in \mathbb{N}$, $\mathsf{Gap}_{c,s}$ q-CSP Reconfiguration requests to determine for a q-ary constraint graph G and its two assignments ψ^{ini} and ψ^{tar}, whether $\mathsf{val}_G(\psi^{\mathsf{ini}} \leftrightsquigarrow \psi^{\mathsf{tar}}) \geqslant c$ (the input is a YES instance) or $\mathsf{val}_G(\psi^{\mathsf{ini}} \leftrightsquigarrow \psi^{\mathsf{tar}}) < s$ (the input is a NO instance). Here, c and s are respectively called *completeness* and *soundness*. ⌟

We can assume ψ^{ini} and ψ^{tar} satisfy G whenever $c = 1$. The *Reconfiguration Inapproximability Hypothesis* (RIH) [23] postulates that $\mathsf{Gap}_{1,1-\varepsilon}$ q-CSP_W Reconfiguration is **PSPACE**-hard for some $\varepsilon \in (0,1)$ and $q, W \in \mathbb{N}$, which has been recently proven by Hirahara and Ohsaka [14] and Karthik C. S. and Manurangsi [17].

3 Alphabet Reduction for Maxmin 2-CSP Reconfiguration

In this section, we prove the main result of this paper, i.e., an explicit construction of *alphabet reduction* for Maxmin 2-CSP Reconfiguration, as formally stated below.

▶ **Theorem 3.1** (Alphabet reduction). *There exist universal constants $W_0 \in \mathbb{N}$ and $\kappa \in (0,1)$, and a polynomial-time algorithm $\mathcal{A}$ that takes an instance $(G, \psi^{\mathsf{ini}}, \psi^{\mathsf{tar}})$ of* Maxmin 2-CSP_W Reconfiguration *with alphabet size $W \in \mathbb{N}$ and produces an instance $(G', \psi'^{\mathsf{ini}}, \psi'^{\mathsf{tar}})$ of* Maxmin 2-CSP_{W_0} Reconfiguration *with alphabet size W_0 such that the following hold:*

- *(Perfect completeness) If $\mathsf{val}_G(\psi^{\mathsf{ini}} \leftrightsquigarrow \psi^{\mathsf{tar}}) = 1$, then $\mathsf{val}_{G'}(\psi'^{\mathsf{ini}} \leftrightsquigarrow \psi'^{\mathsf{tar}}) = 1$.*
- *(Soundness) If $\mathsf{val}_G(\psi^{\mathsf{ini}} \leftrightsquigarrow \psi^{\mathsf{tar}}) < 1 - \varepsilon$, then $\mathsf{val}_{G'}(\psi'^{\mathsf{ini}} \leftrightsquigarrow \psi'^{\mathsf{tar}}) < 1 - \kappa \cdot \varepsilon$.*

In particular, for every $\varepsilon \in (0,1)$ and $W \in \mathbb{N}$, $\mathcal{A}$ is a gap-preserving reduction from $\mathsf{Gap}_{1,1-\varepsilon}$ 2-CSP_W Reconfiguration *to* $\mathsf{Gap}_{1,1-\kappa\cdot\varepsilon}$ 2-CSP_{W_0} Reconfiguration.

The remainder of this section is organized as follows: Section 3.1 introduces and proves the reconfigurability of Hadamard codes, which will be applied to robustization of Maxmin 2-CSP Reconfiguration in Section 3.2. Subsequently, Section 3.3 composes the assignment tester of [7, 22] into Circuit SAT Reconfiguration, concluding the proof of Theorem 3.1.

3.1 Reconfigurability of Hadamard Codes

Here, we prove the reconfigurability of Hadamard codewords. A *reconfiguration sequence from f^{ini} to f^{tar}* over $\mathbb{F}_2^N$ is a sequence $(f^{(1)}, \dots, f^{(T)})$ such that $f^{(1)} = f^{\mathsf{ini}}$, $f^{(T)} = f^{\mathsf{tar}}$, and every two neighboring functions $f^{(t)}$ and $f^{(t+1)}$ differ in at most one bit.

▶ **Lemma 3.2** (Reconfigurability of Hadamard codes). *Let n be a positive integer at least 9, $\delta_0 \coloneqq \frac{1}{400}$ be a universal constant, and $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{F}_2^n$ be two distinct strings. Then, there exists a reconfiguration sequence $\Pi = (\mathsf{Had}(\boldsymbol{\alpha}), \dots, \mathsf{Had}(\boldsymbol{\beta}))$ from $\mathsf{Had}(\boldsymbol{\alpha})$ to $\mathsf{Had}(\boldsymbol{\beta})$ such that for every string $\boldsymbol{\gamma} \in \mathbb{F}_2^n \setminus \{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$ and every function $f \colon \mathbb{F}_2^n \to \mathbb{F}$ in Π,*

$$\min\Big\{\Delta(f, \mathsf{Had}(\boldsymbol{\alpha})), \Delta(f, \mathsf{Had}(\boldsymbol{\beta}))\Big\} \leqslant \frac{1}{4}, \tag{3.1}$$

$$\Delta(f, \mathsf{Had}(\boldsymbol{\gamma})) > \frac{1}{4} + \delta_0. \tag{3.2}$$

Before going to its proof, we remark that Lemma 3.2 *does not* hold if $n = 3$.

▶ **Observation 3.3** (∗). *For $n = 3$ and $\boldsymbol{\alpha} \neq \boldsymbol{\beta} \in \mathbb{F}_2^n$, let Π be a reconfiguration sequence from $\mathsf{Had}(\boldsymbol{\alpha})$ to $\mathsf{Had}(\boldsymbol{\beta})$ such that $\min\{\Delta(f, \mathsf{Had}(\boldsymbol{\alpha})), \Delta(f, \mathsf{Had}(\boldsymbol{\beta}))\} \leqslant \frac{1}{4}$ for every function f in Π. Then, Π contains a function f° such that $\Delta(f^\circ, \mathsf{Had}(\boldsymbol{\gamma})) \leqslant \frac{1}{4}$ for some $\boldsymbol{\gamma} \in \mathbb{F}_2^n \setminus \{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$.*

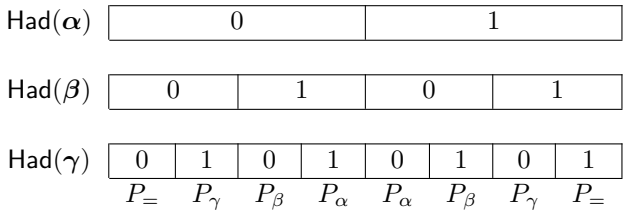


Figure 1 Illustration of $(P_\alpha, P_\beta, P_\gamma, P_=)$ for three distinct nonzero vectors $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{F}_2^n$.

To prove Lemma 3.2, we first analyze the partial sum of a random sequence consisting of an equal number of plus ones and minus ones.

▶ **Lemma 3.4** (∗). *Let $N > N_0 := 100$ be any positive integer, $\eta_0 := \frac{1}{100}$, and $\mathbf{a} = (a_1, \dots, a_{2N})$ be a random sequence made up of N plus ones and N minus ones obtained by applying a random permutation of $\mathfrak{S}_{2N}$ to $(\underbrace{+1, \dots, +1}_{N \text{ times}}, \underbrace{-1, \dots, -1}_{N \text{ times}})$. Then, the minimum k-partial sum over all $k \in [2N]$; i.e.,*

$$\operatorname*{argmin}_{1 \leqslant k \leqslant 2N} \sum_{1 \leqslant i \leqslant k} a_i = \operatorname*{argmin}_{1 \leqslant k \leqslant 2N} \sum_{k+1 \leqslant i \leqslant 2N} a_i, \tag{3.3}$$

is at most $-(1-\eta_0)N = -0.99N$ with probability at most 0.9^N.

Besides, given the Hadamard codewords of any three distinct strings, we partition their bits into four equal-sized groups.

▷ Claim 3.5 (∗). For three distinct vectors $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{F}_2^n$, the following hold:

$$\begin{aligned}
&\mathop{\mathbb{P}}_{\mathbf{x} \in \mathbb{F}_2^n}\Big[\langle \boldsymbol{\alpha}, \mathbf{x}\rangle \neq \langle \boldsymbol{\beta}, \mathbf{x}\rangle = \langle \boldsymbol{\gamma}, \mathbf{x}\rangle\Big] = \frac{1}{4}, \quad \mathop{\mathbb{P}}_{\mathbf{x} \in \mathbb{F}_2^n}\Big[\langle \boldsymbol{\beta}, \mathbf{x}\rangle \neq \langle \boldsymbol{\gamma}, \mathbf{x}\rangle = \langle \boldsymbol{\alpha}, \mathbf{x}\rangle\Big] = \frac{1}{4}, \\
&\mathop{\mathbb{P}}_{\mathbf{x} \in \mathbb{F}_2^n}\Big[\langle \boldsymbol{\gamma}, \mathbf{x}\rangle \neq \langle \boldsymbol{\alpha}, \mathbf{x}\rangle = \langle \boldsymbol{\beta}, \mathbf{x}\rangle\Big] = \frac{1}{4}, \quad \mathop{\mathbb{P}}_{\mathbf{x} \in \mathbb{F}_2^n}\Big[\langle \boldsymbol{\alpha}, \mathbf{x}\rangle = \langle \boldsymbol{\beta}, \mathbf{x}\rangle = \langle \boldsymbol{\gamma}, \mathbf{x}\rangle\Big] = \frac{1}{4}.
\end{aligned} \tag{3.4}$$

Using Lemma 3.4 and Claim 3.5, we now prove Lemma 3.2.

Proof of Lemma 3.2. Fix two strings $\boldsymbol{\alpha} \neq \boldsymbol{\beta} \in \mathbb{F}_2^n$ for $n \geqslant 9$. Let $D \subset \mathbb{F}_2^n$ be a set of strings on which $\mathsf{Had}(\boldsymbol{\alpha})$ and $\mathsf{Had}(\boldsymbol{\beta})$ disagree with each other; namely,

$$D := \Big\{ \mathbf{x} \in \mathbb{F}_2^n \;\Big|\; \langle \boldsymbol{\alpha}, \mathbf{x}\rangle \neq \langle \boldsymbol{\beta}, \mathbf{x}\rangle \Big\}. \tag{3.5}$$

The random subsum principle ensures $|D| = 2^{n-1}$ (cf. [3, Claim A.31]). Consider a random reconfiguration sequence $\Pi = (\mathsf{Had}(\boldsymbol{\alpha}), \dots, \mathsf{Had}(\boldsymbol{\beta}))$ obtained by the following procedure:

Random reconfiguration Π from Had(α) to Had(β).

1: $(\mathbf{x}_1, \dots, \mathbf{x}_{2^{n-1}}) \leftarrow$ a sequence obtained by applying a random permutation of $\mathfrak{S}_{2^{n-1}}$ to D.
2: **for** $i = 1$ **to** 2^{n-1} **do**
3: flip $\mathbf{x}_i{}^{\text{th}}$ entry of the current function.

Observe easily that any intermediate function of Π is always $\frac{1}{4}$-close to either $\mathsf{Had}(\boldsymbol{\alpha})$ or $\mathsf{Had}(\boldsymbol{\beta})$. Fix any string $\boldsymbol{\gamma} \in \mathbb{F}_2^n \setminus \{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$. We would like to show that with probability at least $1 - 0.9^{2^{n-2}}$, every function of Π is $\left(\frac{1}{4} + \delta_0\right)$-far from $\mathsf{Had}(\boldsymbol{\gamma})$; i.e.,

$$\Delta(\mathsf{Had}(\boldsymbol{\gamma}), \Pi) := \min_{f \in \Pi} \Delta(\mathsf{Had}(\boldsymbol{\gamma}), f) > \frac{1}{4} + \delta_0. \tag{3.6}$$

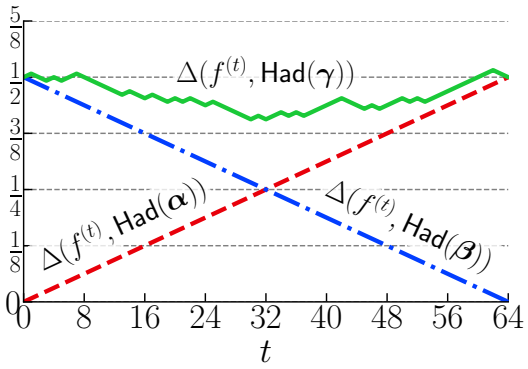


Figure 2 Plot of the distance from $f^{(t)}$ to $\mathsf{Had}(\boldsymbol{\alpha})$, $\mathsf{Had}(\boldsymbol{\beta})$, and $\mathsf{Had}(\boldsymbol{\gamma})$ for a random reconfiguration Π from $\mathsf{Had}(\boldsymbol{\alpha})$ to $\mathsf{Had}(\boldsymbol{\beta})$ described in the proof of Lemma 3.2.

By Claim 3.5, there exists a partition $(P_\alpha, P_\beta, P_\gamma, P_=)$ of $\mathbb{F}_2^n$ such that $|P_\alpha| = |P_\beta| = |P_\gamma| = |P_=| = 2^{n-2}$ and

$$\begin{aligned} &\langle \boldsymbol{\alpha}, \mathbf{x}\rangle \neq \langle \boldsymbol{\beta}, \mathbf{x}\rangle = \langle \boldsymbol{\gamma}, \mathbf{x}\rangle \text{ for all } \mathbf{x} \in P_\alpha, \quad \langle \boldsymbol{\beta}, \mathbf{x}\rangle \neq \langle \boldsymbol{\gamma}, \mathbf{x}\rangle = \langle \boldsymbol{\alpha}, \mathbf{x}\rangle \text{ for all } \mathbf{x} \in P_\beta, \\ &\langle \boldsymbol{\gamma}, \mathbf{x}\rangle \neq \langle \boldsymbol{\alpha}, \mathbf{x}\rangle = \langle \boldsymbol{\beta}, \mathbf{x}\rangle \text{ for all } \mathbf{x} \in P_\gamma, \quad \langle \boldsymbol{\alpha}, \mathbf{x}\rangle = \langle \boldsymbol{\beta}, \mathbf{x}\rangle = \langle \boldsymbol{\gamma}, \mathbf{x}\rangle \text{ for all } \mathbf{x} \in P_=. \end{aligned} \tag{3.7}$$

See also Figure 1. Here, we always have $P_\alpha \uplus P_\beta = D$ (though P_α and P_β themselves depend on $\boldsymbol{\gamma}$).

For any intermediate function $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ of Π, if its entry on P_α is flipped, its Hamming distance to $\mathsf{Had}(\boldsymbol{\gamma})$ must decrease by 1, whereas if its entry on P_β is flipped, its Hamming distance to $\mathsf{Had}(\boldsymbol{\gamma})$ must increase by 1; see also Figure 2. Since $|P_\alpha| = |P_\beta| = 2^{n-2} > 100$ and $\|\mathsf{Had}(\boldsymbol{\alpha}) - \mathsf{Had}(\boldsymbol{\gamma})\| = \|\mathsf{Had}(\boldsymbol{\beta}) - \mathsf{Had}(\boldsymbol{\gamma})\| = 2^{n-1}$, we can apply Lemma 3.4 with $N = 2^{n-2}$ to conclude that

$$\begin{aligned} &\mathop{\mathbb{P}}_{\Pi}\left[\min_{f\in\Pi}\|\mathsf{Had}(\boldsymbol{\gamma}) - \Pi\| \leqslant 2^{n-1} - 0.99N\right] \leqslant 0.9^N \\ \Longrightarrow\ &\mathop{\mathbb{P}}_{\Pi}\left[\Delta(\mathsf{Had}(\boldsymbol{\gamma}), \Pi) \leqslant \frac{1}{4} + \frac{1}{400}\right] \leqslant 0.9^{2^{n-2}}. \end{aligned}$$

Taking a union bound over all possible strings $\boldsymbol{\gamma} \in \mathbb{F}_2^n \setminus \{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$, we derive

$$\begin{aligned} &\mathop{\mathbb{P}}_{\Pi}\left[\exists \boldsymbol{\gamma} \notin \{\boldsymbol{\alpha}, \boldsymbol{\beta}\} \text{ s.t. } \Delta(\mathsf{Had}(\boldsymbol{\gamma}), \Pi) \leqslant \frac{1}{4} + \frac{1}{400}\right] \\ &\leqslant \sum_{\boldsymbol{\gamma}\notin\{\boldsymbol{\alpha},\boldsymbol{\beta}\}} \mathop{\mathbb{P}}_{\Pi}\left[\Delta(\mathsf{Had}(\boldsymbol{\gamma}), \Pi) \leqslant \frac{1}{4} + \frac{1}{400}\right] < 2^n \cdot 0.9^{2^{n-2}} < 1 \quad (\text{for all } n \geqslant 9). \end{aligned} \tag{3.8}$$

Consequently, the probabilistic method guarantees the existence of a reconfiguration sequence $\Pi = (\mathsf{Had}(\boldsymbol{\alpha}), \ldots, \mathsf{Had}(\boldsymbol{\beta}))$ that is entirely $\left(\frac{1}{4} + \delta_0\right)$-far from $\mathsf{Had}(\boldsymbol{\gamma})$ for every $\boldsymbol{\gamma} \notin \{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$. ◀

3.2 Robustization

Subsequently, we advance to *robustization* of Maxmin 2-CSP Reconfiguration, relying on the reconfigurability of Hadamard codes. For a system of Boolean circuits $\mathscr{C}$ and its two satisfying truth assignments $\sigma^{\mathsf{ini}}, \sigma^{\mathsf{tar}}\colon \mathbb{F}_2^N \to \mathbb{F}_2$, Circuit SAT Reconfiguration requires to decide the existence of a reconfiguration sequence from σ^{ini} to σ^{tar} over $\mathbb{F}_2^{\mathbb{F}_2^N}$ consisting only of satisfying truth assignments to $\mathscr{C}$.

▶ **Lemma 3.6** (Robustization). *There exists a polynomial-time algorithm that takes an instance* $(G, \psi^{\mathsf{ini}}, \psi^{\mathsf{tar}})$ *of* Maxmin 2-CSP$_W$ Reconfiguration *with alphabet size* $W \in \mathbb{N}$*, where* ψ^{ini} *and* ψ^{tar} *satisfy* G*, and then produces an instance* $(\mathscr{C}, \sigma^{\mathsf{ini}}, \sigma^{\mathsf{tar}})$ *of* Circuit SAT Reconfiguration*, where* $\mathscr{C} = (C_e)_{e \in E}$ *is a system of circuits and* σ^{ini} *and* σ^{tar} *satisfy* $\mathscr{C}$*, such that the following hold*:

- (*Perfect completeness*) *If* $\mathsf{val}_G(\psi^{\mathsf{ini}} \leftrightsquigarrow \psi^{\mathsf{tar}}) = 1$*, there exists a reconfiguration sequence from* σ^{ini} *to* σ^{tar} *made up of satisfying truth assignments to* $\mathscr{C}$*.*
- (*Soundness*) *If* $\mathsf{val}_G(\psi^{\mathsf{ini}} \leftrightsquigarrow \psi^{\mathsf{tar}}) < 1 - \varepsilon$*, any reconfiguration sequence from* σ^{ini} *to* σ^{tar} *includes assignment* σ° *such that for more than* ε*-fraction of edges* e *of* G*,* $\sigma^\circ|_{[\![e]\!]}$ *is* $\frac{\delta_0}{8}$*-far from any satisfying truth assignment to* C_e*, where* $\delta_0 = \frac{1}{400}$ *as in Lemma 3.2.*

Reduction. Our polynomial-time robustization of Maxmin 2-CSP$_W$ Reconfiguration into Circuit SAT Reconfiguration is described as follows. Let $(G, \psi^{\mathsf{ini}}, \psi^{\mathsf{tar}})$ be an instance of Maxmin 2-CSP$_W$ Reconfiguration, where $G = (V, E, \Sigma, \Pi)$ is a binary constraint graph, and ψ^{ini} and ψ^{tar} satisfy G. Without loss of generality, we can assume that $W = |\Sigma| = 2^n$ for some integer $n \geqslant 9$,[4] and we can identify $\mathbb{F}_2^n$ with Σ.

Consider replacing binary constraints of G by a system of circuits. We first specify a *truth assignment* to the entire circuit system by a function $\sigma \colon \mathbb{F}_2^n \times V \to \mathbb{F}_2$, which can be thought of as a concatenation of functions $\sigma_v \colon \mathbb{F}_2^n \to \mathbb{F}_2$ associated with each vertex $v \in V$. For vertex $v \in V$, let $[\![v]\!]$ denote the set of 2^n Boolean variables associated with v, and for edge $e = (v, w) \in E$, let $[\![e]\!] \coloneqq [\![v]\!] \uplus [\![w]\!]$.[5] By this representation, we can identify $\mathbb{F}_2^n \times V$ with $\biguplus_{v \in V} [\![v]\!]$. In particular, for edge $e = (v, w) \in E$, $\sigma|_{[\![e]\!]}$ is equal to $\sigma|_{[\![v]\!]} \circ \sigma|_{[\![w]\!]}$. For each edge $e = (v, w)$ of G and its constraint π_e, we define a circuit $C_e \colon ([\![v]\!] \to \mathbb{F}_2) \times ([\![w]\!] \to \mathbb{F}_2) \to \mathbb{F}_2$ (or equivalently, $C_e \colon \mathbb{F}_2^{[\![v]\!]} \times \mathbb{F}_2^{[\![w]\!]} \to \mathbb{F}_2$) that depends *only* on $\sigma|_{[\![e]\!]} = \sigma|_{[\![v]\!]} \circ \sigma|_{[\![w]\!]}$ such that $C_e(\sigma|_{[\![v]\!]} \circ \sigma|_{[\![w]\!]}) = 1$ if and only if

$$\begin{aligned} &\Delta(\sigma|_{[\![v]\!]}, \mathsf{Had}(\cdot)) \leqslant \frac{1}{4} \text{ and } \Delta(\sigma|_{[\![w]\!]}, \mathsf{Had}(\cdot)) \leqslant \frac{1}{4}, \\ &\forall \alpha, \beta \in \Sigma, \Delta(\sigma|_{[\![v]\!]}, \mathsf{Had}(\alpha)) \leqslant \frac{1}{4} + \frac{\delta_0}{2} \text{ and } \Delta(\sigma|_{[\![w]\!]}, \mathsf{Had}(\beta)) \leqslant \frac{1}{4} + \frac{\delta_0}{2} \implies (\alpha, \beta) \in \pi_e, \end{aligned} \tag{3.9}$$

where $\delta_0 = \frac{1}{400}$ as in Lemma 3.2. Note that each C_e has constant size and can be constructed in constant time since $n = \mathcal{O}(1)$. Consequently, we obtain a system of circuits, denoted $\mathscr{C} = (C_e)_{e \in E}$. Given a satisfying assignment $\psi \colon V \to \Sigma$ for G, we can construct a satisfying truth assignment $\sigma \colon \mathbb{F}_2^n \times V \to \mathbb{F}_2$ such that $\sigma|_{[\![v]\!]} \coloneqq \mathsf{Had}(\psi(v))$ for all $v \in V$. Constructing σ^{ini} from ψ^{ini} and σ^{tar} from ψ^{tar} according to this procedure, we obtain an instance $(\mathscr{C}, \sigma^{\mathsf{ini}}, \sigma^{\mathsf{tar}})$ of Circuit SAT Reconfiguration. Observe that the above reduction completes in polynomial time.

Proof of Lemma 3.6. We first prove the perfect completeness. It suffices to consider the case that ψ^{ini} and ψ^{tar} differ in exactly one vertex, say, $v^\star \in V$. Using Lemma 3.2, we obtain a reconfiguration sequence $(f^{(1)}, \dots, f^{(T)})$ from $\mathsf{Had}(\psi^{\mathsf{ini}}(v))$ to $\mathsf{Had}(\psi^{\mathsf{tar}}(v))$. Construct then a reconfiguration sequence $\boldsymbol{\sigma} = (\sigma^{(1)}, \dots, \sigma^{(T)})$ from σ^{ini} to σ^{tar} such that for all t, $\sigma^{(t)}|_{[\![w]\!]} \coloneqq \sigma^{\mathsf{ini}}|_{[\![w]\!]} = \sigma^{\mathsf{tar}}|_{[\![w]\!]}$ for all $w \neq v^\star$, and $\sigma^{(t)}|_{[\![v^\star]\!]} \coloneqq f^{(t)}$. For each edge $e = (v^\star, w)$ of G, any intermediate function $\sigma^{(t)}$ of $\boldsymbol{\sigma}$ satisfies the following:

[4] Otherwise, we can augment Σ by padding so that $|\Sigma| \geqslant 2^9$.
[5] Similar notations are used in [7].

- By Lemma 3.2, $\sigma^{(t)}|_{[\![v^\star]\!]}$ is $\frac{1}{4}$-close to $\mathsf{Had}(\psi^{\mathsf{ini}}(v))$ or $\mathsf{Had}(\psi^{\mathsf{tar}}(v))$, but $\left(\frac{1}{4}+\delta_0\right)$-far from $\mathsf{Had}(\gamma)$ for every $\gamma \notin \{\psi^{\mathsf{ini}}(v), \psi^{\mathsf{tar}}(v)\}$.
- $\sigma^{(t)}|_{[\![w]\!]}$ is equal to $\mathsf{Had}(\psi^{\mathsf{ini}}(w)) = \mathsf{Had}(\psi^{\mathsf{tar}}(w))$; i.e., it is $\left(\frac{1}{2}-o(1)\right)$-far from $\mathsf{Had}(\gamma)$ for every $\gamma \notin \{\psi^{\mathsf{ini}}(w), \psi^{\mathsf{tar}}(w)\}$.

Since $\{\psi^{\mathsf{ini}}(v^\star), \psi^{\mathsf{tar}}(v^\star)\} \times \{\psi^{\mathsf{ini}}(w), \psi^{\mathsf{tar}}(w)\} = \{(\psi^{\mathsf{ini}}(v^\star), \psi^{\mathsf{ini}}(w)), (\psi^{\mathsf{tar}}(v^\star), \psi^{\mathsf{tar}}(w))\} \subseteq \pi_e$, it turns out that $\sigma^{(t)}|_{[\![e]\!]}$ satisfies C_e, and thus every $\sigma^{(t)}$ in $\boldsymbol{\sigma}$ satisfies $\mathscr{C}$ entirely.

We then prove the soundness. Suppose $\mathsf{val}_G(\psi^{\mathsf{ini}} \leftrightsquigarrow \psi^{\mathsf{tar}}) < 1-\varepsilon$ and we are given a reconfiguration sequence $\boldsymbol{\sigma} = (\sigma^{(1)}, \dots, \sigma^{(T)})$ from σ^{ini} to σ^{tar}. Construct then a reconfiguration sequence $\boldsymbol{\psi} = (\psi^{(1)}, \dots, \psi^{(T)})$ from ψ^{ini} to ψ^{tar} such that $\psi^{(t)}(v)$ is defined as a value of Σ whose Hadamard codeword is closest to $\sigma^{(t)}|_{[\![v]\!]}$; namely,[6]

$$\psi^{(t)}(v) \coloneqq \operatorname*{argmin}_{\alpha\in\Sigma} \Delta(\sigma^{(t)}|_{[\![v]\!]}, \mathsf{Had}(\alpha)). \tag{3.10}$$

Since $\boldsymbol{\psi}$ is a valid reconfiguration sequence, there exists some $\psi^{(t)}$ that violates more than $\varepsilon\cdot|E|$ edges.

Hereafter, we denote $\psi \coloneqq \psi^{(t)}$ and $\sigma \coloneqq \sigma^{(t)}$ for notational simplicity. Suppose ψ violates edge $e=(v,w)$; i.e., $(\psi(v),\psi(w)) \notin \pi_e$. We would like to show that $\sigma|_{[\![e]\!]}$ is $\frac{\delta_0}{8}$-far from any satisfying truth assignment to C_e. Let $f\circ g\colon [\![e]\!] \to \mathbb{F}_2$ be a satisfying truth assignment to C_e. In particular, there exists a pair $(\alpha^\star,\beta^\star) \in \pi_e$ such that $\Delta(f, \mathsf{Had}(\alpha^\star)) \leqslant \frac{1}{4}$ and $\Delta(g, \mathsf{Had}(\beta^\star)) \leqslant \frac{1}{4}$. Observe now that "$f$ is $\left(\frac{1}{4}+\frac{\delta_0}{2}\right)$-far from $\mathsf{Had}(\psi(v))$" or "g is $\left(\frac{1}{4}+\frac{\delta_0}{2}\right)$-far from $\mathsf{Had}(\psi(w))$" because otherwise, $C_e(f\circ g)=0$.

Suppose first $\Delta(f, \mathsf{Had}(\psi(v))) > \frac{1}{4}+\frac{\delta_0}{2}$, implying that $\alpha^\star \neq \psi(v)$. Putting together, we have the following three inequalities in hand:

$$\Delta(f, \mathsf{Had}(\alpha^\star)) \leqslant \frac{1}{4} \qquad \text{by assumption,} \tag{3.11}$$

$$\Delta(f, \mathsf{Had}(\psi(v))) > \frac{1}{4}+\frac{\delta_0}{2} \qquad \text{by assumption,} \tag{3.12}$$

$$\Delta(\sigma|_{[\![v]\!]}, \mathsf{Had}(\psi(v))) \leqslant \Delta(\sigma|_{[\![v]\!]}, \mathsf{Had}(\alpha^\star)) \qquad \text{by construction of } \sigma|_{[\![v]\!]}. \tag{3.13}$$

Simple calculation using the triangle inequality derives

$$\begin{aligned} \Delta(f, \mathsf{Had}(\psi(v))) &\leqslant \Delta(f, \sigma|_{[\![v]\!]}) + \Delta(\sigma|_{[\![v]\!]}, \mathsf{Had}(\psi(v))) \\ &\leqslant \Delta(f, \sigma|_{[\![v]\!]}) + \Delta(\sigma|_{[\![v]\!]}, \mathsf{Had}(\alpha^\star)) \\ &\leqslant \Delta(f, \sigma|_{[\![v]\!]}) + \Delta(\sigma|_{[\![v]\!]}, f) + \Delta(f, \mathsf{Had}(\alpha^\star)) \\ &= 2\cdot\Delta(\sigma|_{[\![v]\!]}, f) + \Delta(f, \mathsf{Had}(\alpha^\star)) \end{aligned} \tag{3.14}$$

$$\implies 2\cdot\Delta(\sigma|_{[\![v]\!]}, f) \geqslant \underbrace{\Delta(f, \mathsf{Had}(\psi(v)))}_{>\frac{1}{4}+\frac{\delta_0}{2}} - \underbrace{\Delta(f, \mathsf{Had}(\alpha^\star))}_{\leqslant\frac{1}{4}} \tag{3.15}$$

$$\implies \Delta(\sigma|_{[\![v]\!]}, f) > \frac{\delta_0}{4}. \tag{3.16}$$

Consequently, $\sigma|_{[\![e]\!]}$ should be $\frac{\delta_0}{8}$-far from $f\circ g$.

Suppose next $\Delta(g, \mathsf{Had}(\psi(w))) > \frac{1}{4}+\frac{\delta_0}{2}$, implying that $\beta^\star \neq \psi(w)$. Similarly to the first case, we can show that $\Delta(\sigma|_{[\![w]\!]}, g) > \frac{\delta_0}{4}$, deriving that $\sigma|_{[\![e]\!]}$ is $\frac{\delta_0}{8}$-far from $f\circ g$. This completes the proof of the soundness. ◀

[6] Ties are broken according to any prefixed order of Σ.

Example 3.7 explains why the reconfigurability of Hadamard codes is needed, by using a slightly different definition of circuits that fails robustization.

▶ **Example 3.7.** For edge $e = (v, w)$, define a binary constraint $\pi_e := \{(\alpha_1, \beta_1), (\alpha_2, \beta_2)\} \subset \mathbb{F}_2^n \times \mathbb{F}_2^n$. Construct a circuit $\widetilde{C}_e \colon (\llbracket v \rrbracket \to \mathbb{F}_2) \times (\llbracket w \rrbracket \to \mathbb{F}_2) \to \mathbb{F}_2$ such that $\widetilde{C}_e(\sigma|_{\llbracket v \rrbracket} \circ \sigma|_{\llbracket w \rrbracket}) = 1$ if and only if

$$\begin{aligned} &\Delta(\sigma|_{\llbracket v \rrbracket}, \mathsf{Had}(\cdot)) \leqslant \frac{1}{4} \text{ and } \Delta(\sigma|_{\llbracket w \rrbracket}, \mathsf{Had}(\cdot)) \leqslant \frac{1}{4}, \\ &\forall \alpha, \beta \in \Sigma, \Delta(\sigma|_{\llbracket v \rrbracket}, \mathsf{Had}(\alpha)) \leqslant \frac{1}{4} \text{ and } \Delta(\sigma|_{\llbracket w \rrbracket}, \mathsf{Had}(\beta)) \leqslant \frac{1}{4} \implies (\alpha, \beta) \in \pi_e. \end{aligned} \tag{3.17}$$

Note that reconfiguring from (α_1, β_1) to (α_2, β_2) over $\Sigma \times \Sigma$ (not $\mathbb{F}_2^n \times \mathbb{F}_2^n$) must break π_e (at some point). Analogously, we might expect that any reconfiguration sequence from $\mathsf{Had}(\alpha_1) \circ \mathsf{Had}(\beta_1)$ to $\mathsf{Had}(\alpha_2) \circ \mathsf{Had}(\beta_2)$ over $\mathbb{F}_2^{\llbracket v \rrbracket} \times \mathbb{F}_2^{\llbracket w \rrbracket}$ includes a function that is $\Theta(1)$-far from any satisfying truth assignment to $\widetilde{C}_e$. Consider now the following reconfiguration:

Reconfiguration Π from $\mathsf{Had}(\alpha_1) \circ \mathsf{Had}(\beta_1)$ to $\mathsf{Had}(\alpha_2) \circ \mathsf{Had}(\beta_2)$.

1: $f :=$ a function $\frac{1}{4}$-close to both $\mathsf{Had}(\alpha_1)$ and $\mathsf{Had}(\alpha_2)$.
2: $g :=$ a function $\frac{1}{4}$-close to both $\mathsf{Had}(\beta_1)$ and $\mathsf{Had}(\beta_2)$.
3: change $\mathsf{Had}(\alpha_1)$ to f one by one.
4: change $\mathsf{Had}(\beta_1)$ to g one by one.
5: ▷ *obtain $f \circ g$.* ◁
6: change f to $\mathsf{Had}(\alpha_2)$ one by one.
7: change g to $\mathsf{Had}(\beta_2)$ one by one.

Changing particular two bits of $f \circ g$, we obtain $f^\star \circ g^\star$, which is $\left(\frac{1}{4} - \frac{1}{2^n}\right)$-close to $\mathsf{Had}(\alpha_1) \circ \mathsf{Had}(\beta_1)$, implying $\widetilde{C}_e(f^\star \circ g^\star) = 1$. Thus, $f \circ g$ is $\frac{1}{2^n}$-close to some satisfying truth assignment to $\widetilde{C}_e$. Similarly, every intermediate function of Π is $\frac{1}{2^n}$-close to some satisfying truth assignment to $\widetilde{C}_e$. ⌟

3.3 Composition of Assignment Testers

We are now ready to compose an assignment tester into Circuit SAT Reconfiguration to accomplish alphabet reduction of Maxmin 2-CSP Reconfiguration. Here, we recapitulate *assignment testers* [7, 8], a.k.a. *PCPs of proximity* [4], and refer to an explicit construction due to Dinur [7] and O'Donnell [22].[7]

▶ **Definition 3.8** ([8, 4])**.** An *assignment tester* over alphabet $\Sigma_0 \supset \mathbb{F}_2$ with *rejection rate* $\rho \in (0, 1)$ is an algorithm $\mathcal{P}$ that takes a circuit $\Phi \colon \mathbb{F}_2^X \to \mathbb{F}_2$ over Boolean variables X as input, and produces a binary constraint graph $G = (V = X \uplus Y, E, \Sigma_0, \Pi)$ over X and auxiliary variables Y such that the following hold for any truth assignment $\sigma \colon X \to \mathbb{F}_2$ for Φ:

- (Perfect completeness) If σ satisfies Φ, there exists an assignment $\tau \colon Y \to \Sigma_0$ such that $\mathsf{val}_G(\sigma \circ \tau) = 1$.
- (Soundness) If σ is δ-far from any satisfying truth assignment to Φ, for every assignment $\tau \colon Y \to \Sigma_0$, $\mathsf{val}_G(\sigma \circ \tau) < 1 - \rho \cdot \delta$. ⌟

[7] Note that an assignment tester of O'Donnell [22, Theorem 7.16] takes the form of verifiers, which can be represented as a binary constraint graph by a standard reduction from probabilistically checkable proofs to two-prover games, e.g., [9, 29].

▶ **Theorem 3.9** ([7, Theorem 5.1] and [22, Theorem 7.16]). *There exists an explicit construction of an assignment tester $\mathcal{P}$ with alphabet $\Sigma_0 = \mathbb{F}_2^3$ and rejection rate $\rho := \frac{1}{10{,}000}$.*

▶ **Proposition 3.10** (Composition). *There exist universal constants $\widetilde{W}_0 := 8$ and $\widetilde{\kappa} := \frac{\delta_0^2 \rho^2}{64} \in (0,1)$, and a polynomial-time algorithm that takes an instance $(G, \psi^{\mathsf{ini}}, \psi^{\mathsf{tar}})$ of* Maxmin 2-CSP$_W$ Reconfiguration *with alphabet size $W \in \mathbb{N}$, where ψ^{ini} and ψ^{tar} satisfy G, and then produces an instance $(G', \psi'^{\mathsf{ini}}, \psi'^{\mathsf{tar}})$ of* Maxmin 4-CSP$_{\widetilde{W}_0}$ Reconfiguration *with alphabet size $\widetilde{W}_0$, where ψ'^{ini} and ψ'^{tar} satisfy G', such that the following hold:*

- *(Perfect completeness) If $\mathsf{val}_G(\psi^{\mathsf{ini}} \leftrightsquigarrow \psi^{\mathsf{tar}}) = 1$, then $\mathsf{val}_{G'}(\psi'^{\mathsf{ini}} \leftrightsquigarrow \psi'^{\mathsf{tar}}) = 1$.*
- *(Soundness) If $\mathsf{val}_G(\psi^{\mathsf{ini}} \leftrightsquigarrow \psi^{\mathsf{tar}}) < 1 - \varepsilon$, then $\mathsf{val}_{G'}(\psi'^{\mathsf{ini}} \leftrightsquigarrow \psi'^{\mathsf{tar}}) < 1 - \widetilde{\kappa} \cdot \varepsilon$.*

Reduction. We now describe a polynomial-time reduction from Circuit SAT Reconfiguration introduced in the previous subsection to Maxmin 4-CSP$_8$ Reconfiguration. Let $(\mathscr{C}, \sigma^{\mathsf{ini}}, \sigma^{\mathsf{tar}})$ be an instance of Circuit SAT Reconfiguration obtained by applying Lemma 3.6 to an instance $(G, \psi^{\mathsf{ini}}, \psi^{\mathsf{ini}})$ of Maxmin 2-CSP$_W$ Reconfiguration. Here, $\mathscr{C} = (C_e)_{e \in E}$ is a system of circuits over Boolean variables $\mathbb{F}_2^n \times V$, associated with underlying graph (V, E), and σ^{ini} and σ^{tar} entirely satisfy $\mathscr{C}$.

Running the assignment tester $\mathcal{P}$ of Theorem 3.9 on each circuit $C_e \colon \mathbb{F}_2^{[\![e]\!]} \to \mathbb{F}_2$ for edge $e \in E$ produces a binary constraint graph $G_e = (V_e = [\![e]\!] \uplus Y_e, E_e, \Sigma_0, \widetilde{\Pi}_e = (\widetilde{\pi}_{\widetilde{e}})_{\widetilde{e} \in E_e})$, where Y_e is the set of auxiliary variables and $|\Sigma_0| = 8$. Create a pair of copies of G_e "sharing" $[\![e]\!]$, denoted G_e^1 and G_e^2; namely,

$$G_e^1 := (V_e^1 = [\![e]\!] \uplus Y_e^1, E_e^1, \Sigma_0, \widetilde{\Pi}_e^1), \tag{3.18}$$

$$G_e^2 := (V_e^2 = [\![e]\!] \uplus Y_e^2, E_e^2, \Sigma_0, \widetilde{\Pi}_e^2). \tag{3.19}$$

We then "superimpose" G_e^1 and G_e^2 to obtain a 4-ary constraint graph $G'_e = (V'_e, E'_e, \Sigma_0, \Pi'_e = (\pi'_{(\widetilde{e}_1, \widetilde{e}_2)})_{(\widetilde{e}_1, \widetilde{e}_2) \in E_e})$, where

$$\begin{aligned} V'_e &:= [\![e]\!] \uplus Y_e^1 \uplus Y_e^2, \text{ and } E'_e := E_e^1 \times E_e^2, \\ \pi'_{(\widetilde{e}_1, \widetilde{e}_2)} &:= \widetilde{\pi}_{\widetilde{e}_1} \times \widetilde{\pi}_{\widetilde{e}_2} = \left\{ (\alpha_1, \beta_1, \alpha_2, \beta_2) \in \Sigma^4 \;\middle|\; (\alpha_1, \beta_1) \in \widetilde{\pi}_{\widetilde{e}_1} \vee (\alpha_2, \beta_2) \in \widetilde{\pi}_{\widetilde{e}_2} \right\} \\ &\qquad \text{for all } (\widetilde{e}_1, \widetilde{e}_2) \in E_e^1 \times E_e^2. \end{aligned} \tag{3.20}$$

Note that each pair of edges from E_e^1 and E_e^2 forms a hyperedge of G'_e, which would be satisfied if so is either of the two edges. We can safely assume that E'_e has the same size for all $e \in E$.

Finally, the new 4-ary constraint graph $G' = (V', E', \Sigma_0, \Pi')$ is defined as follows:

$$\begin{aligned} V' &:= \bigcup_{e \in E} V'_e = \left(\biguplus_{v \in V} [\![v]\!] \right) \uplus \left(\biguplus_{e \in E} Y_e^1 \uplus Y_e^2 \right), \\ E' &:= \biguplus_{e \in E} E'_e \text{ and } \Pi' := \biguplus_{e \in E} \Pi'_e. \end{aligned} \tag{3.21}$$

For any satisfying truth assignment $\sigma \colon \biguplus_{v \in V} [\![v]\!] \to \mathbb{F}_2$ of $\mathscr{C}$, consider an assignment $\psi' \colon V' \to \Sigma_0$ such that $\psi'|_{[\![v]\!]} := \sigma|_{[\![v]\!]}$ for all $v \in V$ and $\psi'|_{Y_e^1} = \psi'|_{Y_e^2} = \tau_e$ for all $e \in E$, where $\tau_e \colon Y_e \to \Sigma_0$ is an assignment to auxiliary variables Y_e such that $\sigma|_{[\![e]\!]} \circ \tau_e$ satisfies G'_e, whose existence is guaranteed by Definition 3.8. Observe easily that ψ' satisfies G'. Constructing ψ'^{ini} from σ^{ini} and ψ'^{tar} from σ^{tar} according to this procedure, we obtain an instance $(G', \psi'^{\mathsf{ini}}, \psi'^{\mathsf{tar}})$ of Maxmin 4-CSP$_8$ Reconfiguration, completing the reduction.

Proof of Proposition 3.10. Recall that $(G, \psi^{\mathsf{ini}}, \psi^{\mathsf{tar}})$ is an instance of Maxmin 2-CSP$_W$ Reconfiguration, $(\mathscr{C}, \sigma^{\mathsf{ini}}, \sigma^{\mathsf{tar}})$ is an instance of Circuit SAT Reconfiguration obtained by applying Lemma 3.6, and $(G', \psi'^{\mathsf{ini}}, \psi'^{\mathsf{tar}})$ is an instance of Maxmin 4-CSP$_8$ Reconfiguration obtained by composing the assignment tester [7] as described above.

We first prove the perfect completeness. By Lemma 3.6, it suffices to consider the case that σ^{ini} and σ^{tar} differ in exactly one variable, say, $(\mathbf{x}, v^\star) \in \mathbb{F}_2^n \times V$. Consider a reconfiguration sequence $\boldsymbol{\psi}'$ from ψ'^{ini} to ψ'^{tar} obtained by the following procedure:

Reconfiguration $\boldsymbol{\psi}'$ from ψ'^{ini} to ψ'^{tar}.

1: **for all** edge $e = (v^\star, w) \in E$ **do**
2: let $\tau_e^{\mathsf{tar}} \colon Y_e \to \Sigma_0$ be assignment such that $\sigma^{\mathsf{tar}}|_{[\![e]\!]} \circ \tau_e^{\mathsf{tar}}$ satisfies G_e.
3: change the entries on Y_e^1 to τ_e^{tar} one by one.
4: flip $\mathbf{x}^{\text{th}}$ entry of $[\![v^\star]\!]$.
5: **for all** edge $e = (v^\star, w) \in E$ **do**
6: change the entries on Y_e^2 to τ_e^{tar} one by one.

Observe easily that for any edge $e = (v^\star, w) \in E$, either of G_e^1 or G_e^2 is entirely satisfied by any intermediate assignment, implying that $\mathsf{val}_{G'}(\boldsymbol{\psi}') = 1$, as desired.

We then prove the soundness. Suppose we are given a reconfiguration sequence $\boldsymbol{\psi}' = (\psi'^{(1)}, \dots, \psi'^{(T)})$ from ψ'^{ini} to ψ'^{tar} such that $\mathsf{val}_{G'}(\boldsymbol{\psi}') = \mathsf{val}_{G'}(\psi'^{\mathsf{ini}} \leftrightsquigarrow \psi'^{\mathsf{tar}})$. Consider a reconfiguration sequence $\boldsymbol{\sigma} = (\sigma^{(1)}, \dots, \sigma^{(T)})$ such that $\sigma^{(t)} \coloneqq \psi'^{(t)}|_{\biguplus_{v \in V} [\![v]\!]}$ for all t. Since $\boldsymbol{\sigma}$ is a valid reconfiguration sequence from σ^{ini} to σ^{tar}, by Lemma 3.6, there exists some $\sigma^{(t)}$ such that for more than ε-fraction of edges e of G, $\sigma^{(t)}|_{[\![e]\!]} = \psi'^{(t)}|_{[\![e]\!]}$ is $\frac{\delta_0}{8}$-far from any satisfying truth assignment to C_e. Let $F \subset E$ be the set of such edges of G; note that $|F| \geqslant \varepsilon |E|$. By Theorem 3.9, $\psi'^{(t)}$ violates more than $\frac{\delta_0 \rho}{8}$-fraction of edges of each G_e^1 and G_e^2 for any $e \in F$. Since $\psi'^{(t)}$ violates hyperedge $(\widetilde{e}_1, \widetilde{e}_2) \in E_e^1 \times E_e^2$ if and only if it violates $\widetilde{e}_1 \in E_e^1$ with respect to $\widetilde{\Pi}_e^1$ and $\widetilde{e}_2 \in E_e^2$ with respect to $\widetilde{\Pi}_e^2$ *simultaneously*, there are more than $\left(\frac{\delta_0 \rho}{8}\right)^2$-fraction of hyperedges of G'_e that are violated by $\psi'^{(t)}$; i.e., $1 - \mathsf{val}_{G_e}(\psi'^{(t)}) > \frac{\delta_0^2 \rho^2}{64}$. Consequently, we derive

$$\begin{aligned}
1 - \mathsf{val}_{G'}(\boldsymbol{\psi}') &\geqslant 1 - \mathsf{val}_{G'}(\psi'^{(t)}) \\
&= \frac{1}{|E|} \sum_{e \in E} \left(1 - \mathsf{val}_{G'_e}(\psi'^{(t)})\right) \quad \text{(since every } E_e \text{ has the same size)} \\
&\geqslant \frac{1}{|E|} \sum_{e \in F} \left(1 - \mathsf{val}_{G'_e}(\psi'^{(t)})\right) \qquad (3.22) \\
&> \frac{|F|}{|E|} \frac{\delta_0^2 \rho^2}{64} > \varepsilon \cdot \underbrace{\frac{\delta_0^2 \rho^2}{64}}_{=\widetilde{\kappa}},
\end{aligned}$$

implying that $\mathsf{val}_{G'}(\psi'^{\mathsf{ini}} \leftrightsquigarrow \psi'^{\mathsf{tar}}) = \mathsf{val}_{G'}(\boldsymbol{\psi}') < 1 - \widetilde{\kappa} \cdot \varepsilon$, as desired. ◀

Proof of Theorem 3.1. Our construction of alphabet reduction for Maxmin 2-CSP Reconfiguration follows from Lemma 3.6 and Proposition 3.10 and a gap-preserving reduction [26, Lemma 5.4] (which is in fact approximation-preserving) from Gap$_{1,1-\varepsilon}$ 4-CSP$_{\widetilde{W}_0}$ Reconfiguration to Gap$_{1,1-\frac{\varepsilon}{4}}$ 2-CSP$_{W_0}$ Reconfiguration, where $W_0 = \left(\frac{\widetilde{W}_0(\widetilde{W}_0+1)}{2}\right)^4 = 36^4$. The value of κ in Theorem 3.1 should be $\frac{\widetilde{\kappa}}{4} = \frac{\delta_0^2 \rho^2}{256} = \frac{1}{256 \cdot 400^2 \cdot 10{,}000^2} = \frac{1}{8{,}000^4}$. ◀

4 Conclusions

We presented Dinur's style alphabet reduction [7] for Maxmin 2-CSP Reconfiguration, which now makes both the degree of inapproximability and alphabet size oblivious to the soundness error of the PCRP system [14, 17]. The main ingredient of its construction is the *reconfigurability of Hadamard codes*, which may be of independent interest and have further applications. We leave some open questions:

- **(Question 1).** Can we prove RIH [23] by Dinur's style gap amplification [7]? As discussed in Section 1.4, an approximation-preserving version for degree reduction and gap amplification of Maxmin 2-CSP Reconfiguration [23, 26] seems mandatory.
- **(Question 2).** Can we derive more meaningful inapproximability factors? Alas, we acknowledge that the current inapproximability factor is so small as to be almost meaningless in practice.
- **(Question 3).** Given the reconfigurability of Hadamard codes (Lemma 3.2), it is natural to ask that of other error-correcting codes: One may say that an error-correcting code enc is (δ, μ)-*reconfigurable* if for any $\boldsymbol{\alpha} \neq \boldsymbol{\beta}$, there exists a reconfiguration sequence from $\mathsf{enc}(\boldsymbol{\alpha})$ to $\mathsf{enc}(\boldsymbol{\beta})$ such that every function in it is
  - δ-close to either $\mathsf{enc}(\boldsymbol{\alpha})$ or $\mathsf{enc}(\boldsymbol{\beta})$, and
  - $(\delta + \mu)$-far from $\mathsf{enc}(\boldsymbol{\gamma})$ for every $\boldsymbol{\gamma} \neq \boldsymbol{\alpha}, \boldsymbol{\beta}$.

  Is there any such reconfigurable error-correcting code? Also, is there any general composition scheme for PCRPs [14]?

References

1 Dimitris Achlioptas, Amin Coja-Oghlan, and Federico Ricci-Tersenghi. On the solution-space geometry of random constraint satisfaction problems. *Random Struct. Algorithms*, 38(3):251–268, 2011.

2 Noga Alon. Explicit expanders of every degree and size. *Comb.*, 41(4):447–463, 2021.

3 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

4 Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006.

5 Amey Bhangale, Prahladh Harsha, Orr Paradise, and Avishay Tal. Rigid matrices from rectangular PCPs or: Hard claims have complex proofs. In *FOCS*, pages 858–869, 2020.

6 Nicolas Bousquet, Amer E. Mouawad, Naomi Nishimura, and Sebastian Siebertz. A survey on the parameterized complexity of the independent set and (connected) dominating set reconfiguration problems. *CoRR*, abs/2204.10526, 2022. `arXiv:2204.10526`.

7 Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):12, 2007.

8 Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP theorem. *SIAM J. Comput.*, 36(4):975–1024, 2006.

9 Lance Fortnow, John Rompel, and Michael Sipser. On the power of multi-prover interactive protocols. *Theor. Comput. Sci.*, 134(2):545–557, 1994.

10 David Gamarnik. The overlap gap property: A topological barrier to optimizing over random structures. *Proc. Natl. Acad. Sci. U.S.A.*, 118(41):e2108492118, 2021.

11 David Gamarnik and Madhu Sudan. Limits of local algorithms over sparse random graphs. *Ann. Probab.*, 45(4):2353–2376, 2017.

12 Johan Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Math.*, 182:105–142, 1999.

13 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

14 Shuichi Hirahara and Naoto Ohsaka. Probabilistically checkable reconfiguration proofs and inapproximability of reconfiguration problems. In *STOC*, 2024. to appear.

15 Takehiro Ito and Erik D. Demaine. Approximability of the subset sum reconfiguration problem. *J. Comb. Optim.*, 28(3):639–654, 2014.

16 Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theor. Comput. Sci.*, 412(12-14):1054–1065, 2011.

17 Karthik C. S. and Pasin Manurangsi. On inapproximability of reconfiguration problems: PSPACE-hardness and some tight NP-hardness results. *CoRR*, abs/2312.17140, 2023. `arXiv:2312.17140`.

18 Marc Mézard, Thierry Mora, and Riccardo Zecchina. Clustering of solutions in the random satisfiability problem. *Phys. Rev. Lett.*, 94(19):197205, 2005.

19 Sidhanth Mohanty, Ryan O'Donnell, and Pedro Paredes. Explicit near-Ramanujan graphs of every degree. *SIAM J. Comput.*, 51(3):STOC20–1–STOC20–23, 2021.

20 C. M. Mynhardt and S. Nasserasr. Reconfiguration of colourings and dominating sets in graphs. In *50 years of Combinatorics, Graph Theory, and Computing*, pages 171–191. CRC Press, 2019.

21 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018.

22 Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

23 Naoto Ohsaka. Gap preserving reductions between reconfiguration problems. In *STACS*, pages 49:1–49:18, 2023.

24 Naoto Ohsaka. On approximate reconfigurability of label cover. *CoRR*, abs/2304.08746, 2023. `arXiv:2304.08746`.

25 Naoto Ohsaka. Alphabet reduction for reconfiguration problems. *CoRR*, abs/2402.10627, 2024. `arXiv:2402.10627`.

26 Naoto Ohsaka. Gap amplification for reconfiguration problems. In *SODA*, pages 1345–1366, 2024.

27 Naoto Ohsaka and Tatsuya Matsuoka. Reconfiguration problems on submodular functions. In *WSDM*, pages 764–774, 2022.

28 Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3):425–440, 1991.

29 Jaikumar Radhakrishnan and Madhu Sudan. On Dinur's proof of the PCP theorem. *Bull. Am. Math. Soc.*, 44(1):19–61, 2007.

30 Ran Raz. A parallel repetition theorem. *SIAM J. Comput.*, 27(3):763–803, 1998.

31 Jan van den Heuvel. The complexity of change. In *Surveys in Combinatorics 2013*, volume 409, pages 127–160. Cambridge University Press, 2013.

32 Alexander S. Wein. Optimal low-degree hardness of maximum independent set. *Math. Stat. Learn.*, 4(3/4):221–251, 2021.

Delineating Half-Integrality of the Erdős-Pósa Property for Minors: The Case of Surfaces

Christophe Paul
LIRMM, Univ Montpellier, CNRS, Montpellier, France

Evangelos Protopapas
LIRMM, Univ Montpellier, CNRS, Montpellier, France

Dimitrios M. Thilikos
LIRMM, Univ Montpellier, CNRS, Montpellier, France

Sebastian Wiederrecht
Discrete Mathematics Group, Institute for Basic Science, Daejeon, South Korea

Abstract

In 1986 Robertson and Seymour proved a generalization of the seminal result of Erdős and Pósa on the duality of packing and covering cycles: A graph has the *Erdős-Pósa property* for minors if and only if it is planar. In particular, for every non-planar graph H they gave examples showing that the Erdős-Pósa property does not hold for H. Recently, Liu confirmed a conjecture of Thomas and showed that every graph has the *half-integral* Erdős-Pósa property for minors. Liu's proof is non-constructive and to this date, with the exception of a small number of examples, no constructive proof is known.

In this paper, we initiate the delineation of the half-integrality of the Erdős-Pósa property for minors. We conjecture that for every graph H, there exists a *unique* (up to a suitable equivalence relation on graph parameters) graph parameter EP_H such that H has the Erdős-Pósa property in a minor-closed graph class $\mathcal{G}$ if and only if $\sup\{\mathsf{EP}_H(G) \mid G \in \mathcal{G}\}$ is finite. We prove this conjecture for the class $\mathcal{H}$ of Kuratowski-connected shallow-vortex minors by showing that, for every non-planar $H \in \mathcal{H}$, the parameter $\mathsf{EP}_H(G)$ is *precisely* the maximum order of a Robertson-Seymour counterexample to the Erdős-Pósa property of H which can be found as a minor in G. Our results are constructive and imply, for the first time, parameterized algorithms that find either a packing, or a cover, or one of the Robertson-Seymour counterexamples, certifying the existence of a half-integral packing for the graphs in $\mathcal{H}$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Graph theory

Keywords and phrases Erdős-Pósa property, Erdős-Pósa pair, Graph parameters, Graph minors, Universal obstruction, Surface containment

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.114

Category Track A: Algorithms, Complexity and Games

Funding *Christophe Paul*: Supported by the French-German Collaboration ANR/DFG Project UTMA (ANR-20-CE92-0027).
Evangelos Protopapas: Supported by the French-German Collaboration ANR/DFG Project UTMA (ANR-20-CE92-0027).
Dimitrios M. Thilikos: Supported by the French-German Collaboration ANR/DFG Project UTMA (ANR-20-CE92-0027) and the MEAE and the MESR via the Franco-Norwegian project PHC Aurora project n. 51260WL (2024).
Sebastian Wiederrecht: Supported by the Institute for Basic Science (IBS-R029-C1).

1 Introduction

In 1965 Erdős and Pósa published a paper [9] proving the following min-max duality theorem.

For every positive integer k and every graph G, either G contains k pairwise vertex-disjoint cycles, or there exists a set $S \subseteq V(G)$ with $|S| = \mathcal{O}(k \cdot \log(k))$ such that $G - S$ has no cycles.

This result has since become central in both graph theory and algorithm design [36, 4, 21, 45, 26]. A collection of pairwise vertex-disjoint cycles is called a (cycle) *packing*, while a set S as above is commonly referred to as a (cycle) *cover* or *transversal*. In a more general context, one may consider any family $\mathcal{M}$ of graphs and define $\mathsf{pack}_{\mathcal{M}}(G)$ to be the largest size of a *packing* of members of $\mathcal{M}$ in G, while $\mathsf{cover}_{\mathcal{M}}(G)$ is the minimum size of a set $S \subseteq V(G)$ such that $G - S$ contains[1] no member of $\mathcal{M}$. Clearly $\mathsf{pack}_{\mathcal{M}}(G) \leq \mathsf{cover}_{\mathcal{M}}(G)$. We say that $\mathcal{M}$ has the *Erdős-Pósa property* (EP-property) in a graph class $\mathcal{G}$ if there exists a function f such that, for every $G \in \mathcal{G}$, it holds that $\mathsf{cover}_{\mathcal{M}}(G) \leq f(\mathsf{pack}_{\mathcal{M}}(G))$.

If we now fix some graph H and select $\mathcal{M}_H$ to be the class of all graphs containing H as a minor, we enter the realm of the *Graph Minors Series* of Robertson and Seymour. In Graph Minors V. [36], as an implication of their min-max duality between the treewidth of a graph and its largest grid-minor, they prove that

For every graph H, $\mathcal{M}_H$ has the EP-property in the class of all graphs if and only if H is planar. (1)

The tools and ideas of Erdős-Pósa-type dualities have since found many applications and interpretations [34, 27, 18, 3, 12, 33]. Moreover, the study of Erdős-Pósa dualities has led to important advances in structural graph theory. As an example, the proof for the directed version of Erdős and Pósa's result [35], known as *Younger's Conjecture* has paved the way for proving the *Directed Grid Theorem* [24].

Half-integral Erdős-Pósa. We call a collection $\mathcal{C}$ of subgraphs of G a *half-integral packing* of $\mathcal{M}$ in G if every graph in $\mathcal{C}$ belongs to $\mathcal{M}$ and no vertex of G belongs to more than two of them. We define $\frac{1}{2}\text{-}\mathsf{pack}_{\mathcal{M}}(G)$ to be the maximum size of such a half-integral packing. Accordingly, $\mathcal{M}$ has the $\frac{1}{2}$EP-property in a graph class $\mathcal{G}$ if there exists a function f such that, for every $G \in \mathcal{G}$, it holds that $\mathsf{cover}_{\mathcal{M}}(G) \leq f(\frac{1}{2}\text{-}\mathsf{pack}_{\mathcal{M}}(G))$.

Attempting to generalize Robertson and Seymour's seminal result on planar graphs, Robin Thomas conjectured the following relaxation of the EP-property (see [22, 26]).

For every graph H, $\mathcal{M}_H$ has the $\frac{1}{2}$EP-property in the class of all graphs. (2)

The above conjecture was recently proven by Liu [26]. As before, it is apparent from the definition that $\frac{1}{2}\text{-}\mathsf{pack}_{\mathcal{M}}(G) \leq 2 \cdot \mathsf{cover}_{\mathcal{M}}(G)$. Hence, Liu's theorem reveals a min-max duality between half-integral packing and covering in all graphs. Moreover, it is a consequence of the Graph Minors Theorem [37] that for every graph H and every graph parameter $\mathsf{p} \in \{\mathsf{pack}_{\mathcal{M}_H}, \frac{1}{2}\text{-}\mathsf{pack}_{\mathcal{M}_H}, \mathsf{cover}_{\mathcal{M}_H}\}$ one can decide in time $f_{H,\mathsf{p}}(k)|V(G)|^3$ if $\mathsf{p}(G) \geq k$ (or $\mathsf{p}(G) \leq k$ in the case where $\mathsf{p} = \mathsf{cover}_{\mathcal{M}_H}$) [10] for some function $f_{H,\mathsf{p}}$.

In light of the above results, it appears that the story of the Erdős-Pósa property in the regime of graph minors, from both a structural and an algorithmic perspective, is quite complete. However, we should stress the following two points.

First: The algorithm from [10] is inherently *non-constructive*. Indeed, while for $\mathsf{pack}_{\mathcal{M}_H}$ and $\mathsf{cover}_{\mathcal{M}_H}$ constructive algorithms are known [40, 39, 23], with the exception of some small special cases [22], *no* such results exist for $\frac{1}{2}\text{-}\mathsf{pack}_{\mathcal{M}_H}$, not even approximation algorithms.

[1] At this point we consider containment to be defined through the subgraph relation.

Second: Let $\mathcal{C}$ be a graph class and let p be a graph parameter. We say that p is *bounded* in $\mathcal{G}$ if there exists $c \in \mathbb{N}$ such that, for every $G \in \mathcal{G}$, it holds that $\mathsf{p}(G) \leq c$. The proof of the "if" direction of (1) was based on the fact that, for every H, $\mathcal{M}_H$ has the EP-property in every graph class of bounded treewidth. This leads to the following *intermediate* question: *For which graph parameters* p *it holds that* $\mathcal{M}_H$ *has the EP-property in every class where* p *is bounded?* To be specific, if we fix some graph H, is it possible to find a graph parameter EP_H such that $\mathcal{M}_H$ has the EP-property in some minor-closed[2] graph class $\mathcal{G}$ *if and only if* EP_H is bounded in $\mathcal{G}$? Indeed, we conjecture that for every graph H, such a graph parameter exists and *precisely* delineates the half-integrality of the Erdős-Pósa property of $\mathcal{M}_H$.

▶ **Conjecture 1.** *For every graph H, there exists a minor-monotone graph parameter EP_H such that $\mathcal{M}_H$ has the Erdős-Pósa property in a minor-closed graph class $\mathcal{G}$ if and only if EP_H is bounded in $\mathcal{G}$.*

Notice that for any planar graph P, we can simply set EP_P to be the constant zero-function and thus, Conjecture 1 trivially holds for all planar graphs, because of (1). However, for non-planar graphs, the existence of such a parameter does not follow from any known results. Even if EP_H would exist for some particular non-planar graph H, it would be desirable to have some constructive, and ideally canonical, characterization of EP_H. That is, we aim at a description of EP_H that allows for algorithmic applications.

There are reasons to believe that EP_H exists and moreover has some canonical representation. It has recently been shown in [31], that this assertion is tied to the conjecture that graphs are ω^2-well quasi ordered by minors, which is a wide open question in order theory (see the classic result of Thomas in [44] for the most advanced result on this conjecture).

The contribution of this paper is *resolving* Conjecture 1 for an infinite family of non-planar graphs. Moreover, our results are constructive and provide a canonical representation of EP_H yielding parameterized approximation algorithms[3] for $\frac{1}{2}$-pack_H for any H in our family.

1.1 The threshold of half-integrality

In Graph Minors V [36], towards proving the "only if" direction of (1), Robertson and Seymour gave counterexamples of graphs where non-planar graphs cannot have the EP-property. Let us investigate such an example for the graph K_5. One may embed K_5 in both the projective plane and the torus, but it is impossible to have two disjoint drawings of K_5 in either of them.

Consider the two graphs in the middle of Figure 1 and notice that the number of cycles and paths can be scaled. We call the infinite sequences defined by such "scalable graphs" *parametric graphs*[4]. These parametric graphs are the *handle grid* $\mathcal{H}$ and the *cross-cap grid* $\mathcal{C}$ and represent the torus and the projective plane respectively. None of them contains two disjoint copies of graphs from $\mathcal{M}_{K_5}$, both have a half-integral packing of $\Omega(k)$ members of $\mathcal{M}_{K_5}$, and any minimum-size cover of all $\mathcal{M}_{K_5}$ has $\Omega(k)$ vertices.

The seminal theorem of Reed [34] on the $\frac{1}{2}$EP-property of odd cycles exhibits exactly this kind of behaviour. Reed showed that odd cycles have the EP-property in every *odd-minor*[5]-closed graph class excluding an *Escher-wall*, while the Escher-wall itself is a counterexample

[2] A graph class is *minor-closed* if it contains all minors of its graphs.
[3] This means that our algorithms run in time $f(k) \cdot |V(G)|^{\mathcal{O}(1)}$ for some computable function f where k is the size of the half-integral packing we are looking for.
[4] We postpone the formal definition of parametric graphs to a later point. See Section 2.
[5] *Odd-minors* are a variant of the minor relation that preserves the parity of cycles. For example, bipartite graphs are exactly the K_3-odd-minor-free graphs. We refer the interested reader to [14] for a formal definition.

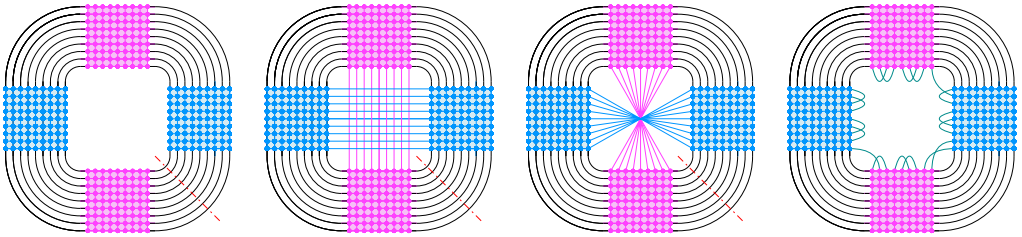


Figure 1 The parametric graphs representing the annulus grid $\mathcal{A}_k$, the handle grid $\mathcal{H}_k$, the cross-cap grid $\mathcal{C}_k$, and the shallow-vortex grid $\mathcal{V}_k$.

to the EP-property of odd cycles. Here, the k-Escher-wall is obtained by taking exactly the bipartite graphs from the parametric graph $\mathcal{C}$, representing the projective plane, and subdividing each of the "crossing" edges once. The result is a non-bipartite graph where every odd cycle must use an odd number of these subdivided edges. In the realm of odd-minors, this establishes a positive instance of Conjecture 1: pick EP_{K_3} as the maximum k for which G contains the k-Escher-wall as an odd minor.

It is tempting to suspect that Reed's strategy can apply for the Erdős-Pósa property for minors. That is, for K_5, the two parametric graphs $\mathcal{H}$ and $\mathcal{C}$ are essentially the only counterexamples for the EP-property of $\mathcal{M}_{K_5}$ and excluding both of them as minors always yields a class in which $\mathcal{M}_{K_5}$ exhibits the EP-property. Notice that this would imply that the $\subseteq$-minimal minor-closed classes where the EP-property fails for $\mathcal{M}_{K_5}$ are precisely two: the class of graphs embeddable in the projective plane and the class of graphs embeddable in the torus. Clearly, both these two classes have bounded Euler genus. Our next step is to observe that this is not true in general.

Kuratowski-connectivity. We say that a graph G is *Kuratowski-connected* if for every separation (A, B) of G of order at most 3, if there is a component C of $G[A \setminus B]$ and a component D of $G[B \setminus A]$, such that every vertex in $A \cap B$ has a neighbour in $V(C)$ and a neighbour in $V(D)$, then one of $G[A]$, $G[B]$ can be drawn in a disc Δ with $A \cap B$ drawn in the boundary of Δ. We denote by $\mathcal{K}$ the set of all Kuratowski-connected graphs. This definition was introduced by Robertson, Seymour, and Thomas as a tool for their characterization of *linklessly embeddable graphs* via a finite set of minimal obstructions [38] (see also [46, 30]).

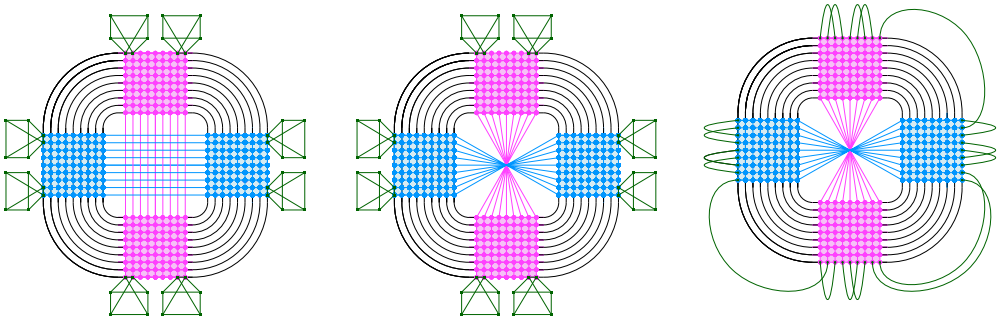


Figure 2 The two first parametric graphs serve as counterexamples for the Erdős-Pósa property of the graph J. The third parametric graph is a counterexample for the Erdős-Pósa property of K_8. All three parametric graphs have unbounded Euler-genus. For the first two this is witnessed by a large packing of $K_{3,3}$ while the last one can be observed to contain $K_{3,r}$ as a minor.

Consider the graph J obtained by identifying two adjacent vertices of $K_{3,3}$ with two vertices of K_5 and observe that J is not Kuratowski-connected. Similar to K_5, there cannot be two disjoint drawings of $K_{3,3}$ on the torus. So, if we take the parametric graph representing the torus ($\mathcal{H}_k$) or the projective plane ($\mathcal{C}_k$) from Figure 1 and paste "many" copies of $K_{3,3}$ around the "outer cycle", we obtain a parametric graph without two disjoint J-minors but where no small vertex-set can hit all J-minors (see the two first graphs in Figure 2).

Shallow-vortex minors. There is a second property, that poses a similar issue. In [41] Thilikos and Wiederrecht introduced the parametric graph $\mathcal{V}$ of *shallow-vortex grids* where $\mathcal{V}_k$ is obtained from the annulus grid $\mathcal{A}_k$ by adding k consecutive crossings in its internal cycle (see the fourth graph in Figure 1 for an illustration of $\mathcal{V}_9$). The class $\mathcal{V}$ of *shallow-vortex minors* was defined in [41] as the class containing all minors of $\mathcal{V}_k$, for all $k \in \mathbb{N}$. Notice that K_8 is a Kuratowski-connected graph. It was shown by Curticapean and Xia [6] that K_8 is not a shallow-vortex minor. However, this is the case for $K_{3,r}$, for every $r \in \mathbb{N}$, which implies that the parametric graph $\mathcal{V}_k$ has unbounded Euler-genus. If we now paste the k extra crossings of $\mathcal{V}_k$ to the "outer cycle" of $\mathcal{C}_k$, we obtain a parametric graph that is a counterexample for the EP-property of K_8 but which is of unbounded Euler-genus (see the last graph in Figure 2). These observations indicate that, if we want to understand the graphs for which the counterexamples of Robertson and Seymour precisely define the boundary to the $1/2$EP-property, we have to consider the graphs in $\mathcal{K} \cap \mathcal{V}$.

Our contribution. The main combinatorial result (stated in Theorem 2 in its full generality) is that Conjecture 1 holds, for every graph H that is Kuratowski-connected and a shallow vortex minor. Moreover, for every such non-planar H, $\mathsf{EP}_H(G)$ is equivalent to the exclusion of the parametric graphs representing some particular set of surfaces where H embeds. Therefore, for the non-planar graphs $H \in \mathcal{K} \cap \mathcal{V}$, the boundary between the Erdős-Pósa property and its half-integral relaxation is drawn *precisely* by a set of surfaces, depending on H. Notice, that the class $\mathcal{K} \cap \mathcal{V}$ encompasses, apart from planar graphs, several important graphs such as $K_5, K_{3,3}, K_{4,4}, K_6, K_7$, and the entire *Petersen family*. These last observations imply that our results extend, both algorithmically and combinatorially, to the half-integral packing of *links* and *knots*.

2 Notation and definitions

Let us introduce some notation in order to present our results in full generality. A *minor antichain* is a family $\mathcal{A}$ of graphs such that no graph $G_1 \in \mathcal{A}$ is a minor of another graph $G_2 \in \mathcal{A} \setminus \{G_1\}$. Since we focus on the minor relation, we refer to minor antichains simply as *antichains*. Let us denote by $\mathbb{K}$ the collection of all antichains $\mathcal{A}$ where *every* member of $\mathcal{A}$ is Kuratowski-connected. Moreover, let us denote by $\mathbb{V}$ the collection of all antichains containing *at least* one shallow-vortex minor. Finally, let $\mathbb{P}$ be the collection of all antichains containing at least one planar graph and set $\mathbb{H} := \mathbb{K} \cap \mathbb{V}$ and $\mathbb{H}^- := \mathbb{H} \setminus \mathbb{P}$.

The Erdős-Pósa property for antichains. Let H and G be graphs. A subgraph $H' \subseteq G$ is an *H-host* in G if H is a minor of H'. An *H-packing* in G is a collection of pairwise vertex-disjoint H-hosts in G. An *H-cover* is a set $S \subseteq V(G)$ such that $G - S$ is H-minor-free. A *half-integral H-packing* is a collection of H-hosts in G such that no vertex of G belongs to more than two of them.

Given an antichain $\mathcal{Z}$, we say that a subgraph $H' \subseteq G$ is a $\mathcal{Z}$-host in G if it is an H-host for some $H \in \mathcal{Z}$. A *$\mathcal{Z}$-packing* is an H-packing of *some* $H \in \mathcal{Z}$ and a *$\mathcal{Z}$-cover* is an H-cover for *all* $H \in \mathcal{Z}$, finally a *half-integral $\mathcal{Z}$-packing* is a half-integral H-packing for some $H \in \mathcal{Z}$. We define the two graph parameters $\mathsf{cover}_{\mathcal{Z}}$ and $\mathsf{pack}_{\mathcal{Z}}$ as follows.

$\mathsf{cover}_{\mathcal{Z}}(G) \coloneqq \min\{k \mid G \text{ has an } \mathcal{Z}\text{-cover of size } k\}$ and

$\mathsf{pack}_{\mathcal{Z}}(G) \coloneqq \max\{k \mid G \text{ has an } \mathcal{Z}\text{-packing of size } k.$

We say that $\mathcal{Z}$ has the *Erdős-Pósa property* in a graph class $\mathcal{G}$ if there exists some function $f\colon \mathbb{N} \to \mathbb{N}$ such that $\mathsf{cover}_{\mathcal{Z}}(G) \leq f(\mathsf{pack}_{\mathcal{Z}}(G))$, for all $G \in \mathcal{G}$.

Equivalence of graph parameters. We use $\mathcal{G}_{\mathsf{all}}$ for the class of all graphs. Given two graph parameters $\mathsf{p}, \mathsf{q}\colon \mathcal{G}_{\mathsf{all}} \to \mathbb{N}$, we say that p and q are *equivalent*, and write $\mathsf{p} \sim \mathsf{q}$, if there exists a function $f\colon \mathbb{N} \to \mathbb{N}$ such that, for every graph G, $\mathsf{p}(G) \leq f(\mathsf{q}(G))$ and $\mathsf{q}(G) \leq f(\mathsf{p}(G))$. We refer to the function f as the *gap* of this equivalence.

Our result is the identification of a graph parameter EP such that $\mathcal{Z}$ has the Erdős-Pósa property in a minor-closed graph class $\mathcal{G}$ with single-exponential gap if and only if EP is bounded in $\mathcal{G}$, for every $\mathcal{Z} \in \mathbb{H}$.

Surfaces and embeddability. We consider a containment relation $\preceq$ between surfaces where we write $\Sigma \preceq \Sigma'$ if the surface Σ' can be obtained by adding handles or cross-caps to the surface Σ. The *empty surface* will be denoted by $\Sigma^{\varnothing}$ and the surface obtained by adding h handles and c cross-caps to the sphere $\Sigma^{(0,0)}$ is denoted by $\Sigma^{(h,c)}$. Its *Euler-genus* is defined to be $2h + c$. Notice that, by Dyck's Theorem [8], we may assume that $c \leq 2$ for all surfaces. Let $\mathbb{S}$ be a set of surfaces. We say that $\mathbb{S}$ is *closed*, if $\Sigma \in \mathbb{S}$ and $\Sigma' \preceq \Sigma$ imply that $\Sigma' \in \mathbb{S}$ and that it is *proper*, if it does not contain all surfaces. If $\mathbb{S}$ is closed and proper we define the "surface obstruction set" $\mathsf{sobs}(\mathbb{S})$ as the set of all $\preceq$-minimal surfaces which do not belong to $\mathbb{S}$. It is easy to observe that $\mathsf{sobs}(\mathbb{S})$ always consists of one or two surfaces [43]. Notice that $\mathsf{sobs}(\emptyset) = \{\Sigma^{\varnothing}\}$, $\mathsf{sobs}(\{\Sigma^{\varnothing}\}) = \{\Sigma^{(0,0)}\}$, $\mathsf{sobs}(\{\Sigma^{\varnothing}, \Sigma^{(0,0)}\}) = \{\Sigma^{(1,0)}, \Sigma^{(0,1)}\}$, and, for a more complicated example, $\mathsf{sobs}(\{\Sigma^{\varnothing}, \Sigma^{(0,0)}, \Sigma^{(0,1)}, \Sigma^{(0,2)}\}) = \{\Sigma^{(1,0)}\}$.

We say that a graph G is *embeddable* in a surface Σ (or Σ-embeddable) if it has a drawing in Σ without crossings. The *Euler genus* of a graph G, denoted by $\mathsf{eg}(G)$, is the smallest Euler genus of a surface where G is embeddable.

Parametric graphs and Dyck-grids. A *parametric graph* is a sequence $\mathcal{G} = \langle \mathcal{G}_i \rangle_{i \in \mathbb{N}}$ of graphs indexed by non-negative integers. We say that $\mathcal{G}$ is *minor-monotone* if for every $i \in \mathbb{N}$ we have that $\mathcal{G}_i$ is a minor of $\mathcal{G}_{i+1}$. All parametric graphs considered in this paper are minor-monotone. We write $\mathcal{G}^{(1)} \lesssim \mathcal{G}^{(2)}$ for two minor-monotone parametric graphs $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$ if there exists a function $f\colon \mathbb{N} \to \mathbb{N}$ such that for every $i \in \mathbb{N}$ it holds that $\mathcal{G}^{(1)}_i$ is a minor of $\mathcal{G}^{(2)}_{f(i)}$. A *minor-monotone parametric family* is a finite collection of $\mathfrak{G} = \{\mathcal{G}^{(j)} \mid j \in [r]\}$ of minor-monotone parametric graphs such that for distinct $i, j \in [r]$ it holds that $\mathcal{G}^{(i)} \not\lesssim \mathcal{G}^{(j)}$ and $\mathcal{G}^{(j)} \not\lesssim \mathcal{G}^{(i)}$. We define the minor-monotone parameter

$$\mathsf{p}_{\mathfrak{G}}(G) \coloneqq \max\{k \mid \text{there exists } i \in [r] \text{ such that } G \text{ contains } \mathcal{G}^{(i)}_k \text{ as a minor}\}. \tag{3}$$

The three parametric graphs $\mathcal{A} = \langle \mathcal{A}_k \rangle_{k\in\mathbb{N}}$, $\mathcal{H} = \langle \mathcal{H}_k \rangle_{k\in\mathbb{N}}$, and $\mathcal{C} = \langle \mathcal{C}_k \rangle_{k\in\mathbb{N}}$ are defined as follows: The *annulus grid* $\mathcal{A}_k$ is the $(4k, k)$-cylindrical grid[6] depicted in the far left of Figure 1. The *handle grid* $\mathcal{H}_k$ (resp. *cross-cap grid* $\mathcal{C}_k$) is obtained by adding in $\mathcal{A}_k$ edges as indicated in the middle left (resp. middle right) part of Figure 1. We refer to the added edges as *transactions* of the handle grid $\mathcal{H}_k$ or of the cross-cap grid $\mathcal{C}_k$.

Let now $h \in \mathbb{N}$ and $c \in [0, 2]$. We define the parametric graph $\mathcal{D}^{(h,c)} = \langle \mathcal{D}_k^{(h,c)} \rangle_{k\in\mathbb{N}}$ by taking one copy of $\mathcal{A}_k$, h copies of $\mathcal{H}_k$, and $c \in [0, 2]$ copies of $\mathcal{C}_k$, then "cut" them along the dotted red line, as in Figure 1, and join them together in the cyclic order $\mathcal{A}_k, \mathcal{H}_k, \ldots, \mathcal{H}_k, \mathcal{C}_k, \ldots, \mathcal{C}_k$, as indicated in Figure 3.

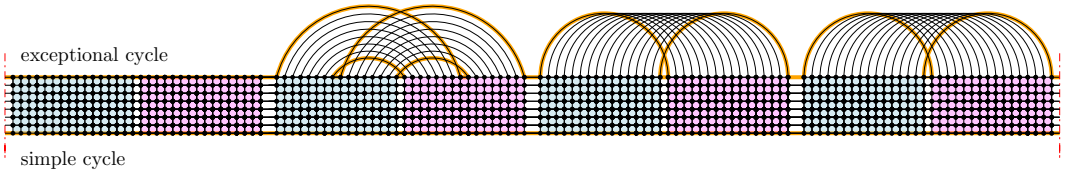


Figure 3 The Dyck-grid $\mathcal{D}_8^{1,2}$. The simple and the exceptional cycles are drawn in orange.

We call the graph $\mathcal{D}_k^{(h,c)}$ the *Dyck-grid* of *order* k *with* h *handles and* c *cross-caps*. Given some surface $\Sigma = \Sigma^{(h,c)}$, we say that the graph D is the $(\Sigma; d)$-*Dyck-grid* if $D = \mathcal{D}_d^{(h,c)}$ and we use $\mathcal{D}^\Sigma$ to denote the parametric graph $\langle \mathcal{D}_i^\Sigma \rangle_{i\in\mathbb{N}}$, where $\mathcal{D}_i^\Sigma$ is the $(\Sigma; i)$-Dyck-grid.

Let us return to our antichain $\mathcal{Z} \in \mathbb{H}^-$. We denote by $\mathbb{S}_\mathcal{Z}$ the set of surfaces where none of the graphs in $\mathcal{Z}$ can be embedded. Notice that $\mathbb{S}_\mathcal{Z}$ is closed and proper and, for every $\Sigma \in \mathsf{sobs}(\mathbb{S}_\mathcal{Z})$, there exists some $H \in \mathcal{Z}$ such that H embeds in Σ.

3 Our results

We associate with $\mathcal{Z}$ the parametric family $\mathfrak{D}_\mathcal{Z} := \{\mathcal{D}^\Sigma \mid \Sigma \in \mathsf{sobs}(\mathbb{S}_\mathcal{Z})\}$. Let $\mathsf{EP}_\mathcal{Z} := \mathsf{p}_{\mathfrak{D}_\mathcal{Z}}$. Our combinatorial result determines precisely when a member in $\mathbb{H}^-$ has the Erdős-Pósa in some minor-closed graph class.

▶ **Theorem 2.** *For every $\mathcal{Z} \in \mathbb{H}^-$, for every minor-closed graph class $\mathcal{G}$, $\mathcal{Z}$ has the Erdős-Pósa property in $\mathcal{G}$ if and only if $\mathsf{EP}_\mathcal{Z}$ is bounded in $\mathcal{G}$.*

Let $h_\mathcal{Z} := \max\{|V(H)| \mid H \in \mathcal{Z}\}$ and $\gamma_\mathcal{Z} := \max\{\mathsf{eg}(H) \mid H \in \mathcal{Z}\}$. The engine that drives the proof of Theorem 2 and which represents our first main algorithmic result is the following.

▶ **Theorem 3.** *There exists a function $f_3 : \mathbb{N}^4 \to \mathbb{N}$ such that, for every antichain $\mathcal{Z} \in \mathbb{H}^-$, there exists an algorithm such that, given $k, t \in \mathbb{N}$ and a graph G, outputs one of the following:*

- *a $\mathcal{D}_t^\Sigma$-host in G, for some $\Sigma \in \mathsf{sobs}(\mathbb{S}_\mathcal{Z})$, or*
- *an $\mathcal{Z}$-packing of size at least k in G, or*
- *an $\mathcal{Z}$-cover of size at most $f_3(\gamma_\mathcal{Z}, h_\mathcal{Z}, t, k)$ in G.*

Moreover, the algorithm runs in time $2^{2^{\mathcal{O}_{\gamma_\mathcal{Z}}(\mathsf{poly}(t))+\mathcal{O}_{h_\mathcal{Z}}(\mathsf{poly}(k))}} \cdot |V(G)|^3 \cdot \big(\log(|V(G)|)\big)^2$ and $f_3(\gamma_\mathcal{Z}, h_\mathcal{Z}, t, k) = 2^{\mathcal{O}_{\gamma_\mathcal{Z}}(\mathsf{poly}(t))+\mathcal{O}_{h_\mathcal{Z}}(\mathsf{poly}(k))}$.

By a recent result of Gavoille and Hilaire [13], it holds that there exists some constant c such that for every $\mathcal{Z} \in \mathbb{H}^-$ and $\Sigma \in \mathsf{sobs}(\mathbb{S}_\mathcal{Z})$, there exists some $H \in \mathcal{Z}$ such that H is a minor of $\mathcal{D}^\Sigma_{c\gamma_\mathcal{Z}^4 h_\mathcal{Z}^2}$. Moreover, as observed in [43], every Dyck-grid of big enough order

[6] An $(n \times m)$-cylindrical grid is a Cartesian product of a cycle on n vertices and a path on m vertices.

contains a large half-integral packing of itself of smaller order. Combining these two results with Theorem 3, yields the following (constructive) parameterized approximation algorithm for $1/2\text{-}\mathsf{pack}_{\mathcal{Z}}$.

▶ **Theorem 4.** *There exists a function $f_4 : \mathbb{N}^2 \to \mathbb{N}$ such that, for every antichain $\mathcal{Z} \in \mathbb{H}^-$, there exists an algorithm such that, given $k \in \mathbb{N}$ and a graph G, outputs one of the following:*
1. *a half-integral $\mathcal{Z}$-packing of size at least k in G, or*
2. *an $\mathcal{Z}$-cover of size at most $f_4(h_{\mathcal{Z}}, k)$ in G.*

Moreover, the algorithm runs in time[7] $2^{2^{\mathsf{poly}_{h_{\mathcal{Z}}}(k)}} \cdot |V(G)|^3 \cdot \big(\log(|V(G)|)\big)^2$ *and* $f_4(h_{\mathcal{Z}}, k) = 2^{\mathsf{poly}_{h_{\mathcal{Z}}}(k)}$.

We wish to stress that, given the combinatorial bounds of Theorem 3, we may directly apply the minor-checking algorithm of [23] for the two first outcomes of Theorem 3 and the algorithm of [29] for its third outcome. Both these algorithms are quadratic on $|V(G)|$ and this implies alternative quadratic algorithms to those in Theorem 3 and Theorem 4. However, this would come with the cost of enormous parametric dependencies on k.

3.1 Some implications of our results

Half-integral Erdős-Pósa for linked pairs and knots. As mentioned above, $\mathbb{H} = \mathbb{K} \cap \mathbb{V}$ contains several antichains of particular interest. A first example is the *Petersen family*, which is exactly the (minor) obstruction set[8] for the so-called *linklessly embeddable* graphs (in short, *link-less* graphs). Indeed, the origin of the definition of Kuratowski-connectivity comes from the paper of Robertson, Seymour, and Thomas [38], where this obstruction set was found. All obstructions for link-less graphs as well as those for knot-less graphs are Kuratowski-connected. Moreover, as the shallow-vortex minor K_6 (resp. K_7) is a member of the obstruction set of link-less (resp. knot-less) graphs, we also have that both these obstruction sets belong to $\mathbb{H}^-$. This insight allows us to apply Theorem 4 to topological objects such as *links* and *knots*.

Let G be a graph and let $\mathcal{C} = \{C_1, \ldots, C_k\}$ be a collection of subgraphs of G. The *intersection graph* of $\mathcal{C}$ is the graph $I(\mathcal{C}) = (\mathcal{C}, E_{\mathcal{C}})$ where $CC' \in E_{\mathcal{C}}$ if and only if $C \cap C'$ is not the empty graph. We say that $\mathcal{C}$ is a *collection of double cycles* (resp. *cycles*) if each C_i is union of two disjoint cycles (resp. a cycle).

Given a collection $\mathcal{C}$ of double cycles (resp. cycles) of G, and some $\mathbb{R}^3$-embedding of G, we say that $\mathcal{C}$ is a $1/2$-*packing* of links (resp. knots) if for every $i \in [k]$, the two components of C_i are linked (resp. the cycle C_i is knotted) in this particular embedding (see [1] for more on links and knots). The *half-integral linked pair (resp. knot) packing number of a graph G*, denoted by $1/2\text{-}\mathsf{lppack}(G)$ (resp. $1/2\text{-}\mathsf{knpack}(G)$), is the maximum k such that, for every $\mathbb{R}^3$-embedding of G, there exists a $1/2$-packing of links (resp. knots) in G of size k. Both $1/2\text{-}\mathsf{lppack}(G)$ and $1/2\text{-}\mathsf{knpack}(G)$ are minor-monotone parameters, therefore we know (non-constructively) that there exists an algorithm for checking whether $1/2\text{-}\mathsf{lppack}(G) \geq k$ ($1/2\text{-}\mathsf{knpack}(G) \geq k$) in time $f(k) \cdot |V(G)|^2$. Up to now, no constructive (on k) algorithm is known for these problems. Our results imply the following.

[7] Given two functions $\chi, \psi \colon \mathbb{N} \to \mathbb{N}$, we write $\chi(n) = \mathcal{O}_x(\psi(n))$ in order to denote that there exists a computable function $f \colon \mathbb{N} \to \mathbb{N}$ such that $\chi(n) = \mathcal{O}(f(x) \cdot \psi(n))$. We also use $\chi(n) = \mathsf{poly}_x(\psi(n))$ instead of $\chi(n) = \mathcal{O}_x((\psi(n))^c)$, for some $c \in \mathbb{N}$.

[8] The obstruction set of some minor-closed class $\mathcal{G}$ is the set $\mathsf{obs}(\mathcal{G})$ of the minor-minimal graphs that are not in $\mathcal{G}$.

▶ **Theorem 5.** *There exists a function $f : \mathbb{N} \to \mathbb{N}$ and algorithms that, given a graph G and a $k \in \mathbb{N}$, outputs either that $1/2$-$\mathsf{lppack}(G) \geq k$ (resp. $1/2$-$\mathsf{knpack}(G) \geq k$) or a vertex set A of at most $f(k)$ vertices such that $G - A$ has a link-less (knot-less) $\mathbb{R}^3$-embedding. Moreover, both algorithms run in time $2^{2^{\mathsf{poly}(k)}} \cdot |V(G)|^3 \cdot \big(\log(|V(G)|)\big)^2$ and $f(k) = 2^{\mathsf{poly}(k)}$.*

Theorem 5 implies that both the parameter $1/2$-lppack as well as the parameter $1/2$-knpack admit FPT-approximation algorithms with exponential approximation gap. Moreover, in case the output is that $1/2$-$\mathsf{lppack}(G) \geq k$ (resp. $1/2$-$\mathsf{knpack}(G) \geq k$), the algorithms output a $1/2$-packing of k graphs certifying that, every $\mathbb{R}^3$-embedding of G contains a $1/2$-packing $\mathcal{C}$ of k links (resp. knots) such that $I(\mathcal{C})$ is either edgeless or a clique. We stress that the above algorithms become constructive (on k) as we know the obstructions of link-less/knot-less graphs or at least an upper bound to their size. For the later class not such bound is known.

Other implications of our results, related to canonical approximate characterizations of the parameters we study, are discussed in the conclusion section (Section 4).

3.2 Outline of the proof

We begin the description of the main ideas of our proof with the definition of a tree decomposition.

Tree decompositions. Let G be a graph. A *tree decomposition* of a graph G is a pair (T, β) where T is a tree and $\beta : V(T) \to 2^{V(G)}$ is a function, whose images are called the *bags* of $\mathcal{T}$, such that $\bigcup_{t \in V(T)} \beta(t) = V(G)$, for every $e = xy \in E(G)$, there exists $t \in V(T)$ with $\{x, y\} \subseteq \beta(t)$, and for every $v \in V(G)$, the set $\{t \in V(T) \mid v \in \beta(t)\}$ induces a subtree of T. We refer to the vertices of T as the *nodes* of the tree decomposition $\mathcal{T}$. The *width* of $\mathcal{T}$ is the value $\max_{t \in V(T)} |\beta(t)| - 1$. The *treewidth* of G, denoted by $\mathsf{tw}(G)$, is the minimum width over all tree decompositions of G.

The classic approach. In order to facilitate the presentation of our proof, let us briefly explain the two main ideas of the proof that planar graphs enjoy the Erdős-Pósa property in the set of all graphs. The key ingredient is that every planar graph is a minor of a graph of sufficiently large treewidth. The proof follows in two steps.

Step 1. Assuming that $\mathsf{pack}_H(G) \leq k$, based on the grid theorem by Robertson and Seymour, we may assume that the treewidth of G is bounded by some function of k.

Step 2. With the tree decomposition (T, β) of G at hand, we build an H-cover A of G by adding to it (if any exists) an adhesion $D_{xy} = \beta(x) \cap \beta(y)$ such that both $G_x := G[\beta(V(T_x)) \setminus D_{xy}]$ and $G_y := G[\beta(V(T_y)) \setminus D_{xy}]$ contain H as a minor (here T_x and T_y are the two components of $T - xy$) and then recursing on the corresponding tree decompositions of G_x and G_y. If $\mathsf{pack}_H(G) \leq k$, eventually this procedure returns an H-cover of size at most $k \cdot (\mathsf{tw}(G) + 1)$.

Throughout the present outline we describe arguments that can be paralleled to the two steps above. Moreover, in each step we explain the challenges that are met and the way we deal with them in our proof.

For simplicity, instead of an antichain $\mathcal{Z}$, we consider a non-planar graph H that is Kuratowski-connected and a shallow-vortex minor. We denote by $\mathbb{S}_H$ the set of all surfaces where H cannot be embedded and by $\mathbb{S}'_H := \mathsf{sobs}(\mathbb{S}_H)$ the corresponding surface obstruction set. We stress that the graphs in $\mathfrak{D}_H = \{\mathcal{D}^\Sigma \mid \Sigma \in \mathbb{S}'_H\}$ can be seen as "generators of

half-integrality". Indeed, it is possible to prove that, for every $t \in \mathbb{N}$, $\mathsf{pack}_H(\mathcal{D}_t^\Sigma) \leq 1$, and $\mathsf{cover}_H(\mathcal{D}_t^\Sigma) = \Theta(1/2\text{-}\mathsf{pack}_H(\mathcal{D}_t^\Sigma)) = \Omega(t)$. This already proves the easy direction of Theorem 2.

Let $\mathcal{T} = (T, \beta)$ be a tree decomposition of a graph G. For each $t \in V(T)$, we define the *adhesions* of t as the sets in $\{\beta(t) \cap \beta(d) \mid d \text{ adjacent with } t\}$ and the maximum size of them is called the *adhesion* of t. The *adhesion* of $\mathcal{T}$ is the maximum adhesion of a node of $\mathcal{T}$. The *torso* of $\mathcal{T}$ on a node t is the graph, denoted by G_t, obtained by adding edges between every pair of vertices of $\beta(t)$ which belongs to a common adhesion of t.

We now consider a graph G where $\mathsf{pack}_H(G) \leq k$ and we assume that G excludes as a minor the Dyck grid $\mathcal{D}_t^\Sigma$, for every $\Sigma \in \mathbb{S}'_H$. Under these circumstances, our aim is to find an H-cover whose size is bounded by some function of t and k.

Graphs excluding Dyck grids. As a first step, we need a deeper understanding of how the graphs excluding $\mathcal{D}_t^\Sigma$ look like. In general, the structure of graphs excluding a given graph as a minor is given by the Graph Minors Structure theorem (in short GMST). However, the formal definition of GMST involves complicated concepts which we prefer not to introduce in this brief outline. Instead we give a more compact statement, proved in [43].

Given a graph H and a set $A \subseteq V(G)$, we say that H is an *A-minor of G* if there is a collection $\mathcal{S} = \{S_v \mid v \in V(H))\}$ of pairwise vertex-disjoint connected[9] subsets of $V(G)$, each containing at least one vertex of A and such that, for every edge $xy \in E(H)$, the set $S_x \cup S_y$ is connected in G. Given an annotated graph (G, A) where G is a graph and $A \subseteq V(G)$, we define $\mathsf{tw}(G, A)$ as the maximum treewidth of an A-minor of G. A streamlined way to restate the GMST is the following.

▶ **Proposition 6** ([43]). *There exists a function $f : \mathbb{N} \to \mathbb{N}$ such that every graph G excluding a graph on k vertices as a minor, has a tree decomposition (T, β) where, for every $t \in V(T)$, the torso G_t contains some set A_t where $\mathsf{tw}(G_t, A_t) \leq f(k)$ and such that $G_t - A_t$ can be embedded in a surface of Euler genus at most $f(k)$.*

To deal with the exclusion of Dyck grids (corresponding to surfaces), we need a more refined version of Proposition 6 that works for every (closed and proper) set of surfaces $\mathbb{S}$. In this direction, Thilikos and Wiederrecht defined in [43] an extension of treewidth, namely $\mathbb{S}$-tw, where for a graph G,

$$\begin{aligned}&\mathbb{S}\text{-}\mathsf{tw}(G) \text{ is the minimum } k \text{ for which } G \text{ has a tree decomposition } (T, \beta) \text{ where, for every}\\ &t \in V(T), \text{ the torso } G_t \text{ contains some set } A_t \text{ where } \mathsf{tw}(G_t, A_t) \leq k \text{ and } G_t - A_t \text{ is}\\ &\text{embeddable in a surface in } \mathbb{S}.\end{aligned} \tag{4}$$

The main result of [43] is that in order to exclude the graphs in $\mathfrak{D}_H = \{\mathcal{D}_t^\Sigma \mid \Sigma \in \mathsf{cobs}(\mathbb{S})\}$, we have to fix the surface of Proposition 6 to be one of the surfaces in $\mathbb{S}$.

▶ **Proposition 7.** *For every closed and proper set of surfaces $\mathbb{S}$, there exists some function $f : \mathbb{N} \to \mathbb{N}$ such that, for every graph G, if G excludes all graphs in $\{\mathcal{D}_t^\Sigma \mid \Sigma \in \mathsf{sobs}(\mathbb{S})\}$ as minors, then $\mathbb{S}$-$\mathsf{tw}(G) \leq f(t)$.*

Notice that the above proposition already gives us the grid theorem when applied for the set $\mathbb{S}_\emptyset$ containing the empty surface $\Sigma^\varnothing$. It is easy to verify that $\mathsf{tw} + 1 = \mathbb{S}_\emptyset$-$\mathsf{tw}$. As $\mathsf{sobs}(\mathbb{S}_\emptyset) = \{\Sigma^{(0,0)}\}$, Proposition 7 implies that graphs excluding $\mathcal{D}_t^{\Sigma^{(0,0)}} = \mathcal{A}_t$ have bounded treewidth (see Figure 1 for an example of an annulus grid).

[9] A set $X \subseteq V(G)$ is *connected* in G if the induced subgraph $G[X]$ is a connected graph.

From small treewidth modulators to small size modulators. Proposition 7 gives valuable information on the structure of the graphs that exclude the "half-integrality generators" in $\mathfrak{D}_H = \{\mathcal{D}_t^\Sigma \mid \Sigma \in \mathsf{sobs}(\mathbb{S}_H)\}$. Therefore, we can assume that $\mathbb{S}\text{-}\mathsf{tw}(G) \leq f(k)$, which provides a tree decomposition as the one in (4). In order to make progress, we need to further refine this decomposition as small treewidth modulators are not particularly helpful in finding an H-cover of small size. For this we exploit the assumption that H is a shallow-vortex minor.

To elaborate, we need some additional information, analogous to the exclusion of a planar graph in **Step 1**. This corresponds to the assumption that H is a minor of the shallow-vortex grid $\mathcal{V}_{h'}$ for some h' depending on H. One can observe that $\mathcal{V}_{3(k+1)h'}$ contains a $\mathcal{V}_{h'}$-packing of size $(k+1)$. Therefore, the assumption that $H\text{-}\mathsf{pack}(G) \leq k$ gives us the right to additionally assume that G also excludes the shallow-vortex minor $\mathcal{V}_{3(k+1)h'}$. Using this and the fact that $\mathbb{S}\text{-}\mathsf{tw}(G) \leq f(k)$, we are able to further restrict the decomposition of (4). To quantify this, we introduce a new graph parameter $\mathbb{S}\text{-}\mathsf{tw}_{\mathsf{apex}}$ defined as follows.

$$\begin{aligned}&\mathbb{S}\text{-}\mathsf{tw}_{\mathsf{apex}}(G) \text{ is the minimum } k \text{ for which } G \text{ has a tree decomposition } (T,\beta) \text{ where, for}\\ &\text{every } t \in V(T), \text{ the torso } G_t \text{ contains some set } A_t \text{ where } |A_t| \leq k \text{ and } G_t - A_t \text{ is}\\ &\text{embeddable in a surface in } \mathbb{S}.\end{aligned} \tag{5}$$

Notice that the only difference between (4) and (5) is the measure defined on the "modulator" A_t. While in (4) it is the treewidth of the annotated graph (G_t, A_t), in (5) it is the *size* of A_t. The first ingredient of our proof is that, under the absence of some shallow-vortex minor, the two parameters $\mathbb{S}\text{-}\mathsf{tw}_{\mathsf{apex}}$ and $\mathbb{S}\text{-}\mathsf{tw}$ are equivalent. This is proved by combining the results of [43] with the results of [41] on the structure of the graphs excluding a shallow-vortex grid.

As a consequence, we may now assume that we have a tree decomposition (T,β) as the one in (5). This decomposition is not yet in position to play the role of the tree decomposition in **Step 2**, as its torsos may have unbounded size. To circumvent this issue, we instead prove a local structure theorem for the exclusion of $\mathfrak{D}_H \cup \{\mathcal{V}\}$, that can be extended to a global one (the desired tree decomposition), using the results of [42]. The general approach is to consider some big enough wall W_t and locally focus on a torso G_t that contains most of the essential part of W_t.

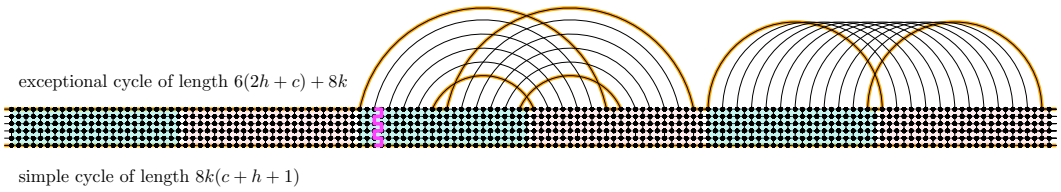


Figure 4 The elementary $(h, c; k)$-Dyck wall, where $h = 1$, $c = 1$, and $k = 6$.

Torsos with Dyck walls. According to Proposition 7, (5), and the equivalence of $\mathbb{S}\text{-}\mathsf{tw}_{\mathsf{apex}}$ and $\mathbb{S}\text{-}\mathsf{tw}$, G_t comes together with a set A_t such that the graph $G_t' := G_t - A_t$ is accompanied by some Σ-embedding for a surface $\Sigma \in \mathbb{S}_H$, where H cannot be embedded. However, we require some additional infrastructure in G_t that will come in the form of a large wall-like object that is controlled by our Σ-embedding.

Notice that every adhesion $\beta(t) \cap \beta(t')$ of t defines a separation $(X_{t'}, Y_{t'})$ of $G - A_t$ of order at most 3 where $G[X_{t'} \cap Y_{t'}]$ is drawn in Σ as a clique. We fix the orientation $(X_{t'}, Y_{t'})$ such that $V(G_t) \subseteq Y_{t'}$, thereby indicating that $Y_{t'}$ is the "important" part of the separation.

Due to the results in [43], G_t contains a $(\Sigma; d)$-*Dyck wall*[10] D_t, which is highly linked to the wall W_t above. Here d is chosen "large enough" so as to ensure the applicability of the next steps of our proof. Also, we may assume that the "essential" part of D_t is drawn "inside" G_t in the sense that, for each $(X_{t'}, Y_{t'})$, at most one branch vertex of D_t is in $Y_{t'} \setminus X_{t'}$. The wall W_t is chosen large enough to represent some *tangle*, that is an orientation of the separations of G of some suitably bounded order. The way to algorithmically detect such a big wall W_t is given in [43].

The role of Kuratowki-connectivity. We next make some observations on how "models" of H can behave with respect to the Σ-embedding of G_t. These observations will play a key role in understanding how to "attack" and later "kill" copies of H in our graph.

The first comes from the non-Σ-embeddability property of H: "minimal" H-hosts in G, called *H-inflated copies*, *cannot be entirely inside* G_t, otherwise we would be able to embed H in a surface where it cannot be embedded. Another important feature comes from the fact that H is Kuratowski-connected: every H-inflated copy M in G is "well oriented" with respect to the adhesions of t in the sense that, when M traverses some adhesion $X_{t'} \cap Y_{t'} = \beta(t) \cap \beta(t')$ of G, *exactly one* of the two parts of M induced by $X_{t'}$ and $Y_{t'}$ should *not* be embeddable in the disk bounding $\beta(t) \cap \beta(t')$ with the vertices of $\beta(t) \cap \beta(t')$ on its boundary. This implies that the "non-disk-embeddable" part will always lie inside the set $X_{t'}$ of the separation $(X_{t'}, Y_{t'})$ above. Given now some adhesion $\beta(t) \cap \beta(t')$, we say that it is *H-red* if it is intersected by the (unique, due to Kuratowski-connectivity) non-disk-embeddable part of some H-inflated copy M in G. That way, it is convenient to visualize H-red adhesions as the "entrances" from which the H-inflated copies of G "invade" G_t.

Updating the Σ-embedding. From our previous observations it follows that to eliminate all copies of H locally in G_t it suffices to deal with all H-inflated copies that invade G_t through H-red adhesions. Therefore, our next objective is to update A_t, $G'_t = G_t - A_t$, and the Σ-embedding of G'_t in a way that the remaining part of G'_t will not contain any H-red adhesions, i.e., in a way that no invading H-inflated copy survives.

During our proof, this updating procedure will focus on some closed disk Δ containing some collection of H-red adhesions (these disks will be gathered together in what we call *H-red railed flat vortices*) and detect some separation (X, Y) of G where $X \setminus Y$ contains the vertices of the Dyck wall D_t and Y contains all H-red adhesions in Δ. We call such a separation a *carving separation*. Each time we find such a separation, we move $X \cap Y$ to A and also move $Y \setminus X$ "outside" G_t. As the set $X \cap Y$ adds up to the size of A we also need that $X \cap Y$ has "small" order. We refer to this operation as *taking a carving* of our Σ-embedding at the carving separation (X, Y). When the whole procedure terminates, none of the adhesions of the updated G'_t is H-red. This implies that $(V(G_t), A_t)$ is what we call an *H-dominion* of G, that is: *if the non-disk-embeddable part of some H-inflated copy in G intersects $V(G_t)$ then it also intersects A_t.*

To achieve the previously described objective we adopt the following strategy. Recall that in the Σ-embedding of G_t, H-red adhesions are cliques of size at most three that may be drawn all around Σ. Our first step is to show that H-red adhesions can be cornered in the "interior" of less than k pairwise-disjoint territories of Σ, each maintaining a large enough "buffer" around a disk where the H-red adhesions reside. Afterwards we refine these territories in order to bound their complexity in the sense that there is no large flow in G_t

[10] Here a $(\Sigma; d)$-Dyck wall is certifying the existence of the Dyck grid $\mathcal{D}_d^{\Sigma}$ as a minor. See Figure 4.

that crosses through these territories. Through this refinement step we obtain some some additional structural information so that in the last part of the proof, these territories along with their infrastructure will allow us to finally eliminate all H-inflated copies by removing a bounded number of vertices from their interiors.

Redrawing H-inflated copies inside a railed flat vortex. To formalize the aforementioned territories that will encapsulate the H-red adhesion of our embedding, we utilize the concept of a *railed nest* $(\mathcal{C}, \mathcal{P})$ of G *around* some closed disk Δ^{int} of Σ. Here $\mathcal{C} = \langle C_1, \dots, C_\ell \rangle$ is a sequence of ℓ disjoint cycles of G_t, where each C_i bounds some closed disk Δ_i in Σ, where $\Delta^{\text{int}} \subseteq \Delta_1 \subsetneq \cdots \subsetneq \Delta_\ell$, along with a set of paths $\mathcal{P} = \langle P_1, \dots, P_\ell \rangle$, drawn in $\Delta^{\text{ext}} := \Delta_\ell$, not traversing the interior of Δ^{int}, joining vertices of C_1 with vertices of C_ℓ, and traversing the cycles in $\mathcal{C}$ *orthogonally*, that is $P_i \cap C_j$ is connected for every $(i, j) \in [\ell]^2$. We refer to such a railed nest, as a *railed flat vortex* and we refer to the disk Δ^{int} (resp. Δ^{ext}) as its *internal* (resp. *external disk*). Moreover, if all H-red adhesions drawn in Δ^{ext} are also drawn inside Δ^{int}, then we call it an *H-red railed flat vortex*. An important ingredient of our proof is to show that we may use the infrastructure of the cycles and the paths in $(\mathcal{C}, \mathcal{P})$ in order to *redraw* inside Δ^{ext} every H-inflated copy M that invades G_t via an H-red adhesion of Δ^{ext}. Even if the part of M that is embedded inside Δ^{ext} is not necessarily a disk embedding, we can make this redrawing possible by using disk embedability properties emerging from the Kuratowski-connectivity of H and the "linkage combing" lemma from [17, 16, 15]. We refer to this as *the redrawing lemma*.

Gathering H-adhesions in railed flat vortices. The next step of our strategy, is to corner all H-red adhesions in the interior of less than k H-red railed flat vortices. Towards this, we take advantage of the infrastructure provided by the $(\Sigma; d)$-*Dyck wall* D_t. A brick of D_t is called H-red if it "contains" an H-red adhesion. More precisely, this is formalized by the notion of the *influence* of a brick which roughly corresponds to a set of H-red adhesions that are intersected or contained by a closed disk in Σ that bounds the "area" that is enclosed by the corresponding brick. This assigns each H-red adhesion to the influence of at most three neighbouring H-red bricks and defines a notion of distance between H-red adhesions expressed by the distance of the corresponding H-red bricks in D_t. Next, we prove that under this distance notion, no scattered enough set of H-red bricks of size k can exist. For this, we use the fact that each H-red brick B implies the existence of an H-inflated copy in G that, due to the aforementioned "redrawing lemma", can be redrawn in a small radius around B. This radius is bounded but also big enough so as to permit the redrawing. Likewise, we prove that there are few H-red bricks away from the exceptional and the simple cycle of D_t (see Figure 4 for a visualization of these two cycles). Next we use a greedy procedure in order to group together this bounded number of bricks and maintain enough railed nest infrastructure around them to cluster them into less than k railed flat vortices. The construction is completed by creating two more railed flat vortices, one for the simple cycle of D_t and one for the exceptional one.

Refining H-red railed flat vortices. We are now in the position where we have defined a set of less than k many H-red railed flat vortices whose internal disks contain all H-red adhesions and whose external disks are pairwise disjoint. The next step is to further refine these flat vortices.

In our proof, we treat what is drawn in the external disk Δ^{ext} as a vortex in the classic sense and our goal is to bound their *depth*, that is to ensure that no large *transaction* goes through the society defined by each railed flat vortex. Each of them consists of a

subgraph $G_{\Delta^{\text{ext}}}$ of G (the one that is drawn in Δ^{ext}) where the vertices in the boundary of the external disk Δ^{ext} are arranged in some cyclic ordering $\Omega_{\Delta^{\text{ext}}}$. A *segment* of $\Omega_{\Delta^{\text{ext}}}$ is a set $S \subseteq V(\Omega_{\Delta^{\text{ext}}})$ such that there do not exist $s_1, s_2 \in S$ and $t_1, t_2 \in V(\Omega_{\Delta^{\text{ext}}}) \setminus S$ such that s_1, t_1, s_2, t_2 occur in Ω_Δ in the order listed. A *transaction* in $(G_{\Delta^{\text{ext}}}, \Omega_{\Delta^{\text{ext}}})$ is a set of pairwise disjoint paths, drawn in Δ, between two disjoint segments A, B of $\Omega_{\Delta^{\text{ext}}}$. The *depth* of $(G_{\Delta^{\text{ext}}}, \Omega_{\Delta^{\text{ext}}})$ is the maximum size of a transaction in $(G_{\Delta^{\text{ext}}}, \Omega_{\Delta^{\text{ext}}})$. Our next objective is to refine each of our H-red railed flat vortices so that, in the end, some disk $\Delta' \subseteq \Delta^{\text{ext}}$ defines a vortex $(G_{\Delta'}, \Omega_{\Delta'})$ of *bounded* depth and, moreover, the vertices in the boundary of Δ' are all connected with disjoint paths to the boundary of the external disk Δ^{ext}. We do this as follows: If there is no transaction in $(G_{\Delta^{\text{ext}}}, \Omega_{\Delta^{\text{ext}}})$ where a big part of its paths also traverse Δ^{int}, we make use of the "nest tightening"-lemma from [41] in order to either update the nest to a "tighter" one (which allows us to recurse), or find the disk Δ' claimed above, or find a small-order carving separation (X, Y) (again defined by some closed disk) at which we may take a carving of our Σ-embedding. If there is a transaction in $(G_{\Delta^{\text{ext}}}, \Omega_{\Delta^{\text{ext}}})$ where a big enough part of its paths also traverse Δ^{int}, then we use this transaction in order to split the vortex into two vortices and recurse. This split is performed using the path infrastructure offered by the transaction, along with the cycles of the railed nest and may result in either a "tighter" H-red railed flat vortex around Δ^{int} or in two H-red railed flat vortices. In both cases, this allows us to recurse. As we know by the redrawing lemma, that k such H-red railed flat vortices may give an H-packing, this procedure will end and will produce less than k H-red railed flat vortices, each with some closed disk Δ' defining a bounded depth vortex, as above.

Killing H-red flat vortices. In the next and final step we exploit all the additional structure we obtained via the refinement step and "attack" each of the obtained H-red railed flat vortices separately. For each of them we "kill" all H-red adhesions residing in its internal disk $\Delta^{\text{int}} \subseteq \Delta'$ by identifying a bounded set of vertices drawn within Δ^{int}.

Towards this, recall that the refinement step ensures that the vortex $(G_{\Delta'}, \Omega_{\Delta'})$ has bounded depth. Using a known result of [19], we construct a *bounded width linear decomposition* of $G_{\Delta'}$, that is a path decomposition $\langle X_1, X_2, \ldots, X_n \rangle$ where every bag X_i contains some vertex x_i of the boundary of Δ' in a way that these $x_1, \ldots, x_n$ are the vertices of $V(\Omega_{\Delta'})$, appearing in the same order as they appear in $\Omega_{\Delta'}$. We next partition $\langle X_1, X_2, \ldots, X_n \rangle$ into r segments $\{\langle X_{p_{i-1}}, \ldots, X_{p_i - 1}, X_{p_i} \rangle, i \in [r]\}$ each "minimally capable" to host some H-red adhesion from which an H-inflated copy invades G_t. Likewise, we find equally many H-inflated copies in G where the parts drawn inside Δ' are disjoint. Then we bound the number of these segments by proving that they may be extended to an H-packing of size r, inside Δ^{ext}. For this, we use the full power of the redrawing lemma along with the infrastructure offered by the railed nest. As long as there are less than k segments in $\{\langle X_{p_{i-1}}, \ldots, X_{p_i} \rangle, i \in [r]\}$ we define a carving separation (X, Y) of G where Y contains the union of all $X_{p_{i-1}} \cup X_{p_i}$, $i \in [r]$ and $X \cap Y$ contains the union of all $(X_{p_{i-1}} \cap X_{p_i}) \cup (X_{p_i} \cap X_{p_{i+1}})$, $i \in [r]$. As the size of $X \cap Y$ depends on k and the width of the decomposition (that is bounded), we have that (X, Y) has bounded order. Therefore, we may take a carving of our Σ-embedding at the carving separation (X, Y). When this is done for all H-red flat vortices, we know that what remains from G'_t has a Σ-embedding that has no H-red adhesions.

From local to global. Recall that all above steps were applied to an initial torso G_t and, in particular, to the corresponding Σ-embedding of $G'_t = G_t - A_t$. In the end, what we obtained is a new G'_t and A_t and a Σ-embedding of G'_t with no H-red adhesions. The elimination

of H-red adhesions was done by taking successive carvings of the Σ-embedding of G'_t at a bounded number of carving separations (X, Y), each of bounded order. This came at some cost: By taking these carvings, we added all $X \cap Y$'s to A_t and, moreover, removed all $Y \setminus X$'s from G'. As we already mentioned above, the resulting pair $(V(G_t), A_t)$ is an *H-dominion* of G, which means that if the non-disk-embeddable part of some H-inflated copy in G intersects $V(G_t)$, then it also intersects A_t. At this point we should forget the initial tree decomposition and just keep in mind that we started with a wall W_t of some torso G_t and we finally computed an H-dominion (X_t, A_t) of G where X_t still maintains a big part of the Dyck grid D_t that is the "essential" part of W_t. This constitutes the proof of a "local structure theorem" that, apart from excluding the Dyck graphs in $\{\mathcal{D}_t^\Sigma \mid \Sigma \in \mathsf{sobs}(\mathbb{S}_H)\}$, assumes that G has no H-packing of size k, and, given a big enough wall W, returns an H-dominion (X, A) of G where the "essential part" of W is intact in X. What we need now is to bring this result to the form of a global structure theorem, that is a new tree decomposition (T, β) where each node t is accompanied by a set $\alpha(t) \subseteq \beta(t)$ where $(\beta(t), \alpha(t))$ is an H-dominion of G. This decomposition may serve as the analogue of the tree decomposition in **Step 2**. For this we use an appropriate application of a recent result in [5].

From connected to disconnected. Given the decomposition (T, β) from above, we may now delete adhesions, as it was performed in **Step 2**. After this, we obtain an $\mathcal{Z}$-dominion (X, A) of G such that $G - X$ is H-minor-free and $|A|$ is bounded. With some more preprocessing, this decomposition may be used to obtain a separation (X, Y) of G of bounded order where $(X, X \cap Y)$ is an $\mathcal{Z}$-dominion and $G[Y \setminus X]$ is H-minor-free. Notice that at this point, if H is connected, then we are done. To deal with the case where H is not connected, we set up a recursive algorithm which uses the connected case as the base case and each time it is called, it is called for the union of a smaller number of connected components of H. The final outcome is an H-cover of G whose size depends single-exponentially on the size of the excluded Dyck grids from $\mathfrak{D}_H$ and the size of the maximum H-packing in G.

4 Conclusion and open problems

Obstructions of graph classes. The (minor)-*obstruction set* of a graph class $\mathcal{G}$, denoted by $\mathsf{obs}(\mathcal{G})$, consists of the minor-minimal elements of $\mathcal{G}_{\mathsf{all}} \setminus \mathcal{G}$. Clearly $\mathsf{obs}(\mathcal{G})$ is an antichain. Moreover, it is finite by Robertson's & Seymour's theorem. Obstruction sets permit the following equivalent statement of Theorem 2.

▶ **Theorem 8.** *Let $\mathcal{Z}$ be an antichain in $\mathbb{H}^-$ and let $\mathcal{G}$ be a minor-closed graph class. $\mathcal{Z}$ has the Erdős-Pósa property in $\mathcal{G}$ if and only if, for every surface $\Sigma \in \mathsf{sobs}(\mathbb{S}_{\mathcal{Z}})$, there exists an obstruction in $\mathsf{obs}(\mathcal{G})$ which is Σ-embeddable.*

Universal obstructions. Let $\mathsf{p} : \mathcal{G}_{\mathsf{all}} \to \mathbb{N}$ be a minor-monotone graph parameter. We say that a set $\mathfrak{H}$ of minor-monotone parametric graphs is a (minor-)*universal obstruction* for p if $\mathsf{p} \sim \mathsf{p}_{\mathfrak{H}}$ (recall (3) for the definition of $\mathsf{p}_{\mathfrak{H}}$). Universal obstruction may serve as canonical representations of graph parameters. (For more on the foundation of universal obstructions of parameters, see [31, 32].) From this point of view, Theorem 2 can be restated follows:

▶ **Theorem 9.** *For every $\mathcal{Z} \in \mathbb{H}^-$, the set of parametric graphs $\mathfrak{D}_{\mathcal{Z}} = \{\mathcal{D}^\Sigma \mid \Sigma \in \mathsf{sobs}(\mathbb{S}_{\mathcal{Z}})\} \cup \{\langle k \cdot H \rangle_{k \in \mathbb{N}} \mid H \in \mathcal{Z}\}$ is a universal obstruction for both $\mathsf{cover}_{\mathcal{Z}}$ and $1/2\text{-}\mathsf{pack}_{\mathcal{Z}}$.*

Given some $\mathcal{Z} \in \mathbb{H}$, for every $k \in \mathbb{N}$, we define $\mathcal{C}_k^{\mathcal{Z}} = \{G \mid \mathsf{cover}_{\mathcal{Z}}(G) \leq k\}$. Theorem 9 (or the equivalent Theorem 8) gives us some valuable information about the obstructions in $\mathsf{obs}(\mathcal{C}_k^{\mathcal{Z}})$, for every k.

Certainly, the simplest antichain in $\mathbb{H}^-$ is the one consisting of the two Kuratowski graphs $\mathcal{K} = \{K_5, K_{3,3}\}$. The parameter $\mathsf{cover}_{\mathcal{K}}$ is the *planarizer number* that is the minimum number of vertices whose removal can make a graph planar. The obstruction $\mathsf{obs}(\mathcal{C}_k^{\mathcal{K}})$ is unknown for every positive value of k and its size is expected to grow rapidly as a function of k (see [7] for an exponential lower bound and [40] for a triply exponential upper bound). The identification of $\mathsf{obs}(\mathcal{C}_k^{\mathcal{K}})$ is a non-trivial problem even for small values of k. In particular, it has been studied extensively for the case where $k = 1$ in [25, 28, 47]. In this direction, Mattman and Pierce conjectured that $\mathsf{obs}(\mathcal{C}_k^{\mathcal{K}})$ contains the $Y\Delta Y$-families of K_{n+5} and $K_{3^2,2^n}$ and provided evidence towards this in [11]. Recently, Jobson and Kézdy identified *all* graphs in $\mathsf{obs}(\mathcal{C}_1^{\mathcal{K}})$ of connectivity two in [20], where they also reported that $|\mathsf{obs}(\mathcal{C}_1^{\mathcal{K}})| \geq 401$.

It is easy to see that $\{(k+1)\cdot K_5, (k+1)\cdot K_{3,3}\} \subseteq \mathsf{obs}(\mathcal{C}_k^{\mathcal{K}})$, for every $k \in \mathbb{N}$. Our results, together with the fact that $\mathsf{sobs}(\mathbb{S}_{\mathcal{K}}) = \{\Sigma^{(1,0)}, \Sigma^{(0,1)}\}$, provide the following extra information about $\mathsf{obs}(\mathcal{C}_k^{\mathcal{K}})$: for every $k \in \mathbb{N}$, it contains some graph, say G_k^{t}, embeddable in the torus and some graph, say G_k^{p}, embeddable in the projective plane. Most importantly, our results indicate, that the four-member subset $\{(k+1)\cdot K_5, (k+1)\cdot K_{3,3}, G_k^{\mathsf{t}}, G_k^{\mathsf{p}}\}$ of $\mathsf{obs}(\mathcal{C}_k^{\mathcal{K}})$ is sufficient to determine the approximate behaviour of the planarizer number.

Similar implications can be derived for every $\mathcal{Z} \in \mathbb{H}^-$. For instance, if $\mathcal{P}$ is the Petersen family, we again have that $\mathsf{sobs}(\mathbb{S}_{\mathcal{P}}) = \{\Sigma^{(1,0)}, \Sigma^{(0,1)}\}$. Therefore the parameter defined as the minimum number of vertices to remove so as to make a graph linkless, is approximately characterized by picking only nine graphs of $\mathsf{obs}(\mathcal{C}_k^{\mathcal{P}})$, for every $k \in \mathbb{N}$.

Other examples of surface obstructions corresponding to graphs that are known to be both Kuratowski-connected and shallow-vortex minors are $\mathsf{sobs}(\mathbb{S}_{\{K_5\}}) = \mathsf{sobs}(\mathbb{S}_{\{K_6\}}) = \mathsf{sobs}(\mathbb{S}_{\{M_{2n}\}}) = \{\Sigma^{(1,0)}, \Sigma^{(0,1)}\}$, where M_{2n} is the $2n$-Möbius ladder,[11] for $n \in \mathbb{N}_{\geq 3}$. Two other examples are $\mathsf{sobs}(\mathbb{S}_{\{K_{4,4}\}}) = \{\Sigma^{(1,0)}, \Sigma^{(0,2)}\}$ and $\mathsf{sobs}(\mathbb{S}_{\{K_7\}}) = \{\Sigma^{(1,0)}\}$.

Another implication of our results is the following.

▶ **Theorem 10.** *For every closed and proper set of surfaces $\mathbb{S}$, the set of parametric graphs $\mathfrak{V}_{\mathcal{Z}} = \{\mathcal{D}^{\Sigma} \mid \Sigma \in \mathsf{sobs}(\mathbb{S})\} \cup \{\langle \mathcal{V}_k \rangle_{k\in\mathbb{N}}\}$ is a universal obstruction for $\mathbb{S}$-$\mathsf{tw}_{\mathsf{apex}}$.*

The theorem above is a direct consequence of the second step in our proof outline, that is the step "From small treewidth modulators to small size modulators", where we obtain a local structure theorem for graphs excluding the parametric graphs in $\mathfrak{V}_{\mathcal{Z}}$, i.e., graphs where $\mathsf{p}_{\mathfrak{V}_{\mathcal{Z}}}$ is bounded. The parameter $\mathbb{S}$-$\mathsf{tw}_{\mathsf{apex}}$ (defined in (5)) corresponds to the global version of this theorem. In particular, the equivalence between $\mathbb{S}$-$\mathsf{tw}_{\mathsf{apex}}$ and $\mathsf{p}_{\mathfrak{V}_{\mathcal{Z}}}$ follows directly by [42, Theorem 5.18] or, alternatively, by applying [5, Theorem 6.17]. Notice that $\mathbb{S}$-$\mathsf{tw}_{\mathsf{apex}}$ can be seen as a parametric extension of graph embeddability and that the exclusion of shallow-vortex minors is pivotal for its definition. The potential algorithmic applications of $\mathbb{S}$-$\mathsf{tw}_{\mathsf{apex}}$ are open to investigate.

Notice that for both the equivalences in Theorem 9 and in Theorem 10 we have a single exponential gap which, in turn, determines the gap of our FPT-approximations. Is it possible to reduce this to a polynomial one? Certainly, this requires a polynomial dependency on k and t in Theorem 3. There are two sources of exponentiality in the proof of Theorem 3. The first is in the exclusion of the Dyck grid $\mathcal{D}_t^{\Sigma'}$, for $\Sigma' \in \mathsf{sobs}(\mathbb{S})$ that comes from [43], where we have an exponential dependency on t. This dependency already emerges from the bounds in [19]. On the other hand the exponential dependency on k emerges from the redrawing lemma, where the exponential bound comes from the dependencies of the planar linkage

[11] The *Möbius ladder* M_{2n} is formed if we consider a cycle on $2n$ vertices and then connect by edges the n anti-diametrical pairs of vertices. Notice that $M_6 = K_{3,3}$. M_8 is called the *Wagner Graph*.

theorem in [2]. Avoiding these two sources of exponentiality appears to be a hard task. An alternative approach is to try instead to "enlarge" the size of the universal obstructions to obtain a polynomial parametric graph. This would also be desirable for the purposes of better FPT-approximation algorithms.

Going further than $\mathbb{K} \cap \mathbb{V}$. The central question proposed by this work is to chart the threshold of half-integrality when covering and packing graphs as minors. In this paper we resolved this question for every antichain in $\mathbb{K} \cap \mathbb{V}$. The wide open question is whether and how this can be done for more general families of antichains. For this, one needs to prove structure theorems on the exclusion of parameterized graphs of unbounded genus, as those in Figure 2. The challenges that have to be met for this, when going beyond $\mathbb{K}$, are different from those encountered when going beyond $\mathbb{V}$. We believe that the proof strategy of our paper can serve as a starting point for both directions towards the general case. The resolution of the general case is highly non-trivial and requires new tools and ideas.

We conclude with a conjecture. Our guess is that when we insist on universal obstructions of bounded genus, then we cannot go much further than the horizon of $\mathbb{K} \cap \mathbb{V}$. Let $\mathbb{B}$ be the set of all antichains consisting of graphs where each of them can be embedded in both the torus and the projective plane. As an example, observe that $\{K_{3,4}\} \in \mathbb{B} \setminus \mathbb{K}$, while $\{K_{3,5}\} \notin \mathbb{B}$. We conjecture the following.

▶ **Conjecture 11.** *Let $\mathcal{Z}$ be an antichain and let $\mathsf{EP}_{\mathcal{Z}} : \mathcal{G}_{\mathsf{all}} \to \mathbb{N}$ be a graph parameter such that $\mathcal{Z}$ has the Erdős-Pósa property in a minor-closed graph class $\mathcal{G}$ if and only if $\mathsf{EP}_{\mathcal{Z}}$ is bounded in $\mathcal{G}$. Then $\mathcal{Z} \in (\mathbb{K} \cap \mathbb{V}) \cup \mathbb{B}$ if and only there exists some $g_{\mathcal{Z}}$ such that all universal obstructions of $\mathsf{EP}_{\mathcal{Z}}$ consist of parametric graphs of Euler genus $\leq g_{\mathcal{Z}}$.*

References

1 Colin C. Adams. *The knot book*. American Mathematical Society, Providence, RI, 2004. An elementary introduction to the mathematical theory of knots, Revised reprint of the 1994 original.

2 Isolde Adler, Stavros G. Kolliopoulos, Philipp Klaus Krause, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Irrelevant vertices for the planar disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 122:815–843, 2017. doi:10.1016/j.jctb.2016.10.001.

3 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. A complexity dichotomy for hitting connected minors on bounded treewidth graphs: the chair and the banner draw the boundary. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 951–970, 2020. doi:10.1137/1.9781611975994.57.

4 Hans L. Bodlaender. On disjoint cycles. *International Journal of Foundations of Computer Science*, 5(1):59–68, 1994. doi:10.1142/S0129054194000049.

5 Rutger Campbell, J. Pascal Gollin, Kevin Hendrey, and Sebastian Wiederrecht. Bipartite treewidth – The structure of non-zero cycles in group-labelled graphs, 2023, Manuscript submitted to SODA 2024, private communication.

6 Radu Curticapean and Mingji Xia. Parameterizing the permanent: Hardness for fixed excluded minors. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 297–307. SIAM, 2022. doi:10.1137/1.9781611977066.23.

7 Michael J. Dinneen. Too many minor order obstructions (for parameterized lower ideals). *JUCS – Journal of Universal Computer Science*, 3(11):1199–1206, 1997. doi:10.3217/jucs-003-11-1199.

8 Walther Dyck. Beiträge zur Analysis situs. *Mathematische Annalen*, 32(4):457–512, 1888. doi:10.1007/BF01443580.

9 P. Erdős and L. Pósa. On independent circuits contained in a graph. *Canadian Journal of Mathematics*, 17:347–352, 1965. doi:10.4153/CJM-1965-035-8.

10 Michael R Fellows and Michael A Langston. Nonconstructive tools for proving polynomial-time decidability. *Journal of the ACM*, 35(3):727–739, 1988. doi:doi.org/10.1145/44483.44491.

11 Erica Flapan, Allison Henrich, Aaron Kaestner, and Sam Nelson. *Knots, links, spatial graphs, and algebraic invariants*, volume 689 of *Contemporary Mathematics*. American Mathematical Society, Providence, RI, 2017. doi:10.1090/conm/689.

12 Fedor V Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Hitting topological minors is FPT. In *ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1317–1326, 2020. doi:10.1145/3357713.3384318.

13 Cyril Gavoille and Claire Hilaire. Minor-universal graph for graphs on surfaces, 2023. arXiv:2305.06673.

14 Jim Geelen, Bert Gerards, Bruce Reed, Paul Seymour, and Adrian Vetta. On the odd-minor variant of hadwiger's conjecture. *Journal of Combinatorial Theory, Series B*, 99(1):20–29, 2009. doi:10.1016/j.jctb.2008.03.006.

15 Petr A. Golovach, Giannos Stamoulis, and Dimitrios M. Thilikos. Hitting topological minor models in planar graphs is fixed parameter tractable. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 931–950, 2020. doi:10.1137/1.9781611975994.56.

16 Petr A. Golovach, Giannos Stamoulis, and Dimitrios M. Thilikos. Combing a linkage in an annulus, 2022. arXiv:2207.04798.

17 Petr A. Golovach, Giannos Stamoulis, and Dimitrios M. Thilikos. Hitting topological minor models in planar graphs is fixed parameter tractable. *ACM Transactions on Algorithms*, 19(3):23:1–29, 2023. doi:10.1145/3583688.

18 Tony Huynh, Felix Joos, and Paul Wollan. A unified Erdős–Pósa theorem for constrained cycles. *Combinatorica*, 39(1):91–133, 2019. doi:10.1007/s00493-017-3683-z.

19 Ken ichi Kawarabayashi, Robin Thomas, and Paul Wollan. Quickly excluding a non-planar graph, 2021. arXiv:2010.12397.

20 Adam S. Jobson and André E. Kézdy. All minor-minimal apex obstructions with connectivity two. *Electronic Journal of Combinatorics*, 28(1):1.23, 58, 2021. doi:10.37236/8382.

21 Naonori Kakimura, Ken-ichi Kawarabayashi, and Dániel Marx. Packing cycles through prescribed vertices. *Journal of Combinatorial Theory, Series B*, 101(5):378–381, 2011. doi:10.1016/j.jctb.2011.03.004.

22 Ken-ichi Kawarabayashi. Half integral packing, Erdős-Pósa-property and graph minors. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1187–1196, 2007. URL: dl.acm.org/citation.cfm?id=1283383.1283511.

23 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce A. Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2):424–435, 2012. doi:10.1016/j.jctb.2011.07.004.

24 Ken-ichi Kawarabayashi and Stephan Kreutzer. The directed grid theorem. In *ACM Symposium on Theory of Computing (STOC)*, pages 655–664, 2015. doi:10.1145/2746539.2746586.

25 Max Lipton, Eoin Mackall, Thomas W. Mattman, Mike Pierce, Samantha Robinson, Jeremy Thomas, and Ilan Weinschelbaum. Six variations on a theme: almost planar graphs. *Involve. A Journal of Mathematics*, 11(3):413–448, 2018. doi:10.2140/involve.2018.11.413.

26 Chun-Hung Liu. Packing topological minors half-integrally. *Journal of the London Mathematical Society*, 106(3):2193–2267, 2022. doi:10.1112/jlms.12633.

27 Dániel Marx and Ildikó Schlotter. Obtaining a planar graph by vertex deletion. *Algorithmica*, 62(3-4):807–822, 2012. doi:10.1007/s00453-010-9484-z.

28 Thomas W. Mattman. Forbidden minors: finding the finite few. In *A primer for undergraduate research*, Found. Undergrad. Res. Math., pages 85–97. Birkhäuser/Springer, Cham, 2017.

29 Laure Morelle, Ignasi Sau, Giannos Stamoulis, and Dimitrios M. Thilikos. Faster Parameterized Algorithms for Modification Problems to Minor-Closed Classes. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 93:1–93:19, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2023.93.

30 Sergey Norin, Robin Thomas, and Hein van der Holst. On 2-cycles of graphs. *Journal of Combinatorial Theory. Series B*, 162:184–222, 2023. doi:10.1016/j.jctb.2023.06.003.

31 Christophe Paul, Evangelos Protopapas, and Dimitrios M. Thilikos. Graph parameters, universal obstructions, and wqo, 2023. arXiv:2304.03688.

32 Christophe Paul, Evangelos Protopapas, and Dimitrios M. Thilikos. Universal obstructions of graph parameters, 2023. arXiv:2304.14121.

33 Jean-Florent Raymond and Dimitrios M. Thilikos. Recent techniques and results on the Erdős-Pósa property. *Discrete Applied Mathematics*, 231:25–43, 2017. doi:10.1016/j.dam.2016.12.025.

34 Bruce Reed. Mangoes and blueberries. *Combinatorica*, 19(2):267–296, 1999. doi:10.1007/s004930050056.

35 Bruce Reed, Neil Robertson, Paul Seymour, and Robin Thomas. Packing directed circuits. *Combinatorica*, 16(4):535–554, 1996. doi:10.1007/BF01271272.

36 Neil Robertson and P. D. Seymour. Graph minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, 1986. doi:10.1016/0095-8956(86)90030-4.

37 Neil Robertson and P. D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. doi:10.1016/j.jctb.2004.08.001.

38 Neil Robertson, Paul Seymour, and Robin Thomas. Sach's linkless embedding conjecture. *Journal of Combinatorial Theory, Series B*, 64(2):185–227, 1995. doi:10.1006/jctb.1995.1032.

39 Ignasi Sau, Giannos Stamoulis, and Dimitrios M. Thilikos. k-apices of minor-closed graph classes. II. Parameterized algorithms. *ACM Transactions on Algorithms*, 18(3):Art. 21, 30, 2022. doi:10.1145/3519028.

40 Ignasi Sau, Giannos Stamoulis, and Dimitrios M. Thilikos. k-apices of minor-closed graph classes. I. Bounding the obstructions. *Journal of Combinatorial Theory, Series B*, 161:180–227, 2023. doi:10.1016/j.jctb.2023.02.012.

41 Dimitrios M. Thilikos and Sebastian Wiederrecht. Killing a vortex. In *IEEESymposium on Foundations of Computer Science (FOCS)*, pages 1069–1080, 2022. doi:10.1109/FOCS54457.2022.00104.

42 Dimitrios M. Thilikos and Sebastian Wiederrecht. Approximating branchwidth on parametric extensions of planarity, 2023. arXiv:2304.04517.

43 Dimitrios M. Thilikos and Sebastian Wiederrecht. Excluding surfaces as minors in graphs, 2023. arXiv:2306.01724.

44 Robin Thomas. Well-quasi-ordering infinite graphs with forbidden finite planar minor. *Transactions of the American Mathematical Society*, 312(1):279–313, 1989. doi:10.2307/2001217.

45 Wouter Cames Van Batenburg, Tony Huynh, Gwenaël Joret, and Jean-Florent Raymond. A tight Erdős-Pósa function for planar minors. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1485–1500, 2019. doi:10.1137/1.9781611975482.90.

46 Hein van der Holst. A polynomial-time algorithm to find a linkless embedding of a graph. *Journal of Combinatorial Theory, Series B*, 99(2):512–530, 2009. doi:10.1016/j.jctb.2008.10.002.

47 Yaming Yu. More forbidden minors for Wye-Delta-Wye reducibility. *Electronic Journal of Combinatorics*, 13(1):7:15, 2006. doi:10.37236/1033.

On the Cut-Query Complexity of Approximating Max-Cut

Orestis Plevrakis
Department of Computer Science, Princeton University, NJ, USA

Seyoon Ragavan
Computer Science and Artificial Intelligence Lab,
Massachusetts Institute of Technology, Cambridge, MA, USA

S. Matthew Weinberg
Department of Computer Science, Princeton University, NJ, USA

Abstract

We consider the problem of query-efficient global max-cut on a weighted undirected graph in the value oracle model examined by [31]. Graph algorithms in this cut query model and other query models have recently been studied for various other problems such as min-cut, connectivity, bipartiteness, and triangle detection. Max-cut in the cut query model can also be viewed as a natural special case of submodular function maximization: on query $S \subseteq V$, the oracle returns the total weight of the cut between S and $V \backslash S$.

Our first main technical result is a lower bound stating that a deterministic algorithm achieving a c-approximation for any $c > 1/2$ requires $\Omega(n)$ queries. This uses an extension of the cut dimension to rule out approximation (prior work of [20] introducing the cut dimension only rules out exact solutions). Secondly, we provide a randomized algorithm with $\tilde{O}(n)$ queries that finds a c-approximation for any $c < 1$. We achieve this using a query-efficient sparsifier for undirected weighted graphs (prior work of [31] holds only for unweighted graphs).

To complement these results, for most constants $c \in (0, 1]$, we nail down the query complexity of achieving a c-approximation, for both deterministic and randomized algorithms (up to logarithmic factors). Analogously to general submodular function maximization in the same model, we observe a phase transition at $c = 1/2$: we design a deterministic algorithm for global c-approximate max-cut in $O(\log n)$ queries for any $c < 1/2$, and show that any randomized algorithm requires $\Omega(n/\log n)$ queries to find a c-approximate max-cut for any $c > 1/2$. Additionally, we show that any deterministic algorithm requires $\Omega(n^2)$ queries to find an exact max-cut (enough to learn the entire graph).

2012 ACM Subject Classification Mathematics of computing → Approximation algorithms

Keywords and phrases query complexity, maximum cut, approximation algorithms, graph sparsification

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.115

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2211.04506

Funding *Seyoon Ragavan*: Supported by an Akamai Presidential Fellowship.
S. Matthew Weinberg: Supported by NSF CCF-1955205.

Acknowledgements SR would like to thank Georgy Noarov and Dmitry Paramonov for collaborating on a course project that led to this work. We would also like to thank Sepehr Assadi for helpful discussions and anonymous reviewers for feedback and suggested changes.

1 Introduction

For the most part, the field of graph algorithms has focused on extensive study in the "standard" model where the entire graph is given to the algorithm as input e.g. in the form of an adjacency list or an adjacency matrix. However, another growing body of work focuses on the case where the algorithm does not know the graph; rather, it has query access to an

oracle that knows the graph, and aims to make as few queries as possible. Different models restrict the algorithm to make different types of queries e.g. individual edge queries, linear measurements [1, 5], matrix-vector products [32], etc.

One model that has attracted particular attention is the *cut query model* [21, 15, 31, 20, 30, 28, 4, 14]. In this model, the algorithm has query access to the cut function of a graph G with vertex set $[n]$: it provides subsets $S \subseteq [n]$ as queries and the oracle returns the total weight of all edges of G with exactly one endpoint in S. An additional reason for the wide interest in this model is that the graph's cut function is in particular a submodular function; [20] and [28] obtained new lower bounds for the query complexity of submodular function minimization by considering the special case of graph min-cut in the cut query model.

A range of graph problems has been studied in the cut query model and related models. Perhaps the most fundamental is that of learning the entire graph [21, 16, 9, 15, 10]; once the entire graph is known to the algorithm, it has all the standard graph algorithm literature at its disposal to solve the problem of interest. A more subtle question is whether the given problem can be solved with fewer queries than one would need to learn the graph. This was first answered affirmatively for the case of min-cut [31, 30, 4]: an undirected, unweighted graph requires $\Omega(n^2/\log n)$ queries to learn [21] but $O(n)$ queries suffice to find its min-cut. Since then, novel algorithms have been developed in the cut query model and related models for problems such as connectivity [1, 32, 5, 4, 14], testing bipartiteness [1], and triangle detection [32].

In this work, we initiate the study of max-cut in the cut-query model. We aim to understand how many queries are necessary and sufficient for an algorithm to find a c-approximate global max-cut in an undirected, weighted graph.

1.1 Our Results

Our two main results are stated below. Both are concerned with the setting where $c \in (1/2, 1)$ i.e. one wants to do better than a straightforward greedy algorithm or guessing a random cut, but does not want an exact max-cut. The first result is a lower bound against deterministic algorithms, and the second result is a randomized algorithm.

▶ **Theorem 1** (See Corollary 7 for a precise statement). *For $c > 1/2$, any deterministic algorithm achieving a c-approximation requires $\Omega(n)$ queries.*

▶ **Theorem 2** (See Corollary 29 for a precise statement). *For $c < 1$, there exists a randomized algorithm with query complexity $\tilde{O}(n)$ that achieves a c-approximation.*

These results naturally beg the question of how the query complexity behaves for other ranges of c, for both deterministic and randomized algorithms. We also answer this question in most cases. The content of all of our results is summarized in the following three theorems, and also tabulated in Figure 1. Soon after, we provide context for these results.

▶ **Theorem 3** (See Theorems A.1 and B.1 in the full version for precise statements). *For $c = 1$, the query complexity for a deterministic algorithm to achieve a c-approximation is $\Theta(n^2)$ and the query complexity for a randomized algorithm to do the same is between $\tilde{\Omega}(n)$ and $O(n^2)$.*

▶ **Theorem 4** (See Corollary 29, Corollary 7, and Theorem B.1 in the full version for precise statements). *For $c \in (1/2, 1)$, the query complexity for a deterministic algorithm to achieve a c-approximation is between $\Omega(n)$ and $O(n^2)$ and the query complexity for a randomized algorithm to do the same is $\tilde{\Theta}(n)$.*

c	Deterministic	Randomized
1	$\Theta(n^2)$	$(\tilde{\Omega}(n), O(n^2))$
$(1/2, 1)$	$(\Omega(n), O(n^2))$	$\tilde{\Theta}(n)$
$(0, 1/2)$	$\Theta(\log n)$	$\Theta(1)$

Figure 1 Summary of our results. For each range of c, we state the query complexity (up to constant and logarithmic factors) that we show for achieving a c-approximation in both the deterministic and randomized settings. (n, n^2) indicates settings where we have a lower bound of $\tilde{\Omega}(n)$ and an upper bound of $O(n^2)$.

▶ **Theorem 5** (See Corollary E.2, Theorem E.4, and Corollary C.3 in the full version for precise statements). *For $c \in (0, 1/2)$, the query complexity for a deterministic algorithm to achieve a c-approximation is $\Theta(\log n)$ and the query complexity for a randomized algorithm to do the same is $\Theta(1)$.*

We now provide context for each of our results, beginning with Theorem 1. Observe that there is a straightforward non-adaptive $O(n^2)$-query deterministic algorithm to find the max-cut exactly (observed in [31], as it applies to the min-cut as well). The algorithm can query all singletons and sets of size 2 and from this learn $w_{i,j} = \frac{1}{2}(F(\{i\}) + F(\{j\}) - F(\{i,j\}))$ for all i, j and thus the entire graph. The algorithm can then find the max cut exactly using brute force, since no more queries need to be made. This addresses all of the $O(n^2)$ upper bounds stated in the theorems. Theorem 3 shows that this trivial algorithm is optimal among deterministic algorithms: up to constant factors, any deterministic algorithm must learn the entire graph in order to find the global max-cut with cut queries.

Next, observe that there is a phase transition at $c = 1/2$. Theorem 5 establishes that a $(1/2 - \varepsilon)$-approximation can be achieved deterministically with $O(\log n)$ queries (which happen to also be necessary). On the other hand, even a randomized algorithm needs $\tilde{\Omega}(n)$ queries to guarantee a $(1/2 + \varepsilon)$-approximation.

Finally, observe that we resolve the asymptotic query complexity for most cases. For randomized algorithms, the only unresolved cases are $c = 1/2$ and $c = 1$.[1] For deterministic algorithms, the range $[1/2, 1)$ remains unresolved.

Before continuing, we note several facts about our results. In the positive direction, we note that all of our algorithms find a set S that is a c-approximation to the global max-cut (for undirected, weighted graphs with arbitrarily large weights), rather than simply estimating the value of the max-cut within a factor of c. We refer to these two settings as the *cut finding* and *value estimation* settings respectively. On the other hand, most of our lower bounds hold even against algorithms in the value estimation setting. The only exception is our $\tilde{\Omega}(n)$ lower bound on randomized algorithms for $c > 1/2$, which we only prove in the cut finding setting. Similarly, all algorithms we discuss, except for our $\tilde{O}(n)$-query randomized algorithm for $c \in (1/2, 1)$ and the well-known $O(n)$-query deterministic greedy algorithm for $c = 1/2$, are non-adaptive. On the other hand, all of our lower bounds hold even against adaptive algorithms.

It may appear at first glance that Theorem 1 is subsumed by the randomized lower bound in Theorem 4. Both results are lower bounds for $c > 1/2$: the former states that a deterministic algorithm in this setting requires $\Omega(n)$ queries, while the latter says that

[1] For the $c = 1/2$ case, simply outputting a random cut will achieve a $(1/2)$-approximation in expectation in $O(1)$ queries. But we are concerned with the algorithm's ability to achieve a c-approximation with some probability $p > 0$. In this setting, there exists a deterministic $O(n)$-query algorithm (see Section E.3 of the full version) but we do not have a matching lower bound.

a randomized algorithm requires $\tilde{\Omega}(n)$ queries. However, the deterministic lower bound is interesting on its own for two reasons. The first reason is that the randomized lower bound is only $\Omega(n/\log n)$, which is weaker than the $\Omega(n)$ we are able to prove in the deterministic case. Secondly, and more critically, the deterministic lower bound holds even for the value estimation setting while the randomized lower bound is only for the cut finding setting. Our argument for the randomized lower bound does not appear adaptable to the value estimation setting; indeed, the hard distribution used in our proof is actually very easy in the value estimation setting.[2]

In the negative direction, we note here that most of our algorithms have exponential time complexity even though they are query-efficient. For example, the $O(n^2)$-query algorithm we just described learns the graph in polynomial time, but then uses exponential brute force search to actually find its max-cut. This illustrates that, by focusing on query complexity, we are primarily concerned with the information theoretic question of how much the algorithm must learn about the graph to be able to estimate its max-cut, even with unlimited computation. Note that the most query-efficient algorithms for submodular function minimization are exponential time as well [24], so this is not uncommon when prioritizing query complexity.

1.2 Technical Highlights

Our results follow from a wide array of techniques. Some results (e.g. our $\tilde{\Omega}(n)$ lower bound on randomized algorithms for c-approximate max-cut and $c > 1/2$) follow from direct constructions of hard distributions – we defer further discussion of these constructions to the corresponding technical sections and appendices. Some of our results follow from novel techniques that are likely of independent interest for subsequent work – we quickly highlight these below.[3]

The key ingredient in our $\Omega(n^2)$-lower bound for exact max-cut is the *cut dimension* introduced by [20] for min-cuts (which has also been studied in follow-up work [28]). The lower bound follows immediately after establishing that the complete graph on n vertices has max-cut dimension $\Omega(n^2)$. More interestingly, we extend the concept of the cut-dimension to what is roughly a notion of "c-approximate cut-dimension" using strong LP duality. We show that this technique provides a lower bound on the number of queries needed by a deterministic algorithm to find a c-approximate max-cut for $c > 1/2$, and that for the complete graph on n vertices this gives a bound of $\Omega(n)$. This approach should also be of independent interest, given recent interest in the exact cut dimension. Completing this analysis also requires a technical lemma (Lemma 13) stating a simple geometric property of the Boolean hypercube, which may additionally be of independent interest.

Our $\tilde{O}(n)$-query randomized c-approximation for $c \in (1/2, 1)$ follows from a query-efficient sparsifier for global cuts in undirected, weighted graphs (once we have learned a sparsifier for the graph, we can just exhaust to find its max-cut, which is a $(1-\varepsilon)$-approximation). A query-efficient sparsifier for unweighted graphs appears in [31], based on ideas by [7]. We first give a natural extension of their sparsifier that accommodates graphs with weights in $[1, \text{poly}(n)]$. We then go a step further and provide a novel query-efficient sparsifier for *all*

[2] In more detail, the hard distribution is a random complete bipartite graph with edge weights 1. For this distribution in the value estimation setting, an algorithm could just immediately output $n^2/4$ without making any queries at all.

[3] Additionally, some of our results are reasonably straightforward (e.g. our $O(\log n)$ deterministic algorithm for c-approximate max-cut when $c < 1/2$) – we include discussion of these results in Appendices B, C, and E of the full version for completeness.

weighted graphs, using other ideas from [7] relating edge strengths to maximum spanning trees. For completeness, we show in Appendix D of the full version that in fact a slight modification of the [31] sparsifier also suffices for approximate max-cut for all weighted graphs (however, this modification may not produce a sparsifier). Our stronger sparsifier is hence not "necessary" to resolve approximate max-cut, but is of independent interest as not all weighted graph problems can be reduced to one with weights in $[1, \text{poly}(n)]$ (e.g. exact min-cut). In fact, our stronger sparsifier is already used in recent work [30].[4]

1.3 Related Work

Learning graphs in the cut-query model. The first natural question when an algorithm has restricted query access to a graph is how many queries it needs to learn the entire graph. For the case of cut queries, this was first studied by [21], who show that the query complexity of deterministically learning an unweighted graph is $\Theta(n^2/\log n)$, and that this can be attained non-adaptively. A later line of work [16, 9, 15, 10] answered this question for weighted graphs while also being sensitive to the number m of edges in the graph with nonzero weight, finding that the query complexity in this case is $\Theta(m \log n/\log m)$.

Min-cut in the cut-query model. A very active line of work is that of understanding what properties an algorithm can learn about a graph that it only has query access to, without using up enough queries to learn the entire graph. The particular case of most relevance to this work is that of min-cut in the cut-query model. The first progress on this problem was the algorithm by [31] for *unweighted* graphs using $\tilde{O}(n)$ queries. This was later improved to $O(n)$ queries by [4]. For *weighted* graphs, an algorithm using $\tilde{O}(n)$ cut queries was presented by [30] (as previously noted, their result leverages a query-efficient sparsifier for weighted graphs, which our paper is the first to provide).

On the lower bound side, [20] show using a linear algebraic argument that $\Omega(n)$ queries are necessary for weighted graphs (the constant-factors have since been improved by [28]), thus resolving the query complexity of exact min-cut in weighted graphs up to logarithmic factors. As mentioned earlier, our deterministic lower bounds arise from adapting and extending the concept of the cut dimension to approximation.

The cut dimension was introduced by [20], and further studied in [28]. [28] further nail down that undirected graphs have a min-cut dimension of no more than $2n - 3$ (and there exist graphs realizing this bound). They also introduce the ℓ_1-approximate cut dimension. The ℓ_1-approximate cut dimension is motivated by a similar thought experiment as our "c-approximate cut dimension" technique – both consider an adversary changing the original graph in a way that respects all queries made, and both consider the magnitude of the changes made to the max/min-cut (rather than just whether or not the cut is changed). However, their ℓ_1-approximate cut dimension does not imply lower bounds on the query complexity of finding approximately-optimal cuts. Finally, they further consider "the dimension of approximate min-cuts", and prove a linear upper bound on this. This concept does not have any relation to our approximate cut dimension technique (and in particular, only our technique implies lower bounds on the deterministic query complexity). In summary, the cut dimension is an active concept of study in related work, but our work is the first to use this concept to lower bound the query complexity of approximately-optimal cuts.

[4] [30, Theorem 5.2] seems to credit [31] for handling weighted graphs (whereas their query-efficient sparsifier only accommodates unweighted graphs). They use this result as part of their $\tilde{O}(n)$ query algorithm for *weighted* min-cut.

Other graph algorithms in other query models. Many works have considered other graph problems in other query models as well. These query models include cut queries [14], linear measurements [1, 5], matrix-vector products [32], OR queries [5, 8], XOR queries [8], and AND queries [8]. Some of these models e.g. linear measurements and matrix-vector products generalize the cut query model, while the others do not appear directly related. These works also consider a variety of graph problems, including connectivity [1, 32, 5, 4, 14], bipartiteness [1], triangle detection [32], minimum spanning tree [1], and maximum matching [1, 8].

Graph sparsification. [31] also provide an algorithm for graph sparsification in $\tilde{O}(n)$ queries for unweighted graphs, based on the idea of *edge strength-based sampling* used by [7] to construct sparsifiers in the classical computational model. There are also other lines of work that adapt the ideas from [7] to construct sparsifiers in other limited-access computational models, particularly (semi-)streaming [1, 2, 3, 25, 32]. Some of these algorithms are adaptable to the cut-query model. However, these algorithms either only support unweighted graphs or incur a performance cost for weighted graphs. For example, the natural extension of the algorithm in [31] to weighted graphs requires $\tilde{O}(n \log W)$ queries, where W is the ratio between the largest and smallest nonzero edge weight in the graph. Our novel sparsifier avoids this pitfall by using results from [7] connecting edge strengths to maximum spanning trees to remove the $\log W$ factor.

While these results and ours all use randomization to construct a sparsifier, there are also classical algorithms for constructing sparsifiers deterministically [6, 17]. Implementing such an algorithm directly would require being able to learn the spectrum of the graph's Laplacian. It may indeed be possible to do this (exactly or approximately) in a query-efficient way, but doing so would require significantly new ideas. A deterministic query-efficient implementation of these algorithms would also yield a deterministic query-efficient algorithm for a $(1-\epsilon)$-approximation to max cut.

Max-cut in the classical computational model. Max-cut is extremely well-studied in the computational model, but algorithms in the cut-query model differ significantly. For example, [19] present a randomized polynomial-time algorithm using semidefinite programming that achieves a ≈ 0.878 approximation in expectation. However, this algorithm does not imply anything in our model; even formulating the necessary SDP requires access to the entire graph.

It has since been shown by [23] that it is NP-hard to beat a $16/17 \approx 0.941$ approximation. Moreover, [27] show that it is NP-hard to beat the constant achieved by Goemans and Williamson assuming the unique games conjecture [26]. These lower bounds do not imply anything in our model either. In fact, many of our algorithms (and those in prior work) involve exponential computation even if they use an efficient number of queries. Once we have either learned the entire graph ($c = 1$) or constructed a sparsifier of the graph ($c \in (1/2, 1)$), we use brute force search to find the exact max cut on the graph we have learned.

Submodular function maximization. Our problem can be viewed as a special case of symmetric submodular function maximization, by taking $f(\cdot)$ to be the cut function of a graph. However, the additional structure imposed by the cut function of a graph opens up possibilities for new algorithms and requires new lower bound arguments. For example, when $c > 1/2$, [18] show that achieving a c-approximation to symmetric submodular function maximization requires $\exp(\Omega(n))$ queries to the oracle. For global max-cut, the situation

is significantly different: $O(n^2)$ queries suffice to trivially learn the entire function (and therefore, there seems to be little hope of embedding hard SFM instances as graphs). Still, we show that this trivial algorithm is optimal among deterministic algorithms when $c = 1$, and show that $\tilde{\Omega}(n)$ queries are necessary for even a randomized algorithm to achieve a c-approximation for $c > 1/2$.

For $c \leq 1/2$, randomized [12] and deterministic [11] algorithms are known that achieve a $1/2$-approximation with $O(n)$ and $O(n^2)$ queries respectively, even when the submodular function is not symmetric. [18] show in the symmetric case that picking a random set ($O(1)$ queries) achieves a $1/2$-approximation in expectation. They also provide a deterministic algorithm based on local search that achieves a c-approximation for any $c < 1/2$ in $\tilde{O}(n^3)$ queries. When restricting to max-cut, [18] immediately implies a randomized algorithm for all $c \leq 1/2$ with $O(1)$ queries (which is optimal). For deterministic algorithms, however, our bound of $\Theta(\log n)$ is again an exponential improvement over the general case of submodular function maximization.

In summary, global max-cut demonstrates a similar phase transition at $c = 1/2$ as general submodular function maximization: the query complexity for $c < 1/2$ is exponentially smaller than the query complexity for $c > 1/2$. However, the distinction for max-cut is between logarithmic and polynomial queries, whereas the distinction for general submodular functions is between polynomial and exponential.

Submodular function minimization. Just as submodular function maximization generalizes max-cut in the cut query model, submodular function minimization generalizes min-cut. Unlike the maximization case which requires exponentially many queries, general submodular function minimization can be solved exactly with polynomially many queries; state-of-the-art algorithms use $\tilde{O}(n^2)$ queries [24, 29].

However, until relatively recently, most query lower bounds for submodular function minimization were only $\Omega(n)$ [22, 20, 28]. The last two results proceed using the aforementioned cut dimension technique. This was recently improved to $\Omega(n \log n)$ by [13], using different ideas unrelated to graph cuts.

2 Preliminaries

G is an undirected, weighted graph with weights $w_{i,j}$ on the edge (i, j). G induces a cut function F. We denote the cut function by $F(\cdot)$, so $F(S) := \sum_{i \in S, j \notin S} w_{i,j}$. Additionally, for any cut $S \subseteq [n]$, we define the indicator vector $v_S \in \mathbb{R}^{\binom{n}{2}}$ by $(v_S)_{i,j} = 1$ if (i, j) crosses the cut defined by S and 0 otherwise.

We consider algorithms that have black-box access to $F(\cdot)$, and aim to find $\arg\max_{S \subseteq [n]}\{F(S)\}$ (exact maximization) or a set T such that $F(T) \geq c \cdot \max_{S \subseteq [n]}\{F(S)\}$ (c-approximation). We refer to the query complexity as the number of queries to $F(\cdot)$ that the algorithm makes (the algorithm has no other access to G or F, and may perform unlimited computation).

3 Lower Bound for Deterministic $(1/2 + \epsilon)$-approximation

In this section, we extend the cut dimension technique by [20] using linear programming and the strong duality theorem to show the deterministic hardness part of Theorem 4. (We refer the reader to Appendix A of the full version for a discussion of this technique and its direct application to show the deterministic lower bound in Theorem 3.) Our exact result is as follows:

▶ **Theorem 6.** *Suppose we have* $c \in (1/2, 1), \epsilon \in (0, 1), \alpha > 0$ *such that* $c > \frac{1}{1+\epsilon^2}$ *and* $\alpha < \frac{(1-\epsilon)^3}{108(1+\epsilon)}$. *Then for* n *sufficiently large,* αn *queries do not suffice for a deterministic algorithm to estimate the max cut value within a factor of* c.

Before proving this theorem, we state a cleaner lower bound as a corollary:

▶ **Corollary 7.** *For* $c \in (1/2, 1)$, *a deterministic algorithm that estimates the max cut value within a factor of* c *requires at least* $n(\frac{(\sqrt{c}-\sqrt{1-c})^4}{108c(2c-1)} - o(1))$ *queries.*

This implies that the query complexity for a deterministic algorithm to achieve a c*-approximation for global max-cut on a weighted undirected graph in the value estimation setting is* $\Omega(n)$.

Proof. See Appendix F.1 of the full version. ◀

The first step is conceptually similar to the cut dimension argument from Theorem A.1 in the full version. We consider an adversary that answers all queries as if the graph were K_n (there is an edge of weight 1 between all pairs of vertices). Then we would like to find a perturbation z to the weight vector of K_n such that $w, w + z$ agree on all queries but have differing max cut values. The difference here is that we would like to show that the algorithm cannot even achieve a c-approximation, so we require the perturbation to be so large that the max cut value of $w + z$ is a multiplicative factor greater than the max cut of w. To do this, we will have to go beyond linear algebraic tools and consider linear programming instead. Our argument comprises the following steps:

1. Write the conditions we require of the perturbation z as linear constraints and thus formulate a linear program LP1, which we would like to show has high value.
2. LP1 works with vectors in $\mathbb{R}^{\binom{n}{2}}$ that represent cuts, which are unwieldy and unnatural. Rewrite this in terms of indicator vectors in $\mathbb{R}^n$.
3. Define another linear program LP2 and show that a high value for LP2 implies that LP1 must also have high value.
4. Show that LP2 has high value by taking its dual, and showing that the dual has high value. This comes down to showing a key technical lemma, which essentially states that the n-dimensional hypercube cannot be covered by an ℓ_1 neighborhood of an αn-dimensional subspace of $\mathbb{R}^n$. We show this using an ℓ_1 ϵ-net argument.

We now work through each step in detail. We retain all notation used in Appendix A of the full version for the cut dimension argument, and introduce additional notation as necessary.

3.1 Step 1: Formulating LP1

Throughout these proofs, we use $\mathbf{1}$ to denote a vector with all entries equal to 1 in either $\mathbb{R}^n$ or $\mathbb{R}^{\binom{n}{2}}$. It will be clear from context which of these we are referring to at any given time, but for now we are taking $\mathbf{1} \in \mathbb{R}^{\binom{n}{2}}$ to denote the weight vector of K_n. Let $q = \alpha n$ and $Q_1, Q_2, \ldots, Q_q \subseteq [n]$ be the q queried cuts. As in Appendix A of the full version, they have the corresponding 0/1 indicator vectors $v_{Q_1}, v_{Q_2}, \ldots, v_{Q_q} \in \mathbb{R}^{\binom{n}{2}}$. We are interested in finding a perturbation z such that $\mathbf{1}, \mathbf{1} + z$ are both weighted, undirected graphs (with non-negative edge weights) and agree on all queries but have max cut values differing by a multiplicative factor.

First, we require $\mathbf{1} + z$ to define a valid graph i.e. its entries should all be non-negative since these correspond to edge weights:

$$\mathbf{1} + z \geq 0 \Leftrightarrow z \geq -\mathbf{1}. \tag{1}$$

Next, we need $\mathbf{1}, \mathbf{1} + z$ to agree on all queries. This guarantees that the algorithm cannot tell the difference between $\mathbf{1}$ and $\mathbf{1} + z$ based only on the queries made so far.

$$\mathbf{1}^T v_{Q_i} = (\mathbf{1} + z)^T v_{Q_i},\ \forall i \Leftrightarrow z^T v_{Q_i} = 0,\ \forall i. \tag{2}$$

Finally, we would like the graph corresponding to $\mathbf{1} + z$ to have a much larger max cut value than the graph corresponding to $\mathbf{1}$. We capture this in the following definition and lemma:

▶ **Definition 8.** *Define a* near-max cut *to be any cut* $C \subseteq [n]$ *such that* $n/2 - \sqrt{n \log n} \leq |C| \leq n/2 + \sqrt{n \log n}$.

Note that a near-max cut is nearly a max-cut in K_n. As hinted at earlier, we will show that we can find a near-max cut C and a perturbation z to the graph that will preserve the value of all queries while blowing up the value of C by a factor of c. In this case, the algorithm cannot distinguish between K_n and the perturbed graph and thus cannot achieve a c-approximation.

▶ **Lemma 9.** *To prove Theorem 6, it suffices to show that there exists a near-max cut* C *such that the following linear program has value at least* $\epsilon^2 n^2/4$. *We call this linear program LP1.*

$$\begin{aligned} &\textit{Maximize } z^T v_C \\ &\textit{subject to } z^T v_{Q_j} = 0 \quad \textit{for all } j \in [q], \textit{ and} \\ &\qquad\qquad z \geq -\mathbf{1}. \end{aligned}$$

Proof. We have already explained how the constraints arise. To justify that the objective corresponds to a bound on the approximation ratio, see Appendix F.2 of the full version. ◀

3.2 Step 2: Rewriting LP1

As already mentioned, the vectors $v_S \in \mathbb{R}^{\binom{n}{2}}$ are unnatural and difficult to work with. Intuitively, the reason for this is that a cut only has n degrees of freedom (each vertex can be included or not included in S), but we are representing it with a vector with $\binom{n}{2}$ entries, thereby creating unwanted dependencies between entries of these vectors.

We would thus like to find a more natural parametrization of these cuts that can still be connected naturally to the vectors v_S. To construct such a parametrization, we assign to each cut S a ± 1 indicator vector $u_S \in \mathbb{R}^n$. Entries are indexed by vertices in $[n]$, and for each $i \in [n]$ we have $(u_S)_i = 1$ if $i \in S$ and -1 otherwise.

Now we connect these new indicator vectors to v_S as follows. Define the matrix $M_S \in \mathbb{R}^{n \times n}$ by $M_S = \frac{\mathbf{1}\mathbf{1}^T - u_S u_S^T}{2}$. (Note that from here onwards, $\mathbf{1}$ now refers to the vector in $\mathbb{R}^n$ with all entries equal to 1.) M_S's entries are indexed by ordered pairs of vertices. Now observe that:

$$
\begin{aligned}
(M_S)_{i,j} &= \frac{\mathbf{1}_i\mathbf{1}_j - (u_S)_i(u_S)_j}{2} \\
&= \frac{1 - (u_S)_i(u_S)_j}{2} \\
&= \begin{cases} 0, & i,j \in S \text{ or } i,j \notin S, \\ 1, & \text{otherwise} \end{cases} \\
&= \begin{cases} (v_S)_{i,j}, & i \neq j, \\ 0, & i = j. \end{cases}
\end{aligned}
$$

Thus if we flatten M_S into a vector, it consists of two copies of v_S (each unordered vertex pair (i,j) in v_S appears twice in M_S since the vertex pairs indexing M_S are ordered) and n 0's (corresponding to vertex pairs (i,i) for $i \in [n]$). This allows us to rewrite LP1 in terms of the u_S's as stated in the following lemma. For matrices A, B of the same shape, we use $\langle A, B\rangle$ to denote the matrix inner product $\mathrm{tr}(A^TB) = \sum_{i,j} A_{i,j}B_{i,j}$.

▶ **Lemma 10.** *For any C, LP1 has value $\geq \epsilon^2n^2/4$ if and only if the following LP has value at least $\epsilon^2n^2/2$. We call this the "matrix LP". Here, $Z \in \mathbb{R}^{n\times n}$.*

$$
\begin{aligned}
&\textit{Maximize } \langle Z, M_C\rangle \\
&\textit{subject to } \langle Z, M_{Q_j}\rangle = 0 \quad \textit{for all } j \in [q], \textit{ and} \\
&\qquad\qquad Z \geq -\mathbf{1}.
\end{aligned}
$$

Proof. See Appendix F.3 of the full version. ◀

3.3 Step 3: Defining LP2 and Connecting LP2 to LP1

For a near-max cut C, define a new linear program which we call LP2 as follows. Here $y \in \mathbb{R}^n$.

$$
\begin{aligned}
&\text{Maximize } y^Tu_C \\
&\text{subject to } y^Tu_{Q_j} = 0 \quad \text{for all } j \in [q], \\
&\qquad\qquad y^T\mathbf{1} = 0, \text{ and} \\
&\qquad\qquad -\mathbf{1} \leq y \leq \mathbf{1}.
\end{aligned}
$$

We claim that it suffices to show that LP2 has value at least ϵn:

▶ **Lemma 11.** *If there exists a near-max cut C such that LP2 has value at least ϵn then the matrix LP for C has value at least $\epsilon^2n^2/2$, which would imply Theorem 6.*

Proof Sketch. Take such a near-max cut C and $y \in \mathbb{R}^n$ such that y is in the feasible region of LP2 and $y^Tu_C \geq \epsilon n$. Then we claim that $Z = -yy^T$ is feasible for the matrix LP and attains a value $\geq \epsilon^2n^2/2$. Intuitively, y can be thought of as a vector representing a "pseudo-cut" in the same way that u_S represents S. LP2 having high value means that y aligns non-trivially with C, and what we are claiming is that perturbing towards the "pseudo-cut" corresponding to y will align the graph's weights with the cut corresponding to C. We provide details in Appendix F.4 of the full version. ◀

3.4 Step 4: Showing that LP2 has High Value

Finally, we show that we can find a near-max cut C such that LP2 has value at least ϵn, which by Lemma 11 will complete the proof of Theorem 6. We do this by taking the dual of LP2, which has a simple characterization captured by the following lemma:

▶ **Lemma 12.** *Consider vectors $w, w_1, w_2, \ldots, w_k \in \mathbb{R}^d$, and the following LP:*

$$\begin{aligned} &\textit{Maximize } z^T w \\ &\textit{subject to } z^T w_i = 0 \quad \textit{for all } i \in [k], \textit{ and} \\ &\qquad -\mathbf{1} \leq z \leq \mathbf{1}. \end{aligned}$$

Let $W = \text{span}(w_1, w_2, \ldots, w_k)$. Then the value of this LP is $\min_{v \in W} ||v - w||_1$. (If there is no such v then by this minimum we mean ∞.)

Proof. See Appendix F.5 of the full version. ◀

To use this lemma, let $V = \text{span}(u_{Q_1}, \ldots, u_{Q_q}, \mathbf{1})$. Then Lemma 12 tells us that LP2 has value equal to $\min_{u \in V} ||u - u_C||_1$.

Now note that V depends only on the set of queries and not at all on C. Thus we would like to show that there exists a near-max cut C such that $\min_{u \in V} ||u - u_C||_1 \geq \epsilon n$. We will show the strict version of this inequality i.e. that $\min_{u \in V} ||u - u_C||_1 > \epsilon n$. Denote by B_r the ℓ_1 ball of radius r in $\mathbb{R}^n$. Then what we want to show is that there exists a near-max cut C such that $u_C \notin V + B_{\epsilon n}$. This brings us to our key technical lemma, which has little to do with graphs and may be of independent interest:

▶ **Lemma 13.** *Let $\epsilon \in (0,1)$ and $d \leq \alpha' n$ for $\alpha' < \frac{(1-\epsilon)^3}{108(1+\epsilon)}$. Suppose D is a d-dimensional subspace of $\mathbb{R}^n$. Denote by B_r the ℓ_1 ball of radius r in $\mathbb{R}^n$. Then there exists $p \in \{-1,1\}^n$ such that $p \notin D + B_{\epsilon n}$ and $|\mathbf{1}^T p| \leq 2\sqrt{n \log n}$.*

Proof. We show this by a volume argument. Specifically, we estimate the size of $(D + B_{\epsilon n}) \cap \{-1,1\}^n$ using an ℓ_1 ϵ-net argument and show that this must be much less than 2^n. We provide details in Appendix F.6 of the full version. ◀

With this lemma, we can complete this step and thus the proof of Theorem 6:

▶ **Corollary 14.** *For n sufficiently large, there exists a near-max cut C such that $u_C \notin V + B_{\epsilon n}$.*

Proof. See Appendix F.7 of the full version. ◀

4 Sparsifier-based Randomized Algorithms for $(1-\epsilon)$-approximation

Here we address the randomized upper bound part of Theorem 4, namely that a $(1-\epsilon)$-approximation can be achieved in the cut finding setting with $\tilde{O}(n)$ queries. Our algorithms are adaptive. The key notion is that of a *sparsifier*:

▶ **Definition 15.** *Given weighted graphs G, H on the same set of n vertices with non-negative weights, we say H is an ϵ-sparsifier of G if all of the following conditions hold:*
1. *H has $\tilde{O}_\epsilon(n)$ edges with nonzero weight.*
2. *For any cut $S \subseteq [n]$, we have $(1-\epsilon)F(S;G) \leq F(S;H) \leq (1+\epsilon)F(S;G)$.*

Here $F(S;G)$ denotes the value of the cut defined by S on the graph G, and similarly for $F(S;H)$.

▶ **Lemma 16.** *If an algorithm can compute an ϵ-sparsifier of G in $\tilde{O}(n/\epsilon^2)$ queries with high probability, then it can also find a $(1-2\epsilon)$-approximate max cut with no additional queries.*

Proof. Once the algorithm has a sparsifier H, it can just try all possible cuts and output the cut U maximizing $F(U;H)$. Indeed, for any other cut S, we would have $F(U;G) \geq \frac{F(U;H)}{1+\epsilon} \geq \frac{F(S;H)}{1+\epsilon} \geq \frac{(1-\epsilon)F(S;G)}{1+\epsilon} \geq (1-2\epsilon)F(S;G)$, so U is indeed a $(1-\epsilon)$-approximate max cut. ◀

Throughout this section, let $d > 5$ be constant, and let $\delta \in (1/2, 1)$ be a constant sufficiently close to 1. Then let c_0 and c_1 be sufficiently large positive constants. Also, let W_{tot} be the total weight of all edges in the graph, and W the ratio between the largest and smallest nonzero edge weights. Finally, for a vertex set $S \subseteq V(G)$, let $G[S]$ denote the vertex-induced subgraph of G on S. Note that we do not require that G be connected. We organize the remainder of this section as follows:

1. In Section 4.1, we set up and analyze the algorithmic tools necessary to adapt [31]'s algorithm to weighted graphs.
2. In Section 4.2, we present the direct adaptation of [31]'s algorithm to weighted graphs and show that it constructs a sparsifier in $\widetilde{O}(n \log W)$ queries.
3. In Section 4.3, we use ideas introduced by [7] relating edge strengths to maximum spanning trees in order to construct a sparsifier for weighted graphs in $\widetilde{O}(n)$ queries, thus eliminating the dependence of the query complexity on W.
4. Additionally, in Appendix D of the full version, we show that we can achieve a $(1-\epsilon)$-approximation for max-cut in $\widetilde{O}(n)$ queries without needing the optimizations in Section 4.3. This is achieved by essentially stopping our adaptation of [31]'s algorithm early. This will not construct a sparsifier, but it will construct something "close enough" to suffice for max-cut. Intuitively, we do this by discarding edges of weight $< W_{\text{tot}}/\text{poly}(n)$ since these will not have much effect on the max cut, thereby reducing the problem to one where $W = \text{poly}(n)$.
   Given this result, our sparsifier for weighted graphs in Section 4.3 is not necessary for max-cut specifically, but it makes for a conceptually cleaner algorithm for max-cut and may be applicable to other problems.

4.1 Setup and Algorithmic Tools

The key idea is the notion of edge strength introduced by [7].

▶ **Definition 17** (([7], as stated in [31])**.** *The* strong connectivity *of G, denoted $K(G)$, is the value of G's min cut.*

▶ **Definition 18** ([7], as stated in [31])**.** *Given an edge e in G, the strong connectivity or* edge strength *k_e of e is the maximum min cut over all vertex-induced subgraphs of G containing e:*

$$k_e = \max_{S \subseteq V : u,v \in S} K(G[S]).$$

The idea introduced by [7] and used by [31] is that subsampling edges of G with probabilities inversely proportional to their strength will give a sparsifier. We cannot do exactly this in the cut query model, but we can subsample in a way that is "close enough" to independent. We need some basic primitives to support our algorithm, and we capture all of them in the following lemma:

▶ **Lemma 19.** *There exists a data structure supporting the following operations.*

1. *InitializeDS(H): Initialize the structure's state and carry out any preprocessing needed with the starting graph H.*
2. *Contract(S): Contract a given supernode set S.*
3. *GetEdge(): Find and return an edge from H (that has not been contracted) with weight at least $\frac{2W(H)}{n^2}$. Here $W(H)$ is the total weight of not-yet-contracted edges.*
4. *GetTotalWeight(S): Return the total weight of all edges with both endpoints in S (that have not yet been contracted). S must once again be a set of supernodes.*
5. *Sample(S): Sample a random edge with both endpoints in S (that has not yet been contracted), with probability proportional to its weight. S must be a set of supernodes here as well.*

It takes $O(n)$ queries for each call to InitializeDS, $O(1)$ queries for each call to Contract, $O(\log n)$ queries for each call to GetEdge, $O(1)$ queries for each call to GetTotalWeight, and $O(\log n)$ queries for each call to Sample.

Proof. The pre-processing and queries when contracting are due to having to keep track of (super)node degrees. GetEdge and Sample can be handled using recursive bisection procedures similar to that used to prove Corollary 2.2 in [31]. We provide details in Appendix G.1 of the full version. ◀

Before we go any further, we make an important comment about how we regard contraction in our algorithms (including Lemma 19). When we contract a set of vertices, we regard that set of vertices as one supernode as usual, but we do not merge any edges that have now become parallel. So the set of edges will always be a subset of the edges from the original graph.

It will be convenient to regard our algorithms as having two separate stages, although the two stages share some ingredients. In the first stage, the algorithm iteratively subsamples and contracts the graph to estimate the strengths of all edges within a constant factor. In the second stage, the algorithm uses these edge strength estimates to construct the sparsifier, following the ideas of [7]. We address these two stages in the next two subsections respectively.

We note that these algorithms are very similar to those presented by [31]; the key difference is that whenever the sparsification algorithm in [31] subsamples the graph, it does so independently for each edge. This works because [31] focuses on unweighted graphs. With weighted graphs, we would like to sample edges proportionately to their weight, and this cannot be done independently without knowledge of the graph. We modify their subsampling procedures to obtain algorithms that do not sample completely independently, but still have the concentration properties that we need.

4.1.1 Constant-Factor Edge Strength Estimates

We next describe the main tool of our edge strength estimation, which we call EstimateAndContract. It takes in the input graph G where some vertex sets S_i have already been contracted and a strength parameter κ, and further contracts G while also estimating the strength of any edges that get contracted. For any graph H, let $W(H)$ denote the total edge weight of H.

We state some key properties of Algorithm 4.1 and defer their proofs to Appendix G.4 of the full version:

▶ **Lemma 20** (Analogous to claim 3.6 from [31]). *With probability $1 - O(n^{1-d})$, for all e such that $k_e \geq \kappa$ and e is not contracted by any of the sets in $\mathcal{C}$, it will be assigned $k'_e = \kappa/2$ and contracted.*

Algorithm 4.1 EstimateAndContract$(G, \mathcal{C}, \kappa, X)$.

Data: Initial graph G with vertex set $V(G) \subseteq [n]$, disjoint collection $\mathcal{C}$ of contracted sets, strength parameter κ, partial list X of strength estimates
Result: Updated collection of contracted sets $\mathcal{C}$, updated list X of strength estimates
Let G' be G with all sets from $\mathcal{C}$ contracted and $\lambda = \frac{c_0 \log^2 n}{\kappa} W(G')$. (We can find $W(G')$ using GetTotalWeight from Lemma 19.)

1. Sample λ edges from G', proportionally to their weights, with replacement. If an edge e is sampled at least once, assign weight $\frac{w(e)}{p(e)}$ to it, where $p(e) = 1 - (1 - \frac{w_e}{W(G')})^\lambda$. These newly weighted edges form a new graph G''.
2. While there exists a connected component of G'' with a cut of size $\leq (1-\delta)\kappa$, delete all edges of that cut from G''. Let the resulting connected components of G'' be $C_1, C_2, \ldots, C_r$.
3. For each $i \in [r]$, append the tuple $(C_i, \kappa/2)$ to X. (Here, what we are saying is "assign a strength estimate of $\kappa/2$ to any edge with both endpoints in C_i that has not already been contracted", but we phrase it differently to account for the fact that the algorithm may not actually know these edges.)
4. Add C_i to $\mathcal{C}$ (and remove any subsets of C_i to maintain disjointness) for all $i \in [r]$.

▶ **Lemma 21** (Analogous to claim 3.7 from [31]). *Assume that any edge contracted by $\mathcal{C}$ has strength $\geq \kappa/2$. Then with probability $1 - O(n^{1-d})$, no edge e such that $k_e < \kappa/2$ is assigned a strength estimate or contracted.*

4.1.2 Sparsifier Construction

Next, we describe our algorithm that will construct a sparsifier if provided constant-factor strength estimates for all edges in the graph, which we call ConstructSparsifier.

We capture the desired sparsification properties in the following lemma:

▶ **Lemma 22.** *Fix $X = [(C_1, \beta_1), (C_2, \beta_2), \ldots, (C_r, \beta_r)]$ as in ConstructSparsifier (X might be random, but we condition on a particular list of values for now). Then for each i, define:*

$$E(C_i) = \{e \in G : \text{ both endpoints of } e \text{ are in } C_i\}$$
$$\widetilde{E}(C_i) = E(C_i) \setminus \bigcup_{j : C_j \subset C_i} E(C_j)$$

(Thus $\widetilde{E}(C_i)$ is the set of edges that will be contracted at the time that C_i is contracted.)

Assume each of the following conditions:

1. *Each connected component of G is contained in at least one C_i (every edge in G gets contracted), and*
2. *For all i and edges $e \in \widetilde{E}(C_i)$, we have $\beta_i \in [k_e/4, k_e]$ (edge strength estimates are correct within a constant factor).*

Then with probability $1 - O(n^{-d})$, ConstructSparsifier will output a sparsifier H that approximates the cuts of G within a factor of $1 \pm 2\epsilon$. (Thus this probability is only considering the randomness of ConstructSparsifier.)

Proof. This follows using similar ideas to [7], but we need to take some extra care because the subsampling we use to construct the sparsifier is not independent. We provide details in Appendix G.2.2 of the full version. ◀

Algorithm 4.2 ConstructSparsifier(G, X).

Data: Graph G with vertex set $[n]$, list $X = [(C_1, \beta_1), (C_2, \beta_2), \ldots, (C_r, \beta_r)]$ of strength estimates. We assume that $\{C_1, \ldots, C_r\}$ is a laminar family of vertex sets and that if $C_i \subset C_j$ then $i < j$. (This is because we will be contracting $C_1, \ldots, C_r$ in that order.) Note that we do **not** assume here that X includes a strength estimate for every edge.
Result: Potential sparsifier H
Initialize $G' = G$ and H to be empty.
InitializeDS(G'). (This is just to reset and ignore any previous contractions we may have done in EstimateAndContract.)
for $i \leftarrow 1$ **to** r **do**
1. Let $\mu_i = \frac{c_1 \log^2 n}{\epsilon^2 \beta_i} W(C_i)$. (Here $W()$ is with respect to G', and once again can be found using GetTotalWeight from Lemma 19.)
2. Sample μ_i edges from C_i proportionally to their weights, with replacement. If an edge e is sampled at least once, add it to H with weight $\frac{w_e}{p_e}$, where $p_e = 1 - (1 - \frac{w_e}{W(C_i)})^{\mu_i}$ is the probability that e is sampled at least once. (Note that this sampling would be done by calling Sample(C_i) from Lemma 19.)
3. Contract C_i in G'.

end

4.2 Sparsification with $\widetilde{O}(n \log W)$ Queries

Here we analyze the naive generalization of [31]'s sparsifier to weighted graphs. The procedure is described in Algorithm 4.3.

Algorithm 4.3 NaiveWeightedSubsample(G, T).

Data: Graph G on n vertices, positive real parameter T
Result: A potential (2ϵ)-sparsifier H of G.
Initialize $\mathcal{C} = \emptyset$ and X as an empty list.
InitializeDS(G).
Find W_{tot} by running GetTotalWeight(G).
Initialize κ to be the smallest power of 2 that is at least W_{tot}.
while *GetTotalWeight*$(G) > 0$ *and* $\kappa > W_{tot}/T$ **do**
  EstimateAndContract$(G, \mathcal{C}, \kappa, X)$
  $\kappa \leftarrow \kappa/2$
end
$H \leftarrow$ ConstructSparsifier(G, X).

▶ **Theorem 23.** *NaiveWeightedSubsample*(G, ∞) *runs in* $O(n \log^3 n(\log n + \log W + \frac{1}{\epsilon^2}))$ *queries and outputs a* (2ϵ)*-sparsifier* H *with probability* $1 - O(n^{1-d}(\log n + \log W))$.

Proof. We outline the proof here and provide details in Appendices G.5 (correctness) and G.8 (efficiency) of the full version. The proof proceeds in two parts.

First, we address the calls to EstimateAndContract. Given Lemmas 4.6 and 4.7, EstimateAndContract can be thought of as estimating the strengths of and then contracting edges whose strengths are within a window that is a factor of 4 wide. The lemmas tell us that these strength estimates are accurate within a factor of 4. Then NaiveWeightedSubsample

essentially "slides" this window across all possible edge strengths so that all edges of G are assigned a strength estimate. This part has query complexity $O(n \log^3 n(\log n + \log W))$; the $\log W$ term is because the outer loop could run for $O(\log n + \log W)$ iterations. This is also why the success probability depends on W, as this is obtained by taking a union bound over all iterations.

Secondly, because all edges are assigned an accurate strength estimate (within a constant factor), Lemma 22 tells us that ConstructSparsifier will output a (2ϵ)-sparsifier with high probability. This part has query complexity $O(n \log^3 n/\epsilon^2)$. ◀

4.3 Sparsification with $\widetilde{O}(n)$ Queries

We now show how to eliminate the dependence of the query complexity and success probability on W, thus constructing sparsifiers in $\widetilde{O}(n)$ queries. We begin by setting up the necessary ideas.

4.3.1 Crude Edge Strength Estimates

Recall that the key problem with Algorithm 4.3 was that our "sliding window" for edge strength estimation could potentially repeat $O(\log n + \log W)$ times. The key idea is to mitigate this by finding very crude (within a factor of n^4) estimates for the edge strengths for all edges in G, before refining these estimates using EstimateAndContract. We do this using the idea of [7] to estimate edge strengths from the maximum spanning forest (MSF) of G. Fix an MSF $\mathcal{T}$ of G. Then for any edge e with endpoints i, j, define $d_e = d_{i,j}$ to be the minimum weight of an edge on the MSF path between the endpoints of e. We first state a lemma shown in [7]:

▶ **Lemma 24** ([7]). *For all edges e, we have $d_e \leq k_e \leq n^2 d_e$.*

This would immediately give us sufficient edge strength estimates but we do not know of a way to find the MSF efficiently in the cut query model. In fact, we can adapt the max cut dimension argument from Lemma A.3 in the full version to a "max tree dimension" argument to show that deterministically finding the MSF requires $\Omega(n^2)$ queries; see Appendix A.1 of the full version for details. So instead, we run what we call "approximate Kruskal" using the primitives we have available from Lemma 19.

Algorithm 4.4 ApproximateKruskal(G).

Data: Graph G on n vertices
Result: A forest $\widetilde{\mathcal{T}}$ using the edges of G.
Initialize $\widetilde{\mathcal{T}}$ to be the empty graph on n vertices.
InitializeDS(G).
while *GetTotalWeight*(G) > 0 **do**
  e = GetEdge()
  Contract(e)
  Add e to $\widetilde{\mathcal{T}}$.
end

It follows from Lemma 19 that ApproximateKruskal can be run in $O(n \log n)$ queries. Once we run ApproximateKruskal, we will have a spanning forest of G so we will also know its connected components. Moreover, the following lemma tells us that even this crude approximation to the MSF suffices to give us edge strength estimates. We defer the proofs of this lemma and its straightforward corollary to Appendix G.6.1 of the full version.

▶ **Lemma 25.** *For any distinct i, j, define $\tilde{d}_{i,j}$ as follows:*

- *If i, j are connected in G, then let $\tilde{d}_{i,j}$ be the minimum weight of an edge on the path in $\widetilde{\mathcal{T}}$ connecting i and j.*
- *Otherwise, let $\tilde{d}_{i,j} = 0$.*

Then for any i, j that are connected in G, we have $\frac{2}{n^2} d_{i,j} \leq \tilde{d}_{i,j} \leq d_{i,j}$ for all i, j.

Note that we only assume here that $\mathcal{T}$ is maximal; any assumptions we make about $\widetilde{\mathcal{T}}$ are baked into GetEdge.

▶ **Corollary 26.** *For all distinct i, j, we have $k_{i,j} \in [\tilde{d}_{i,j}, \frac{n^4}{2}\tilde{d}_{i,j}]$.*

4.3.2 Fast Weighted Subsampling

We now describe how to use our crude edge strength estimates to construct a sparsifier in $\tilde{O}(n)$ queries. The idea is that by Lemma 21, any edges with strength $< \kappa/2$ will be deleted in Step 2 of Algorithm 4.1. But we can use our crude edge strength estimates to preemptively identify some edges that will definitely be deleted, and then delete these edges to disconnect the graph a bit *before* running EstimateAndContract. We describe this procedure in Algorithm 4.5.

Algorithm 4.5 FastWeightedSubsample(G).

Data: Graph G on n vertices
Result: A potential (2ϵ)-sparsifier H of G.
Run ApproximateKruskal on a copy of G and calculate $\tilde{d}_{i,j}$ for all i, j.
Initialize L to a list of all nonzero values of $\tilde{d}_{i,j}$.
Initialize $\mathcal{C} = \emptyset$, $\kappa = \infty$, and X as an empty list.
InitializeDS(G).
while $L \neq \emptyset$ **do**
  Let $\tilde{D} = \max L$ and $C = \{(i, j) : \tilde{d}_{i,j} < \tilde{D}/(2n^5)\}$.
  Let $S_1, \ldots, S_r$ be the connected components of K_n after we remove all edges from C.
  Let κ' be the smallest power of 2 that is at least $n^4\tilde{D}/2$.
  $\kappa \leftarrow \min(\kappa, \kappa')$
  while $\kappa \geq \tilde{D}/(2n)$ **do**
    for $i \leftarrow 1$ **to** r **do**
      EstimateAndContract($G[S_i], \mathcal{C}, \kappa, X$)
      Remove all contracted edges from L.
    end
    $\kappa \leftarrow \kappa/2$
  end
end
$H \leftarrow$ ConstructSparsifier(G, X).

We make one comment here about FastWeightedSubsample: for the algorithm as presented to even be well-defined, we need to check that each S_i at any stage of the algorithm is the union of some collection of supernodes. If not, it does not make sense to run EstimateAndContract on each $G[S_i]$. This condition is also necessary to ensure the applicability of Lemma 19 to each call to EstimateAndContract; Contract, GetTotalWeight, and Sample all require that their input S be a union of supernodes. We defer the verification of this and the proof of our final theorem to Appendices G.7 (correctness) and G.8 (efficiency) of the full version:

▶ **Theorem 27.** *FastWeightedSubsample runs in* $O(n \log^3 n(\log n + \frac{1}{\epsilon^2}))$ *queries and outputs a* (2ϵ)*-sparsifier* H *with probability* $1 - O(n^{5-d})$.

▶ **Corollary 28.** *For any* $\epsilon > 0$*, the query complexity for a randomized algorithm to construct an* ϵ*-sparsifier with* $1 - o(1)$ *probability for a weighted undirected graph is* $\widetilde{O}(n)$*. This algorithm is adaptive.*

By Lemma 16, this yields our desired algorithmic result for max-cut:

▶ **Corollary 29.** *For any* $c < 1$*, the query complexity for a randomized algorithm to achieve a c-approximation with* $1 - o(1)$ *probability for global max-cut on a weighted undirected graph in the cut finding setting is* $\tilde{O}(n)$*. This algorithm is adaptive.*

We remind the reader that Theorem 27 extends the query-efficient sparsifier of [31] to weighted graphs (as we saw in Section 4.2, a naive generalization of [31] requires $\widetilde{O}(n \log W)$ queries).

References

1 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 459–467. SIAM, 2012. doi:10.1137/1.9781611973099.40.

2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In Michael Benedikt, Markus Krötzsch, and Maurizio Lenzerini, editors, *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 5–14. ACM, 2012. doi:10.1145/2213556.2213560.

3 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Spectral sparsification in dynamic graph streams. In Prasad Raghavendra, Sofya Raskhodnikova, Klaus Jansen, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013. Proceedings*, volume 8096 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2013. doi:10.1007/978-3-642-40328-6_1.

4 Simon Apers, Yuval Efron, Pawel Gawrychowski, Troy Lee, Sagnik Mukhopadhyay, and Danupon Nanongkai. Cut query algorithms with star contraction. *CoRR*, abs/2201.05674, 2022. arXiv:2201.05674.

5 Sepehr Assadi, Deeparnab Chakrabarty, and Sanjeev Khanna. Graph connectivity and single element recovery via linear and OR queries. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 7:1–7:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ESA.2021.7.

6 Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM Rev.*, 56(2):315–334, 2014. doi:10.1137/130949117.

7 András A. Benczúr and David R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM J. Comput.*, 44(2):290–319, 2015. doi:10.1137/070705970.

8 Joakim Blikstad, Jan van den Brand, Yuval Efron, Sagnik Mukhopadhyay, and Danupon Nanongkai. Nearly optimal communication and query complexity of bipartite matching. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 1174–1185. IEEE, 2022. doi:10.1109/FOCS54457.2022.00113.

9 Nader H. Bshouty and Hanna Mazzawi. Toward a deterministic polynomial time algorithm with optimal additive query complexity. *Theor. Comput. Sci.*, 417:23–35, 2012. doi:10.1016/J.TCS.2011.09.005.

10 Nader H. Bshouty and Hanna Mazzawi. On parity check (0, 1)-matrix over F_p. *SIAM J. Discret. Math.*, 29(1):631–657, 2015. doi:10.1137/120881129.

11 Niv Buchbinder and Moran Feldman. Deterministic algorithms for submodular maximization problems. *ACM Trans. Algorithms*, 14(3):32:1–32:20, 2018. doi:10.1145/3184990.

12 Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 649–658. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.73.

13 Deeparnab Chakrabarty, Andrei Graur, Haotian Jiang, and Aaron Sidford. Improved lower bounds for submodular function minimization. *CoRR*, abs/2207.04342, 2022. doi:10.48550/arXiv.2207.04342.

14 Deeparnab Chakrabarty and Hang Liao. A query algorithm for learning a spanning forest in weighted undirected graphs. In Shipra Agrawal and Francesco Orabona, editors, *International Conference on Algorithmic Learning Theory, February 20-23, 2023, Singapore*, volume 201 of *Proceedings of Machine Learning Research*, pages 259–274. PMLR, 2023. URL: https://proceedings.mlr.press/v201/chakrabarty23a.html.

15 Sung-Soon Choi. Polynomial time optimal query algorithms for finding graphs with arbitrary real weights. In Shai Shalev-Shwartz and Ingo Steinwart, editors, *Proceedings of the 26th Annual Conference on Learning Theory*, volume 30 of *Proceedings of Machine Learning Research*, pages 797–818, Princeton, NJ, USA, 12–14 June 2013. PMLR. URL: https://proceedings.mlr.press/v30/Choi13.html.

16 Sung-Soon Choi and Jeong Han Kim. Optimal query complexity bounds for finding graphs. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 749–758. ACM, 2008. doi:10.1145/1374376.1374484.

17 Marcel Kenji de Carli Silva, Nicholas J. A. Harvey, and Cristiane M. Sato. Sparse sums of positive semidefinite matrices. *ACM Trans. Algorithms*, 12(1):9:1–9:17, 2016. doi:10.1145/2746241.

18 Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. *SIAM J. Comput.*, 40(4):1133–1153, 2011. doi:10.1137/090779346.

19 Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995. doi:10.1145/227683.227684.

20 Andrei Graur, Tristan Pollner, Vikram Ramaswamy, and S. Matthew Weinberg. New query lower bounds for submodular function minimization. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 64:1–64:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.64.

21 Vladimir Grebinski and Gregory Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica*, 28(1):104–124, 2000. doi:10.1007/s004530010033.

22 Nicholas J. A. Harvey. *Matchings, matroids and submodular functions*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2008. URL: http://hdl.handle.net/1721.1/44416.

23 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001. doi:10.1145/502090.502098.

24 Haotian Jiang. Minimizing convex functions with integral minimizers. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 976–985. SIAM, 2021. doi:10.1137/1.9781611976465.61.

25 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 561–570. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.66.

26 Subhash Khot. On the power of unique 2-prover 1-round games. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 767–775. ACM, 2002. doi:10.1145/509907.510017.

27 Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM J. Comput.*, 37(1):319–357, 2007. doi:10.1137/S0097539705447372.

28 Troy Lee, Tongyang Li, Miklos Santha, and Shengyu Zhang. On the cut dimension of a graph. In Valentine Kabanets, editor, *36th Computational Complexity Conference, CCC 2021, July 20-23, 2021, Toronto, Ontario, Canada (Virtual Conference)*, volume 200 of *LIPIcs*, pages 15:1–15:35. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CCC.2021.15.

29 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1049–1065. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.68.

30 Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: sequential, cut-query, and streaming algorithms. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 496–509. ACM, 2020. doi:10.1145/3357713.3384334.

31 Aviad Rubinstein, Tselil Schramm, and S. Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pages 39:1–39:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ITCS.2018.39.

32 Xiaoming Sun, David P. Woodruff, Guang Yang, and Jialin Zhang. Querying a matrix through matrix-vector products. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 94:1–94:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.94.

One-Way Communication Complexity of Partial XOR Functions

Vladimir V. Podolskii
Tufts University, Medford, MA, USA

Dmitrii Sluch
Nebius, Tel Aviv, Israel

Abstract

Boolean function $F(x, y)$ for $x, y \in \{0,1\}^n$ is an XOR function if $F(x, y) = f(x \oplus y)$ for some function f on n input bits, where $\oplus$ is a bit-wise XOR. XOR functions are relevant in communication complexity, partially for allowing the Fourier analytic technique. For total XOR functions, it is known that deterministic communication complexity of F is closely related to parity decision tree complexity of f. Montanaro and Osbourne (2009) observed that one-way communication complexity $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F)$ of F is exactly equal to non-adaptive parity decision tree complexity $\mathrm{NADT}^{\oplus}(f)$ of f. Hatami et al. (2018) showed that unrestricted communication complexity of F is polynomially related to parity decision tree complexity of f.

We initiate the study of a similar connection for partial functions. We show that in the case of one-way communication complexity whether these measures are equal, depends on the number of undefined inputs of f. More precisely, if $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F) = t$ and f is undefined on at most $O\left(\frac{2^{n-t}}{\sqrt{n-t}}\right)$ inputs, then $\mathrm{NADT}^{\oplus}(f) = t$. We also provide stronger bounds in extreme cases of small and large complexity.

We show that the restriction on the number of undefined inputs in these results is unavoidable. That is, for a wide range of values of $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F)$ and $\mathrm{NADT}^{\oplus}(f)$ (from constant to $n-2$) we provide partial functions (with more than $\Omega\left(\frac{2^{n-t}}{\sqrt{n-t}}\right)$ undefined inputs, where $t = \mathrm{D}^{\rightarrow}_{\mathrm{cc}}$) for which $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F) < \mathrm{NADT}^{\oplus}(f)$. In particular, we provide a function with an exponential gap between the two measures. Our separation results translate to the case of two-way communication complexity as well, in particular showing that the result of Hatami et al. (2018) cannot be generalized to partial functions.

Previous results for total functions heavily rely on the Boolean Fourier analysis and thus, the technique does not translate to partial functions. For the proofs of our results we build a linear algebraic framework instead. Separation results are proved through the reduction to covering codes.

2012 ACM Subject Classification Theory of computation → Communication complexity; Theory of computation → Oracles and decision trees; Theory of computation → Error-correcting codes

Keywords and phrases Partial functions, XOR functions, communication complexity, decision trees, covering codes

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.116

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2023/157/`

Acknowledgements We would like to thank the anonymous reviewers for useful comments that helped us improve the presentation.

1 Introduction

In communication complexity model two players, Alice and Bob, are computing some fixed function $F\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ on a given input (x, y). However, Alice knows only x and Bob knows only y. The main object of studies in communication complexity is the amount of communication $\mathrm{D}_{\mathrm{cc}}(F)$ needed between Alice and Bob to compute the function.

Function F is a XOR-function if for all $x, y \in \{0,1\}^n$ we have $F(x,y) = f(x \oplus y)$ for some $f\colon \{0,1\}^n \to \{0,1\}$, where $x \oplus y$ is a bit-wise XOR of Boolean vectors x and y. XOR-functions are important in communication complexity [27, 19, 25, 26, 3, 13, 15, 1, 23, 21, 5, 2, 8, 11, 9], on one hand, since there are important XOR-functions defined based on Hamming distance between x and y, and on the other hand, since the structure of XOR-functions allows for the Fourier analytic techniques. In particular, this connection suggests an approach for resolving Log-rank Conjecture for XOR-functions [27, 13].

In recent years there was considerable progress in the characterization of communication complexity of a XOR-function F in terms of the complexity of f in parity decision tree model. In this model the goal is to compute a fixed function f on an unknown input $x \in \{0,1\}^n$ and in one step we are allowed to query XOR of any subset of input bits. We want to minimize the number of queries that is enough to compute f on any input x. The complexity of f in this model is denoted by $\mathrm{DT}^{\oplus}(f)$. It was shown by Hatami et al. [13] that for any total f we have $\mathrm{D}_{\mathrm{cc}}(F) = \mathrm{poly}(\mathrm{DT}^{\oplus}(f))$.

Even stronger connection holds for one-way communication complexity case. In this setting only very restricted form of communication is allowed: Alice sends Bob a message based on x and Bob has to compute the output based on this message and y. We denote the complexity of F in this model by $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F)$. The relevant model of decision trees is the model of non-adaptive parity decision trees. In this model we still want to compute some function f on an unknown input and we still can query XORs of any subsets of input bits, but now all queries should be provided at once (in other words, each query cannot depend on the answers to the previous queries). The complexity of f in this model is denoted by $\mathrm{NADT}^{\oplus}(f)$. It follows from the results of Montanaro, Osbourne [19] and Gopalan et al. [10] that for any total XOR-function $F(x,y) = f(x \oplus y)$ we have $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F) = \mathrm{NADT}^{\oplus}(f)$.

These results on the connection between communication complexity and parity decision trees can be viewed as lifting results. This type of results have seen substantial progress in recent years (see [20]). The usual structure of a lifting result is that we start with a function f that is hard in some weak computational model (for example, a decision tree type model), compose it with some gadget function g to obtain $f \circ g$ (each variable of f is substituted by a copy of g defined on fresh variables) and show that $f \circ g$ is hard in a stronger computational model (for example, a communication complexity type model). The results on XOR-functions can be viewed as lifting results for $g = \mathrm{XOR}$.

The results on the connection between communication complexity of XOR-functions and parity decision trees discussed above are proved only for total functions f for the reason that the proofs heavily rely on the Fourier techniques. However, in communication complexity and decision tree complexity it is often relevant to consider a more general case of partial functions, and many lifting theorems apply to this type of functions as well, see e.g. [7, 17, 4, 22]. In particular, there are some lifting results for partial functions for gadgets that are stronger than XOR: Mande et al. [18] proved such a result for one-way case for inner product gadget (inner product is XOR applied to ANDs of pairs of variables) and Loff, Mukhopadhyay [17] proved a result on lifting with equality gadget for general case (note that equality for inputs of length 1 is practically XOR function). In [17] a conjecture is mentioned that for partial XOR-functions $\mathrm{D}_{\mathrm{cc}}(F)$ is approximately equal to $\mathrm{DT}^{\oplus}(f)$ as well.

Our results

In this paper we initiate the studies of the connection between communication complexity for the case of partial XOR functions and parity decision trees. It turns out that for one-way case whether they are equal depends on the number of inputs on which the function is undefined: if the number of undefined inputs is small, then the complexity measures are equal and if it is too large, they are not equal.

More specifically, we show that for $t = \mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F)$ the equality $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F) = \mathrm{NADT}^{\oplus}(f)$ holds if f is undefined on at most $O\left(\frac{2^{n-t}}{\sqrt{n-t}}\right)$ inputs.

On the other hand, we provide a family of partial functions for which $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F) < \mathrm{NADT}^{\oplus}(f)$[1]. More specifically, we show that for any constant $0 < c < 1$ there is a function f with $\mathrm{NADT}^{\oplus}(f) = cn$ and $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F) \leq c'n$ for some $c' < c$.

The number of undefined inputs for the function is $O\left(\frac{2^{dn}}{\sqrt{n}}\right)$ if $c > 1/2$, is equal to 2^{n-1} if $c = 1/2$, and is $2^n - O\left(\frac{2^{dn}}{\sqrt{n}}\right)$ if $c < 1/2$, where $0 < d < 1$ is some constant (depending of c).

We provide a function f for which $\mathrm{NADT}^{\oplus}(f) = \sqrt{n \log n}$ and $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F) \leq O(\log n)$, the number of undefined inputs for f is $2^n - 2^{\Theta(\sqrt{n}\log^{3/2} n)}$. Thus, we provide an exponential gap between the two measures.

We provide stronger bounds for small and large values of complexity. For $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F) = 1$ we show that the equality $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F) = \mathrm{NADT}^{\oplus}(f)$ is true for all partial f. For $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F) = 2$ the equality is true for at most $2^{n-3} - 1$ undefined inputs. The smallest values of measures for which we provide a separation are $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F) = 7$ and $\mathrm{NADT}^{\oplus}(f) = 8$. On the other end of the spectrum we show that for any partial function if $\mathrm{NADT}^{\oplus}(f) \geq n - 1$, then $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F) = \mathrm{NADT}^{\oplus}(f)$. The largest value of $\mathrm{NADT}^{\oplus}$ for which we provide a separation is $n - 2$, this complements the result that starting with $\mathrm{NADT}^{\oplus}(f) = n - 1$ the measures are equal.

All our separation results translate to the setting of two-way communication complexity vs. parity decision trees. In particular, we provide a partial function f with exponential gap between $\mathrm{D}_{\mathrm{cc}}(F)$ and $\mathrm{DT}^{\oplus}(f)$, which refutes the conjecture mentioned in [17]. It is an interesting open problem whether the polynomial relation between these measures discovered by Hatanami et al. for total functions holds for partial functions with some restriction on the number of undefined points.

The techniques behind the results on the connections between communication complexity of XOR-functions and parity decision tree complexity for total functions heavily rely on the Fourier analysis. However, it is not clear how to translate this technique to partial functions. To prove our results, we instead translate the Fourier-based approach of [19, 10] into the language of linear algebra. We design a framework to capture the notion of one-way communication complexity of partial XOR-functions and use this framework to establish equality of $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F)$ and $\mathrm{NADT}^{\oplus}(f)$ for the small number of undefined points. The separation results can be proved using our framework, but in these version of the paper we provide self-contained proof. The separation results are proved by a reduction to the covering codes.

The rest of the paper is organized as follows. In Section 2 we provide necessary preliminary information and introduce the notations. In Section 3 we introduce our linear-algebraic framework. In Section 4 we prove main results on the equality of complexity measures. In Section 5 we prove separation results. In Section 6 we provide results for extreme cases. Some of the technical proofs are edited out of this version and can be found in the full paper `https://eccc.weizmann.ac.il/report/2023/157/`.

[1] Note that the gap in the other direction is impossible: it is easy to see that $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F) \leq \mathrm{NADT}^{\oplus}(f)$ for all f (see Lemma 4 below). Similar inequality (with an extra factor of 2) holds for general communication complexity and parity decision tree complexity.

2 Preliminaries

2.1 Boolean cube

A Boolean cube is a graph on the set $\{0,1\}^n$ of Boolean strings of length n. We connect two vertices with an edge if they differ in a single bit only. The set $\{0,1\}^n$ can also be thought of as the vector space $\mathbb{F}_2^n$, with the bitwise XOR as the group operation. An inner product over this space can be defined as

$$\langle x, y\rangle = \bigoplus_i x_i \wedge y_i.$$

Hamming weight of x denoted $|x|$ is defined as the number of coordinates of x equal to 1. Hamming distance $\mathrm{dist}(x,y)$ between $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$ is the number of coordinates at which x and y differ. The Hamming ball of radius r is a set of vertices of Boolean cube $\{0,1\}^n$ with Hamming weight not exceeding r. We denote by $V(n,r)$ the volume of a Hamming ball in $\{0,1\}^n$ of radius r.

2.2 Isoperimetric inequalities

▶ **Definition 1.** *For a set A we denote the set of neighbors of elements of A as ΓA. We denote $\Gamma' A := \Gamma A \setminus A$.*

We will need the vertex isoperimetric inequality for a Boolean cube known as Harper's theorem. To state it we first define Hales order.

▶ **Definition 2** (Hales order [12, Page 56])**.** *Consider two subsets $x, y \subseteq [m]$ for some natural m. We define $x \prec y$ if $|x| < |y|$ or $|x| = |y|$ and the smallest element of the symmetric difference of x and y belongs to x. In other words, there exists an i such that $i \in x, i \notin y$, and i is the smallest element in which x and y differ. Here is an example of Hales order for $m = 4$:*

$$\varnothing, \{1\}, \{2\}, \{3\}, \{4\}, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\},$$
$$\{1,2,3\}, \{1,2,4\}, \{1,3,4\}, \{2,3,4\}, \{1,2,3,4\}.$$

We can induce Hales order on the set $\{0,1\}^m$ by identifying subsets of $[m]$ with their characteristic vectors. We define I_a^m to be the set of the first a elements of $\{0,1\}^m$ in Hales order.

▶ **Theorem 3** (Harper's theorem [12, Theorem 4.2])**.** *Let $A \subseteq \{0,1\}^m$ be a subset of vertices of m-dimensional Boolean cube and denote $a = |A|$. Then $|\Gamma A| \geq |\Gamma I_a^m|$.*

2.3 Communication Complexity and Decision Trees

Throughout this paper, f denotes a partial function $\{0,1\}^n \to \{0,1,\perp\}$, we let $\mathrm{Dom}(f) = f^{-1}(\{0,1\})$. We define an XOR-function $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1,\perp\}$ as

$$F(x,y) = f(x \oplus y).$$

In communication complexity model two players, Alice and Bob, are computing some fixed function $F\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ on a given input (x,y). However, Alice knows only x and Bob knows only y. The main subject of studies in communication complexity is the amount of communication $\mathrm{D}_{\mathrm{cc}}(F)$ needed between Alice and Bob to compute the function. Formal definition of the model can be found in [16].

We will be mostly interested the in one-way communication model. This is a substantially restricted setting, in which only Alice is permitted to send bits to Bob. Formally, the one-way communication complexity $D^{\rightarrow}_{cc}(F)$ is defined to be the smallest integer t, allowing for a protocol where Alice knowing her input x sends t bits to Bob, which together with Bob's input y enable Bob to calculate the value of F.

The bits communicated by Alice depend only on x, that is Alice's message to Bob is $h(x)$ for some fixed total function $h\colon \{0,1\}^n \to \{0,1\}^t$. Bob computes the output $F(x,y)$ based on $h(x)$ and his input y. That is, Bob outputs $\varphi(h(x),y)$ for some fixed total function $\varphi\colon \{0,1\}^t \times \{0,1\}^n \to \{0,1\}$. If (x,y) is within the domain of F, then the equality $\varphi(h(x),y) = F(x,y)$ must be true.

The notion of parity decision tree complexity is a generalization of the well-known decision tree complexity model. In this model, to evaluate a function f for a given input x the protocol queries the parities of some subsets of the bits in x. The cost of the protocol on specified input x is the number of queries the protocol makes on that input. The cost of the protocol (sometimes referred to as the worst-case cost) is maximum over all inputs x, costs of protocol on the input x. The complexity of problem f in the model of parity decision trees $\mathrm{DT}^{\oplus}(f)$ is the minimal over all valid protocols, cost of a protocol for f.

We consider the non-adaptive parity decision tree complexity $\mathrm{NADT}^{\oplus}(f)$. This version differs from its adaptive counterpart in that all the queries should be fixed at once. In other words, each next query should not depend on the answers to previous queries. Next, we give a more formal definition of $\mathrm{NADT}^{\oplus}(f)$.

The protocol of complexity p is defined by n-bit strings $s_1,\dots,s_p$ and a total function $l\colon \{0,1\}^p \to \{0,1\}$. On input x the protocol queries the values of

$$\langle s_1, x\rangle, \dots, \langle s_p, x\rangle$$

and outputs

$$l(\langle s_1, x\rangle, \dots, \langle s_p, x\rangle).$$

The protocol computes partial function f, if for any $x \in \mathrm{Dom}(f)$ we have

$$l(\langle s_1, x\rangle, \dots, \langle s_p, x\rangle) = f(x).$$

Throughout the paper $t, h, \varphi, p, s_1, \dots, s_p, l$ have the same meaning as defined above.

It is easy to see that there is a simple relation between $\mathrm{NADT}^{\oplus}(f)$ and $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F)$.

▶ **Lemma 4.** *For any f we have $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F) \le \mathrm{NADT}^{\oplus}(f)$.*

Proof. Alice and Bob can compute $F(x,y)$ by a simple simulation of $\mathrm{NADT}^{\oplus}$ protocol for f. The idea is that they privately calculate the parities of their respective inputs according to $\mathrm{NADT}^{\oplus}$ protocol, then Alice sends the computed values to Bob, who XORs them with his own parities, and then computes the value of F.

More formally, assume that $\mathrm{NADT}^{\oplus}(f) = p$ and the corresponding protocol is given by $s_1,\dots,s_p \in \{0,1\}^n$ and a function l, that is

$$\forall x \in \mathrm{Dom}(f), f(x) = l(\langle s_1, x\rangle, \dots, \langle s_p, x\rangle).$$

For $i \in [p]$, we let

$$h_i(x) := \langle s_i, x\rangle.$$

For the communication protocol of complexity p we let

$$h(x) = (h_1(x), \dots, h_p(x)),$$
$$\varphi(a, y) := l(a_1 \oplus \langle s_1, y\rangle, \dots, a_p \oplus \langle s_p, y\rangle).$$

Then for any (x, y) such that $x \oplus y \in \mathrm{Dom}(f)$ we have

$$\begin{aligned}\varphi(h(x), y) = l(h_1(x) \oplus \langle s_1, y\rangle, \dots, h_p(x) \oplus \langle s_p, y\rangle) = \\ l(\langle s_1, x\rangle \oplus \langle s_1, y\rangle, \dots, \langle s_p, x\rangle \oplus \langle s_p, y\rangle) = \\ l(\langle s_1, x \oplus y\rangle, \dots, \langle s_p, x \oplus y\rangle) = f(x \oplus y) = F(x, y).\end{aligned}$$

We constructed a p-bit communication protocol for F, and thus

$$\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F) \le p = \mathrm{NADT}^{\oplus}(f). \qquad \blacktriangleleft$$

In this paper, we are mainly interested in whether the inequality in the opposite direction is true.

2.4 Covering Codes

▶ **Definition 5.** *A subset $\mathcal{C} \subseteq \{0,1\}^n$ is a (n, K, R) covering code if $|\mathcal{C}| \le K$ and for any $x \in \{0,1\}^n$ there is $c \in \mathcal{C}$ such that $\mathrm{dist}(x, c) \le R$. In other words, all points in $\{0,1\}^n$ are covered by balls of radius R with centers in $\mathcal{C}$.*

The following general bounds on K are known for covering codes.

▶ **Theorem 6** ([6, Theorem 12.1.2])**.** *For any (n, K, R) covering code we have*

$$\log K \ge n - \log V(n, R).$$

For any n and any $R \le n$ there is a (n, K, R) covering code with

$$\log K \le n - \log V(n, R) + \log n.$$

We will use the following well known fact.

▶ **Theorem 7** ([6, Section 2.6])**.** *If $n = 2^m - 1$ for some m, then Boolean cube $\{0,1\}^n$ can be splitted into disjoint balls of radius 1.*

This construction is known as a Hamming error correcting code. Note that it is a $(n = 2^m - 1, \frac{2^n}{n+1}, 1)$ covering code.

▶ **Definition 8.** *For two covering codes $\mathcal{C}_1$ and $\mathcal{C}_2$ their direct sum is*

$$\mathcal{C}_1 \oplus \mathcal{C}_2 = \{(c_1, c_2) \mid c_1 \in \mathcal{C}_1, c_2 \in \mathcal{C}_2\}.$$

▶ **Lemma 9** ([6, Theorem 12.1.2])**.** *If $\mathcal{C}_1$ is a (n_1, K_1, R_1) covering code and $\mathcal{C}_2$ is a (n_2, K_2, R_2) covering code, then $\mathcal{C}_1 \oplus \mathcal{C}_2$ has parameters $(n_1 + n_2, K_1K_2, R_1 + R_2)$.*

We need the following bounds on the sizes of Hamming balls (see, e.g. [14, Appendix A]).

▶ **Lemma 10.** *For any n and $k \le n$ we have*

$$\left(\frac{n}{k}\right)^k \le V(n, k) \le \left(\frac{en}{k}\right)^k.$$

▶ **Lemma 11.** *For any constant $0 < c < 1$ we have*

$$\binom{n}{cn} = O\left(\frac{1}{\sqrt{n}} 2^{H(c)n}\right).$$

For any constant $0 < c < 1/2$ we have

$$V(n, cn) = O\left(\frac{1}{\sqrt{n}} 2^{H(c)n}\right),$$

where H is the binary entropy function.

▶ **Lemma 12** ([24, Section 5.4])**.**

$$V\left(n, \frac{n}{2} - \Theta(\sqrt{n \log n})\right) = \frac{2^n}{poly(n)}.$$

For the binary entropy function $H(x)$ we will use the following simple fact.

▶ **Lemma 13.** *For any constant $c \in (0, 1)$ and for any $\alpha_n \xrightarrow[n\to\infty]{} 0$ we have*

$$H(c + \alpha_n) = H(c) + O(\alpha_n),$$

where the constant in O-notation might depend on c, but not on n.

This is true since the derivative of H is upper bounded by a constant in any small enough neighborhood of c.

3 Linear-algebraic framework

3.1 Graph-based analysis of one-way communication protocols

Recall that in a one-way communication protocol of complexity t for $F(x, y) = f(x \oplus y)$ Alice on input $x \in \{0,1\}^n$ first sends to Bob $h(x)$ for some fixed $h\colon \{0,1\}^n \to \{0,1\}^t$. After that Bob computes the output $\varphi(h(x), y)$, where $y \in \{0,1\}^n$ is Bob's input and $\varphi\colon \{0,1\}^t \times \{0,1\}^n \to \{0,1\}$.

Let's consider the partition $\mathcal{H} = \{H_a \mid a \in \{0,1\}^t\}$, where for any $a \in \{0,1\}^t$

$$H_a = h^{-1}(a).$$

We refer to $\mathcal{H}$ as *h-induced partition*. A class H_a of this partition is the set of inputs for which Alice sends Bob the same message.

Consider two arbitrary inputs $x, y \in \{0,1\}^n$. We call the vector $\Delta = x \oplus y$ the *shift* between x and y. The intuition is that y is equal to the shift $x \oplus \Delta$ of x by y (and vise versa).

We say that $\Delta \in \{0,1\}^n$ is a *good shift* if there is a pair $x, y \in \{0,1\}^n$ such that $x \oplus y = \Delta$ and $h(x) = h(y)$, or equivalently, if x and y belong to the same class of $\mathcal{H}$. Note that f does not necessarily need to be defined on inputs x and y. However, it turns out that on the domain of f the value of f is invariant under good shifts.

▶ **Lemma 14.** *Assume that Δ is a good shift. Consider any $v, u \in \mathrm{Dom}(f)$ such that $v \oplus u = \Delta$. Then, $f(v) = f(u)$.*

Proof. Since Δ is good, there are x and y such that $h(x) = h(y)$ and $x \oplus y = \Delta$. Then

$$f(v) = \varphi(h(x), x \oplus v) = \varphi(h(y), x \oplus v) = f(v \oplus x \oplus y) = f(v \oplus \Delta) = f(u). \quad \blacktriangleleft$$

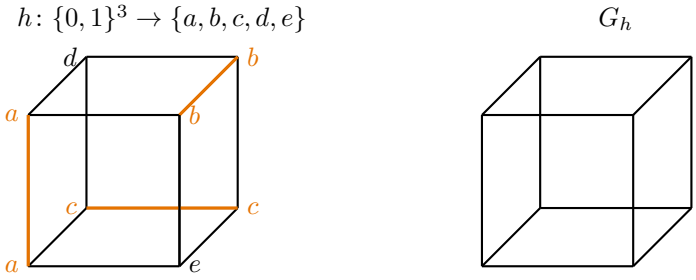


Figure 1 Example of total h-induced graph.

This leads us to the following notion.

▶ **Definition 15.** *For the functions $f : \{0,1\}^n \to \{0,1\}$, $f(x \oplus y) = \varphi(h(x), y)$ let the* total h-induced graph *be the graph with vertices $\{0,1\}^n$ and with an edge between $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$ if $x \oplus y$ is a good shift for h. Now remove vertices where the function f is undefined. The resulting graph is called the* partial h-induced graph.

There is an alternative way of thinking about total h-induced graph. Consider a graph with vertices labeled $\{0,1\}^n$ in which we connect two vertices if the value of h on these vertices is the same. Clearly it is a subgraph of the total h-induced graph. Now consider a shift of this graph, that is, a graph in which we XORed labels of all vertices with some fixed vector. This graph is also a subset of the total h-induced graph. By considering all possible shifts and taking the union of all graphs we will get the total h-induced graph. See Figure 1 for an example of total h-induced graph.

By transitivity, if (h, φ) form a valid communication protocol then f assigns identical values to each connected component in partial h-induced graph. The converse is also true.

▶ **Theorem 16.** *Consider a function $f : \{0,1\}^n \to \{0,1\}$. For a function $h : \{0,1\}^n \to \{0,1\}^t$ there is a function $\varphi : \{0,1\}^t \times \{0,1\}^n \to \{0,1\}$ such that (h, φ) form a valid communication protocol for f if and only if f assigns the same value to each connected component in the partial h-induced graph.*

Proof. As discussed above, if (h, φ) forms a valid communication protocol, then f assigns the same value to each connected component of the partial h-induced graph.

It remains to prove the converse statement. We assume that f assigns the same value to each connected component and we need to show that there is φ such that

$$\forall (x, y) \in \mathrm{Dom}(F), \quad F(x, y) = \varphi(h(x), y).$$

The proof idea is the following. Each input $(x, y) \in \{0,1\}^n \times \{0,1\}^n$ to F yields an input $(a, y) \in \{0,1\}^t \times \{0,1\}^n$ to φ where $\alpha = h(x)$. We define φ on (α, y) to be equal to F on a single corresponding F-input (x', y) with $x' \in h^{-1}(\alpha)$. Then we prove that φ defined that way gives a communication protocol computing F correctly on all inputs $(x, y) \in \mathrm{Dom}(f)$

Formally, we define φ as follows. For each $\alpha \in \{0,1\}^t$ and $y \in \{0,1\}^n$, consider $x' \in \{0,1\}^n$ such that $h(x') = \alpha$ and $(x', y) \in \mathrm{Dom}(F)$. If there is no such x' we define $\varphi(\alpha, y)$ arbitrarily. If there is such an x', let

$$\varphi(\alpha, y) := F(x', y).$$

Now we show that the resulting protocol computes $F(x, y)$ correctly for any (x, y).

Consider arbitrary $(x, y) \in \mathrm{Dom}(F)$. Consider x' chosen for $\alpha = h(x)$ and y (it exists, since clearly x itself satisfies all the necessary conditions).

Thus, we have

$$\varphi(h(x), y) = F(x', y).$$

It remains to prove that

$$F(x', y) = F(x, y)$$

or equivalently,

$$f(x' \oplus y) = f(x \oplus y).$$

For XOR of these two inputs of f we have

$$(x' \oplus y) \oplus (x \oplus y) = x' \oplus x.$$

Since $h(x) = h(x')$, we have that $x' \oplus x$ is a good shift. And since

$$(x, y), (x', y) \in \mathrm{Dom}(F),$$

we have

$$x \oplus y, x' \oplus y \in \mathrm{Dom}(f).$$

We have that vertices $x \oplus y$ and $x' \oplus y$ are connected in the partial h-induced graph and by Lemma 14 f assigns the same value to them. Hence, the function φ, together with h, forms a communication protocol for F. ◀

3.2 Using coset structures on a Boolean cube to analyze non-adaptive parity decision trees

We consider the vertices of the Boolean cube as a vector space $\mathbb{F}_2^n$. We show that a NADT$^{\oplus}$ protocol corresponds to a linear subspace of $\mathbb{F}_2^n$ such that f is constant on each of its cosets (the coset for a linear subspace L and a vector l is defined as the set $\{x + l | l \in L\}$ and denoted $L + l$).

▶ **Theorem 17.** *Let $f\colon \{0,1\}^n \to \{0,1\}$. There is a p-bit* NADT$^{\oplus}$ *protocol for f if and only if there exists an $n - p$ dimensional subspace of $\{0,1\}^n$ such that for each coset of that subspace, f assigns the same value to all inputs of the coset where f is defined.*

Proof. Suppose $s_1, \ldots, s_p, l$ form a valid NADT$^{\oplus}$ protocol for f. We construct a matrix S with rows $s_1, \ldots, s_p$. If some of the rows are linearly dependent, we add rows arbitrarily to make the rank of S equal to p. When S is multiplied on the right by some vector x, we obtain all inner products of x with vectors $s_1, \ldots, s_p$ (and possibly other bits if we added rows).

Consider the vector subspace $\{x | Sx = 0\}$. This is an $n - p$ dimensional space. For all points in the same coset of this subspace, the tuple consisting of values of the inner products $(\langle s_1, x\rangle, \ldots, \langle s_p, x\rangle)$ is the same, so is the value of $l(\langle s_1, x\rangle, \ldots, \langle s_p, x\rangle)$. For all points where f is defined and lying in the same coset, the value of f must be equal to the value of l and thus the same for all points in the coset.

In the reverse direction, let $\langle e_1, \ldots, e_{n-p} \rangle$ be an $n-p$ dimensional subspace of $\{0,1\}^n$ such that for each of its cosets f is constant on all points of that coset on which it is defined. We can represent this subspace in the form $\{x | Sx = 0\}$ for some matrix S of size $p \times n$.

Vectors x and y are in the same coset of $\langle e_1, \ldots, e_{n-p} \rangle$ iff $Sx = Sy$. Thus, to compute $f(x)$ it is enough to compute the inner product of x with the rows of S. ◀

▶ **Corollary 18.** *Consider a function $f : \{0,1\}^n \to \{0,1\}$ having valid communication protocol $f(x \oplus y) = \varphi(h(x), y)$ where $h : \{0,1\}^n \to \{0,1\}^t, \varphi : \{0,1\}^t \times \{0,1\}^n \to \{0,1\}$. Suppose there is an $n-t$ dimensional subspace L of $\{0,1\}^n$ and consider subgraphs of partial h-induced graph each over vertices belonging to different cosets of L. If all of these subgraphs are connected then $\mathrm{NADT}^{\oplus}(f) \leq t$.*

Proof. By Theorem 16 f is constant on each coset. By Theorem 17 it follows that $\mathrm{NADT}^{\oplus}(f) \leq t$. ◀

4 Equality between $\mathrm{D}^{\to}_{\mathrm{cc}}(F)$ and $\mathrm{NADT}^{\oplus}(f)$

In this section we will show that if $\mathrm{D}^{\to}_{\mathrm{cc}}(F) = t$ and the number of undefined inputs is small, then $\mathrm{NADT}^{\oplus}(f) = t$ as well. More specifically, we prove the following theorem.

▶ **Theorem 19.** *If for the function $f\colon \{0,1\}^n \to \{0,1\}$ we have $\mathrm{D}^{\to}_{\mathrm{cc}}(F) = t$, where $F(x,y) = f(x \oplus y)$, and f is undefined on less than $\binom{n-t+1}{\lfloor \frac{n-t}{2} \rfloor - 1}$ inputs, then $\mathrm{NADT}^{\oplus}(f) = t$.*

By Lemma 11 we have that $\binom{n-t+1}{\lfloor \frac{n-t+1}{2} \rfloor} = O(\frac{2^{n-t}}{\sqrt{n-t}})$ and since $\lfloor \frac{n-t}{2} \rfloor - 1$ differs from $\lfloor \frac{n-t+1}{2} \rfloor$ by only a constant, it is easy to see that the same estimate applies to $\binom{n-t+1}{\lfloor \frac{n-t}{2} \rfloor - 1}$ as well. Thus, the number of undefined inputs is $O(\frac{2^{n-t}}{\sqrt{n-t}})$.

The rest of the section is devoted to the proof of Theorem 19. The idea of the proof is as follows. Consider the h-induced partition $\mathcal{H}$ corresponding to the communication protocol of complexity t. We show that either the partition $\mathcal{H}$ corresponds to the cosets of an $n-t$ dimensional subspace of $\{0,1\}^n$, which allows us to construct an $\mathrm{NADT}^{\oplus}$ protocol, or there exist *many* good shifts. The structure of these good shifts imposes restrictions on f that again allow us to construct an $\mathrm{NADT}^{\oplus}$ protocol.

We start with a simple case.

▶ **Lemma 20.** *If there exists t-bit communication protocol, (h, φ) for a function $f\colon \{0,1\}^n \to \{0,1\}$, and the h-induced partition $\mathcal{H}$ corresponds to cosets of an $n-t$ dimensional subspace L of $\{0,1\}^n$, then $\mathrm{NADT}^{\oplus}(f) \leq t$.*

Proof. Since the partition $\mathcal{H}$ corresponds to the cosets of L, we have that for any inputs x and y, if $h(x) = h(y)$, then $x \oplus y \in L$ and vice versa. In other words, all good shifts are in L and any shift in L is good. Thus, connected components of the total h-induced graph are cosets of L and are fully connected. By Corollary 18 we have that $\mathrm{NADT}^{\oplus}(f) \leq t$. ◀

The structure of the proof for the other case is the following. We show that the total h-induced graph is structured into connected components, each of which is a coset of a k-dimensional subspace of $\{0,1\}^n$ for $k \geq n-t$. We show that there is a bijective graph homomorphism of the k-dimensional Boolean cube onto each component. Furthermore, each vertex in the total h-induced graph has a degree of at least $\frac{2^n}{2^t} - 1$. We show that if we remove fewer than $\binom{n-t+1}{\lfloor \frac{n-t}{2} \rfloor - 1}$ vertices, each coset still contains one connected component. By the way of contradiction, suppose this is not the case and some coset contains more than one

connected component. We consider the smallest of these components, denote the set of its nodes by S. We show that the number of neighboring vertices of S in the total h-induced graph (excluding S itself) is not less than $\binom{n-t+1}{\lfloor\frac{n-t}{2}\rfloor-1}$. This implies that after removing the undefined inputs of f S cannot not be separated from other nodes in the coset. To show this we treat separately cases of large and small $|S|$. For small $|S|$ we use the fact that vertices have high degree. For large $|S|$ we use the vertex-isoperimetric inequality for the Boolean cube.

▶ **Lemma 21.** *Suppose there exists a t-bit communication protocol (h, φ) for $f\colon \{0,1\}^n \to \{0,1\}$ and the h-induced partition $\mathcal{H}$ classes do not correspond to cosets of an $n-t$-dimensional subspace of $\{0,1\}^n$. Let D be the set of good shifts for h. Then D contains a minimum of $n-t+1$ linearly independent vectors.*

Proof. Suppose there are at most $n-t$ linearly independent good shifts $e_1, \dots, e_{n-t}$. Consider a linear subspace of $\{0,1\}^n$ spanned over by these shifts and add some vectors to it to make it exactly $n-t$ dimensional if needed. Denote the resulting subspace L. As classes of $\mathcal{H}$ do not correspond to the cosets of L and there are 2^t of both classes and cosets there exist two elements belonging to the same class and different cosets. Their XOR is a good shift linearly independent with $e_1, \dots, e_{n-t}$. We got a contradiction implying the lemma. ◀

▶ **Lemma 22.** *Suppose there exists t-bit communication protocol (h, φ) for f. Let D be the set of all good shifts for h and $\{e_1, \dots, e_k\}$ be the largest linearly independent subset of D. Then the total h-induced graph $\mathcal{H}$ has the following properties.*

- *Cosets of the subspace $\langle e_1, \dots, e_k\rangle$ are connected components of $\mathcal{H}$.*
- *There is a bijective graph homomorphism of k-dimensional Boolean cube into each coset.*

Proof. It is easy to see that all vertices in any coset are connected to each other. Let's show that no edges exist between vertices of different cosets. Assume by contradiction that there is an edge between vertices v and u from different cosets. Note that $u \oplus v \notin \langle e_1, \dots, e_k\rangle$. Thus, vectors $e_1, \dots, e_k, u \oplus v$ form a linearly independent system of size $k+1$, which is a contradiction.

Now, let's construct a homomorphism q from the Boolean cube $\{0,1\}^k$ into the coset $v + \langle e_1, \dots, e_k\rangle$ for an arbitrary vertex v. Consider a matrix B that has vectors $e_1, \dots, e_k$ as its columns and let $q(x) = v \oplus Bx$. The image of q is within the coset $v + \langle e_1, \dots, e_k\rangle$, as columns of B belong to the subspace $\langle e_1, \dots, e_k\rangle$. The mapping is bijective on $v + \langle e_1, \dots, e_k\rangle$, as B's columns are linearly independent. Finally, consider a pair of vertices x, y adjacent in a Boolean cube. Since the vertices are adjacent, they only differ in a single bit i. Thus,

$$q(x) \oplus q(y) = (v \oplus Bx) \oplus (v \oplus By) = B(x \oplus y) = e_i.$$

Since $e_i \in D$, an edge exists between $q(x)$ and $q(y)$, implying that q is a graph homomorphism. ◀

▶ **Lemma 23.** *Suppose there exists t-bit communication protocol (h, φ) for $f\colon \{0,1\}^n \to \{0,1\}$. Then in the total h-induced graph, the degree of any vertex is not less than $\frac{2^n}{2^t} - 1$.*

Proof. Let's consider the largest class in the h-induced partition $\mathcal{H}$. Since the number of classes is at most 2^t, the largest class contains at least $\frac{2^n}{2^t}$ elements. Fix an element of the class and compute its XOR with all elements in the same class $\mathcal{H}$. We have $\frac{2^n}{2^t}$ XORs in total, $\frac{2^n}{2^t} - 1$ of which are non-zero. Since each XOR is computed between elements in the same class, these XORs are good shifts. For all vertices in the h-induced graph for each good shift we draw an edge from the vertex corresponding to this shift. Therefore, the degree of any vertex is at least $\frac{2^n}{2^t} - 1$. ◀

▶ **Lemma 24.** *If A is a subset of k-dimensional Boolean cube satisfying $V\left(m, \left\lfloor \frac{m-1}{2} \right\rfloor - 2\right) \leq |A| \leq 2^{k-1}$ for some m, then $|\Gamma' A| \geq \binom{m}{\left\lfloor \frac{m-1}{2} \right\rfloor - 1}$.*

The proof of the lemma can be found in the full version of the paper. The proof heavily relies on Theorem 3. Finally, we are ready to prove Theorem 19.

Proof of Theorem 19. We are given t-bit communication protocol (h, φ) for F. By Lemma 21, the h-induced partition $\mathcal{H}$ either corresponds to cosets of an $n-t$ dimensional subspace of $\{0,1\}^n$ (and then by Lemma 20 we have $\mathrm{NADT}^{\oplus}(f) \leq t$), or the set of good shifts D contains at least $n-t+1$ linearly independent vectors. Let $\langle e_1, \ldots, e_k \rangle$, where $k \geq n-t+1$, be the largest subset of linearly independent vectors in D. Consider the cosets of the subspace $\langle e_1, \ldots, e_k \rangle$. We will show that if we remove fewer than $\binom{n-t+1}{\left\lfloor \frac{n-t}{2} \right\rfloor - 1}$ vertices from the total h-induced graph, each coset will contain no more than one connected component. Assume by contradiction that after removing the vertices, some coset splits into several connected components. Let A be the smallest of these components. If there are at most $V(n-t+1, \lfloor \frac{n-t}{2} \rfloor - 2) - 1$ vertices in A, consider a vertex a in A. Given the degree of a is at least $2^{n-t} - 1$, a has at least

$$2^{n-t} - V\left(n-t+1, \left\lfloor \frac{n-t}{2} \right\rfloor - 2\right) \geq V\left(n-t+1, \left\lfloor \frac{n-t}{2} \right\rfloor\right) - V\left(n-t+1, \left\lfloor \frac{n-t}{2} \right\rfloor - 2\right) \geq \binom{n-t+1}{\left\lfloor \frac{n-t}{2} \right\rfloor - 1}$$

neighbors outside A.

On the other hand, suppose A has at least $V(n-t+1, \lfloor \frac{n-t}{2} \rfloor - 2)$ vertices. Since A is the smallest connected component in its coset it also follows that A has no more than 2^{k-1} vertices. By Lemma 24 we have $|\Gamma' A| \geq \binom{n-t+1}{\left\lfloor \frac{n-t}{2} \right\rfloor - 1}$, which is more than the number of removed vertices, a contradiction. Thus, cosets cannot be split into several components and by Corollary 18 we have $\mathrm{NADT}^{\oplus}(f) \leq n-k \leq t-1$, which is a contradiction. ◀

5 Separations between $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F)$ and $\mathrm{NADT}^{\oplus}(f)$

In this section we show that if the number of undefined inputs is large, there is a gap between $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F)$ and $\mathrm{NADT}^{\oplus}(f)$. That is, we aim to come up with a function f such that $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F)$ is small and $\mathrm{NADT}^{\oplus}(f)$ is large.

The key idea in our construction is that in h-induced graph for the intended communication protocol the edges connect only vertices with small Hamming distance between them. Then, if the function f has 0-inputs and 1-inputs far away from each other, they are not connected and h corresponds to a valid protocol. We will ensure that at the same time f has large $\mathrm{NADT}^{\oplus}$ complexity.

We start with the construction of the functions, then investigate their $\mathrm{NADT}^{\oplus}$ complexity and then prove upper bounds on $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}$ complexity of the corresponding XOR functions. The latter part is through the reduction to covering codes.

▶ **Definition 25.** *For a parameter k define $f_k \colon \{0,1\}^n \to \{0,1,\perp\}$ in the following way.*

$$f_k(x) = \begin{cases} 0 & \text{for } |x| \leq k, \\ \perp & \text{for } k+1 \leq |x| \leq n-1, \\ 1 & \text{for } |x| = n. \end{cases}$$

We denote the corresponding XOR function by F_k.

Note, that the number of undefined inputs in f_k is $V(n, n-k-1)-1$.

It turns out that f_k has reasonably large $\mathrm{NADT}^{\oplus}$ and $\mathrm{DT}^{\oplus}$ complexities.

▶ **Theorem 26.** $\mathrm{NADT}^{\oplus}(f_k) = \mathrm{DT}^{\oplus}(f_k) = k+1$.

Proof. Since $\mathrm{DT}^{\oplus}(f) \le \mathrm{NADT}^{\oplus}(f)$ for any f, it is enough to prove that $\mathrm{NADT}^{\oplus}(f_k) \le k+1$ and $\mathrm{DT}^{\oplus}(f_k) \ge k+1$.

For the upper bound, observe that it is enough to query variables $x_1, \dots, x_{k+1}$. If all of them are equal to 1, we output 1, otherwise we output 0. It is easy to see that this protocol computes f_k correctly.

For the lower bound suppose, for the sake of contradiction, that an adaptive parity decision tree exists that can compute the function f with k or fewer queries. Consider the path corresponding to the input $e = (1, \dots, 1)$. Let's assume that the decision tree queried the parities $\langle s_i, e\rangle$ for $s_1, \dots, s_k$. The answers to the queries are equal to $\langle s_1, e\rangle, \dots, \langle s_k, e\rangle$. Consider a matrix $B \subseteq \mathbb{F}^{k\times n}$ consisting of rows $s_1, \dots, s_k$.

Denote $a = Be$. In particular, we have that a lies in the subspace generated by columns of B. Since the rank of B is at most k (the matrix has k rows), there is a subset of at most k columns generating this subspace. In particular, there is $x \in \{0,1\}^n$ with $|x| \le k$, such that $a = Bx$. That is, $Be = Bx$ and the protocol behaves the same way on e and x, which is a contradiction, since $f_k(e) = 1$ and $f_k(x) = 0$. ◀

▶ Remark 27. Since f_k has large (adaptive) parity decision tree complexity and for any $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ we have $\mathrm{D}^{\to}_{\mathrm{cc}}(F) \ge \mathrm{D}_{\mathrm{cc}}(F)$, all separations provided by functions f_k translate into the same separations between $\mathrm{DT}^{\oplus}$ and D_{cc}.

Next, we proceed to the upper bound on the $\mathrm{D}^{\to}_{\mathrm{cc}}(F_k)$.

▶ **Theorem 28.** *Suppose for some n, k and t there is a $(n, 2^t, R)$ covering code $\mathcal{C}$ for $R = \left\lfloor \frac{n-k-1}{2} \right\rfloor$. Then, $\mathrm{D}^{\to}_{\mathrm{cc}}(F_k) \le t$.*

Proof. Split the points of $\{0,1\}^n$ into balls with radius R with centers in the points of $\mathcal{C}$ (if some point belongs to several balls, attribute it to one of them arbitrarily). This results in a partition of the cube into 2^t subsets with the diameter of each subset at most $n-k-1$.

The proof can be finished through Theorem 16, but to make it more self-contained we directly describe communication protocol.

On input x Alice sends as $h(x)$ the index of the ball containing x. Bob computes $\neg y$, componentwise negation of y, and outputs 1 if it is in the same ball. If this is not the case, Bob outputs 0.

Clearly, the complexity of this protocol is at most t. For the correctness of the protocol, if $f(x \oplus y) = 1$, then $x = \neg y$ and the protocol clearly outputs 1. However, if $f(x \oplus y) = 0$, then $|x \oplus y| \le k$ and thus $\mathrm{dist}(x, \neg y) \ge n-k$. In this case x and $\neg y$ are not in the same ball and the protocol outputs 0. ◀

▶ **Theorem 29.** *For any n and k we have*

$$\mathrm{D}^{\to}_{\mathrm{cc}}(F_k) \le n - \log V(n, R) + \log n$$

for $R = \left\lfloor \frac{n-k-1}{2} \right\rfloor$.

Proof. By Theorem 6 there exists a $(n, 2^t, R)$ covering code for

$$\log 2^t = t \le n - \log V(n, R) + \log n.$$

The theorem follows from Theorem 28. ◀

From this we can get a separation for a wide range of parameters.

▶ **Corollary 30.** *Suppose $k = cn$ for some constant $0 < c < 1$. Then* $\mathrm{NADT}^{\oplus}(f_k) = cn + 1$ *and*

$$\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F_k) \leq \left(1 - H\left(\frac{1-c}{2}\right)\right) n + O(\log n).$$

In particular, $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F_k) < \mathrm{NADT}^{\oplus}(f_k)$. *The number of undefined inputs for f_k is $2^n - O(\frac{2^{H(c)n}}{\sqrt{n}})$ if $c < 1/2$, is equal to $(1 + o(1))2^{n-1}$ if $c = 1/2$, and is $O(\frac{2^{H(1-c)n}}{\sqrt{n}})$ if $c > 1/2$.*

Proof. The equality for $\mathrm{NADT}^{\oplus}$ is proved in Theorem 26.

For communication complexity bound we apply Theorem 29. We have $R = \left\lfloor \frac{(1-c)n-1}{2} \right\rfloor = \frac{(1-c)n}{2} + O(1)$ and by Lemmas 11 and 13 we have

$$\log V(n, R) = H\left(\frac{1-c}{2}\right) n - O(\log n).$$

By Theorem 29 we have

$$\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F_k) \leq n - \log V(n, R) + \log n = \left(1 - H\left(\frac{1-c}{2}\right)\right) n + O(\log n).$$

To show that $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F_k) < \mathrm{NADT}^{\oplus}(f_k)$ we need to compare $k = cn$ with the bound on communication complexity. It is easy to see that

$$1 - H\left(\frac{1-c}{2}\right) < c$$

for all $0 < c < 1$ (the left hand-side and the right hand-side are equal for $c = 0$ and $c = 1$ and the left hand-side is concave in c).

The bounds on the number of undefined inputs follow easily from Lemma 11. ◀

The largest gap we can get is the following.

▶ **Corollary 31.** *For $k = \Theta(\sqrt{n \log n})$ we have that* $\mathrm{NADT}^{\oplus}(f_k) = \Theta(\sqrt{n \log n})$ *and* $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F_k) = O(\log n)$. *The number of undefined inputs for f_k is $2^n - 2^{\Theta(\sqrt{n}\log^{3/2} n)}$.*

Proof. For $k = \Theta(\sqrt{n \log n})$ we have $R = \frac{n}{2} - \Theta(\sqrt{n \log n})$ in Theorem 29. By Lemma 12 we have $V(n, R) = \frac{2^n}{\mathrm{poly}(n)}$ and as a result $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F_k) = O(\log n)$.

For the number of undefined inputs, we apply Lemma 10:

$$\left(\frac{n}{k}\right)^k \leq V(n, k) \leq \left(\frac{en}{k}\right)^k.$$

For $k = \Theta(\sqrt{n \log n})$ it is easy to see that both sides are $2^{\Theta(\sqrt{n}\log^{3/2} n)}$. From this the estimate on the number of undefined inputs follows. ◀

6 Extreme Cases

In this section we discuss extreme cases. All proves can be found in the full version of the paper.

For small values of complexity measures we have the following equality results.

▶ **Theorem 32.** *Suppose F satisfies* $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F) = 1$. *It then follows that* $\mathrm{NADT}^{\oplus}(f) = 1$.

▶ **Theorem 33.** *If function f is undefined on fewer than 2^{n-3} inputs and $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F) = 2$, then $\mathrm{NADT}^{\oplus}(f) = 2$.*

On the other end of the spectrum, we show that if $\mathrm{NADT}^{\oplus}(f)$ is really large, then it is equal for all partial functions.

▶ **Theorem 34.** *For any partial function $f\colon \{0,1\}^n \to \{0,1,\perp\}$, if $\mathrm{NADT}^{\oplus}(f) \geq n-1$, then $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F) = \mathrm{NADT}^{\oplus}(f)$.*

The largest value of $\mathrm{NADT}^{\oplus}$ for which we get separation is $n-2$.

▶ **Theorem 35.** *$\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F_{n-3}) \leq n - \Theta(\log n)$, whereas $\mathrm{NADT}^{\oplus}(f_{n-3}) = n-2$. The number of undefined inputs for f_{n-3} is $\frac{n(n+1)}{2}$.*

The smallest value of $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}$ for which we get a separation is 7.

▶ **Theorem 36.** *For any $n \geq 32$ we have $\mathrm{D}^{\rightarrow}_{\mathrm{cc}}(F_7) \leq 7$, whereas $\mathrm{NADT}^{\oplus}(f_7) = 8$.*

References

1. Anurag Anshu, Naresh Goud Boddu, and Dave Touchette. Quantum log-approximate-rank conjecture is also false. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 982–994. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00063.
2. Arkadev Chattopadhyay, Ankit Garg, and Suhail Sherif. Towards stronger counterexamples to the log-approximate-rank conjecture. In Mikolaj Bojanczyk and Chandra Chekuri, editors, *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15-17, 2021, Virtual Conference*, volume 213 of *LIPIcs*, pages 13:1–13:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSTTCS.2021.13.
3. Arkadev Chattopadhyay and Nikhil S. Mande. A lifting theorem with applications to symmetric functions. In Satya V. Lokam and R. Ramanujam, editors, *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017, December 11-15, 2017, Kanpur, India*, volume 93 of *LIPIcs*, pages 23:1–23:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.FSTTCS.2017.23.
4. Arkadev Chattopadhyay, Nikhil S. Mande, Swagato Sanyal, and Suhail Sherif. Lifting to parity decision trees via stifling. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*, volume 251 of *LIPIcs*, pages 33:1–33:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ITCS.2023.33.
5. Arkadev Chattopadhyay, Nikhil S. Mande, and Suhail Sherif. The log-approximate-rank conjecture is false. *J. ACM*, 67(4):23:1–23:28, 2020. doi:10.1145/3396695.
6. Gérard D. Cohen, Iiro S. Honkala, Simon Litsyn, and Antoine Lobstein. *Covering Codes*, volume 54 of *North-Holland mathematical library*. North-Holland, 2005.
7. Susanna F. de Rezende, Or Meir, Jakob Nordström, Toniann Pitassi, Robert Robere, and Marc Vinyals. Lifting with simple gadgets and applications to circuit and proof complexity. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 24–30. IEEE, 2020. doi:10.1109/FOCS46700.2020.00011.
8. Uma Girish, Ran Raz, and Avishay Tal. Quantum versus randomized communication complexity, with efficient players. *Comput. Complex.*, 31(2):17, 2022. doi:10.1007/s00037-022-00232-7.
9. Uma Girish, Makrand Sinha, Avishay Tal, and Kewen Wu. Fourier growth of communication protocols for XOR functions. *CoRR*, abs/2307.13926, 2023. doi:10.48550/arXiv.2307.13926.

10 Parikshit Gopalan, Ryan O'Donnell, Rocco A. Servedio, Amir Shpilka, and Karl Wimmer. Testing fourier dimensionality and sparsity. *SIAM J. Comput.*, 40(4):1075–1100, July 2011. doi:10.1137/100785429.

11 Lianna Hambardzumyan, Hamed Hatami, and Pooya Hatami. Dimension-free bounds and structural results in communication complexity. *Israel Journal of Mathematics*, 253:555–616, 2023. doi:10.1007/s11856-022-2365-8.

12 L. H. Harper. *Global Methods for Combinatorial Isoperimetric Problems*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2004. doi:10.1017/CBO9780511616679.

13 Hamed Hatami, Kaave Hosseini, and Shachar Lovett. Structure of protocols for XOR functions. *SIAM J. Comput.*, 47(1):208–217, 2018. doi:10.1137/17M1136869.

14 Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-24508-4.

15 Sampath Kannan, Elchanan Mossel, Swagato Sanyal, and Grigory Yaroslavtsev. Linear sketching over f_2. In Rocco A. Servedio, editor, *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, volume 102 of *LIPIcs*, pages 8:1–8:37. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CCC.2018.8.

16 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1996. doi:10.1017/CBO9780511574948.

17 Bruno Loff and Sagnik Mukhopadhyay. Lifting theorems for equality. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPIcs*, pages 50:1–50:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.50.

18 Nikhil S. Mande, Swagato Sanyal, and Suhail Sherif. One-way communication complexity and non-adaptive decision trees. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPIcs*, pages 49:1–49:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.49.

19 Ashley Montanaro and Tobias Osborne. On the communication complexity of XOR functions. *CoRR*, abs/0909.3392, 2009. arXiv:0909.3392.

20 A. Rao and A. Yehudayoff. *Communication Complexity: and Applications*. Cambridge University Press, 2020. URL: https://books.google.com/books?id=emw8PgAACAAJ.

21 Swagato Sanyal. Fourier sparsity and dimension. *Theory Comput.*, 15:1–13, 2019. doi:10.4086/toc.2019.v015a011.

22 Alexander A. Sherstov, Andrey A. Storozhenko, and Pei Wu. An optimal separation of randomized and quantum query complexity. *SIAM J. Comput.*, 52(2):525–567, 2023. doi:10.1137/22m1468943.

23 Makrand Sinha and Ronald de Wolf. Exponential separation between quantum communication and logarithm of approximate rank. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 966–981. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00062.

24 J.H. Spencer and L. Florescu. *Asymptopia*. Student mathematical library. American Mathematical Society, 2014. URL: https://books.google.com/books?id=uBMLugEACAAJ.

25 Hing Yin Tsang, Chung Hoi Wong, Ning Xie, and Shengyu Zhang. Fourier sparsity, spectral norm, and the log-rank conjecture. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 658–667. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.76.

26 Shengyu Zhang. Efficient quantum protocols for XOR functions. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1878–1885. SIAM, 2014. doi:10.1137/1.9781611973402.136.

27 Zhiqiang Zhang and Yaoyun Shi. On the parity complexity measures of boolean functions. *Theor. Comput. Sci.*, 411(26-28):2612–2618, 2010. doi:10.1016/j.tcs.2010.03.027.

Bounds on the Total Coefficient Size of Nullstellensatz Proofs of the Pigeonhole Principle

Aaron Potechin
University of Chicago, IL, USA

Aaron Zhang
The Voleon Group, Berkeley, CA, USA

Abstract

We show that the minimum total coefficient size of a Nullstellensatz proof of the pigeonhole principle on $n+1$ pigeons and n holes is $2^{\Theta(n)}$. We also investigate the ordering principle and construct an explicit Nullstellensatz proof for the ordering principle on n elements with total coefficient size $2^n - n$.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases Proof complexity, Nullstellensatz, pigeonhole principle, coefficient size

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.117

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2205.03577` [29]

Funding *Aaron Potechin*: NSF grant CCF-2008920
Aaron Zhang: NDSEG Fellowship 9422254702

1 Introduction

Given a system $\{p_i = 0 : i \in [m]\}$ of m polynomial equations, a Nullstellensatz proof of infeasibility is an equality of the form $1 = \sum_{i=1}^{m} p_i q_i$ for some polynomials $\{q_i = 0 : i \in [m]\}$. Hilbert's Nullstellensatz[1] says that the Nullstellensatz proof system is complete over algebraically closed fields, i.e., a system of polynomial equations has no solutions over an algebraically closed field if and only if there is a Nullstellensatz proof of infeasibility. However, Hilbert's Nullstellensatz does not give any bounds on the degree or size needed for Nullstellensatz proofs.

The degree of Nullstellensatz proofs has been extensively studied. Grete Hermann showed a doubly exponential degree upper bound for the ideal membership problem [24] which implies the same upper bound for Nullstellensatz proofs. Several decades later, W. Dale Brownawell gave an exponential upper bound on the degree required for Nullstellensatz proofs over algebraically closed fields of characterisic zero [11]. A year later, János Kollár showed that this result holds for all algebraically closed fields [27].

For specific problems, the degree of Nullstellensatz proofs can be analyzed using designs [14]. Using designs, Nullstellensatz degree lower bounds have been shown for many problems including the pigeonhole principle, the induction principle, the housesitting principle, and the mod m matching principles [6, 5, 15, 16, 12]. More recent work showed that there is a close connection between Nullstellensatz degree and reversible pebbling games [19] and that lower bounds on Nullstellensatz degree can be lifted to lower bounds on monotone span programs, monotone comparator circuits, and monotone switching networks [28].

[1] Technically, this is the weak form of Hilbert's Nullstellensatz. Hilbert's Nullstellensatz actually says that given polynomials $p_1, \ldots, p_m$ and another polynomial p, if $p(x) = 0$ for all x such that $p_i(x) = 0$ for each $i \in [m]$ then there exists a natural number r such that p^r is in the ideal generated by $p_1, \ldots, p_m$.

For analyzing the size of Nullstellensatz proofs (i.e., the number of monomials in the proof), a powerful technique is the size-degree relation shown by Russell Impagliazzo, Pavel Pudlák, and Jiří Sgall for polynomial calculus [25], which also holds for resolution proofs [7] and for sum of squares proofs with $\{0,1\}$ variables [3]. The size-degree relation says that if there is a size S polynomial calculus proof then there is a polynomial calculus proof of degree $O(\sqrt{n \log S})$. Thus, if we have an $\Omega(n)$ degree lower bound for polynomial calculus, this implies a $2^{\Omega(n)}$ size lower bound for polynomial calculus (which also holds for Nullstellensatz as Nullstellensatz is a weaker proof system). However, the size-degree relation does not give any size lower bound when the degree is $O(\sqrt{n})$, and we know of few other techniques for analyzing the size of Nullstellensatz proofs.

In this paper, instead of investigating the degree or size of Nullstellensatz proofs, we investigate the total coefficient size of Nullstellensatz proofs, i.e., the sum of the magnitudes of the coefficients of the monomials in the proof. Total coefficient size is a reasonably natural measure which is relatively unexplored (though as we discuss below, there has been considerable research on closely related measures such as unary Nullstellensatz size, unary Sherali-Adams size, and the total bit complexity of proofs [2, 1, 9, 20, 31]). There are several reasons why total coefficient size bounds in particular are interesting.

First, analyzing the total coefficient size of proofs may give insight into proof size in settings where we currently cannot prove size lower bounds. If we can prove a large total coefficient size lower bound, this shows that any proof must either have large size or involve large coefficients. Unless there is a reason to suspect that large coefficients are helpful for making the proof shorter, this gives considerable evidence for a lower bound on proof size.

Second, lower bounds on total coefficient size have some direct implications. As observed by [20], a total coefficient size lower bound for the stronger Sherali-Adams proof system implies a lower bound for the reversible resolution proof system which captures the Max-SAT resolution proof system (see [10]) for Max SAT. Similarly, [20] observes that a total coefficient size lower bound for Nullstellensatz implies a lower bound for the reversible resolution with terminals proof system, which is a weaker variant of reversible resolution.

Finally, investigating the total coefficient size of proofs gives insight into the following question. Are there natural examples where having fractional coefficients greatly reduces the total coefficient size needed for Nullstellensatz and/or Sherali-Adams proofs? We note that this question is not addressed by [20]. For example, [20] shows that there are n-variate CNF formulas F such that F can be refuted by constant width resolution proofs but any Sherali-Adams proof of F requires either exponentially many monomials or requires coefficients of exponential size (see Theorem 1 and the last paragraph of Section 1.1 in [20]). However, this does not rule out the existence of a proof where there are exponentially many monomials but the coefficient for each monomial is exponentially small so the total coefficient size is still small.

Proving total coefficient size lower bounds for a problem rules out this possibility. Conversely, if there is a natural example where the minimum proof size is large but the total coefficient size is small, that would be quite interesting.

1.1 Our results

In this paper, we show that the minimum total coefficient size of a Nullstellensatz proof of the pigeonhole principle is $2^{\Theta(n)}$. More precisely, we show the following bounds.

▶ **Theorem 1.** *For all $n \geq 1$, any Nullstellensatz proof of the pigeonhole principle with $n+1$ pigeons and n holes has total coefficient size $\Omega\left(n^{\frac{3}{4}}\left(\frac{2}{\sqrt{e}}\right)^n\right)$.*

We note that this lower bound also holds for the functional pigeonhole principle, where each pigeon must go to exactly one hole (instead of at least one hole).

▶ **Theorem 2.** *For all $n \geq 1$, there is a Nullstellensatz proof of the pigeonhole principle with $n+1$ pigeons and n holes with total coefficient size at most $2^{5(n+1)}$.*

▶ Remark 3. Note that Nullstellensatz size lower bounds do not imply total coefficient size lower bounds, because we could have a proof with many monomials but a small coefficient (absolute value less than 1) on each monomial. Indeed, in Appendix A, we show an example where the minimum total coefficient size of a Nullstellensatz proof is smaller than the minimum size of a Nullstellensatz proof. Thus, the exponential size polynomial calculus lower bounds for the pigeonhole principle from Razborov's $\Omega(n)$ degree lower bound for polynomial calculus [30] and the size-degree relation [25] do not imply total coefficient size lower bounds for the pigeonhole principle.

In addition, we investigate the total coefficient size of Nullstellensatz proofs of the ordering principle in Appendix C. We show the following upper bound by constructing an explicit Nullstellensatz proof.

▶ **Theorem 4.** *For all $n \geq 3$, there is a Nullstellensatz proof of the ordering principle on n elements with size and total coefficient size $2^n - n$. This upper bound is tight for $n \leq 5$.*

In the full version of this paper [29], we also discuss total coefficient size for the Sherali-Adams and sum of squares proof systems. We observe that even though resolution is a dynamic proof system, the $O(n^3)$ size resolution proof of the ordering principle found by Gunnar Stålmark [32] can be captured by a one line sum of squares proof with small size and coefficients.

1.2 Comparison with related work

Like previous resolution, polynomial calculus, and Sherali-Adams lower bounds for the pigeonhole principle (e.g. [22, 30, 18]), our analysis is inspired by the idea that if we only look at a small number of pigeons, we cannot detect a problem. That said, our analysis differs considerably from previous analyses of the pigeonhole principle as we need to bound the value of a linear program by constructing a dual certificate (see Proposition 12). To construct this dual certificate, we need to assign a value to every possible assignment of the variables, so we need to consider all n pigeons at once which requires a different analysis.

In terms of the overall framework, the work which is most similar to ours is that of De Rezende, Potechin, and Risse [31] which shows a total coefficient size Sherali-Adams lower bound for showing that a random graph does not contain a large clique. Like our paper, [31] constructs a dual certificate which assigns a value to every possible assignment of the variables. That said, [31] uses different techniques to construct and analyze their dual certificate. In particular, while the construction in [31] is inspired by the pseudo-calibration technique used to prove SoS lower bounds for planted clique [4] and the analysis heavily uses the fact that the graph is random, our construction and analysis is combinatorial and takes advantage of symmetry.

Another work which is closely related to ours is that of Göös et al. [20]. [20] analyzes the size of unary Nullstellensatz and Sherali-Adams proofs, which is equivalent to analyzing the total coefficient size of Nullstellensatz and Sherali-Adams proofs with the added restriction that all coefficients are integers. The authors show that there are deep connections between unary Nullstellensatz, unary Sherali-Adams, resolution, and total NP search problems (TFNP). In particular, they prove the following results (among others) which show that there are considerable advantages to having the restriction that all coefficients are integers.

1. Resolution is not polynomially simulated by unary Sherali-Adams, and reversible resolution is not polynomially simulated by Nullstellensatz. Since unary Sherali-Adams can simulate reversible resolution, this implies that resolution is not simulated by reversible resolution.
2. Roughly speaking, unary Nullstellensatz corresponds to the TFNP class PPAD which corresponds to the principle that every directed graph with an unbalanced node (i.e., a node whose indegree is not equal to its outdegree) must have another unbalanced node. Similarly, unary Sherali-Adams corresponds to the TFNP class PPADS which corresponds to the principle that every directed graph with a postitively unbalanced node (outdegree exceeds indegree) must have a negatively unbalanced node (indegree exceeds outdegree).
3. There is a reversible resolution refutation of a CNF F if and only if there is both a resolution refutation of F and a unary Sherali-Adams refutation of F. Similarly, there is a reversible resolution with terminals refutation of a CNF F if and only if there is both a resolution refutation of F and a unary Nullstellensatz refutation of F.

In this paper, we show that there are also advantages to allowing fractional coefficients. Proving a total coefficient size lower bound when fractional coefficients are allowed removes the possibility of having a proof with many monomials but a small total coefficient size. In addition, allowing fractional coefficients gives us a linear program for minimum total coefficient size which can be analyzed directly. As a result, while [20] needs several steps to show their separations, we show our bounds directly.

Finally, a natural alternative to analyzing the size or total coefficient size of proofs is to analyze the bit complexity of proofs. One way to prove a lower bound on the bit complexity of a proof is to show an exponentially larger lower bound on the total coefficient size of the proof. Hakoniemi [23] uses this approach to give an example where there is a polynomial size sum of squares proof of degree 2 but every sum of squares proof requires doubly exponential total coefficient size and thus exponential bit complexity.

While it is generally hard to lower bound the bit complexity of a proof without lower bounding the proof size or total coefficient size, this has been done for the binary value principle which says that a number written in binary with no minus sign must be non-negative. More precisely, if $x_1, \ldots, x_n \in \{0, 1\}$ then we cannot have that $1 + x_1 + 2x_2 + \ldots + 2^{n-1}x_n = 0$. By considering the primes $p \in [1, 2^n]$ and showing that the proof must involve a coefficient which is divisible by all such primes, [1] and [2] show bit complexity lower bounds for powerful proof systems, namely polynomial calculus with extensions and the ideal proof system (see [21]) where the latter bound is conditional on the Shub-Smale hypothesis. This technique is powerful but is specialized to this problem and is very different from our techniques.

2 Nullstellensatz total coefficient size

We start by defining total coefficient size for Nullstellensatz proofs and describing a linear program for finding the minimum total coefficient size of a Nullstellensatz proof. In this paper, we only consider problems on Boolean variables, so we give definitions which are specialized for this setting.

▶ **Definition 5.** *For each Boolean variable x_i, we define the twin variable $\bar{x}_i$ to be $\bar{x}_i = 1 - x_i$.*

▶ **Definition 6.** *Given Boolean variables $x_1, \ldots, x_N$, we define a monomial to be a product of the form $\left(\prod_{i \in S} x_i\right)\left(\prod_{j \in T} \bar{x}_j\right)$ for some disjoint subsets S, T of $[N]$.*

▶ **Definition 7.** *Given a polynomial f on Boolean variables $x_1, \dots, x_N$, we define the total coefficient size $T(f)$ of f to be the minimum sum of the magnitudes of coefficients when we decompose f into monomials. For example, if $f(x_1, x_2) = 1 - x_1 - x_2 + 2x_1x_2$, then $T(f) = 2$ as we can write $f = \bar{x}_1\bar{x}_2 + x_1x_2 = (1 - x_1)(1 - x_2) + x_1x_2$.*

We will use the following terminology:

▶ **Definition 8.** *Given a system $\{p_i = 0 : i \in [m]\}$ of polynomial equations, we call each of the p_i an axiom. We say that a polynomial W is a weakening of the axiom p_i if $W = rp_i$ for some monomial r.*

We now define Nullstellensatz proofs and their total coefficient size.

▶ **Definition 9.** *Given a system $\{p_i = 0 : i \in [m]\}$ of polynomial equations on Boolean variables $x_1, \dots, x_N$, a Nullstellensatz proof of infeasibility is an equality of the form*

$$1 = \sum_{i=1}^{m} p_i q_i + \sum_{j=1}^{N} (x_j^2 - x_j) g_j + \sum_{j=1}^{N} (x_j + \bar{x}_j - 1) h_j$$

for some polynomials $\{q_i : i \in [m]\}$, $\{g_j : j \in [N]\}$, and $\{h_j : j \in [N]\}$. We define the total coefficient size of such a Nullstellensatz proof to be $\sum_{i=1}^{m} T(q_i)$.

▶ Remark 10. We do not include the total coefficient size of p_i, g_j, or h_j in the total coefficient size of the proof as we want to focus on the complexity of the proof as opposed to the complexity of the axioms and manipulating the Boolean variables. That said, in this paper we only consider systems of polynomial equations where each p_i is a monomial, so this choice does not matter: in this setting $T(p_i) = 1$ for all i, and it is both possible and optimal to take $g_j = 0$ for all j.[2] In terms of weakenings, in this setting a Nullstellensatz proof is an equality

$$1 = \sum_{W} c_W W,$$

where W ranges over all possible weakenings of axioms and $c_W \in \mathbb{R}$. The total coefficient size of a Nullstellensatz proof is $\sum_W |c_W|$.

The minimum total coefficient size of a Nullstellensatz proof can be found using a linear program. To illustrate this, we now give an example.

▶ **Example 11.** Consider the following system of equations on two variables x_1, x_2:

$$1 - x_1 = 0$$
$$1 - x_2 = 0$$
$$x_1x_2 = 0$$

Given these axioms, the possible weakenings W (modulo the Boolean axioms) are $1 - x_1$, $(1 - x_1)x_2$, $(1 - x_1)(1 - x_2)$, $1 - x_2$, $x_1(1 - x_2)$, and x_1x_2.

[2] In other words, we can assume without loss of generality that all terms in a Nullstellensatz proof have degree at most 1 in each variable. If an axiom contains a variable x_i, there is no point in multiplying the axiom by x_i or $\bar{x}_i = (1 - x_i)$, because $x_i^2 = x_i$ and $x_i(1 - x_i) = 0$ modulo the Boolean axioms. The reasoning is similar in the case that an axiom contains a variable $\bar{x}_i = (1 - x_i)$.

To find a Nullstellensatz proof with minimum total coefficient size, we write a linear program with a variable c_W for each weakening W. We also have a variable b_W for each weakening W representing the absolute value of c_W with the constraints $b_W - c_W \geq 0$ and $b_W + c_W \geq 0$. The objective is to minimize $\sum_W b_W$.

To ensure that $\sum_W c_W W = 1$, we have a constraint for each of the 4 possible assignments of values to the variables. For example, one possible assignment is $x_1 = 0, x_2 = 0$. We ensure that $\sum_W c_W W$ evaluates to 1 on this assignment by having the constraint

$$c_{(1-x_1)} + c_{(1-x_2)} + c_{(1-x_1)(1-x_2)} = 1,$$

because the weakenings $1 - x_1$, $1 - x_2$, and $(1 - x_1)(1 - x_2)$ evaluate to 1 on this assignment while the other weakenings evaluate to 0. The analogous constraints for the other 3 possible assignments of values to the variables are as follows:

1. $c_{(1-x_1)} + c_{(1-x_1)x_2} = 1$ for the assignment $x_1 = 0$, $x_2 = 1$
2. $c_{(1-x_2)} + c_{x_1(1-x_2)} = 1$ for the assignment $x_1 = 1$, $x_2 = 0$
3. $c_{x_1x_2} = 1$ for the assignment $x_1 = 1$, $x_2 = 1$.

The set of optimal solutions to this linear program is

$$\{c_{x_1x_2} = 1, c_{(1-x_1)} = c_{x_1(1-x_2)} = a, c_{(1-x_1)x_2} = c_{(1-x_2)} = 1-a, c_{(1-x_1)(1-x_2)} = 0 : a \in [0,1]\}$$

which corresponds to the equality

$$1 = x_1x_2 + a\left((1 - x_1) + x_1(1 - x_2)\right) + (1 - a)\left((1 - x_1)x_2 + (1 - x_2)\right).$$

In the same way as the above example, we can find the minimum total coefficient size of any system of equations with a linear program. In order to show a lower bound on total coefficient size, we will analyze the dual of this linear program. Because the primal has a constraint for each assignment of values to the variables $x \in \{0,1\}^N$, the dual has a variable for each assignment $x \in \{0,1\}^N$. We will let $D : \{0,1\}^N \to \mathbb{R}$ denote the dual.

We observe that D induces a linear map $\widehat{D}$ from polynomials to $\mathbb{R}$ in a natural way, by taking $\widehat{D}(f) = \sum_{x\in\{0,1\}^N} D(x)f(x)$. It turns out that the dual is equivalent to:

Maximize $\widehat{D}(1)$ subject to the constraint that for each weakening W, $|\widehat{D}(W)| \leq 1$.

Weak duality, which is what we need to prove lower bounds on total coefficient size, can be seen directly as follows.

▶ **Proposition 12.** *If $\widehat{D}$ is a linear map from polynomials to $\mathbb{R}$ such that $|\widehat{D}(W)| \leq 1$ for all weakenings W, then any Nullstellensatz proof has total coefficient size at least $\widehat{D}(1)$.*

Proof. Given a Nullstellensatz proof $1 = \sum_{i=1}^m p_iq_i$, applying $\widehat{D}$ to both sides gives $\widehat{D}(1) = \sum_{i=1}^m \widehat{D}(p_iq_i) \leq \sum_{i=1}^m T(q_i)$. The inequality holds because for any q_i, for any way of writing q_i in terms of monomials r as $q_i = \sum_r c_{ir}r$, we have $\widehat{D}(p_iq_i) = \sum_r c_{ir}\widehat{D}(rp_i) \leq \sum_r |c_{ir}|$ because rp_i is a weakening. ◀

3 Total coefficient size lower bound for the pigeonhole principle

In this section, we prove Theorem 1, our total coefficient size lower bound for the pigeonhole principle. We start by formally defining the pigeonhole principle.

▶ **Definition 13** (pigeonhole principle (PHP_n)). *Intuitively, the pigeonhole principle says that if $n+1$ pigeons are assigned to n holes, then some hole must have more than one pigeon. Formally, for $n \geq 1$, we define PHP_n to be the statement that the following system of axioms is infeasible:*

- *For each $i \in [n+1]$ and $j \in [n]$, we have a variable $x_{i,j}$ and the Boolean axiom $x_{i,j}^2 - x_{i,j} = 0$. $x_{i,j} = 1$ represents pigeon i being in hole j, and $x_{i,j} = 0$ represents pigeon i not being in hole j.*
- *For each $i \in [n+1]$, we have the axiom $\prod_{j=1}^{n} \bar{x}_{i,j} = 0$ representing the constraint that each pigeon must be in at least one hole.*
- *For each pair of distinct pigeons $i_1, i_2 \in [n+1]$ and each hole $j \in [n]$, we have the axiom $x_{i_1,j}x_{i_2,j} = 0$ representing the constraint that pigeons i_1 and i_2 cannot both be in hole j.*

We prove our lower bound on total coefficient size for PHP_n by constructing and analyzing a dual solution $D : \{0,1\}^{(n+1)n} \to \mathbb{R}$. In our dual solution, the only assignments of values to the variables $x \in \{0,1\}^{(n+1)n}$ for which $D(x) \neq 0$ are those where each pigeon goes to exactly one hole, i.e., for each pigeon i, exactly one of the $x_{i,j}$ is 1. As a result, Theorem 1 also applies to the functional pigeonhole principle. Note that there are n^{n+1} such assignments. In the rest of this section, when we refer to assignments or write a summation or expectation over assignments x, we refer specifically to these n^{n+1} assignments.

Recall that the dual constraints are

$$\widehat{D}(W) = \sum_{x \in \{0,1\}^N} D(x)W(x) \in [-1,1]$$

for all weakenings W. Note that since $D(x)$ is only nonzero for assignments x where each pigeon goes to exactly one hole, for any weakening W of an axiom of the form $\prod_{j=1}^{n} \bar{x}_{i,j} = 0$, we have $\widehat{D}(W) = 0$. Thus, it is sufficient to consider weakenings W of the axioms $x_{i_1,j}x_{i_2,j} = 0$.

For simplicity, in order to construct a dual solution, we first ignore the constraints $|\widehat{D}(W)| \leq 1$. Then, we obtain a dual solution by normalizing D, i.e., dividing D by $\max_W |\widehat{D}(W)|$. Thus, we can rewrite the objective value of the dual program as $\frac{\widehat{D}(1)}{\max_W |\widehat{D}(W)|}$. Letting $\mathbb{E}$ denote the expectation over a uniform assignment where each pigeon goes to exactly one hole, $\frac{\widehat{D}(1)}{\max_W |\widehat{D}(W)|} = \frac{\mathbb{E}(D)}{\max_W |\mathbb{E}(DW)|}$. Thus, it is sufficient to construct D and analyze $\mathbb{E}(D)$ and $\max_W |\mathbb{E}(DW)|$.

Before constructing and analyzing D, we provide some intuition for our construction. The idea is that if we consider a subset of n pigeons then D should behave like the indicator function for whether those n pigeons all go to different holes. More concretely, for any polynomial p which does not depend on some pigeon i (i.e., p does not contain $x_{i,j}$ or $\bar{x}_{i,j}$ for any $j \in [n]$), we want

$$\mathbb{E}(Dp) = \frac{n!}{n^n}\mathbb{E}(p \mid \text{all pigeons in } [n+1] \setminus \{i\} \text{ go to different holes})$$

Given this intuition, we now present our construction. Our dual solution D will be a linear combination of the following functions:

▶ **Definition 14** (functions J_S). *For each subset of pigeons $S \subsetneq [n+1]$ of size at most n, we define the function J_S that maps assignments to $\{0,1\}$ so that for each assignment x, $J_S(x) = 1$ if all pigeons in S are in different holes according to x and $J_S(x) = 0$ otherwise.*

Note that if $|S| = 0$ or $|S| = 1$, then J_S is the constant function 1. In general, the expectation of J_S over a uniform assignment is $\mathbb{E}(J_S) = \left(\prod_{k=1}^{|S|}(n+1-k)\right)/n^{|S|}$.

▶ **Definition 15** (dual solution D)**.** *Our dual solution D is:*

$$D = \sum_{S \subsetneq [n+1]} c_S J_S,$$

where the coefficients c_S are $c_S = \frac{(-1)^{n-|S|}(n-|S|)!}{n^{n-|S|}}$.

We will lower-bound the dual value $\mathbb{E}(D)/\max_W |\mathbb{E}(DW)|$ by computing $\mathbb{E}(D)$ and then upper-bounding $\max_W |\mathbb{E}(DW)|$. In both calculations, we will use the following key property of D which we introduced in our intuition for the construction:

▶ **Lemma 16.** *If p is a polynomial which does not depend on pigeon i (i.e., p does not contain any variables of the form $x_{i,j}$ or $\bar{x}_{i,j}$), then $\mathbb{E}(Dp) = \mathbb{E}(J_{[n+1]\setminus\{i\}}p)$.*

Proof. Without loss of generality, suppose p does not contain any variables of the form $x_{1,j}$ or $\bar{x}_{1,j}$. Let T be any subset of pigeons that does not contain pigeon 1 and that has size at most $n-1$. Observe that

$$\mathbb{E}(J_{T\cup\{1\}}p) = \frac{n-|T|}{n}\mathbb{E}(J_T p)$$

because when the pigeons in T go to different holes, the probability that pigeon 1 goes to a different hole is $\frac{n-|T|}{n}$, and p does not depend on the location of pigeon 1. Since

$$\begin{aligned} c_{T\cup\{1\}} &= \frac{(-1)^{n-1-|T|}(n-1-|T|)!}{n^{n-1-|T|}} \\ &= -\frac{n}{n-|T|}\cdot\frac{(-1)^{n-|T|}(n-|T|)!}{n^{n-|T|}} = -\frac{n}{n-|T|}c_T \end{aligned}$$

we have that for all $T \subsetneq \{2,\ldots,n+1\}$, $\mathbb{E}(c_{T\cup\{1\}}J_{T\cup\{1\}}p) + \mathbb{E}(c_T J_T p) = 0$. Thus, all terms in the sum $\mathbb{E}(Dp) = \sum_{S\subsetneq[n+1]}\mathbb{E}(c_S J_S p)$ cancel, except $J_{\{2,3,\ldots,n+1\}}$. Since $c_{\{2,3,\ldots,n+1\}} = 1$, we have that $\mathbb{E}(Dp) = \mathbb{E}(J_{\{2,3,\ldots,n+1\}}p)$, as needed. ◀

The value of $\mathbb{E}(D)$ follows immediately:

▶ **Corollary 17.**

$$\mathbb{E}(D) = \frac{n!}{n^n}.$$

Proof. Let $p = 1$. By Lemma 16, $\mathbb{E}(D) = \mathbb{E}(J_{\{2,\ldots,n+1\}}) = \frac{n!}{n^n}$. ◀

3.1 Upper bound on $\max_W |\mathbb{E}(DW)|$

We now upper bound $\max_W |\mathbb{E}(DW)|$. To do this, we introduce the following notation:

▶ **Definition 18** ($H_{W,i}$)**.** *Given a weakening W, we define a set of holes $H_{W,i} \subseteq [n]$ for each pigeon $i \in [n+1]$ so that $W(x) = 1$ if and only if each pigeon $i \in [n+1]$ is mapped to one of the holes in $H_{W,i}$. More precisely,*

- *If W contains terms x_{i,j_1} and x_{i,j_2} for distinct holes j_1, j_2, then $H_{W,i} = \emptyset$ (i.e., it is impossible that $W(x) = 1$ because pigeon i cannot go to both holes j_1 and j_2).*
- *If W contains exactly one term of the form $x_{i,j}$, then $H_{W,i} = \{j\}$. (i.e., for all x such that $W(x) = 1$, pigeon i goes to hole j).*

- *If W contains no terms of the form $x_{i,j}$, then $H_{W,i}$ is the subset of holes j such that W does not contain the term $\bar{x}_{i,j}$. (i.e., if W contains the term $\bar{x}_{i,j}$, then for all x such that $W(x) = 1$, pigeon i does not go to hole j.)*

The key property we will use to bound $\max_W |\mathbb{E}(DW)|$ follows immediately from Lemma 16:

▶ **Lemma 19.** *Let W be a weakening. If there exists some pigeon $i \in [n+1]$ such that $H_{W,i} = [n]$ (i.e., W does not contain any terms of the form $x_{i,j}$ or $\bar{x}_{i,j}$), then $\mathbb{E}(DW) = 0$.*

Proof. Without loss of generality, suppose W is a weakening of the axiom $x_{2,1}x_{3,1} = 0$ and $H_{W,1} = [n]$. By Lemma 16, $\mathbb{E}(DW) = \mathbb{E}(J_{\{2,\dots,n+1\}}W)$. However, $\mathbb{E}(J_{\{2,\dots,n+1\}}W) = 0$ because if $W(x) = 1$, then pigeons 2 and 3 must both go to hole 1. ◀

We now make the following definition and then state a corollary of Lemma 19.

▶ **Definition 20** (W_S^{flip})**.** *Let W be a weakening of the axiom $x_{i_1,j}x_{i_2,j} = 0$ for pigeons i_1, i_2 and hole j. Let $S \subseteq [n+1] \setminus \{i_1, i_2\}$. We define W_S^{flip}, which is also a weakening of the axiom $x_{i_1,j}x_{i_2,j} = 0$, as follows.*

- *For each pigeon $i_3 \in S$, we define W_S^{flip} so that $H_{W_S^{\text{flip}},i_3} = [n] \setminus H_{W,i_3}$.*
- *For each pigeon $i_3 \notin S$, we define W_S^{flip} so that $H_{W_S^{\text{flip}},i_3} = H_{W,i_3}$.*

Note: Technically, there are multiple possible weakenings W_S^{flip} which satisfy these properties (e.g. if $n = 2$, $W = x_{1,1}x_{2,1}x_{3,1}$, and $S = \{3\}$, then W_S^{flip} can be $x_{1,1}x_{2,1}\bar{x}_{3,1}$ or $x_{1,1}x_{2,1}x_{3,2}$ or even $x_{1,1}x_{2,1}\bar{x}_{3,1}x_{3,2}$, among others). We arbitrarily choose any such weakening W_S^{flip}.

In other words, W_S^{flip} is obtained from W by flipping the sets of holes that the pigeons in S can go to in order to make the weakening evaluate to 1.

▶ **Corollary 21.** *Let W be a weakening of the axiom $x_{i_1,j}x_{i_2,j} = 0$ for pigeons i_1, i_2 and hole j. Let $S \subseteq [n+1] \setminus \{i_1, i_2\}$. Then*

$$\mathbb{E}\left(DW_S^{\text{flip}}\right) = (-1)^{|S|} \cdot \mathbb{E}(DW).$$

Proof. It suffices to show that for $i_3 \in [n+1] \setminus \{i_1, i_2\}$, we have $\mathbb{E}\left(DW_{\{i_3\}}^{\text{flip}}\right) = -\mathbb{E}(DW)$. Indeed, let W' be a weakening such that $W'(x) = W(x) + W_{\{i_3\}}^{\text{flip}}(x)$ for all assignments x where each pigeon goes to exactly one hole. (For example, if $n = 2$, $W = x_{1,1}x_{2,1}x_{3,1}$, and $i_3 = 3$, then we can take $W_{\{3\}}^{\text{flip}}$ to be $x_{1,1}x_{2,1}x_{3,2}$, in which case $W' = x_{1,1}x_{2,1}$.) Then $\mathbb{E}(DW') = 0$ by Lemma 19 because $H_{W',i_3} = [n]$, so $\mathbb{E}\left(DW_{\{i_3\}}^{\text{flip}}\right) = -\mathbb{E}(DW)$. ◀

Using Corollary 21, we can bound $\max_W |\mathbb{E}(DW)|$ using Cauchy-Schwarz. We first show an approach that does not give a strong enough bound. We then show how to modify the approach to achieve a better bound.

▶ **Definition 22.** *Given functions F, G on the assignments mapping each pigeon to exactly one hole, we define $\langle F, G\rangle = \mathbb{E}(FG)$. We define $\|F\| = \sqrt{\langle F, F\rangle} = \sqrt{\mathbb{E}(F^2)}$.*

3.1.1 Unsuccessful approach to upper bound $\max_W |\mathbb{E}(DW)|$

Consider $\max_W |\mathbb{E}(DW)|$. By Lemma 21, it suffices to take the max over weakenings W such that, if W is a weakening of the axiom $x_{i_1,j}x_{i_2,j} = 0$, then for all pigeons $i_3 \in [n+1] \setminus \{i_1, i_2\}$, we have $|H_{W,i_3}| \leq \lfloor n/2 \rfloor$ (because if $|H_{W,i_3}| > \lfloor n/2 \rfloor$, we can flip H_{W,i_3} without changing $|\mathbb{E}(DW)|$). For any such W, we have

$$\|W\| = \sqrt{\mathbb{E}(W^2)} \leq \sqrt{\left(\frac{1}{n}\right)^2 \left(\frac{1}{2}\right)^{n-1}} = n^{-1}2^{-(n-1)/2}.$$

By Cauchy-Schwarz,

$$\begin{aligned} |\mathbb{E}(DW)| &\leq \|D\|\|W\| \\ &\leq \|D\|n^{-1}2^{-(n-1)/2}. \end{aligned}$$

Using the value of $\mathbb{E}(D)$ from Corollary 17, the dual value $\mathbb{E}(D)/\max_W |\mathbb{E}(DW)|$ is at least

$$\frac{n!}{n^n} \cdot \frac{n2^{(n-1)/2}}{\|D\|} = \widetilde{\Theta}\left(\left(\frac{e}{\sqrt{2}}\right)^{-n} \cdot \frac{1}{\|D\|}\right)$$

by Stirling's formula. Thus, in order to achieve an exponential lower bound on the dual value, we would need $1/\|D\| \geq \Omega(c^n)$ for some $c > e/\sqrt{2}$. However, this requirement is too strong, as we will show in Lemma 26 that $1/\|D\| = \widetilde{\Theta}\left((\sqrt{e})^n\right)$. Directly applying Cauchy-Schwarz results in too loose of a bound on $\max_W |\mathbb{E}(DW)|$, so we now modify our approach.

3.1.2 Successful approach to upper bound $\max_W |\mathbb{E}(DW)|$

▶ **Definition 23** ($W_{i_1,i_2}^{\{-1,0,1\}}$). *Let W be a weakening of the axiom $x_{i_1,j}x_{i_2,j} = 0$ for pigeons i_1, i_2 and hole j. We define a function $W_{i_1,i_2}^{\{-1,0,1\}}$ that maps assignments to $\{-1, 0, 1\}$. For an assignment x,*

- *If pigeons i_1 and i_2 do not both go to hole j, then $W_{i_1,i_2}^{\{-1,0,1\}}(x) = 0$.*
- *Otherwise, let $V(x) = |\{i_3 \in [n+1] \setminus \{i_1, i_2\} : \text{pigeon } i_3 \text{ does not go to } H_{W,i_3}\}|$. Then $W_{i_1,i_2}^{\{-1,0,1\}}(x) = (-1)^{V(x)}$.*

Note that $W_{i_1,i_2}^{\{-1,0,1\}}$ is a linear combination of the W_S^{flip}:

▶ **Lemma 24.** *Let W be a weakening of the axiom $x_{i_1,j}x_{i_2,j} = 0$ for pigeons i_1, i_2 and hole j. We have:*

$$W_{i_1,i_2}^{\{-1,0,1\}} = \sum_{S \subseteq [n+1] \setminus \{i_1,i_2\}} (-1)^{|S|} \cdot W_S^{\text{flip}}.$$

It follows that:

$$\mathbb{E}\left(DW_{i_1,i_2}^{\{-1,0,1\}}\right) = 2^{n-1} \cdot \mathbb{E}(DW).$$

Proof. To prove the first equation, consider any assignment x. If pigeons i_1 and i_2 do not both go to hole j, then both $W_{i_1,i_2}^{\{-1,0,1\}}$ and all the W_S^{flip} evaluate to 0 on x. Otherwise, exactly one of the $W_S^{\text{flip}}(x)$ equals 1, and for this choice of S we have $W_{i_1,i_2}^{\{-1,0,1\}}(x) = (-1)^{|S|}$.

The second equation follows because:

$$\begin{aligned}\mathbb{E}\left(DW_{i_1,i_2}^{\{-1,0,1\}}\right) &= \sum_{S\subseteq[n+1]\setminus\{i_1,i_2\}} (-1)^{|S|}\cdot\mathbb{E}\left(DW_S^{\text{flip}}\right)\\ &= \sum_{S\subseteq[n+1]\setminus\{i_1,i_2\}} (-1)^{|S|}(-1)^{|S|}\cdot\mathbb{E}(DW) && \text{(Corollary 21)}\\ &= 2^{n-1}\cdot\mathbb{E}(DW).\end{aligned}$$

◀

Using Lemma 24, we now improve on the approach to upper-bound $\max_W |\mathbb{E}(DW)|$ from section 3.1.1:

▶ **Lemma 25.** *The dual value* $\mathbb{E}(D)/\max_W|\mathbb{E}(DW)|$ *is at least* $\frac{n!}{n^n}\cdot\frac{n2^{n-1}}{\|D\|}$.

Proof. If W is a weakening of the axiom $x_{i_1,j}x_{i_2,j}=0$ for pigeons i_1,i_2 and hole j,

$$\begin{aligned}|\mathbb{E}(DW)| &= 2^{-(n-1)}\cdot\left|\mathbb{E}\left(DW_{i_1,i_2}^{\{-1,0,1\}}\right)\right| && \text{(Lemma 24)}\\ &\le 2^{-(n-1)}\cdot\|D\|\|W_{i_1,i_2}^{\{-1,0,1\}}\| && \text{(Cauchy-Schwarz)}\\ &= 2^{-(n-1)}\cdot\|D\|\sqrt{\mathbb{E}\left(\left(W_{i_1,i_2}^{\{-1,0,1\}}\right)^2\right)}\\ &= n^{-1}2^{-(n-1)}\cdot\|D\|.\end{aligned}$$

Using the value of $\mathbb{E}(D)$ from Corollary 17, the dual value $\mathbb{E}(D)/\max_W|\mathbb{E}(DW)|$ is at least $\frac{n!}{n^n}\cdot\frac{n2^{n-1}}{\|D\|}$. ◀

It only remains to compute $\|D\|$:

▶ **Lemma 26.**

$$\|D\|^2 = \frac{n!}{n^n}\cdot(n+1)!\cdot\sum_{c=0}^{n}\frac{(-1)^{n-c}}{n+1-c}\cdot\frac{1}{n^{n-c}c!}$$

Proof. Recall the definition of D (Definition 15):

$$\begin{aligned}D &= \sum_{S\subsetneq[n+1]} c_S J_S,\\ c_S &= \frac{(-1)^{n-|S|}(n-|S|)!}{n^{n-|S|}}.\end{aligned}$$

We compute $\|D\|^2=\mathbb{E}(D^2)$ as follows.

$$\mathbb{E}(D^2) = \sum_{S\subsetneq[n+1]}\sum_{T\subsetneq[n+1]} c_S c_T\mathbb{E}(J_S J_T).$$

Given $S,T\subsetneq[n+1]$, we have:

$$\begin{aligned}\mathbb{E}(J_S J_T) &= \mathbb{E}(J_S)\mathbb{E}(J_T\mid J_S=1)\\ &= \left(\left(\prod_{i=1}^{|S|}(n+1-i)\right)/n^{|S|}\right)\left(\left(\prod_{j=|S\cap T|+1}^{|T|}(n+1-j)\right)/n^{|T\setminus S|}\right).\end{aligned}$$

Therefore,

$$c_S c_T \mathbb{E}(J_S J_T) = \left(c_S \left(\prod_{i=1}^{|S|} (n+1-i) \right) / n^{|S|} \right) \left(c_T \left(\prod_{j=|S\cap T|+1}^{|T|} (n+1-j) \right) / n^{|T\setminus S|} \right).$$

Note that the product of $(-1)^{n-|S|}$ (from the c_S) and $(-1)^{n-|T|}$ (from the c_T) is $(-1)^{-|S|-|T|} = (-1)^{|S|-|T|}$, so the above equation becomes:

$$c_S c_T \mathbb{E}(J_S J_T) = (-1)^{|S|-|T|} \left(\frac{n!}{n^n} \right) \left(\frac{(n-|S\cap T|)!}{n^{n-|S\cap T|}} \right).$$

Now, we rearrange the sum for $\mathbb{E}(D^2)$ in the following way:

$$\begin{aligned} \mathbb{E}(D^2) &= \sum_{S\subsetneq[n+1]} \sum_{T\subsetneq[n+1]} c_S c_T \mathbb{E}(J_S J_T) \\ &= \frac{n!}{n^n} \sum_{c=0}^{n} \frac{(n-c)!}{n^{n-c}} \sum_{\substack{S,T\subsetneq[n+1],\\ |S\cap T|=c}} (-1)^{|S|-|T|}. \end{aligned}$$

To evaluate this expression, fix $c \le n$ and consider the inner sum. Consider the collection of tuples $\{(S,T) \mid S,T \subsetneq [n+1], |S\cap T| = c\}$. We can pair up most of these tuples in the following way. For each S, let m_S denote the minimum element in $[n+1]$ that is not in S (note that m_S is well defined because S cannot be $[n+1]$). We pair up the tuple (S,T) with the tuple $(S, T\triangle\{m_S\})$, where $\triangle$ denotes symmetric difference. The only tuples (S,T) that cannot be paired up in this way are those where $|S| = c$ and $T = [n+1]\setminus\{m_S\}$, because T cannot be $[n+1]$. There are $\binom{n+1}{c}$ unpaired tuples (S,T), and for each of these tuples, we have $(-1)^{|S|-|T|} = (-1)^{n-c}$. On the other hand, each pair $(S,T), (S, T\triangle\{m_S\})$ contributes 0 to the inner sum. Therefore, the inner sum equals $(-1)^{n-c}\binom{n+1}{c}$, and we have:

$$\begin{aligned} \mathbb{E}(D^2) &= \frac{n!}{n^n} \sum_{c=0}^{n} \frac{(-1)^{n-c}(n-c)!}{n^{n-c}} \binom{n+1}{c} \\ &= \frac{n!}{n^n} \sum_{c=0}^{n} \frac{(-1)^{n-c}(n-c)!}{n^{n-c}} \cdot \frac{(n+1)!}{c!(n+1-c)!} \\ &= \frac{n!}{n^n} \cdot (n+1)! \cdot \sum_{c=0}^{n} \frac{(-1)^{n-c}}{n+1-c} \cdot \frac{1}{n^{n-c}c!}. \end{aligned}$$

◀

▶ **Corollary 27.** $\mathbb{E}(D^2) \le \frac{(n+1)!}{n^n}$

Proof. Observe that the sum $\sum_{c=0}^{n} \frac{(-1)^{n-c}}{n+1-c} \cdot \frac{1}{n^{n-c}c!}$ is an alternating series where the magnitudes of the terms decrease as c decreases. The two largest magnitude terms are $\frac{1}{n!}$ and $-\frac{1}{2}\cdot\frac{1}{n!}$. Therefore, the sum is at most $\frac{1}{n!}$, and we conclude that $\mathbb{E}(D^2) \le \frac{n!}{n^n} \cdot \frac{(n+1)!}{n!} = \frac{(n+1)!}{n^n}$, as needed. ◀

We can now complete the proof of Theorem 1.

Proof of Theorem 1. By Lemma 25, any Nullstellensatz proof for PHP_n has total coefficient size at least $\frac{n!}{n^n} \cdot \frac{n2^{n-1}}{\|D\|}$. By Corollary 27, $\|D\| \le \sqrt{\frac{(n+1)!}{n^n}}$. Combining these results, any Nullstellensatz proof for PHP_n has total coefficient size at least

$$\frac{n!}{n^n} \cdot \frac{n2^{n-1}}{\sqrt{\frac{(n+1)!}{n^n}}} = \frac{n2^{n-1}}{\sqrt{(n+1)}} \cdot \frac{\sqrt{n!}}{n^{\frac{n}{2}}}$$
$$= \frac{n2^{n-1}}{\sqrt{n+1}} \sqrt{\frac{n!}{n^n}}$$

Using Stirling's approximation that $n!$ is approximately $\sqrt{2\pi n}\left(\frac{n}{e}\right)^n$, $\sqrt{\frac{n!}{n^n}}$ is approximately $\sqrt[4]{2\pi n}\left(\frac{1}{\sqrt{e}}\right)^n$, and this expression is $\Omega\left(n^{\frac{3}{4}}\left(\frac{2}{\sqrt{e}}\right)^n\right)$, as needed. ◀

4 Open problems

Our work raises a number of open problems. First, while we showed that the minimum total coefficient size of a Nullstellensatz proof of the pigeonhole principle on $n+1$ pigeons and n holes is $2^{\Theta(n)}$, it is natural to ask what happens when we increase the number of pigeons.

1. If we increase the number of pigeons from $n+1$ to $n+2$ while still having n holes, our lower bound proof no longer applies. Can we prove a total coefficient size lower bound on Nullstellensatz when there are m pigeons where $m \geq n+2$? More ambitiously, how does the minimum total coefficient size of a proof depend on m and whether or not we add the axioms that pigeons can only go to one hole (i.e., considering the functional pigeonhole principle rather than the pigeonhole principle)?

Second, we are still far from understanding the total coefficient size of Nullstellensatz proofs of the ordering principle. In Appendix C we construct an explicit Nullstellensatz proof for the ordering principle on n elements with total coefficient size $2^n - n$, but we have no non-trivial lower bounds.

2. Can we prove superpolynomial lower bounds on the total coefficient size of Nullstellensatz proofs of the ordering principle and/or improve the $O(2^n)$ upper bound?

In the full version of this paper [29], we also discuss total coefficient size for the Sherali-Adams and sum of squares proof systems. Some questions regarding these related proof systems are:

3. Are there Sherali-Adams proofs for the ordering principle with polynomial total coefficient size? If so, this shows that the seemingly dynamic $O(n^3)$ size resolution proof of the ordering principle [32] can be captured by a one line Sherali-Adams proof. If not, this gives a natural example separating resolution proof size and the total coefficient size of Sherali-Adams proofs. We note that this separation has been shown by [20] for unary Sherali-Adams using pebbling principles.
4. Are there natural examples where the minimum total coefficient size is very different (either larger or smaller) than the minimum size for Nullstellensatz, Sherali-Adams, or sum of squares proofs?
5. Can the minimum total coefficient size of a strong proof system be used to lower bound the size of another proof system? For example, can resolution proof size be lower bounded by the minimum total coefficient size of a sum of squares proof, or can we find an example where there is a polynomial size resolution proof but any sum of squares proof has superpolynomial total coefficient size?

References

1. Yaroslav Alekseev. A Lower Bound for Polynomial Calculus with Extension Rule. In *Proceedings of the 36th Computational Complexity Conference*, CCC '21, Dagstuhl, DEU, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CCC.2021.21.
2. Yaroslav Alekseev, Dima Grigoriev, Edward A. Hirsch, and Iddo Tzameret. Semi-algebraic proofs, IPS lower bounds, and the tau-conjecture: can a natural number be negative? In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 54–67, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3357713.3384245.
3. Albert Atserias and Tuomas Hakoniemi. Size-Degree Trade-Offs for Sums-of-Squares and Positivstellensatz Proofs. In *Proceedings of the 34th Computational Complexity Conference*, CCC '19, Dagstuhl, DEU, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CCC.2019.24.
4. Boaz Barak, Samuel Hopkins, Jonathan Kelner, Pravesh K Kothari, Ankur Moitra, and Aaron Potechin. A Nearly Tight Sum-of-Squares Lower Bound for the Planted Clique Problem. *SIAM Journal on Computing*, 48(2):687–735, 2019.
5. P. Beame, R. Impagliazzo, J. Krajicek, T. Pitassi, and P. Pudlak. Lower Bounds on Hilbert's Nullstellensatz and Propositional Proofs. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 794–806, 1994. doi:10.1109/SFCS.1994.365714.
6. Paul Beame, Stephen Cook, Jeff Edmonds, Russell Impagliazzo, and Toniann Pitassi. The Relative Complexity of NP Search Problems. *J. Comput. Syst. Sci.*, 57(1):3–19, August 1998. doi:10.1006/jcss.1998.1575.
7. Eli Ben-Sasson and Avi Wigderson. Short Proofs are Narrow—Resolution made Simple. *J. ACM*, 48(2):149–169, March 2001. doi:10.1145/375827.375835.
8. Olaf Beyersdorff, Nicola Galesi, and Massimo Lauria. A lower bound for the pigeonhole principle in tree-like resolution by asymmetric Prover–Delayer games. *Information Processing Letters*, 110(23):1074–1077, 2010. doi:10.1016/j.ipl.2010.09.007.
9. Ilario Bonacina and Maria Luisa Bonet. On the Strength of Sherali-Adams and Nullstellensatz as Propositional Proof Systems. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '22, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3531130.3533344.
10. María Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for Max-SAT. *Artificial Intelligence*, 171(8):606–618, 2007. doi:10.1016/j.artint.2007.03.001.
11. W. Dale Brownawell. Bounds for the Degrees in the Nullstellensatz. *Annals of Mathematics*, 126(3):577–591, 1987. URL: http://www.jstor.org/stable/1971361.
12. S. Buss, R. Impagliazzo, J. Krajíček, P. Pudlák, A. A. Razborov, and J. Sgall. Proof Complexity in Algebraic Systems and Bounded Depth Frege Systems with Modular Counting. *Comput. Complex.*, 6(3):256–298, December 1997. doi:10.1007/BF01294258.
13. Sam Buss and Toniann Pitassi. Resolution and the weak pigeonhole principle. In Mogens Nielsen and Wolfgang Thomas, editors, *Computer Science Logic*, pages 149–156, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.
14. Samuel R. Buss. Lower Bounds on Nullstellensatz Proofs via Designs. In Paul Beame and Samuel R. Buss, editors, *Proof Complexity and Feasible Arithmetics, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, April 21-24, 1996*, volume 39 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 59–71. DIMACS/AMS, 1996. doi:10.1090/dimacs/039/04.
15. S.R. Buss and T. Pitassi. Good Degree Bounds on Nullstellensatz Refutations of the Induction Principle. In *Proceedings of Computational Complexity (Formerly Structure in Complexity Theory)*, pages 233–242, 1996. doi:10.1109/CCC.1996.507685.
16. Matthew Clegg, Jeffery Edmonds, and Russell Impagliazzo. Using the Groebner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 174–183, New York, NY, USA, 1996. Association for Computing Machinery. doi:10.1145/237814.237860.

17 S. Dantchev and S. Riis. Tree resolution proofs of the weak pigeon-hole principle. In *Proceedings 16th Annual IEEE Conference on Computational Complexity*, pages 69–75, 2001. doi:10.1109/CCC.2001.933873.

18 Stefan Dantchev, Barnaby Martin, and Mark Rhodes. Tight rank lower bounds for the Sherali–Adams proof system. *Theoretical Computer Science*, 410(21):2054–2063, 2009. doi:10.1016/j.tcs.2009.01.002.

19 Susanna F. de Rezende, Jakob Nordström, Or Meir, and Robert Robere. Nullstellensatz Size-Degree Trade-offs from Reversible Pebbling. In Amir Shpilka, editor, *34th Computational Complexity Conference (CCC 2019)*, volume 137 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CCC.2019.18.

20 M. Goos, A. Hollender, S. Jain, G. Maystre, W. Pires, R. Robere, and R. Tao. Separations in Proof Complexity and TFNP. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1150–1161, Los Alamitos, CA, USA, November 2022. IEEE Computer Society. doi:10.1109/FOCS54457.2022.00111.

21 Joshua A. Grochow and Toniann Pitassi. Circuit Complexity, Proof Complexity, and Polynomial Identity Testing: The Ideal Proof System. *J. ACM*, 65(6), November 2018. doi:10.1145/3230742.

22 Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985. Third Conference on Foundations of Software Technology and Theoretical Computer Science. doi:10.1016/0304-3975(85)90144-6.

23 Tuomas Hakoniemi. Monomial size vs. Bit-Complexity in Sums-of-Squares and Polynomial Calculus. In *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '21, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1109/LICS52264.2021.9470545.

24 Grete Hermann. Die Frage der endlich vielen Schritte in der Theorie der Polynomideale. *Mathematische Annalen*, 95:736–788, 1926.

25 Russell Impagliazzo, Pavel Pudlák, and Jiří Sgall. Lower Bounds for the Polynomial Calculus and the Gröbner Basis Algorithm. *Comput. Complex.*, 8(2):127–144, November 1999. doi:10.1007/s000370050024.

26 Kazuo Iwama and S. Miyazaki. Tree-Like Resolution is Superpolynomially Slower than DAG-Like Resolution for the Pigeonhole Principle. In *International Symposium on Algorithms and Computation*, 1999.

27 János Kollár. Sharp Effective Nullstellensatz. *Journal of the American Mathematical Society*, pages 963–975, 1988.

28 Toniann Pitassi and Robert Robere. Lifting Nullstellensatz to Monotone Span Programs over Any Field. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 1207–1219, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3188745.3188914.

29 Aaron Potechin and Aaron Zhang. Bounds on the Total Coefficient Size of Nullstellensatz Proofs of the Pigeonhole Principle and the Ordering Principle, 2022. arXiv:2205.03577.

30 Alexander A Razborov. Lower bounds for the polynomial calculus. *computational complexity*, 7(4):291–324, 1998.

31 S. F. De Rezende, A. Potechin, and K. Risse. Clique is Hard on Average for Unary Sherali-Adams. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 12–25, Los Alamitos, CA, USA, November 2023. IEEE Computer Society. doi:10.1109/FOCS57990.2023.00008.

32 Gunnar Stålmarck. Short resolution proofs for a sequence of tricky formulas. *Acta Informatica*, 33(3):277–280, 1996.

A Nullstellensatz total coefficient size can be smaller than size

The following example shows that Nullstellensatz total coefficient size can be smaller than Nullstellensatz proof size. (See Section 2 for the definition of total coefficient size.)

The idea behind our example is as follows. If we have three points p_1, p_2, p_3 and three polynomials f_1, f_2, f_3 such that

1. $f_1(p_1) = 1$, $f_1(p_2) = 1$, $f_1(p_3) = 0$
2. $f_2(p_1) = 1$, $f_2(p_2) = 0$, $f_2(p_3) = 1$
3. $f_3(p_1) = 0$, $f_3(p_2) = 1$, $f_3(p_3) = 1$

then given the axioms $f_1 = 0$, $f_2 = 0$, and $f_3 = 0$, the equality $\frac{1}{2}f_1 + \frac{1}{2}f_2 + \frac{1}{2}f_3 = 1$ is a Nullstellensatz proof of infeasibility which has total coefficient size $\frac{3}{2}$. However, if we want to use integer coefficients then we need coefficient size 2 as we need two of f_1, f_2, and f_3 in order to cover the three points p_1, p_2, p_3.

Our actual example is as follows. We have variables $x_1, x_2, x_3, x_4, x_5, x_6$ and we have the following axioms:

1. For all $I \subseteq \{4,5,6\}$, $x_1 \left(\prod_{i\in I} x_i\right)\left(\prod_{j\in\{4,5,6\}\setminus I} \bar{x}_j\right) = 0$
2. For all $I \subseteq \{4,5,6\}$, $x_2 \left(\prod_{i\in I} x_i\right)\left(\prod_{j\in\{4,5,6\}\setminus I} \bar{x}_j\right) = 0$
3. For all $I \subseteq \{4,5,6\}$, $x_3 \left(\prod_{i\in I} x_i\right)\left(\prod_{j\in\{4,5,6\}\setminus I} \bar{x}_j\right) = 0$
4. $x_1x_2x_3 = 0$
5. $\bar{x}_1\bar{x}_2 = 0$, $\bar{x}_1\bar{x}_3 = 0$, $\bar{x}_2\bar{x}_3 = 0$

We now observe that $1 = \frac{1}{2}\bar{x}_1\bar{x}_2 + \frac{1}{2}\bar{x}_1\bar{x}_3 + \frac{1}{2}x_1\bar{x}_2\bar{x}_3 + \frac{1}{2}(x_1 + x_2 + x_3) - \frac{1}{2}x_1x_2x_3$. We can show this by checking that the right hand side is 1 for all $(x_1, x_2, x_3) \in \{0,1\}^3$.

1. If $x_1 = x_2 = x_3 = 0$ then the first two terms are $\frac{1}{2}$ and the remaining terms are 0.
2. If $x_1 + x_2 + x_3 = 1$ then the fourth term and exactly one of the first three terms are $\frac{1}{2}$ and the remaining terms are 0.
3. If $x_1 + x_2 + x_3 = 2$ then the fourth term is 1 and the remaining terms are 0.
4. If $x_1 = x_2 = x_3 = 1$ then the fourth term is $\frac{3}{2}$, the fifth term is $-\frac{1}{2}$, and the remaining terms are 0.

Using this equation, we have that

$$1 = \frac{1}{2}\bar{x}_1\bar{x}_2 + \frac{1}{2}\bar{x}_1\bar{x}_3 + \frac{1}{2}x_1\bar{x}_2\bar{x}_3 - \frac{1}{2}x_1x_2x_3 + \frac{1}{2}(x_1 + x_2 + x_3)\sum_{I\subseteq\{4,5,6\}}\left(\prod_{i\in I} x_i\right)\left(\prod_{j\in\{4,5,6\}\setminus I} \bar{x}_j\right)$$

This Nullstellensatz proof has total coefficient size $4 * \frac{1}{2} + \frac{3*8}{2} = 14$. However, for each $I \subseteq \{4,5,6\}$, two of the axioms of the form $x_k \left(\prod_{i\in I} x_i\right)\left(\prod_{j\in\{4,5,6\}\setminus I} \bar{x}_j\right) = 0$ for $k \in \{1,2,3\}$ are needed to prove infeasibility. We also need one of the three axioms $\bar{x}_1\bar{x}_2 = 0$, $\bar{x}_1\bar{x}_3 = 0$, and $\bar{x}_2\bar{x}_3 = 0$. Thus, any Nullstellensatz proof of infeasibility must have size at least $2 * 8 + 1 = 17$.

B Total coefficient size upper bound for the pigeonhole principle

In this section, we use a divide and conquer approach to give a unary Nullstellensatz proof of the pigeonhole principle with size $2^{O(n)}$. Before giving our proof, we discuss other potential approaches for constructing a Nullstellensatz proof for the pigeonhole principle and why they were insufficient for our purposes.

One approach is to use the observation that if we have a tree-like resolution proof with S leaves, this gives us a Nullstellensatz proof of size S where every coefficient is 1.

There is a simple tree-like resolution proof of size $O((n+1)!)$ which works as follows. For each pigeon i, we query the variables $\{x_{ij} : j \in [n]\}$ one by one and stop when we find a j such that $x_{i,j} = 1$ or we have queried all of these variables. If we already had that $x_{i',j} = 1$ for some $i' < i$ then this contradicts the axiom $\neg x_{i',j} \vee \neg x_{i,j}$. If none of the $x_{i,j}$ are 1 then this contradicts the axiom $\bigvee_{j\in[n]} x_{i,j}$. If pigeon i was placed in a new hole j then we continue on to pigeon $i+1$.

This gives an upper bound of $O((n+1)!)$. However, it has been shown [8, 17, 26] that every tree-like resolution proof for the pigeonhole principle has size $n^{\Omega(\log(n))}$ so this is essentially the best that we can do with this approach.

Buss and Pitassi [13] showed that there is a resolution proof of size $O(n^3 2^n)$ for the pigeonhole principle. The idea behind this proof is as follows.

▶ **Definition 28.** *Given $S = \{s_1, \ldots, s_j\} \subseteq [n+1]$, define $C_{S,[j,n]}$ to be the clause*

$$C_{S,[j,n]} = \bigvee_{i\in[j],k\in[j,n]} x_{s_i,k}.$$

In other words, the clause $C_{S,[j,n]}$ says that at least one of the j pigeons in S must go into one of the holes in $[j,n]$.

At stage j, we start with the clauses $\{C_{S,[j,n]} : S \subseteq [n+1], |S| = j\}$ and derive the clauses $\{C_{S',[j+1,n]} : S' \subseteq [n+1], |S'| = j+1\}$. After stage n, this gives us the empty clause, which proves that the pigeonhole axioms are infeasible. However, it is not clear how to translate this proof into a Nullstellensatz proof without blowing up the size and total coefficient size.

We now give a unary Nullstellensatz proof of the pigeonhole principle which has size $2^{O(n)}$.

▶ **Theorem 29.** *For all $n \in \mathbb{N}$, there is a unary Nullstellensatz proof of size at most $2^{5(n+1)}$ for the pigeonhole principle with $n+1$ pigeons and n holes.*

Proof. We construct this proof recursively as follows. Given $k \in \mathbb{N}$, a set $S = \{p_1, \ldots, p_{k+1}\}$ of $k+1$ pigeons, and a set $H = \{h_1, \ldots, h_k\}$ of k holes, we want to show the equality

$$\prod_{a=1}^{k+1}\left(1 - \prod_{b=1}^{k}(1 - x_{p_a,h_b})\right) = \prod_{a=1}^{k+1}\left(1 - \prod_{b=1}^{k}\bar{x}_{p_a,h_b}\right) = 0$$

using the hole axioms. Note that this equality coresponds to the statement that there is at least one pigeon in S which does not go to any of the holes in H.

We do this as follows. If $k = 1$ then this equality is a hole axiom. If $k > 2$ then

1. For each $a \in [k]$, we decompose the term $\left(1 - \prod_{b=1}^{k}\bar{x}_{p_a,h_b}\right)$ as

$$\left(1 - \prod_{b=1}^{k}\bar{x}_{p_a,h_b}\right) = \left(1 - \prod_{b=1}^{\lfloor\frac{k+1}{2}\rfloor}\bar{x}_{p_a,h_b}\right) + \left(\prod_{b=1}^{\lfloor\frac{k+1}{2}\rfloor}\bar{x}_{p_a,h_b}\right)\left(1 - \prod_{b=\lfloor\frac{k+1}{2}\rfloor+1}^{k}\bar{x}_{p_a,h_b}\right)$$

This gives us the equality

$$\prod_{a=1}^{k+1}\left(1-\prod_{b=1}^{k}\bar{x}_{p_a,h_b}\right)=$$
$$\sum_{A\subseteq[k+1]}\left(\prod_{a\in A}\left(1-\prod_{b=1}^{\lfloor\frac{k}{2}\rfloor}\bar{x}_{p_a,h_b}\right)\right)\left(\prod_{a\in[k+1]\setminus A}\left(\prod_{b=1}^{\lfloor\frac{k+1}{2}\rfloor}\bar{x}_{p_a,h_b}\right)\right)\left(1-\prod_{b=\lfloor\frac{k+1}{2}\rfloor+1}^{k}\bar{x}_{p_a,h_b}\right)\right)$$

2. For each of the 2^{k+2} resulting terms, we check whether $|A|\geq\lfloor\frac{k+1}{2}\rfloor+1$ or $|A|\leq\lfloor\frac{k+1}{2}\rfloor$. If $|A|\geq\lfloor\frac{k+1}{2}\rfloor+1$ then letting A' be the first $\lfloor\frac{k+1}{2}\rfloor+1=\lceil\frac{k+2}{2}\rceil$ elements of A, we recursively construct a proof that $\prod_{a\in A'}\left(1-\prod_{b=1}^{\lfloor\frac{k+1}{2}\rfloor}\bar{x}_{p_a,h_b}\right)=0$. If $|A|\leq\lfloor\frac{k+1}{2}\rfloor$ then $|[k+1]\setminus A|\geq\lceil\frac{k+1}{2}\rceil$ so letting A'' be the first $\lceil\frac{k+1}{2}\rceil$ elements of $[k+1]\setminus A$, we recursively construct a proof that $\prod_{a\in A''}\left(1-\prod_{b=\lfloor\frac{k+1}{2}\rfloor+1}^{k}\bar{x}_{p_a,h_b}\right)=0$.

To obtain our Nullstellensatz proof, we construct this proof for $S=[n+1]$ and $H=[n]$. We then use the following equality (recall that the pigeon axioms are $\{\prod_{b=1}^{n}\bar{x}_{a,b}=0:a\in[n+1]\}$):

$$1=\prod_{a=1}^{n+1}\left(1-\prod_{b=1}^{n}\bar{x}_{a,b}\right)+\sum_{j=1}^{n+1}\left(\prod_{b=1}^{n}\bar{x}_{j,b}\right)\left(\prod_{a=1}^{j-1}\left(1-\prod_{b=1}^{n}\bar{x}_{a,b}\right)\right)$$

The size of the resulting unary Nullstellensatz proof can be upper bounded by $S(n)+2^{n+1}$ where $S(n)$ is the solution to the recurrence relation $S(n)=2^{2(n+1)}S(\lceil\frac{n+2}{2}\rceil)$ where $S(1)=1$. It is not hard to show by induction that $S(n)\leq 2^{5(n+1)}-2^{(n+1)}$ so this gives an upper bound of $2^{5(n+1)}$. ◀

C Total coefficient size upper bound for the ordering principle

In this section, we construct an explicit Nullstellensatz proof for the ordering principle on n elements with total coefficient size 2^n-n. In the full version of this paper [29], we also present experimental results obtained by implementing the linear program for minimum total coefficient size. One of our experimental results is that the 2^n-n upper bound is tight for $n\leq 5$.

We start by formally defining the ordering principle.

▶ **Definition 30** (ordering principle (ORD_n)). *Intuitively, the ordering principle says that any well-ordering on n elements must have a minimum element. Formally, for $n\geq 1$, we define ORD_n to be the statement that the following system of axioms is infeasible:*

- *We have a variable $x_{i,j}$ for each pair $i,j\in[n]$ with $i<j$, with the Boolean axiom $x_{i,j}^2-x_{i,j}=0$. $x_{i,j}=1$ represents element i being less than element j in the well-ordering, and $x_{i,j}=0$ represents element i being more than element j in the well-ordering. We write $x_{j,i}$ as shorthand for $\bar{x}_{i,j}=1-x_{i,j}$.*
- *For each $i\in[n]$, we have the axiom $\prod_{j\in[n]\setminus\{i\}}x_{i,j}=0$ which represents the constraint that element i is not a minimum element. We call these axioms non-minimality axioms.*
- *For each triple $i,j,k\in[n]$ where $i<j<k$, we have the two axioms $x_{i,j}x_{j,k}x_{k,i}=0$ and $x_{k,j}x_{j,i}x_{i,k}=0$ which represent the constraints that elements i,j,k satisfy transitivity. We call these axioms transitivity axioms.*

In our Nullstellensatz proof $1=\sum_W c_W W$, each c_W is either 0 or 1. Non-minimality axioms have coefficient 1, and all weakenings of transitivity axioms that have coefficient 1 must have a special form:

▶ **Definition 31** (nice transitivity weakening). *Let W be a weakening of the axiom $x_{i,j}x_{j,k}x_{k,i}$ or the axiom $x_{k,j}x_{j,i}x_{i,k}$ for some $i < j < k$. Let $G(W)$ be the following directed graph. The vertices of $G(W)$ are $[n]$. For distinct $i', j' \in [n]$, $G(W)$ has an edge from i' to j' if W contains the term $x_{i',j'}$. We say that W is a nice transitivity weakening if $G(W)$ has exactly n edges and all vertices are reachable from vertex i.*

In other words, if W is a weakening of the axiom $x_{i,j}x_{j,k}x_{k,i}$ or the axiom $x_{k,j}x_{j,i}x_{i,k}$, then $G(W)$ contains a 3-cycle on vertices $\{i, j, k\}$. W is a nice transitivity weakening if and only if contracting this 3-cycle results in a directed spanning tree rooted at the contracted vertex. Note that if W is a nice transitivity weakening and x is an assignment with a minimum element, then $W(x) = 0$.

▶ **Theorem 32.** *There is a Nullstellensatz proof for* ORD_n *satisfying:*

1. *The total coefficient size is $2^n - n$.*
2. *Each c_W is either 0 or 1.*
3. *If A is a non-minimality axiom, then $c_A = 1$, and $c_W = 0$ for all other weakenings W of A.*
4. *If W is a transitivity weakening but not a nice transitivity weakening, then $c_W = 0$.*

Proof. We prove Theorem 32 by induction on n. When $n = 3$, the desired Nullstellensatz proof sets $c_A = 1$ for each axiom A. It can be verified that $\sum_W c_W W$ evaluates to 1 on each assignment, and that this Nullstellensatz proof satisfies the properties of Theorem 32.

Now suppose we have a Nullstellensatz proof for ORD_n satisfying Theorem 32, and let S_n denote the set of transitivity weakenings W for which $c_W = 1$. The idea to obtain a Nullstellensatz proof for ORD_{n+1} is to use two copies of S_n, the first copy on elements $\{1, \ldots, n\}$ and the second copy on elements $\{2, \ldots, n+1\}$. Specifically, we construct the Nullstellensatz proof for ORD_{n+1} by setting the following c_W to 1 and all other c_W to 0.

1. For each non-minimality axiom A in ORD_{n+1}, we set $c_A = 1$.
2. For each $W \in S_n$, we define the transitivity weakening W' on $n+1$ elements by $W' = W \cdot x_{1,n+1}$ and set $c_{W'} = 1$.
3. For each $W \in S_n$, first we define the transitivity weakening W'' on $n+1$ elements by replacing each variable $x_{i,j}$ that appears in W by $x_{i+1,j+1}$ (e.g., if $W = x_{1,2}x_{2,3}x_{3,1}$, then $W'' = x_{2,3}x_{3,4}x_{4,2}$). Then, we define $W''' = W''x_{n+1,1}$ and set $c_{W'''} = 1$.
4. For each $i \in \{2, \ldots, n\}$, for each of the 2 transitivity axioms A for elements $\{1, i, n+1\}$, we set $c_W = 1$ for the following weakening W of A:

$$W = A\left(\prod_{j \in \{2,\ldots,n\} \setminus \{i\}} x_{i,j}\right).$$

In other words, $W(x) = 1$ if and only if $A(x) = 1$ and i is the minimum element among $\{2, \ldots, n\}$.

The desired properties 1 through 4 in Theorem 32 can be verified by induction. It remains to show that for each assignment x, there is exactly one nonzero c_W for which $W(x) = 1$. If x has a minimum element $i \in [n+1]$, then the only nonzero c_W for which $W(x) = 1$ is the non-minimality axiom for i. Now suppose that x does not have a minimum element. Consider two cases: either $x_{1,n+1} = 1$, or $x_{n+1,1} = 1$. Suppose $x_{1,n+1} = 1$. Consider the two subcases:

1. Suppose that, if we ignore element $n+1$, then there is still no minimum element among the elements $\{1, \ldots, n\}$. Then there is exactly one weakening W in point 2 of the construction for which $W(x) = 1$, by induction.
2. Otherwise, for some $i \in \{2, \ldots, n\}$, we have that i is a minimum element among $\{1, \ldots, n\}$ and $x_{n+1,i} = 1$. Then there is exactly one weakening W in point 4 of the construction for which $W(x) = 1$ (namely, the weakening W of the axiom $A = x_{i,1}x_{1,n+1}x_{n+1,i}$).

The case $x_{n+1,1} = 1$ is handled similarly by considering whether there is a minimum element among $\{2, \ldots, n+1\}$. Assignments that do have a minimum element among $\{2, \ldots, n+1\}$ are handled by point 3 of the construction, and assignments that do not are handled by point 4 of the construction. ◀

Adaptive Sparsification for Matroid Intersection

Kent Quanrud
Dept. of Computer Science, Purdue University, West Lafayette, IN, USA

Abstract

We consider the matroid intersection problem in the independence oracle model. Given two matroids over n common elements such that the intersection has rank k, our main technique reduces approximate matroid intersection to logarithmically many primal-dual instances over subsets of size $\tilde{O}(k)$. This technique is inspired by recent work by [2] and requires additional insight into structuring and efficiently approximating the dual LP. This combination of ideas leads to faster approximate maximum cardinality and maximum weight matroid intersection algorithms in the independence oracle model. We obtain the first nearly linear time/query approximation schemes for the regime where $k \leq n^{2/3}$.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Matroid intersection, adaptive sparsification, multiplicative-weight udpates, primal-dual

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.118

Category Track A: Algorithms, Complexity and Games

Funding *Kent Quanrud*: Supported in part by NSF grant CCF-2129816.

Acknowledgements We thank the reviewers for their careful and helpful feedback. We thank Adrian Vladu for teaching us about [2].

1 Introduction

Matroid intersection is a classical problem in combinatorial optimization for which faster algorithms have been a recent topic of interest.

A *matroid*, $\mathcal{M} = (\mathcal{N}, \mathcal{I})$, consists of a set of n elements $\mathcal{N}$ and a collection of subsets $\mathcal{I}$ of $\mathcal{N}$, known as the independent sets, that satisfy the following properties: (i) the empty set is independent, (ii) every subset of an independent set is independent (hereditary property), and (iii) if A and B are two independent sets with $|A| > |B|$, then there exists an element in $A \setminus B$ that can be added to B to still have an independent set (exchange property). These properties imply that every *maximal* set also has maximum cardinality. The maximum cardinality of any independent set is called the *rank*. Examples of matroids include the family of forests of a graph (the graphic matroid) and the family of independent sets of vectors in a vector space (the linear matroid).

Matroid intersection. The problem of matroid intersection considers two matroids, $\mathcal{M}_1 = (\mathcal{N}, \mathcal{I}_1)$ and $\mathcal{M}_2 = (\mathcal{N}, \mathcal{I}_2)$, defined on a common ground set $\mathcal{N}$, and seeks the largest set that is independent in both matroids. Formally, the goal is to maximize the size of a set $S \subseteq \mathcal{N}$ such that $S \in \mathcal{I}_1 \cap \mathcal{I}_2$. Unlike matroids, a maximal cardinality independent set in the matroid intersection is not necessarily a maximum cardinality independent set. The maximum cardinality, denoted OPT, is also called the *rank* of the matroid intersection and denoted by k. Matroid intersection generalizes bipartite matching, and has other connections in combinatorial optimization. For example, by Edmonds' directionless tree packing theorem,

matroid intersection captures the maximum number of rooted arborescences that can be packed into a directed graph, and the directed rooted connectivity [11]. See [27, 15] for additional background and connections.

Algorithms addressing matroid intersection in general are commonly framed in the *independence oracle model*. Here the algorithm is allowed to query if a given set S is independent in a matroid. When stating running times in this model, we let Q denote the running time of a single independence oracle.

The first polynomial time algorithm for matroid intersection was given by [10] by reduction to matroid union [12]. This algorithm ran in $O\big(n^4Q\big)$ time. More direct augmenting path algorithms were developed by [1, 21]. The algorithm in [21] ran in $O\big(nk^2Q\big)$ time. Generalizing ideas from bipartite matching [17], [9] gave a faster matroid intersection algorithm running in $O\big(nk^{1.5}Q\big)$ time. Truncating the algorithm early implies a $(1-\epsilon)$-approximation in $O(nkQ/\epsilon)$ time for $\epsilon \in (0,1)$ [8].

Recently there has been a resurgence of interest in faster algorithms in the independence oracle model. [7, 23] gave $O(nk\log(k)Q)$ time algorithms, leveraging the observation that the auxiliary graph can be searched faster than it can be built out explicitly. [5] pushed this direction further and obtained $\tilde{O}\big(n^{9/5}Q\big)$ randomized and $\tilde{O}\big(n^{11/6}Q\big)$ deterministic time algorithms, the first $o(n^2)$ time algorithms for $k = \Omega(n)$.[1] Finally, [4] obtained a $\tilde{O}\Big(n\sqrt{k}Q/\epsilon\Big)$ time deterministic algorithm for $(1-\epsilon)$-matroid intersection. This faster approximation algorithm implied faster exact algorithms running in $\tilde{O}\big(nk^{3/4}Q\big)$ randomized time and $\tilde{O}\big(nk^{5/6}Q\big)$ deterministic time. [4]'s algorithms represent the state of the art.

Weighted matroid intersection. In the weighted matroid intersection problem, we are also given weights $c : \mathcal{N} \to \mathbb{R}_{>0}$. The goal is to compute $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ of maximum weight $c(I)$. Edmonds [10] gave the first polynomial time algorithm. Faster algorithms were developed in [21, 13, 6, 16, 22]. Frank's citeFrank1981a algorithm runs in $O(k(T + n\log n))$ time, where T is the running time of any exact matroid intersection algorithm. The algorithm in [22] runs in $O\big(n^2\log(n)Q + n^3\,\mathrm{polylog}(n)\big)$ time.

There is also recent interest in fast $(1-\epsilon)$-approximation algorithms [18, 8]. The $(1-\epsilon)$-approximation algorithm in [8] runs in $\tilde{O}\big(nkQ/\epsilon^2\big)$ time.

1.1 Results

Our primary focus is on faster approximation algorithms. As alluded to above, approximation algorithms can play a role in exact algorithms. (The fastest algorithms use augmenting paths to extend approximate solutions to exact ones.) They are also useful in their own right when one is willing to tolerate some error in exchange for scalability.

We introduce adaptive sparsification to matroid intersection in order to develop faster approximation algorithms. The new sparsification technique reduces approximate matroid intersection to $O(\log(n))$ instances of approximate matroid intersection and the dual problem over a subset of $O(k\log(n))$ elements. The technique holds for both weighted and unweighted matroid intersection. We leverages these ideas to obtain improved running times for approximating the unweighted and weighted settings.

Maximum cardinality matroid intersection. Henceforth, let $\mathcal{M}_1 = (\mathcal{N}, \mathcal{I}_1)$ and $\mathcal{M}_2 = (\mathcal{N}, \mathcal{I}_2)$ be two matroids over a common groundset of n elements, let k be the rank of their intersection, and let $\epsilon \in (0,1)$. In the following, the "dual" refers to the dual of the standard

[1] $\tilde{O}(\cdots)$ hides polylogarithmic factors.

packing LP for matroid intersection, and is introduced formally below in Section 1.2. We first show how to reduce approximate matroid intersection over n elements to approximating $O(\log n)$ primal-dual instances of matroid intersection over $\tilde{O}(k)$ elements.

▶ **Lemma 1.** *Suppose a $(1-\epsilon)$-maximum matroid intersection and a $(1+\epsilon)$-approximate dual solution in $\mathcal{I}_1 \cap \mathcal{I}_2$, over a subset of m elements, can be computed with high probability in $T_\epsilon(m)$ time in the independence oracle model. Then a $(1-\epsilon)$-approximate matroid intersection can be computed with high probability in $O(n\log(n)Q/\epsilon + \mathcal{T}_\epsilon(k\log(n)/\epsilon)/\epsilon)$ randomized time in the independence oracle model, where Q represents an independence query.*

To apply Lemma 1 to matroid intersection, we need a fast algorithm to compute both a $(1-\epsilon)$-maximum matroid intersection and an $(1+\epsilon)$-dual solution. Recall that [4] computes a $(1-\epsilon)$-maximum matroid intersection in $\tilde{O}\left(n\sqrt{k}Q/\epsilon\right)$ time. The solution returned by [4] has stronger properties (based on the length of augmenting paths), and we leverage these properties to compute a $(1+\epsilon)$-approximate dual solution without increasing the running time.

▶ **Lemma 2.** *A $(1-\epsilon)$-maximum matroid intersection I, and an $(1+\epsilon)$-dual solution (S,T), can be computed in $\tilde{O}\left(n\sqrt{k}Q/\epsilon\right)$ deterministic time.*

Using this as a $(1\pm\epsilon)$-primal dual approximation algorithm in Lemma 1, with $\mathcal{T}_\epsilon(n) = \tilde{O}\left(n\sqrt{k}Q/\epsilon\right)$, we have the following improved running time for approximate matroid intersection.

▶ **Theorem 3.** *A $(1-\epsilon)$-maximum matroid intersection and an $(1+\epsilon)$-dual solution can be computed with high probability in $O\left(n\log(n)Q/\epsilon + k^{3/2}\log^{O(1)}(k)Q/\epsilon^3\right)$ randomized time in the independence oracle model.*

Compared to previous results, the improved running time in Theorem 3 removes the $\text{poly}(k)$-factor from the dominant term of n. Theorem 3 gives the first nearly linear time approximation scheme for the regime where $k \le n^{2/3}$.

Theorem 3 does not imply a faster algorithm for exact matroid intersection, but brings us significantly closer. This is because there are two bottlenecks in [4]. Theorem 3 addresses one of them. The remaining bottleneck is a subroutine augmenting an independent set one element at a time. The current best bound for this subroutine is $\tilde{O}\left(n\sqrt{k}Q\right)$ time per augmenting path [5].

Maximum weight matroid intersection. We now consider maximum weight matroid intersection. In addition to the inputs $\mathcal{M}_1$, $\mathcal{M}_2$, and $\epsilon \in (0,1)$, let $c : \mathcal{N} \to \mathbb{R}_{\ge 0}$ be an input weight vector.

As in the unweighted case, we show how to reduce approximate maximum weight matroid intersection to approximating $O(\log(n))$ primal-dual instances over subsets of $\tilde{O}(k)$ elements. Here the "dual" refers to the dual LP, introduced later in Section 3. For the weighted setting, the sparsification technique requires particularly structured dual solutions which we call "compact" dual solutions. We elaborate more on compact dual solutions in Section 3 and for the time being state the lemma informally.

▶ **Lemma 4** (Informal). *Suppose that a $(1-\epsilon)$-maximum weight matroid intersection and a "compact" $(1+\epsilon)$-minimum dual solution over a subset of m elements can be computed in $\mathcal{T}_\epsilon(m)$ time with high probability. Then a $(1-\epsilon)$-maximum weight matroid intersection and a $(1+\epsilon)$-minimum dual solution can be computed with high probability in $O(n\log(n)Q/\epsilon + \mathcal{T}_\epsilon(k\log(n)/\epsilon)/\epsilon)$ randomized time in the independence oracle model.*

Using this technique requires a $(1 \pm \epsilon)$-primal-dual approximation algorithm where the dual solution has nice "compact" properties. A fast primal algorithm is given by [8] (accelerated by [4]) and we extend it to give an approximate dual solution that is also compact.

▶ **Lemma 5** (Informal). *A $(1-\epsilon)$-maximum weight matroid intersection and a "compact" $(1+\epsilon)$-minimum dual solution can be computed in $\tilde{O}\left(n\sqrt{k}Q/\epsilon^2\right)$ time.*

Putting these two results together gives the following improved running time for approximate maximum weight matroid intersection.

▶ **Theorem 6.** *A $(1-\epsilon)$-maximum weight matroid intersection and a $(1+\epsilon)$-minimum dual solution can be computed with high probability in $O\left(n\log(n)Q/\epsilon + k^{3/2}Q/\epsilon^4\right)$ randomized time in the independence oracle model.*

Compared to the previous state of the art, Theorem 6 removes the poly(k)-factor from the dominant term n. It also reduces the poly$(1/\epsilon)$-factor against n, from $1/\epsilon^2$ to $1/\epsilon$. Theorem 6 is the first approximation scheme for weighted matroid intersection running in nearly linear time for all $k \leq n^{2/3}$.

1.2 High-level overview of the algorithms and techniques

The first technique we introduce to matroid intersection is adaptive sparsification. In graph algorithms, sparsification is a powerful and (by now) standard technique where a dense input graph is reduced to a sparse one, while (approximately) preserving salient properties like the size of every cut [3] or the Laplacian [29, 28]. These algorithms are also fast. There is previous work sparsifying matroids individually [20, 25, 26], generalizing cut sparsification.

For matroid intersection, we are aware of two instances of sparsification. [7] sparsifies matroid intersection by first approximating a linear relaxation for matroid intersection. They then randomly round their solution x to a solution y with support of size $O\left(k\log(n)/\epsilon^2\right)$. Concentration bounds from [20] imply that the support of y contains a $(1-\epsilon)$-approximate matroid intersection with high probability. The only catch to this approach is that it takes $\tilde{O}\left(n^2Q/k\epsilon^2\right)$ time to compute the point x. The second instance is the very recent work of [19], which reduces the number of elements to $\tilde{O}\left(k/\epsilon^{O(1)}\right)$ while preserving the value of the matroid intersection up to a $(3/2+\epsilon)$-factor. However this is far from preserving the intersection up to a $(1+\epsilon)$-factor. (We note that the techniques of [19] have additional motivating factors including communication complexity and the streaming model.) It seems difficult to accurately preserve the matroid intersection via a "one-shot" static sparsifier without introducing another bottleneck in the running time.

This work pivots away from static sparsifiers to *adaptive* ones, where a large instance of matroid intersection is reduced to a limited number of sparse instances. The sparse instances are generated sequentially by random samples, where the distribution of each sample adapts to the outcomes of previous iterations. We are inspired by and build upon a recent and elegant work by Assadi [2], which used adaptive sampling to compute $(1-\epsilon)$-approximate maximum weight matchings in the semi-streaming model. We briefly sketch the ideas from [2]. While [2]'s techniques extend to general graphs, we restrict our discussion to bipartite matching as it is a special case of matroid intersection. The input is a bipartite graph $G = (V, E)$, and we are constrained to memory of size $\tilde{O}(|V|)$. In particular, for dense graphs, one cannot hold the entire graph in memory. The algorithm may read the edges E one by one in a streaming fashion; each iteration over E is called a "pass".

There are several known results in this model and the contribution of [2] was to give a simpler algorithm competitive with the state of the art. [2] reduces $(1-\epsilon)$-maximum weight matching to $O(\log(n)/\epsilon)$ successive instances of $(1-\epsilon)$-maximum weight matching and $(1+\epsilon)$-minimum vertex cover over subgraphs of $\tilde{O}(|V|/\epsilon)$ edges. Each $\tilde{O}(|V|/\epsilon)$-size instance is obtained by a nonuniform sample of the edges that can be implemented in a single pass over the edges.

The probabilities are based on multiplicative weights. Initially, all edges have the same weight and are sampled uniformly. Each iteration, the algorithm obtains a $(1+\epsilon)$-minimum vertex cover that covers all the sampled edges, but not necessarily all the input edges. The sampling probability of each uncovered edge is doubled. Intuitively the algorithm is trying to sample a small set of edges that forces the dual covering solutions of the sample to cover all the edges, at least on average. Edges that are frequently covered have exponentially smaller weight; edges that are not covered much have exponentially larger weight.

While [2] gives a direct analysis, one can interpret their algorithm a little more generally within a standard MWU framework applied to the dual vertex cover LP. We broaden the argument to covering problems in general. In the following, a $(1+\epsilon)$-approximation algorithm refers to a point $x \in \mathcal{P}$ such that $\langle b, x\rangle \le (1+\epsilon)\,\mathrm{OPT}$ and $(1+\epsilon)Ax \ge \mathbb{1}$.

▶ **Lemma 7.** *Consider a covering LP of the form*

$$\textit{mimimize } \langle b, x\rangle \textit{ over } x \in \mathcal{P} \textit{ s.t. } Ax \ge \mathbb{1},$$

where $\mathcal{P}$ is a convex set, and A has nonnegative coefficients and m constraints. Suppose one has access to an oracle that, given any nonnegative set of weights $w \in \mathbb{R}^m_{\ge 0}$, computes a point x such that:

(a) $\langle b, x\rangle \le (1+\epsilon)\,\mathrm{OPT}$

(b) $\sum_{i:(Ax)_i \ge 1} w_i \ge (1-\epsilon)\sum_i w_i$.

Suppose the oracle also returns a list of all constraints $i \in [m]$ covered by x (i.e., such that $(Ax)_i \ge 1$). Then one can compute a $(1+\epsilon)$-approximation solution as the average of $L = O(\log(m)/\epsilon)$ solutions returned by the oracle for an adaptively chosen sequence of L weight vectors.

The intuition for the oracle problem is as follows. We are given a covering LP, and the challenge is to satisfy *all* the covering constraints simultaneously. The oracle problem relaxes this uniform requirement by assigning nonnegative weights to each constraint, and asks for a solution that satisfies *most* of the constraints by weight. The small difference between satisfying all constraints, and satisfying *almost* all the constraints, is just large enough to permit random sampling and other techniques that trade a controlled amount of error for significantly faster running times. The surrounding framework adjusts the weights dynamically so that on average, the oracle solutions (approximately) cover all the constraints simultaneously.

Matroid intersection. To apply this framework to matroid intersection we must understand the dual covering problem, which requires the notions of a rank function and a span function. For a given matroid $\mathcal{M} = (\mathcal{N}, \mathcal{I})$, and set $S \subseteq \mathcal{N}$, the *rank of* S, denoted $\mathrm{rank}(S)$, is the maximum cardinality of any independent subset of S. For $S \subseteq \mathcal{N}$, the *span* of S, $\mathrm{span}(S)$, is the set of elements whose inclusion does not increase the rank, including the elements in S: $\mathrm{span}(S) = \{e \in \mathcal{N} : \mathrm{rank}(S+e) = \mathrm{rank}(S)\}$. A set S is *closed* if $S = \mathrm{span}(S)$. Let rank_1 and rank_2 denote the rank functions of $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively. Similarly, let span_1 and span_2 denote the span functions of $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively.

The standard LP relaxation for matroid intersection asks for the maximum nonnegative modular function dominated by rank_1 and rank_2:

$$\begin{aligned} &\text{maximize } x(\mathcal{N}) \text{ over } x : \mathcal{N} \to \mathbb{R}_{\geq 0} \\ &\quad \text{s.t. } x(S) \leq \text{rank}_1(S) \text{ and } x(S) \leq \text{rank}_2(S) \text{ for all } S \subseteq \mathcal{N}. \end{aligned} \tag{1}$$

Here we denote the sum $x(S) \stackrel{\text{def}}{=} \sum_{e \in S} x_e$ for $S \subseteq \mathcal{N}$.

From a dual perspective, for all partitions $(S, \bar{S})$ of the ground set (where $\bar{S} = \mathcal{N} \setminus S$), $\text{rank}_1(S) + \text{rank}_2(\bar{S})$ is an upper bound on the size of any matroid intersection $I \in \mathcal{I}_1 \cap \mathcal{I}_2$. (We have $|I \cap S| \leq \text{rank}_1(S)$ and $|I \cap \bar{S}| \leq \text{rank}_2(\bar{S})$). It is also an upper bound on the LP relaxation as can be seen from duality as follows.

Consider the problem of minimizing $\text{rank}_1(S) + \text{rank}_2(\bar{S})$ over all $S \subseteq V$. The standard LP relaxation is the dual LP of the matroid intersection LP (1):

$$\begin{aligned} &\text{minimize } \sum_{S \subseteq \mathcal{N}} y_S \, \text{rank}_1(S) + z_S \, \text{rank}_2(S) \text{ over } y, z : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0} \\ &\quad \text{s.t. } \sum_{S : e \in S} y(S) + z(S) \geq 1 \text{ for all } e \in \mathcal{N}. \end{aligned} \tag{2}$$

A classical theorem by [10] states that $\max_{I \in \mathcal{I}_1 \cap \mathcal{I}_2} |I| = \min_{S \subseteq V} \text{rank}_1(S) + \text{rank}_2(\bar{S})$, hence both LPs (1) and (2) have integral optimum solutions.

Lemma 7 applies a variation of the MWU framework to the dual covering LP (2). The MWU framework incrementally builds a fractional solution (y, z) over $L = O(\log(n)/\epsilon)$ iterations. Initially $(y, z) = (\mathbb{0}, \mathbb{0})$. Each iteration ℓ queries the oracle for a particular set of weights $w^{(\ell)} \in \mathbb{R}^{\mathcal{N}}_{\geq 0}$, returning $(\tilde{y}^{(\ell)}, \tilde{z}^{(\ell)})$ as described in Lemma 7. We increase y by $\tilde{y}^{(\ell)}/L$ and z by $\tilde{z}^{(\ell)}/L$. The key point is how the weights are chosen. For each element e, in the ℓth iteration, we have

$$w^{(\ell)}(e) = \exp(-(\#\text{ iterations } k < \ell \text{ where } (\tilde{y}^{(k)}, \tilde{z}^{(k)}) \text{ covers } e)).$$

The weight of an element e decays exponentially with the number of oracle solutions that cover e.

To implement the oracle for matroid intersection, given a set of weights $w : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$, we sample $O(k \log(n)/\epsilon)$ elements $\mathcal{N}' \subseteq \mathcal{N}$ in proportion to w. We then compute a $(1-\epsilon)$-maximum matroid intersection I and a dual $(1+\epsilon)$-minimum dual integral solution (S', T'), where $S', T', \subseteq \mathcal{N}'$, for the subproblem over $\mathcal{N}'$. Of course S' and T' do not cover any elements outside of $\mathcal{N}'$, and would fail to satisfy Item b of Lemma 7. We enlarge these sets by taking their spans, $S = \text{span}_1(S')$ and $T = \text{span}_2(T')$, which hopefully includes most of the elements from $\mathcal{N} \setminus \mathcal{N}'$ (by weight w). (S, T) (encoded in the LP by their indicator vectors) is the solution returned by our oracle.

Assuming each iteration implements the oracle of Lemma 7, Lemma 7 asserts that the average of the dual solutions gives a $(1+\epsilon)$-approximate dual solution to the matroid intersection problem. We really want a $(1 - O(\epsilon))$-approximate matroid intersection. Recall that each iteration also gives a matroid intersection I within a $(1 - O(\epsilon))$-factor of a dual solution over the subproblem. Therefore the maximum cardinality of I over all iterations is within a $(1 - O(\epsilon))$-factor of the average dual solution. The average dual solution is feasible (up to scaling by $(1 + O(\epsilon))$), certifying that I is a $(1 - O(\epsilon))$-matroid intersection.

This describes the approximation algorithm for maximum cardinality matroid intersection. Pseudocode is given in Figure 1. The analysis requires both an MWU analysis of Lemma 7, and a more matroid-specific analysis to implement the oracle. The former is similar to

1. Let $w(d) = 1$ for all $d \in \mathcal{N}$.
2. For $\ell = 1, \dots, L$, where $L = O(\log(n)/\epsilon)$:
   A. Let $\mathcal{N}' \subseteq \mathcal{N}$ sample $O(k \log(n)/\epsilon)$ elements with repetition in proportion to $w(e)$. // $\tilde{O}(n + k/\epsilon)$
   B. Compute a $(1-\epsilon)$-approximate matroid intersection $I^{(\ell)}$ and a $(1+\epsilon)$-approximate dual solution $(\tilde{S}^{(\ell)}, \tilde{T}^{(\ell)})$. // $\tilde{O}(k^{3/2}Q/\epsilon^2)$
   C. Let $S^{(\ell)} = \mathrm{span}_1(\tilde{S}^{(\ell)})$ and $T^{(\ell)} = \mathrm{span}_2(\tilde{T}^{(\ell)})$. // $\tilde{O}(nQ)$
   D. For all elements $d \in S^{(\ell)} \cup T^{(\ell)}$, set $w(d) = w(d)/e$. // $\tilde{O}(n)$
3. Let $I^{(\ell)}$ maximize $|I^{(\ell)}|$ over all iterations $\ell \in [L]$. Define (y, z) to be the fractional average of the dual solutions,

$$y = \frac{1}{L}\sum_{\ell=1}^{L} 1_{S^{(\ell)}} \text{ and } z = \frac{1}{L}\sum_{\ell=1}^{L} 1_{T^{(\ell)}},$$

   where 1_X denotes the indicator vector of $X \subseteq \mathcal{N}$.
   Return $(I^{(\ell)}, (1+O(\epsilon))y, (1+O(\epsilon))z)$.

Figure 1 A randomized, $(1 \pm \epsilon)$-approximation algorithm for maximum cardinality matroid intersection and the dual LP.

the analysis given in [2]. Implementing the oracle has two components. First we need to show how to extend $(1-\epsilon)$-approximate matroid intersection algorithms to also produce a $(1+\epsilon)$-approximate dual solutions, without increasing the running time. We also need to prove that a $(1+\epsilon)$-approximate dual solution on the sampled subset $\mathcal{N}'$ extends to a solution satisfying the oracle model of Lemma 7.

Consider this latter point regarding sampling and the oracle problem. For the special case of bipartite matching, where dual solutions are vertex covers, the argument in [2] takes a union bound over all $2^{|V|}$ possible subsets of vertices; the logarithm of this bound is then approximately the size of the sample that is needed. For matroid intersection the dual is more abstract. There are naively 2^n dual integral solutions, which is too large. To transfer [2]'s argument to matroid intersection, we need a bound of the order of $n^{O(k)}$ on the number of dual solutions. We obtain this bound by restricting our attention to closed sets, and identifying closed sets with maximal independent subsets.

Weighted matroid intersection. One can approach weighted matroid intersection similarly. At a high level, to implement the oracle of Lemma 7, we reduce the input size in each iteration by random sampling, and build on previous $(1+\epsilon)$-maximum weight matroid intersection algorithms to extract good dual solutions. The weights inject additional technical details to each component. Greater effort is needed to bound the number of dual solutions, and this motivates the notion of "compact" dual solutions (defined in Section 3). Computing a compact dual solution efficiently requires a closer examination of the "approximate weight splitting" certificate of [8]. To this end, we give a new primal-dual proof of correctness that also efficiently constructs a compact dual solution. After addressing these combinatorial components, we recover the high-level theme of using adaptive sampling to reduce a problem over n elements to smaller primal-dual instances of the problem over roughly k elements.

Pseudocode for the approximate maximum weight matroid intersection algorithm is given in Figure 2. Several of the steps require technical elaboration and we defer a detailed discussion to Section 3.

1. Let $w(e) = 1$ for all $e \in \mathcal{N}$.
2. For $\ell = 1, \ldots, L$, where $L = O(\log(n)/\epsilon)$:
   A. Let $\mathcal{N}' \subseteq \mathcal{N}$ sample $O(k \log(n)/\epsilon)$ elements with repetition in proportion. // $\tilde{O}(k/\epsilon)$
   B. Compute a $(1-\epsilon)$-approximate matroid intersection $I^{(\ell)}$ and a ϵ-approximate cost-splitting certificate $(c_{1,\ell}, c_{2,\ell})$. // $\tilde{O}\big(k^{3/2}/\epsilon^2\big)$
   C. Extract from $(I^{(\ell)}, c_{1,\ell}, c_{2,\ell})$ a compact, $(1+\epsilon)$-approximate dual solution $\tilde{y}^{(\ell)}, \tilde{z}^{(\ell)} : 2^{\mathcal{N}'} \to \mathbb{R}_{\geq 0}$ over $\mathcal{N}'$. // $\tilde{O}(kQ)$
   D. Define $y^{(\ell)}, z^{(\ell)} : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ by

$$y^{(\ell)}(\mathrm{span}_1(S')) = \lceil \tilde{y}^{(\ell)}(S') \rceil_{1+\epsilon} \qquad \text{for all } S' \in \mathrm{support}(\tilde{y}^{(\ell)}),$$
$$z^{(\ell)}(\mathrm{span}_2(T')) = \lceil \tilde{z}^{(\ell)}(T') \rceil_{1+\epsilon} \qquad \text{for all } T' \in \mathrm{support}(\tilde{z}^{(\ell)}),$$

   where $\lceil x \rceil_{1+\epsilon} \stackrel{\text{def}}{=} (1+\epsilon)^{\lceil \log_{1+\epsilon} x \rceil}$ rounds x up to the nearest power of $1+\epsilon$. // $\tilde{O}(nQ)$
   E. For all elements $e \in \mathcal{N}$, if e is covered by $(y^{(\ell)}, z^{(\ell)})$, set $w(e) = e^{-1} w(e)$. // $O(n \log(k) Q)$
3. Let $I^{(\ell)}$ maximize $c(I^{(\ell)})$ over $\ell \in [L]$. Return $I^{(\ell)}$ and $\frac{1+O(\epsilon)}{L} \sum_{\ell=1}^{L} (y^{(\ell)}, z^{(\ell)})$.

Figure 2 A randomized, $(1 - O(\epsilon))$-approximation algorithm for weighted matroid intersection and the dual LP.

Conclusion. These algorithms are natural extensions of [2]'s algorithm for approximate bipartite matching. Conceptually, they highlight two new perspectives beyond improved running times for approximate matroid intersection. The first is exposing the versatility of the techniques in [2], beyond matchings in graphs. While we focus on matroid intersection abstractly in the independence oracle model, the techniques are simple and high-level enough to be applied to the diverse family of concrete instances of matroid intersection studied elsewhere. We also give an explicit interface to an oracle model for positive LPs that extends beyond matroid intersection. The second is to reiterate the importance of the dual of matroid intersection problems, at least from the perspective of fast approximation algorithms. Improvements for the dual approximation problems were critical to sparsifying and ultimately accelerating approximation algorithms for the primal problem.

The clear open problem is improving the running time for exact unweighted matroid intersection. We were surprised to realize that the improved approximation algorithm did not immediately imply a faster exact running time. We hope this work draws attention to the remaining bottleneck mentioned above.

Organization. We divide the rest of the article into three main parts:

Section 2: The full details of the matroid intersection algorithm and analysis, completing the description in Section 1.2, assuming and interfacing with the oracle framework of Lemma 7.

Section 3: The weighted matroid intersection algorithm and analysis, again interfacing with the oracle framework of Lemma 7.

Section 4: An MWU analysis of the general oracle framework described in Lemma 7.

2 Matroid intersection

The matroid intersection algorithm was described in Section 1.2, along with an overview of the techniques and the analysis. Pseudocode was presented in Figure 1. To briefly review, the overall algorithm follows the MWU framework described in Lemma 7, applied to the dual LP for matroid intersection, (2). The dual LP has a constraint for each element, so the MWU framework maintains a weight for each element reflecting how well the constraint is being met. The oracle problem in Lemma 7 is a relaxation of (2) where we are only required to satisfy most, rather than all, of the dual covering constraints by weight. To solve the oracle problem quickly, we first randomly sample $O(k\log(n)/\epsilon)$ elements in proportion to their weights. Then we compute a $(1-\epsilon)$-approximate matroid intersection, and a $(1+\epsilon)$-approximate dual solution over the sampled elements. We then extend the dual solution to an infeasible dual solution over all the elements, that satisfies the oracle.

The MWU framework has $O(\log(n)/\epsilon)$ iterations. Each iteration has two bottlenecks. The first bottleneck comes from sampling and updating the weights of each element, and takes $O(nQ)$ time. The second bottleneck is approximating the matroid intersection and its dual over the sampled set of elements. If we let $\mathcal{T}_\epsilon(m)$ denote the time of this step, then the algorithm takes $O(n\log(n)/\epsilon + \mathcal{T}_\epsilon(k\log(n)/\epsilon)\log(n)/\epsilon)$ randomized time overall.

To complete the proof of Theorem 3, there are two points to address:

Section 2.1: Given an $(1\pm\epsilon)$-primal dual oracle for matroid intersection running in $\mathcal{T}_\epsilon(m)$ time, we implement the oracle from Lemma 7 in $O(nQ + \mathcal{T}_\epsilon(k\log(n)/\epsilon))$ randomized time. This implies Lemma 1, which formalizes the reduction to approximate primal-dual matroid intersection.

Section 2.2: Recall that the framework requires solutions to both matroid intersection and its dual LP. We extend the $(1-\epsilon)$-maximum matroid intersection algorithm for [4] to give a $(1+\epsilon)$-dual solution without increasing the running time. This gives $\mathcal{T}_\epsilon(m) = \tilde{O}\big(m^{1.5}Q/\epsilon\big)$, hence Lemma 2, and completes the proof of Theorem 3 via Lemma 1.

2.1 Implementing the oracle

In this section, we assume access to a $(1\pm\epsilon)$-primal dual approximation algorithm for matroid intersection in $\mathcal{M}_1$ and $\mathcal{M}_2$, running in $\mathcal{T}_\epsilon(m)$ time for any subset of m elements. We show how to use this algorithm to implement an oracle satisfying Lemma 7 for the dual LP of matroid intersection.

Recall that the algorithm takes as input w, samples $\tilde{O}(k/\epsilon)$ elements $\mathcal{N}' \subseteq \mathcal{N}$ in proportion to w, computes $(1\pm\epsilon)$-primal and dual solutions over $\mathcal{N}'$, and expands out the dual solution by taking their spans in all of $\mathcal{N}$. The key point of the analysis is understanding the random sample $\mathcal{N}'$. We want to ensure that any approximate dual solution (S', T') over $\mathcal{N}'$, when expanded out to (S, T) where $S = \mathrm{span}_1(S')$ and $T = \mathrm{span}_2(T')$, covers most of $\mathcal{N}$ by weight. The approach is based on [2] and will use an upper bound the number of distinct pairs (S, T) with certain nice properties. The first step in this direction is to count the number of closed sets in a given matroid.

▶ **Lemma 8.** *Let $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ be a matroid with n elements and rank k. Then there are at most n^k closed sets in $\mathcal{M}$.*

Proof. For every closed set S, let I_S be a maximum independent subset of S. We have, $I_S \in \mathcal{I}$, $I_S \subseteq S$, and $\mathrm{span}(I_S) = \mathrm{span}(S)$. We claim the mapping from S to I_S is injective. Indeed, if S and T are closed and $I_S = I_T$, then $S = \mathrm{span}(S) = \mathrm{span}(I_S) = \mathrm{span}(I_T) = \mathrm{span}(T) = T$, so $S = T$. Thus the sets I_S are distinct. Meanwhile, every $I \in \mathcal{I}$ has cardinality at most k. So there are at most n^k closed sets. ◀

The following lemma shows that with high probability, any closed pair of sets (S, T) (with S closed in $\mathcal{M}_1$ and T closed in $\mathcal{M}_2$) either covers almost all of $\mathcal{N}$ by weight, or with high probability, does not cover at least one element in the random sample $\mathcal{N}'$.

▶ **Lemma 9.** *Let $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$ be a set of nonnegative weights. Let $\mathcal{N}'$ sample $O(k \log(n)/\epsilon)$ elements from $\mathcal{N}$ with repetition. Then with high probability we have the following: for all $A, B \subseteq \mathcal{N}$ such that A is closed in $\mathcal{M}_1$, B is closed in $\mathcal{M}_2$, $\operatorname{rank}_1(A) = \operatorname{rank}_2(B) \leq O(k)$, and $w(A \cup B) < (1-\epsilon)w(\mathcal{N})$, $\mathcal{N}'$ samples at least one element outside $A \cup B$.*

Proof. Since $\operatorname{rank}_1(A), \operatorname{rank}_2(B) \leq O(k)$, we can assume that $\mathcal{M}_1$ and $\mathcal{M}_2$ each have rank $O(k)$. Then, by Lemma 8, there are at most $n^{O(k)}$ choices of sets $A, B \subseteq \mathcal{N}$ such that A is closed in $\mathcal{M}_1$ and B is closed in $\mathcal{M}_2$.

Now fix such a pair A, B, and suppose $w(A \cup B) \leq (1-\epsilon)w(\mathcal{N})$. The probability that $\mathcal{N}' \subseteq A \cup B$ is

$$\left(\frac{w(A \cup B)}{w(\mathcal{N})}\right)^{|\mathcal{N}'|} \leq e^{-\epsilon|\mathcal{N}'|} \leq n^{-Ck}$$

for an arbitrarily large constant C. The claim now follows by taking the union bound over A, B. ◀

Now we put everything together. The following lemma gives a subroutine satisfying the requirements of the oracle in Lemma 7.

▶ **Lemma 10.** *Let $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$ be a set of nonnegative weights.*
In $O(n \log(n) Q + \mathcal{T}_\epsilon(k \log(n)/\epsilon))$ randomized time, one can compute an independent set I and sets S closed in $\mathcal{M}_1$ and T closed in $\mathcal{M}_2$ such that, with high probability:

- **i** $\operatorname{rank}(S) + \operatorname{rank}(T) \leq (1+\epsilon)|I|$.
- **ii** $w(S \cup T) \geq (1-\epsilon)w(\mathcal{N})$.

Proof. For ease of convention, we prove the claim with ϵ replaced by $O(\epsilon)$; the constant can then be removed by decreasing ϵ by a constant factor. (We adopt the same convention in subsequent proofs.)

Let $\mathcal{N}'$ sample $O(k \log(n)/\epsilon)$ elements with repetition in proportion to w. In $\mathcal{T}_\epsilon(k \log(n)/\epsilon)$ time, we compute a $(1-\epsilon)$-maximum matroid intersection I and a $(1+\epsilon)$-dual solution (A', B') in the restriction to $\mathcal{N}'$. We compute $A = \operatorname{span}_1(A')$ and $B = \operatorname{span}_2(B')$ in $O(nQ)$ time. We claim that I, A, and B satisfy the lemma.

First, we have $\operatorname{rank}_1(A) + \operatorname{rank}_2(B) = \operatorname{rank}_1(A') + \operatorname{rank}_2(B') \leq (1 + O(\epsilon))|I|$. Second, by Lemma 9, with high probability; since $\mathcal{N}' \subseteq A' \cup B' \subseteq A \cup B$, A is closed in $\mathcal{M}_1$, and B is closed in $\mathcal{M}_2$; we have $w(A \cup B) \geq (1-\epsilon)w(\mathcal{N})$, as desired. ◀

2.2 Fast primal-dual approximations for matroid intersection

It remains to give a fast primal-dual approximation for matroid intersection. The algorithm in [4] gives a $(1-\epsilon)$-maximum matroid intersection but not a $(1+\epsilon)$-dual solution. That said, one might expect a $(1+\epsilon)$-dual solution to be implicit in any given $(1-\epsilon)$-maximum matroid intersection algorithm in order to certify that the solution I is $(1-\epsilon)$-maximum. Such is the case here and we show how to extract a $(1+\epsilon)$-dual solution efficiently.

The $(1-\epsilon)$-maximum matroid intersection algorithm in [4], like other $(1-\epsilon)$-approximation algorithms, outputs an independent set $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ for which the length of the minimum "augmenting path" (defined in a moment) is at least $2/\epsilon$. This length implies that I is a $(1-\epsilon)$-maximum matroid intersection (just as in bipartite matching). We use this bound on the minimum length of an augmenting bound to extract a $(1+\epsilon)$-dual solution.

To go into further detail we must first introduce augmenting paths in the context of matroid intersection. The notion generalizes augmenting paths in bipartite matching. In the context of matroid intersection, for a fixed independent set I, an augmenting path is a sequence $P = (e_0, e_1, \dots, e_h)$ of elements alternating between $\mathcal{N} \setminus I$ and I. Additionally, an augmenting path must start and end with elements in $\mathcal{N} \setminus I$, and its symmetric difference with I, $I \triangle P$, must be independent in both matroids.

Augmenting paths are paths in an auxiliary directed bipartite graph between $\mathcal{N} \setminus I$ and I called the *exchange graph*. We have a directed edge (e, d) from $\mathcal{N} \setminus I$ to I iff $I - d + e \in \mathcal{I}_2$. We have a directed edge (d, e) from I to $\mathcal{N} \setminus I$ iff $I - d + e \in \mathcal{I}_1$.

Let $F_1 = \mathcal{N} \setminus \text{span}_1(I)$ be the set of free/uncovered elements in $\mathcal{M}_1$, and $F_2 = \mathcal{N} \setminus \text{span}_2(I)$ be the free elements in $\mathcal{M}_2$. All augmenting paths are between F_1 and F_2, but unlike bipartite matching, not all paths from F_1 to F_2 in the exchange graph are augmenting paths. However, all shortest paths from F_1 to F_2 are always augmenting paths. Thus many matroid intersection algorithms augment along shortest (F_1, F_2)-paths in the exchange graph.

If the minimum length of any augmenting path for I is at least $2/\epsilon$, then the typical argument that I is $(1-\epsilon)$-maximum is as follows. Given an optimal solution I^*, by contracting $I \cap I^*$, we may assume I^* and I are disjoint. $I \triangle I^*$ decomposes into even-length cycles and augmenting paths in the exchange graph. I and I^* have the same number of elements in each even-length cycle. For an augmenting path P, if P has length at least $2/\epsilon$, then $|I^* \cap P| \leq (1+\epsilon)|I \cap P|$. It follows that $|I^*| \leq (1+\epsilon)|I|$.

This argument does not imply an algorithm for a $(1+\epsilon)$-dual solution. (We do not have access to I^*.) Still, we can use the length bound to obtain a $(1+\epsilon)$-dual solution, giving another proof that I is $(1-\epsilon)$-maximum. Formally we prove the following.

▶ **Lemma 11.** *Let $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ be an independent point in the intersection. Suppose the minimum length of any augmenting path is at least $2/\epsilon$. Then in $O(n \log(k) Q/\epsilon)$ time, one can compute sets $S, T \subseteq \mathcal{N}$ such that $S \cup T = \mathcal{N}$ and $\text{rank}_1(S) + \text{rank}_2(T) \leq (1+\epsilon)|I|$.*

Proof. The desired sets S and T will be induced by the distances layers from F_1 in the exchange graph. For each index $i \in \mathbb{Z}_{\geq 0}$, let L_i be the set of elements at distance i from F_1. For example, $L_0 = F_1$, $L_i \subseteq \mathcal{N} \setminus I$ for even i, and $L_i \subseteq I$ for odd i.

To extract a good dual solution from these layers, we first need to construct them quickly. [7, 23] provide the following.

▶ **Fact 12.** *For $h = O(1/\epsilon)$, the first h layers $L_0, L_1, \dots, L_{h-1}$ of the exchange graph can be computed in $O(n \log(k) Q/\epsilon)$ time.*

To refer to elements in I and $\mathcal{N}$ by distance layer, we introduce the following notation. For all i, let $\mathcal{N}_i = \mathcal{N} \cap L_i$, and for odd i, let $\mathcal{I}_i = \mathcal{I} \cap L_i$. These layers partition the ground set, with the even layers partitioning $\mathcal{N} \setminus I$ and the odd layers partitioning I.

Suppose we construct the layers $(L_0 = F_1), \dots, L_h$ up to distance h from F_1 for $h \geq 2/\epsilon$. Since $h \geq 2/\epsilon$, there is an odd index i with $1 \leq i \leq h$ such that $|I_i| \leq \epsilon |I|$. Let $S = \mathcal{N} \setminus \left(\bigcup_{j<i} \mathcal{N}_i \right)$ be all elements in layer i and beyond, and $T = \bigcup_{j \leq i} \mathcal{N}_j$ be all the elements up to layer i. (S, T) is a discrete and feasible solution to the dual because $S \cup T = \mathcal{N}$. To prove that it is a $(1+\epsilon)$-dual solution, it suffices to show that $\text{rank}_1(S) + \text{rank}_2(T) \leq (1+\epsilon)|I|$.

Let $I^+ = I \setminus \left(\bigcup_{j<i} I_j \right) = I \cap S$. We claim that $S \subseteq \text{span}_1(I^+)$. Clearly $S \cap I = I^+ \subseteq \text{span}_1(I^+)$. Now consider an element $e \in S \setminus I$. We have $e \in \text{span}_1(I)$ because $e \notin \mathcal{N}_0$. If $e \notin \text{span}_1(I^+)$, then $I - d + e \in \mathcal{I}_1$ for some $d \in I \setminus I^+$. Then (d, e) is an exchange in the exchange graph. We have $d \in I_j$ for some $j \leq i - 2$, hence $d \in \mathcal{N}_\ell$ for some $\ell < i$. But then $e \notin S$, a contradiction.

Now let $I^- = \bigcup_{j \le i} I_j = I \cap T$. We claim that $T \subseteq \text{span}_2(I^-)$. We have $T \cap I \subseteq \text{span}_2(I^-)$. Consider any element $e \in T \setminus I$. We have $e \in \text{span}_2(I)$ since $e \in L_j$ for some $j < i \le h$, and there are no (F_1, F_2)-paths of length $< h$. If $e \notin \text{span}_2(I^-)$, then $I - d + e \in \mathcal{I}_2$ for some $d \in I \setminus I^-$. That is, (e, d) is an edge in the exchange graph. $e \in T$ implies that the distance from F_1 to e is at most $i - 1$, and the distance from F_1 to d is at most i. But $d \notin I^-$ implies that the distance from F_1 to d is at least $i + 2$, a contradiction.

Thus $S \subseteq \text{span}_1(I^+)$ and $T \subseteq \text{span}_2(I^-)$ where I^+ and I^- are subsets of I overlapping on I^-. Since $I^+ \cap I^- = I_i$, we have

$$\begin{aligned}\text{rank}(S) + \text{rank}(T) &\le \text{rank}(\text{span}_1(I^+)) + \text{rank}(\text{span}_2(I^-)) \\ &= |I^+| + |I^-| = |I| + |I_i| \le (1+\epsilon)|I|,\end{aligned}$$

as desired.

To recap, given an independent set $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ for which the length of the minimum augmenting path is at least $2/\epsilon$, we build out the first $O(1/\epsilon)$ layers of the exchange graph in $O(n \log(k) Q/\epsilon)$ time. We identify an index i such that $|I_i| \le \epsilon |I|$. We assemble the set $S = \mathcal{N} \setminus (L_0 \cup L_1 \cup \cdots \cup L_{i-1})$ of all elements in the ith layer and beyond, and the set $T = L_0 \cup L_1 \cdots \cup L_i$ of all elements up to the ith layer. This takes $O(n)$ time given the first $O(1/\epsilon)$ layers. (S, T) is the desired $(1+\epsilon)$-dual solution, completing the proof of Lemma 11. ◀

This concludes our discussion on maximum cardinality matroid intersection.

3 Weighted matroid intersection

We now consider the weighted matroid intersection problem. Similar to the unweighted setting, the high-level approach combines adaptive sparsification with a fast $(1 \pm \epsilon)$-primal-dual approximation algorithm to improve the running time dependence on the input n. These components are more technically challenging in the weighted setting.

Recall that $\mathcal{M}_1 = (\mathcal{N}, \mathcal{I}_1)$ and $\mathcal{M}_2 = (\mathcal{N}, \mathcal{I}_2)$ are two matroids over the same ground set $\mathcal{N}$, $c : \mathcal{N} \to \mathbb{R}_{\ge 1}$ is a set of input costs over $\mathcal{N}$, and $\epsilon \in (0, 1)$ is an input parameter.[2] The goal is to compute $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ with cost $c(I)$ at least a $(1 - \epsilon)$-fraction of the maximum cost of any such I. We assume without loss of generality that $c(e) \in [1, \text{poly}(n)]$ for all e.[3]

We start by introducing the LP relaxation of weighted matroid intersection:

$$\begin{aligned}&\text{maximize } \langle c, x \rangle \text{ over } x \in \mathbb{R}^E_{\ge 0} \\ &\quad \text{s.t. } x(S) \le \text{rank}_1(S) \text{ and } x(S) \le \text{rank}_2(S) \text{ for all } S \subseteq \mathcal{N}.\end{aligned} \tag{3}$$

The dual LP is as follows.

$$\begin{aligned}&\text{minimize } \sum_S \text{rank}_1(S) y_S + \text{rank}_2(S) z_S \text{ over } y, z : 2^{\mathcal{N}} \to \mathbb{R}_{\ge 0} \\ &\quad \text{s.t. } \sum_{S : e \in S} y_S + z_S \ge c_e \text{ for all } e \in \mathcal{N}.\end{aligned} \tag{4}$$

For $e \in \mathcal{N}$, we say that (y, z) covers e if it meets the covering constraint for e in the dual LP (4).

---

[2] c is normally called "weights" in the weighted matroid intersection problem, but we will refer to them as "costs" to help distinguish them from the auxiliary weights generated by the framework.

[3] We assume each singleton set $\{e\}$ is independent for all e by removing all violating e in a preprocessing step. Then $\text{OPT} \ge \max_e c(e)$, and we can drop any element e with $c(e) \le (\epsilon/k) \max_f c(f)$ without decreasing the optimum value by more than an ϵ-fraction. Rescaling, all weights lie in the range $[1, k/\epsilon]$.

Recall that the overall framework maintains auxiliary weights $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$. For a fixed set of weights $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$, we say that y, z covers a $(1-\epsilon)$-fraction of elements by w if the total weight of elements covered by e is at least $(1-\epsilon)$-fraction of the total weight of all elements.

To apply Lemma 7 to the LPs (3) and (4), we need to fulfill an oracle problem defined as follows. The oracle takes as input a set of auxiliary weights $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$. With high probability, the oracle must return a dual solution y, z that covers a $(1-\epsilon)$-fraction of elements by weight.

Our implementation of the oracle is broken down into two components: (a) a randomized sparsification step applying a $(1 \pm \epsilon)$-primal-dual approximation algorithm to a randomly sampled subset of $\tilde{O}(k \log(n)/\epsilon)$ elements; and (b) the $(1 \pm \epsilon)$-primal-dual approximation oracle. Formally stating the interface of these two parts requires the notion of "compact" dual solutions and compact primal-dual algorithms. We define these now and give further background in appropriate subsections later.

We say that (y, z) is *compact* if the supports of y and z are of the form

$$\text{support}(y) \subseteq \{\text{span}_1(e_1), \text{span}_1(e_1, e_2), \ldots, \text{span}_1(e_1, \ldots, e_k)\}$$
$$\text{support}(z) \subseteq \{\text{span}_2(f_1), \text{span}_2(f_1, f_2), \ldots, \text{span}_2(f_1, \ldots, f_k)\}$$

for two sequences of k elements $e_1, \ldots, e_k \in \mathcal{N}$ and $f_1, \ldots, f_k \in \mathcal{N}$. We define a *compact* $(1 \pm \epsilon)$*-primal-dual weighted matroid intersection* algorithm that takes as input $\mathcal{M}_1 = (\mathcal{N}, \mathcal{I}_1)$, $\mathcal{M}_2 = (\mathcal{N}, \mathcal{I}_2)$, and $c : \mathcal{N} \to \mathbb{R}_{\geq 0}$, and returns $(1-\epsilon)$-maximum weight independent set and a compact $(1+\epsilon)$-minimum solution to the dual LP (4).

Now we can state the guarantees of part (a).

▶ **Lemma 13.** *Let $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$, and suppose there is a compact $(1 \pm \epsilon)$-primal-dual weighted matroid intersection algorithm running in $\mathcal{T}_\epsilon(m)$ on subsets of $\mathcal{N}$ of size m. Then in $O(\mathcal{T}_\epsilon(k \log(n)/\epsilon))$ time, one can compute an independent set I and a compact dual solution (y, z) such that (y, z) covers a $(1-\epsilon)$-fraction of $\mathcal{N}$ by w.*

Now we formally state the guarantees for part (b); namely, a compact $(1 \pm \epsilon)$-primal-dual approximation algorithm for weighted matroid intersection. The following algorithm extends the $(1-\epsilon)$-approximation for weighted matroid intersection to also give a compact dual solution.

▶ **Lemma 14.** *There is a compact $(1 \pm \epsilon)$-primal dual algorithm running in $\mathcal{T}_\epsilon(m) = \tilde{O}\left(m\sqrt{k}Q/\epsilon^2\right)$ time.*

We prove Lemma 13 in Section 3.1 and Lemma 14 in Section 3.2. Combining Lemmas 13 and 14 gives a subroutine implementing the oracle of Lemma 7 running in $\tilde{O}\left(nQ/\epsilon + k^{1.5}/\epsilon^3\right)$ time. Because the dual solution (y, z) returned by the oracle is compact, we can identify all the elements covered by (y, z) in $O(n \log(k)Q)$ time.[4] Lemma 7 repeats these two steps for $O(\log(n)/\epsilon)$ iterations, giving a total running time of $O\left(n \log(n)Q/\epsilon + k^{1.5} \log^{O(1)}(n)Q/\epsilon^4\right)$.

[4] For example, suppose the support of y is generated by the elements $e_1, \ldots, e_k$. Given an element d, one can binary search for the first index i such that $d \in \text{span}_1(e_1, \ldots, e_i)$. Then d is covered by $\text{span}_1(e_1, \ldots, e_j)$ for all $j \geq i$. With simple preprocessing, one then extracts the total amount that y covers d in $O(1)$ time. Similarly the amount that z covers d can be computed in $O(\log(k)Q)$ time with simple preprocessing.

3.1 Sparse reduction to compact $(1 \pm \epsilon)$-primal-dual approximations: proof of Lemma 13

In this section we prove Lemma 13. Let $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$. We need to compute an independent set $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ and a compact $y, z : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ such that:

(i) $(1 + O(\epsilon))c(I) \geq \sum_S \operatorname{rank}_1(S) y_S + \operatorname{rank}_2(S) z_S$.

(ii) (y, z) covers a $(1 - \epsilon)$-fraction of $\mathcal{N}$ by w.

The algorithm is as follows.

1. Let $\mathcal{N}' \subseteq \mathcal{N}$ sample $O(k \log(n)/\epsilon)$ elements, with repetition, in proportion to w.
2. Run the $(1 \pm \epsilon)$-primal dual approximation algorithm on $\mathcal{N}'$, producing $I \subseteq \mathcal{N}'$ with $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ and a compact solution $y', z' : 2^{\mathcal{N}'} \to \mathbb{R}_{\geq 0}$. Without loss of generality, the nonzero coordinates of y' and z' are in the range $[1/\operatorname{poly}(n), \operatorname{poly}(n)]$.
3. Define $y, z : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ by setting $y(\operatorname{span}_1(S)) = \lceil y'(S) \rceil_{1+\epsilon}$ for $S \in \operatorname{support}(y')$ and similarly set $z(\operatorname{span}_2(T)) = \lceil z'(T) \rceil_{1+\epsilon}$ for $T \in \operatorname{support}(z')$, where $\lceil x \rceil_{1+\epsilon} \stackrel{\text{def}}{=} (1+\epsilon)^{\lceil \log_{1+\epsilon} x \rceil}$ rounds x up to the nearest power of $1 + \epsilon$.
4. Return I and (y, z).

Consider the range of (y, z) output by the algorithm. We first observe that (y, z) is compact. Indeed, since (y', z') was compact, there is a sequence of elements $e_1, \ldots, e_k$ such that all sets in the support of y' have the form $S'_i = \operatorname{span}_1(e_1, \ldots, e_i) \cap \mathcal{N}'$ for some prefix $e_1, \ldots, e_i$. Then the sets in the support of y have the form $S_i = \operatorname{span}_1(S'_i) = \operatorname{span}_1(e_1, \ldots, e_i)$. Thus the support of y is generated by prefixes of $e_1, \ldots, e_k$. Symmetrically the same holds for z for a different sequence of k elements depending on z'. Thus (y, z) is compact.

We also observe that each nonzero value of y or z is one of the $O(\log(n)/\epsilon)$ powers of $(1 + \epsilon)$ in the range $[1/\operatorname{poly}(n), \operatorname{poly}(n)]$.

For y, there are at most n^k ways to choose the sequence $e_1, \ldots, e_k$ that determines its support, and $(C \log(n)/\epsilon)^k$ ways to assign their values for some constant $C > 0$. Likewise for z. Altogether, there are at most $n^{O(k)}$ choices of (y, z) in the range of the algorithm.

Call a solution (y, z) in the range *good* if it covers an $(1 - \epsilon)$-fraction of $\mathcal{N}$ by w, and *bad* otherwise. We want to argue that with high probability, the solution (y, z) returned by the algorithm is good. We know that the output (y, z) covers all the elements in $\mathcal{N}'$. Thus it suffices to show that for all bad (y, z) in the range, $\mathcal{N}'$ samples at least one element that is not covered by (y, z).

To this end, fix a bad (y, z) in the range. A random element sampled from $\mathcal{N}$ in proportion to w is covered by (y, z) with probability at most $1 - \epsilon$. The probability that all of $\mathcal{N}'$ is covered by (y, z) is bounded above by $(1 - \epsilon)^{|\mathcal{N}'|} = n^{-O(k)}$. Taking the union bound over all bad (y, z) in the range, we conclude with high probability, $\mathcal{N}'$ samples at least one uncovered element for every bad (y, z). In this event, since the output (y, z) covers $\mathcal{N}'$, (y, z) must be good.

This completes the proof of Lemma 13.

3.2 Compact $(1 \pm \epsilon)$-primal-dual weighted matroid intersection: proof of Lemma 14

In this section we prove Lemma 14. We are given two matroids $\mathcal{M}_1 = (\mathcal{N}, \mathcal{I}_1)$ and $\mathcal{M}_2 = (\mathcal{N}, \mathcal{I}_2)$ over a common set $\mathcal{N}$ of size n, $c : \mathcal{N} \to \mathbb{R}_{\geq 1}$, and $\epsilon \in (0, 1)$. We let k be the rank of the intersection $\mathcal{I}_1 \cap \mathcal{I}_2$. We will compute $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ and a *compact* dual solution $y, z : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ such that $(1 + O(\epsilon))c(I) \geq \sum_S \operatorname{rank}_1(S) y(S) + \operatorname{rank}_2(S) z(S)$.

As mentioned previously, a fast approximation algorithm for weighted matroid intersection is given by [8]. The stated running time is $\tilde{O}\left(nkQ/\epsilon^2\right)$ and breaks down as follows. There are $\tilde{O}(1/\epsilon)$ outer iterations. Each iteration invokes an approximation algorithm for unweighted matroid intersection that returns an independent set I with minimum augmenting path length $O(1/\epsilon)$. [8] truncate Cunningham's algorithm to execute an inner iteration in $\tilde{O}(nkQ/\epsilon)$ time. The subroutine using Cunningham's algorithm can be replaced by the $\tilde{O}\left(n\sqrt{k}Q/\epsilon\right)$-time algorithm of [4].[5] This gives a $(1-\epsilon)$-approximation algorithm for weighted matroid intersection running in $\tilde{O}\left(n\sqrt{k}Q/\epsilon^2\right)$ time.

The real challenge is to also compute a compact $(1+\epsilon)$-approximate dual solution within the same running time. This requires additional background on the framework of [8].

Cost-splitting. The fast approximation algorithm in [8] is based on the cost-splitting approach to matroid intersection described in [14]. A *cost-splitting* of c is a decomposition $c_1 + c_2$ where $c_1, c_2 : \mathcal{N} \to \mathbb{R}_{\geq 0}$.[6] A cost-splitting can be used to certify a maximum cost independent set as follows.

▶ **Fact 15** ([14]). *Suppose $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ and $c = c_1 + c_2$ is a cost-splitting such that I is a c_1-maximum independent set in $\mathcal{M}_1$ and c_2-maximum independent set in $\mathcal{M}_2$. Then I is a maximum cost independent set in the intersection of $\mathcal{M}_1$ and $\mathcal{M}_2$.*

The proof is immediate: given (I, c_1, c_2) as in fact 15, and letting I^* denote an optimum solution, we have $c_1(I) \geq c_1(I^*)$ and $c_2(I) \geq c_2(I^*)$. Since $c = c_1 + c_2$, $c(I) \geq c(I^*)$. Note that this proof does not involve the dual LP.

An approximate version of fact 15 is given in [8] to certify $(1-\epsilon)$-approximate solutions. This proof also does not construct a dual solution, let alone compact one.

Before describing how to extract the desired dual solution from [8], we give additional background on the LPs Equations (3) and (4). This background is not necessary for our analysis, but it gives some intuition for the eventual claim.

[10] proved that LP (3) is totally dual integral. Moreover, any optimal solution (y, z) can be uncrossed and merged so that the supports of y and z each form a chain. Expanding on the latter point, suppose the support of y is a chain of the form $S_1 \subsetneq S_2 \subsetneq \cdots \subsetneq S_\ell$. We can replace each S_i with $\text{span}(S_i)$ without increasing the objective or decreasing the coverage on any element. Thus, replacing each S_i with $\text{span}(S_i)$, we can assume each S_i is closed. We have $0 < \text{rank}_1(S_1) < \text{rank}_1(S_2) < \cdots < \text{rank}_1(S_\ell) \leq \text{rank}_1(\mathcal{N})$. Let $I_0 = \emptyset$, and for $i = 1, 2, \ldots, \ell$ in sequence, let I_i extend I_{i-1} to a maximum cardinality independent set of S_i. Then $S_i = \text{span}(I_i)$ for each i. Let $e_1, \ldots, e_h$ enumerate the elements of I_1, then $I_2 \setminus I_1$, and so forth, so that each I_i is a prefix of the form $I_i = \{e_1, \ldots, e_{j_i}\}$ for some index $j_i \in [h]$. Since $I_\ell = \{e_1, \ldots, e_h\} \in \mathcal{I}_1$, $h \leq \text{rank}_1(\mathcal{N})$.

This exercise shows that if the support of y is a chain, then it is induced by prefixes of a sequence of $h \leq \text{rank}_1(\mathcal{N})$ elements $e_1, \ldots, e_h$. Symmetrically if the support of z is a chain, then it is also generated by a sequence of $\text{rank}_2(\mathcal{N})$ elements. Consequently there are at most $n^{\text{rank}_1(\mathcal{N}) + \text{rank}_2(\mathcal{N})}$ ways for y and z to have chain supports.

[5] A minor technical point to address is that the unweighted matroid intersection is not over $\mathcal{M}_1$ and $\mathcal{M}_2$, but auxiliary "weight-induced matroids" $\mathcal{M}_{1,c_1}$ and $\mathcal{M}_{2,c_2}$ induced by weights c_1 and c_2. Fortunately it is easy to identify edges of the exchange graph of $\mathcal{M}_{1,c_1}$ and $\mathcal{M}_{2,c_2}$ via independence oracles to the input matroids $\mathcal{M}_1$ and $\mathcal{M}_2$. The algorithm in [4] can be adapted to $\mathcal{M}_{1,c_1}$ and $\mathcal{M}_{2,c_2}$ just as Cunningham's algorithm was adapted in [8].

[6] [14] calls this a "weight-splitting". We refer to this as a "cost-splitting" because we are referring to c as costs.

Thus a compact solution (y, z) is similar to a solution (y, z) where the support is a chain. However, compact solutions restrict the generating sequences of elements to have length at most k, and $k \leq \min\{\text{rank}_1(\mathcal{N}), \text{rank}_2(\mathcal{N})\}$.

As alluded to earlier, [8] computes a $(1-\epsilon)$-approximation I along with vectors c_1, c_2 that approximate the (exact) cost-splitting describe in [14]. We formalize the approximation conditions as follows: For fixed I and $\epsilon \in (0, 1)$, an *ϵ-approximate cost-splitting certificate* is a pair of weight vectors $c_1, c_2 : \mathcal{N} \to \mathbb{R}_{\geq 0}$ with the following properties:

(a) $(1+\epsilon)(c_1 + c_2) \geq c$ elementwise.
(b) $c_1(I) + c_2(I) \leq (1+\epsilon)c(I)$.
(c) I is a c_1-maximum independent set in $\mathcal{M}_1$ restricted to $\text{span}_1(I)$.
(d) I is a c_2-maximum independent set in $\mathcal{M}_2$ restricted to $\text{span}_2(I)$.
(e) $c_1(e) \leq \epsilon$ for all $e \in \mathcal{N} \setminus \text{span}_1(I)$.
(f) $c_2(e) \leq \epsilon$ for all $e \in \mathcal{N} \setminus \text{span}_2(I)$.

[8] produces an independent set $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ and an ϵ-approximate cost-splitting certificate c_1, c_2. We show how to extract a compact $(1+\epsilon)$-approximate dual solution from I, c_1, and c_2.

▶ **Lemma 16.** *Let $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ and let $c_1, c_2 : \mathcal{N} \to \mathbb{R}_{\geq 0}$ be an ϵ-approximate cost-splitting certificate. Then in $\tilde{O}(nQ)$ time, one can compute $y, z : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ such that*

(i) *(y, z) are feasible for the dual LP (4).*
(ii) *(y, z) are compact.*
(iii) $\sum_S \text{rank}_1(S) y_S + \text{rank}_2(S) y_S \leq (1 + O(\epsilon))c(I)$

The last point says that I and (y, z) mutually certify each other to be $(1 \pm O(\epsilon))$-approximations to their respective problem. Taking ϵ down to 0 gives an exact compact dual optimum solution certifying I to be exactly optimum.

Proof of Lemma 16. For $t \in \mathbb{R}_{\geq 0}$, let

$$Y_t = \text{span}_1(\{e \in I : c_1(e) \geq t\}) \text{ and } Z_t = \text{span}_2(\{e \in I : c_2(e) \geq t\}).$$

Let $y, z : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ be defined by

$$y = (1 + O(\epsilon)) \int_0^\infty 1_{Y_t}\, dt \text{ and } z = (1 + O(\epsilon)) \int_0^\infty 1_{Z_t}\, dt,$$

where 1_S denotes the indicator vector for $S \subseteq \mathcal{N}$ in $\mathbb{R}^{2^{\mathcal{N}}}$. Observe that y is supported by a chain of elements listing I in decreasing order of c_1, and z is supported by a chain of elements listing I in decreasing order of c_2. It remains to show that y, z is feasible for the dual LP (4), and has objective value within a $(1 + O(\epsilon))$-factor of $c(I)$.

To show that (y, z) is feasible, fix $e \in \mathcal{N}$. If $c_1(e) \geq \epsilon$, then $e \in \text{span}_1(I)$. Since I is c_1-maximum in $\mathcal{M}_1$, $e \in \text{span}(Y_t)$ for all $t \leq c_1(e)$. Therefore,

$$\sum_{S : e \in S} y_S \geq (1 + O(\epsilon)) \int_0^{c_1(e)} 1\, dt = (1 + O(\epsilon))c_1(e).$$

Symmetrically, if $c_2(e) \geq \epsilon$, then $\sum_{S : e \in S} z_S \geq (1 + O(\epsilon))c_2(e)$.

Now we have three cases. If $c_1(e) \geq \epsilon$ and $c_2(e) \geq \epsilon$, then

$$\sum_{S : e \in S} y_S + z_S \geq (1 + O(\epsilon))c_1(e) + (1 + O(\epsilon))c_2(e) \geq c(e).$$

If $c_1(e) \le \epsilon$, then $c_1(e) \le \epsilon c(e)$, and

$$\sum_{S:e\in S} y_S + z_S \ge \sum_{S:e\in S} z_S \ge (1+O(\epsilon))c_2(e) \ge (1+O(\epsilon))((1-\epsilon)c(e) - c_1(e))$$
$$\ge (1+O(\epsilon))(1-2\epsilon)c(e) \ge c(e).$$

Symmetrically, if $c_2(e) \le \epsilon$, then $\sum_{S:e\in S} y_S + z_S \ge (1-2\epsilon)c(e)$. Thus (y,z) forms a feasible dual solution if $c_1(e) \le \epsilon$, $c_2(e) \le \epsilon$, or neither.

Now consider the objective value. The contribution from y is

$$\sum_S \operatorname{rank}_1(S) y_S = (1+O(\epsilon)) \int_0^\infty \operatorname{rank}(Y_t)\, dt$$
$$= (1+O(\epsilon)) \int_0^\infty |Y_t \cap I|\, dt = (1+O(\epsilon))c_1(I).$$

Symmetrically, z contributes $\sum_S \operatorname{rank}_2(S) z_S = (1+O(\epsilon))c_2(I)$. Together, we have

$$\sum_S \operatorname{rank}_1(S) y_S + \operatorname{rank}_2(S) z_S = (1+O(\epsilon))(c_1(I)+c_2(I)) \le (1+O(\epsilon))I,$$

as desired. ◀

Now we complete the proof of Lemma 13. Using the algorithm of [8] with the $(1-\epsilon)$-approximated unweighted matroid intersection algorithm of [4] as a subroutine, we compute an $(1-\epsilon)$-approximate maximum cost independent set I and a ϵ-approximate cost-splitting certificate c_1, c_2, in $\tilde{O}\left(n\sqrt{k}Q/\epsilon^2\right)$ time. By Lemma 16, we extract a feasible compact $(1+\epsilon)$-dual solution (y,z) in $\tilde{O}(nQ/\epsilon)$ time. We return I and (y,z). The overall running time is $\tilde{O}\left(nQ/\epsilon + k^{3/2}/\epsilon^3\right)$. This completes the proof of Lemma 13 and of the faster approximate matroid intersection algorithm.

4 MWU analysis

In this section we prove Lemma 7, which claims that given a covering LP, solving a certain relaxed coverage problem for $\tilde{O}(1/\epsilon)$ iterations leads to a $(1-\epsilon)$-approximation overall.

Like other proofs, we fix ϵ, and describe and analyze an algorithm that returns a point $x \in \mathcal{P}$ with objective value $\langle b, x\rangle \le (1+O(\epsilon))\,\mathrm{OPT}$ and coverage $Ax \ge (1-O(\epsilon))\mathbb{1}$. The claimed $(1\pm\epsilon)$ bounds then follow by decreasing ϵ by a constant factor.

There are multiple perspectives on the MWU framework presenting essentially the same proof in different styles. Our presentation follows the conventions of the analysis from [24].

The algorithm builds a solution x incrementally over $L = O(\log(m)/\epsilon)$ iterations. We let $x^{(\ell)}$ denote the value of x after ℓ iterations. It starts with $x^{(0)} = \mathbb{0}$. Each iteration ℓ invoke the oracle for a set of weights $w^{(\ell)}$ and produces a point $y^{(\ell)}$, and updates $x^{(\ell)} = x^{(\ell-1)} + y^{(\ell)}/L$. The output, $x^{(L)} = \sum_\ell y^{(\ell)}/L$, is the average of the points returned by the oracle. The exact steps in the ℓth iteration are as follows:

1. For each $i \in [m]$, compute $w_i^{(\ell)} = e^{-\operatorname{load}^{(\ell-1)}(i)}$, where
$$\operatorname{load}^{(\ell-1)}(i) = |\{k \in \{1,\dots,\ell-1\} : (Ay^{(k)})_i \ge 1\}|$$
is the number of previous iterations k where $y^{(k)}$ covers w_i.
2. Invoke the oracle with respect to $w^{(\ell)}$, returning $y^{(\ell)}$.
3. Set $x^{(\ell)} = x^{(\ell-1)} + y^{(\ell)}/L$.

Each call to the randomized oracle succeeds with high probability, and there are $L = O(\log(m)/\epsilon)$ iterations. By the union bound, all oracle calls succeed with high probability. For the remainder of the analysis we assume this is the case.

We want to show that $x^{(L)} = \sum_{\ell=1}^{L} y^{(\ell)}/L$ is a $(1 + O(\epsilon))$ approximation. It is easy to see that the objective value is good: we have

$$\langle b, x^{(L)} \rangle = \frac{1}{L} \sum_{\ell=1}^{L} \langle b, y^{(\ell)} \rangle \geq (1+\epsilon)\,\mathrm{OPT}$$

because the oracle guarantees $\langle b, y^{(\ell)} \rangle \geq (1+\epsilon)\,\mathrm{OPT}$ for all ℓ. It remains to show that $Ax^{(L)} \geq (1 - O(\epsilon))\mathbb{1}$. The key to the analysis is understanding why the weights are selected as they are.

For each iteration ℓ, let $M_\ell = \{i \in [m] : (Ay^{(\ell)})_i \geq 1\}$ be the set of coordinates covered in the ℓth iteration. The oracle guarantees that for each iteration $\ell \in [L]$,

$$\sum_{i \in M_\ell} w_i^{(\ell-1)} \geq (1-\epsilon)\langle w^{(\ell-1)}, \mathbb{1} \rangle.$$

We claim that

$$\sum_{i \in M_\ell} w_i^{(\ell)} \geq (1 - O(\epsilon))\langle w^{(\ell)}, \mathbb{1} \rangle \tag{5}$$

for all $\ell \in [L]$. To this end, we have

$$\begin{aligned}\sum_{i \notin M_\ell} w_i^{(\ell)} &\overset{(a)}{\leq} \sum_{i \notin M_\ell} w_i^{(\ell-1)} \leq \epsilon \langle w^{(\ell-1)}, \mathbb{1} \rangle \leq \frac{\epsilon}{(1-\epsilon)} \sum_{i \in M_\ell} w_i^{(\ell-1)} \\ &\overset{(b)}{=} \frac{\epsilon e}{(1-\epsilon)} \sum_{i \in M_\ell} w_i^{(\ell)} \leq \frac{\epsilon e}{(1-\epsilon)} \langle w^{(\ell)}, \mathbb{1} \rangle,\end{aligned}$$

as desired. Here, (a) is because $w^{(\ell)}$ is decreasing in ℓ. (b) is because for each $i \in M_\ell$, the weight decreases by e^{-1}.

To complete the analysis, define $f(\ell)$ by

$$f(\ell) = -\frac{1}{L} \log \left(\sum_{i=1}^{m} e^{-\mathrm{load}^{(\ell)}(i)} \right) = -\frac{1}{L} \log \left(\sum_{i=1}^{m} w_i^{(\ell)} \right).$$

$f(\ell)$ gives a smooth approximation of the minimum value of $\mathrm{load}^{(\ell)}$; we have

$$\min_i \frac{\mathrm{load}^{(\ell)}(i)}{L} - \epsilon \leq f(x) \leq \min_i \frac{\mathrm{load}^{(\ell)}(i)}{L}$$

because $L \geq \log(m)/\epsilon$. Since $(Ax^{(L)})_i \geq \mathrm{load}^{(L)}(i)$ for all i, it suffices to show that $f(L) \geq 1 - O(\epsilon)$. We have

$$f(L) = f(0) + \sum_{\ell=1}^{L} f(\ell) - f(\ell-1) = -\epsilon - \frac{1}{L} \sum_{\ell=1}^{L} \log \left(\frac{\sum_{i=1}^{m} w_i^{(\ell)}}{\sum_{i=1}^{m} w_i^{(\ell-1)}} \right)$$

For each iteration ℓ, we have

$$\begin{aligned}\sum_{i=1}^{m} w_i^{(\ell)} &\overset{(c)}{\leq} (1 + O(\epsilon)) \sum_{i \in M_\ell} w_i^{(\ell)} \overset{(d)}{=} (1 + O(\epsilon)) e^{-1} \sum_{i \in M_\ell} w_i^{(\ell-1)} \\ &\leq (1 + O(\epsilon)) e^{-1} \sum_{i} w_i^{(\ell-1)} \leq e^{-(1-O(\epsilon))} \sum_{i} w_i^{(\ell-1)}.\end{aligned}$$

Here (c) critically uses inequality (5). (d) is because all elements $i \in M_\ell$ are covered by $y^{(\ell)}$, increasing their loads by 1 and their weight by e^{-1}. Thus

$$\log\left(\frac{\sum_{i=1}^m w_i^{(\ell)}}{\sum_i w_i^{(\ell-1)}}\right) \leq -(1 - O(\epsilon)).$$

Plugging back in, we have $f(L) \geq -\epsilon - \frac{1}{L}\sum_{\ell=1}^{L}(-(1 - O(\epsilon))) = 1 - O(\epsilon)$, as desired.

References

1. Martin Aigner and Thomas A. Dowling. Matching theory for combinatorial geometries. *Transactions of the American Mathematical Society*, 158(1):231–245, 1971.
2. Sepehr Assadi. A simple (1 - ϵ)-approximation semi-streaming algorithm for maximum (weighted) matching. In Merav Parter and Seth Pettie, editors, *2024 Symposium on Simplicity in Algorithms, SOSA 2024, Alexandria, VA, USA, January 8-10, 2024*, pages 337–354. SIAM, 2024. doi:10.1137/1.9781611977936.31.
3. András A. Benczúr and David R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM J. Comput.*, 44(2):290–319, 2015.
4. Joakim Blikstad. Breaking o(nr) for matroid intersection. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 31:1–31:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.31.
5. Joakim Blikstad, Jan van den Brand, Sagnik Mukhopadhyay, and Danupon Nanongkai. Breaking the quadratic barrier for matroid intersection. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 421–432. ACM, 2021. doi:10.1145/3406325.3451092.
6. Carl Brezovec, Gérard Cornuéjols, and Fred Glover. Two algorithms for weighted matroid intersection. *Mathematical Programming*, 36(1):39–53, October 1986.
7. Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, Sahil Singla, and Sam Chiu-wai Wong. Faster matroid intersection. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1146–1168. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00072.
8. Chandra Chekuri and Kent Quanrud. A fast approximation for maximum weight matroid intersection. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 445–457. SIAM, 2016.
9. William H. Cunningham. Improved bounds for matroid partition and intersection algorithms. *SIAM J. Comput.*, 15(4):948–957, 1986.
10. Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In R. Guy, H. Hanani, N. Sauer, and J. Schönheim, editors, *Combinatorial Structures and Their Applications (Proceedings Calgary International Conference on Combinatorial Structures and Their Applications, Calgary, Alberta, 1969)*, pages 69–87. Gordon and Breach, New York, 1970.
11. Jack Edmonds. Some well-solved problems in combinatorial optimization. In B. Roy, editor, *Combinatorial Programming: Methods and Applications*, pages 285–301, Dordrecht, 1975. Springer Netherlands.
12. Jack Edmonds and Delbert Ray Fulkerson. Transversals and matroid partition. *Journal of Research National Bureau of Standards Section B*, 69:147–153, 1965.
13. András Frank. A weighted matroid intersection algorithm. *Journal of Algorithms*, 2(4):328–336, December 1981.
14. András Frank. A quick proof for the matroid intersection weight-splitting theorem. *EGRES Quick-Proof*, (2008-03), 2008.

15 András Frank. *Connections in combinatorial optimization*. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, 2011.

16 Satoru Fujishige and Xiaodong Zhang. An efficient cost scaling algorithm for the independent assignment problem. *Journal of the Operations Research Society of Japan*, 38(1):124–136, 1995.

17 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

18 Chien-Chung Huang, Naonori Kakimura, and Naoyuki Kamiyama. Exact and approximation algorithms for weighted matroid intersection. *Math. Program.*, 177(1-2):85–112, 2019. doi:10.1007/S10107-018-1260-X.

19 Chien-Chung Huang and François Sellier. Robust sparsification for matroid intersection with applications. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 2916–2940. SIAM, 2024. doi:10.1137/1.9781611977912.104.

20 David R. Karger. Random sampling and greedy sparsification for matroid optimization problems. *Math. Program.*, 82:41–81, 1998.

21 Eugene L. Lawler. Matroid intersection algorithms. *Mathematical Programming*, 9(1):31–56, December 1975.

22 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1049–1065. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.68.

23 Huy L. Nguyen. A note on cunningham's algorithm for matroid intersection. *CoRR*, abs/1904.04129, 2019. arXiv:1904.04129.

24 Kent Quanrud. Nearly linear time approximations for mixed packing and covering problems without data structures or randomization. In Martin Farach-Colton and Inge Li Gørtz, editors, *3rd Symposium on Simplicity in Algorithms, SOSA@SODA 2020, Salt Lake City, UT, USA, January 6-7, 2020*, pages 69–80. SIAM, 2020.

25 Kent Quanrud. Faster exact and approximation algorithms for packing and covering matroids via push-relabel. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 2305–2336. SIAM, 2024. doi:10.1137/1.9781611977912.82.

26 Kent Quanrud. Quotient sparsification for submodular functions. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 5209–5248. SIAM, 2024. doi:10.1137/1.9781611977912.187.

27 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.

28 Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM J. Comput.*, 40(6):1913–1926, 2011.

29 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 81–90. ACM, 2004.

Better Sparsifiers for Directed Eulerian Graphs

Sushant Sachdeva
University of Toronto, Canada

Anvith Thudi
University of Toronto, Canada

Yibin Zhao
University of Toronto, Canada

Abstract

Spectral sparsification for directed Eulerian graphs is a key component in the design of fast algorithms for solving directed Laplacian linear systems. Directed Laplacian linear system solvers are crucial algorithmic primitives to fast computation of fundamental problems on random walks, such as computing stationary distributions, hitting and commute times, and personalized PageRank vectors. While spectral sparsification is well understood for undirected graphs and it is known that for every graph G, $(1+\varepsilon)$-sparsifiers with $O(n\varepsilon^{-2})$ edges exist [Batson-Spielman-Srivastava, STOC '09] (which is optimal), the best known constructions of Eulerian sparsifiers require $\Omega(n\varepsilon^{-2}\log^4 n)$ edges and are based on short-cycle decompositions [Chu et al., FOCS '18].

In this paper, we give improved constructions of Eulerian sparsifiers, specifically:

1. We show that for every directed Eulerian graph $\vec{G}$, there exists an Eulerian sparsifier with $O(n\varepsilon^{-2}\log^2 n \log^2 \log n + n\varepsilon^{-4/3}\log^{8/3} n)$ edges. This result is based on combining short-cycle decompositions [Chu-Gao-Peng-Sachdeva-Sawlani-Wang, FOCS '18, SICOMP] and [Parter-Yogev, ICALP '19], with recent progress on the matrix Spencer conjecture [Bansal-Meka-Jiang, STOC '23].
2. We give an improved analysis of the constructions based on short-cycle decompositions, giving an $m^{1+\delta}$-time algorithm for any constant $\delta > 0$ for constructing Eulerian sparsifiers with $O(n\varepsilon^{-2}\log^3 n)$ edges.

2012 ACM Subject Classification Computing methodologies → Linear algebra algorithms; Theory of computation → Graph algorithms analysis; Mathematics of computing → Computations on matrices

Keywords and phrases Graph algorithms, Linear algebra and computation, Discrepancy theory

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.119

Category Track A: Algorithms, Complexity and Games

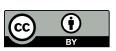
Related Version *Full Version*: `https://arxiv.org/abs/2311.06232`

Funding *Sushant Sachdeva*: NSERC Discovery Grant RGPIN-2018-06398, an Ontario Early Researcher Award (ERA) ER21-16-284, and a Sloan Research Fellowship.
Anvith Thudi: NSERC Vanier Fellowship.
Yibin Zhao: NSERC Discovery Grant RGPIN-2018-06398 ands Ontario Early Researcher Award (ERA) ER21-16-284 awarded to SS.

Acknowledgements We thank Arun Jambulapati for notifying us of an issue in a previous version of this manuscript. SS and YZ would also like to thank the Simons Institute for the Theory of Computing Fall 2023 program for its support and where a significant part of this project evolved.

1 Introduction

Given a graph $G(V, E)$, a sparsifier of G is a graph H on the same set of vertices V, but hopefully supported on a subset of the edges $E' \subset E$ such that H approximately preserves certain properties of G. Several notions of graph sparsification have been well studied for undirected graphs, e.g. spanners (approximately preserving distances), cut sparsifiers, spectral sparsifiers, etc.



Spectral sparsification is a particularly influential notion of undirected graph sparsification [44]. Spectral sparsifiers generalize cut-sparsifiers introduced by Benczur-Karger [10], which guarantees that the total weight of every vertex cut is preserved up to a multiplicative factor of $(1+\varepsilon)$ in the sparsifier. Efficient spectral sparsification was a core development that led to nearly-linear time solvers for Laplacian linear systems [44]. It further inspired the Laplacian paradigm, resulting in faster algorithms for many graph problems including sampling/counting random spanning trees [16, 17] and approximating edge centrality measures [31].

The first construction of spectral sparsifiers for undirected graphs by Spielman and Teng required $\Omega(n\varepsilon^{-2}\text{poly}(\log n))$ number of edges with a large, unspecified power of $\log n$. Subsequently, Spielman and Srivastava [42] constructed a spectral sparsifier with $O(n\varepsilon^{-2}\log n)$ edges probabilistically by independently sampling each edge with probability proportional to its leverage score. In a complete graph, sampling edges independently with probability p requires $p=\Omega(\varepsilon^{-2}\log n)$ to achieve $(1+\varepsilon)$-spectral sparsification; thus such construction requires $\Omega(n\varepsilon^{-2}\log n)$ edges. Batson-Spielman-Srivastava [7] further improved this to show that there exist spectral sparsifiers for undirected graphs with $O(n\varepsilon^{-2})$ edges and that this is tight even for the complete graph. Thus, they essentially settled the question of the optimal size of undirected spectral sparsifiers.

For directed graphs, sparsification has been trickier to define. It is immediate to see that in a complete bipartite graph with all edges directed from the left vertices to the right vertices, if one wishes to approximately preserve all directed cuts, one must preserve all the edges. This means that there is no non-trivial cut-sparsification (or its generalization) for arbitrary directed graphs. Such pathological cases can be avoided if one restricts to Eulerian directed graphs, i.e. a graph where each vertex has its total weighted in-degree equal to its total weighted out-degree, in which case cut sparsification becomes equivalent to cut sparsification of undirected graphs. Indeed, Cohen-Kelner-Peebles-Peng-Rao-Sidford-Vladu [13] defined a meaningful generalization of spectral sparsification (and hence cut sparsification) to Eulerian directed graphs. The standard notion of Eulerian approximation and (sparsification) requires exact preservation of the differences between in and out degrees while ensuring the difference in directed Eulerian Laplacians is small with respect to the Laplacian of the undirectification of the graph. That is, for $\epsilon\in(0,1)$,

$$\left\|\mathbf{L}_G^{\frac{+}{2}}(\mathbf{L}_{\vec{H}}-\mathbf{L}_{\vec{G}})\mathbf{L}_G^{\frac{+}{2}}\right\|\leq\epsilon.$$

We call these sparsifiers Eulerian sparsifiers for brevity. In a manner similar to the original Spielman-Teng construction, [13] gives a nearly-linear time $\widetilde{O}(m)$-time algorithm to build an Eulerian sparsifier with $O(n\varepsilon^{-2}\text{poly}(\log n))$ edges, with a large unspecified power of $\log n$.

Since Eulerian sparsification generalizes undirected spectral sparsification, $\Omega(n\varepsilon^{-2})$ edges are necessary for constructing Eulerian sparsifiers. There has been progress in proving the existence of Eulerian sparsifiers with fewer edges: Chu-Gao-Peng-Sachdeva-Sawlani-Wang [11] introduced the short-cycle decomposition, a decomposition of an unweighted graph as a union of short edge-disjoint cycles, and a few extra edges. As a simple lemma, they showed that every undirected graph can be represented as a union of edge-disjoint cycles of length $2\log n$, with at most $2n$ extra edges. Using this short-cycle decomposition, [11] were able to prove Eulerian sparsifiers with $O(n\varepsilon^{-2}\log^4 n)$ edges exist. However, the following natural question remains unanswered:

What is the best possible sparsity guarantee for constructing Eulerian sparsifiers?

In this paper, we make progress on this question. First, we present an improved analysis of the short-cycle based Eulerian sparsification from [11].

▶ **Theorem 1.** *For every constant $\delta > 0$, there is an algorithm that takes as input a directed Eulerian graph $\vec{G}$ and returns an ε-Eulerian sparsifier of $\vec{G}$ with $O(n\varepsilon^{-2}\log^3 n)$ edges in $m^{1+\delta}$ time.*

The above algorithm relies on independently toggling short cycles: with probability $\frac{1}{2}$ all clockwise edges are deleted and counter-clockwise edges are doubled, otherwise vice-versa. Given that the edges in each $O(\log n)$ length short-cycle are toggled in a completely correlated manner, and the cycles are toggled independently, this approach cannot lead to a sparsity better than $O(n\varepsilon^{-2}\log^3 n)$ (see Remark 9). To go past the above result, we leverage discrepancy theory, specifically recent progress on the matrix Spencer conjecture by Bansal, Jiang, and Meka [6]. (See Section 1.1 for a description of the matrix Spencer conjecture.) While the matrix Spencer conjecture is not directly useful for our application, we utilize the underlying machinery from [6] and the short-cycle decomposition to prove the following:

▶ **Theorem 2** (Informal)**.** *There is an algorithm that given an Eulerian graph $\vec{G}$, can compute in poly-time an ε-Eulerian sparsifier of $\vec{G}$ with $n\varepsilon^{-2}\log^2 n + n\varepsilon^{-3/4}\log^{8/3} n$ edges (up to $\log\log n$ factors).*

For small ε, e.g. $\varepsilon^{-1} = \Omega(\log n)$, the above theorem gives an $n\varepsilon^{-2}\log^2 n$ bound, only a $\log^2 n$ factor away from the lower bound. In Section 4.1 we show that assuming the matrix partial colouring conjecture, one can improve this result to prove the existence of ε-Eulerian sparsifiers with $n\varepsilon^{-2}\log^2 n$ edges for all ε (up to $\log\log n$ factors).

1.1 Related works

Sparsification

There are four major approaches for undirected spectral sparsification: expander decomposition [45, 4, 20], spanners [21, 23, 26], importance sampling [43, 22], and potential function based sparsification [8, 3, 28, 29]. More closely related to Eulerian sparsification is undirected degree preserving sparsification, introduced by Chu-Gao-Peng-Sachdeva-Sawlani-Wang [11]. Degree preserving sparsification is useful for constructing spectral sketches. More importantly for us, techniques for degree preserving sparsification can generally be extended to work for directed Eulerian sparsification.

Cohen-Kelner-Peebles-Peng-Rao-Sidford-Vladu [13] showed the first degree preserving (implicitly) and Eulerian sparsifier using expander decomposition. The algorithm performs random sampling of the directed edges with probability related to the degrees within each expander. Recent work by Ahmadinejad-Peebles-Pyne-Sidford-Vadhan [2] establishes an "equivalence", albeit with significantly stronger requirements than spectral approximations, between degree preserving and Eulerian sparsification under the notion of singular value approximation. They established the first Eulerian sparsifier with both nearly-linear sparsity and nearly-linear runtime, albeit with a large $\text{poly}(\log n)$ factor in both. However, the expander approach bottlenecks at $\Omega(n\varepsilon^2\log^3 n)$ due to a lowerbound on the optimal tradeoff between the expansion factor and the number of expanders [41].

The technique of using short cycles for sparsification [11] also applies to degree preserving and Eulerian sparsifications with sparsity $O(n\varepsilon^{-2}\log^2 n)$ and $O(n\varepsilon^{-2}\log^4 n)$ respectively. Improved short cycle decompositions were subsequently designed in [32, 35] to facilitate faster construction of sparsifiers. Our first result Theorem 1 follows closely to [11] and reduces the gap between degree-preserving and Eulerian sparsification under this technique.

Recently Jambulapati-Reis-Tian [19] constructed new degree preserving sparsifiers using discrepancy theory. They showed operator norm discrepancy bodies are well conditioned[1] for the symmetric and PSD matrices that arise from undirected sparsification and used an approximate version of the framework from Reis-Rothvoss [39] to give a colouring of the edges (corresponding to adding and deleting edges) under the linear constraint needed for degree preservation. However, the underlying discrepancy bodies studied by Jambulapati-Reis-Tian [19] do not align with Eulerian sparsification where matrices are no longer positive semidefinite and the primary statistic one has control over is matrix variance (see Section 4).

Directed Laplacian solvers

Cohen-Kelner-Peebles-Peng-Sidford-Vladu [14] initiated the line of work that studies solving directed Laplacian linear systems. They established a reduction from solving general directed Laplacian systems to Eulerian Laplacian systems.Cohen-Kelner-Peebles-Peng-Rao-Sidford-Vladu [13] gave an almost linear time algorithm for solving Eulerian Laplacians using the squaring identities from Peng-Spielman [37]. Subsequently, Cohen-Kelner-Kyng-Peebles-Peng-Rao-Sidford [12] gave the first nearly linear time solver using the standard approximate LU factorization techniques that enjoyed great success in undirected Laplacian solvers [27, 24, 40]. Ahmadinejad-Jambulapati-Saberi-Sidford [1] further established a reduction from solving systems of (asymmetric) M-matrices to Eulerian Laplacian systems, giving fast computation of several problems closely associated with the Perron-Frobenius theorem. Peng-Song [36] extended the approach from [12] and gave an approach for extending an algorithm for building Eulerian sparsifiers to a fast solver for Eulerian Laplacian linear systems. Combined with Theorem 1, they give an $O(n \log^4 n \log(\frac{n}{\varepsilon}))$ time solver with $m^{1+\delta}$ preprocessing time for any constant $\delta > 0$. Kyng-Meierhans-Probst-Gutenberg [25] established the first derandomized directed Laplacian solver in almost linear time.

Discrepancy theory

The Matrix Spencer Conjecture [47, 34] is a major open problem in discrepancy theory:

▶ **Conjecture 3** (Matrix Spencer Conjecture). *Given $n \times n$ symmetric matrices $\mathbf{A}_1, \ldots, \mathbf{A}_m \in \mathbb{R}^{n \times n}$ with $\|\mathbf{A}_i\| \leq 1$, there exist signs $x \in \{\pm 1\}^m$ such that $\|\sum_{i=1}^m x_i \mathbf{A}_i\| \leq O(\sqrt{m} \cdot \max\{1, \sqrt{\min\{1, \log(\frac{n}{m})\}}\})$.*

As a natural comparison, for a uniform random colouring $x \in \{\pm 1\}^m$, the matrix Chernoff bound [46] gives the following bound which has a gap of $\sqrt{\log n}$ to Conjecture 3 when $m \geq n$:

$$\mathbb{E}\left[\left\|\sum_i x_i \mathbf{A}_i\right\|\right] = O\left(\sqrt{\log n}\right) \cdot \left\|\sum_i \mathbf{A}_i^2\right\|^{\frac{1}{2}} \leq O(\sqrt{m \log n}).$$

We refer readers to [30, 18, 15] for recent progress toward solving this conjecture.

Many natural problems in studying the spectra of matrices can be viewed as discrepancy theory problems, e.g., graph sparsification [8, 38] and the Kadison-Singer problem [33]. Reis-Rothvoss [38] studies the geometry of operator norm balls for a collection of matrices where, $\|\sum_i |\mathbf{A}_i|\|$ is small. This was subsequently used in Jambulapati-Reis-Tian [19] to show optimal degree preserving sparsification. As previously mentioned, this line of work does not apply to Eulerian sparsification since matrices that emerge from our setting do not satisfy

[1] I.e., satisfy certain Gaussian measure lowerbound

that $\|\sum_i |\mathbf{A}_i|\|$ is small. Bansal-Jiang-Meka [6] resolved the Matrix Spencer Conjecture for matrices of rank $n/(\log^{O(1)} n)$ using a recent advancement in matrix concentration bounds due to Bandeira-Boedihardjo-van-Handel [5]. The partial colouring result for controlling operator norm used in [6] serves as the main machinery in our existential results (see Lemma 4). Specifically, the matrices we study naturally satisfy $\|\sum_i \mathbf{A}_i^2\|$ is small.

1.2 Technical overview

Our approach to constructing Eulerian sparsifiers builds on the framework of Chu-Gao-Peng-Sachdeva-Sawlani-Wang [11]. The sparsification algorithm in [11] combines importance sampling of edges with a short cycle decomposition. At each iteration, the algorithm restricts its attention to edges with small "importance" in the undirected graph (edges with leverage score $w_e \mathbf{b}_e^\top \mathbf{L}_G^+ \mathbf{b}_e$ at most constant times the average leverage score, $O(\frac{n}{m})$). The algorithm then performs a short cycle decomposition on these edges – expressing the graph as a union of uniformly weighted edge-disjoint short cycles and a few extra edges. For each short cycle, the algorithm independently keeps either the clockwise edges or the counter-clockwise edges with probability $\frac{1}{2}$ each. The number of edges reduces by a constant fraction in expectation at each iteration. After doubling the weights of the cycle edges retained, the algorithm guarantees that the Eulerianess of each cycle is preserved and, hence, the entire graph. Moreover, when combined with the undirected leverage score condition above, changes in directed short cycles also have a small variance overall. The matrix Bernstein inequality for asymmetric matrices guarantees a small approximation error for this randomized step. We repeat this process until the desired approximation error is met.

Our improved result of this algorithm is due to th improved variance bounds in Lemma 8 for random matrices corresponding to short cycles. Rather than bounding the variance through complete graphs as in [11], we bound it with respect to the undirected cycle. This improved variance also serves a critical role in our partial colouring approach in Section 4.

In the rest of our paper, we present our existential result which uses the partial colouring lemma, Lemma 4, from [6] to choose how to sparsify the short cycles. The algorithm follows the same high-level approach as the random sampling construction above. For each directed short cycle, instead of independently sampling cycle edges, we will use the partial colouring given by Lemma 4. In each iteration, Lemma 4 gives a partial colouring with sufficiently many fully coloured entries (i.e., ± 1 entries) on all cycles. It then allows us to remove a constant fraction of the cycle edges with less error than random sampling.

▶ **Lemma 4** ([6] Lemma 3.1). *There exists constants $c, c' > 0$ such that given symmetric matrices $\mathbf{A}_1, \ldots, \mathbf{A}_m \in \mathbb{R}^{n \times n}$ satisfying $\|\sum_{i=1}^m \mathbf{A}_i^2\| \leq \sigma^2$ and $\sum_{i=1}^m \|\mathbf{A}_i\|_F^2 \leq mf^2$ and a point $\mathbf{y} \in (-1,1)^m$, there is an algorithm* PartialColour *that returns in polynomial time a point $\mathbf{x} \in [-1,1]^m$ such that $|\{i : x_i \in \{\pm 1\}\}| > c'm$ and*

$$\left\| \sum_{i=1}^m (x_i - y_i)\mathbf{A}_i \right\| \leq c(\sigma + (\log^{\frac{3}{4}} n)\sqrt{\sigma f}). \tag{1}$$

There are two major challenges in applying Lemma 4. Firstly, within each iteration, we cannot afford to fully colour all the cycles by recursively applying Lemma 4, since we might have to perform the partial colouring $O(\log n)$ times, resulting in an additional log factor in the sparsity. Thus, we are always left with fractionally coloured cycles (i.e., entries with magnitude < 1). Such cycles must still be incorporated into the sparsified graph to guarantee the error given by Lemma 4. However, we cannot explicitly modify the graph to include edges corresponding to these cycles, as we would lose the integral and polynomially

bounded weight conditions and the short cycle decomposition could no longer be applied to this new graph. The second challenge also comes from incorporating fractionally coloured cycles in the next iteration. Unlike the undirected case, the two parts of a directed cycle do not necessarily have the same number of edges. For example, a directed cycle with all edges in the same direction has all the edges in one part and none in the other. If we start our colouring process from a non-zero initial partial colouring (i.e., a non-zero $\mathbf{y}$ to Lemma 4), we could end up at a colouring where almost no edges are removed.

To deal with these problems, our algorithm handles the integral weighted portion $\vec{G}$ of the graph $\vec{G}'$ and the fractionally coloured cycles $\overline{S}$ separately (see Algorithm 4). For the integral weighted portion, we perform the partial colouring to guarantee at least a constant fraction of edges are removed. We then add the fractionally coloured cycles into the set $\overline{S}$. For the set of fractionally coloured cycles $\overline{S}$, we adjust their colouring by considering the difference between the partial colours and ± 1 to ensure that a good portion of cycles in $\overline{S}$ are fully coloured after the procedure to guarantee the size of $\overline{S}$ does not blow up. In both cases, the error incurred by the partial colouring operation is controlled to guarantee our desired final error (Theorem 2).

2 Preliminaries

We use $\widetilde{O}(\cdot)$ to suppress polylog factors in n, m. We say "with high probability in n" for an event occuring with probability $1 - n^{-\Omega(1)}$. For graphs, n is assumed to be the number of vertices and is often omitted. All logarithms in the paper are base 2.

Linear Algebra

We use boldface to denote vectors, and use $\mathbf{0}$ and $\mathbf{1}$ for the all-zeros and all-ones vectors. We let $\mathbf{e}_u$ denote the vector that is 1 in the uth coordinate and 0 elsewhere. We denote $\mathbf{b}_{uv} = \mathbf{e}_u - \mathbf{e}_v$ for any $u \neq v$. For vectors $\mathbf{u}, \mathbf{v}$ of equal dimension, $\mathbf{u} \circ \mathbf{v}$ is the entrywise product. For a linear subspace $\mathcal{W}$ of a vector space $\mathcal{V}$, we denote $\mathcal{W}^{\perp}$ as the orthogonal complement of $\mathcal{W}$ in $\mathcal{V}$.

Matrices are denoted in boldface capticals. We use $\ker(\mathbf{A}), \mathrm{im}(\mathbf{A})$ to denote the kernel and image of $\mathbf{A}$. For any u, we let $(\mathbf{A})_u$ denote the uth column of $\mathbf{A}$. The Kronecker product of matrices $\mathbf{A}$ and $\mathbf{B}$ is denoted $\mathbf{A} \otimes \mathbf{B}$. A symmetric matrix $\mathbf{A}$ is positive semidefinite (PSD) (resp. positive definite (PD)) if, for any vector $\mathbf{x}$ of compatible dimension, $\mathbf{x}^\top \mathbf{A}\mathbf{x} \geq 0$ (resp. $\mathbf{x}^\top \mathbf{A}\mathbf{x} > 0$). Let $\mathbf{A}$ and $\mathbf{B}$ be two symmetric matrices of the same dimension, then we write $\mathbf{B} \preccurlyeq \mathbf{A}$ or $\mathbf{A} \succcurlyeq \mathbf{B}$ if $\mathbf{A} - \mathbf{B}$ is PSD. The ordering given by $\preccurlyeq$ is called Loewner partial order.

▶ **Lemma 5.** *If $\mathbf{A} \succcurlyeq \mathbf{B}$ and $\mathbf{C}$ is any matrix of compatible dimension, then $\mathbf{C}\mathbf{A}\mathbf{C}^\top \succcurlyeq \mathbf{C}\mathbf{B}\mathbf{C}^\top$.*

Let $\|\mathbf{A}\|$ and $\|\mathbf{A}\|_F = \sqrt{\mathrm{Tr}\,(\mathbf{A}^*\mathbf{A})}$ denote the operator norm and Frobenius norm of a matrix $\mathbf{A}$. The operator norm is equal to the largest singular value of $\mathbf{A}$. For a matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$, we define the Hermitian (symmetric) lift of $\mathbf{A}$ by

$$\mathsf{hlift}(\mathbf{A}) = \begin{bmatrix} & \mathbf{A} \\ \mathbf{A}^\top & \end{bmatrix} \in \mathbb{R}^{(n+m)\times(n+m)}$$

The norms of Hermitian lifts satisfy $\|\mathsf{hlift}(\mathbf{A})\| = \|\mathbf{A}\|$ and $\|\mathsf{hlift}(\mathbf{A})\|_F = 2\|\mathbf{A}\|_F$. Given a symmetric matrix with eigenvalue decomposition $\mathbf{A} = \sum_i \lambda_i \mathbf{v}_i \mathbf{v}_i^\top$, where $\{\mathbf{v}_i\}_i$ form an orthonormal basis, the pseudoinverse is defined as $\mathbf{A}^+ = \sum_{i:\lambda_i \neq 0} \frac{1}{\lambda_i} \mathbf{v}_i \mathbf{v}_i^\top$. The absolute value of $\mathbf{A}$ on eigenvalues is defined as $|\mathbf{A}| = \sum_{i:\lambda_i \neq 0} |\lambda_i| \mathbf{v}_i \mathbf{v}_i^\top$. Note that $|\mathbf{A}|$ is PSD. Similarly for symmetric PSD matrix $\mathbf{A}$ we have $\mathbf{A}^{1/2} = \sum_{i:\lambda_i \neq 0} \sqrt{\lambda_i} \mathbf{v}_i \mathbf{v}_i^\top$ and $\mathbf{A}^{+/2} = \sum_{i:\lambda_i \neq 0} \frac{1}{\sqrt{\lambda_i}} \mathbf{v}_i \mathbf{v}_i^\top$.

Graphs and Laplacians

$\vec{G} = (V, E, \mathbf{w})$ denotes a weighted directed graph (allowing multi-edges) with edge weights $\mathbf{w} : E \to \mathbb{R}_{\geq 0}$. G denotes the undirected graph of $\vec{G}$ where directed edgex $e \in E(\vec{G})$ correspond to undirected edges between the same vertices with half the weight. $\vec{G}$ is Eulerian if the weighted in degree equals the weighted out degree for each vertex $v \in V$.

We define the adjacency matrix of $\vec{G}$ as a non-negative matrix $\mathbf{A}_{\vec{G}}$ with $\mathbf{A}_{uv} = w_{uv}$ if $(u, v) \in E$ and 0 otherwise. The weighted degree matrix of $\vec{G}$ is a non-negative diagonal matrix $\mathbf{D}_{\vec{G}}$ corresponding to the weighted out-degrees of $\vec{G}$. We define the directed Laplacian of $\vec{G}$ as $\mathbf{L}_{\vec{G}} = \mathbf{D}_{\vec{G}} - \mathbf{A}_{\vec{G}}^\top$ and it satisfies $\mathbf{1}^\top \mathbf{L}_{\vec{G}} = \mathbf{0}^\top$, i.e. $(\mathbf{L}_{\vec{G}})_{uu} = -\sum_{v \neq u} \mathbf{L}_{vu}$ for all $u \in V$. For a weighted Eulerian directed graph $\vec{G}$, its graph Laplacian additionally satisfies $\mathbf{L}_{\vec{G}}\mathbf{1} = \mathbf{0}$. Assuming Eulerian graph $\vec{G}$, the associated undirected graph Laplacian matrix of G is $\mathbf{L}_G = \frac{1}{2}(\mathbf{L}_{\vec{G}} + \mathbf{L}_{\vec{G}}^\top)$. $\mathbf{L}_G$ is symmetric and PSD. For an undirected Laplacian $\mathbf{L}_G$, the effective resistance and leverage score of an edge $e \in E(G)$ are defined by $\mathrm{Reff}_G(e) = \mathbf{b}_e^\top \mathbf{L}_G^+ \mathbf{b}_e$ and $\boldsymbol{\tau}_G(e) = w_e \mathrm{Reff}_G(e)$ where we fixed an arbitrary orientation for the undirected edge e. We use n and m for the number of vertices and edges in G. As is standard, we study strongly connected Eulerian graphs with positive integral and polynomially bounded edge weights (i.e., weights bounded by $n^{O(1)}$).

3 Eulerian sparsification via short cycle decomposition

We first present an improved analysis of constructing Eulerian sparsifiers using short cycle decompositions analogous to [11]. In particular, we provide a better variance analysis of the error terms in sparsification than what was used by [11]; by Matrix Bernstein [46] this will allow us to use fewer edges to retain a desired error bound.

We first recall the definition of a short cycle decomposition of a graph G.

▶ **Definition 6.** *An $(\hat{m}, L)$-short cycle decomposition of an unweighted undirected graph G, decomposes G into several edge-disjoint cycles, each of length at most L, and at most $\hat{m}$ edges are not in the union of the cycles.*

We let CycleDecomposition be an algorithm that takes as input an unweighted graph with n vertices and m edges and returns a $(\hat{m}, L)$-short cycle decomposition in time $\mathrm{T}_{\mathrm{CD}}(m, n)$. As in [11], we assume the *super-additivity* of T_{CD}, $\sum_i \mathrm{T}_{\mathrm{CD}}(m_i, n) \leq \mathrm{T}_{\mathrm{CD}}\left(\sum_i m_i, n\right)$, for all $m_i \geq n$. Relevant to us is a construction of short cycle decompositions which for every constant $\delta > 0$, gives an $(O(n \log n), O(\log n))$-short cycle decomposition in time $m^{1+\delta}$.

▶ **Lemma 7** ([35] Theorem 2). *For any $\delta > 0$, there is an algorithm that computes an $(O(n \log n), O(2^{\frac{1}{\delta}} \log n))$-short cycle decomposition of an undirected unweighted graph in $2^{O(\frac{1}{\delta})} mn^{\delta}$ time.*

Our random sampling based sparsification algorithm is the same as [11]. We repeatedly sparsify an Eulerian graph by keeping only the "clockwise" or "counter-clockwise" edges of each cycle in a short cycle decomposition of the graph, see CycleSparsify in Algorithm 2 and CycleSparsifyOnce in Algorithm 3.

Stated in other words, we will sparsify a cycle by partitioning it into two sets and removing one randomly. For a directed cycle $\vec{C}$, we take $\vec{F}, S$ to be the outputs of CorrectOrientation$(\vec{C})$. In particular, $\vec{F}$ is the cycle $\vec{C}$ corrected so that every vertex has an incoming edge and an outgoing edge, and S is the undirected graph coming from the set of edges in $\vec{C}$ whose direction we reversed (where the edge weight in S are the same as the original edge

Algorithm 1 CorrectOrientation($\vec{C}$).

1 Pick an arbitrary edge e_1 in $\vec{C}$ and let v_1 be its tail vertex. Define $V_{\vec{C}}$ as the vertex set of $\vec{C}$.
2 Initialize $E_{\vec{S}} \leftarrow \emptyset$, $E_{\vec{F}} \rightarrow \{e_1\}$, $V_{\vec{F}} = \{v_1\}$, $i = 1$
3 **while** $|V_{\vec{C}} \setminus V_{\vec{F}}| > 0$ **do**
4 $\quad i \leftarrow i + 1$
5 $\quad$ Take e_{i+1} be the other edge incident on v_i.
6 $\quad$ If e_{i+1} is outgoing from v_i, take v_{i+1} the head of e_{i+1} and update $E_{\vec{F}} \leftarrow E_{\vec{F}} \cup \{e_{i+1}\}$, $V_{\vec{F}} \leftarrow V_{\vec{F}} \cup \{v_{i+1}\}$.
7 $\quad$ Else let v_{i+1} be the tail of e_{i+1} and update $E_{\vec{S}} \leftarrow E_{\vec{S}} \cup \{e_{i+1}\}$, $E_{\vec{F}} \leftarrow E_{\vec{F}} \cup \{\text{rev}(e_{i+1})\}$, $V_{\vec{F}} \leftarrow V_{\vec{F}} \cup \{v_{i+1}\}$.
8 **return** $\vec{F}$ *defined by* $E_{\vec{F}}$ *and* $V_{\vec{F}}$*, and* S *the undirected graph defined by* $E_{\vec{S}}$ *and the incident vertices of* $E_{\vec{S}}$.

Algorithm 2 CycleSparsify($\vec{G}, \varepsilon,$ CycleDecomposition).

1 Decompose each edge by its binary representation.
2 Compute $\mathbf{r}$ a 1.5-approximate effective resistances in G.
3 **while** $|E(\vec{G})| \geq O(\hat{m} \log n + \varepsilon^{-2} n L^2 \log n)$ **do**
4 $\quad \vec{G} \leftarrow$ CycleSparsifyOnce($\vec{G}, \mathbf{r},$CycleDecomposition).
5 **return** $\vec{G}$.

weights). We consider the direction of edges defined by $\vec{F}$ as clockwise. Then, the edges in S are all the counter-clockwise edges in $\vec{C}$. For a cycle C and its corresponding directed cycle $\vec{C}$, the directed graph Laplacian added at line 7 in CycleSparsifyOnce is the following:

$$\begin{cases} \mathbf{L}_{\vec{C}} + \mathbf{L}_{\vec{F}} - \mathbf{L}_S & \text{w.p. } \frac{1}{2} \\ \mathbf{L}_{\vec{C}} - \mathbf{L}_{\vec{F}} + \mathbf{L}_S & \text{w.p. } \frac{1}{2} \end{cases}$$

which means the changes incurred on the directed graph Laplacian is

$$\begin{cases} \widetilde{\mathbf{L}} & \text{w.p. } \frac{1}{2} \\ -\widetilde{\mathbf{L}} & \text{w.p. } \frac{1}{2} \end{cases}, \text{where } \widetilde{\mathbf{L}} = \mathbf{L}_{\vec{F}} - \mathbf{L}_S. \tag{2}$$

Note that this change preserves the difference between the in and out degrees of $\vec{C}$. Either a vertex had an incoming and outgoing edge (and so difference 0), in which case both edges are either in $\vec{F} \setminus S$ or in S and hence always added together with the same weights (so still difference 0). Alternatively a vertex had two incoming or outgoing edges, in which case only one is ever added with twice the weight, which then still preserves the difference between in and out degree.

To obtain the improved approximation error guarantees, we show Lemma 8 that bounds the effect of $\mathbf{L}_{\vec{F}}$. Compared to Lemma 5.6 in [11], our result improves the bound by a factor of L.

▶ **Lemma 8.** *If* $\vec{C}$ *is a equal weighted directed cycle of length* L *contained in a graph* $\vec{G}$ *where each edge* $\vec{e} \in \vec{C}$ *satisfies* $\boldsymbol{\tau}_G(e) \leq \rho$*. Then,* $\mathbf{L}_{\vec{F}}^\top \mathbf{L}_G^+ \mathbf{L}_{\vec{F}} \preceq O(L^2 \rho) \mathbf{L}_C$.

Algorithm 3 CycleSparsifyOnce($\vec{G}, \mathbf{r}$, CycleDecomposition).

Input: A directed Eulerian graph $\vec{G}$ where edge weights are integral powers of 2,a 2-approximate effective resistances $\mathbf{r}$ in G, a short cycle decomposition algorithm CycleDecomposition.

Output: A directed Eulerian graph $\vec{H}$ where edge weights are integral powers of 2.

$\vec{H} \leftarrow \vec{G}$ with only the edges which satisfies $w_e r_e > \frac{4n}{m}$ and remove these edges from $\vec{G}$.

Partition $\vec{G}$ into uniformly weighted graph $\vec{G}_1, \ldots, \vec{G}_s$ where $\vec{G}_i$ has all edge weights 2^i and $s = O(\log n)$.

for *each* $\vec{G}_i$ **do**

  $\{C_{i,1}, \ldots, C_{i,t}\} \leftarrow$ CycleDecomposition(G_i) and let $\vec{C}_{i,j}$ be the corresponding directed graph of $C_{i,j}$ in $\vec{G}_i$.

  $\vec{H} \leftarrow \vec{H} + \vec{G}_i \setminus \left(\bigcup_{j=1}^{t} \vec{C}_{i,j}\right)$.

  for *each cycle* $\vec{C}_{i,j}$ **do**

    With probability $\frac{1}{2}$, add all its clockwise edges with twice their weight to $\vec{H}$. Otherwise, add the counter-clockwise edges instead.

return $\vec{H}$.

Proof. Let $\mathbf{\Pi}_C = \mathbf{I}_C - \frac{1}{L}\mathbf{1}_C\mathbf{1}_C^\top$ be the projection matrix on the support of C except the all one vector on C. Notice that $\ker^\perp(\mathbf{L}_C) = \mathrm{im}(\mathbf{L}_C) = \mathrm{im}(\mathbf{\Pi}_C)$. Furthermore, we have $\mathrm{im}(\mathbf{L}_{\vec{F}}) \subset \mathrm{im}(\mathbf{\Pi}_C)$ (as $\mathbf{1}_C \in \mathrm{im}^\perp(\mathbf{L}_{\vec{F}})$) so $\mathbf{\Pi}_C\mathbf{L}_{\vec{F}} = \mathbf{L}_{\vec{F}}$, and also $\mathrm{im}^\perp(\mathbf{\Pi}_C) \subset \ker(\mathbf{L}_{\vec{F}}^\top)$ hence $\mathbf{L}_{\vec{F}}^\top\mathbf{\Pi}_C = \mathbf{L}_{\vec{F}}^\top$. Thus, $\mathbf{L}_{\vec{F}}^\top\mathbf{L}_G^+\mathbf{L}_{\vec{F}} = \mathbf{L}_{\vec{F}}^\top\mathbf{\Pi}_C\mathbf{L}_G^+\mathbf{\Pi}_C\mathbf{L}_{\vec{F}}$.

Let w be the weight of each edge in $\vec{C}$. Then, $\mathbf{L}_{\vec{F}} = w(\mathbf{I} - \mathbf{P})$ where $\mathbf{P}$ is a permutation matrix on the vertices of C corresponding to the transition matrix $\vec{F}$ and $\mathbf{L}_C = \frac{w}{2}(2\mathbf{I} - \mathbf{P} - \mathbf{P}^\top)$. Now, $\mathbf{L}_{\vec{F}}^\top\mathbf{\Pi}_C\mathbf{L}_{\vec{F}} = \mathbf{L}_{\vec{F}}^\top\mathbf{L}_{\vec{F}} = w^2(\mathbf{I} - \mathbf{P}^\top)(\mathbf{I} - \mathbf{P}) = w^2(2\mathbf{I} - \mathbf{P} - \mathbf{P}^\top) = 2w\mathbf{L}_C$. As $\ker(\mathbf{L}_G^+) \subseteq \ker(\mathbf{\Pi}_C)$, it suffices to show $\|\mathbf{\Pi}_C\mathbf{L}_G^+\mathbf{\Pi}_C\| = O(\frac{L^2\rho}{w})$. We can write out each column of $\mathbf{\Pi}_C$ by $(\mathbf{\Pi}_C)_u = \frac{1}{L}\sum_{v \in C, v \neq u} \mathbf{b}_{uv}$ for $u \in C$ and $\mathbf{0}$ otherwise. As effective resistance is a metric, $w\mathbf{b}_{uv}^\top\mathbf{L}_G^+\mathbf{b}_{uv} \leq (L-1)\rho$ for any distinct vertices $u, v \in C$. Note that this factor of L is an upperbound on the combinatorial distance from u to v in C. Then,

$$\begin{aligned}
\left|(\mathbf{\Pi}_C)_x^\top\mathbf{L}_G^+(\mathbf{\Pi}_C)_u\right| &= \left|\left(\frac{1}{L}\sum_{y \in C, y \neq x} \mathbf{L}_G^{\frac{+}{2}}\mathbf{b}_{xy}\right)^\top\left(\frac{1}{L}\sum_{v \in C, v \neq u} \mathbf{L}_G^{\frac{+}{2}}\mathbf{b}_{uv}\right)\right| \\
&\leq \sum_{y \neq x, y \in C}\sum_{v \neq u, v \in C} \frac{1}{wL^2}\left\|w^{\frac{1}{2}}\mathbf{L}_G^{\frac{+}{2}}\mathbf{b}_{xy}\right\| \cdot \left\|w^{\frac{1}{2}}\mathbf{L}_G^{\frac{+}{2}}\mathbf{b}_{uv}\right\| \\
&\leq (L-1)^2 \times \frac{(L-1)\rho}{wL^2} \leq \frac{L\rho}{w}.
\end{aligned}$$

By Gershgorin circle theorem and the length of C, any eigenvalue of $\mathbf{\Pi}_C\mathbf{L}_G^+\mathbf{\Pi}_C$ cannot exceed $\frac{L^2\rho}{w}$ as required. ◀

▶ Remark 9. There is still a gap of factor L when comparing Lemma 8 to the undirected case. It turns out Lemma 8 is tight. Consider the multi-graph $\vec{G}$ that consists of a directed cycle with edges of weight 1 in the same orientation $\vec{F}$ and a undirected cycle C on the same vertices of edge weight ρ^{-1} for $\rho < \frac{1}{2}$. Then, each edge of the directed graph has undirected leverage score $\Theta(\rho)$ while $\mathbf{L}_{\vec{F}}^\top\mathbf{L}_G^+\mathbf{L}_{\vec{F}} = \Theta(\frac{\rho}{L})\mathbf{L}_K$ where $\mathbf{L}_K$ is the Laplacian of unit clique on the vertices of $\vec{G}$. Since $\mathbf{L}_K$ cannot be bounded by $o(L^3)\mathbf{L}_F$, this gives the lowerbound.

When combined with Lemma 5.5 of [11], we obtain the following spectral bounds on matrices which appear later in our variance analysis. We refer readers to [11] and the full version of our paper for the proofs of all subsequent claims in this section.

▶ **Lemma 10.** *Let $\vec{C}$ is an equal weighted directed cycle of length L contained in a graph $\vec{G}$ where each edge $\vec{e} \in \vec{C}$ satisfies $\boldsymbol{\tau}_G(e) \leq \rho$. Then $\mathbf{L}_G^{\frac{+}{2}}(\widetilde{\mathbf{L}}^\top \mathbf{L}_G^{+}\widetilde{\mathbf{L}})\mathbf{L}_G^{\frac{+}{2}} \preceq O(L^2\rho)\cdot \mathbf{L}_G^{\frac{+}{2}}\mathbf{L}_C\mathbf{L}_G^{\frac{+}{2}}$ and $\mathbf{L}_G^{\frac{+}{2}}(\widetilde{\mathbf{L}}\mathbf{L}_G^{+}\widetilde{\mathbf{L}}^\top)\mathbf{L}_G^{\frac{+}{2}} \preceq O(L^2\rho)\cdot \mathbf{L}_G^{\frac{+}{2}}\mathbf{L}_C\mathbf{L}_G^{\frac{+}{2}}$.*

The matrix Bernstein's inequality [46] then gives the sparsification and error guarantees of running CycleSparsifyOnce in Lemma 11.

▶ **Lemma 11.** *Given a directed Eulerian graph $\vec{G}$ whose edge weights are integral powers of 2, and additionally 2-approximate effective resistances $\mathbf{r}$ in G, the algorithm* CycleSparsify-Once *returns in $O(m) + \mathrm{T}_{\mathrm{CD}}(m,n)$ time a directed Eulerian graph $\vec{H}$ with edge weights still being powers of 2 such that if the number of edges in G satisfy $m = \Omega(\hat{m}\log n + nL^2\log n)$, then with high probability, the number of edges in $\vec{H}$ is at most $\frac{15}{16}m$ and*

$$\left\|\mathbf{L}_G^{\frac{+}{2}}(\mathbf{L}_{\vec{G}} - \mathbf{L}_{\vec{H}})\mathbf{L}_G^{\frac{+}{2}}\right\| \leq O\left(\sqrt{\frac{nL^2\log n}{m}}\right).$$

We now provide the guarantees of CycleSparsify, which repeatedly calls CycleSparsifyOnce until a criterion on the number of edges is met.

▶ **Theorem 12.** *Given as input an Eulerian graph $\vec{G}$ with polynomial bounded integral edge weights and $\varepsilon \in (0, \frac{1}{2})$, the algorithm* CycleSparsify *returns in $O(m\log^2 n) + \mathrm{T}_{\mathrm{CD}}(O(m\log n), n)$ time a Eulerian graph $\vec{H}$ with $O(\hat{m}\log n + \varepsilon^{-2}nL^2\log n)$ edges such that with high probability,*

$$\left\|\mathbf{L}_G^{\frac{+}{2}}(\mathbf{L}_{\vec{G}} - \mathbf{L}_{\vec{H}})\mathbf{L}_G^{\frac{+}{2}}\right\| \leq \varepsilon.$$

Plugging in Lemma 7, we obtain the improved results on constructing Eulerian Sparsifiers with short cycle decompositions, summarized in Theorem 1.

▶ **Theorem 1.** *For every constant $\delta > 0$, there is an algorithm that takes as input a directed Eulerian graph $\vec{G}$ and returns an ε-Eulerian sparsifier of $\vec{G}$ with $O(n\varepsilon^{-2}\log^3 n)$ edges in $m^{1+\delta}$ time.*

4 Sparsification via partial colouring

In the previous algorithm CycleSparsify, the approach to sparsifying was to randomly pick one part of each cycle (out of a partitioning of the cycle into two parts) to remove from the graph. The analysis then followed by observing on average this leads to a good approximation, and that furthermore the variance in this random construction is sufficiently small. In this section, we show, however, that by using recent partial colouring results on operator norm discrepancy bodies to pick what parts of a cycle to remove, we can obtain better sparsifiers. The main partial colouring result we use, relevant for picking a subset of matrices to keep with minimal error, is restated below.

Algorithm 4 ColourSparsify($\vec{G}, \varepsilon$).

Decompose each edge by its binary representation.
Compute $\mathbf{r}$ a 1.5-approximate effective resistances in G.
Let $\overline{S}$ be a set of cycles initialized to empty and let $\overline{x}$ be its corresponding partial colouring.
Set $\vec{G}' \leftarrow \vec{G}+$ ColourWeights($\overline{S}, \overline{x}$).
while $m' \geq O(n\varepsilon^{-2}\log^2 n(\log\log n)^2 + n\varepsilon^{-\frac{4}{3}}\log^{\frac{8}{3}} n)$ **do**
  if $4m \geq m'$ **then**
    $\vec{G}, \vec{G}', \overline{S}, \overline{x} \leftarrow$ ColourSparsifyGraph($\vec{G}, \vec{G}', \overline{S}, \overline{x}, \mathbf{r}$).
  else
    $\vec{G}, \vec{G}', \overline{S}, \overline{x} \leftarrow$ ColourSparsifyCycle($\vec{G}, \vec{G}', \overline{S}, \overline{x}, \mathbf{r}$).
return $\vec{G}'$.

Algorithm 5 ColourWeights(S, x).

Let $\vec{H}$ be an empty directed graph.
for *each cycle* $C \in S$ *and corresponding directed cycle* $\vec{C}$ **do**
  Add all the clockwise (resp. counter-clockwise) edges in $\vec{C}$ with $1 + x_C$ (resp. $1 - x_C$) times their weight to $\vec{H}$. Note if $1 + x_C = 0$ (resp. $1 - x_C = 0$) the corresponding edge is not added.
return $\vec{H}$.

▶ **Lemma 4** ([6] Lemma 3.1). *There exists constants $c, c' > 0$ such that given symmetric matrices $\mathbf{A}_1, \ldots, \mathbf{A}_m \in \mathbb{R}^{n\times n}$ satisfying $\|\sum_{i=1}^m \mathbf{A}_i^2\| \leq \sigma^2$ and $\sum_{i=1}^m \|\mathbf{A}_i\|_F^2 \leq mf^2$ and a point $\mathbf{y} \in (-1,1)^m$, there is an algorithm* PartialColour *that returns in polynomial time a point $\mathbf{x} \in [-1,1]^m$ such that $|\{i : x_i \in \{\pm 1\}\}| > c'm$ and*

$$\left\|\sum_{i=1}^m (x_i - y_i)\mathbf{A}_i\right\| \leq c(\sigma + (\log^{\frac{3}{4}} n)\sqrt{\sigma f}). \tag{1}$$

For this section, we assume the short cycle decomposition guarantees by Lemma 7 with $\hat{m} = O(n\log n)$ and $L = O(\log n)$. For each cycle C with its corresponding directed cycle $\vec{C}$, we set $\mathbf{A}(C) = \mathsf{hlift}(\mathbf{L}_{\vec{G}'}^{\frac{+}{2}}(\mathbf{L}_{\vec{F}_C} - \mathbf{L}_{S_C})\mathbf{L}_{\vec{G}'}^{\frac{+}{2}})$ where $\vec{F}_C$ is the cycle $\vec{C}$ with all edges set in clockwise direction and S_C is undirected graph with the set of edges corresponding to the counter-clockwise edges in $\vec{C}$, same as in Section 3. Note that this orientation is set initialy by CorrectOrientation after a short cycle decomposition step and fixed through out the execution. Given a set of cycles S, we let $\mathcal{A}[S]$ be the collection $\{\mathbf{A}(C)\}_{C\in S}$.

ColourWeights is our partial colouring alternative of the random selection of edges in a cycle in CycleSparsifyOnce. It similarly does not change the difference between the in-degree and out-degree and preserves integral weights, stated in Lemma 13.

▶ **Lemma 13.** *Given a set of cycles S where each cycle is uniformly weighted, and any partial colouring $x \in [-1,1]^S$, the algorithm* ColourWeights *returns a directed graph $\vec{H}$ such that the difference in the in and out degrees are the same as in $\sum_{C\in S}\vec{C}$. If $x \in \{\pm 1\}^S$, $\vec{H}$ also has integral edge weights with the largest edge weight at most twice the largest edge weight in cycles in S.*

Algorithm 6 ColourSparsifyGraph$(\vec{G}, \vec{G}', \overline{S}, \overline{\mathbf{x}}, \mathbf{r})$.

Input: A directed Eulerian graph $\vec{G}$ where edge weights are integral powers of 2, a set of cycles $\overline{S}$ where each cycle is edge disjoint from G, a partial colouring $\overline{\mathbf{x}} \in (-1,1)^{\overline{S}}$, a graph $\vec{G}' = \vec{G}+$ ColourWeights$(\overline{S}, \overline{\mathbf{x}})$, a 2-approximate effective resistances $\mathbf{r}$ in G'.

Output: A directed Eulerian graph $\vec{H}$ where edge weights are integral powers of 2, a set of cycles $\overline{T}$ where each cycle is edge disjoint from H, a partial colouring $\overline{\mathbf{z}} \in (-1,1)^{\overline{T}}$, a graph $\vec{H}' = \vec{H}+$ ColourWeights$(\overline{T}, \overline{z})$.

1 Let $\vec{H} \leftarrow \vec{G}$ with only the edges which satisfy $w_e r_e > \frac{16n}{m'}$ and remove them from $\vec{G}$.

2 Partition $\vec{G}$ into uniformly weighted graph $\vec{G}_1, \ldots, \vec{G}_q$ where $\vec{G}_i$ has all edge weights 2^i and $q = O(\log n)$.

3 Let S be the set of all cycles after applying CycleDecomposition on $\vec{G}_1, \ldots, \vec{G}_s$ and set $\vec{H} \leftarrow \vec{H} + \sum_{i=1}^{s} \vec{G}_i \setminus \left(\bigcup_{j=1}^{t} \vec{C}_{i,j}\right)$.

4 $T', \overline{T}', \mathbf{y}, \overline{\mathbf{y}} \leftarrow$ ColourTarget$(S, \mathbf{0}, \frac{1}{8}m)$.

5 If ColourWeights $(T', \mathbf{y})$ has more edges than ColourWeights $(T', -\mathbf{y})$, we take $\mathbf{y} \leftarrow -\mathbf{y}$ and $\overline{\mathbf{y}} \leftarrow -\overline{\mathbf{y}}$.

6 $\vec{H} \leftarrow \vec{H} +$ ColourWeights$(T', \mathbf{y})$.

7 $\overline{T} \leftarrow \overline{T}' \cup \overline{S}$ and set $\overline{\mathbf{z}} \leftarrow \overline{\mathbf{y}} + \overline{\mathbf{x}}$.

8 $\vec{H}' \leftarrow \vec{H}+$ ColourWeights$(\overline{T}, \overline{\mathbf{z}})$.

9 **return** $\vec{H}, \vec{H}', \overline{T}, \overline{\mathbf{z}}$.

Proof. For the degree condition, it suffices to consider a single cycle C and show that the reweighted directed cycle, say $\vec{C}'$ in line 3 preserves the differences of the in and out degrees of $\vec{C}$. Recall the definition of $\vec{F}$ and S of C, see CorrectOrientation,and the argument in Section 3 for showing degree differences preservation under the special case of $x \in \{\pm 1\}$. Note first that the edge weights are the same. Either a vertex has an incoming and outgoing edge (and so difference 0), in which case both edges are either in $\vec{F} \setminus S$ or in S and hence always added together with the same weights of (so still difference 0). Alternatively a vertex has two incoming or outgoing edges, in which case one edge gets a new weight of $1 + x$ and the other gets $1 - x$, which then still preserves the difference between in and out degree.

If $x \in \{\pm 1\}$ the edge weights of $\vec{C}'$ is exactly twice that of C unless $\vec{C}'$ is emtpy. Thus, $\vec{H}$ still has integral edge weights with largest weight at most doubled. ◀

For the rest of this section, we refer to a set of uniformly weighted cycles (two cycles can have different weights) as a set of cycles for simplicity. We write $m(S) = \sum_{C \in S} |E(C)|$ as the total number of edges in S. In ColourSparsify, ColourSparsifyGraph and ColourSparsifyCycle, by applying $'$ to a graph we mean $\vec{G}' = \vec{G}+$ ColourWeights$(\overline{S}, \overline{x})$. We denote m' as the number of edges in $\vec{G}'$. Note that this is the primary number of edges we consider rather than m.

Towards analyzing ColourSparsify, we first state the guarantees of the ColourTarget subroutine which guarantees a partial colouring of at least a specified size.

▶ **Lemma 14.** *The outputs of* ColourTarget(S, y, m_t) *satisfy that* $m(\overline{S}) \le m_t$ *and the number of calls to* PartialColour *is* $O\left(\log(\frac{|S|L}{m_t})\right)$. *If additionally the set of cycles* S *satisfies* $\sum_{C \in S} \|\mathbf{A}(C)\| \le \sigma^2$ *and* $\sum_{C \in S} \|\mathbf{A}(C)\|_F^2 \le v$, *then the outputs also satisfy*

$$\left\| \sum_{C \in S} (x + \overline{x} - y)\mathbf{A}(C) \right\| \le O\left(\sigma \cdot \log\left(\frac{|S|L}{m_t}\right) + (\log^{\frac{3}{4}} n)\sigma^{\frac{1}{2}} \left(\frac{vL}{m_t}\right)^{\frac{1}{4}} \right)$$

Algorithm 7 ColourSparsifyCycle$(\vec{G}, \vec{G}', \overline{S}, \overline{\mathbf{x}}, \mathbf{r})$.

Input: A directed Eulerian graph $\vec{G}$ where edge weights are integral powers of 2, a set of cycles $\overline{S}$ where each cycle is edge disjoint from G, a partial colouring $\overline{\mathbf{x}} \in (-1,1)^{\overline{S}}$, a graph $\vec{G}' = \vec{G} +$ ColourWeights$(\overline{S}, \overline{\mathbf{x}})$, a 2-approximate effective resistances $\mathbf{r}$ in G'.

Output: A directed Eulerian graph $\vec{H}$ where edge weights are integral powers of 2, a set of cycles $\overline{T}$ where each cycle is edge disjoint from H, a partial colouring $\overline{\mathbf{z}} \in (-1,1)^{\overline{T}}$, a graph $\vec{H}' = \vec{H} +$ ColourWeights$(\overline{T}, \overline{z})$.

Set $\overline{S}'$ be an empty set of cycles initialy. For each $C \in \overline{S}$, let C' be C with its weight by $(1 - |\overline{x}_C|)$ and add C' to $\overline{S}'$.

$T', \overline{T}', \mathbf{y}, \overline{\mathbf{y}} \leftarrow$ ColourTarget$(\overline{S}', \mathbf{0}, \frac{1}{4}m')$

if $m(\{C' \in T' : |\overline{x}_C - (1 - |\overline{x}_C|)y_{C'}| = 1\}) > m(\{C' \in T' : |\overline{x}_C + (1 - |\overline{x}_C|)y_{C'}| = 1\})$ **then**

  $\mathbf{y} \leftarrow -\mathbf{y}, \overline{\mathbf{y}} \leftarrow -\overline{\mathbf{y}}$.

Set $\mathbf{z}, \overline{\mathbf{z}}$ to be the parts of $\overline{\mathbf{x}} + (1 - |\overline{\mathbf{x}}|) \circ (\mathbf{y} + \overline{\mathbf{y}})$ with magnitude 1 and < 1 respectively. Here we abused $\circ$ to let C and C' refering to the same index, Set the partition $T, \overline{T}$ of $\overline{S}$ accordingly.

$\vec{H} \leftarrow \vec{H} +$ ColourWeights$(T, \mathbf{z})$.

$\vec{H}' \leftarrow \vec{H} +$ ColourWeights$(\overline{T}, \overline{\mathbf{z}})$.

return $\vec{H}, \vec{H}', \overline{T}, \overline{\mathbf{z}}$.

Proof. Notice that each cycle has its number of edges bounded by L. We have $m(\overline{S}) \leq L|\overline{S}| \leq m_t$ by the terminating condition of the while loop in ColourTarget. Since the size of $\overline{S}$ decreases by a factor of $1 - c'$ by Lemma 4, by the ith round we have $|\overline{S}| \leq (1 - c')^i |S|$ and at termination this is $\leq \frac{m_t}{L}$. This then gives the claimed number of iterations.

Consider the error bound. Combine the number of iterations with the first term in (1) of Lemma 4, we get our desired first term. For the second term, recall from above that $|\overline{S}|$ decreases geometrically. Then $f = \left(\frac{v}{|\overline{S}|}\right)^{\frac{1}{2}}$ increases exponentially over the iterations. Hence the sum of the second terms in (1) is bounded by the last one with $f = O\left((\frac{vL}{m_t})^{\frac{1}{2}}\right)$, giving

$$O\left((\log^{\frac{3}{4}} n)\sigma^{\frac{1}{2}} f^{\frac{1}{2}}\right) = O\left((\log^{\frac{3}{4}} n)\sigma^{\frac{1}{2}} \left(\frac{vL}{m_t}\right)^{\frac{1}{4}}\right)$$

as required. ◀

Now, parallel to Lemma 11, we state the approximation guarantees of ColourSparsifyGraph and ColourSparsifyCycle in Lemmas 15 and 16. The proof of Lemma 15 follows closely to that of Lemma 16 and we refer reader to the full version of our paper.

▶ **Lemma 15.** *If the input graphs $\vec{G}, \vec{G}'$ satisfy $4m \geq m'$ and the input set of cycles $\overline{S}$ and it corresponding partial colours $\overline{x}$ satisfies that each cycle $C \in \overline{S}$ has $w_e r_e \leq \frac{4n}{m'}$ for each edge $e \in C$, the algorithm* ColourSparsifyGraph *returns $\vec{H}$ with edge weights still being powers of 2 and at most twice the largest weight in $\vec{G}$, a set of cycles $\overline{T}$ with its corresponding partial colours $\overline{\mathbf{z}}$ satisfying $\vec{H}' = \vec{H} +$ ColourWeights$(\overline{T}, \overline{\mathbf{z}})$ is an Eulerian graph and each cycle $C \in \overline{T}$ also has $w_e r_e \leq \frac{4n}{m'_H}$ for each edge $e \in C$, where $m'_H = |E(\vec{H})|$. and,*

$$\left\| \mathbf{L}_{G'}^{\frac{+}{2}} (\mathbf{L}_{\vec{G}'} - \mathbf{L}_{\vec{H}'}) \mathbf{L}_{G'}^{\frac{+}{2}} \right\| \leq O\left(\sqrt{\frac{n \log^2 n}{m'}} \log\log n + \left(\frac{n \log^{\frac{8}{3}} n}{m'} \right)^{\frac{3}{4}} \right).$$

Algorithm 8 ColourTarget(S, y, m_t).

Input: A set of cycles S of size $s = |S|$, a partial colouring $y \in (-1,1)^S$, and a target mass of m_t edges.

Output: A set of fully coloured cycles $S\backslash\overline{S}$ with colouring x, A set of partially coloured cycles $\overline{S}$ with colouring $\overline{x}$ satisfying $\overline{x} \in (-1,1)^{\overline{S}}$.

1 Initialize $x = 0$ be a empty colouring over S.
2 Define $\overline{S}$ to always be the set of fractionally coloured cycles in S and let $\overline{s} = |\overline{S}|$ always. Set $\overline{x}$ be the partial colour on $\overline{S}$ always.
3 **while** $\overline{s} > \frac{m_t}{L}$ **do**
4 $\quad x[\overline{S}] \leftarrow$ PartialColour$(\mathcal{A}[\overline{S}], \overline{x})$.
5 Let $\overline{x} \leftarrow x$ with entries of magnitude < 1 and set $x \leftarrow x - \overline{x}$.
6 **return** $S\backslash\overline{S}, \overline{S}, x, \overline{x}$.

▶ **Lemma 16.** *If the input set of cycles $\overline{S}$ and it corresponding partial colours $\overline{x}$ satisfies that each cycle $C \in \overline{S}$ has $w_e r_e \le \frac{4n}{m'}$ for each edge $e \in C$, the algorithm* ColourSparsifyCycle *returns $\vec{H}$ with edge weights still being powers of 2 and at most twice the largest weight in $\vec{G}$, a set of cycles $\overline{T}$ with its corresponding partial colours $\overline{y}$ satisfying $\vec{H}' = \vec{H} +$* ColourWeights$(\overline{T}, \overline{y})$ *is an Eulerian graph and each cycle $C \in \overline{T}$ also has $w_e r_e \le \frac{4n}{m'_H}$ for each edge $e \in C$, where $m'_H = |E(\vec{H})|$. and,*

$$\left\| \mathbf{L}_{G'}^{\frac{+}{2}} (\mathbf{L}_{\vec{G}'} - \mathbf{L}_{\vec{H}'}) \mathbf{L}_{G'}^{\frac{+}{2}} \right\| \le O\left(\sqrt{\frac{n \log^2 n}{m'}} \log\log n + \left(\frac{n \log^{\frac{8}{3}} n}{m'} \right)^{\frac{3}{4}} \right).$$

Before we prove Lemma 16, we need Lemma 17 regarding scaling matrices in the set of extra cycles $\overline{S}$.

▶ **Lemma 17.** *For directed Eulerian graph $\vec{G}$, a set of cycles $\overline{S}$ where each cycle $C \in \overline{S}$ satisfies that $\vec{G}$ and $\vec{C}$, the corresponding directed cycle of C, are edge-disjoint. Let $\overline{x} \in (-1,1)^S$ be a fractional colouring on $\overline{S}$. Then the Eulerian graph $\vec{G}' = \vec{G} +$* ColourWeights$(\overline{S}, \overline{x})$ *satisfies*

$$\mathbf{L}_G + \sum_{C \in \overline{S}} (1 - |\overline{x}_C|) \mathbf{L}_C \preceq \mathbf{L}_{G'}.$$

Proof. For any $C \in \overline{S}$, let $\vec{C}' =$ ColourWeights$(C, \overline{x}_C)$ where we abused the definition to take in a single cycle instead of a set of cycles. Note that the undirectification $\mathbf{L}_{G'} = \mathbf{L}_G + \sum_{C \in \overline{S}} \mathbf{L}_{C'}$. Since $|\overline{x}_C| < 1$, all edges in C must be present in C' and the minimum edge weight is at least $1 - |\overline{x}_C|$ times the original uniform edge weights of C. Then, $(1 - |\overline{x}_C|)\mathbf{L}_C \preceq \mathbf{L}_{C'}$. Summing over all C, we get

$$\mathbf{L}_G + \sum_{C \in \overline{S}} (1 - |\overline{x}_C|) \mathbf{L}_C \preceq \mathbf{L}_G + \sum_{C \in \overline{S}} \mathbf{L}_{C'} = \mathbf{L}_{G'}. \qquad \blacktriangleleft$$

Proof of Lemma 16. The edge weights condition of $\vec{H}$ is guaranteed by Lemma 13. Also by Lemma 13, both $\vec{H}$ and $\vec{H}'$ are Eulerian. Observe that $m'_H \le m$ always, and $\overline{T} \subset \overline{S}$. Then, the output cycles still satisfy the approximate leverage score condition. Now, by line 5, the output Eulerian graph $\vec{H}'$ satisfies

$$\mathsf{hlift}\left(\mathbf{L}_{G'}^{\frac{+}{2}} (\mathbf{L}_{\vec{H}'} - \mathbf{L}_{\vec{G}'}) \mathbf{L}_{G'}^{\frac{+}{2}} \right) = \sum_{C \in \overline{S}} (z_C + \overline{z}_C - x_C) \mathbf{A}(C) = \sum_{C \in \overline{S}} (1 - |x_C|)(y_C + \overline{y}_C) \mathbf{A}(C)$$

where all vectors are taken as the final values. By definition $\mathbf{A}(C') = (1 - |x_C|)\mathbf{A}(C)$ and

$$\sum_{C \in \overline{S}} (1 - |x_C|)(y_C + \overline{y}_C)\mathbf{A}(C) = \sum_{C' \in \overline{S}'} (y_C + \overline{y}_C)\mathbf{A}(C')$$

By definition of hlift, each matrix $\mathbf{A}(C)^2$ is block diagonal with blocks $\mathbf{L}_{G'}^{\pm\frac{2}{2}} \widetilde{\mathbf{L}}_{\vec{C}}^{\top} \mathbf{L}_{G'}^{+} \widetilde{\mathbf{L}}_{\vec{C}} \mathbf{L}_{G'}^{\pm\frac{2}{2}}$ and $\mathbf{L}_{G'}^{\pm\frac{2}{2}} \widetilde{\mathbf{L}}_{\vec{C}} \mathbf{L}_{G'}^{+} \widetilde{\mathbf{L}}_{\vec{C}}^{\top} \mathbf{L}_{G'}^{\pm\frac{2}{2}}$. Here $\widetilde{\mathbf{L}}_{\vec{C}} = \mathbf{L}_{\vec{F}} - \mathbf{L}_S$ with fixed orientation (recall CorrectOrientation). Since every cycle $C \in \overline{S}$ satisfies $\boldsymbol{\tau}_{G'}(e) \leq \rho$ for each $e \in C$, by Lemma 10, both matrices are spectrally bounded by $O(L\rho) \cdot \mathbf{L}_{G'}^{\pm\frac{2}{2}} \mathbf{L}_C \mathbf{L}_{G'}^{\pm\frac{2}{2}}$. Thus, by the disjointness of G and $\overline{S}$,

$$\begin{aligned}\sum_{C' \in \overline{S}'} \mathbf{A}(C')^2 &\preceq \sum_{C \in \overline{S}} (1 - |x_C|)\mathbf{A}(C)^2 \preceq O(L\rho) \cdot \mathbf{I}_2 \otimes \mathbf{L}_{G'}^{\pm\frac{2}{2}} \left(\sum_{C \in \overline{S}} (1 - |x_C|)\mathbf{L}_C \right) \mathbf{L}_{G'}^{\pm\frac{2}{2}} \\ &\preceq O(L^2\rho) \cdot \mathbf{I}_{2n},\end{aligned}$$

where we used the PSD property of $\mathbf{A}(C)^2$ and the fact $1 - |x_C| \leq 1$ for the first inequality and Lemma 17 for the second inequality. The sum of Frobenius norm squared is then

$$\sum_{C' \in \overline{S}'} \|\mathbf{A}(C')\|_F^2 \leq \sum_{C \in \overline{S}} (1 - |x_C|) \operatorname{Tr}\left(\mathbf{A}(C)^2\right) = \operatorname{Tr}\left(\sum_{C \in \overline{S}} (1 - |x_C|)\mathbf{A}(C)^2 \right) = O(nL^2\rho).$$

We can now apply Lemma 14 with $m_t = \frac{1}{4}m', \sigma^2 = O(L^2\rho)$ and $v = O(nL^2\rho)$ to get

$$\begin{aligned}\left\| \sum_{C' \in \overline{S}'} (y_C + \overline{y}_C - 0)\mathbf{A}(C') \right\| &\leq O\left(\sqrt{L^2\rho} \cdot \log\left(\frac{4|\overline{S}|L}{m'} \right) + (\log^{\frac{3}{4}} n)(L^2\rho)^{\frac{1}{4}} \left(\frac{4nL^3\rho}{m'} \right)^{\frac{1}{4}} \right) \\ &= O\left(\sqrt{\frac{nL^2}{m'}} \log L + \left(\frac{nL^{\frac{5}{3}} \log n}{m'} \right)^{\frac{3}{4}} \right)\end{aligned}$$

where we used $|\overline{S}'| = |\overline{S}| \leq m'$. Finally, note that

$$\left\| \mathbf{L}_{G'}^{\pm\frac{2}{2}} (\mathbf{L}_{\vec{G}'} - \mathbf{L}_{\vec{H}'}) \mathbf{L}_{G'}^{\pm\frac{2}{2}} \right\| = \left\| \mathsf{hlift}\left(\mathbf{L}_{G'}^{\pm\frac{2}{2}} (\mathbf{L}_{\vec{G}'} - \mathbf{L}_{\vec{H}'}) \mathbf{L}_{G'}^{\pm\frac{2}{2}} \right) \right\| = \left\| \sum_{C' \in \overline{S}'} (y_C + \overline{y}_C - 0)\mathbf{A}(C) \right\|. \blacktriangleleft$$

The sparsification induced by ColourSparsifyGraph is conditional, and we state the condition and sparisification induced in Lemma 18. However, even when the condition is not met, we are guaranteed each ColourSparsifyCycle will geometrically make progress towards satisfying the condition needed for Lemma 18. This is stated in Lemma 19.

▶ **Lemma 18.** *For inputs $\vec{G}, \vec{G}', \overline{S}, \overline{x}, \mathbf{r}$ to* ColourSparsifyGraph *satisfying that $4m \geq m' \geq \Omega(n \log^2 n)$, the outputs satisfy that the number of edges in $\vec{H}'$ is upperbounded by $m'_H \leq \frac{63}{64}m'$.*

Proof. Since $\mathbf{r}$ is 2-approximate effective resistances, $\sum_e w_e r_e \leq 2(n-1)$, we have at most $\frac{1}{8}m' \leq \frac{1}{2}m$ edges are removed from $\vec{G}$ in line 1. Since $m \geq \frac{1}{4}m' = \Omega(n \log^2 n)$ and the number of edges not in any cycle is $\hat{m}q = O(n \log^2 n)$, by picking an appropriate constant in $\Omega(n \log^2 n)$, we can guarantee the total number of edges in all cycles satisfies $m(S) \geq \frac{1}{4}m$. Lemma 14 then guarantees $m(\overline{T}') \leq \frac{1}{8}m$ and that $m(T') \geq \frac{1}{8}m$.

Now, by ColourWeights, the total number of edges in ColourWeights$(T', \mathbf{y})$ and ColourWeights$(T', -\mathbf{y})$ is exactly $m(T')$. Thus, line 5 means at least $\frac{1}{2}m(T') \geq \frac{1}{16}m \geq \frac{1}{64}m'$ edges are removed in total as required. ◀

▶ **Lemma 19.** *If inputs $\vec{G}, \vec{G}', \overline{S}, \overline{x}, \mathbf{r}$ to* ColourSparsifyCycle *satisfies that $4m < m'$, then either the number of edges in $\vec{H}'$ decreases to $m'_H \leq \frac{63}{64}m'$, or the number of edges in $\vec{H}$ satisfies $4m_H \geq m'_H$.*

Proof. Suppose $m'_H > \frac{63}{64}m'$. By Lemma 14, $m(\overline{T'}) \leq \frac{1}{8}m'$. Since $\mathbf{y} \in \{\pm 1\}^{T'}$, we have $\{C' \in T' : |\overline{x}_C - (1 - |\overline{x}_C|)y_{C'}| = 1\} \cup \{C' \in T' : |\overline{x}_C + (1 - |\overline{x}_C|)y_{C'}| = 1\} = T'$. Let the two sets above be T'_1 and T'_2, Then, $m(T'_1) + m(T'_2) \geq m(T')$ [2]. This means, after re-adjusting the colouring in line 4,

$$m(\overline{T}) \leq \frac{1}{2}m(T') + m(\overline{T'}) \leq \frac{1}{2}m' + \frac{1}{8}m' = \frac{5}{8}m' \leq \frac{40}{63}m'_H.$$

Then, we get the desired inequality, $m_H = m'_H - m(\overline{T}) \geq \frac{23}{63}m'_H \geq \frac{1}{4}m'_H$. ◀

With these analyses above, we can now formally state and prove Theorem 2.

▶ **Theorem 20** (Theorem 2 Formal)**.** *Given input a Eulerian graph $\vec{G}$ with polynomial bounded integral edge weights and $\varepsilon \in (0, \frac{1}{2})$, the algorithm* ColourSparsify *returns in polynomial time a Eulerian graph $\vec{H}$ with $O(n\varepsilon^{-2}\log^2 n(\log\log n)^2 + n\varepsilon^{-\frac{3}{4}}\log^{\frac{8}{3}} n)$ edges satisfying*

$$\left\| \mathbf{L}_{\vec{G}}^{\frac{+}{2}}(\mathbf{L}_{\vec{G}} - \mathbf{L}_{\vec{H}})\mathbf{L}_{\vec{G}}^{\frac{+}{2}} \right\| \leq \varepsilon.$$

Proof. By Lemmas 18 and 19, in every two iterations the number of edges must decreases by at least a constant fraction, as the condition $4m \geq m'$ must be satisfied at least once. Note that initialy $m = m' \geq \frac{1}{4}m'$ is satisfied. Thus, the total number of iterations is at most $O\left(\log(\frac{m\log n}{n})\right) = O(\log n)$ where the extra $\log n$ comes from the decomposition by weights.

By Lemmas 15 and 16, the largest edge weight doubles each iteration. Thus, the edge weights in each $\vec{G}$ are still integral and polynomially bounded over $O(\log n)$ iterations.

As the number of edges decreases geometrically every $O(1)$ iterations, the total error is asymptotically bounded by the error in the last round for both terms in Lemmas 15 and 16:

$$O\left(\sqrt{\frac{n\log^2 n}{m'}}\log\log n + \left(\frac{n\log^{\frac{8}{3}} n}{m'}\right)^{\frac{3}{4}}\right).$$

where m' is the number of edges in $\vec{G}'$ in the last round. Since the algorithm stops at $m' \geq \Omega(n\varepsilon^{-2}\log^2 n(\log\log n)^2)$ and $m' \geq \Omega(n\varepsilon^{-\frac{3}{4}}\log^{\frac{8}{3}} n)$ edges, the largest of both terms must be bounded by $\frac{1}{2}\varepsilon$ by picking appropriate constant for the stopping condition.

This small error also implies that our 1.5-approximate effective resistances $\mathbf{r}$ stays as 2-approximate throughout the algorithm. Then, by Lemma 15 and Lemma 16, the set of cycles $\overline{S}$ always satisfy $w_e r_e \leq \frac{4n}{m'}$ where m' is the number of edges in $\vec{G}'$ throughout as required. Lemma 4 guarantees the polynomial running time of our algorithm. ◀

4.1 Conjectural improvements

In this section we consider an improvement on our existential results due to the partial colouring conjecture, Conjecture 21. Corollary 22 then follows by changing the termination condition of the while loop on line 5 to $m' \geq O(n\varepsilon^{-2}\log^2 n(\log\log n)^2)$.

[2] Contrary to the proof of Lemma 18, this is an inequality since magnitude of 1 can be achieved using both $y_{C'}$ and $-y_{C'}$ if $x_C = 0$.

▶ **Conjecture 21** (Matrix partial colouring conjecture). *There exists constants $c_1, c_2 > 0$ and $c_3 > 1$ such that the following holds. Given symmetric matrices $\mathbf{A}_1, \ldots, \mathbf{A}_m \in \mathbb{R}^{n \times n}$ that satisfy $m \geq c_3 n$, $\|\sum_{i=1}^m \mathbf{A}_i^2\| \leq \sigma^2$, and a point $\mathbf{y} \in (-1,1)^m$, there exists a point $\mathbf{x} \in [-1,1]^m$ such that $|\{i : x_i \in \{\pm 1\}\}| > c_2 m$ and*

$$\left\| \sum_{i=1}^m (x_i - y_i)\mathbf{A}_i \right\| \leq c_1 \sigma. \tag{3}$$

▶ **Corollary 22.** *Assume Conjecture 21. There is an algorithm that given a Eulerian graph $\vec{G}$, computes a ε-Eulerian sparsifier of $\vec{G}$ with $n\varepsilon^{-2}\log^2 n$ edges (up to $\log\log n$ factors).*

▶ Remark 23. While improvements on matrix concentration results for Gaussian random variables [5] naturally leads to improved matrix partial colouring through the Gaussian measure analysis of matrix discrepancy bodies, Conjecture 21 need not rely on this approach (e.g. [9, 15]). On the other hand, even if the matrix concentration guarantees of [5] hold for Rademacher random variables, it does not lead to an efficient algorithm for Theorem 2. This is due to the difficulties in controlling the matrix covariance factor in Theorem 1.2 of [5]. We refer reader to the proof of Lemma 3.1 in [6].

References

1. AmirMahdi Ahmadinejad, Arun Jambulapati, Amin Saberi, and Aaron Sidford. *Perron-Frobenius Theory in Nearly Linear Time: Positive Eigenvectors, M-matrices, Graph Kernels, and Other Applications*, pages 1387–1404. Society for Industrial and Applied Mathematics, 2019. doi:10.1137/1.9781611975482.85.
2. AmirMahdi Ahmadinejad, John Peebles, Edward Pyne, Aaron Sidford, and Salil Vadhan. Singular value approximation and sparsifying random walks on directed graphs. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 846–854, 2023. doi:10.1109/FOCS57990.2023.00054.
3. Zeyuan Allen-Zhu, Zhenyu Liao, and Lorenzo Orecchia. Spectral sparsification and regret minimization beyond matrix multiplicative updates. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 237–245, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2746539.2746610.
4. Alexandr Andoni, Jiecao Chen, Robert Krauthgamer, Bo Qin, David P. Woodruff, and Qin Zhang. On sketching quadratic forms. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, ITCS '16, pages 311–319, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2840728.2840753.
5. Afonso S Bandeira, March T Boedihardjo, and Ramon van Handel. Matrix concentration inequalities and free probability. *Inventiones mathematicae*, pages 1–69, 2023.
6. Nikhil Bansal, Haotian Jiang, and Raghu Meka. Resolving matrix spencer conjecture up to poly-logarithmic rank. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 1814–1819, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3564246.3585103.
7. Joshua Batson, Daniel A Spielman, and Nikhil Srivastava. Twice-Ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012.
8. Joshua Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM Rev.*, 56(2):315–334, January 2014. doi:10.1137/130949117.
9. József Beck. Roth's estimate of the discrepancy of integer sequences is nearly sharp. *Combinatorica*, 1(4):319–325, 1981.
10. András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in Õ(n2) time. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 47–55, New York, NY, USA, 1996. Association for Computing Machinery. doi:10.1145/237814.237827.

11 Timothy Chu, Yu Gao, Richard Peng, Sushant Sachdeva, Saurabh Sawlani, and Junxing Wang. Graph sparsification, spectral sketches, and faster resistance computation, via short cycle decompositions. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 361–372, 2018. doi:10.1109/FOCS.2018.00042.

12 Michael B. Cohen, Jonathan Kelner, Rasmus Kyng, John Peebles, Richard Peng, Anup B. Rao, and Aaron Sidford. Solving directed laplacian systems in nearly-linear time through sparse lu factorizations. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 898–909, 2018. doi:10.1109/FOCS.2018.00089.

13 Michael B. Cohen, Jonathan Kelner, John Peebles, Richard Peng, Anup B. Rao, Aaron Sidford, and Adrian Vladu. Almost-linear-time algorithms for markov chains and new spectral primitives for directed graphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 410–419, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3055399.3055463.

14 Michael B. Cohen, Jonathan Kelner, John Peebles, Richard Peng, Aaron Sidford, and Adrian Vladu. Faster algorithms for computing the stationary distribution, simulating random walks, and more. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 583–592, 2016. doi:10.1109/FOCS.2016.69.

15 Daniel Dadush, Haotian Jiang, and Victor Reis. A new framework for matrix discrepancy: Partial coloring bounds via mirror descent. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 649–658, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3519967.

16 David Durfee, Rasmus Kyng, John Peebles, Anup B Rao, and Sushant Sachdeva. Sampling random spanning trees faster than matrix multiplication. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 730–742. ACM, 2017. arXiv:1611.07451.

17 David Durfee, John Peebles, Richard Peng, and Anup B. Rao. Determinant-preserving sparsification of SDDM matrices with applications to counting and sampling spanning trees. In *FOCS*, pages 926–937. IEEE Computer Society, 2017. arXiv:1705.00985.

18 Samuel B. Hopkins, Prasad Raghavendra, and Abhishek Shetty. Matrix discrepancy from quantum communication. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 637–648, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3519954.

19 Arun Jambulapati, Victor Reis, and Kevin Tian. *Linear-Sized Sparsifiers via Near-Linear Time Discrepancy Theory*, pages 5169–5208. Society for Industrial and Applied Mathematics, 2023. doi:10.1137/1.9781611977912.186.

20 Arun Jambulapati and Aaron Sidford. Efficient $\tilde{O}(n/\epsilon)$ spectral sketches for the laplacian and its pseudoinverse. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 2487–2503, USA, 2018. Society for Industrial and Applied Mathematics.

21 Michael Kapralov and Rina Panigrahy. Spectral sparsification via random spanners. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 393–398, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2090236.2090267.

22 Ioannis Koutis, Alex Levin, and Richard Peng. Improved Spectral Sparsification and Numerical Algorithms for SDD Matrices. In Thomas Wilke Christoph Dürr, editor, *STACS'12 (29th Symposium on Theoretical Aspects of Computer Science)*, volume 14, pages 266–277, Paris, France, February 2012. LIPIcs. URL: https://hal.science/hal-00678205.

23 Ioannis Koutis and Shen Chen Xu. Simple parallel and distributed algorithms for spectral graph sparsification. *ACM Trans. Parallel Comput.*, 3(2), August 2016. doi:10.1145/2948062.

24 Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. Sparsified cholesky and multigrid solvers for connection laplacians. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 842–850, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2897518.2897640.

25 Rasmus Kyng, Simon Meierhans, and Maximilian Probst Gutenberg. Derandomizing directed random walks in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 407–418, 2022. doi:10.1109/FOCS54457.2022.00046.

26 Rasmus Kyng, Jakub Pachocki, Richard Peng, and Sushant Sachdeva. *A Framework for Analyzing Resparsification Algorithms*, pages 2032–2043. Society for Industrial and Applied Mathematics, 2017. doi:10.1137/1.9781611974782.132.

27 Rasmus Kyng and Sushant Sachdeva. Approximate gaussian elimination for laplacians - fast, sparse, and simple. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 573–582, October 2016. doi:10.1109/FOCS.2016.68.

28 Yin Tat Lee and He Sun. Constructing linear-sized spectral sparsification in almost-linear time. In *Proceedings of the 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, FOCS '15, pages 250–269, USA, 2015. IEEE Computer Society. doi:10.1109/FOCS.2015.24.

29 Yin Tat Lee and He Sun. An sdp-based algorithm for linear-sized spectral sparsification. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 678–687, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3055399.3055477.

30 Avi Levy, Harishchandra Ramadas, and Thomas Rothvoss. Deterministic discrepancy minimization via the multiplicative weight update method. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 380–391. Springer, 2017.

31 Huan Li and Zhongzhi Zhang. Kirchhoff index as a measure of edge centrality in weighted networks: Nearly linear time algorithms. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2377–2396. SIAM, 2018. arXiv:1708.05959.

32 Yang P. Liu, Sushant Sachdeva, and Zejun Yu. Short cycles via low-diameter decompositions. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 2602–2615, USA, 2019. Society for Industrial and Applied Mathematics.

33 Adam W Marcus, Daniel A Spielman, and Nikhil Srivastava. Interlacing families ii: Mixed characteristic polynomials and the kadison—singer problem. *Annals of Mathematics*, pages 327–350, 2015.

34 Raghu Meka. Discrepancy and beating the union bound. *Windows on theory, a research blog*, 2014.

35 Merav Parter and Eylon Yogev. Optimal Short Cycle Decomposition in Almost Linear Time. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 89:1–89:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2019.89.

36 Richard Peng and Zhuoqing Song. Sparsified block elimination for directed laplacians. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 557–567, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3520053.

37 Richard Peng and Daniel A. Spielman. An efficient parallel solver for sdd linear systems. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 333–342, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591832.

38 Victor Reis and Thomas Rothvoss. Linear size sparsifier and the geometry of the operator norm ball. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, pages 2337–2348, USA, 2020. Society for Industrial and Applied Mathematics.

39 Victor Reis and Thomas Rothvoss. Vector balancing in lebesgue spaces. *Random Structures & Algorithms*, 62(3):667–688, 2023.

40 Sushant Sachdeva and Yibin Zhao. A simple and efficient parallel laplacian solver. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '23, pages 315–325, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3558481.3591101.

41 Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 2616–2635, USA, 2019. Society for Industrial and Applied Mathematics.

42 D. Spielman and N. Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011. doi:10.1137/080734029.

43 Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 563–568, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1374376.1374456.

44 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 81–90, New York, NY, USA, 2004. Association for Computing Machinery. doi:10.1145/1007352.1007372.

45 Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM J. Comput.*, 40(4):981–1025, July 2011. doi:10.1137/08074489X.

46 Joel A Tropp. User-friendly tail bounds for sums of random matrices. *Foundations of computational mathematics*, 12(4):389–434, 2012.

47 Anastasios Zouzias. A matrix hyperbolic cosine algorithm and applications. In *International Colloquium on Automata, Languages, and Programming*, pages 846–858. Springer, 2012.

Caching Connections in Matchings

Yaniv Sadeh
Tel Aviv University, Israel

Haim Kaplan
Tel Aviv University, Israel

Abstract

Motivated by the desire to utilize a limited number of configurable optical switches by recent advances in Software Defined Networks (SDNs), we define an online problem which we call the *Caching in Matchings* problem. This problem has a natural combinatorial structure and therefore may find additional applications in theory and practice.

In the *Caching in Matchings* problem our cache consists of k matchings of connections between servers that form a bipartite graph. To cache a connection we insert it into one of the k matchings possibly evicting at most two other connections from this matching. This problem resembles the problem known as *Connection Caching* [20], where we also cache connections but our only restriction is that they form a graph with bounded degree k. Our results show a somewhat surprising qualitative separation between the problems: The competitive ratio of any online algorithm for caching in matchings must depend on the size of the graph.

Specifically, we give a deterministic $O(nk)$ competitive and randomized $O(n \log k)$ competitive algorithms for caching in matchings, where n is the number of servers and k is the number of matchings. We also show that the competitive ratio of any deterministic algorithm is $\Omega(\max(\frac{n}{k}, k))$ and of any randomized algorithm is $\Omega(\log \frac{n}{k^2 \log k} \cdot \log k)$. In particular, the lower bound for randomized algorithms is $\Omega(\log n)$ regardless of k, and can be as high as $\Omega(\log^2 n)$ if $k = n^{1/3}$, for example. We also show that if we allow the algorithm to use at least $2k - 1$ matchings compared to k used by the optimum then we match the competitive ratios of connection catching which are independent of n. Interestingly, we also show that even a single extra matching for the algorithm allows to get substantially better bounds.

2012 ACM Subject Classification Theory of computation → Caching and paging algorithms; Mathematics of computing → Graph coloring

Keywords and phrases Caching, Matchings, Caching in Matchings, Edge Coloring, Online Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.120

Category Track A: Algorithms, Complexity and Games

Related Version *Extended Version*: `https://arxiv.org/abs/2310.14058` [30]

Funding The work of the authors is partially supported by Israel Science Foundation (ISF) grant number 1595-19, German Science Foundation (GIF) grant number 1367 and the Blavatnik research fund at Tel Aviv University.

1 Introduction

We define the *Caching in Matchings* online problem, on a fixed set of n nodes. Requests are edges between these node. The algorithm maintains a cache of k matchings, i.e. a k-edge-colorable graph. To serve a request for an edge (u, v) which is not in its cache (i.e. a miss), the algorithm has to insert it into one of its matchings. To do this it may need to evict the edges incident to u and v in this specific matching. Note that an evicted edge may later be re-inserted into a different matching. The algorithm has to choose which matching to use for each miss in order to minimize its total number of misses.

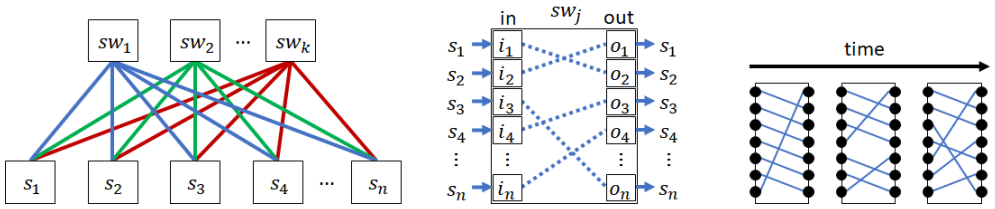



Figure 1 The physical topology that motivates our problem: n servers $s_1, \ldots, s_n$, each is connected to the in/out ports of k optical switches sw_1 through sw_k. Each switch sw_j uses mirrors to switch optical links, effectively inducing an in/out permutation, which may change over time at a reconfiguration cost of 1 per each new pairing. Abstractly, we get a bipartite graph with n nodes on each side (one per server), and each permutation is a matching that caches links.

One can look at this problem as a new variation of the online Connection Caching problem. In *Connection Caching* [20] the setup is the same, but the cache maintained by the algorithm must be a graph in which each node is of degree at most k. In case of a miss on an edge (u, v) we may choose any edge incident to u and any edge incident to v to evict. We do not have to maintain the edges partitioned into a particular set of k matchings. Thus in Caching in Matchings we are less flexible in our eviction decisions. Once we color the new edge then the two edges we have to evict are determined.

At a first glance, the two caching problems seem similar. In fact, the only difference is the added restriction of the coloring (matchings) that affects how the cache is maintained. Interestingly, it turns out that this seemingly small difference makes Caching in Matchings a much harder online problem compared to Connection Caching.

A common measure to evaluate online algorithms is their competitive ratio. We say that an online algorithm is c-competitive if its cost (in our case, miss count) on every input sequence is at most c times the minimal possible cost for serving this sequence. One would like to design algorithms with as small c as possible. The problem of Connection Caching is known to be $\Theta(k)$ (deterministic) and $\Theta(\log k)$ (randomized) competitive, and in contrast we show that the dependence on n (the number of nodes) in Caching in Matchings cannot be avoided.

The motivation to our Caching in Matchings problem comes from a data-center architecture described in [4]. In this setting we have n servers connected via a communication network which is equipped with a set O of k optical switches. Each server is connected to all the k optical switches and in each of them it is connected to both an input and an output port. Each switch is configured to implement a matching between the input and the output ports of the servers, see Figure 1. Since each server is connected to both input and output sides, the optical switches effectively induce a degree k bipartite graph with $2n$ nodes (two nodes per server). Each optical switch corresponds to a matching in our cache. It is dynamic as we can insert and evict connections from the switch, but we try to minimize these reconfigurations since they are costly (involve shifting mirrors, and down-time).

At this point we clarify that there are two "kinds" of optical switching architectures. The one which we model, as explained, is based on off-the-shelf commodity switches and is sometimes referred to as Optical Circuit Switching (OCS). Each switch is a separate box, and each box, at any time, implements a matching between its ports. We use k switches and connect every server to every switch, so this architecture induces k matching at any time. To add a connection between two servers we have to choose through which box we want to do it (choose a matching to insert it to) and then reconfigure the matching implemented by this particular box to include this edge. The other kind of switching is known as Free Space Optics (FSO) where every transmitter can point towards any receiver. When each server is

connected to k transmitters and k receivers we get the standard connection caching setting. This is *not* the architecture that we model here. See Table-1 of [26] for several references and their architecture types.

Several cost models considering both communication and adjustment cost were suggested for this setting [4]. We choose to work with arguably the simplest model of paying 1 for an insertion of a new edge (formally defined in Section 2). This simple model already captures the qualitative properties of the problem. We note that the competitive results shown here can be adapted (up to constant factors) to a more complicated cost model that has additional communication costs per request. We believe that our combinatorial abstraction of this setting is natural and will find additional applications.

Here is a detailed summary of our results.

Our contributions

1. We define a new caching problem, "Caching in Matchings" (Problem 1), on a bipartite graph with n nodes on each side.[1] In this problem, the cache is a union of k matchings. When we insert an edge we pick the matching to insert it to and evict edges from this matching if necessary.
2. We show that the competitive ratio of Caching in Matchings depends not only on the cache size k as is common for caching problems, but also on the number of nodes in the network n. One might argue that since we define the cache to be k matchings, its size is $\Theta(nk)$ rather than k, so the dependency on n is not surprising. But such an argument also applies to Connection Caching [20] and in that problem the competitive ratio does not depend on n. In other words, Caching in Matchings is provably harder than Connection Caching.[2] Specifically we prove the following.
   a. An $\Omega(\max(\frac{n}{k}, k))$ lower bound on the competitive ratio of deterministic algorithms, and we give a deterministic algorithm with nk competitive ratio. For $k = O(1)$ this gives a tight bound of $\Theta(n)$ on the competitive ratio.
   b. In contrast, in the randomized case we have a larger gap. We describe an $O(n \log k)$ competitive algorithm and prove a lower bound of $\Omega(\log \frac{n}{k^2 \log k} \cdot \log k)$ on the competitive ratio. This bound is $\Omega(\log n)$ for any k, and can get as worse as $\Omega(\log^2 n)$, for example if $k = n^{1/3}$. This is in contrast to other caching problems whose randomized competitive ratio is logarithmic.[3]
3. We show that resource augmentation of almost-twice as many matchings, specifically $2k - 1$ for the algorithm versus k for the optimum, allows to get rid of the dependence on n. Specifically, we show a deterministic $O(k)$ competitive algorithm and a randomized $O(\log k)$ competitive algorithm for this case. Furthermore, with $2(1+\alpha)k$ matchings we get a deterministic $O(1+\frac{1}{\alpha})$ competitive algorithm. We also show that a single extra matching already helps by allowing us to "trade" $\sqrt{n}$ for $\sqrt{k}$ in the competitive ratio. Concretely and more generally, with $h \geq 1$ extra matchings we get a deterministic $O(n^{1/2}(k/h)^{3/2})$ and a randomized $O(n^{1/2}(k/h)^{1/2} \log \frac{2k+h}{h})$ competitive algorithms. Moreover, it is even possible to reduce the dependence on n to polylogarithmic at the cost of higher polynomial dependency on k, which is beneficial for small k. Concretely, following [19], we get a deterministic $O\left(\frac{k^6 \log n}{h} \min(k, \log n)\right)$ and a randomized $O\left(\left(\frac{k \log k}{h}\right)^6 \log \frac{2k+h}{h} \log^9 n\right)$ competitive algorithms. The deterministic algorithm is not efficient.

[1] The problem makes sense on a general graph as well.
[2] In terms of the architecture, we show that the FSO architecture has a better competitive ratio than the the OCS architecture.
[3] Throughout the paper, where it matters, our logarithms are in base 2.

Our problem is a special case of a more general problem of convex body chasing in L_1. Bhattacharya et al. [11] gave a fractional algorithm for this body chasing problem with packing and covering constraints. Their fractional algorithm requires a slight resource augmentation. For a few special cases, they show how to round their fractional solution to an integral solution that does not use additional resources. Our problem is another interesting test-case of this general setting (for more details, see the appendix in [30]).

Our full list of results is summarized in Table 1. The rest of the paper is structured as follows. Section 2 formally defines the model, the notations that we use, and the caching problems. Section 3 studies in depth the Caching in Matchings problem (Problem 1). Section 4 surveys related work on caching and coloring problems, and in Section 5 we conclude and list a few open questions. Section 6 serves as an appendix that contains deferred proofs, and a few additional discussions. Due to a strict page limit, the complete appendix can be found in the extended version [30].

2 Model and Definitions

In the following we formally define two caching problems of interest, the premise of each of them is a graph with a set V of n nodes. Every turn, a new edge is requested. If it is already cached, we have a "hit" and no cost is paid. Otherwise, we have a "miss", and the edge must be brought into the cache at a cost of 1, possibly at the expense of evicting other edges. In fact, the problem that arises from [4] consists of a bipartite graph in which each server v is associated with two nodes $v^{in} \in V^{in}$ and $v^{out} \in V^{out}$, modeling its receiving and sending ports, respectively. Each among v^{in} and v^{out} can be incident to one edge in each matching.[4] Formally the problem is as follows.

▶ **Problem 1** (Caching in Matchings)**.** Requests arrive for edges $(u, v) \in V^{in} \times V^{out}$. The cache M is a union of k matchings. When a requested edge is missing from all the matchings, an algorithm must fetch it into one of the matchings (possibly evicting other edges from this matching). In addition, the algorithm may choose to add any edge to the cache at any time (while maintaining the cache's restrictions), the cost of adding an edge to the cache is 1. It is not allowed to move an edge between matchings, but an edge may be evicted and immediately re-fetched into a new matching.

▶ Remark 1. There are other caching models in which reorganizing the cache is free, such as [16, 25]. In our model reorganizing the matchings incurs a cost. This is because we model a setting where changing the cache (physical links) is slow. In other cases accessing the slow memory is the costly operation.

We use the terminology of coloring edges when discussing Caching in Matchings (Problem 1). Recoloring an edge implies that we evict it, and then immediately fetch it back into a different matching according to the new color of the edge. Recoloring is not free, but has the same cost of standard fetching. This models, for example, the physical setting in which such a rearrangement requires reconfiguring the link in a different optical switch.

▶ **Problem 2** (Connection Caching [20])**.** Requests arrive for edges $(u, v) \in V^{in} \times V^{out}$. The cache M is a set of edges such that every node is of degree at most k in the sub-graph induced by M. When a requested edge is missing from M, an algorithm must fetch it (possibly

[4] Technically, the physical switch can be configured with links of the form (v^{in}, v^{out}), but it makes no sense and practically such requests do not exist. However, our algorithms can deal with all possible requests, and our lower bounds are proven without relying on such requests, so we ignore this nuance.

evicting other edges). In addition, the algorithm may choose to add any edge to the cache at any time while maintaining the degrees at most k. Adding an edge to the cache costs 1.

▶ Remark 2. Note that in both problems that we defined, an algorithm is allowed to add (fetch) and remove (evict) additional edges. Technically, it is not strictly necessary because a non-lazy algorithm can always be simulated by a lazy version that fetches an edge only when it is actually needed. This is also true for the offline optimum. We will describe non-lazy algorithms for Caching in Matchings, that recolor edges, to simplify the presentation.

To emphasize the difference between the problems see Figure 2, which shows the difference on bipartite graphs, as well as on general graphs (for the generalized problem).

Figure 2 An example of the difference between connection caching versus caching in matchings. (left) With $k = 2$ max degree and $n = 3$ nodes, all the connections can be cached simultaneously, but not in 2 matchings, red and blue. (right) Bipartite example: Caching the edge (s_1^i, s_2^o) (i/o for in/out) is not possible without changing the matchings (red: $\{(s_1^i, s_3^o), (s_3^i, s_1^o)\}$; blue: $\{(s_3^i, s_2^o), (s_2^i, s_1^o)\}$), although both s_1^i and s_2^o only have a single connection.

The objective of an online algorithm is to minimize the number of fetched edges. We are interested in the competitive-ratio of our algorithms.

▶ **Definition 3** (Cost, Competitive Ratio)**.** Consider a specific caching problem. Let A be an online algorithm that serves requests, and let σ be a sequence of requests. We denote by $A(\sigma)$ the execution of A on σ, and $cost(A(\sigma))$ for the cost of A when processing σ.

We denote by $OPT(\sigma)$ the optimum (offline) algorithm to serve the sequence, or simply OPT when σ is clear from the context. If there exist functions of the problem's parameters (in our case: k and n) $c = c(n, k)$ and $d = d(n, k)$ such that $\forall\sigma : cost(A(\sigma)) \leq c \cdot cost(OPT(\sigma)) + d$ then we say that A is c-competitive. Note that σ may be arbitrarily long, so the "asymptotic ratio" is indeed c.

▶ Remark 4. Denote the optima for Caching in Matchings and Connection Caching by OPT_m and OPT_c, respectively. Since OPT_m implicitly maintains a connections cache as required by Connection Caching (ignore the colors), then for any sequence of edge requests σ, $cost(OPT_c(\sigma)) \leq cost(OPT_m(\sigma))$.

3 Caching in Matchings

In this section we study the problem of Caching in Matchings (Problem 1). We summarize the results of this section in Table 1. We start with upper bounds (Section 3.1), then lower bounds (Section 3.2). Then we study resource augmentation (Section 3.3). Some additional discussion on randomization is detailed in the appendix in [30].

3.1 Upper Bounds for Bipartite Graphs

In this section we prove upper bounds on the competitive ratio of algorithms for Caching in Matchings, focusing on the non-trivial case of $2 \leq k \leq n - 1$. Indeed, if $k = 1$ there are no eviction-decisions to take so the only (lazy) algorithm is the optimal one. The other

Table 1 Our bounds on the competitive ratio for Caching in Matchings (Problem 1), for $2 \leq k < n$. If $k = 1$ or $k \geq n$ optimality is trivial. Results marked with $*$ also apply to general graphs. "RA: x" shortens Resource-Augmentation, i.e., the algorithm has more matchings (x) than the optimum (k).

Result		Deterministic	Randomized	Notes
Thm. 7		$\leq nk$	$O(n \log k)$	The standard scenario
Thm. 9	$*$	.	$\Omega(\log \frac{n}{k^2 \log k} \cdot \log k)$	.
Cor. 10	$*$	$\Omega(max(\frac{n}{k}, k))$	$\Omega(\log n)$	Due to Theorems 8+9
Cor. 10	$*$	.	$\Omega(\epsilon \cdot \log n \cdot \log k)$	$k = O(n^{1/2-\epsilon})$; Due to Theorem 9
Cor. 25(1)		$O(n^{1/2}(k/h)^{3/2})$	$O(n^{1/2}(k/h)^{1/2} \log \frac{2k+h}{h})$	RA: $k+h$ for $1 \leq h \leq k$
Cor. 25(2)	$*$	$O\left(\frac{k^6 \log n}{h} \min(k, \log n)\right)$	$O\left(\left(\frac{k \log k}{h}\right)^6 \log \frac{2k+h}{h} \log^9 n\right)$	RA: $k+h$ for $1 \leq h \leq k$
Cor. 25(3)	$*$	$\leq k$	$O(\log k)$	RA: $2k-1$
Cor. 25(4)	$*$	.	$O(\alpha^4 \log k)$	RA: $(1+O(\frac{1}{\alpha}))k$ for $k \geq \Theta(\alpha^4 \log n)^{\Theta(\alpha \log \alpha)}$
Thm. 27	$*$	$O(1+\frac{1}{\alpha})$	.	RA: $(2+\alpha)k$ for $\alpha > 0$

extreme case of $k \geq n$ in bipartite graphs is also easy since we can just cache the entire graph: Number the nodes 0 to $n-1$ on each side, and use matching i to store edges from node j to $i+j$ modulo n.

Our general technique is to reduce the problem of Caching in Matchings to Connection Caching. Our algorithm, A_m, will run a Connection Caching algorithms A_c with cache parameter k to insert requested edges into the cache. Then, layered on top of A_c, we have the "coloring component" of A_m that chooses the color of the new edge, and also recolors existing edges in order to produce a proper Caching in Matchings algorithm. A_m can be thought of as an edge coloring algorithm in the dynamic graph settings, and in this context A_c is the adversary that tells A_m which edges are inserted and which are removed (with a guarantee of bounded degree k). As a consequence we would like to use algorithms that are efficient in terms of recoloring, to achieve the best competitive results. Unfortunately, since edge coloring of graphs of bounded degree k may require $k+1$ colors by Vizing's theorem, the dynamic graph coloring literature studies this coloring problem while typically allowing more than k colors. The number of extra colors ranges from $k+1$ colors [29, 10], to $(1+\epsilon)k$ colors [21, 19, 13], to $2k-1$ colors [9, 12], and sometimes even more [8, 31] (the last citations actually study vertex coloring). Extra colors correspond to resource augmentation, which we study later in Section 3.3.

▶ Remark 5. There are known algorithms that are k competitive deterministic and $O(\log k)$ competitive randomized for Connection Caching, as studied in [20].

▶ Remark 6. Due to Remark 4 and Remark 5, it suffices to analyze the cost ratio between A_m and A_c. A ratio of ρ implies a $\rho \cdot k$ deterministic and a $O(\rho \cdot \log k)$ randomized competitive algorithms for Caching in Matchings.

▶ **Theorem 7.** There exist nk deterministic and $O(n \log k)$ randomized competitive algorithms in bipartite graphs for Caching in Matchings.

Proof. By Remark 5 and Remark 6, it suffices to show that A_m pays no more than n times compared to A_c. Whenever an edge (u, v) is requested, A_m has it cached if and only if A_c has it cached. Therefore when A_m has a miss, so does A_c. To accommodate for the edge, A_c ensures that u and v are both of degree $k-1$ before (u, v) is inserted. Now consider how many edge recolorings are required from A_m. Nodes u and v each have at least one free color. If both have some common free color c, we are done. Otherwise, u has c_1 free and v has $c_2 \neq c_1$ free. Let P_u and P_v be the (c_1, c_2) bi-colored paths that originate in u and v

respectively. P_u and P_v must be disjoint because the graph is bipartite and does not contain odd cycles. Flipping the colors ($c_1 \leftrightarrow c_2$) for each edge on either P_u or P_v enables A_m to insert and color (u, v). since P_u, P_v and (u, v) form a simple path in a graph with $2n$ nodes, by flipping the shorter bi-colored path, A_m colors at most n edges when inserting (u, v). ◀

3.2 Lower Bounds

Caching in Matchings is a generalization of caching, if we restrict the requests to edges of a single fixed node. Observe, therefore, that any c-competitive online algorithm for Caching in Matchings with $2 \leq k < n$ satisfies $c = \Omega(\log k)$. Moreover, if the algorithm is deterministic then $c \geq k$. The following lower bounds depend on n as well as k. These bounds hold for the non-trivial case of $2 \leq k < n$, in bipartite graphs, and therefore also hold for general graphs. Theorem 8 is proven later in this section, the proof of Theorem 9 is deferred to Appendix 6.1.

▶ **Theorem 8.** Any *deterministic* Caching in Matchings algorithm is $\Omega(\frac{n}{k})$ competitive.

▶ **Theorem 9.** *Any* Caching in Matchings algorithm is $\Omega(\log \frac{n}{k^2 \log k} \cdot \log k)$ competitive.

▶ **Corollary 10.** *Any* online algorithm for Caching in Matchings with $2 \leq k < n$ is $\Omega(\log n)$ competitive. Moreover, if $k = O(n^{1/2-\epsilon})$ for some $\epsilon > 0$, we get that any online algorithm for Caching in Matchings is $\Omega(\epsilon \cdot \log n \cdot \log k)$ competitive. If the algorithm is deterministic then the competitive ratio is $\Omega(\max\{\frac{n}{k}, k\})$.

Proof. The deterministic claim follows from the initial observation and Theorem 8. In the general case (randomized), we get $\Omega(\log n)$ from the maximum between the observation (when $k \geq n^{1/3}$) and Theorem 9 (when $k < n^{1/3}$). The $\Omega(\epsilon \cdot \log n \cdot \log k)$ bound follows from Theorem 9: If $k \leq c \cdot n^{1/2-\epsilon}$ for some constant c then $\log \frac{n}{k^2 \log k} > \log \frac{n^{2\epsilon}}{c^2 \log(cn)} = 2\epsilon \log n - 2 \log c - \log \log(cn) = \Omega(\epsilon \cdot \log n)$. ◀

We prove the lower bounds in a setting that is closer to dynamic graph coloring. Specifically, we define Problem 3 below, where we control which edges must be cached both by the algorithm and the optimum. We prove (Lemma 11 below, proven in Appendix 6.1) that lower bounds for algorithms for Problem 3 imply lower bounds for Caching in Matchings, and then study lower bounds for Problem 3.

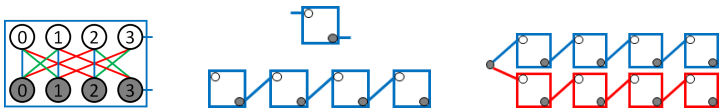
▶ **Problem 3.** Given a graph with n vertices, we get a sequence of actions that define a subset of edges at any time. Each action either adds a missing edge or deletes an existing edge. We are guaranteed that at any point in time the graph induced by existing edges, denote it (or the set of edges) by G, has a proper k-edge-coloring. An algorithm, online or OPT, must maintain k matchings, denote their union by M, such that G is a subgraph of M. For every edge that is added to a matching, the algorithm pays 1.

Note that Problem 3 is similar but not equivalent to dynamic edge coloring. On one hand a dynamic edge coloring algorithm that recolors $O(C)$ edges per update is not necessarily $O(C)$ competitive for Problem 3. The reason for this is that we allow M to contain G. By maintaining an edge in M we can avoid paying for it when it is inserted again. For example, in the proof of Theorem 8 an algorithm may do $O(1)$ worst-case recolorings per step, but its competitive ratio is $\Omega(\frac{n}{k})$ since OPT stores in M extra edges that this online algorithm keeps paying for. On the other hand, an algorithm that is $O(C)$ competitive for Problem 3 does not give a dynamic edge coloring algorithm that recolors $O(C)$ edges per update, even amortized, because it could be that both OPT and the algorithm pay a lot per edge update on some sequence, and while the ratio is $O(C)$, the absolute cost is large.

▶ **Lemma 11.** A lower bound of C on the competitive ratio of an online algorithm for Problem 3 implies a lower bound of C on the competitive ratio of an online algorithm for Caching in Matchings.

We now focus on deriving lower bounds for Problem 3. We define a *road* gadget (Definition 13) which is a connected component with a large diameter, that is also very restricted in the way it can be colored. A road is constructed from *brick* sub-gadgets (Definition 12), each of size $\Theta(k)$ nodes and $\Theta(k^2)$ edges. By connecting $r \geq 2$ roads together we get the *r-road* gadget, whose structure enforces those roads to be colored in a distinct and different way.

▶ **Definition 12** (Brick)**.** A *colorless-brick* is a union of k perfect matchings in a bipartite graph with the following structure. Each side has w nodes where w is the unique power of 2 that satisfies $\frac{w}{2} < k \leq w$. Number the nodes on each side $0, \ldots, w-1$, and number the colors $0, \ldots, k-1$. The matching of color c matches node i with node $i \oplus c$ where $\oplus$ is the bitwise exclusive-or. See Figure 3a for an example. Note that every color c indeed defines a matching that is in fact a permutation of order 2, and that $v \oplus c \neq v \oplus c'$ for any two colors $c \neq c'$ so the matchings are all disjoint. When we remove an edge from a colorless-brick, we get a *brick* whose color is associated with the color of the non-perfect matching. The two nodes of degree $k-1$ are the *endpoints* of the brick.



(a) Brick. **(b)** Road. **(c)** r-Road for $r = 2$.

Figure 3 Visualization of a *brick* (Definitions 12), a *road* and an *r-road* (Definition 13). (a) A brick for $k = 3$ ($w = 4$). The number of each node is written, and the colors are as follows: blue (0), green (1), red (2). Thus, for example, $1 \oplus red = 3$. We removed a single edge, blue $(3, 3)$, thus the brick is blue. (b) A schematic way to draw a brick (top) and a road of length 4 (bottom) which is a chain of bricks connected to each other by their endpoints. The color of a road is well-defined by the color of its bricks. (c) An r-road for $r = 2$, of length 4. The node that connects to both roads is its hub. Nodes are colored by gray and white according to their side in the bipartite graph.

▶ **Definition 13** (Road, r-road)**.** A *road* of length $d \geq 1$ is an edge colored graph obtained by connecting a sequence of d bricks. Each brick is connected by an edge to the next brick in the sequence. The edge connecting two consecutive bricks is adjacent to an endpoint of each brick. Note that the color of two connected bricks must be the same since they must agree on their free color which is the color of the edge which connects them. Therefore, we define the *color of a road* to be the color of its bricks. A road has two *ends*, which are the endpoints of its first and last bricks. We also refer to r $(2 \leq r \leq k)$ roads of the same length d that are all connected to a single shared node as an *r-road* of length d. The shared node is its *hub*. The edges of the hub all have different colors, therefore all the roads of an r-road have different colors. See Figure 3 for examples.

▶ **Lemma 14.** Given a brick B of color c_1, and a new color $c_2 \neq c_1$, it is always possible to recolor 3 edges to change the color of B to c_2.

Note that when we recolor B, it no longer satisfies the $\oplus$-property of Definition 12, but for convenience we still consider it as a brick. This would not affect our arguments below (by more than a constant factor) since we will make sure to always return to the original coloring (undo) before recoloring again.

Proof. Denote by u one endpoint of the brick. By definition of the matching scheme, the other endpoint is $u \oplus c_1$, and when we restrict the graph to edges of colors c_1 and c_2, we find that the path between u and $u \oplus c_1$ is of length 3: $u \oplus c_1 \to u \oplus c_1 \oplus c_2 \to u \oplus c_2 \to u$. Therefore, it suffices to flip the color of these three edges from c_2 to c_1 and vice versa. ◀

In the remainder of this section we prove the deterministic lower bound, and a simpler but weaker version of the randomized lower bound. The more involved randomized lower bound is proven in Appendix 6.1, using the same gadgets.

Proof of Theorem 8. We prove the lower bound for Problem 3. Then the theorem follows by Lemma 11. We present an adversarial construction against a given algorithm ALG.

We begin by setting aside one special node u to serve as a 2-road hub, and divide the rest of the vertices into bricks. We construct from these bricks the longest possible road, of length $N = \Theta(\frac{n}{k})$. We number the bricks in order, from 1 to N, and denote the edge between bricks i and $i+1$ by $(i, i+1)$. Let $L = \lfloor \frac{N}{3} \rfloor$. Initially we insert all the edges of all the bricks, without the edges connecting the bricks. These edges are never deleted. Our sequence has as many steps as we like as follows, based on the state of ALG, see also Figure 4:

1. Simple step: If there exist consecutive bricks i and $i+1$ of different colors, we insert the edge $(i, i+1)$. This forces ALG to recolor at least one of the bricks and pay $\Omega(1)$ (it could pay more if it recolors more bricks or does other actions). We then delete $(i, i+1)$.
2. Split step: Otherwise, all the bricks of ALG have the same color. We insert all the edges between bricks 1 through to L, and between $N+1-L$ through to N. We also insert edges from u to bricks 1 and $N+1-L$. These insertions construct a 2-road with u as its hub, that guarantees different colors for bricks 1 to L compared to bricks $N+1-L$ to N. ALG must recolor at least $L = \Omega(N)$ bricks. We then delete the edges that we inserted.

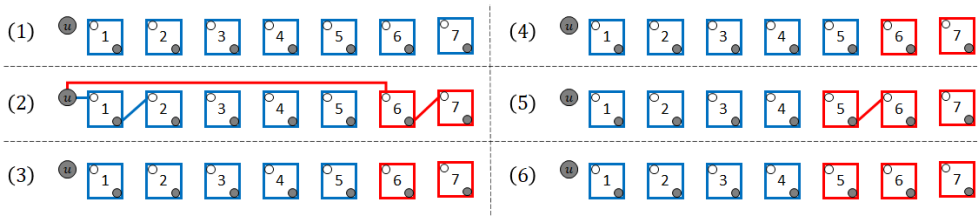


Figure 4 Visualization for the proof of Theorem 8, for $N = 7$, $L = \lfloor \frac{N}{3} \rfloor = 2$. (1)-(3) A *split step* temporarily creates a 2-road of length L out of the first and last bricks, with u as the hub, to guarantee consecutive bricks with different colors. (4)-(6) A *simple step* finds two consecutive bricks with different color (here: 5 and 6), and inserts temporarily the edge between them to enforce recoloring one of them.

In simple terms, we maintain a hole in the road which is where the color of the bricks changes (there could be multiple holes). In every step we request this hole, and once the hole disappears, the split action re-introduces a hole back near the middle of the road.[5]

Now let us analyze the costs. If the sequence contains m simple steps and s split steps, then $cost(ALG) = \Omega(m + s \cdot N)$. For OPT, we define a family of strategies B_i for $L < i < N+1-L$, to bound its cost. We define B_i to store all the edges of all the bricks, all the edges connecting them, and the edge that connects u to the first brick, except for

[5] This idea is similar to the way one can prove a deterministic lower bound for k-server [15], by always requesting a server-less location. The analogy stops here, since in our case sometimes there is no hole.

the edges that connect brick i to its neighbours, paying an initial $O(N \cdot k^2)$ cost. B_i colors all the bricks from 1 to i in one color, and all the bricks from $i+1$ to N in another color. Whenever a simple step happens in $(i-1, i)$ or in $(i, i+1)$, B_i simply recolors brick i to have the same color of the neighbour it connects to, and also inserts the connecting edge. Simple steps at other locations do not affect B_i. When a split step happens, B_i pays exactly 2: It inserts the edge that connects u to brick $N+1-L$ instead of the edge $(N-L, N+1-L)$. When the split step ends, it undoes the change, re-inserting $(N-L, N+1-L)$ instead of the edge of u. Since $L < i < N+1-L$, B_i does not have to recolor any other edge.

Thus, if we denote by m_i the number of simple steps that insert the edge $(i, i+1)$, then $cost(OPT) \leq cost(B_i) = O(m_i + m_{i-1} + s + N \cdot k^2)$. Now: $\frac{N}{3} \cdot cost(OPT) \leq (N - 2L) \cdot cost(OPT) \leq \sum_{i=L+1}^{N-L} cost(B_i) = O(m + N \cdot s + N^2 k^2)$. Note that $N^2 k^2 = O(n^2)$. Thus by extending the sequence such that $m + s = \Omega(n^2)$, we get: $\frac{N}{3} \cdot cost(OPT) = O(m + N \cdot s) = O(cost(ALG))$. Therefore, $\frac{cost(ALG)}{cost(OPT)} = \Omega(N)$. Recall that $N = \Theta(\frac{n}{k})$, and the claim follows. ◀

When proving the deterministic lower bound (Theorem 8) we heavily relied on determinism to know where to find neighbouring bricks of different colors. In the randomized case, we may not know where colors mismatch. Instead, we use a different and weaker construction for the randomized case. Relying on Yao's principle [32] (see also [15]), we define a distribution over sequences that is hard for any algorithm.

The following Lemma 15 is a weaker but also simpler version of Theorem 9 (proven in Appendix 6.1). Proving this lemma demonstrates our main technique.

▶ **Lemma 15.** *Any* Caching in Matchings algorithm with $2 \leq k < n$ matchings is $\Omega(\log \frac{n}{k})$ competitive.

Proof. We prove the lower bound for Problem 3. Then the theorem follows by Lemma 11. For convenience, since any action of fetching or recoloring can be lazily postponed to the next request for adding an edge to G, we assume that the algorithm does nothing else when an edge is removed.

Given a fixed k, we divide the nodes to r-roads of length d_0. We aim to have 2^h r-roads in total, for h as large as possible. We can bound h from below by noting that a brick requires at most $4k$ nodes, a road of length d_0 requires at most $4k \cdot d_0$ nodes (d_0 bricks), and an r-road requires at most $4k \cdot d_0 \cdot r + 1$ nodes (r roads and a hub). So we get that $h \geq \lfloor \log \frac{n}{4k \cdot d_0 \cdot r + 1} \rfloor$. Simplified, we get $h = \Omega(\log \frac{n}{k \cdot d_0 \cdot r})$. Eventually we will choose $r = 2$ and $d_0 = 1$ to get $h = \Omega(\log \frac{n}{k})$.

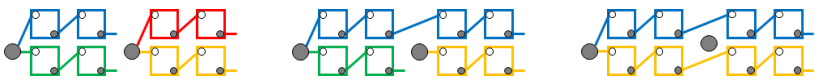
We construct the distribution over request sequences in phases. A phase begins with 2^h r-roads. Then, during each phase we have h rounds, numbered from $i = 1$ to $i = h$. In each round we pair the r-roads, and merge each pair into a single twice-longer r-road. Note that in round i there are 2^{h-i} pairs of r-roads whose length is $d_i = d_0 \cdot 2^{i-1}$. Once we get to a final single r-road of length d_h, we delete the edges that were used to connect the roads of length d_0, and insert back hub edges to re-form the initial r-roads of length d_0. Then a new phase begins.

We explain later exactly how a pair of r-roads of length d_i are merged, for now just assume that OPT pays $O(r)$ and that any algorithm pays $\Omega(r \cdot d_i)$ in expectation for such merge, and let us analyze the competitive ratio. When a phase begins, OPT can choose a consistent color for each road of the r-roads such that no further recoloring is necessary during this phase, at a cost of $O(2^h \cdot r \cdot d_0)$ by recoloring $O(1)$ edges in each brick (according to Lemma 14) and each edge that connects to a brick to the desired color. Then, throughout the rounds it pays additional $\sum_{i=1}^{h} 2^{h-i} \cdot O(r) = O(2^h \cdot r)$. Finally, when a phase ends it pays $O(2^h \cdot r)$

more to re-attach hubs when re-creating the initial r-roads of length d_0, and $O(2^h \cdot r \cdot d_0)$ more to undo any recoloring made when the phase begun.[6] Overall, OPT pays per phase $O(2^h \cdot r \cdot d_0)$. In contrast, ALG pays at least $\sum_{i=1}^{h} 2^{h-i} \cdot \Omega(r \cdot 2^{i-1} \cdot d_0) = \Omega(h \cdot 2^h \cdot r \cdot d_0)$.

Let t be the number of phases. The one-time initialization cost is $c_0 = \Theta(k^2 \cdot (2^h \cdot r \cdot d_0))$ for inserting $\Theta(k^2)$ edges per brick. Therefore, the competitive ratio that we get is $\frac{\mathbb{E}[cost(ALG)]}{cost(OPT)} \geq \frac{c_0 + t \cdot \Omega(h \cdot 2^h \cdot r \cdot d_0)}{c_0 + t \cdot O(2^h \cdot r \cdot d_0)}$. For $t = \Omega(k^2)$ we can neglect c_0 and get that $\frac{\mathbb{E}[cost(ALG)]}{cost(OPT)} = \Omega(h)$. Recall that $h = \Omega(\log \frac{n}{k \cdot r \cdot d_0})$, so we choose $r = 2$ and $d_0 = 1$ to maximize the competitive ratio and get $\Omega(\log \frac{n}{k})$, as claimed.

It remains to explain how we merge a pair of r-roads of length d, X and Y, see Figure 5 for a visual example for $r = 2$ and $k = 4$. We have r iterations, where iteration i cuts the ith road of Y away from the hub, and extends a uniformly random not yet extended road of X. OPT pays at most r for the newly introduced r edges because it can refrain from recoloring roads. As for ALG, observe that it must recolor a road if the colors of the extended road and its extension do not match. For the first two roads that we combine (one from each r-road), there is a probability of at most $\frac{1}{r}$ for the colors to agree (the probability is maximized if X and Y use the same colors for their roads, out of the k possible colors). More generally, in the ith iteration there is a probability of at most $\frac{1}{r+1-i}$ for the colors to agree, maximized if the remaining roads of Y share their colors with the not yet extended roads of X. Thus ALG recolors in expectation at least $\sum_{i=1}^{r} (1 - \frac{1}{r+1-i}) = r - H_r$ roads throughout the process, where H_r is the rth harmonic number. For $r \geq 2$, this amounts to a cost of $\Omega(r \cdot d)$ recolorings. ◀



(a) Initial state. **(b)** One road extended. **(c)** Merge complete.

Figure 5 Visualization of merging a pair of 2-roads of length $d = 2$ to a single twice longer 2-road. In this example $k \geq 4$. Initially the 2-roads are disjoint. Then, we cut a road from the right 2-road and extend another road in the left 2-road. If necessary, the algorithm recolors one or more of the roads. Then we do the same for the remaining road. In the end, the hub of the cut-down 2-road is a node of degree 0.

3.3 Upper Bounds with Resource Augmentation

In this section we study upper bounds with resource augmentation. That is, we assume that the optimum still has k matchings, but our algorithm has more. Interestingly, it dramatically improves the competitive ratios, in both deterministic and randomized settings.

Recall our general approach and notations: our caching in matchings algorithm A_m implements a component of an edge-coloring algorithm, over a component of a connection caching algorithm which we denote by A_c. By Remark 6, we can divide our attention between connection caching and dynamic edge-coloring. Concretely, given $h \geq 1$ extra matchings, we maintain connection caching with $k' \equiv k + h_1$ connections per node, and maintain edge-coloring of a graph of maximum degree k', with $k' + h_2$ colors, such that $h_1 + h_2 = h$. We choose $h_2 \geq 1$ because a single extra color yields a dramatic improvement. We use either $h_2 = \lceil \frac{h}{2} \rceil$ or $h_2 = h$, Corollary 25 summarizes our choices.

[6] It is necessary to revert to the exact initial coloring before the next phase because Lemma 14 for recoloring bricks requires a very specific coloring scheme.

We begin by listing three important facts about caching and connection caching algorithms, which we use as A_c. Concretely, Lemma 16 details a deterministic algorithm for Connection Caching, and Lemma 17 together with Theorem 18 yield a randomized algorithm for Connection Caching in Corollary 19.

▶ **Lemma 16** (Corollary 8 of [20]). There is a deterministic Connection Caching algorithm with cache of size r, that is $\frac{2r}{r-k+1}$-competitive against the optimum with cache of size $k \leq r$.

▶ **Lemma 17** (Section 2.2 of [33]). Let r be the cache size of the randomized caching algorithm *MARK* [24], and let k be the cache size of the optimum. Then *MARK* is: $O(\log r)$-competitive if $r = k$; $O(\log \frac{r}{r-k})$-competitive if $\frac{e-1}{e}r < k < r$; and, 2-competitive if $k \leq \frac{e-1}{e}r$.

▶ **Theorem 18** (Theorem 7 of [20]). Let A be a $c(r,k)$-competitive caching algorithm, with additive term δ, where r and k are the cache sizes of the algorithm and the optimum, respectively. Then there is a $2 \cdot c(r,k)$-competitive algorithm for Connection Caching, with additive term $|V| \cdot \delta$ where $|V|$ is the number of nodes in the graph.

The explicit reduction and proof of Theorem 18 can be found in [30].

▶ **Corollary 19.** There exists a randomized connection caching algorithm, with cache size r compared to k of the optimum, that is $O(\log r)$-competitive if $r = k$; $O(\log \frac{r}{r-k})$-competitive if $\frac{e-1}{e}r < k < r$; and, 4-competitive if $k \leq \frac{e-1}{e}r$.

Next, we list several results for dynamic edge coloring.

▶ **Lemma 20** (Greedy, Folklore). Let G be a dynamic graph with the guarantee that its maximum degree is at most k at any time. Then we can maintain for it a $2k-1$ edge-coloring without needing to recolor any edge.

Proof. When (u,v) is inserted, both u and v are of degree at most $k-1$, thus each has at least k free colors, and they must have at least one common free color that we can use. ◀

▶ **Lemma 21.** Let G be a dynamic *bipartite* graph with the guarantee that its maximum degree is at most k at any time, and let $h \geq 1$. We can maintain a deterministic $(k+h)$-edge-coloring of G in amortized $O(\sqrt{nk/h})$ recolorings per insertion.

The proof of Lemma 21 is deferred to Appendix 6.2. It is like the proof of Theorem 7 but we only have few edges from each extra color, such that we can recolor bi-chromatic paths quickly. Periodically, we recolor the graph using only k colors and amortize this work over several operations.

The following results are particularly useful when k is small. The high probability in Theorem 23 below is for bounding the running time, not for getting a proper coloring.

▶ **Theorem 22.** Let $G' \equiv G \cup \{e\}$ be a graph with maximum degree k, such that G is $(k+1)$-edge-colored, and e is uncolored. Then there is a $(k+1)$-edge-coloring of G' which recolors only N edges in G where $N = O(k^7 \log n)$ by [19] (Theorem 3), or $N = (k+1)^6 \log^2 n$ by [10] (Corollary 6.4).

▶ **Theorem 23** (Theorem 6 of [19]). Let G be a dynamic graph such that its maximum degree never exceeds k. Then there exists a fully-dynamic algorithm that maintains a $\lceil (1+\epsilon)k \rceil$-edge-coloring with $O(\epsilon^{-6} \log^6 k \log^9 n)$ worst-case update time with high probability.

The paper [13] gives an efficient randomized edge-coloring for sufficiently large k.

▶ **Theorem 24** ([13]). Let G be a dynamic graph such that its maximum degree never exceeds k. If $k \geq (100\alpha^4 \log n)^{30\alpha \log \alpha}$, there is a fully-dynamic algorithm maintaining a $(1 + O(\frac{1}{\alpha}))k$-edge-coloring with $O(\alpha^4)$ edge recolorings in expectation per update.

Finally, we combine the various results of connection caching and edge coloring to derive the following competitive algorithms.

▶ **Corollary 25.** Given resource augmentation of extra $1 \leq h = O(k)$ matchings, that is $k + h$ for the algorithm versus k for OPT, the following competitive algorithms for caching in matchings exist:

1. $O(n^{1/2}(k/h)^{3/2})$ deterministic and $O(n^{1/2}(k/h)^{1/2} \log \frac{2k+h}{h})$ randomized competitive algorithms in bipartite graphs.
2. $O\left(\frac{k^6 \log n}{h} \min(k, \log n)\right)$ deterministic and $O\left(\left(\frac{k \log k}{h}\right)^6 \log \frac{2k+h}{h} \log^9 n\right)$ randomized competitive algorithms in general graphs. The deterministic algorithm is inefficient.
3. If $h = k - 1$ we can remove the dependence on n, yielding k deterministic and $O(\log k)$ randomized competitive algorithms in general graphs.
4. $O(\alpha^4 \log k)$ randomized competitive algorithm, where $\alpha = O(\frac{k}{h})$ and provided that $k \geq (100\alpha^4 \log n)^{30\alpha \log \alpha}$.

Proof. We use the augmentation to have extra h_2 colors for the coloring component, where $h_2 = h$ for Part-(3), and $h_2 = \lceil \frac{h}{2} \rceil$ for the rest. Part-(1) is by Lemma 21 with Lemma 16 and Corollary 19. Part-(2) is by Theorem 22 with Lemma 16, and by Theorem 23 with Corollary 19. The deterministic algorithm is inefficient because Theorem 22 only proves the existence of the stated recoloring by probabilistic arguments. Regarding the randomized part, Theorem 23 guarantees recoloring that is cheap with high probability. We can choose the constants such that the failure probability is $\leq \frac{1}{n^2}$, and fully recolor the graph if the cheap method fails. The expected number of recolorings is negligibly affected, and proper edge coloring is guaranteed. Notice that we set $\epsilon = \frac{h}{k}$. Part-(3) is by Lemma 20. Part-(4) is by Theorem 24 and Corollary 19. ◀

▶ Remark 26. Choosing $h_2 = \lceil \frac{h}{2} \rceil$ in Corollary 25 divides the augmentation into equal halves and is good enough if we do not optimize the constants, since essentially both caching and coloring components "benefit" from $\Theta(h)$ augmentation.

In Part-(4) it is simplest to think of α as a constant, in which case the requirement $k \geq f(n, \alpha)$ for the function f given in the statement, requires $k = \Omega(poly(\log n))$. However, α can also depend on k, as long as there are values of k that satisfy $k \geq f(n, \alpha(k))$. Observe that because $(\log n)^{\log n / \log \log n} = n$, for $k \leq n$ it must be that $\alpha = O(\frac{\log n}{\log \log n})$ or else the inequality cannot be satisfied. Then in particular $\alpha = o(\log n)$, and $k^{1/\tilde{\Theta}(\alpha)} \geq \log n$ (for the appropriate constants) implies $k \geq f(n, \alpha(k))$. Crudely simplified for the sake of a clean example, if $k^{\frac{1}{\alpha}} \geq \log n$ we could choose $\alpha = \frac{\log k}{2 \log \log k}$, and still have a non-empty range of applicable k values.

Finally, we improve the competitive ratio further with a larger resource augmentation.

▶ **Theorem 27.** Given $k+h$ matchings to the algorithm compared to only k matchings to the optimum, for $h \geq k - 1$, there is a deterministic algorithm that is $2(1 + \frac{k-1}{\lfloor \frac{h+3-k}{2} \rfloor})$-competitive for Caching in Matchings. In particular, with $h = (1 + \alpha)k$ extra matchings we get a competitive ratio of $O(1 + \frac{1}{\alpha})$.

The proof is in Appendix 6.2. It follows from Lemma 20 and Lemma 16.

▶ **Corollary 28.** Consider the Caching in Matchings problem where the optimum is given k matchings. There is a 6-competitive algorithm that uses $3k - 3$ matchings ($h = 2k - 3$).

A note on the running times. This work focuses on competitive analysis and therefore we do not attempt to optimize polynomial running time. We note that the algorithms in Corollary 25(1),(3) take $O(n)$ time: maintaining connection caching takes $O(k)$ time, and the coloring algorithms in Lemma 20 and Lemma 21 naively take time of $O(\Delta)$ to find a color and $O(\rho)$ to recolor a path of length ρ. The randomized algorithms of Corollary 25(2),(4) also run in polynomial time per update.

4 Related Work

Caching Problems. Caching problems have been studied in many variants and cost models. *Connection caching* is the caching variant closest to our problem. We presented it in a centralized setting, Cohen et al. [20] introduced it in a distributed setting. Albers [3] studies generalized connection caching. Bienkowski et al. [14] study connection caching in a cost model that is similar to that of caching with rejections [23]. Another related variant is *restricted caching* [16, 25] where not every page can be put into every cache slot. Buchbinder et al. [16] study the case where each page p has a subset of cache slots in which it can be cached. In our problem we also have a restriction of similar flavour, implied by the separation into matchings. We note that the cost model in [16, 25] only counts cache-misses, while we also pay for rearranging the cache.

Coloring Problems. As mentioned in Section 3, an efficient dynamic edge coloring that uses a small number of colors can be useful for competitive analysis. Subsection 3.3 covers results which we use for our advantage. The coloring literature studies the tradeoff between the number of colors, amount of recoloring (sometimes called *recourse*), and the running time of the algorithms. Some algorithms require a bound Δ on the maximum degree of the dynamic graph, while others are adaptive with respect to the maximum degree in their running time or recoloring. Literature on vertex coloring also exists, but reducing edge coloring to vertex coloring by coloring the line-graph is too wasteful in the number of colors, whether this number is parameterized by Δ, or by the arboricity of the graph as in [27]. Works on maintaining dynamically an *implicit coloring* [18, 27] cannot apply to our case because the matchings form an explicit coloring. Azar et al. [5] study dynamic vertex coloring in the context of competitive analysis.

El-Hayek et al. [22] are motivated by the same architecture as us. They solve a problem of dynamically maximizing the size of a k-edge-colorable subgraph of a dynamic graph.

Linear Programming and Convex Body Chasing in L_1. The aforementioned caching problems, like many other combinatorial problems, can be formulated as a linear program [17]. This line of research led to the development of competitive algorithms for weighted and generalized caching [1, 2, 6, 7]. A recent result of Bhattacharya et al. [11] uses linear programming with packing and covering constraints to formulate and frame the problem as convex body-chasing in L_1. They give a fractional algorithm that requires a slight resource augmentation, along with some rounding schemes to get randomized algorithms for specific problems. Our problem can be thought of as another special case of the problem considered by [11], see the appendix in the extended version [30] for this formulation and further details.

5 Conclusions and Future Work

In this paper we studied the online Caching in Matchings problem, in which we receive requests for edges in a graph and need to maintain a cache of the edges which is a union of k matchings. The problem abstracts some hardware architecture in which a datacenter is enhanced with reconfigurable optical links. Interestingly, we proved that the Caching in Matchings problem is inherently harder than the similar-looking Connection Caching problem and other caching problems. Specifically, its competitive ratio depends not only on the number of matchings k ("cache size") but also on the number of nodes in the graph. Our randomized lower bound rules out an $O(\log n)$ competitive algorithm, and the best competitive ratio we can hope for is $O(poly(\log n))$. Our lower bound for deterministic algorithms is linear in n.

We derived our algorithms by running a coloring algorithm that maintains a coloring of the cache of a connection caching algorithm. This approach is simple to describe and analyze, but inherently multiplies the competitive ratios of the two algorithms. It is natural to ask whether a "direct" algorithm for Caching in Matchings exists, and if so does it improve the competitive ratio?

Regarding resource augmentation of $h \geq 1$ extra matchings, we see that there are two interesting "discontinuities". First, immediately for $h = 1$ the competitive ratio drops to $poly(k, \log n)$, in particular "breaking" the deterministic lower bound. Second, there seems to be a point in which the competitive ratio becomes independent of n. It clearly happens for $h = k - 1$, and even sooner if k is large enough (revisit Corollary 25). These "discontinuities" beg the following two questions. First, is there an $1 \leq h < k - 1$ and an algorithm that uses h extra matchings with a competitive ratio of $O(poly(\log n))$ for any $2 \leq k < n$? Differently phrased, can we achieve a competitive ratio that is $poly(\log k, \log n)$ instead of $poly(k, \log n)$? Second, is it possible to remove the dependence on n using less than $h = k-1$ extra matchings for any k, and if so how small can h be?

A natural generalization would be to study upper bounds for Caching in Matchings in general graphs. When $k = 1$ optimality is still trivial, and when $k = n$, by Vizing's theorem, we are also optimal since we can edge-color the full n-clique with n colors. In fact, for $n \geq 2$, $k = n - 1$ colors are sufficient if and only if n is even. In the non-trivial regime $2 \leq k < n$, there exists a naive deterministic $O(n^2k^2)$ competitive algorithm (the extended version [30] contains exlicit proofs). Resource augmentation of an extra matching ($k + 1$ colors) dramatically reduces the competitive ratio to $O(nk)$ (deterministic) and $O(n \log k)$ (randomized) by allowing us to update the coloring of the graph when a new edge is inserted according to a single step of the Misra-Gries algorithm [29],[7] or to $O(poly(k, \log n))$ as in Corollary 25(2). It is an interesting question whether the problem without augmentation is indeed that much harder in general graphs. General graphs also provide additional difficulties, such as the fact that finding minimal edge coloring for $k \geq 3$ is generally NP-complete [28].

References

1 Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. An $O(\log k)$-competitive algorithm for generalized caching. In *SODA*, pages 1681–1689. SIAM, 2012.

2 Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. An $O(\log k)$-competitive algorithm for generalized caching. *ACM Trans. Algorithms*, 15(1), 2018.

[7] The Misra-Gries algorithm edge-colors an uncolored graph with m edges and n nodes in m iterations. In each iteration it colors an edge and fixes the colors of previously colored edges, by recoloring $O(n)$ of them in $O(n)$ time. In our case, the graph is always fully colored up to the newly requested edge, so we get the competitive ratio of a connection caching algorithm, multiplied by $O(n)$.

3 Susanne Albers. Generalized connection caching. In *ACM SPAA*, pages 70–78, 2000.

4 Chen Avin, Chen Griner, Iosif Salem, and Stefan Schmid. An online matching model for self-adjusting ToR-to-ToR networks, 2020. `arXiv:2006.11148`.

5 Yossi Azar, Chay Machluf, Boaz Patt-Shamir, and Noam Touitou. Competitive vertex recoloring. *Algorithmica*, 85:2001–2027, 2023.

6 Nikhil Bansal, Niv Buchbinder, and Joseph (Seffi) Naor. Randomized competitive algorithms for generalized caching. In *STOC*, pages 235–244, 2008.

7 Nikhil Bansal, Niv Buchbinder, and Joseph (Seffi) Naor. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59(4), 2012.

8 Luis Barba, Jean Cardinal, Matias Korman, Stefan Langerman, André Renssen, Marcel Roeloffzen, and Sander Verdonschot. Dynamic graph coloring. *Algorithmica*, 81(4):1319–1341, 2019.

9 Leonid Barenboim and Tzalik Maimon. Fully dynamic graph algorithms inspired by distributed computing: Deterministic maximal matching and edge coloring in sublinear update-time. *ACM J. Exp. Algorithmics*, 24:1–24, 2019.

10 Anton Bernshteyn. A fast distributed algorithm for $(\Delta+1)$-edge-coloring. *Journal of Combinatorial Theory, Series B*, 152:319–352, 2022.

11 S. Bhattacharya, N. Buchbinder, R. Levin, and T. Saranurak. Chasing positive bodies. In *FOCS*, pages 1694–1714, 2023.

12 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In *SODA*, pages 1–20, 2018.

13 Sayan Bhattacharya, Martín Costa, Nadav Panski, and Shay Solomon. Nibbling at long cycles: Dynamic (and static) edge coloring in optimal time. In *SODA*. SIAM, 2024.

14 Marcin Bienkowski, David Fuchssteiner, Jan Marcinkowski, and Stefan Schmid. Online dynamic b-matching: With applications to reconfigurable datacenter networks. *SIGMETRICS Perform. Eval. Rev.*, 48(3), 2021.

15 Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

16 Niv Buchbinder, Shahar Chen, and Joseph (Seffi) Naor. Competitive algorithms for restricted caching and matroid caching. In *ESA*, pages 209–221, 2014.

17 Niv Buchbinder and Joseph (Seffi) Naor. *The Design of Competitive Online Algorithms via a Primal-Dual Approach*. Now Foundations and Trends, 2009.

18 Aleksander B. G. Christiansen and Eva Rotenberg. Fully-Dynamic $\alpha + 2$ Arboricity Decompositions and Implicit Colouring. In *ICALP*, pages 42:1–42:20, 2022.

19 Aleksander Bjørn Grodt Christiansen. The power of multi-step vizing chains. In *STOC*, pages 1013–1026, 2023.

20 Edith Cohen, Haim Kaplan, and Uri Zwick. Connection caching under various models of communication. *ACM SPAA*, 2000.

21 Ran Duan, Haoqing He, and Tianyi Zhang. *Dynamic Edge Coloring with Improved Approximation*, pages 1937–1945. SIAM, 2019.

22 Antoine El-Hayek, Kathrin Hanauer, and Monika Henzinger. On b-matching and fully-dynamic maximum k-edge coloring, 2023. `arXiv:2310.01149`.

23 Leah Epstein, Csanád Imreh, Asaf Levin, and Judit Nagy-György. Online file caching with rejection penalties. *Algorithmica*, 71(2):279–306, 2015.

24 Amos Fiat, Richard M Karp, Michael Luby, Lyle A McGeoch, Daniel D Sleator, and Neal E Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

25 Amos Fiat, Manor Mendel, and Steven S. Seiden. Online companion caching. In *ESA*, pages 499–511, 2002.

26 Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel Kilper. ProjecToR: Agile reconfigurable data center interconnect. In *ACM SIGCOMM*, pages 216–229, 2016.

27 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Explicit and implicit dynamic coloring of graphs with bounded arboricity, 2020. `arXiv:2002.10142`.

28 Ian Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, 1981.

29 J. Misra and David Gries. A constructive proof of Vizing's theorem. *Information Processing Letters*, 41(3):131–133, 1992.

30 Yaniv Sadeh and Haim Kaplan. Caching connections in matchings, 2024. `arXiv:2310.14058`.

31 Shay Solomon and Nicole Wein. Improved dynamic graph coloring. *ACM Trans. Algorithms*, 16(3):1–24, 2020.

32 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *SFCS*, pages 222–227, 1977.

33 Neal Young. *Competitive paging and dual-guided algorithms for weighted caching and matching*. PhD thesis, Computer Science Dept., Princeton University, 1991.

6 Appendix: Deferred Proofs and Discussions

6.1 Caching in Matchings Lower Bounds (Proofs)

▶ **Lemma 11.** A lower bound of C on the competitive ratio of an online algorithm for Problem 3 implies a lower bound of C on the competitive ratio of an online algorithm for Caching in Matchings.

Proof. Let A_1 be a c-competitive algorithm for Caching in Matchings (Problem 1), with some additive term d. We show how to derive from it an algorithm A_3 that is c-competitive for Problem 3, which proves the claim. We will also use corresponding subscripts OPT_1 and OPT_3 for the optimum of each problem (with respect to a given sequence).

Given a sequence τ, Algorithm A_3 takes its decisions while processing τ by simulating A_1 on a sequence σ which is constructed as follows. We traverse τ in order, and whenever an edge is inserted, we add to σ a batch of requests which is a concatenation of $r = nk$ identical subsequences, each subsequence contains all the edges currently in G (in some arbitrary order). When an edge is deleted, we do nothing.

We now specify A_3 such that $cost(A_3(\tau))) \leq cost(A_1(\sigma))$ by having A_3 maintain its state such that it "jumps" between "check-points" in the state of A_1.

A_3 works as follows. When an edge is inserted by τ, A_3 feeds σ to A_1 until one of two things happens: either (1) the state of A_1 provides a proper coloring of G, or (2) it reaches the end of the batch that corresponds to the current edge inserted by τ. In case (1), A_3 changes its state by replaying the changes that A_1 made. Then by definition of this case, it ends up with a proper coloring of G. In case (2), we know that during the whole batch A_1 did not have a proper coloring of G, which means that in each of the r rounds it paid at least 1 for a missing edge, for a total of at least r. Rather than replaying the changes and ending up with an illegal state for A_3, we have a budget of r to completely change its state. A_3 uses half of the budget to completely empty its state and fetch all of G with some proper coloring (such coloring exists by definition of the problem). Indeed it has the budget, $|G| \leq |M| \leq \frac{nk}{2}$. The other half of its budget is used to copy the state from which A_1 continues to process σ, by flushing everything, and fetching into the cache the state of A_1. This ensures that the state of A_3 is once again identical to A_1. We showed $cost(A_3(\tau))) \leq cost(A_1(\sigma))$. It holds in the deterministic case, and also for the randomized case for every fixing of the random coins.

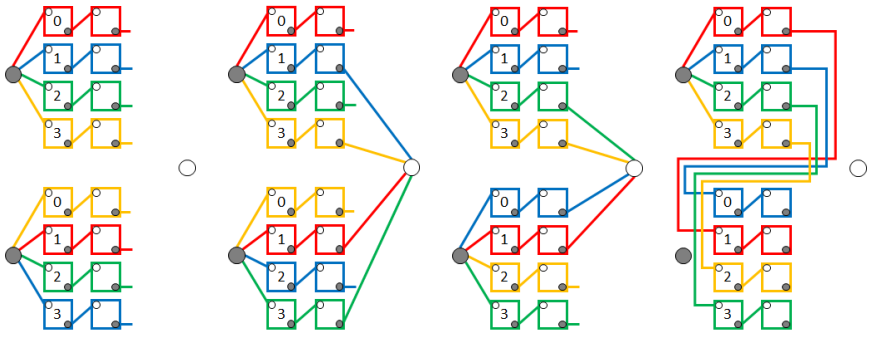
Now observe that $cost(OPT_1(\sigma)) \leq cost(OPT_3(\tau))$. Indeed, OPT_1 may simulate the behavior of OPT_3 by making changes to its own state at the beginning of each batch in σ. In conclusion, we get that $cost(A_3(\tau))) \leq cost(A_1(\sigma)) \leq c \cdot cost(OPT_1(\sigma)) + d \leq$

$c \cdot cost(OPT_3(\tau)) + d$ in the deterministic case, or similarly $\mathbb{E}[cost(A_3(\tau))] \leq \mathbb{E}[cost(A_1(\sigma))] \leq c \cdot cost(OPT_1(\sigma)) + d \leq c \cdot cost(OPT_3(\tau)) + d$ in the randomized case (recall that c and d were defined at the beginning of the proof). ◀

▶ **Theorem 9.** *Any* Caching in Matchings algorithm is $\Omega(\log \frac{n}{k^2 \log k} \cdot \log k)$ competitive.

Proof. This proof uses a similar high-level construction as the one used to prove Lemma 15, and the only difference is in the way we merge pairs of r-roads. Here we set $r = k$. Assume for now that when merging two k-roads of length d, OPT pays $O(k \log k)$ and any algorithm pays $\Omega(k \log k \cdot d)$ in expectation.

With this in mind, revisit the competitive analysis: when a phase begins OPT pays $O(2^h \cdot k \cdot d_0)$ to recolor bricks to their desired color, it then pays $\sum_{i=1}^{h} 2^{h-i} \cdot O(k \log k) = O(2^h \cdot k \log k)$ in the merging rounds, and finally it pays $O(2^h \cdot k \cdot d_0)$ to restore the original k-roads for the next phase. Overall, its cost is $O(2^h \cdot k \cdot (d_0 + \log k))$. In comparison, ALG pays for recoloring, in expectation, at least $\sum_{i=1}^{h} 2^{h-i} \cdot \Omega(k \log k \cdot 2^{i-1} \cdot d_0) = \Omega(h \cdot 2^h \cdot k \log k \cdot d_0)$. Assuming a sequence with $t = \Omega(k^2)$ phases, we get that the competitive ratio is $\frac{\mathbb{E}[cost(ALG)]}{cost(OPT)} = \Omega(\frac{h \cdot 2^h \cdot k \log k \cdot d_0}{2^h \cdot k \cdot (d_0 + \log k)}) = \Omega(\frac{h \cdot \log k \cdot d_0}{d_0 + \log k})$. To maximize the expression we balance and choose $d_0 = \lceil \log k \rceil$, getting $\Omega(h \log k)$. We determine h as before, except that the complicated merging technique requires a reusable extra node, so we have that $h \geq \lfloor \log \frac{n-1}{4k^2 \cdot d_0 + 1} \rfloor$. Simplified, and with $d_0 = \lceil \log k \rceil$, we get $h = \Omega(\log \frac{n}{k^2 \log k})$, therefore the competitive ratio is $\Omega(\log \frac{n}{k^2 \log k} \cdot \log k)$.



(a) Initial state. **(b)** Step 1. **(c)** Step 2. **(d)** Final merge.

Figure 6 Visualization of merging a pair of k-roads, for $k = 4$, of length $d = 2$ to a single twice longer k-road, by "negative information". (a) The roads are numbered from 0 to 3, with their number written inside their first brick. There are $\log k = 2$ steps. (b) In the first step we temporarily connect roads $\{1, 3\}$ (least significant bit 1) of the top k-road with either roads $\{0, 2\}$ or $\{1, 3\}$ of the bottom k-road, to a shared hub (the white node). In this example we connect $\{1, 3\}$, and as a results roads 2 and 3 of the bottom k-road were recolored. (c) In the second step we temporarily connect roads $\{2, 3\}$ (second bit is 1) of the top k-road with either roads $\{0, 1\}$ or $\{2, 3\}$ of the bottom k-road to a shared hub. In this example we connect $\{0, 1\}$, and as a result roads 0 and 2 of the bottom k-road were recolored. (d) Finally there is a round of "positive information" in which we simply extend each road on the top color-consistently with a road on the bottom. The consistency depends on the choices of the previous steps, and in this example it matches road x on the top with road $x \oplus 1$ on the bottom. The recoloring in this example is such that in the final extension no road is recolored.

Now we explain and analyze the merging of two k-roads, denote them by X and Y. See Figure 6 for a visual example with $k = 4$. For simplicity, let us start with k being an integer power of 2, say $k = 2^\ell$. We start with ℓ steps of "negative information" in which we reveal

roads that are of *different* colors, and do so by connecting the free end of these roads to a new shared hub, denote it by v. Concretely, number the roads of X from 0 to $k-1=2^\ell-1$ and denote by $X_{i,b}$ the roads of X whose ith bit is b. We define similarly the subsets of roads for Y. In round i, we connect to v the roads $X_{i,1}$ and Y_{i,b_i} where b_i is chosen uniformly at random between 0 and 1. Note that $|X_{i,1}|=|Y_{i,b_i}|=\frac{k}{2}$ so the degree of v is k (legal). When the round ends, we delete the edges of v. Finally, in the $\ell+1$ step we produce the longer k-road with "positive information" by cutting the roads of Y from their hub and extending the roads of X, according to the unique way which does not contradict the previous ℓ steps. This way exists: road $y\in Y$ extends the road x whose binary representation is $x=y\oplus B$ where the bits of B are b_i and $\oplus$ is the bitwise exclusive-or operation.

Let us analyze the costs of OPT and ALG. Since OPT knows the correct colors in advance, it can pay at most 1 per edge that is inserted. We insert k edges per step (even if most of them are later deleted), in a total of $\ell+1$ steps. This totals to $O(k\log k)$. We argue that ALG recolors in expectation $\Omega(k)$ roads per each of the first ℓ steps. To simplify the analysis, assume that ALG recolors after learning b_i rather than being introduced online each edge of v one by one (which can only hurt ALG). Then indeed every road in $X_{i,1}$ has probability $\frac{1}{2}$ to be in a conflict of color with a road of Y_{i,b_i}, so by the linearity of expectation, we get at least $\frac{k}{2}=\Omega(k)$ road recolorings (if ALG is "reasonable", it recolors $O(k)$ roads per step, so allowing it to be semi-offline did not lose more than a constant factor). Observe that our bound for round i is not affected by previous rounds. So we conclude that ALG pays $\Omega(\ell\cdot k\cdot d)=\Omega(k\log k\cdot d)$ for recoloring in expectation.

The case of k not being a power of 2 is similar. Each road is still assigned a number, and we regard its binary representation with $\lceil\log k\rceil$ bits, but only make $\ell=\lfloor\log k\rfloor$ rounds. Note that now $X_{i,1}$ is not necessarily of size $\frac{k}{2}$, but rather might be smaller. The bias is always in favor of 0 because of how counting works, and it is such that $|X_{i,1}|\geq\frac{k-2^i}{2}$ ($i=0$ is the least significant bit). So we can still choose Y_{i,b_i} with b_i uniformly random, there is no problem to connect all the roads of $X_{i,1}$ and Y_{i,b_i} to their shared hub. Also, each road in $X_{i,1}$ still has a color conflict with probability $\frac{1}{2}$. The only thing that changes is that the expectation of road recolorings is not $\frac{k}{2}$ per round, but rather $|X_{i,1}|$ in round i. This yields at least $\sum_{i=0}^{\ell-1}|X_{i,1}|\geq\ell\cdot\frac{k}{2}-\frac{1}{2}\sum_{i=0}^{\ell-1}2^i>\ell\cdot\frac{k}{2}-\frac{2^\ell}{2}>\frac{(\ell-1)k}{2}$ road recolorings in expectation for ALG, which is still $\Omega(k\log k)$ in total. The analysis of OPT is unchanged, and its total cost is $O(k\log k)$ in total per merging a pair of k-roads (of any length). ◀

▶ Remark 29. A few notes on the proofs of Lemma 15 and Theorem 9:

1. The random choices of the adversary can be boiled-down to the random order of extending roads (in Lemma 15) and the bits b_i (in Theorem 9). The 2-roads and k-roads themselves are chosen once, and even the pairings of each merging round may be fixed.
2. For clarity, we presented it as if we need 2^h different hubs, one per r-road. In fact, we only need $h+1$ hubs if we reuse them: h of them to maintain an r-road for each unique length, and another one for the length in which we currently merge a pair of r-roads. This saving is negligible compared to the number of nodes used to compose the roads.

6.2 Caching in Matchings With Resource Augmentation (Proofs)

In this section we restate and prove the claims from Section 3.1 that we did not prove there.

▶ **Lemma 21.** Let G be a dynamic *bipartite* graph with the guarantee that its maximum degree is at most k at any time, and let $h\geq 1$. We can maintain a deterministic $(k+h)$-edge-coloring of G in amortized $O(\sqrt{nk/h})$ recolorings per insertion.

Proof. Recall the proof of Theorem 7, we will use the same idea of a color-swap on a bi-colored path. The key difference is how we use the $h \geq 1$ extra colors.

First consider $h = 1$ and denote the extra color as yellow. We allow at most y yellow edges in the graph, and if we need more, we recolor the whole graph from scratch without using yellow. Such a recoloring is possible because the graph is bipartite and every node is of degree at most k. When coloring a newly inserted edge (u, v), both u and v have at least one free color. We have three cases:

1. If u and v share a free color, including yellow: Then use this color.
2. If u does not have a yellow edge and v does (the other case is symmetric): Let c be a free color of v, and apply a color-swap of c and yellow with respect to v. This makes yellow a free color of v. Note that u is unaffected by the color-swap, because the graph is bipartite (affecting u implies that the path of the swap closes an odd cycle with the edge (u, v)). Now color (u, v) in yellow.
3. If both u and v have a yellow edge: Let c be a free color of u. Apply a color-swap of c and yellow with respect to u to make yellow a free color of u. Now apply the previous case.

We apply up to two color-swaps, each of length $O(y)$ because there are at most y yellow edges in the whole graph. Recall that we might have a global recoloring once we reach y yellow edges. We charge these recolorings to the yellow edges. Formally, we define a potential for the cache which equals $\frac{nk}{y} \cdot i$ when there are i yellow edges. Thus when we accumulate y yellow edges, the potential can pay for the global recoloring. Each insertion of an edge causes $O(y)$ recoloring and increases the potential by at most $\frac{nk}{y}$, due to possibly inserting a yellow edge (our color-swaps never increase the number of yellow edges). We conclude that the amortized cost is $O(y + \frac{nk}{y})$ per insertion. Balancing with $y = \sqrt{nk}$ gives $O(\sqrt{nk})$.

We generalize the previous logic for $h \geq 1$ by allowing each extra color to have at most y edges, and when it fills up we proceed to use the next extra color. Only when all h colors have y edges we invoke a full recoloring. The potential in this case is $\frac{nk}{hy}$ per edge, and the amortized cost is therefore $O(y + \frac{nk}{hy})$. Balancing with $y = \sqrt{nk/h}$ gives $O(\sqrt{nk/h})$. ◀

▶ **Theorem 27.** Given $k+h$ matchings to the algorithm compared to only k matchings to the optimum, for $h \geq k - 1$, there is a deterministic algorithm that is $2(1 + \frac{k-1}{\lfloor \frac{h+3-k}{2} \rfloor})$-competitive for Caching in Matchings. In particular, with $h = (1 + \alpha)k$ extra matchings we get a competitive ratio of $O(1 + \frac{1}{\alpha})$.

Proof. Let σ be a sequence of requests. Denote an algorithm A with cache parameter x as A^x, and use subscripts m for Caching in Matchings and c for Connection Caching. We have $cost(A_m^{2r-1}(\sigma)) = cost(A_c^r(\sigma))$ for any integer $r \geq 1$ by considering $r-1$ matchings as resource augmentation, such that we require no recoloring (by Lemma 20). Since this reduction halves the cache parameter, and our algorithm initially has cache of size $k + h$, we use $r = \lfloor \frac{k+h+1}{2} \rfloor$. If $k + h = 2r$, we do not use one of the colors, on purpose, to ensure using exactly $2r - 1$ colors. Taking A_c to be the algorithm that satisfies Lemma 16, $cost(A_c^r(\sigma)) \leq \frac{2r}{r-k+1} \cdot cost(OPT_c^k(\sigma)) + d$ for some fixed term d. By Remark 4, $cost(OPT_c^k(\sigma)) \leq cost(OPT_m^k(\sigma))$. Plugging everything together we get that $cost(A_m^{2r-1}(\sigma)) \leq \frac{2r}{r-k+1} \cdot cost(OPT_m^k(\sigma)) + d$, hence A_m^{2r-1} is $\frac{2r}{r-k+1} = 2(1 + \frac{k-1}{r-k+1}) = 2(1 + \frac{k-1}{\lfloor \frac{h+3-k}{2} \rfloor})$ competitive for Caching in Matchings. ◀

Streaming Edge Coloring with Asymptotically Optimal Colors

Mohammad Saneian
Northeastern University, Boston, MA, USA

Soheil Behnezhad
Northeastern University, Boston, MA, USA

Abstract

Given a graph G, an *edge-coloring* is an assignment of colors to *edges* of G such that any two edges sharing an endpoint receive different colors. By Vizing's celebrated theorem, any graph of maximum degree Δ needs at least Δ and at most $(\Delta + 1)$ colors to be properly edge colored. In this paper, we study edge colorings in the *streaming* setting. The edges arrive one by one in an *arbitrary* order. The algorithm takes a single pass over the input and must output a solution using a much smaller space than the input size. Since the output of edge coloring is as large as its input, the assigned colors should also be reported in a streaming fashion.

The streaming edge coloring problem has been studied in a series of works over the past few years. The main challenge is that the algorithm cannot "remember" all the color assignments that it returns. To ensure the validity of the solution, existing algorithms use many more colors than Vizing's bound. Namely, in n-vertex graphs, the state-of-the-art algorithm with $\widetilde{O}(ns)$ space[1] requires $O(\Delta^2/s + \Delta)$ colors. Note, in particular, that for an asymptotically optimal $O(\Delta)$ coloring, this algorithm requires $\Omega(n\Delta)$ space which is as large as the input. Whether such a coloring can be achieved with sublinear space has been left open.

In this paper, we answer this question in the affirmative. We present a randomized algorithm that returns an asymptotically optimal $O(\Delta)$ edge coloring using $\widetilde{O}(n\sqrt{\Delta})$ space. More generally, our algorithm returns a proper $O(\Delta^{1.5}/s + \Delta)$ edge coloring with $\widetilde{O}(ns)$ space, improving prior algorithms for the whole range of s.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Graph coloring

Keywords and phrases Streaming, Edge coloring, Adversarial order

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.121

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2305.01714` [18]

1 Introduction

Motivated by its applications in processing massive graphs, the *graph streaming model* has gained significant attention over the past two decades. This model assumes that the input graph is too large to be stored in memory and is presented to the algorithm in a sequence of edges. Many graph problems have been studied in this model, including maximum matching [7, 36, 5, 28, 6], graph connectivity [2, 13], vertex coloring [10, 12, 9, 19, 21, 8], edge coloring [17, 22, 4], cut and spectral sparsifiers [3, 37, 11, 24, 38], and graph clustering [15, 16, 26, 1] among many others (this is by no means a comprehensive list). In this paper, we continue the line of work on the *edge coloring* problem in the streaming model. The goal of edge coloring is to assign colors to edges of a graph such that no two adjacent edges share the same color.

[1] Here and throughout the paper, $\widetilde{O}(f) = O(f \cdot \operatorname{poly}\log n)$.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 121; pp. 121:1–121:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Since the output of edge coloring is as large as its input, a streaming algorithm cannot return it in memory. Instead, the goal is to also return the solution in a streaming fashion. Doing so, the main challenge is that the algorithm cannot "remember" all the reported edge colors, yet has to ensure that any two incident edges receive different colors.

Edge coloring is a fundamental problem in graph theory and has many practical applications in areas such as scheduling, communication networks, and VLSI design. By a classic result of Vizing, any graph of maximum degree Δ needs at least Δ and at most $(\Delta+1)$ colors to be properly edge colored.[2] While existing algorithms for finding a $(\Delta+1)$ edge coloring are rather complicated, a $(2\Delta-1)$ coloring can be found by a simple greedy algorithm that iterates over the edges and chooses an arbitrary available color for each. Unfortunately, even this simple greedy algorithm is hard to implement in the streaming setting. Recall that the algorithm cannot remember all the assigned edge colors in memory, hence it is unclear how to verify which color is available for the next edge that arrives.

Prior works. The streaming edge coloring problem was first studied by Behnezhad, Derakhshan, Hajiaghayi, Knittel, and Saleh [17] who gave a randomized algorithm for $O(\Delta^2)$ edge-coloring with $\widetilde{O}(n)$ space, where n is the number of vertices. In a follow up work, Charikar and Liu [22] showed that, more generally, for any parameter $s \geq 1$ there is a randomized streaming algorithm that $O(\Delta^2/s+\Delta)$ edge-colors the graph using $\widetilde{O}(ns)$ space. Later, Ansari, Saneian, and Zarrabi-Zadeh [4] obtained the same bound but using a simple and clean *deterministic* algorithm. Note that for an asymptotically optimal $O(\Delta)$ edge coloring, the algorithms above require $O(n\Delta)$ space. This, unfortunately, is *not* sublinear in the input size as any graph of maximum degree Δ has at most $O(n\Delta)$ edges. Put differently, for the case of $O(\Delta)$ coloring, no improvement over the trivial algorithm that stores the whole graph in memory and then colors it is known. This state of affairs leaves an important question open:

Does there exist a streaming algorithm with sublinear space for $O(\Delta)$ edge coloring?

Our contribution. In this paper, we answer the question above in the affiramtive. Our main result is the following algorithm:

▶ **Theorem 1.1.** *For any $s \geq 1$, there is a randomized streaming algorithm that with high probability reports a $O(\Delta^{1.5}/s+\Delta)$ edge-coloring under arbitrary edge arrivals using $\widetilde{O}(ns)$ space.*

Setting $s=\sqrt{\Delta}$, we obtain the following corollary, answering the question above:

▶ **Corollary 1.2.** *There is a randomized streaming algorithm that with high probability reports a $O(\Delta)$ edge-coloring under arbitrary edge arrivals using $\widetilde{O}(n\sqrt{\Delta})$ space.*

We show that the space-complexity can be further improved to $\widetilde{O}(n)$ under arbitrary *vertex* arrivals. Here instead of edges arriving in an arbitrary order, the vertices of the graph arrive one by one and when a vertex v arrives, all of its edges to the previous vertices are revealed.

[2] To see the lower bound, note that Δ colors are needed just to color the edges of a vertex with degree Δ.

▶ **Theorem 1.3.** *There is a randomized streaming algorithm that $O(\Delta)$ edge colors the graph under arbitrary vertex arrivals using $\widetilde{O}(n)$ space.*

1.1 Further Related Work

Streaming algorithms for edge coloring have also been studied under the extra assumption that the edges arrive in a random order. Behnezhad, Derakhshan, Hajiaghayi, Knittel, and Saleh [17] showed that there is a single pass $\widetilde{O}(n)$ space algorithm that obtains a 5.44Δ coloring. Charikar and Liu [22] later showed the number of colors can in fact be improved to $(1 + o(1))\Delta$ under random arrivals while keeping the memory $\widetilde{O}(n)$. Both of these algorithms rely heavily on the random-arrival assumption and do not have any implications for adversarial edge arrivals, which is the focus of this paper.

Online edge-coloring is another related problem. In this problem, the algorithm has no space constraints, but edges arrive one by one and each edge has to be colored upon arrival irrevocably. Note that the greedy algorithm can easily be implemented in the online setting; therefore much of the research has been focused on whether the greedy bound can be improved. For low-degree graphs, Bar-Noy, Motwani, and Naor [14] showed that the greedy algorithm is optimal for online edge coloring. However, there has been several improvements over the past few years for graphs of degree at least $\omega(\log n)$ starting from the work of Cohen, Peng, and Wajc [25]; see [25, 41, 20, 40] and the references therein. The best current bound for general edge arrivals is a beautiful $\frac{e}{e-1}\Delta$ coloring algorithm of Kulkarni, Liu, Sah, Sawhney, and Tarnawski [40]. Whether the number of colors can be improved to $(1 + o(1))\Delta$ for $\Delta = \omega(\log n)$ under arbitrary edge arrivals remains an important open problem.

Independent work. Two other works by Ghosh and Stoeckl [32] and Chechik, Mukhtar, and Zhan [23], which appeared concurrently with our paper, also consider the streaming edge coloring problem. The algorithm of Ghosh and Stoeckl [32], for any $1 \leq s \leq \sqrt{\Delta}$, obtains a (Δ^2/s^2)-edge-coloring using $\widetilde{O}(ns)$ space.[3] The algorithm of Chechik, Mukhtar, and Zhan [23] obtains a $\widetilde{O}(\Delta^{1.5})$-edge-coloring using $\widetilde{O}(n)$ space.

Our Theorem 1.1 subsumes both of these results: for any $1 \leq s \leq \sqrt{\Delta}$, we achieve a $(\Delta^{1.5}/s)$-edge-coloring using $\widetilde{O}(ns)$ space. Letting $s = 1$, this gives an $O(\Delta^{1.5})$-edge-coloring with $\widetilde{O}(n)$ space matching the result of Chechik, Mukhtar, and Zhan [23]. Moreover, since $\Delta^{1.5}/s \leq \Delta^2/s^2$ for the whole range of $1 \leq s \leq \sqrt{\Delta}$, our color/space trade-off is never worse than that of Ghosh and Stoeckl [32], but is strictly better when s is small. For instance, letting $s = 1$, the algorithm of Ghosh and Stoeckl [32] requires $O(\Delta^2)$ colors whereas ours requires $O(\Delta^{1.5})$ colors.

Finally, we note that Ghosh and Stoeckl [32] also consider vertex arrival streaming algorithms achieving the same bound as our Theorem 1.3. They also consider online algorithms.

1.2 Preliminaries

Unless otherwise stated, we use $G = (V, E)$ to denote the input graph. We use $n := |V|$ and $m := |E|$ to respectively denote the number of vertices and edges in G. We use Δ to denote the maximum degree of the graph G. For any integer k, we use $[k]$ to denote the set $\{1, \ldots, k\}$.

[3] In their paper this is equivalently stated as an $O(\Delta t)$ coloring using $\widetilde{O}(n\sqrt{\Delta/t})$ space.

The graph streaming model. In the standard graph streaming model, edges of an arbitrary graph G arrive one by one in an arbitrary order. The algorithm has a space much smaller than the total number of edges, can take few – preferrably just one – pass over the input, and should return the output. In the case of edge coloring, the output size is as large as the input, making it impossible to store the entire output and return it all at once. Therefore, we allow the algorithm to output the solution in a streaming manner as well. This model is also referred to as the "W-streaming" model in the literature [33, 27]. All of the algorithms presented in this paper take only a single pass over the input. We measure the space in the number of words, each consisting of $\Theta(\log n)$ bits.

Our algorithms for the general edge arrival model build on algorithms that we develop for two more restrictive arrival models of *general vertex arrivals* and *one-sided vertex arrivals in bipartite graphs*. We present the definition of these standard models below.

▶ **Definition 1.4** (vertex arrival model)**.** *In this model, vertices of the input graph G arrive one by one according to some arbitrary permutation π. Upon arrival of a vertex, all of its edges to previous vertices in the permutation π arrive.*

▶ **Definition 1.5** (one-sided vertex arrivals in bipartite graphs)**.** *In this model, the input graph G is assumed to be bipartite with vertex sets U and V. The "offline" vertices in V are present from the beginning, but the "online" vertices in U arrive one by one in an arbitrary order. Every time an online vertex u arrives, all of its edges to the offline vertices V are revealed.*

In our proofs, we use the following standard variant of the Chernoff bound.

▶ **Proposition 1.6** (Chernoff bound)**.** *Let $X_1, ..., X_n$ be independent random variables in $[0, 1]$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathbb{E}[X]$. Then for all $\delta \geq 0$ and $\mu' \geq \mu$, $\Pr[X \geq (1+\delta)\mu'] \leq \exp\left(-\frac{\delta^2}{2+\delta}\mu'\right)$.*

2 Overview of Techniques

In this section, we give an informal high-level overview of our algorithms.

As we discussed, the main challenge in solving the streaming edge-coloring problem is that the algorithm cannot "remember" all the colors that we assign to the edges as this takes too much space. This turns out to be a challenge particularly when the degrees evolve unevenly. To convey the key intuitions in this section, let us first focus on the one-sided vertex arrival model in bipartite graphs (Definition 1.5). We note that even in this restricted arrival model, the best known algorithm from the literature remains to be those of [17, 22, 4] which require $O(\Delta^2)$ colors with $O(n)$ space. Here, we describe how this can be improved to an asymptotically optimal $O(\Delta)$ coloring with only $\widetilde{O}(n)$ space. Our final algorithm of Theorem 1.1 in the more general edge-arrival model builds on this vertex-arrival algorithm.

Since all edges of an online vertex arrive at the same time, it is not hard to ensure they receive different colors. What is challenging is to do so while ensuring that all edges of an *offline* vertex receive different colors too. Towards this, we first describe an algorithm that uses $O(\Delta \log n)$ colors, $O(n\Delta)$ "pre-processing space," and $O(n)$ working space. We then show how this can be turned into an $O(\Delta)$ coloring algorithm that uses $\widetilde{O}(n)$ space overall.

With regards to the second reviewer's question, for the streaming model, we measure the space in terms of the number of words. We point this out in the preliminaries section. In the paragraph describing the algorithm that uses $O(\triangle \log n)$ colors the counters keep track of a number which is at most $K = \Theta(\Delta \log n)$. Thus it can be stored in $O(\log(K)) = O(\log n)$ bits of space. Since we set the size of words to be $\Theta(\log n)$, each counter can be stored in $O(1)$ space. Hence, to store counters for all the $O(n)$ vertices we need $O(n)$ space.

An algorithm with $O(\Delta \log n)$ colors but large space. Let $K = \Theta(\Delta \log n)$ be the number of colors we use. In the pre-processing step (i.e., before seeing any edges of the graph), for every offline vertex $v \in V$, we store a random permutation π_v of the colors $\{1, \ldots, K\}$ in memory which overall takes $\widetilde{O}(n\Delta)$ pre-processing space. Now suppose that the first online vertex u arrives. For each of its offline neighbors v_i, we consider the first random color $\pi_{v_i}(1)$ of v_i. Since the random permutations of offline vertices are independent, when $\Delta \geq 2$ we get the color chosen for v_i is different from $\pi_{v_j}(1)$ for all other neighbors v_j of u with probability $(1 - \frac{1}{K})^{\deg(u)} \geq (1 - \frac{1}{\Delta})^{\Delta} \geq 0.25$. Note that $(1 - \frac{1}{\Delta})^{\Delta}$ is a monotonically increasing and for $\Delta = 2$ the value is 0.25. If this happens, we assign color $\pi_{v_i}(1)$ to edge (u, v_i). If the first color of v_i is not unique, we discard it and reveal the next color $\pi_{v_i}(2)$ of v_i. Every round of this process successfully colors a constant fraction of the remaining uncolored edges of u. Thus it takes $O(\log n)$ rounds to color all edges of u w.h.p. On the other hand, since we have reserved $K = \Theta(\Delta \log n)$ colors for each offline vertex, we can afford to reveal up to $\Theta(\log n)$ colors for each of their edges, hence fully coloring the graph w.h.p. To implement this algorithm, we only need to maintain a counter d_v for every offline vertex v on how many colors of its random permutation we have revealed so far, which can be done with $O(n)$ total working memory.

Reducing space. To get rid of the huge pre-processing space of $\widetilde{O}(n\Delta)$ in the previous algorithm, we limit the amount of randomness needed. To do so, instead of choosing a fully random permutation of $\{1, \ldots, K\}$ for every offline vertex v, which requires $\Theta(nK) = \widetilde{\Theta}(n\Delta)$ space, we just pick a random number $r_v \in [K]$ and use the randomly *shifted* permutation $(r_v, r_v+1, \ldots, K, 1, \ldots, r_v-1)$. The advantage of doing so is that this shifted permutation can be stored using $O(1)$ space, by just storing the random number r_v in memory. Its downside is that lack of independence breaks the analysis above. For example, if $\pi_{v_i}(1) = \pi_{v_j}(1)$ then we also have $\pi_{v_i}(2) = \pi_{v_j}(2)$ and so cannot argue that every round colors a constant fraction of the edges of the online vertex u. To fix this, we take $t = \Theta(\log n)$ random numbers $r_v^1, \ldots, r_v^t$ for each offline vertex v and considering the t shifted permutations

$$[r_v^1, \ldots, \Delta, 1, \ldots, r_v^1 - 1], \ldots, [r_v^t, \ldots, \Delta, 1, \ldots, r_v^t - 1].$$

Now if the first color choice of v_i and v_j according to the first shifted permutation are the same, we consider the second shifted permutations, then the thirds, and so on and so forth. This gets rid of the dependence between the colors proposed for the edges of an online vertex, but not the edges of an offline vertex. Luckily, the latter is not needed for the analysis to go through and we can implement this algorithm with $\widetilde{O}(n)$ space overall.

Reducing colors via k-out sampling. The algorithm above is greedy in that we first reveal a random color for each edge of the online vertex u, greedily color those whose proposed colors are unique, then reveal the next batch of proposals. Instead of this greedy algorithm which takes up to $O(\log n)$ rounds, inevitably stretching the number of colors to $O(\Delta \log n)$, we first draw 3 random colors x_i^1, x_i^2, x_i^3 for each edge $e_i = (u, v_i)$ of the online vertex u and consider all these 3 colors *at the same time*. We show that with probability $1 - 1/\operatorname{poly}(\Delta)$, it is possible to pick one of the colors x_i^1, x_i^2, x_i^3 for each edge e_i of u such that all edges of u receive different colors. Our proof of this theorem builds on a new lemma (Lemma 3.6) that we prove on the existence of perfect matchings in a *one-sided random k-out model*. This lemma, which might be of independent interest, says that if we have a random bipartite graph with vertex sets V and U, and every vertex in V is made adjacent to exactly k vertices in U chosen uniformly, then G has a perfect matching provided that $k \geq 3$ and $|U| > e|V|$.

While k-out sampling has been used recently in obtaining more efficient algorithms for distributed graph connectivity [34] and minimum cuts [31, 11], this is to our knowledge its first application in graph coloring.

From vertex-arrivals to edge-arrivals. We now overview how we go from the vertex arrival model to the more general edge arrival model. To convey the key intuitions, we focus only on the simpler special case of Theorem 1.1 where $s = 1$. That is, an algorithm that uses $O(n)$ space and returns an $O(\Delta^{1.5})$ edge coloring. Our first step is to generalize the algorithm above to work in the non-bipartite vertex arrival model (Definition 1.4). This step, in fact, follows more or less from a known random bipartization technique of the literature that we present in Appendix A. Our next, more challenging, step is to generalize the algorithm to a *batch* arrival setting, where at each step instead of all of the edges of a vertex, we see some $\Theta(\sqrt{\Delta})$ edges of it (without having any guarantee about when the next batch of this vertex arrives). Once we achieve this more general algorithm, we run an instance $\mathcal{A}$ of it. To feed our edges to this batch arrival algorithm $\mathcal{A}$, we keep storing edges in a set H. Whenever the number of edges in H reaches n, we look at the vertex v with the largest degree in H. If $\deg_H(v) \geq \sqrt{\Delta}$, we feed $\Theta(\sqrt{\Delta})$ of these edges of vertex v as the next batch to algorithm $\mathcal{A}$ and remove them from H. Otherwise, the maximum degree in H is less than $\sqrt{\Delta}$; in this case, we edge color all edges in H greedily using $O(\sqrt{\Delta})$ colors and remove them all from H. Every time that we color H greedily, we color n edges of the graph. Therefore this happens at most $m/n = O(\Delta)$ times, requiring a total of $O(\Delta^{1.5})$ colors. The proof of Theorem 1.1 for larger values of s requires a more involved white-box application of the vertex-arrival algorithm; see the edge arrival section in the full version of our paper [18] for the details of the algorithm.

3 Streaming Edge Coloring Under Vertex Arrivals

In this section, we start by proving Theorem 1.3, restated below, for the streaming edge-coloring under vertex arrivals. Our algorithm in the more general edge-arrival model builds on the vertex-arrival algorithm that we describe in this section.

▶ **Theorem 1.3** (restated). *There is a randomized streaming algorithm that $O(\Delta)$ edge colors the graph under arbitrary vertex arrivals using $\widetilde{O}(n)$ space.*

3.1 Basic Reductions

We start with two basic reductions that essentially reduce the edge coloring problem in general graphs under vertex arrivals to the same problem in bipartite graphs under one-sided vertex arrivals.

The following lemma asserts that instead of general graphs, we can focus on bipartite graphs. The idea is to randomly partition the vertex set V into two sets A and B, edge color the bipartite subgraph of G between A and B, and recurse on the induced subgraphs $G[A]$ and $G[B]$. Since the idea is standard and follows from known reductions, we present its proof in Appendix B.

▶ **Lemma 3.1.** *Suppose there is a streaming algorithm that edge colors any n-vertex bipartite graph of maximum degree Δ with $f(\Delta) \geq \Delta$ colors using $s(n, \Delta)$ words of space. Then there is a streaming algorithm that edge colors any general (i.e., not necessarily bipartite) n-vertex graph of maximum degree Δ using*

$$f(\Delta) + f(\Delta/2) + f(\Delta/4) + ... + f(1)$$

colors and $\widetilde{O}(n + s(n, \Delta))$ space. If the algorithm for bipartite graphs works under vertex arrivals (resp. edge arrivals), then the algorithm for general graphs also works under vertex arrivals (resp. edge arrivals).

We emphasize that to solve general graphs under vertex arrivals, we have to ensure that the bipartite algorithm that we feed into Lemma 3.1 works under two-sided vertex arrivals. The next reduction shows that any one-sided vertex arrival algorithm also solves two-sided vertex arrivals by losing a factor of 2 in the number of colors.

▷ Claim 3.2. Suppose there is a streaming algorithm A, that edge colors any n-vertex bipartite graph of maximum degree Δ under one-sided vertex arrivals using $f(\Delta)$ colors. Then this can be turned into an algorithm that $2f(\Delta)$ colors the bipartite graph under two-sided vertex arrivals using asymptotically the same space.

Proof. Let $G = (V, U, E)$ be the bipartite graph under two-sided vertex arrivals. We partition its edges into two subgraphs G_U and G_V. If a vertex in V arrives, we add its edges to G_V, and if a vertex in U arrives, we add its edges to G_U. This way, G_V (resp. G_U) will be a bipartite graph under one-sided vertex arrivals with the online part being vertex set V (resp. U). Since both G_U and G_V will have maximum degree upper bounded by Δ (as they are subgraphs of G), we can run one instance of A on G_U and one on G_V in parallel using disjoint colors. This only multiplies the number of colors by two and the space by two, finishing the proof. ◁

3.2 The Algorithm

The reductions of the previous section show that instead of general graphs, we can focus on bipartite graphs under one-sided vertex arrivals. The following lemma is our main contribution in the rest of Section 3.

▶ **Lemma 3.3.** *There is a streaming algorithm that edge colors any bipartite graph of maximum degree Δ, using $O(\Delta)$ colors under one-sided vertex arrivals and uses $O(n)$ space w.h.p. (The coloring is always valid and the space-bound holds with probability $1 - 1/\operatorname{poly}(n)$.)*

Let us first see how Lemma 3.3 proves Theorem 1.3 using the reductions of the previous section.

Proof of Theorem 1.3. First, we feed the algorithm of Lemma 3.3 to Claim 3.2 to obtain a streaming algorithm that for some constant C, $C(\Delta)$ colors a bipartite graph under two-sided vertex arrivals with $O(n)$ space. We then plug in this algorithm into Lemma 3.1 to obtain an algorithm for general graphs. The algorithm uses $\widetilde{O}(n)$ space and the number of its colors by Lemma 3.1 is

$$C\Delta + C(\Delta/2) + C(\Delta/4) + \ldots + C \leq 2C\Delta = O(\Delta).$$ ◀

So it only remains to prove Lemma 3.3. In this section, we present the algorithm formalized below as Algorithm 1. We analyze the space complexity of the algorithm in Section 3.3 and analyze its correctness and the number of used by it in Section 3.4. Finally, we show in Section 3.5 that the algorithm can be generalized to a batch arrival model as well.

Algorithm 1. Streaming edge coloring for one-sided vertex arrivals in bipartite graphs.

Parameter: $c := 2.72$

1. For each offline vertex v draw 3 *distinct* random numbers r_v^1, r_v^2 and r_v^3 from $[c\Delta]$ uniformly and store them in memory.[a]
2. Additionally, for each offline vertex v, we store a counter $\deg_v$ in memory to keep track of its degree as the edges arrive.
3. Upon arrival of an online vertex u:
   a. For each edge $e_i = (u, v_i)$ consider the following three colors x_i^1, x_i^2, x_i^3:

$$x_i^1 := \big((r_{v_i}^1 + \deg_{v_i}) \bmod c\Delta\big),$$
$$x_i^2 := \big((r_{v_i}^2 + \deg_{v_i}) \bmod c\Delta\big) + c\Delta,$$
$$x_i^3 := \big((r_{v_i}^3 + \deg_{v_i}) \bmod c\Delta\big) + 2c\Delta.$$

   b. If there is an assignment of colors to edges of u such that each edge e_i receives a color from $\{x_i^1, x_i^2, x_i^3\}$ and all the edges of u receive different colors, then we assign these colors, stream them out, and remove edges of u from memory. Otherwise, we add all edges of u to set S which we store in memory.
   (In Claim 3.4 we show u's edges are successfully colored with probability $1 - O(1/\Delta^5)$.)
4. It only remains to color the edges in S. Note that since S is a subgraph of G, its maximum degree is no larger than Δ. Thus we can edge-color it using Δ fresh colors via existing offline edge-coloring algorithms for bipartite graphs; see e.g. [30].

[a] For brevity, here and in the rest of the analysis we assume that $c\Delta$ is an integer. If it is not, $c\Delta$ must be replaced by $\lceil c\Delta \rceil$.

3.3 Space Complexity

In Algorithm 1, for every vertex, we keep its degree and three random numbers which together can all be stored with $O(n)$ words. The only non-trivial part of the space that we need to bound is the size of the set S of the edges that we store in memory and color at the end. Our main result in this section is to show that the set S has size $O(n)$ w.h.p. We start by bounding the expected size of S in Claim 3.4 using a connection to k-out subgraphs (formalized in Lemma 3.6). We then prove a high probability bound on the size of S in Claim 3.7.

▷ Claim 3.4. Take any online vertex u. The probability that we store the edges of u in Step 3b of Algorithm 1 is at most $O(1/\Delta^5)$. This, by linearity of expectation, implies that

$$\mathbb{E}|S| \leq O(n/\Delta^4).$$

Proof. Our main idea to prove this claim is to use the maximum matching problem on an appropriately defined "color graph" defined below. When an online vertex u arrives in the input we construct the bipartite graph H_u in the following way.

▶ **Definition 3.5** (the color graph). *For any online vertex u, we define the color graph $H_u = (X, Y, E)$ as follows. The set X corresponds to the edges of u, i.e., for each edge (u, v_i), we have a vertex corresponding to v_i in X. For each color in the range $[c\Delta]$ we have a corresponding vertex in set Y. Each vertex in X has exactly three edges to Y which are to the three colors:*

$$y_i^1 := \big((r^1_{v_i} + \deg_{v_i}) \bmod c\Delta\big),$$
$$y_i^2 := \big((r^2_{v_i} + \deg_{v_i}) \bmod c\Delta\big),$$
$$y_i^3 := \big((r^3_{v_i} + \deg_{v_i}) \bmod c\Delta\big),$$

where v_i is any offline vertex adjacent to u. Note that since $r^1_{v_i}, r^2_{v_i}, r^3_{v_i}$ are distinct in Algorithm 1, the three colors y_i^1, y_i^2, y_i^3 will be distinct as well.

We first describe that a perfect matching in H_u corresponds to a valid edge-coloring of all edges of u. Let M be a perfect matching in H_u. Suppose that the node corresponding to edge e_i is matched in M to color y_i^j (which is distinct for all e_i's). In this case, we assign color x_i^j to e_i where x_j^i is as defined in Algorithm 1. To see why all edges of u receive different colors note that

$$y_i^j = x_i^j \bmod c\Delta.$$

In addition, all matched y_i^j are unique because M is a matching. Therefore, if we find a perfect matching in the color graph H_u we can easily color edges incident to u using this matching.

Thus, it remains to show that H_u does indeed have a perfect matching (with large enough probability). We do so using the following Lemma 3.6 on existence of perfect matchings in a so-called k-out model. Lemma 3.6 proves that if each vertex of one side of a bipartite graph is made adjacent to 3 random vertices on the other side, then the graph has a perfect matching provided that the other side is sufficiently large. We note that by a classic result of Frieze [29] from 1986, if vertices in *both* sides of the bipartite graph pick k random edges, then the graph has a perfect matching w.h.p. if and only if $k \geq 2$. However, this does not hold when only one side of the graph pick random edges, which is the focus of the following Lemma 3.6.

▶ **Lemma 3.6.** *Consider a random bipartite graph G with vertex sets V and U where each vertex $v \in V$ is adjacent to exactly 3 distinct vertices in U picked uniformly (from all subsets of size 3 of U) and independently from the choice of the rest of vertices in V. If $|V| \leq n$ and $|U| = cn$ for fixed $c > e$, the graph G has a perfect matching with probability at least $1 - O(1/n^5)$.*

Proof. First, we argue that we can w.l.o.g. assume that $|V| = n$. Since if we have $|V| < n$ we can add some dummy vertices to side V and draw random neighbors for them to U. Any perfect matching in this new graph gives us a perfect matching in the original graph by removing its dummy edges. Therefore the probability of finding a perfect matching in this graph is no smaller than the case with $|V| = n$.

We let $c = (1 + \varepsilon)e$ for some fixed $\varepsilon > 0$. We assume that n is larger than any needed constant (possibly a function of ε). Note that if n is fixed, then by adjusting the constants we can ensure $1 - O(1/n^5) = 0$, making the probablistic statement of the lemma trivial.

By Hall's theorem, G has a perfect matching iff for every subset $S \subseteq V$ it holds that $|N_G(S)| \geq |S|$. This condition deterministically holds if $|S| \leq 3$ as every vertex in V has degree exactly 3. Therefore it suffices to show that it holds with high probability for all sets of size at least 4. Let us fix $4 \leq k \leq n$. The probability that all the edges of a vertex $v \in V$ go to a fixed subset U of size at most $k - 1$ in U is exactly $\binom{k-1}{3}/\binom{|U|}{3}$. Therefore, we have

$$\Pr[\text{exists } S \subseteq V : |S| = k \text{ and } |N(S)| < k] \leq \binom{n}{k} \cdot \binom{|U|}{k-1} \cdot \left(\frac{\binom{k-1}{3}}{\binom{|U|}{3}}\right)^k$$

$$\leq \left(\frac{ne}{k}\right)^k \cdot \left(\frac{cne}{k-1}\right)^{k-1} \cdot \left(\frac{(k-1)(k-2)(k-3)}{cn(cn-1)(cn-2)}\right)^k$$

(since $\binom{n}{k} \leq \left(\frac{ne}{k}\right)^k$ for all n and k)

$$\leq \left(\frac{ne}{k}\right)^k \cdot \left(\frac{cne}{k-1}\right)^{k-1} \cdot \left(\frac{k-1}{cn/(1+0.1\varepsilon)}\right)^{3k}$$

(Here we use $cn/(1+0.1\varepsilon) \leq cn-2$ which holds for n larger than some constant.)

$$= \frac{(1+0.1\varepsilon)^{3k} e^{2k-1} (k-1)^{2k+1}}{c^{2k+1} \cdot k^k \cdot n^{k+1}}$$

$$\leq \left(\frac{(1+0.5\varepsilon)e}{c}\right)^{2k-1} \cdot \left(\frac{k}{n}\right)^{k+1}$$

(Here we use $(1+0.1\varepsilon)^2 \leq 1+0.5\varepsilon$ for small enough ε)

$$\leq \left(\frac{1+0.5\varepsilon}{1+\varepsilon}\right)^{2k-1} \left(\frac{k}{n}\right)^{k+1}. \qquad \text{(Since } c = (1+\varepsilon)e.\text{)}$$

To finish the proof, note that as discussed, for $k \leq 3$ the Hall's guarantee holds deterministically. For $k = 4$, the inequality above is upper bounded by $(k/n)^{k+1} = O(1/n^5)$ and thus all subsets of size $k = 4$ in V have at least 4 neighbors in U with probability at least $1 - O(1/n^5)$. For each choice of $5 \leq k \leq n^{0.1}$, the inequality above is upper bounded by $(k/n)^{k+1} \leq n^{-0.9\times 6} = n^{-5.4}$; a union bound over all $O(n^{0.1})$ such choices of k gives that with probability at least $1 - O(n^{-5.3})$, any set of size $5 \leq k \leq n^{0.1}$ in V has at least k neighbors in U as well. Finally, for the case where $n^{0.1} < k \leq n$, the inequality above is upper bounded by

$$\left(\frac{1+0.5\varepsilon}{1+\varepsilon}\right)^{2k-1} \left(\frac{k}{n}\right)^{k+1} \leq \left(\frac{1+0.5\varepsilon}{1+\varepsilon}\right)^{2k-1} = 2^{-O_\varepsilon(k)} = 2^{-O_\varepsilon(n^{0.1})} \ll O(1/n^6).$$

Thus, again by a union bound over all such choices of k, we get that with probability at least $1 - O(1/n^5)$, any set of size $n^{0.1} < k \leq n$ in V has at least k neighbors in U as well. Putting together all of these cases of k, we get that with probability at least $1 - O(1/n^5)$, Hall's condition holds and G has a perfect matching. ◀

Now observe that the color graph H_u meets the conditions of Lemma 3.6. There are at most Δ edges incident to u so $|X| \leq \Delta$. On the other hand $|Y| \geq c\Delta$. Moreover, for each edge $e_i = (u, v_i)$ its corresponding vertex in X is made adjacent to the vertices corresponding to colors y_i^1, y_i^2, y_i^3 as specified in Definition 3.5. Since the $r_{v_i}^j$'s are distinct and uniform, each vertex in X is made adjacent to 3 distinct uniform vertices in Y in graph H_u and so we can apply Lemma 3.6. This implies that H_u has a perfect matching with probability at least $1 - O(1/\Delta^5)$. This means that we store the edges of each online vertex in S with probability at most $O(1/\Delta^5)$, concluding the proof of Claim 3.4. ◁

It is worth noting that to find the maximum matching in H_u we use the algorithm of [35] that has space complexity of $O(|V| + |E|)$ which is $O(\Delta)$ here. We can delete this part of the memory before seeing the next online vertex.

While Claim 3.4 bounds the expected space of Algorithm 1, our next Claim 3.7 bounds the space by $O(n)$ w.h.p.

▷ Claim 3.7. Algorithm 1 uses $O(n)$ space with probability $1 - 1/\operatorname{poly}(n)$.

Proof. For an online vertex u, define X_u to be the indicator random variable where $X_u = 1$ iff we store edges of u in S. Additionally, define $X = \sum_{u \in U} X_u$. We know that

$$\Pr[|S| > O(n)] \leq \Pr[\Delta \cdot X > O(n)] = \Pr[X > O(n/\Delta)].$$

We know that $\mathbb{E}[X] \leq n \cdot O(1/\Delta^5)$ by Claim 3.4. However, since the X_u's are *not* independent we can not use Chernoff bound to provide a concentration bound on their sum. To see why, take two online vertices u_1, u_2 that share the same set of offline neighbors and come one after another in the input. If $X_{u_1} = 1$ then also $X_{u_2} = 1$ since the colors used for H_{u_2} are exactly the same as the colors used for H_{u_1} except they are shifted by one.

To bound X our plan is to divide all X_u's into groups such that in each group all the random variables are independent and then apply the Chernoff bound on each group separately. We know that if two online vertices u and w do not share any offline vertex as their neighbor then X_u and X_w are independent. This is because the colors specified in Step 3a of Algorithm 1 are all a function of the randomness on the set of neighbors an online vertex has. So if these sets are disjoint then X_u and X_w are independent.

Let us define the dependency graph between these variables as follows: The vertex set of this dependency graph is $\{X_u : u \in U\}$ and there is an edge between X_u, X_w with $u, w \in U$ iff u and w share an offline neighbor. The maximum degree in this graph is at most Δ^2 since each vertex $u \in U$ has at most Δ offline neighbors that each has at most Δ online neighbors.

Since any graph of maximum degree Δ' can be *vertex* colored using at most $\Delta' + 1$ colors via a simple sequential greedy algorithm, the dependency graph can be vertex colored via at most $\Delta^2 + 1$ colors. Doing so, note that vertices in each color class become independent random variables by definition of the dependency graph. Therefore, we can apply the Chernoff bound on the sum in each color class. With a slight abuse of notation, let $X_1, ..., X_t$ be a group of variables in a color class. Define $\mu = \mathbb{E}[X_1 + X_2 + ... + X_t]$. Since $\mathbb{E}[X_i] = O(1/\Delta^5)$ (by Claim 3.4) by linearity of expectation we get that for some constant C,

$$\mu = O(t/\Delta^5) \leq C \cdot t/\Delta^5. \tag{1}$$

Applying the Chernoff bound we get that for any $\delta > 0$ and $\mu' \geq \mu$,

$$\Pr[X_1 + X_2 + ... + X_t \geq (1+\delta)\mu'] \leq \exp\left(\frac{-\delta^2}{2+\delta}\mu'\right). \tag{2}$$

Let us call this color class *small* if $t \leq n/\Delta^3$ and *large* otherwise. For small groups, we do not need to prove any concentration bound, since a total of $(\Delta^2+1)(n/\Delta^3) = O(n/\Delta)$ vertices belong to small groups and each stores Δ edges in S, giving a deterministic upper bound of $O(n)$ on the number of such edges in S. It thus remains to analyze large groups only.

To deal with large groups we consider two cases for Δ.

Case 1: $\Delta = O((n/\log n)^{1/4})$. Letting $\delta = \Delta^4$ in Equation (2), we get:

$$\Pr\left[\sum_{i=1}^{t} X_i \geq (1+\Delta^4)\mu'\right] \leq \exp\left(\frac{-\Delta^8}{2+\Delta^4}\mu'\right).$$

Let $\mu' = C \cdot t/\Delta^5$. Note that $\mu \leq C \cdot t/\Delta^5 = \mu'$ and $t \geq \frac{n}{\Delta^3}$ since we are only considering large groups we get,

$$\exp\left(\frac{-\Delta^8}{2+\Delta^4}\mu'\right) \leq \exp\left(\frac{-\Delta^8}{2+\Delta^4} \cdot \frac{Cn}{\Delta^8}\right) = \exp\left(\frac{-Cn}{2+\Delta^4}\right).$$

Since we have $\Delta \leq O\left(\frac{n}{\log n}\right)^{1/4}$ the last term can be upper bounded by n^{-10}. Take $t_1, t_2, ..., t_k$ to be the size of the groups. We know that $k \leq n$; thus by a union bound over k, the probability that there is one group that deviates from its mean with a multiplicative factor of more than Δ^4 is at most n^{-9}. Therefore, with probability at least $1 - n^{-9}$, all groups deviate from their mean with a multiplicative factor of less than Δ^4. In this case, noting that $\sum_i t_i = n$, we get

$$X \leq \Delta^4 \cdot \sum_i \frac{C \cdot t_i}{\Delta^5} = O(n/\Delta).$$

Recalling that X is the total number of online vertices that store their edges in S, we get that the total number of edges in S is upper bounded by $\Delta \cdot O(n/\Delta) = O(n)$ w.h.p.

Case 2: $\Delta = \Omega((n/\log n)^{1/4})$. Notice that the probability of storing any u edges in S is $n^{-5/4+o(1)}$. By union bounding over all n online vertices, none will be stored in S with a probability of $O(n^{-0.25+o(1)})$. ◁

3.4 Correctness and the Number of Colors

In this section, we discuss why any two adjacent edges receive different colors in our algorithm. We can ignore the edges colored in Step 4 of Algorithm 1 as they use a totally new set of colors from the rest of the edges. So we focus on the rest of the edges in the remainder of this section.

For an online vertex u, its edges by Step 3b all receive different colors (and if this is not possible, we store u's edges in S which we discussed can be ignored earlier). So it remains to prove that there are no conflicts for the offline vertices.

For an offline vertex v, firstly, note that for any edge e_i:

$$x_i^1 \in [0, c\Delta), \qquad x_i^2 \in [c\Delta, 2c\Delta), \qquad x_i^3 \in [2c\Delta, 3c\Delta).$$

Since these ranges are disjoint, for two edges to receive the same color they must be in the same range.

Note that when the algorithm is running $\deg_v$ is increasing by one after we see an edge of v. Let $x_i^1 = x_j^1$ for two edges of v where e_i arrives before e_j. Define α to be the value of $\deg_v$ when the edge e_i arrives and β to be the value of $\deg_v$ when the edge e_j arrives. Then we get that,

$$\big((r_v^1 + \alpha) \bmod c\Delta\big) = \big((r_v^1 + \beta) \bmod c\Delta\big),$$

which is equivalent to $\alpha - \beta \bmod c\Delta = 0$. Since $|\alpha - \beta| \leq \Delta$ and $c > 1$, this is a contradiction and thus we cannot have $x_i^1 \neq x_j^1$. Same can be applied to the second and the third ranges of colors. Therefore, offline vertices also recieve distinct colors on all their edges and the coloring our algorithm finds is a valid edge coloring.

Finally, it can be verified from Algorithm 1 that the number of colors used is $3c\Delta + \Delta = O(\Delta)$. This section, put together with the previous section concludes the proof of Lemma 3.3 and, as discussed, this finishes the proof of Theorem 1.3.

3.5 Generalization to Batch Arrivals

Our Algorithm 1 considers the vertex arrival model, i.e., all edges of an online vertex arrive at the same time. Here, we consider a more general *batch arrival* model that interpolates between edge arrivals and vertex arrivals. We show that our Algorithm 1 can be generalized to this model.

▶ **Definition 3.8** (one-sided batch arrival model)**.** *In this model, we have a bipartite graph $G = (U, V, E)$ with* online *vertices U and* offline *vertices V. For a parameter k of the problem, edges arrive in batches of size k where all edges in the same batch are incident to the same online vertex u. The edges of a single online vertex v can arrive in up to Δ/k different batches. Importantly, there is no guarantee that different batches of the same online vertex arrive consecutively (otherwise the model would be equivalent to the vertex arrival model). We use $B_i(u)$ to denote the i'th batch of online vertex u.*

We prove that Algorithm 1 leads to the following bound in the one-sided batch arrival model. Note that setting $k = \Delta$ recovers Theorem 1.3, so this is only more general.

▶ **Lemma 3.9.** *There is a streaming algorithm that in the one-sided batch arrival model with batches of size k, uses $O(n)$ space and w.h.p. reports a proper $O(\Delta^2/k)$ edge coloring.*

Proof. Note that the only part of Algorithm 1 where we use the assumption that all edges of the online vertices arrive at the same is in Step 3b. There, the offline vertex of each edge e_i of u proposes three colors $\{x_i^1, x_i^2, x_i^3\}$ for e_i. Committing to choose a color from $\{x_i^1, x_i^2, x_i^3\}$ for each edge e_i already ensures there will be no conflicts among the edges of the offline vertices. On the other hand, our Lemma 3.6 guarantees that for each online vertex u, with probability $1 - O(1/\Delta^5)$, there is a proper coloring of all of its edges using their proposed colors. However, to find this proper coloring, it is important to have all edges of the online vertex at once. If, as in our case in this lemma, the edges of the online vertex arrive in batches, then we cannot ensure that edges in two different batches of the same online vertex u receive different colors. To fix this, first we only properly color the edges of each batch of the online vertex using their proposed colors. If this assigns color c to edge e, the final color that we report for e is (c, i) where i denotes which batch of u this edge e belongs to. Since there are at most Δ/k batches for each vertex, the total number of colors needed with this approach is $O(\Delta \cdot \Delta/k) = O(\Delta^2/k)$. Finally, note that the space remains $O(n)$ since the only additional information that we need is the number of batches that have arrived for each vertex which can be stored with $O(n)$ counters, one for each online vertex. ◀

4 Edge-Arrivals: $O(\Delta)$ Edge-Coloring with $\widetilde{O}(n\sqrt{\Delta})$ Space

In this section, we prove Theorem 1.1 for the special case where $s = \sqrt{\Delta}$. That is, we obtain an asymptotically optimal $O(\Delta)$ edge-coloring using $\widetilde{O}(n\sqrt{\Delta})$ space.

To solve the edge arrival model, we present a reduction to the one-sided batch arrival model of Definition 3.8. Throughout this section, we only consider the batch arrival model for batches of size $k := \lceil \sqrt{\Delta} \rceil$.

▷ Claim 4.1. Suppose that there is a streaming algorithm $\mathcal{A}$ that edge colors a bipartite graph of maximum degree Δ using $O(\Delta)$ colors under one-sided batch arrivals with batches of size $k = \Theta(\sqrt{\Delta})$ using space $\widetilde{O}(n\sqrt{\Delta})$. Then there is a streaming edge coloring $\mathcal{A}'$ algorithm that $O(\Delta)$ edge colors any general graph of maximum degree Δ under edge arrivals using $\widetilde{O}(n\sqrt{\Delta})$ space.

Proof. Let us first assume that graph G is bipartite with vertex sets U and V but its edges arrive in an arbitrary order. We start with a buffer $T = \emptyset$ and add any edge that arrives in the stream to T. Whenever there is some vertex u that has at least k edges in T, we remove those edges of u from T and feed them into the batch-arrival algorithm $\mathcal{A}$. This way, each batch has exactly k edges all adjacent to the same vertex. However, this vertex could belong to either of U and V. That is, this is not one-sided arrivals but rather two-sided

arrivals. The reduction to one-sided arrivals simply follows from Claim 3.2 at the expense of multiplying the final number of colors by two. Finally, if the stream ends and there are any remaining edges in T, we color them all using Δ colors via offline algorithms. Note that since each vertex in T has degree at most k at any point, the space of this reduction is only $O(nk) = O(n\sqrt{\Delta})$ as desired.

From the reduction above, we get that under edge arrivals, there is an algorithm $\mathcal{A}''$ that edge-colors the graph using $C(\Delta)$ colors for some constant C and $\widetilde{O}(n\sqrt{\Delta})$ space provided that the graph is bipartite. We now apply Lemma 3.1. This gives an algorithm for general graphs under edge-arrivals that uses $\widetilde{O}(n\sqrt{\Delta})$ space and the number of colors that it uses is at most

$$C\Delta + C\Delta/2 + C\Delta/4 + \ldots + C \leq 2C\Delta = O(\Delta). \quad \blacktriangleleft$$

Our goal for the rest of this section is to prove the following algorithm which plugged to Claim 4.1 proves Theorem 1.1 for $s = \sqrt{\Delta}$:

▶ **Lemma 4.2.** *There is a streaming algorithm that edge-colors any bipartite graph of maximum degree Δ under one-sided batch-arrivals (Definition 3.8) with batches of size $k = \Theta(\sqrt{\Delta})$ using $O(\Delta)$ colors and $\widetilde{O}(n\sqrt{\Delta})$ space.*

Proof. One may wonder whether we can now directly apply the naive algorithm of Lemma 3.9 in order to prove Lemma 4.2. This is not doable because Lemma 3.9 would require $O(\Delta^2/k) = O(\Delta^{1.5})$ colors which is much larger than our desired $O(\Delta)$ colors. So we need more ideas.

First, we assume that $\Delta \geq 300 \log^2 n$. If not, the whole graph has at most $O(n\Delta) = \widetilde{O}(n)$ edges, so we can store them all and run an offline Δ edge coloring algorithm.

Consider the subgraphs $H_1, H_2, \ldots, H_k$ where

$$H_i := \bigcup_{u \in U} B_i(u),$$

recalling that $B_i(u)$ is the i'th batch of online vertex u. In words, H_i is the collection of the i'th batches of all online vertices. Observe that the arrival order of the edges of each subgraph H_i follow the vertex arrival model. That is, for each online vertex v, all of its edges arrive at the same time. Note also that the maximum degree on the online side of each subgraph H_i is at most k. Now, if we also had the same upper bound of k on the degrees on the offline side of each subgraph H_i, then we could run $\Delta/k = O(\sqrt{\Delta})$ instances of Algorithm 1, which works under vertex arrivals, for each of the k subgraphs H_i all in parallel, using disjoint color palettes. This way, the total number of colors used would be $\frac{\Delta}{k} \cdot O(k) = O(\Delta)$ and the space would be $\frac{\Delta}{k} \cdot O(n) = O(n\sqrt{\Delta})$. Unfortunately, however, the offline vertices in each H_i may have degree as large as Δ. This is because all edges of an offline vertex might be in the first batch of their corresponding online vertices, making the maximum degree in H_1 equal to Δ rather than k. This is the main obstacle that we overcome in the remainder the proof.

Let us for each online vertex u choose a random number $b_u \in [k]$ at the beginning of the algorithm uniformly and independently at random. Now for the i'th batch of u define its *permuted batch number* $\rho_u(i)$ to be

$$\rho_u(i) := (i + b_u) \bmod k.$$

Doing so, we now redefine H_i for $i \in [k]$ as

$$H_i := \bigcup_{u \in U} B_{\rho_u^{-1}(i)}(u).$$

In words, H_i is now the graph including, for each online vertex u, its edges in its j'th batch such that $\rho_u(j) = i$. Since the total number of batches of each online vertex u is at most $\Delta/k \leq k$, all of u's batches receive different batch numbers. It remains to bound the degree of offline vertices in each H_i. The following lemma asserts that with high probability this is in fact $O(k)$.

▶ **Lemma 4.3.** *All H_i's have maximum degree at most $2k$ with probability $1 - 1/\operatorname{poly}(n)$.*

Due to space constraints, we defer the proof of this lemma to the appendix.

Now that both the offline and online vertices of each H_i have degree at most $2k$, we can follow the approach outline before to achieve a $O(\Delta)$ coloring with $O(n\sqrt{\Delta})$ space. The final algorithm is formalized below as Algorithm 2.

Algorithm 2. A streaming edge coloring for bipartite graphs under one-sided batch arrivals with batches of size $k = \Theta(\sqrt{\Delta})$. The algorithm uses $O(\Delta)$ colors and $O(n\sqrt{\Delta})$ space. This is the algorithm used for Lemma 4.2 which plugged into Claim 4.1 proves Theorem 1.1 for $s = \sqrt{\Delta}$.

Parameter: $k := \sqrt{\Delta}$, $c := 300$ (Lemma 4.3)

1. If $\Delta \leq c \cdot \log^2 n$ we read all the edges from the stream and color them with Δ colors, so assume for the rest of the algorithm that $\Delta > c \cdot \log^2 n$.
2. As the edges of G arrive, we decompose them into k subgraphs $H_1, \ldots, H_k$ and run k instances of the vertex-arrival Algorithm 1 in parallel on these subgraphs using disjoint colors in these instances, and an overall space of $O(nk)$. Each graph H_i will have maximum degree k, so overall we use $O(k \cdot k) = O(\Delta)$ colors.
3. For each online vertex u choose a random number $b_u \in [k]$ uniformly and independently.
4. For each online vertex u store a counter I_u. This will keep track of the number of *batches* of u seen at any point. Initially we have $I_u \leftarrow 0$.
5. Upon arrival of a batch of size k for an online vertex u:
   a. $I_u \leftarrow I_u + 1$.
   b. Add the edges of u to graph H_x where
   $$x := (I_u + b_u) \bmod k.$$
   c. These will be the only edges of u in graph H_x, thus this is indeed a bipartite graph under one-sided vertex arrivals and so we can run the instance of Algorithm 1 on graph H_x to color the edges of u in it.

Space-complexity. Since we are running $k = O(\sqrt{\Delta})$ instances of Algorithm 1 in Algorithm 2, the space needed for keeping all these instances is also $O(n\sqrt{\Delta})$ as each has a space of $O(n)$ words by Claim 3.7. Moreover, note that the size of T never exceeds nk or else there must be a vertex of degree at least k in it. We also use at most $O(n)$ space for the counters and random numbers (specified in Step 3 and 4). So in total, the space that Algorithm 2 uses is $O(nk) = O(n\sqrt{\Delta})$.

Number of colors. For the number of colors used by Algorithm 2, note that we run k instances of Algorithm 1, and in each instance the maximum degree is w.h.p. $2k$ by Lemma 4.3. Since Algorithm 1 uses linear colors in the maximum degree, the total number of colors is thus $O(k \cdot k) = O(\Delta)$ w.h.p.

This completes the proof of Lemma 4.2. ◀

As discussed, plugging Lemma 4.2 into Claim 4.1 completes the proof of Theorem 1.1 for $s = \sqrt{\Delta}$, concluding this section. For the final algorithm that does $O(\Delta^{1.5}/s + \Delta)$ Edge-coloring with $\widetilde{O}(ns)$ space see the full version of our paper at [18].

References

1 Kook Jin Ahn, Graham Cormode, Sudipto Guha, Andrew McGregor, and Anthony Wirth. Correlation clustering in data streams. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2237–2246, 2015. URL: http://proceedings.mlr.press/v37/ahn15.html.

2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 459–467, 2012. doi:10.1137/1.9781611973099.40.

3 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 5–14, 2012. doi:10.1145/2213556.2213560.

4 Mohammad Ansari, Mohammad Saneian, and Hamid Zarrabi-Zadeh. Simple streaming algorithms for edge coloring. In *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, pages 8:1–8:4, 2022. doi:10.4230/LIPICS.ESA.2022.8.

5 Sepehr Assadi. A two-pass (conditional) lower bound for semi-streaming maximum matching. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 708–742, 2022. doi:10.1137/1.9781611977073.32.

6 Sepehr Assadi and Soheil Behnezhad. Beating two-thirds for random-order streaming matching. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, pages 19:1–19:13, 2021. doi:10.4230/LIPICS.ICALP.2021.19.

7 Sepehr Assadi, Soheil Behnezhad, Sanjeev Khanna, and Huan Li. On regularity lemma and barriers in streaming and dynamic matching. In *Proccedings of the 55th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2023, to appear*, 2023.

8 Sepehr Assadi, Amit Chakrabarti, Prantar Ghosh, and Manuel Stoeckl. Coloring in graph streams via deterministic and adversarially robust algorithms. *CoRR*, abs/2212.10641, 2022. doi:10.48550/arXiv.2212.10641.

9 Sepehr Assadi, Andrew Chen, and Glenn Sun. Deterministic graph coloring in the streaming model. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 261–274, 2022. doi:10.1145/3519935.3520016.

10 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for $(\delta+1)$ vertex coloring. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 767–786, 2019. doi:10.1137/1.9781611975482.48.

11 Sepehr Assadi and Aditi Dudeja. A simple semi-streaming algorithm for global minimum cuts. In *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 172–180. SIAM, 2021.

12 Sepehr Assadi, Pankaj Kumar, and Parth Mittal. Brooks' theorem in graph streams: a single-pass semi-streaming algorithm for δ-coloring. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 234–247, 2022. doi:10.1145/3519935.3520005.

13 Sepehr Assadi and Janani Sundaresan. (Noisy) gap cycle counting strikes back: Random order streaming lower bounds for connected components and beyond. In *Proccedings of the 55th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2023, to appear*, 2023.

14 Amotz Bar-Noy, Rajeev Motwani, and Joseph Naor. The greedy algorithm is optimal for on-line edge coloring. *Inf. Process. Lett.*, 44(5):251–253, 1992. doi:10.1016/0020-0190(92)90209-E.

15 Soheil Behnezhad, Moses Charikar, Weiyun Ma, and Li-Yang Tan. Almost 3-approximate correlation clustering in constant rounds. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 720–731, 2022. doi:10.1109/FOCS54457.2022.00074.

16 Soheil Behnezhad, Moses Charikar, Weiyun Ma, and Li-Yang Tan. Single-pass streaming algorithms for correlation clustering. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, to appear*, 2023.

17 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Marina Knittel, and Hamed Saleh. Streaming and massively parallel algorithms for edge coloring. In *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, pages 15:1–15:14, 2019. doi:10.4230/LIPICS.ESA.2019.15.

18 Soheil Behnezhad and Mohammad Saneian. Streaming edge coloring with asymptotically optimal colors. *CoRR*, abs/2305.01714, 2023. doi:10.48550/arXiv.2305.01714.

19 Suman K. Bera, Amit Chakrabarti, and Prantar Ghosh. Graph coloring via degeneracy in streaming and other space-conscious models. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, pages 11:1–11:21, 2020. doi:10.4230/LIPICS.ICALP.2020.11.

20 Sayan Bhattacharya, Fabrizio Grandoni, and David Wajc. Online edge coloring algorithms via the nibble method. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2830–2842. SIAM, 2021. doi:10.1137/1.9781611976465.168.

21 Amit Chakrabarti, Prantar Ghosh, and Manuel Stoeckl. Adversarially robust coloring for graph streams. In *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, pages 37:1–37:23, 2022. doi:10.4230/LIPICS.ITCS.2022.37.

22 Moses Charikar and Paul Liu. Improved algorithms for edge colouring in the w-streaming model. In *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 181–183, 2021. doi:10.1137/1.9781611976496.20.

23 Shiri Chechik, Doron Mukhtar, and Tianyi Zhang. Streaming Edge Coloring with Subquadratic Palette Size. *CoRR*, abs/2305.07090, 2023. arXiv:2305.07090, doi:10.48550/arXiv.2305.07090.

24 Yu Chen, Sanjeev Khanna, and Huan Li. On weighted graph sparsification by linear sketching. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 474–485, 2022. doi:10.1109/FOCS54457.2022.00052.

25 Ilan Reuven Cohen, Binghui Peng, and David Wajc. Tight bounds for online edge coloring. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1–25. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00010.

26 Vincent Cohen-Addad, Silvio Lattanzi, Slobodan Mitrovic, Ashkan Norouzi-Fard, Nikos Parotsidis, and Jakub Tarnawski. Correlation clustering in constant many parallel rounds. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, pages 2069–2078, 2021. URL: http://proceedings.mlr.press/v139/cohen-addad21b.html.

27 Camil Demetrescu, Irene Finocchi, and Andrea Ribichini. Trading off space for passes in graph streaming problems. *ACM Trans. Algorithms*, 6(1), December 2010. doi:10.1145/1644015.1644021.

28 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, pages 531–543, 2004. doi:10.1007/978-3-540-27836-8_46.

29 Alan M. Frieze. Maximum matchings in a class of random graphs. *J. Comb. Theory, Ser. B*, 40(2):196–212, 1986. doi:10.1016/0095-8956(86)90077-8.

30 Harold N Gabow. Algorithms for edge-coloring graphs. *Technical Report*, 1985.

31 Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. Faster algorithms for edge connectivity via random 2-out contractions. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1260–1279. SIAM, 2020. doi:10.1137/1.9781611975994.77.

32 Prantar Ghosh and Manuel Stoeckl. Low-memory algorithms for online and w-streaming edge coloring. *CoRR*, abs/2304.12285, 2023.

33 Christian Glazik, Jan Schiemann, and Anand Srivastav. Finding euler tours in one pass in the w-streaming model with o(n log(n)) RAM. *CoRR*, abs/1710.04091, 2017. arXiv:1710.04091.

34 Jacob Holm, Valerie King, Mikkel Thorup, Or Zamir, and Uri Zwick. Random k-out subgraph leaves only o(n/k) inter-component edges. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 896–909. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00058.

35 John E. Hopcroft and Richard M. Karp. A n5/2 algorithm for maximum matchings in bipartite. In *12th Annual Symposium on Switching and Automata Theory (SWAT 1971)*, pages 122–125, 1971.

36 Michael Kapralov. Space lower bounds for approximating maximum matching in the edge arrival model. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1874–1893, 2021. doi:10.1137/1.9781611976465.112.

37 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 561–570, 2014. doi:10.1109/FOCS.2014.66.

38 Michael Kapralov, Aida Mousavifar, Cameron Musco, Christopher Musco, Navid Nouri, Aaron Sidford, and Jakab Tardos. Fast and space efficient spectral sparsification in dynamic streams. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1814–1833, 2020. doi:10.1137/1.9781611975994.111.

39 Howard J. Karloff and David B. Shmoys. Efficient parallel algorithms for edge coloring problems. *J. Algorithms*, 8(1):39–52, 1987. doi:10.1016/0196-6774(87)90026-5.

40 Janardhan Kulkarni, Yang P. Liu, Ashwin Sah, Mehtaab Sawhney, and Jakub Tarnawski. Online edge coloring via tree recurrences and correlation decay. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 104–116. ACM, 2022. doi:10.1145/3519935.3519986.

41 Amin Saberi and David Wajc. The greedy algorithm is not optimal for on-line edge coloring. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 109:1–109:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.109.

A Reduction To Bipartite Graphs

In this section, we present the proof of Lemma 3.1 using a random bipartization idea that has been used extensively in the literature of edge colorings. To our knowledge, this idea was first used by Karloff and Shmoys [39] in the context of parallel algorithms and by by Behnezhad et al [17] in the context of streaming edge coloring. We emphasize that we claim no novelty in the algorithm of this section and merely present it here for the sake of self-containment.

Proof of Lemma 3.1. If $\Delta \leq 10 \log n$, then we store the whole graph in memory using $\widetilde{O}(n)$ space and edge color the graph via $(\Delta + 1)$ colors using offline algorithms. So assume that $\Delta > 10 \log n$.

Let V be the vertex-set of the graph. We partition V into two subsets A and B by putting each vertex randomly in either A or B chosen uniformly and independently from the rest of the vertices. Now consider the bipartite subgraph G' of G including any edge (u, v) of G that has one endpoint in A and one endpoint in B. Note that conditioned on $v \in A$, each neighbor of v in the original graph belongs to B independently with probability $1/2$. This means that the expected degree of v in G' is $\deg_G(v)/2$. By a simple application of the Chernoff bound and recalling that $\Delta > 10 \log n$, we get that G' has maximum degree $\Delta/2 + O(\sqrt{\Delta \log n}) = (1/2 + o(1))\Delta$ with probability $1 - 1/n^{10}$. We can thus run the bipartite edge coloring algorithm on G'. Notice, however, that this leaves the edges that go from A to A or from B to B uncolored. We recursively apply the same algorithm. That is, we recursively partition A and B into two subsets each. This results in a subgraph of maximum degree $\Delta/4 + O(\sqrt{\Delta \log n})$ w.h.p. We continue this for $O(\log \Delta)$ steps until the resulting subgraph has maximum degree smaller than $O(\log n)$, at which point we store the whole subgraph in memory and color it using offline algorithms. Overall, the total number of colors used by the algorithm is

$$\sum_{i=1}^{\log \Delta} f\left(\left(\frac{1}{2^i} + o(1)\right)\Delta\right) \leq \sum_{i=1}^{\log \Delta} f\left(\frac{\Delta}{2^{i-1}}\right).$$

Note that this partitioning of the vertices can be done at the beginning of the stream before any edges arrive. It only suffices to store, for each vertex, which of the two random subsets it belongs to at each of the $O(\log \Delta)$ levels. So this only requires an overhead space of $O(n \log \Delta)$. Additionally, if the edges of the original graph arrive under vertex arrivals, then so do the edges of the random bipartite graphs. ◀

B Deferred Proofs

▶ **Lemma 4.3** (restated). *All H_i's have maximum degree at most $2k$ with probability $1 - 1/\operatorname{poly}(n)$.*

Proof. Due to symmetry, let us focus on bounding the degree of a given offline vertex v in H_1. Let neighbors of v be $u_1, u_2, ..., u_l$ where each u_i is an online vertex. Define X_i to be the indicator random variable where $X_i = 1$ iff the edge (v, u_i) receives a batch number of 1. The degree of v in H_1 can be written as

$$X = \sum_{i=1}^{l} X_i.$$

Since $l \leq \Delta$ and also $\mathbb{E}[X_i] = \frac{1}{k}$, by linearity of expectation, we get

$$\mathbb{E}[X] = \sum_{i=1}^{l} \mathbb{E}[X_i] \leq \frac{\Delta}{k} \leq k.$$

Observe that all b_{u_i}'s are chosen at the beginning of the algorithm and uniformly at random. Therefore the batch number of any given edge being 1 has a probability of $\frac{1}{k}$. Since all b_{u_i}'s are independent of each other, also all X_i's will be independent. Thus, we can apply the Chernoff bound for $\delta = 1$ and having $\Delta \geq 300 \log^2 n$ (as assumed at the start of the proof of Lemma 4.2) we get,

$$\Pr[\text{degree } v \text{ in } H_1 \geq 2k] = \Pr\left[\sum_{i=1}^{l} X_i \geq 2k\right] \leq \exp\left(\frac{-k}{3}\right) \leq n^{-10}$$

By a union bound over all n choices of v and over k choices of H_i, we get

$$\Pr[\text{exists } v \in V \text{ and } i \in [k] \text{ such that degree } v \text{ in } H_i \geq 2k] \leq n^{-8}.$$ ◀

An Improved Integrality Gap for Disjoint Cycles in Planar Graphs

Niklas Schlomberg ✉
Research Institute for Discrete Mathematics and Hausdorff Center for Mathematics,
University of Bonn, Germany

Abstract

We present a new greedy rounding algorithm for the Cycle Packing Problem for uncrossable cycle families in planar graphs. This improves the best-known upper bound for the integrality gap of the natural packing LP to a constant slightly less than 3.5. Furthermore, the analysis works for both edge- and vertex-disjoint packing. The previously best-known constants were 4 for edge-disjoint and 5 for vertex-disjoint cycle packing.

This result also immediately yields an improved Erdős–Pósa ratio: for any uncrossable cycle family in a planar graph, the minimum number of vertices (edges) needed to hit all cycles in the family is less than 8.38 times the maximum number of vertex-disjoint (edge-disjoint, respectively) cycles in the family.

Some uncrossable cycle families of interest to which the result can be applied are the family of all cycles in a directed or undirected graph, in undirected graphs also the family of all odd cycles and the family of all cycles containing exactly one edge from a specified set of demand edges. The last example is an equivalent formulation of the fully planar Disjoint Paths Problem. Here the Erdős–Pósa ratio translates to a ratio between integral multi-commodity flows and minimum cuts.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Mathematics of computing → Approximation algorithms

Keywords and phrases Cycle packing, planar graphs, disjoint paths

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.122

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2404.17813

Acknowledgements Thanks to Luise Puhlmann and Jens Vygen for listening, reading and improvement ideas.

1 Introduction

Given a family $\mathcal{C}$ of cycles in a (directed or undirected) graph G, the Cycle Packing Problem asks for a maximum-cardinality subset $\mathcal{C}^* \subseteq \mathcal{C}$ of pairwise vertex- or edge-disjoint cycles. It admits the natural packing LP

$$\max \left\{ \sum_{C \in \mathcal{C}} x_C : \sum_{C \in \mathcal{C}: v \in C} x_C \leq 1 \ (v \in V),\ x_C \geq 0 \ (C \in \mathcal{C}) \right\} \tag{1}$$

for vertex-disjoint cycle packing and

$$\max \left\{ \sum_{C \in \mathcal{C}} x_C : \sum_{C \in \mathcal{C}: e \in C} x_C \leq 1 \ (e \in E),\ x_C \geq 0 \ (C \in \mathcal{C}) \right\} \tag{2}$$

for edge-disjoint cycle packing. Despite its exponentially many variables, optimum LP solutions can be computed in polynomial time if $\mathcal{C}$ is given by a *weight oracle* [24]:

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 122; pp. 122:1–122:15


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Definition 1** ([24]). *Let $\mathcal{C}$ be a family of cycles in a graph G. $\mathcal{C}$ has a* weight oracle *if for any edge weights $w\colon E(G) \to \mathbb{R}_{\geq 0}$ we can compute a weight-minimal cycle in $\mathcal{C}$ in polynomial time.*

For arbitrary graphs the integrality gap of the LPs (1) and (2) is unbounded even if $\mathcal{C}$ is the set of all odd cycles in G [20, 18]. For planar graphs however, Schlomberg, Thiele and Vygen [24] have recently shown constant upper bounds for the integrality gaps if the cycle family $\mathcal{C}$ is *uncrossable*.

▶ **Definition 2** (Goemans, Williamson [10]). *A family $\mathcal{C}$ of cycles in a graph is called* uncrossable *if the following property holds.*

Let $C_1, C_2 \in \mathcal{C}$ and let P_2 be a path in C_2 such that P_2 shares only its endpoints with C_1. Then there is a path P_1 in C_1 between these endpoints such that $P_1 + P_2 \in \mathcal{C}$ and $(C_1 - P_1) + (C_2 - P_2)$ contains a cycle in $\mathcal{C}$ (as an edge set).

In their work they give an upper bound of 5 for the vertex-disjoint cycle packing LP (1), using a greedy rounding algorithm. For the edge-disjoint LP (2) they show an upper bound of 4, generalizing a similar result for the edge-disjoint paths problem by Garg, Kumar and Sebő [9]. In this work we modify their greedy rounding algorithm and analyze it using a new structural lemma. This improves the integrality gaps of both LPs to below 3.5:

▶ **Theorem 3.** *Let G be a planar graph, embedded in the sphere, and $\mathcal{C}$ an uncrossable family of cycles in G. Then there exists an integral solution to the vertex- or edge-disjoint cycle packing LP (1) or (2) with at least $\frac{6}{13+3\sqrt{7}} > \frac{1}{3.5}$ the LP value. If $\mathcal{C}$ is given by a weight oracle we can compute such a solution in polynomial time.*

1.1 The Erdős–Pósa ratio

The duals of the LPs (1) and (2) are relaxations of the Cycle Transversal Problem: This asks for a minimum subset of vertices or edges, respectively, that hit each cycle in $\mathcal{C}$. Berman and Yaroslavtsev [2] have shown an upper bound of 2.4 for the integrality gaps of the edge and vertex cycle transversal LPs, improving on a previous bound of 3 by Goemans and Williamson [10]. Multiplying the integrality gaps of the primal and dual LPs directly yields a maximum ratio between integral solutions to the primal and the dual:

▶ **Corollary 4.** *Let G be a planar graph and $\mathcal{C}$ an uncrossable family of cycles in G. Let ν_v respectively ν_e be the maximum number of vertex- and edge-disjoint cycles in $\mathcal{C}$ and let τ_v respectively τ_e be the minimum size of vertex and edge transversals for $\mathcal{C}$.*
Then $\frac{\tau_v}{\nu_v} \leq 2.4 \cdot \frac{13+3\sqrt{7}}{6} \leq 8.38$ and $\frac{\tau_e}{\nu_e} \leq 2.4 \cdot \frac{13+3\sqrt{7}}{6} \leq 8.38$

The supremum of $\frac{\tau_v}{\nu_v}$ respectively $\frac{\tau_e}{\nu_e}$ is known as the Erdős–Pósa ratio for the Cycle Packing and Transversal Problem. The previously best-known upper bound for general uncrossable cycle families was 12 for vertex-disjoint Cycle Packing and 9.6 for edge-disjoint Cycle Packing [24]; both resulting from multiplying the upper bounds for the primal and dual integrality gaps.

1.2 New Techniques

Our main algorithm that we use to find integral solutions to the Cycle Packing LP with the claimed approximation guarantee is similar to the greedy rounding algorithm used in [24]: Similar to [24], we start by solving the LP and applying an uncrossing procedure to obtain an optimum LP solution where the cycles in the support form a *laminar* family $\mathcal{L}$. Afterwards, we iteratively pick a set $\mathcal{F}^* \subseteq \mathcal{L}$ of pairwise disjoint cycles, add them to our solution and remove all "neighbours", i.e. cycles that are not disjoint to $\mathcal{F}^*$, from the support.

Schlomberg, Thiele and Vygen [24] showed that they can always find a single cycle C^* with LP value at most 5 on its neighbourhood. In this work we prove a new Structure Lemma, showing that after a slight modification of our LP solution, for a *one-sided* cycle C (i.e. a cycle with a side that contains no other cycles in $\mathcal{L}$) the average LP value on the neighbours of C without C itself is at most 3.

In Section 3 we observe that this already improves the bound of 5 from [24] to 4. However, by exploiting that we can also add several cycles in a single iteration to our solution, we can improve the bound further to below 3.5 (see Section 5). To this end, we add further candidates for our set $\mathcal{F}^*$: For any $\frac{1}{4} \leq \alpha < \frac{1}{2}$ we can find a large set of pairwise disjoint cycles among the one-sided cycles with LP value at least α due to the Four Colour Theorem. Using either a single cycle or one of those candidates for $\mathcal{F}^*$ in each iteration of the greedy rounding algorithm allows us to bound the integrality gap of the Cycle Packing LP by $\frac{13+3\sqrt{7}}{6}$.

Key of the new results is the Structure Lemma 13, which we prove by constructing a set M^* of LP constraints (i.e. vertices or edges) that cover each cycle in $\mathcal{L}$ enough often. Section 4 is dedicated to the construction of M^*. We proceed by an induction on the number of cycles in $\mathcal{L}$. If all cycles are one-sided, an auxiliary graph, similar to the planar dual, directly yields a feasible set M^*. Otherwise, we pick a minimal two-sided cycle, find feasible sets M^* for both sides of it and carefully put them together to a solution for the whole family.

1.3 Examples for uncrossable cycle families

There are several examples for cycle families in G that are always uncrossable and many of them have been studied individually. A list of the most interesting examples together with proofs of their uncrossability can be found in [24].

The first example of interest is the set of all cycles in an undirected graph G. For this problem Erdős and Pósa [6] showed that even in general, not necessarily planar, graphs with bounded cycle packing number the transversal number is bounded, although in general the ratio is unbounded. This property is known as the Erdős–Pósa property. In planar graphs the Erdős–Pósa ratio is 4 for edge-disjoint packing (the upper bound comes from a result by Ma, Yu and Zang [16], tightness was shown by an example by Král' [16]). For vertex-disjoint packing [4] and [16] gave an upper bound of 3 on the Erdős–Pósa ratio.

Also the set of all directed cycles in a digraph G is uncrossable. Here again the Erdős–Pósa property holds on arbitrary graphs [21]. For planar G the famous Lucchesi-Younger Theorem [15] shows that the edge-disjoint version has Erdős–Pósa ratio 1. For the vertex-disjoint version (1) in planar graphs, Reed and Shepherd [22] gave the first constant upper bound on the Erdős–Pósa ratio. After three improvements by Fox and Pach as well as Cames van Batenburg, Esperet and Müller [3] and then Schlomberg, Thiele and Vygen [24], this work decreases it below 8.38.

The next example of an uncrossable family is the set of all odd cycles in an undirected graph G. In this variant (in planar graphs) the edge-disjoint problem has an Erdős–Pósa ratio of exactly 2 [14]. For the vertex-disjoint problem Fiorini et al. [7] showed that the Erdős–Pósa ratio is at most 10, which was improved to 6 by Král', Sereni and Stacho [13].

Finally, one of the most interesting and well-studied variants of the Cycle Packing Problem is given as follows: Given a graph G and a set D of demand edges, then a D-cycle is a cycle in G that contains exactly one demand edge. Since removing the demand edge from a D-cycle results in a path between the endpoints of the demand edge, the D-Cycle Packing Problem is equivalent to the Disjoint Paths Problem, and D-Cycle Packing in planar graphs corresponds to the Disjoint Paths Problem in fully planar instances.

For the Fully Planar Edge-Disjoint Paths Problem the first constant-factor approximations and bounds on the integrality gap were given by Huang et al. [12] and Garg, Kumar and Sebő [9]; the best upper bound on the integrality gap is 4 [9]. Garg and Kumar [8] showed that also the Erdős–Pósa ratio for this problem is at most 4. Due to a result by Middendorf and Pfeiffer [17] the Fully Planar Vertex-Disjoint Paths Problem contains the edge-disjoint version as a special case. The first constant upper bound on the integrality gap of 5 was found only recently by Schlomberg, Thiele and Vygen [24]. The best upper bound for the Erdős–Pósa ratio of 12 comes in this variant from multiplying the upper bounds for the integrality gaps of the primal and the dual.

This work now decreases both best-known upper bounds on the integrality gaps to the same value below 3.5. For the Fully Planar Vertex-Disjoint Paths Problem we also improve the Erdős–Pósa ratio to below 8.38:

▶ **Corollary 5.** *Given an instance (G, D) of the Fully Planar Vertex-Disjoint Paths Problem, we can compute in polynomial time a set $\mathcal{P}$ of vertex-disjoint D-cycles and $T \subseteq V(G)$ such that every D-cycle contains a vertex of T with $|T| \leq 2.4 \cdot \frac{13+3\sqrt{7}}{6}|\mathcal{P}| \leq 8.38|\mathcal{P}|$.*

For most of the uncrossable families discussed above a lower bound of 2 on the integrality gaps of (1) and (2) is known, which is also the best-known lower bound for general uncrossable families. Most of the corresponding examples can be constructed by modifying a K_4. Regarding the Erdős–Pósa ratio, the best-known lower bound for vertex-disjoint cycle packing (for uncrossable families) is still 2, but for edge-disjoint cycle packing and transversal Král' (see [16]) showed a lower bound of 4 on the Erdős–Pósa ratio for the family of all cycles in G. See [24] for a more detailed overview on lower bounds for the integrality gaps and Erdős–Pósa ratios.

There exist other examples of uncrossable cycle families that have been studied. For example, Rautenbach and Regen [19] considered the Cycle Packing Problem with the family of shortest cycles in G, which is also uncrossable. Furthermore, the (uncrossable) family of all cycles that contain at least one vertex from a specified set $S \subseteq V(G)$ has been considered, for example by Goemans and Williamson [10]. This work yields the best-known upper bounds for the integrality gaps of the corresponding Cycle Packing LPs.

For cycle families $\mathcal{C}$ that are not uncrossable surprisingly few is known. For example, the set of all even cycles is not uncrossable. Here Göke et al. [11] generalized Goemans and Williamson's [10] technique to get a constant upper bound on the vertex transversal LP; for the Cycle Packing Problem no constant-factor approximation algorithm is known.

2 Preliminaries

For the rest of the paper we fix a planar graph G, together with an embedding in the sphere $\mathbb{S}^2$, and an uncrossable family $\mathcal{C}$ of cycles in G. By a result by Schlomberg, Thiele and Vygen [24] we can compute optimum solutions to (1) and (2) with *laminar* support, i.e. any two cycles in the support can only "touch" but not "cross":

▶ **Definition 6.** *Let G be a planar graph, embedded in the sphere. Deleting the embedding of a cycle C in G from the sphere results in two connected components of the sphere, which we call the* sides *of C. Given a side S of C and another cycle C' in G, we say that C' is* inside *S or that S* contains *C' if S contains a side of C'.*

We call a family $\mathcal{L}$ of cycles in G laminar *if for any $C_1, C_2 \in \mathcal{L}$ there exist sides S_1 of C_1 and S_2 of C_2 that are disjoint.*

▶ **Definition 7.** *Let $\mathcal{L}$ be a laminar family of cycles in a planar graph G, embedded in the sphere. By $V(\mathcal{L})$ and $E(\mathcal{L})$, respectively, we denote the set of all vertices, respectively edges, in cycles of $\mathcal{L}$.*

The cycles corresponding to $\subseteq$-minimal sides in $\mathcal{L}$ are called one-sided*, while the others are called* two-sided*. For a one-sided cycle, the $\subseteq$-minimal side is also called* one-sided*.*

We call two cycles $C_1, C_2 \in \mathcal{L}$ homotopic *if there exist sides S_1 of C_1 and S_2 of C_2 that contain the same set of one-sided sides.*

For any connected component D of $E(\mathcal{L})$ we call the set of all cycles in $\mathcal{L}$ that are in D a connected component *of $\mathcal{L}$.*

A chain *is a laminar family of cycles with only two one-sided sides.*

It is easy to see that the sides S_1 and S_2 in Definition 6 are unique if $C_1 \neq C_2$. Also, our notion of laminarity is equivalent to the definition in [24]. In particular we can use the following Lemma from [24]:

▶ **Lemma 8.** *Let G be a planar graph, embedded in the sphere, and $\mathcal{C}$ an uncrossable family of cycles in G. Then there exist optimum solutions to the LPs (1) and (2) with laminar support. If $\mathcal{C}$ has a weight oracle such solutions can be computed in polynomial time.*

3 Bounding the integrality gap

In this section we explain our main algorithm, which is a slight generalization of the greedy rounding algorithm used in [24]. We first only analyze the easiest variant of the algorithm. This already yields an upper bound of 4 on the integrality gap for the cycle packing LP, equalizing the best known upper bounds for edge-disjoint and vertex-disjoint Cycle Packing. In Section 5 we will analyze a more refined version of the algorithm, which yields an upper bound of below 3.5.

Here we only describe the algorithm for vertex-disjoint cycle packing. The edge-disjoint version can be deduced similarly or sometimes even easier; also there exists a reduction for laminar cycle families ([24]) that allows us to immediately transfer our results from vertex-disjoint to edge-disjoint packing. For more details we refer to Section 5.2.

▶ **Definition 9.** *Let $\mathcal{L}$ be a laminar family of cycles in a planar graph G, embedded in the sphere. Let $\mathcal{L}_1$ be the set of one-sided cycles in $\mathcal{L}$. For any $C \in \mathcal{L}$ let $\mathcal{N}_{\mathcal{L}}(C)$ be the set of "neighbours" of C, i.e. cycles in $\mathcal{L}$ that contain a vertex of C. In particular, $C \in \mathcal{N}_{\mathcal{L}}(C)$. Define $\mathcal{N}^1_{\mathcal{L}}(C) := \mathcal{N}_{\mathcal{L}}(C) \cap \mathcal{L}_1$ to be the set of one-sided "neighbours" of C.*

Given this definition, we can outline our algorithm:

Algorithm 1 Greedy Rounding for Cycle Packing.

Input: A planar graph G and an uncrossable family $\mathcal{C}$ of cycles in G.
Output: A set $\mathcal{L}^* \subseteq \mathcal{C}$ of pairwise vertex-disjoint cycles.

Compute an embedding of G in the sphere.
Compute an optimum solution x to the LP (1) with laminar support.
while $x \neq 0$ **do**
  Modify x to make it structured (see Definition 11).
  Let $\mathcal{L}_x := \{C \in \mathcal{C} : x_C > 0\}$ be the support of x.
  Pick a non-empty subset $\mathcal{F}^* \subseteq \mathcal{L}_x$ of pairwise vertex-disjoint cycles.
  Add all cycles in $\mathcal{F}^*$ to the solution $\mathcal{L}^*$.
  Set $x_C := 0$ for all $C \in \bigcup_{C' \in \mathcal{F}^*} \mathcal{N}_{\mathcal{L}_x}(C')$.
end while
Output $\mathcal{L}^*$.

Throughout the algorithm we maintain a feasible solution x to the LP (1) with laminar support $\mathcal{L}_x$ and a set $\mathcal{L}^*$ of pairwise vertex-disjoint cycles with $V(\mathcal{L}_x) \cap V(\mathcal{L}^*) = \emptyset$. Step 1 can be done in polynomial time [5]. For step 2 we apply Lemma 8. Then, in each iteration we add a set $\mathcal{F}^*$ of cycles in the support of x to $\mathcal{L}^*$ and set x on all neighbours of $\mathcal{F}^*$ to 0.

We will make use of the following observation: If in each iteration we find a set $\mathcal{F}^*$ with $x\left(\bigcup_{C\in\mathcal{F}^*} \mathcal{N}_{\mathcal{L}_x}(C)\right) \leq \alpha|\mathcal{F}^*|$ then the algorithm will return a solution of size at least $\frac{1}{\alpha}$ times the LP value. Thus, in order to bound the integrality gap of the cycle packing LP we only need to analyze the minimum values of $\frac{x\left(\bigcup_{C\in\mathcal{F}^*} \mathcal{N}_{\mathcal{L}_x}(C)\right)}{|\mathcal{F}^*|}$ that we can achieve with different choices of $\mathcal{F}^*$.

Note that after step 2 the algorithm only operates on the (explicitly given) set of cycles in the support of the LP solution and does not depend on $\mathcal{C}$ any more. Both the uncrossing property and the weight oracle are used only in this step. In particular, our results apply to any cycle family $\mathcal{C}$ where step 2 can be done, for example if $\mathcal{C}$ is already laminar.

We first explain in more detail what step 4 does:

▶ **Definition 10.** *Let $\mathcal{L}$ be a laminar family of cycles in a planar graph G, embeddded in the sphere. We call a two-sided cycle $C \in \mathcal{L}$* redundant *if it is homotopic to a one-sided cycle in $\mathcal{L}$ (cf. Figure 1).*

▶ **Definition 11.** *Let $x \in \mathbb{R}^{\mathcal{C}}$ be a solution to the LP* (1)*. We call it* structured *if the support of x is laminar and each connected component $\mathcal{L}$ of the support of x contains no redundant cycles.*

▶ **Lemma 12.** *Let $x \in \mathbb{R}^{\mathcal{C}}$ be a feasible solution to the LP* (1) *with laminar support $\mathcal{L}$. Then we can compute a structured solution $x' \in \mathbb{R}^{\mathcal{L}}$ to* (1) *with $\sum_{C\in\mathcal{L}} x_C = \sum_{C\in\mathcal{L}} x'_C$ in polynomial time in the size of $\mathcal{L}$.*

Proof. We can consider connected components of $\mathcal{L}$ separately, so assume w.l.o.g. that $E(\mathcal{L})$ is connected. Assume that x is not structured. Let $C \in \mathcal{L}$ be redundant with a side S that contains no other redundant cycles. In particular, S contains only one cycle $C' \neq C$ in $\mathcal{L}$. Since $E(\mathcal{L})$ is connected, $x(C) + x(C') \leq 1$. Thus, we can shift the LP value from C to C', i.e. set $x'(C') := x(C) + x(C')$ and $x'(C) := 0$, removing C from the support. This does not affect feasibility of the LP solution since it increases the LP value only on vertices strictly inside S, which are contained in no other cycles than C' due to minimality of S. See Figure 1.

Applying this reduction at most $|\mathcal{L}|$ times results in a solution as desired. ◀

Next, we analyze the ratio $\frac{x\left(\bigcup_{C\in\mathcal{F}^*} \mathcal{N}_{\mathcal{L}_x}(C)\right)}{|\mathcal{F}^*|}$ that we can achieve. In this section we only consider the case that $\mathcal{F}^*$ consists of a single one-sided cycle. We use the following Structure Lemma. The proof can be found in Section 4.

▶ **Lemma 13.** *Let $\mathcal{L}$ be a laminar family of cycles in a planar graph G, embedded in the sphere, such that $\mathcal{L}$ contains no redundant cycles. Let $\mathcal{L}_1$ be the set of one-sided cycles in $\mathcal{L}$. Then there is a multi-subset $M^* \subseteq V(G)$ with $|M^*| \leq 3|\mathcal{L}_1|$ such that for any $C \in \mathcal{L}$ we have $|M^* \cap V(C)| \geq |\mathcal{N}^1_{\mathcal{L}}(C) \setminus \{C\}|$.*

▶ **Lemma 14.** *Let x be a structured solution to the vertex-disjoint cycle packing LP. Let $\mathcal{L}$ be a connected component of the support of x and $\mathcal{L}_1$ the set of one-sided cycles in $\mathcal{L}$. Then*

$$\sum_{C\in\mathcal{L}_1} x(\mathcal{N}_{\mathcal{L}}(C) \setminus \{C\}) \leq 3|\mathcal{L}_1|.$$

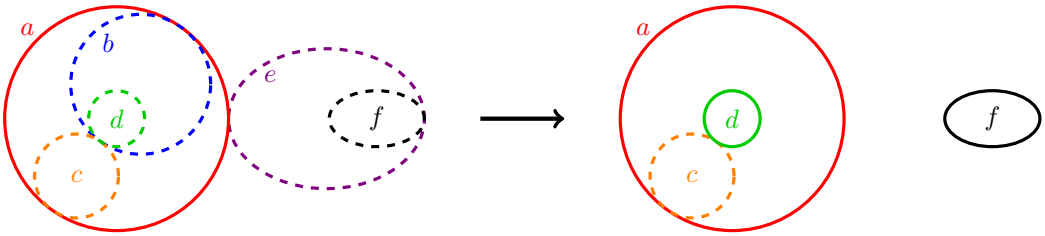


Figure 1 The left picture shows a possible laminar support of a feasible LP solution, consisting of six cycles. Dashed cycles have LP value $\frac{1}{3}$, while the others have LP value $\frac{2}{3}$. The cycles b and e are redundant because their interiors each contain only one other cycle; also a is redundant because its exterior only contains the one-sided side of f.
In the proof of Lemma 12 we would pick the interiors of b and e as S in the first and second step, increasing the LP value of d and f and removing b and e from the support. This yields a support as in the right image. The cycle a is still redundant, however in the laminar family given by its connected component it is one-sided and therefore not redundant. Thus, the solution is structured.

Proof. By the Structure Lemma 13, choose $M^* \subseteq V(G)$ with $|M^*| \leq 3|\mathcal{L}_1|$ such that for any $C \in \mathcal{L}$ we have $|M^* \cap V(C)| \geq |\mathcal{N}^1_{\mathcal{L}}(C) \setminus \{C\}|$. We get

$$\begin{aligned}
&\sum_{C \in \mathcal{L}_1} x(\mathcal{N}_{\mathcal{L}}(C) \setminus \{C\}) \\
&= \sum_{C \in \mathcal{L}} x(C) \cdot |\mathcal{N}^1_{\mathcal{L}}(C) \setminus \{C\}| \\
&\leq \sum_{C \in \mathcal{L}} x(C) \cdot |M^* \cap V(C)| \\
&\leq \sum_{v \in M^*} \sum_{C \in \mathcal{L}: v \in C} x(C) \;\leq\; |M^*| \;\leq\; 3|\mathcal{L}_1|.
\end{aligned}$$

◀

Since the LP value of each single cycle itself is bounded by 1 this immediately yields an upper bound of 4 for the integrality gap of (1):

▶ **Theorem 15.** *Let G be a planar graph, embedded in the sphere, and $\mathcal{C}$ an uncrossable family of cycles in G. Then there exists an integral solution to the vertex-disjoint cycle packing LP with at least $\frac{1}{4}$ the LP value. If $\mathcal{C}$ is given by a weight oracle we can compute such a solution in polynomial time.*

Proof. Let x be an optimum solution to the LP (1) with laminar support, as given by Lemma 8. By applying Lemma 12 we can assume x to be structured. Let $\mathcal{L}_x := \{C \in \mathcal{C} : x_C > 0\}$ be the (laminar) support of x. We proceed on each connected component of $\mathcal{L}_x$ individually, so we may assume $E(\mathcal{L}_x)$ to be connected.

Let $\mathcal{L}_1 \subseteq \mathcal{L}_x$ be the set of one-sided cycles. In each step of our greedy rounding algorithm we add a one-sided cycle C^* in $\mathcal{L}_x$ to our solution and set x on all cycles containing a vertex of C^* to 0, removing them from the support of x.

Lemma 14 implies

$$\sum_{C \in \mathcal{L}_1} x(\mathcal{N}_{\mathcal{L}_x}(C)) \leq 3|\mathcal{L}_1| + \sum_{C \in \mathcal{L}_1} x(C) \leq 4|\mathcal{L}_1|.$$

So there exists a one-sided cycle C^* where removing $\mathcal{N}_{\mathcal{L}_x}(C^*)$ decreases x by at most 4.

After the first iteration we again apply Lemma 12 and split the support of x into connected components. Iterating this procedure until $x = 0$ yields a solution as desired.

Note that Lemma 8 works in polynomial time if $\mathcal{C}$ has a weight oracle. Thus, also the size of $\mathcal{L}$ is polynomial in the size of G and Lemma 12 also works in polynomial time. In each step we can find C^* by picking the one-sided cycle in $\mathcal{L}_x$ minimizing $x(\mathcal{N}_{\mathcal{L}_x}(C^*))$. ◀

The greedy rounding algorithm described above also allows for adding several cycles at once to the solution. We will exploit this in Section 5 to decrease the upper bound for the integrality gap to below 3.5.

4 Proof of the Structure Lemma

The proof of the Structure Lemma 13 is fairly technical. In this version of the paper we only give a full proof for the case that no two-sided cycles exist and afterwards briefly discuss how to extend this proof to the general version of the Lemma. A detailed proof of Lemma 13 can be found in the full version of the paper.

First, let us consider the case that all cycles are one-sided. In this case we start by constructing another planar graph G' on vertex set $\mathcal{L}_1 = \mathcal{L}$ as follows:

For any vertex $v \in V(G)$ let $\mathcal{L}_v \subseteq \mathcal{L}$ be the set of cycles containing v. Since all cycles are one-sided, there is a natural cyclic order $\mathcal{L}_v = \{C_1 =: C_{k+1}, C_2, \dots, C_k\}$ on $\mathcal{L}_v$. Then we add for any $i = 1, \dots, k$ the edge $\{C_i, C_{i+1}\}$ with its obvious planar embedding to G'. Finally, we identify homotopic edges in G' (i.e. parallel edges bounding an area homeomorphic to the disk). See Figure 2.

Now from G' we can construct our multi-set M^*: For each $e = \{C_1, C_2\} \in E(G')$ we add an arbitrary vertex in $V(C_1) \cap V(C_2)$ to M^*; furthermore, for each vertex $v \in V(G)$ that is contained in $k > 3$ cycles we add $k - 3$ copies of v to M^*. Since in this case v lies inside a face of G' with exactly k edges on its boundary we can construct another planar graph G^* from G' by triangulating each such face F with $k - 3$ edges inside F (cf. Figure 2).

This yields a planar graph G^* on vertex set $\mathcal{L}_1$ with $|M^*|$ edges and no homotopic edges. Euler's formula implies $|M^*| = |E(G^*)| \leq 3|V(G^*| - 6 = 3|\mathcal{L}_1| - 6$.

Let now $C \in \mathcal{L}$ and $\mathcal{B} \subseteq \mathcal{N}^1_{\mathcal{L}}(C) \setminus \{C\}$ be the set of all neighbours of C that are not connected to C in G'. By construction of G' this means that for any vertex $v \in V(C)$ that is contained in k cycles at most $k - 3$ of them can be in $\mathcal{B}$. But we added $k - 3$ copies of v to M^*. This proves $|M^* \cap V(C)| \geq |\mathcal{B}| + |\delta_{G'}(C)| \geq |\mathcal{N}^1_{\mathcal{L}}(C) \setminus \{C\}|$.

Next, we have to consider also two-sided cycles. However, we do not know how to extend the relatively easy construction of G' and G^* to this more general case. Instead, we will use the notion of *incidences*:

▶ **Definition 16.** *Let $\mathcal{L}$ be a laminar family of cycles in a planar graph G, embedded in the sphere. Let $\mathcal{L}_1 \subseteq \mathcal{L}$ be the set of one-sided cycles. A* neighbour pair *is a pair $(\{C, N\}, v)$ of a set of two cycles $C, N \in \mathcal{L}$ that are not homotopic and a vertex $v \in V(C) \cap V(N)$. We call two neighbour pairs $(\{C, N\}, v)$ and $(\{C, N\}, v')$* homotopic *if there exist v-v'-paths P in C and P' in N such that $P + P'$ bounds an area that contains all one-sided sides in $\mathcal{L}$.*

It is easy to see that homotopy defines an equivalence relation on neighbour pairs. An equivalence class of neighbour pairs for C and N is called an incidence *between C and N. The* vertex set *$V(I)$ of an incidence I between C and N is the set of all v with $(\{C, N\}, v) \in I$. We also denote I by $I = (\{C, N\}, V(I))$. For a cycle $C \in \mathcal{L}$ let $\mathcal{I}^1_{\mathcal{L}}(C)$ be the set of all incidences between C and one-sided cycles in $\mathcal{N}^1_{\mathcal{L}}(C)$.*

Let now I be an incidence between $C \in \mathcal{L}$ and $N \in \mathcal{N}_{\mathcal{L}}(C)$. Let S_C be a side of C and S_N a side of N such that S_C and S_N are disjoint. We call an incidence $I' = (\{C', N'\}, V(I'))$ a sub-incidence *of I if C' is inside S_C, N' is inside S_N and $V(I') \subseteq V(I)$. We call I*

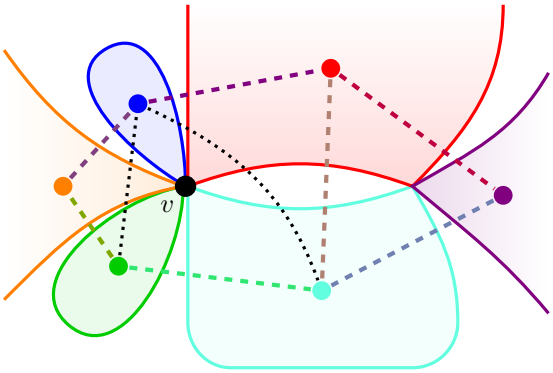


Figure 2 Example for the case that no two-sided cycles exist: The coloured cycles are the elements of $\mathcal{L}_1$. The vertices of G' are drawn as nodes inside the one-sided sides. The edges of G' are drawn as thick dashed lines. Since five cycles meet in the vertex v we would add v twice to M^*, in addition to the vertices in M^* corresponding to edges of G'. This is possible while keeping $|M^*| \leq 3|\mathcal{L}_1| - 6$ because we can triangulate the face of G' that v lies in with two additional edges; as indicated by the dotted lines.

minimal *if any sub-incidence of I is equal to I. We call I* crossing *if $V(I) = \{v\}$ for some $v \in V(G)$ and there exist cycles $C_1, C_2 \in \mathcal{L}$ that also contain v with sides S_1 and S_2 such that S_C, S_1, S_N, S_2 are all disjoint and are ordered in this way around v. Such incidences are also called* v-incidences. *If I is not crossing we call it* non-crossing.

Extending the idea of including the edges of G' in M^*, in order to prove Lemma 13 we will construct a set of incidences instead of a set of vertices.

▶ **Definition 17.** *Let $\mathcal{L}$ be a laminar family of cycles in a planar graph G, embedded in the sphere. Let $\mathcal{L}_1 \subseteq \mathcal{L}$ be the set of one-sided cycles. Let M be a multi-set of incidences in $\mathcal{L}$. We say that an element $I \in M$* hits *a cycle $C \in \mathcal{L}$ if $V(I) \subseteq V(C)$. We call a cycle $C \in \mathcal{L}$* M-good *if at least $|\mathcal{I}^1_{\mathcal{L}}(C)|$ elements of M hit C. We call M* good *if all cycles in $\mathcal{L}$ are M-good and $|M| \leq 3|\mathcal{L}_1| - 6$. Furthermore, we call M* structured *if the following properties hold:*

1. *M contains every non-crossing incidence between one-sided cycles.*
2. *For each $C \in \mathcal{L}_1$ and $v \in V(C)$ there exist at least as many v-incidences in M as there are v-incidences between C and $\mathcal{N}^1_{\mathcal{L}}(C)$ in $\mathcal{L}$.*

This notion of structured incidence sets is inspired from the construction of M^* in the case $\mathcal{L} = \mathcal{L}_1$: The edges in G' correspond to non-crossing incidences between one-sided cycles, which are included in M by property 1. Property 2 makes sure that vertices in which many one-sided cycles meet are included in M. In particular, we get the following as a direct consequence of the above definition:

▶ **Lemma 18.** *Let $\mathcal{L}$ be a laminar family of cycles in a planar graph G, embedded in the sphere. Let M be a structured set of incidences in $\mathcal{L}$. Then every one-sided cycle is M-good.*

Via an induction on the number of cycles in a laminar family $\mathcal{L}$ we can show existence of a good and structured set. The proof is a bit technical and can be found in the full version of the paper.

▶ **Lemma 19.** *Let $\mathcal{L}$ be a laminar family of at least two cycles in a planar graph G, embedded in the sphere. Let $\mathcal{L}_1$ be the set of one-sided cycles in $\mathcal{L}$. Then there exists a good and structured set M of incidences in $\mathcal{L}$.*

As a direct consequence of this lemma we get the Structure Lemma 13:

▶ **Lemma 13.** *Let $\mathcal{L}$ be a laminar family of cycles in a planar graph G, embedded in the sphere, such that $\mathcal{L}$ contains no redundant cycles. Let $\mathcal{L}_1$ be the set of one-sided cycles in $\mathcal{L}$. Then there is a multi-subset $M^* \subseteq V(G)$ with $|M^*| \leq 3|\mathcal{L}_1|$ such that for any $C \in \mathcal{L}$ we have $|M^* \cap V(C)| \geq |\mathcal{N}^1_{\mathcal{L}}(C) \setminus \{C\}|$.*

Proof. W.l.o.g. $|\mathcal{L}| \geq 2$. By Lemma 19 let M be a good set of incidences in $\mathcal{L}$. Let M^* arise from adding one element of $V(I)$ for every $I \in M$. In particular, $|M^*| = |M| \leq 3|\mathcal{L}_1| - 6$.

Let $C \in \mathcal{L}$. Since no side of C is redundant, it is not homotopic to any cycle in $\mathcal{L}_1$. Thus,

$$|\mathcal{N}^1_{\mathcal{L}}(C) \setminus \{C\}| \leq |\mathcal{I}^1_{\mathcal{L}}(C)| \leq |\{I \in M : V(I) \subseteq V(C)\}| \leq |M^* \cap V(C)|. \qquad \blacktriangleleft$$

5 Improving the bounds below 3.5

In this section we improve the bound on the integrality gaps of the LPs (1) and (2) to $\frac{13+3\sqrt{7}}{6} < 3.5$. We first only consider the vertex-disjoint cycle packing LP (1); an extension to the edge-disjoint case is given in Section 5.2.

5.1 The vertex-disjoint version

We still use Algorithm 1 from Section 3 for the improved approximation guarantee, but add further possibilities for the set $\mathcal{F}^*$ of cycles that are added to our solution during a single iteration. Note that the single cycle that we use in Theorem 15 already gives a good approximation guarantee if the average LP value on one-sided cycles is small. On the other hand, if the average LP value on one-sided cycles is large then we will find a large set of pairwise vertex-disjoint cycles with relatively small neighbourhood which we can take as $\mathcal{F}^*$. For analyzing the case of $\mathcal{F}^*$ containing more than one cycle we use the following Lemma:

▶ **Lemma 20.** *Let $\mathcal{L}$ be a laminar family of cycles in a planar graph G, embedded in the sphere. Let $\mathcal{L}_1$ be the set of one-sided cycles in $\mathcal{L}$. Let $\mathcal{F} \subseteq \mathcal{L}_1$. Then there is a set $M \subseteq V(\mathcal{F})$ with $|M| \leq |\mathcal{F}| + |\mathcal{L}_1|$ such that each cycle in $\mathcal{L}$ is either vertex-disjoint to all cycles in $\mathcal{F}$ or contains a vertex from M.*

Proof. W.l.o.g. $|\mathcal{L}| > 1$. We can also assume that there exists a point p_∞ on the sphere that lies neither on the embedding of vertices or edges in G nor in any one-sided side of a cycle in $\mathcal{L}_1$ (if this is not the case we can replace an arbitrary edge in G by two parallel edges, which does not affect the lemma's statement). For this proof, we call the side of a cycle $C \in \mathcal{L}$ that does not contain p_∞ the interior of C and say C contains a cycle $C' \in \mathcal{L}$, or $C' \subseteq C$, if the interior of C' is contained in the interior of C.

Let $\mathcal{B}_{\text{int}} \subseteq \mathcal{L}$ be the set of cycles C such that there is a cycle $C' \in \mathcal{F}$ with $C' \subseteq C$ and $V(C) \cap V(C') \neq \emptyset$. In particular, $\mathcal{F} \subseteq \mathcal{B}_{\text{int}}$. Let $f \colon \mathcal{B}_{\text{int}} \to \mathcal{F}$ such that each $C \in \mathcal{B}_{\text{int}}$ contains $f(C)$ and shares a vertex with $f(C)$. Then for any $C \in \mathcal{F}$ all cycles in $f^{-1}(C)$ must build a chain and therefore meet in some vertex $v_C \in V(C)$. Thus, $M_{\text{int}} := \{v_C : C \in \mathcal{F}\}$ hits all cycles in $\mathcal{B}_{\text{int}}$.

Let now $\mathcal{B}_{\text{ext}} \subseteq \mathcal{L} \setminus \mathcal{B}_{\text{int}}$ be the set of all cycles in $\mathcal{L} \setminus \mathcal{B}_{\text{int}}$ that share a vertex with any cycle in $\mathcal{F}$. We show by induction on $|\mathcal{L}_1|$ that we can hit all cycles in $\mathcal{B}_{\text{ext}}$ with some $M_{\text{ext}} \subseteq V(\mathcal{F})$ with $|M_{\text{ext}}| \leq |\mathcal{L}_1|$: For $|\mathcal{L}_1| = 1$ this is trivial. Otherwise, let $C_1 \in \mathcal{B}_{\text{ext}}$ be minimal w.r.t. $\subseteq$ and $C_2 \in \mathcal{F}$ with some vertex $v \in V(C_1) \cap V(C_2)$ (cf. Figure 3). Construct another laminar family $\mathcal{L}'$ by deleting all cycles inside C_1 and all cycles in $\mathcal{L} \setminus \mathcal{F}$ that contain v. Since C_1 contains some one-sided cycle, $\mathcal{L}'$ contains strictly less one-sided cycles and we

can use the induction hypothesis on $\mathcal{L}'$. Also the deletion of cycles inside C_1 does (except for C_1 itself) not change $\mathcal{B}_{\text{ext}}$ because $C_1 \notin \mathcal{B}_{\text{int}}$ and C_1 was minimal. Thus, the induction hypothesis gives us a set $M'_{\text{ext}} \subseteq V(\mathcal{F})$ that hits all cycles in $\mathcal{B}_{\text{ext}} \cap \mathcal{L}'$ with $|M'_{\text{ext}}| \leq |\mathcal{L}_1| - 1$, so $M_{\text{ext}} := M'_{\text{ext}} \cup \{v\}$ has the desired properties.

Setting $M := M_{\text{int}} \cup M_{\text{ext}}$ yields a set as desired in the Lemma. ◀

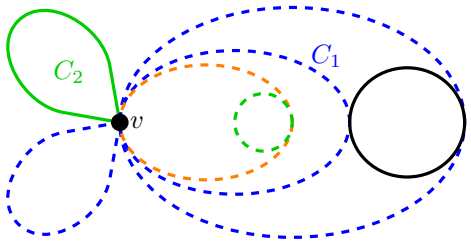


Figure 3 The cycles in $\mathcal{F}$ are drawn in green, the cycles in $\mathcal{B}_{\text{ext}}$ in blue. Note that the orange cycle is not in $\mathcal{B}_{\text{ext}}$ because it is in $\mathcal{B}_{\text{int}}$. The inside of C_1 contains no other cycles in $\mathcal{B}_{\text{ext}}$ and C_1 meets $C_2 \in \mathcal{F}$ in v. We then add v to our set M_{ext} and recurse on the laminar family $\mathcal{L}'$ that is constructed by removing all cycles inside C_1 and all cycles that contain v and are not in $\mathcal{F}$. These cycles are drawn dashed. This step decreases the number of one-sided cycles.

In the following, we assume $\mathcal{C}$ to be an uncrossable family of cycles in a graph G, embedded in the sphere. We further assume $x \in \mathbb{R}^{\mathcal{C}}$ to be a structured solution to the LP (1) with support $\mathcal{L}_x$. As in Theorem 15 we can assume $E(\mathcal{L}_x)$ to be connected. Let $\mathcal{L}_1 \subseteq \mathcal{L}_x$ be the set of one-sided cycles in $\mathcal{L}_x$. For each $0 \leq \alpha < 1$ we define $\mathcal{L}_1^{>\alpha} \subseteq \mathcal{L}_1$ to be the set of one-sided cycles with LP value $> \alpha$ and set $r_\alpha := \frac{|\mathcal{L}_1^{>\alpha}|}{|\mathcal{L}_1|}$.

We will now give two possible choices for $\mathcal{F}^*$ in Algorithm 1. The first possibility is to choose a set consisting of a single cycle in $\mathcal{L}_1$ as $\mathcal{F}^*$, as in Theorem 15. By Lemma 14 we directly get:

▶ **Lemma 21.** *There exists a cycle $C^* \in \mathcal{L}_1$ with $x(\mathcal{N}_{\mathcal{L}_x}(C^*)) \leq 3 + \frac{x(\mathcal{L}_1)}{|\mathcal{L}_1|}$.*

As a second possibility we will define a set $\mathcal{F}^*_\alpha$ for each $\frac{1}{4} \leq \alpha < \frac{1}{2}$. Given such an α, we consider the set $\mathcal{L}_1^{>\alpha}$. We know that at most three cycles in $\mathcal{L}_1^{>\alpha}$ can share a vertex. Let G' be the *conflict graph* for the cycles in $\mathcal{L}_1^{>\alpha}$; i.e. G' is the graph on vertex set $\mathcal{L}_1^{>\alpha}$ such that two cycles in $\mathcal{L}_1^{>\alpha}$ are connected by an edge in G' if and only if they share a vertex in G. Since each vertex is contained in at most three cycles of $\mathcal{L}_1^{>\alpha}$, G' can be constructed similarly to the graph G' in the proof of the Structure Lemma for $\mathcal{L} = \mathcal{L}_1$ (see Section 4). Thus, G' is planar. Furthermore, the cycles in $\mathcal{L}_1^{>1-\alpha} \subseteq \mathcal{L}_1^{>\alpha}$ correspond to isolated vertices in G'. By the Four Colour Theorem [1] we can partition $V(G') - \mathcal{L}_1^{>1-\alpha}$ into four stable sets. The largest of those, together with $\mathcal{L}_1^{>1-\alpha}$, yields a stable set in G' of size at least $|\mathcal{L}_1^{>1-\alpha}| + \frac{1}{4}(|\mathcal{L}_1^{>\alpha}| - |\mathcal{L}_1^{>1-\alpha}|)$. We let $\mathcal{F}^*_\alpha$ be the set of cycles in $\mathcal{L}_1^{>\alpha}$ corresponding to such a stable set in G'. By the definition of G' this means that the cycles in $\mathcal{F}^*_\alpha$ are pairwise vertex-disjoint.

▶ **Lemma 22.** *For any $\frac{1}{4} \leq \alpha < \frac{1}{2}$ we have*

$$\frac{x\left(\bigcup_{C' \in \mathcal{F}^*_\alpha} \mathcal{N}_{\mathcal{L}_x}(C')\right)}{|\mathcal{F}^*_\alpha|} \leq 1 + \frac{4(1-\alpha)}{r_\alpha + 3r_{1-\alpha}}.$$

Proof. Let $\frac{1}{4} \leq \alpha < \frac{1}{2}$. By Lemma 20 there is a set $M \subseteq V(\mathcal{F}^*_\alpha)$ with $|M| \leq |\mathcal{F}^*_\alpha| + |\mathcal{L}_1|$ such that each cycle in $\bigcup_{C' \in \mathcal{F}^*_\alpha} \mathcal{N}_{\mathcal{L}_x}(C')$ contains a vertex of M. Now

$$\begin{aligned} & x\left(\bigcup_{C' \in \mathcal{F}^*_\alpha} \mathcal{N}_{\mathcal{L}_x}(C')\right) \\ \leq & \sum_{v \in M} x(\{C \in \mathcal{L}_x \setminus \mathcal{F}^*_\alpha : v \in V(C)\}) + x(\mathcal{F}^*_\alpha) \\ \leq & |\mathcal{F}^*_\alpha| + (1-\alpha)|\mathcal{L}_1| \end{aligned}$$

holds, where the last inequality follows from the fact that there are $|\mathcal{F}^*_\alpha|$ vertices in M covering $\mathcal{F}^*_\alpha$, and for the other vertices we only have to count the LP value of cycles not in $\mathcal{F}^*_\alpha$. Inserting the bound $|M^*_\alpha| \geq |\mathcal{L}_1^{>1-\alpha}| + \frac{1}{4}(|\mathcal{L}_1^{>\alpha}| - |\mathcal{L}_1^{>1-\alpha}|)$ yields

$$\frac{x\left(\bigcup_{C' \in \mathcal{F}^*_\alpha} \mathcal{N}_{\mathcal{L}_x}(C')\right)}{|\mathcal{F}^*_\alpha|} \leq 1 + \frac{(1-\alpha)|\mathcal{L}_1|}{|\mathcal{F}^*_\alpha|} \leq 1 + \frac{4(1-\alpha)}{r_\alpha + 3r_{1-\alpha}}.$$ ◀

One of these possibilities for $\mathcal{F}^*$ will be sufficient to prove the desired upper bound of $\frac{13+3\sqrt{7}}{6}$ for the integrality gap:

▶ **Lemma 23.** *There exists a set $\mathcal{F}^* \subseteq \mathcal{L}_1$ with $\frac{x\left(\bigcup_{C' \in \mathcal{F}^*} \mathcal{N}_{\mathcal{L}_x}(C')\right)}{|\mathcal{F}^*|} \leq \frac{13+3\sqrt{7}}{6}$.*

Proof. Define $\beta := \frac{13+3\sqrt{7}}{6}$. We will either pick one of the sets $\mathcal{F}^*_\alpha$ for $\frac{1}{4} \leq \alpha < \frac{1}{2}$ from Lemma 22 or we will use $\mathcal{F}^* := \{C^*\}$ with the cycle C^* from Lemma 21. Assume none of these sets $\mathcal{F}^*$ fulfills the above inequality. Then Lemma 22 implies

$$\begin{aligned} & 1 + \frac{4(1-\alpha)}{r_\alpha + 3r_{1-\alpha}} > \beta \\ \Leftrightarrow \quad & r_\alpha + 3r_{1-\alpha} < \frac{4(1-\alpha)}{\beta - 1} \end{aligned}$$

for $\frac{1}{4} \leq \alpha < \frac{1}{2}$. Furthermore, we have

$$x(\mathcal{L}_1) = \sum_{C \in \mathcal{L}_1} \int_0^1 \mathbb{1}_{x(C) > \alpha} d\alpha = \int_0^1 \sum_{C \in \mathcal{L}_1} \mathbb{1}_{x(C) > \alpha} d\alpha = |\mathcal{L}_1| \int_0^1 r_\alpha d\alpha.$$

Thus, Lemma 21 implies

$$\beta < 3 + \int_0^1 r_\alpha d\alpha$$

By using the fact that the r_α are non-increasing, we get:

$$\begin{aligned} \beta < & \ 3 + \int_0^1 r_\alpha d\alpha \\ = & \ 3 + \int_0^{\frac{1}{3}} r_\alpha d\alpha + \int_{\frac{1}{3}}^{\frac{1}{2}} r_\alpha d\alpha + \int_{\frac{1}{2}}^1 r_\alpha d\alpha \\ \leq & \ 3 + \int_0^{\frac{1}{3}} r_\alpha d\alpha + \int_{\frac{1}{3}}^{\frac{1}{2}} r_\alpha d\alpha + 3\int_{\frac{1}{2}}^{\frac{2}{3}} r_\alpha d\alpha \\ = & \ 3 + \int_0^{\frac{1}{3}} r_\alpha d\alpha + \int_{\frac{1}{3}}^{\frac{1}{2}} r_\alpha + 3r_{1-\alpha} d\alpha \end{aligned}$$

$$\leq 3 + \int_0^{\frac{1}{3}} 1 d\alpha + \int_{\frac{1}{3}}^{\frac{1}{2}} \frac{4(1-\alpha)}{\beta - 1} d\alpha$$
$$= \frac{10}{3} + \frac{7}{18(\beta - 1)}.$$

This is a contradiction for $\beta = \frac{13+3\sqrt{7}}{6}$, which finishes the proof. ◀

As an immediate consequence we get our main theorem, similar to Theorem 15:

▶ **Theorem 24.** *Let G be a planar graph, embedded in the sphere, and $\mathcal{C}$ an uncrossable family of cycles in G. Then there exists an integral solution to the vertex-disjoint cycle packing LP with at least $\frac{6}{13+3\sqrt{7}}$ times the LP value. If $\mathcal{C}$ is given by a weight oracle we can compute such a solution in polynomial time.*

Proof. As in Theorem 15 we apply Algorithm 1. In contrast to the procedure in Theorem 15 however we use a set $\mathcal{F}^*$ as guaranteed by Lemma 23 in step 6 of the algorithm instead of a set consisting of just one one-sided cycle. Thus, in each step we increase the number of cycles with LP value 1 by $|\mathcal{F}^*|$ while decreasing the LP value on $\mathcal{L}_x$ by at most $\frac{13+3\sqrt{7}}{6}|\mathcal{F}^*|$. Therefore we arrive at an integral solution to the LP with at least $\frac{6}{13+3\sqrt{7}}$ times the LP value.

Note that a set $\mathcal{F}^*$ as in Lemma 23 can be found in time polynomial in $|\mathcal{L}_x|$: For the cycle guaranteed by Lemma 21 we can try all one-sided cycles. For the sets $\mathcal{F}^*_\alpha$ used in Lemma 22, note that there are only linearly many different sets $\mathcal{L}_1^{>\alpha}$ to consider. From those the sets $\mathcal{F}^*_\alpha$ are constructed by applying the Four Colour Theorem, which can also be done in polynomial time [23]. ◀

The upper bound on the integrality gap can still be slightly improved by also considering the sets $\mathcal{L}_1^{>\alpha}$ for $\alpha \geq \frac{1}{2}$ as candidates for $\mathcal{F}^*$. Using these candidates for large values of α in the inequality in Lemma 23 improves the bound in Lemma 23 and therefore also the bound in Theorem 24 slightly from $\frac{13+3\sqrt{7}}{6} \approx 3.4895$ to $\frac{20+\sqrt{130}}{9} \approx 3.4891$. We omit details here since the improvement is only marginal.

5.2 The edge-disjoint version

This section is dedicated to proving an edge-disjoint version of Theorem 24. One possibility to do this is to give edge-disjoint versions of Algorithm 1, the Structure Lemma 19 as well as Lemma 20 and then Lemma 23. All of this is possible analogous to the vertex-disjoint versions; however, we can also use a simple reduction by Schlomberg, Thiele and Vygen [24]. This does not generally reduce edge-disjoint cycle packing to vertex-disjoint cycle packing, but it does so for cycle packing in laminar cycle families.

▶ **Lemma 25** (similar to Schlomberg, Thiele, Vygen [24]). *Given a planar graph G, embedded in the sphere, and a laminar family $\mathcal{L}$ of cycles in G, we can compute in polynomial time a planar graph G' and a laminar family $\mathcal{L}'$ of cycles in G', together with a bijection $f\colon \mathcal{L} \to \mathcal{L}'$ such that for any $C_1, C_2 \in \mathcal{L}$ we have that $E(C_1) \cap E(C_2) = \emptyset$ if and only if $V(f(C_1)) \cap V(f(C_2)) = \emptyset$.*

Proof. Define $V(G') := E(G)$. For any path $P = e_1e_2$ of length two in a cycle $C \in \mathcal{L}$ we add the edge $e_C^P := \{e_1, e_2\}$ to $E(G')$. For any $C \in \mathcal{L}$ let $f(C)$ be the cycle consisting of all edges $e_C^P \in E(G')$ for any path P of length two inside C.

Since $\mathcal{L}$ is laminar, G' can be embedded planarly such that $\mathcal{L}' := \{f(C) : C \in \mathcal{L}\}$ defines a laminar family of cycles, as shown in Figure 4. By definition, all cycles in $\mathcal{L}'$ are edge-disjoint and two cycles $C_1, C_2 \in \mathcal{L}$ share an edge in G if and only if $f(C_1)$ and $f(C_2)$ share a vertex in G'. ◀

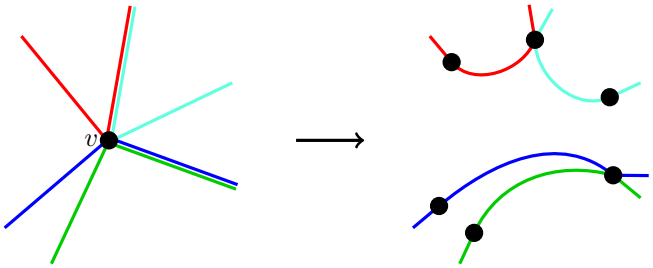


Figure 4 Example for the construction of G' and $\mathcal{L}'$: The left picture shows four cycles in $\mathcal{L}$ containing the vertex v. The six edges incident to v correspond to vertices of G', as shown in the right picture. Since $\mathcal{L}$ is laminar, the paths of length two in cycles of $\mathcal{L}$ can be embedded planarly as edges of G'. Cycles in $\mathcal{L}$ share an edge if and only if the corresponding cycles in $\mathcal{L}'$ share a vertex.

Using this reduction, we can easily extend Theorem 24 to the edge-disjoint case:

▶ **Theorem 26.** *Let G be a planar graph, embedded in the sphere, and $\mathcal{C}$ an uncrossable family of cycles in G. Then there exists an integral solution to the edge-disjoint cycle packing LP (2) a with at least $\frac{6}{13+3\sqrt{7}}$ times the LP value. If $\mathcal{C}$ is given by a weight oracle we can compute such a solution in polynomial time.*

Proof. We first apply Lemma 8 to get an optimum LP solution x to (2) with laminar support $\mathcal{L}$. We then apply Lemma 25 to get a laminar set $\mathcal{L}'$ of cycles in a planar graph G' with a bijection $f\colon \mathcal{L} \to \mathcal{L}'$ such that edge-disjointness in $\mathcal{L}$ translates to vertex-disjointness in $\mathcal{L}'$.

Note that $y \in \mathbb{R}^{\mathcal{L}'}$ with $y_{f(C)} = x_C$ for all $C \in \mathcal{L}$ defines a feasible solution to the LP (1) on $\mathcal{L}'$. Similar to Theorem 24 we can find an integral solution $\bar{y} \in \mathbb{R}^{\mathcal{L}'}$ to (1) on $\mathcal{L}'$ with $\bar{y}(\mathcal{L}') \geq \frac{6}{13+3\sqrt{7}} y(\mathcal{L}')$. Setting $\bar{x}_C := \bar{y}_{f(C)}$ for all $C \in \mathcal{L}$ then yields an integral solution to (2) on $\mathcal{L}$ with $\bar{x}(\mathcal{L}) = \bar{y}(\mathcal{L}') \geq \frac{6}{13+3\sqrt{7}} y(\mathcal{L}') = \frac{6}{13+3\sqrt{7}} x(\mathcal{L})$. ◀

References

1 K. Appel and W. Haken. A proof of the four color theorem. *Discrete Mathematics*, 16:179–180, 1976.

2 Piotr Berman and Grigory Yaroslavtsev. Primal-dual approximation algorithms for node-weighted network design in planar graphs. In *Approximation, Randomization, and Combinatorial Optimization (Proceedings of APPROX 2012)*, pages 50–60, 2012.

3 Wouter Cames van Batenburg, Louis Esperet, and Tobias Müller. Coloring Jordan regions and curves. *SIAM Journal on Discrete Mathematics*, 31(3):1670–1684, 2017.

4 Hong-Bin Chen, Hung-Lin Fu, and Chih-Huai Shih. Feedback vertex set on planar graphs. *Taiwanese Journal of Mathematics*, 16:2077–2082, 2012.

5 Norishige Chiba, Takao Nishizeki, Shigenobu Abe, and Takao Ozawa. A linear algorithm for embedding planar graphs using pq-trees. *Journal of Computer and System Sciences*, 30(1):54–76, 1985. doi:10.1016/0022-0000(85)90004-2.

6 Paul Erdős and Lajos Pósa. On independent circuits contained in a graph. *Canadian Journal of Mathematics*, 17:347–352, 1965.

7 Samuel Fiorini, Nadia Hardy, Bruce Reed, and Adrian Vetta. Approximate min–max relations for odd cycles in planar graphs. *Mathematical Programming*, 110(1):71–91, 2007.

8 Naveen Garg and Nikhil Kumar. Dual half-integrality for uncrossable cut cover and its application to maximum half-integral flow. In *28th Annual European Symposium on Algorithms (ESA 2020)*, pages 55:1–55:13, 2020.

9 Naveen Garg, Nikhil Kumar, and András Sebő. Integer plane multiflow maximisation: one-quarter-approximation and gaps. *Mathematical Programming*, 195:403–419, 2022.

10 Michel X. Goemans and David P. Williamson. Primal-dual approximation algorithms for feedback problems in planar graphs. *Combinatorica*, 18(1):37–59, 1998.

11 Alexander Göke, Jochen Koenemann, Matthias Mnich, and Hao Sun. Hitting weighted even cycles in planar graphs. *SIAM Journal on Discrete Mathematics*, 36(4):2830–2862, 2022. doi:10.1137/21M144894X.

12 Chien-Chung Huang, Mathieu Mari, Claire Mathieu, Kevin Schewior, and Jens Vygen. An approximation algorithm for fully planar edge-disjoint paths. *SIAM Journal on Discrete Mathematics*, 35:752–769, 2021.

13 Daniel Král', Jean-Sebastien Sereni, and Ladislav Stacho. Min-max relations for odd cycles in planar graphs. *SIAM Journal on Discrete Mathematics*, 26(3):884–895, 2012.

14 Daniel Král' and Heinz-Jürgen Voss. Edge-disjoint odd cycles in planar graphs. *Journal of Combinatorial Theory, Series B*, 90(1):107–120, 2004.

15 Claudio L. Lucchesi and Daniel H. Younger. A minimax theorem for directed graphs. *Journal of the London Mathematical Society II*, 17(3):369–374, 1978.

16 Jie Ma, Xingxing Yu, and Wenan Zang. Approximate min-max relations on plane graphs. *Journal of Combinatorial Optimization*, 26(1):127–134, 2013.

17 Matthias Middendorf and Frank Pfeiffer. On the complexity of the disjoint paths problem. *Combinatorica*, 13(1):97–107, 1993.

18 Dieter Rautenbach and Bruce Reed. The Erdős-Pósa property for odd cycles in highly connected graphs. *Combinatorica*, 21(2):267–278, 2001.

19 Dieter Rautenbach and Friedrich Regen. On packing shortest cycles in graphs. *Information Processing Letters*, 109(14):816–821, 2009.

20 Bruce Reed. Mangoes and blueberries. *Combinatorica*, 19(2):267–296, 1999.

21 Bruce Reed, Neil Robertson, Paul Seymour, and Robin Thomas. Packing directed circuits. *Combinatorica*, 16(4):535–554, 1996.

22 Bruce A. Reed and F. Bruce Shepherd. The Gallai–Younger conjecture for planar graphs. *Combinatorica*, 16(4):555–566, 1996.

23 Neil Robertson, Daniel P. Sanders, Paul Seymour, and Robin Thomas. Efficiently four-coloring planar graphs. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 571–575, New York, NY, USA, 1996. Association for Computing Machinery. doi:10.1145/237814.238005.

24 Niklas Schlomberg, Hanjo Thiele, and Jens Vygen. Packing cycles in planar and bounded-genus graphs. In *ACM-SIAM Symposium on Discrete Algorithms (SODA 2023)*, pages 2069–2086, 2023.

Limits of Sequential Local Algorithms on the Random k-XORSAT Problem

Kingsley Yung
The Chinese University of Hong Kong, Hong Kong, China

Abstract

The random k-XORSAT problem is a random constraint satisfaction problem of n Boolean variables and $m = rn$ clauses, which a random instance can be expressed as a $G\mathbb{F}(2)$ linear system of the form $Ax = b$, where A is a random $m \times n$ matrix with k ones per row, and b is a random vector. It is known that there exist two distinct thresholds $r_{core}(k) < r_{sat}(k)$ such that as $n \to \infty$ for $r < r_{sat}(k)$ the random instance has solutions with high probability, while for $r_{core} < r < r_{sat}(k)$ the solution space shatters into an exponential number of clusters. Sequential local algorithms are a natural class of algorithms which assign values to variables one by one iteratively. In each iteration, the algorithm runs some heuristics, called local rules, to decide the value assigned, based on the local neighborhood of the selected variables under the factor graph representation of the instance.

We prove that for any $r > r_{core}(k)$ the sequential local algorithms with certain local rules fail to solve the random k-XORSAT with high probability. They include (1) the algorithm using the Unit Clause Propagation as local rule for $k \geq 9$, and (2) the algorithms using any local rule that can calculate the exact marginal probabilities of variables in instances with factor graphs that are trees, for $k \geq 13$. The well-known Belief Propagation and Survey Propagation are included in (2). Meanwhile, the best known linear-time algorithm succeeds with high probability for $r < r_{core}(k)$. Our results support the intuition that $r_{core}(k)$ is the sharp threshold for the existence of a linear-time algorithm for random k-XORSAT.

Our approach is to apply the Overlap Gap Property OGP framework to the sub-instance induced by the core of the instance, instead of the whole instance. By doing so, the sequential local algorithms can be ruled out at density as low as $r_{core}(k)$, since the sub-instance exhibits OGP at much lower clause density, compared with the whole instance.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms

Keywords and phrases Random k-XORSAT, Sequential local algorithms, Average-case complexity, Phase transition, Overlap gap property

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.123

Category Track A: Algorithms, Complexity and Games

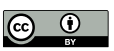
Related Version *Full Version*: `https://arxiv.org/abs/2404.17775`

Acknowledgements The author thanks Andrej Bogdanov for his guidance and many insightful discussions.

1 Introduction

1.1 Background

The k-XORSAT problem is a Boolean constraint satisfaction problem and closely related to the well-known k-SAT problem. An instance Φ of the k-XORSAT problem consists of m clauses in n Boolean variables. Each clause is a Boolean linear equation of k variables of the form $x_{j_1} \oplus x_{j_2} \oplus \cdots \oplus x_{j_k} = b_j$, where $\oplus$ is the modulo-2 addition. By convention, when we say an XORSAT instance, without the prefix "k", we mean the same except we do not require the clauses to have exactly k variables. An **assignment** σ to the n variables is a mapping from the set $\{x_i : i \in [n]\}$ of all n variables to the set $\{0, 1\}$ of the two Boolean values. By abusing



51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 123; pp. 123:1–123:20


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the notation, we can write it as the Boolean vector $\sigma = (\sigma(x_1), \sigma(x_2), \cdots, \sigma(x_n)) \in \{0,1\}^n$ containing the assigned values. The **distance** $d(\sigma, \sigma')$ between any two assignments σ and σ' is defined to be the Hamming distance $d(\sigma, \sigma') = \sum_{i=1}^{n} \mathbb{1}(\sigma(x_i) \neq \sigma'(x_i))$. A clause is **satisfied by** an assignment if the equation of the clause holds when the variables are replaced by the corresponding assigned values, and an instance of the k-XORSAT problem is **satisfied by** an assignment if all its clauses are satisfied by the assignment. An instance is **satisfiable** if it has at least one satisfying assignment, or **unsatisfiable** if it does not have any satisfying assignment. The assignment σ that satisfies the instance Φ is called a **solution** for the instance. The set of all satisfying solutions for the instance Φ is called the **solution space** of the instance, denoted by $\mathcal{S}(\Phi)$. We are interested in the complexity of finding a solution.

Since each clause is just a Boolean linear equation, an instance Φ can be viewed as a Boolean linear system $Ax = b$, where $A \in \{0,1\}^{m \times n}$ is an $m \times n$ Boolean matrix and $b \in \{0,1\}^n$ is a vector of length n. Note that each row in A contains exactly k ones, since each clause has exactly k variables. We can see that finding solutions for a k-XORSAT instance is equivalent to solving a Boolean linear system, and the solution space $\mathcal{S}(\Phi)$ is an affine space inside $\{0,1\}^n$. By abusing the notation, we can simply write $\Phi = (A, b)$, and the terms "clause" and "equation" are interchangeable here.

We are particularly interested in random instances of the k-XORSAT problem. In a random instance $\mathbf{\Phi}$, each clause is drawn over all $2\binom{n}{k}$ possibilities, independently. In particular, the left-hand side of the equation is the modulo-2 sum of k variables chosen uniformly from $\binom{n}{k}$ possibilities, and the right-hand side is either 0 or 1 with even probabilities. Therefore, a random instance $\mathbf{\Phi}$ of the k-XORSAT problem is drawn uniformly from the ensemble $\mathbf{\Phi}_k(n, m)$ of all possible instances of the k-XORSAT problem with n variables and m clauses, each clause containing exactly k variables, and we denote this by $\mathbf{\Phi} \sim \mathbf{\Phi}_k(n, m)$. We focus on the regime in which the number of variables n goes to the infinity and the number of clauses m is proportional to the number of variables n, that is, $m = rn$, where r is a constant independent of n and called the *clause density*.

Since a k-XORSAT instance can be represented by a system of linear equations in $G\mathbb{F}(2)$, given an instance, some standard linear algebra methods such as the Gaussian elimination can determine whether the instance is satisfiable, find a solution, and even count the total number of solutions, in polynomial time. However, beyond this particular algebraic structure, some variants of the k-XORSAT problem is hard to solve. Achlioptas and Molloy mentioned in their paper [6] that random instances of the k-XORSAT problem seems to be extremely difficult for both generic CSP solvers and SAT solvers, which do not take the algebraic structure into account. Guidetti and Young [22] suggested that the random k-XORSAT is the most difficult for random walk type algorithms such as WalkSAT, among many random CSPs. The difficulty of solving random k-XORSAT instances becomes more apparent when we only consider linear-time algorithms as efficient algorithms, since we do not have linear-time algebraic method to solve a linear system in general.

Many studies suggest that the difficulties of solving random CSPs are related to the *phase transition* of the solution spaces when the clause density r grows. (We will have more detailed discussion in Section 1.3.) Pittel and Sorkin [34] obtained the sharp *satisfiability threshold* $r_{sat}(k)$ of random k-XORSAT, for general $k \geq 3$: The random k-XORSAT instance is satisfiable w.h.p. when $r < r_{sat}(k)$, and it is unsatisfiable w.h.p. when $r > r_{sat}(k)$. (We say an event $\mathcal{E}_n$, depending on a number n, occurs **with high probability**, or shortened to **w.h.p.**, if the probability of the event $\mathcal{E}_n$ occurring converges to 1 as n goes to the infinity, that is, $\lim_{n\to\infty} \Pr[\mathcal{E}_n] = 1$.) Furthermore, Ibrahimi, Kanoria, Kraning and Montanari [25] obtained the sharp *clustering threshold* $r_{core}(k)$, which is less than $r_{sat}(k)$, of random k-

XORSAT for $k \geq 3$. When $r < r_{core}(k)$, w.h.p. the solution space of a random k-XORSAT instance is "well-connected". When $r_{core}(k) < r < r_{sat}(k)$, w.h.p. the solution space of a random k-XORSAT instance shatters into an exponential number of "well-separated clusters". In [25], They also provided a linear-time algorithm that can solve a random k-XORSAT instance w.h.p. for $r < r_{core}(k)$. On the other hand, no algorithm is known to be able to find a solution for a random k-XORSAT instance with non-vanishing probability in linear time, for $r_{core}(k) < r < r_{sat}(k)$ in which solutions exist with high probability.

In this work, we consider a natural class of algorithms, called *sequential local algorithms*. A sequential algorithm selects an unassigned variable randomly and assigns a value to it, iteratively, until every variable has an assigned value. In each iteration of the algorithm, to decide the assigned value, the algorithm runs some heuristic called *local rules* which return a value $p \in [0, 1]$, and decide the assigned value to be either 0 or 1 randomly, according to the Bernoulli distribution with parameter p. Ideally, if in each iteration the local rule can calculate the exact marginal probability of the selected variable over a randomly chosen solution for the instance conditioned on fixing all previously selected variables to their assigned values, the algorithm should be able to find a solution. However, we restrict the ability of the local rules by only providing *local* structure to the local rules. To explain the meaning of "local", we first construct a graphical representation for the k-XORSAT instances: the *factor graph*. The *factor graph* G of a k-XORSAT instance Φ is constructed in the following way: (1) each variable is represented by a *variable node*; (2) each equation is represented by an *equation node*; (3) add an undirected edge (v, e) if the variable v is involved in the equation e. Note that since there is a one-to-one correspondence between variables (equations) and variable nodes (equation nodes), in this paper, the terms *variables (equations)* and *variable nodes (equation nodes)* are interchangeable. The *distance* between any two nodes is the number of edges in the shortest path connecting the two nodes. For any integer $R \geq 0$, the *local neighborhood* $B_G(v, R)$ with radius R of a node v is the subgraph of G induced by all nodes with distances less than or equal to R from the node v. By "local" in the name of the algorithms, it means the local rules only takes the local neighborhood of the selected variable, of radius R, as its input.

The actual implementation of a sequential local algorithm depends on the choice for the local rules. To emphasize the choices for the local rules of the algorithms, the sequential local algorithm with the given local rule τ is called the *τ-decimation algorithm* $\texttt{DEC}_\tau$. The formal definitions of the sequential local algorithms, as well as the τ-decimation algorithms, will be given in Section 1.4. Note that if the local rule τ takes constant time, then the τ-decimation algorithm also takes linear time.

We introduce a notion of *freeness* to the sequential local algorithms. For any iteration in which the local rule returns $1/2$, we call it a *free step*. Intuitively, a free step means the local rule cannot obtain useful information from the local structure and let the algorithms make a random guess for the assigned value. We say a τ-decimation algorithm is *δ-free* if w.h.p. it has at least δn free steps. Moreover, we say a τ-decimation algorithm is *strictly δ-free* if it is δ'-free for some $\delta' > \delta$. If the τ-decimation algorithm is δ-free with large $\delta > 0$, it means the algorithm makes a lot of random guess on the assigned values, and it is likely that the local rule τ cannot extract useful information from the local structure to guide the algorithm. This leads to our contribution described in the next section.

1.2 Main contribution

The main contribution of this work consists of two parts. The first part is to show that as $n \to \infty$ if the τ-decimation algorithm is strictly $2\mu(k, r)$-free then w.h.p. it fails to find a solution for the random k-XORSAT instance, when the clause density r is beyond the

clustering threshold $r_{core}(k)$ but below the satisfiability threshold $r_{sat}(k)$. This can be formally written as Theorem 1. The value $\mu(k,r)$ given in Theorem 1 is an upper bound of the number of variables removed from the instance in order to obtain the sub-instance called *core instance*, which is crucial in our proof. We will discuss its meaning in detail in Section 2.

▶ **Theorem 1.** *For any $k \geq 3$ and $r \in (r_{core}(k), r_{sat}(k))$, if the τ-decimation algorithm $\mathtt{DEC}_\tau$ is strictly $2\mu(k,r)$-free on the random k-XORSAT instance $\mathbf{\Phi} \sim \mathbf{\Phi}_k(n, rn)$, then w.h.p. the output assignment $\mathtt{DEC}_\tau(\mathbf{\Phi})$ from the algorithm $\mathtt{DEC}_\tau$ on input $\mathbf{\Phi}$ is not in the solution space $\mathcal{S}(\mathbf{\Phi})$, that is,*

$$\lim_{n\to\infty} \Pr\left[\, \mathtt{DEC}_\tau(\mathbf{\Phi}) \in \mathcal{S}(\mathbf{\Phi}) \,\right] = 0,$$

where Q is the largest solution of the fixed point equation $Q = 1 - \exp(-krQ^{k-1})$, and $\mu(k,r)$ is the real-valued function given by $\mu(k,r) = \exp(-krQ^{k-1}) + krQ^{k-1}\exp(-krQ^{k-1})$.

Note. This theorem can also be applied to the *non-sequential local algorithms* in which the algorithms run their local rules and decide the assigned value for each variable, *in parallel*, without depending on the other assigned values. We will briefly discuss the reason at the end of Section 1.6 where we discuss the proof technique we used.

The best known linear algorithm of finding a solution for the random k-XORSAT instance succeeds w.h.p., for $k \geq 3$ and $r < r_{core}(k)$ [25]. That means these sequential local algorithms do not outperform the best known linear algorithm. Note that $r_{core}(k)$ is where the best known linear algorithm succeeds up to, and where the sequential local algorithms starts failing. These support the intuition that $r_{core}(k)$ is the sharp threshold of the existence of a linear-time algorithm for random k-XORSAT problem.

The second part of our contribution is to verify that the "freeness" condition in Theorem 1 is satisfied by the τ-decimation algorithm with certain local rules τ. One of them is the simplest local rule, the Unit Clause Propagation `UC`, which tries to satisfy the unit clause on the selected variable if exists, or make random guess otherwise. By using the Wormald's method of differential equations to count the number of free steps run by `UC`-decimation algorithm $\mathtt{DEC}_{\mathtt{UC}}$, we can show that it is strictly $2\mu(k,r)$-free on the random k-XORSAT instance $\mathbf{\Phi}$ for $k \geq 9$, which leads to the following theorem.

▶ **Theorem 2.** *For any $k \geq 9$, $r \in (r_{core}(k), r_{sat}(k))$, given a random k-XORSAT instance $\mathbf{\Phi} \sim \mathbf{\Phi}_k(n, rn)$, we denote by $\mathtt{DEC}_{\mathtt{UC}}(\mathbf{\Phi})$ the output assignment from the `UC`-decimation algorithm $\mathtt{DEC}_{\mathtt{UC}}$ on input $\mathbf{\Phi}$. Then, we have $\lim_{n\to\infty} \Pr\left[\, \mathtt{DEC}_{\mathtt{UC}}(\mathbf{\Phi}) \in \mathcal{S}(\mathbf{\Phi}) \,\right] = 0$.*

In each iteration, the role of the local rules is to calculate the marginal probability of the selected variable in the instance conditioned on fixing all previously selected variables to their assigned values. Belief Propagation `BP` and Survey Propagation `SP` are surprisingly good at approximating marginal probabilities of variables over randomly chosen solutions in many random constraint satisfaction problems empirically [29, 21, 7, 36, 27]. In particular, it is well-known that they can calculate the exact marginal probabilities of variables when the underlying factor graph is a tree, which is proved analytically. If Belief Propagation `BP` and Survey Propagation `SP` are used as the local rule τ, it is natural to expect that the τ-decimation algorithm can find a solution. However, we prove that even the local rule τ can give the exact marginal probabilities of variables over a randomly chosen solution for any instance whose factor graph is a tree, the τ-decimation algorithm still cannot find a solution w.h.p. for $k \geq 13$. We know that w.h.p. the local neighborhood of the factor graph of the random k-XORSAT instance is a tree. Therefore, running `BP` and `SP` on the

local neighborhood actually gives the exact marginal probabilities of the selected variables, with respect to the sub-instance induced by the local neighborhood. This implies that both BP-decimation algorithm $\texttt{DEC}_{\texttt{BP}}$ and SP-decimation algorithm $\texttt{DEC}_{\texttt{SP}}$ fail to find a solution w.h.p. for $k \geq 13$.

▶ **Theorem 3.** *For any $k \geq 13$, $r \in (r_{core}(k), r_{sat}(k))$, given a random k-XORSAT instance $\mathbf{\Phi} \sim \mathbf{\Phi}_k(n, rn)$, denote by $\texttt{DEC}_\tau(\mathbf{\Phi})$ the output assignment from the τ-decimation algorithm $\texttt{DEC}_{\texttt{UC}}$ on input $\mathbf{\Phi}$. Assume the local rule τ outputs the exact marginal probability of a selected variable for any instance whose factor graph is a tree. Then, we have* $\lim_{n\to\infty} \Pr\left[\,\texttt{DEC}_\tau(\mathbf{\Phi}) \in \mathcal{S}(\mathbf{\Phi})\,\right] = 0$.

To prove Theorem 2 and Theorem 3, we only need to calculate the number of free steps in $\texttt{DEC}_{\texttt{UC}}$ and the number of free steps in $\texttt{DEC}_\tau$ with the assumption on τ described in Theorem 3. If the number of free steps is strictly greater than $2\mu(k, r)n$ with high probability, we know that they are strictly $2\mu(k, r)$-free, and the results follow immediately by applying Theorem 1. Similarly, to obtain the same results for other τ-decimation algorithms, all we need to do is to calculate the number of free steps for those algorithms. If they are strictly $2\mu(k, r)$-free, we can obtain the same results by applying Theorem 1. Note that, due to certain limitations of our calculation, our results are limited to $k \geq 9$ in Theorem 2 and $k \geq 13$ in Theorem 3. We believe that the results hold for general $k \geq 3$, and can be proved by improving some subtle calculation in our argument.

Although we only show a few of implementations of the sequential local algorithms, we believe that the results are general across many sequential local algorithms with different local rules due to Theorem 3. In the framework of sequential local algorithms, the role of the local rules is to approximate the marginal probabilities of the selected variables over a random solution for the instance induced by the local neighborhood centered at the selected variables. Therefore, we believe that for any local rule that can make a good approximation on the marginals, it shall give similar results as Theorem 3. (Note that a more general definition of "δ-free" may be useful, for example, we can say a τ-decimation algorithm is (δ, ϵ)-*free* if we have $|p - 1/2| < \epsilon$, where p is the value returned by the local rule, for at least δn iterations.)

It is worth to mention the differences between these implementations of the sequential local algorithms and their well-known variants. Firstly, the UC-decimation algorithm is slightly different from the well-known Unit Clause algorithm. Under the framework of sequential local algorithm, the variables are selected in a random order. However, in the well-studied Unit Clause algorithm the variables in unit clauses are selected first [5]. The difference in the variable order could be crucial to the effectiveness of the Unit Clause algorithm. Secondly, the BP-decimation algorithm and the SP-decimation algorithm are slightly different from the Belief Propagation Guided Decimation Algorithm [12] and Survey Propagation Decimation Algorithm [8, 7, 28]. In the framework of sequential local algorithms, we only provide the local neighborhood to BP and SP. It is equivalent to bounding the number of messages passed in each decimation step by the constant $R \geq 0$ in BP-guided Decimation Algorithm and SP-guided Decimation Algorithm. It is in contrast to many empirical studies of BP-guided Decimation Algorithm and SP-guided Decimation Algorithm which allow the message passing iteration to continue until it converges.

Moreover, this work provides a new variant of the *overlap gap property method*, which was originally introduced by Gamarnik and Sudan [20]. Instead of considering the overlap gap property of the whole instance, we utilize that property of a sub-instance of the random k-XORSAT instance. In particular, the proof of this work is inspired by [20], which uses the overlap gap property method to rule out the class of balanced sequential local algorithms

from being able to solve random NAE-k-SAT problem when the clause density is close to the satisfiability threshold. Instead of directly applying the method on the whole instance, we focus on the sub-instance induced by the 2-core of the factor graph of the instance. This modification help us obtain tight bounds of algorithmic threshold unlike [20]. If we apply the original overlap gap property method and use the first moment method to obtain the property, we are able to show that the sequential local algorithms fail to solve the random k-XORSAT problem when the clause density is lower than a certain threshold $r_1(k)$. However, that threshold $r_1(k)$ is much higher than the $r_{core}(k)$. It only tells us that the algorithms fails when the density is very close to the satisfiability threshold $r_{sat}(k)$. With our modification, we can lower that threshold to exactly $r_{core}(k)$, namely, the algorithms fail in finding a solution when the clause density is as low as the clustering threshold. This opens a new possibility to improve other results which use the overlap gap property method on other random constraint satisfaction problems.

1.3 Phase transition of random k-XORSAT

Many random ensembles of constraint satisfaction problems CSPs such as random k-SAT and random NAE-k-SAT are closely related to the random k-XORSAT. For example, the well-known random k-SAT is analogous to the random k-XORSAT, in the sense that we can obtain a k-XORSAT instance from a k-SAT instance by replacing OR operators with XOR operators. We are particularly interested in the existences of some sharp thresholds on the clause density r in which the behaviors of a random instance changes sharply when the clause density r grows and passes through those thresholds. The following three thresholds are particularly of interest.

1. The *satisfiability threshold* separates the regime where w.h.p. the random instance is satisfiable and the regime where w.h.p. it is unsatisfiable.
2. The *clustering threshold* separates the regime where w.h.p. the solution space can be partitioned into well-separated subsets, each containing an exponential small fraction of solutions, and the regime where w.h.p. the solution space cannot be partitioned in this way.
3. The *algorithmic threshold* separates the regime where we have an efficient algorithm that can find a solution for a satisfiable random instance with non-vanishing probability, and the regime where no such algorithm exists.

Many random constraint satisfaction problems such as random k-SAT, random NAE-k-SAT and random graph coloring share the following phenomena related to these thresholds [3].

- There is an upper bound of the (conjectured) satisfiability threshold.
- There is a lower bound of the (conjectured) satisfiability threshold, from the non-constructive proof, and the lower bound is essentially tight.
- There are some polynomial time algorithms that can find a solution when the density is relatively low, but no polynomial time algorithm is known to succeed when the density is close to the satisfiability threshold. This leads to a conjectured algorithmic threshold, which is asymptotically below the (conjectured) satisfiability threshold.
- The clustering phenomenon takes place when the density is greater than a (conjectured) clustering threshold, and this threshold is close to or even asymptotically equal to the algorithmic threshold.

It is worth to mention that not every random constraint satisfaction problems share this set of phenomena. The most notable example is *symmetric binary perceptron* (SBP). Its satisfiability threshold $\alpha_{sat}(k) > 0$ was established by [33, 2]. They also showed that SBP exhibits clustering property and almost all clusters are singletons, for clause density $\alpha > 0$.

On the other hand, [1] gave a polynomial-time algorithm that can find solutions w.h.p., for low clause density. Therefore, there is a regime of low clause density in which SBP exhibits clustering property, and it is solvable in polynomial time, simultaneously. Its clustering phenomenon does not cause the hardness.

Being analogous to the random k-SAT problem, the random k-XORSAT problem shares those phenomena with other random constraint satisfaction problems. However, the story is slightly different in the random k-XORSAT problem. Since the k-XORSAT instances are equivalent to Boolean linear systems, their solution spaces are simply some affine subspaces in the Hamming hypercube $\{0,1\}^n$. Because of their algebraic structures, we are able to obtain the existence and the (n-independent) value of the satisfiability threshold of the random k-XORSAT problem. Dubois and Mandler [14] proved that there exists an n-independent satisfiability threshold $r_{sat}(k)$ for $k = 3$, and determined the exact value of the sharp threshold by the second moment argument. Pittel and Sorkin [34] further extended it for general $k \geq 3$. An independent work on cuckoo hashing from [13] also included an argument of the k-XORSAT satisfiability threshold for general $k \geq 3$. Those proofs consider the 2-core of the hypergraph associated with the random k-XORSAT instance. (In graph theory, a *k-core* of a (hyper)graph is a maximal subgraph in which all vertices have degree at least k.) Based on the 2-core, we can construct a sub-instance, called *2-core instance* or simply *core instance* of the original instance. One can prove that the original instance is satisfiable if and only if core instance is satisfiable. [10, 30] studied the core instance and determined the satisfiability threshold of the core instance, which can be converted to the satisfiability threshold of the random k-XORSAT instance.

Mézard, Ricci-Tersenghi and Zecchina [30] started the study of the clustering phenomenon of random k-XORSAT, and linked it to the existence of the non-empty 2-core instance. From [35, 32, 26], we know the non-empty 2-core of random hypergraphs suddenly emerges at a critical edge density $r_{core}(k)$. After that, Ibrahimi, Kanoria, Kraning and Montanari [25], and Achlioptas and Molloy [6] independently proved that there exists the clustering threshold $r_{clt}(k)$, which is equal to $r_{core}(k)$ and smaller than $r_{sat}(k)$, such that w.h.p. the solution space is a connected component for density $r < r_{core}(k)$, and w.h.p. the solution space shatters into exponentially many $\Theta(n)$-separated components for density $r > r_{core}(k)$, provided that we consider the solution space as a graph in which we add an edge between two solutions if their Hamming distance is $O(\log n)$.

As we mentioned before, the random k-XORSAT instance can be written as a random Boolean linear system, so it can be solved in polynomial time by using linear algebra method, regardless the clause density. For example, Gaussian elimination can solve it in $O(n^3)$ time. Since we do not have linear time algebraic method to solve linear system, we can still study the algorithmic phase transition if we only consider linear-time algorithms as efficient algorithms. In the proofs in [25], they provided an algorithm that can find a solution in linear time when $r < r_{core}(k)$, which implies that $r_{core}(k)$ is a lower bound of the (linear-time version of) algorithmic threshold $r_{alg}(k)$ of the random k-XORSAT problem. We conjecture that no algorithm can solve the random k-XORSAT problem in linear time with non-vanishing probability when $r > r_{core}(k)$, which implies $r_{core}(k)$ is an upper bound of $r_{alg}(k)$ and thus $r_{alg}(k) = r_{core}(k)$. This would lead to the intimate relation between the failure of linear time algorithms on random k-XORSAT and the clustering phenomenon of its solution space.

1.4 Sequential local algorithms

Sequential local algorithms are a class of algorithms parametrized by a local rule τ that specifies how values should be assigned to variables based on the "neighborhoods" of the variables. Given a local rule τ, the sequential local algorithm can be written as the following *τ-decimation algorithm*.

Given a fixed even number $R \geq 0$, we denote by $\mathcal{I}_R$ the set of all instances in which each of those instances has exactly one of its variables selected as *root*, and all nodes in its factor graph have distances from the root variable node at most R. A **local rule** is defined to be a function $\tau : \mathcal{I}_R \to [0,1] \in \mathbb{R}$, mapping from $\mathcal{I}_R$ to the interval $[0,1]$. Given an instance Φ, since the local neighborhood $B_\Phi(x^*, R)$ of a variable node x^* represents a sub-instance of Φ induced by all nodes having distance at most R from the root variable node x^*, we have $B_\Phi(x^*, R) \in \mathcal{I}_R$ and $\tau(B_\Phi(x^*, R))$ is well-defined. Then, the τ-decimation algorithm can be expressed as the followings.

Algorithm 1 τ-decimation algorithm.

Input: an instance of the k-XORSAT problem Φ,
an even number $R \geq 0$, and
a local rule $\tau : \mathcal{I}_R \to [0,1]$.
Set $\Phi_0 = \Phi$.
for $t = 0, ..., n-1$ **do**
Select an unassigned variable x^* from Φ_t, uniformly at random.
Set $\sigma(x^*) = \begin{cases} 1 & \text{with probability } \tau(B_{\Phi_t}(x^*, R)) \\ 0 & \text{with probability } 1 - \tau(B_{\Phi_t}(x^*, R)) \end{cases}$
Obtain Φ_{t+1} from Φ_t by:
(i) remove x^*;
(ii) for any clause having x^* before (i), add $\sigma(x^*)$ to its right-hand-side value;
(iii) remove all clauses that no longer contain any variable.
end for
Output: the assignment σ.

For any $t \in [n]$, if the value $\tau(B_{\Phi_t}(x^*, R))$ given by the local rule τ in the t-th iteration is $1/2$, then we call that iteration a **free step**. In a free step, the τ-decimation algorithm simply assigns a uniformly random Boolean value to the selected variable. On the contrary, if the value $\tau(B_{\Phi_t}(x^*, R))$ given by the local rule τ in the t-th iteration is either 0 or 1, then we call that iteration a **forced step**. In a forced step, the τ-decimation algorithm is forced to assign a particular value to the selected variable according to the value $\tau(B_{\Phi_t}(x^*, R))$. To simplify our discussion, we introduce the following definitions for those τ-decimation algorithms having certain numbers of free steps.

▶ **Definition 4.** *For any $\delta \in [0,1]$, we say a τ-decimation algorithm $\texttt{DEC}_\tau$ is **δ-free** on the random k-XORSAT instance $\mathbf{\Phi} \sim \mathbf{\Phi}_k(n, rn)$ if w.h.p the τ-decimation algorithm $\texttt{DEC}_\tau$ on input $\mathbf{\Phi}$ has at least δn free steps.*

▶ **Definition 5.** *For any $\delta \in [0,1]$, we say a τ-decimation algorithm $\texttt{DEC}_\tau$ is **strictly δ-free** on the random k-XORSAT instance $\mathbf{\Phi} \sim \mathbf{\Phi}_k(n, rn)$ if there exists $\delta' > \delta$ such that the τ-decimation algorithm $\texttt{DEC}_\tau$ is δ'-free on $\mathbf{\Phi}$.*

There are many choices for the local rules τ. The simplest one is the Unit Clause Propagation UC. In each iteration, after selecting the unassigned variable x^*, UC checks whether there exists a unit clause (clause with one variable) on the variable x^*. If yes, then UC sets $\tau(B_{\Phi_t}(x^*, R))$ to be the right-hand-side value of the unit clause, which can force the decimation algorithm to pick the suitable value to satisfy that clause. In this case, this iteration is a forced step. (If there are multiple unit clauses on the selected variable x^*, then only consider the one with the lowest index.) If there is no unit clause on the selected variable x^*, then UC sets $\tau(B_{\Phi_t}(x^*, R))$ to $1/2$, which let the algorithm choose the assigned value randomly. In this case, this iteration is a free step.

Algorithm 2 Unit Clause Propagation `UC`.

Input: the selected variable x^*, and its local neighborhood $B_{\Phi_t}(x^*, 2)$
if there exists any unit clause on the variable x^* **then**
    Pick the unit clause c on the variable x^* (with the lowest index if having multiple such clauses).
    Output: the right-hand-side value of the clause c.
else
    Output: $1/2$.
end if

1.5 Message passing algorithms

A new challenger to break the algorithmic threshold came out from statistical mechanics. In experiments [29, 21, 7, 36, 27], the *message passing algorithms* demonstrated their high efficiency on finding solutions of random k-SAT problem with the densities close to the satisfiability threshold. Those algorithms include *Belief Propagation Guided Decimation Algorithm* and *Survey Propagation Guided Decimation Algorithm*, which are based on the insightful but non-rigorous *cavity method* from statistical mechanics [7, 36]. Unfortunately, several analyses showed that they do not outperform the best known algorithms for some problems. Coja-Oghlan [12] showed that BP-guided Decimation fails to find solutions for random k-SAT w.h.p. for density above $\rho_0 2^k/k$ for a universal constant $\rho_0 > 0$, and thus does not outperform the best known algorithm from [11]. Hetterich [23] also gave the same conclusion for SP-guided Decimation by showing that it fails w.h.p. for density above $(1 + o_k(1))2^k \ln k/k$.

For random NAE-k-SAT, Gamarnik and Sudan [20] showed that the balanced sequential local algorithms fail to find solutions for density above $(1 + o_k(1))2^{k-1} \ln^2 k/k$ for sufficiently large k. This means the algorithms do not outperform the best known algorithm, Unit Clause algorithm, which can find solutions w.h.p. for density up to $\rho\, 2^{k-1}/k$ for some universal constant $\rho > 0$ for sufficiently large k [5]. The framework of balanced sequential local algorithms also covers BP-guided Decimation and SP-guided Decimation with the number of message passing iterations is bounded by $O((\ln \ln n)^{O(1)})$.

In our work, we obtain an analogous result. In Theorem 1, we show that w.h.p. strictly $2\mu(k, r)$-free sequential local algorithms fails to solve the random k-XORSAT problem when the clause density exceeds the clustering threshold. Then, in Theorem 3, we show that any sequential local algorithm with local rule that can compute the exact marginals are strictly $2\mu(k, r)$-free and thus fails to find a solution for random k-XORSAT problem. This theorem covers the sequential local algorithms with Belief Propagation `BP` and Survey Propagation `SP` as local rules.

1.6 Technique

The works from [3, 4] demonstrated the clustering phenomenon for several random CSPs, and conjectured that it could be an obstruction of solving those problems. [19, 20] and subsequent works leveraged a different notion of clustering, named *overlap gap property* (OGP) by [18], to link the clustering phenomenon to the hardness rigorously. Gamarnik gave a detailed survey on it [15].

This paper focuses on the vanilla version of the OGP. Given an instance Φ of the constraint satisfaction problem, we say it exhibits the overlap gap property with values $0 \le v_1 < v_2$ if every two solutions σ and σ' satisfy either $d(\sigma, \sigma') \le v_1$ or $d(\sigma, \sigma') \ge v_2$, where d is a

metric on its solution space. (We assume d is the Hamming distance throughout this paper.) Intuitively, it means every pair of solutions are either close to each other, or far from each other, and thus the solution space of the instance exhibits a topological discontinuity based on the proximity.

Now we illustrate how does the overlap gap property method. (Some details of the overlap gap property method is slightly different if we consider different variants of the OGP, but the overall idea of *topological barrier* stays the same.) Assume we have an algorithm $\mathcal{A}$ that takes a random instance $\mathbf{\Phi}$ as input and outputs an assignment σ for the instance. The output σ can be viewed as a random variable which depends on both the random instance $\mathbf{\Phi}$ and some internal random variables, represented by a random internal vector $\mathbf{I}$, of the algorithm. Let Φ and I be realizations of $\mathbf{\Phi}$ and $\mathbf{I}$ respectively, and denote the output assignment as $\sigma_0 = \sigma_{\Phi,I}$. We then re-randomize the components of the internal vector I *one-by-one*. After each re-randomizing a component of I, we run the algorithm again to generate a new output assignment. Then, we obtain a sequence of assignments $\sigma_0, \sigma_1, \cdots, \sigma_T$ for the instance Φ, where T is the number of components of the random vector I that we have re-randomized. Next, we show that the algorithm is *insensitive to its input* in the sense that when one of the components of the random internal vector $\mathbf{I}$ is re-randomized, the output assignment almost remains unchanged, In particular, $d(\sigma_i, \sigma_{i+1})$ is smaller than $v_2 - v_1$. We also show that the algorithm has certain *freeness* in the sense that when all components in the random internal are re-randomized the output assignment is expected to change a lot. In particular, $\mathbb{E}[d(\sigma_0, \sigma_T)]$ should be larger than v_2. These two properties together imply that the sequence of assignments cannot "jump" over the overlap gap, while two ends of the sequence probably lie in different clusters. Therefore, there should be an assignment σ_{T_0} that falls in the gap, namely, there exists $T_0 > 0$ such that $v_1 \leq d(\sigma_0, \sigma_{T_0}) \leq v_2$. If the probability that the algorithm successfully finds a solution is greater than some small value s_n slowly converging to 0, then there could be a very small probability that both σ_0 and σ_{T_0} are solutions of the instance Φ, with $v_1 \leq d(\sigma_0, \sigma_{T_0}) \leq v_2$. Even though this probability is very small, it still has the chance to violates the OGP of the instance. Then, by contradiction, we could conclude that the probability that the algorithm succeeds in finding a solution is smaller than $s_n = o(1)$, namely, w.h.p. the algorithm fails in finding a solution.

Instead of considering the overlap gap property of the entire instance $\mathbf{\Phi}$, we move our focus to the overlap gap property of a sub-instance of $\mathbf{\Phi}$. Indeed, the sub-instance we consider is the *2-core instance* $\mathbf{\Phi_c}$ induced by the 2-core of the factor graph representation of the random instance $\mathbf{\Phi}$. In [25, 6], they proved that the 2-core instance $\mathbf{\Phi_c}$ exhibits the overlap gap property with $v_1' = o(n)$ and $v_2' = \epsilon_k n$ for some constant $\epsilon_k > 0$ for clause density $r_{core}(k) < r < r_{sat}(k)$. We remove all variables not in the core instance from the sequence of assignments $\sigma_0, \sigma_1, \cdots, \sigma_T$ we obtained above, then it becomes a sequence of assignments $\sigma_0', \sigma_1', \cdots, \sigma_T'$ for the core instance. We also prove that the algorithm is *insensitive to its input* with respect to the core instance in the sense that $d(\sigma_i', \sigma_{i+1}') < v_2' - v_1'$, and has certain *freeness* so that $\mathbb{E}[d(\sigma_0', \sigma_T')] > v_2'$. By repeating the above argument of the overlap gap property method, we can conclude that w.h.p. the algorithm fails in find a solution.

Our proof can also be used for the non-sequential local algorithms. Since the local rule τ runs on the local neighborhood of each variable in parallel, the values assigned to variables do not depend on each other. Informally speaking, there is no long-range dependency among those assigned values. Therefore, re-randomizing one component of the internal vector I, say I_{i+1}, only affects the value $\sigma(x_{i+1})$ assigned to the corresponding variable x_{i+1}. So, we have $d(\sigma_i, \sigma_{i+1}) \leq 1 < v_2' - v_1'$. Hence, we can obtain the same result as Theorem 1 for non-sequential local algorithms with the same proof.

1.7 Related works

The vanilla version of OGP helps us rule out some large classes of algorithms on random CSPs for relatively high clause densities, but it is not sophisticated enough to close the *statistical-to-computational gap* in some cases such as random NAE-k-SAT discussed in [20] and random k-XORSAT discussed in this paper (more details in Section 2). There have been some recent works trying to improve the notion of OGP by developing different variants of OGP. The most notable one is *multi-OGP* [37, 9, 24, 17, 16], which succeeds in closing the statistical-to-computational gap in certain models. However, it is not clear about the relation between the clustering property and the multi-OGP.

2 Overlap Gap Property

We say that a k-XORSAT instance Φ exhibits the **overlap gap property** (or shortened as OGP) with the values $0 \le v_1 < v_2$ if for any two solutions $\sigma, \sigma' \in S(\Phi)$ we have either $d(\sigma,\sigma') \le v_1$ or $d(\sigma,\sigma') \ge v_2$. Informal speaking, any two solutions of the instance are either close to each other or far away from each other, and thus the solution space exhibits a topological discontinuity. Given a random k-XORSAT instance, we can prove that it exhibits the OGP when the clause density is greater than certain value, and obtain the following lemma.

▶ **Lemma 6.** *For any $k \ge 3$, there exists $r_1(k) > 0$ such that for $r > r_1(k)$ and any pair of solutions $\sigma, \sigma' \in S(\boldsymbol{\Phi})$ of the random k-XORSAT instance $\boldsymbol{\Phi} \sim \boldsymbol{\Phi}_n(k, rn)$, w.h.p. the distance $d(\sigma,\sigma')$ between the two solutions is either $\le u_1 n$ or $\ge u_2 n$ for some $0 \le u_1 < u_2$. In particular, the value of $r_1(k)$ is given by*

$$r_1(k) = \min_{0\le\alpha\le 1} \frac{1 + H(\alpha)}{2 - \log(1 + (1-2\alpha)^k)},$$

where H is the binary entropy function, that is, $H(x) = -x\log_2(x) - (1-x)\log_2(1-x)$.

Instead of considering the random k-XORSAT instance $\boldsymbol{\Phi}$ itself, we focus on the sub-instance, called the *core instance* (defined below), of the random k-XORSAT instance $\boldsymbol{\Phi}$, and show that core instance also exhibits the overlap gap property, even when the clause density is much lower. (See Table 1.)

Table 1 Compare $r_{core}(k)$ with $r_1(k)$ for different k. The numeric values in the table are rounded off to 6 decimal places.

k	3	4	5	6	7	8	9
$r_{core}(k)$	0.818470	0.772280	0.701780	0.637081	0.581775	0.534997	0.495255
$r_1(k)$	0.984516	0.943723	0.905812	0.874349	0.848314	0.826470	0.807862

We start from defining the **peeling algorithm** and the **core instances**. Given an XORSAT instance Φ, suppose there exists a variable x of degree 1, which means it is involved in exactly one clause e. We remove the variable x and the only clause c involving x, to obtain a modified instance Φ'. If we have a solution σ' for the modified instance Φ', we can always choose a suitable value for the variable x to satisfy the clause c, and extend the solution σ' to a solution σ for the original instance Φ. Similarly, we can also do the same thing if the variable x is of degree 0, since it does not involve in any equation, and we are free to choose any value for it. By doing this, solving the original instance Φ is reduced to solving the modified instance Φ'.

We can repeat this process until there is no variable of degree at most 1. This process is named the **peeling algorithm** on an instance as its input (Algorithm 3). We call the resultant instance the **2-core instance** (or simply the **core instance**) of the instance Φ, denoted by Φ_c. This name is borrowed from the graph theory. In graph theory, the *k-core* of a graph is the maximal subgraph with minimum degree at least k. It is known that the factor graph of the core instance Φ_c is exactly the maximal subgraph of the factor graph G_Φ of the instance Φ, with minimum variable degree at least 2.

Algorithm 3 Peeling algorithm.

Input: an instance Φ.
while There exists ≥ 1 variable of degree ≤ 1. **do**
    Select a variable x of degree ≤ 1.
    (Pick x with the lowest index if there are > 1 such variables.)
    Update Φ by removing the variable x_i and its only involved clause (if exists).
end while
Output: the resultant instance Φ.

Mézard and Montanari [31] gave a detailed description on the structure of the core instance. Reader can find more details about the core instance in their book. The following theorem is a short summary of some known facts about the core instances we needed in the paper.

▶ **Theorem 7.** *For any $k \geq 3$, there exists $r_{core}(k) > 0$ given by*

$$r_{core}(k) \equiv \sup\{r \in [0,1] : Q > 1 - e^{-krQ^{k-1}} \ \forall Q \in (0,1)\}$$

such that the factor graph G_c of the core instance $\mathbf{\Phi}_c$ of the random k-XORSAT instance $\mathbf{\Phi} \sim \mathbf{\Phi}_n(k, rn)$ have the following properties.

1. *For $r < r_{core}(k)$, w.h.p. the factor graph G_c of the core instance $\mathbf{\Phi}_c$ is an empty graph.*
2. *For $r > r_{core}(k)$, w.h.p. the factor graph G_c of the core instance $\mathbf{\Phi}_c$ have $V(k,r)n + o(n)$ variable nodes, where*

$$V(k,r) = 1 - \exp(-krQ^{k-1}) - krQ^{k-1}\exp(-krQ^{k-1})$$

and Q is the largest solution of the fixed point equation $Q = 1 - \exp(-krQ^{k-1})$. In particular, the fraction of variable nodes of degree l is between $\widehat{\Lambda}_l - \epsilon$ and $\widehat{\Lambda}_l + \epsilon$ with probability greater than $1 - e^{-\Theta(n)}$, where $\widehat{\Lambda}_0 = \widehat{\Lambda}_1 = 0$ and

$$\widehat{\Lambda}_l = \frac{1}{e^{krQ^{k-1}} - 1 - krQ^{k-1}} \frac{1}{l!} (krQ^{k-1})^l \quad \text{for } l \geq 2.$$

3. *Conditioning on the number of variable nodes $V(k,r)n + o(n)$ and the degree profile $\widehat{\Lambda}$, the factor graph G_c of the core instance $\mathbf{\Phi}_c$ is distributed according to the ensemble containing all possible factor graphs of k-XORSAT instances of $V(k,r)n + o(n)$ variables and variable degree distribution Λ.*

Theorem 7 shows that there exists a threshold $r_{core}(k)$ below the satisfiability threshold $r_{sat}(k)$ of random k-XORSAT problem. When the clause density r is below the threshold $r_{core}(k)$, w.h.p. the instance $\mathbf{\Phi}$ does not have a core instance. When the clause density r is above the threshold $r_{core}(k)$, w.h.p. the core instance $\mathbf{\Phi}_c$ emerges. In particular, the variable degree distribution is a Poisson distribution with mean krQ^{k-1} conditioning on $\Lambda_0 = \Lambda_1 = 0$. [31] also showed that the core instance exhibits the OGP.

▶ **Lemma 8.** *For $k \geq 3$ and $r_{core}(k) < r < r_{sat}(k)$, there exists $\epsilon(k,r) > 0$ such that w.h.p. the distance between any two solutions for the core instance $\mathbf{\Phi}_c$ of a random k-XORSAT instance $\mathbf{\Phi} \sim \mathbf{\Phi}_n(k, rn)$ is either $o(n)$ or greater than $\epsilon(k,r)n$.*

Now, we know that w.h.p. the core instance of a random k-XORSAT instance has the overlap gap property with the values $v_1 = o(n)$ and $v_2 = \epsilon(k,r)n$. With OGP, we can partition the solution space of the core instance into multiple groups, each called a **core cluster**, such that the distance between any pair of core solutions in the same core cluster is at most $o(n)$, and the distance between any pair of core solutions in different core clusters is at least $\epsilon(k,r)n$.

Suppose we have a k-XORSAT instance Φ. We first define a binary relation on the solution space of a core instance Φ_c by: for $\sigma_c, \sigma_c' \in \mathcal{S}(\Phi_c)$, we write $\sigma_c \simeq \sigma_c'$ if and only if $d(\sigma_c, \sigma_c') = o(n)$. It is easy to see it is an equivalence relation. Then, we can partition the solution space by the equivalence classes of $\simeq$. We can denote those equivalence classes by $\mathcal{S}_{c,1}, \mathcal{S}_{c,2}, ..., \mathcal{S}_{c,n_c}$. Thus, we have $\mathcal{S}_{c,1} \sqcup \mathcal{S}_{c,2} \sqcup ... \sqcup \mathcal{S}_{c,n_c} = \mathcal{S}(\Phi_c)$ where $\sqcup$ is the disjoint union. Then, we have

$$\begin{aligned} &d(\sigma_c, \sigma_c') = o(n) && \text{if } \ \sigma_c, \sigma_c' \in \mathcal{S}_{c,i}, \quad \text{and} \\ &d(\sigma_c, \sigma_c') \geq \epsilon(k,r)n && \text{if } \ \sigma_c \in \mathcal{S}_{c,i},\ \sigma_c' \in \mathcal{S}_{c,j} \text{ and } \mathcal{S}_{c,i} \neq \mathcal{S}_{c,j} \end{aligned}$$

Now we can partition the solution space $\mathcal{S}(\Phi)$ of the original instance Φ into clusters based on the partition of the solution space of core instance. We set

$$\mathcal{S}(\Phi) = \bigsqcup_{i=1}^{n_c} \mathcal{S}_i \quad \text{and} \quad \mathcal{S}_i = \{\sigma \in \mathcal{S}(\Phi) : \pi(\sigma) \in \mathcal{S}_{c,i}\} \quad \text{for } i = 1, 2, ..., n_c$$

where π is defined to be the **projection** mapping assignments for the instance Φ to assignments for the core instance Φ_c by removing all variables not in the core instance Φ_c. Each $\mathcal{S}_i$ is called a **cluster** in the solutions space $\mathcal{S}(\Phi)$. We can then prove that these clusters are well-separated from each other.

▶ **Lemma 9.** *Let $k \geq 3$ and $r_{core}(k) < r < r_{sat}(k)$. Suppose $\mathbf{\Phi} \sim \mathbf{\Phi}_n(k, rn)$ is a random k-XORSAT instance. Then, w.h.p. there exists a partition $\mathcal{S}(\mathbf{\Phi}) = \mathcal{S}_1 \sqcup \mathcal{S}_1 \sqcup ... \sqcup \mathcal{S}_{n_c}$ for the solutions space $S(\mathbf{\Phi})$ of the random instance $\mathbf{\Phi}$ such that the following statements hold.*

1. *If $\sigma, \sigma' \in \mathcal{S}_i$ for some $i \in [n_c]$, then we have $d(\sigma, \sigma') \leq \mu(k,r)n + o(n)$, where the real-valued function $\mu(k,r)$ is given by $\mu(k,r) = \exp(-krQ^{k-1}) + krQ^{k-1}\exp(-krQ^{k-1})$ and Q is the largest solution of the fixed point equation $Q = 1 - \exp(-krQ^{k-1})$.*
2. *If $\sigma \in \mathcal{S}_i, \sigma' \in \mathcal{S}_j$ and $\mathcal{S}_i \neq \mathcal{S}_j$ for some $i, j \in [n_c]$, then we have $d(\sigma, \sigma') \geq \epsilon(k,r)n$.*

Proof. Assume the instance $\mathbf{\Phi}$ has a non-empty core instance $\mathbf{\Phi}_c$, which exists with high probability according to Theorem 7. We also assume the core instance $\mathbf{\Phi}_c$ exhibits the OGP with $v_1 = o(n)$ and $v_2 = \epsilon(k,r)n$, which occurs with high probability according to Lemma 8. Let σ and σ' be two solutions of the random k-XORSAT instance $\mathbf{\Phi}$, and let $\sigma_c = \pi(\sigma)$ and $\sigma_c' = \pi(\sigma')$ be the projection of σ and σ' on the core solution space $\mathcal{S}(\mathbf{\Phi})$, respectively.

To prove the first part of the lemma, we assume that σ and σ' are in the same cluster, that is, $\sigma, \sigma' \in \mathcal{S}_i$ for some $i \in [n_c]$. By the definition of cluster, we have $d(\sigma_c, \sigma_c') = o(n)$. Therefore, $d(\sigma, \tau)$ is upper bounded by the number of variables not in the core instance, plus $o(n)$. By Theorem 7, the number of variables outside the core instance is given by $(1 - V(k,r))n + o(n)$. Hence, we have $d(\sigma, \tau) \leq (1 - V(k,r))n + o(n) = \left(\exp(-krQ^{k-1}) + krQ^{k-1}\exp(-krQ^{k-1})\right)n + o(n)$.

To prove the second part of the lemma, we assume that σ and τ are in the different clusters, that is, $\sigma \in \mathcal{S}_i$, $\sigma' \in \mathcal{S}_j$ and $\mathcal{S}_i \neq \mathcal{S}_j$ for some $i, j \in [n_c]$. By the definition of cluster and Lemma 8, we have $d(\sigma_c, \sigma_c') \geq \epsilon(k,r)n$ Therefore, we have $d(\sigma, \sigma') \geq d(\pi(\sigma), \pi(\sigma')) = d(\sigma_c, \sigma_c') \geq \epsilon(k,r)n$. ◀

3 Preparation of OGP method

In this section, we introduce some notions and obtain some preliminary results needed by the overlap gap property method to prove the main results.

3.1 Sequence of output assignments

The random k-XORSAT instance $\mathbf{\Phi}$ is a random variable, and the τ-decimation algorithm $\mathtt{DEC}_\tau$ is a randomized algorithm. Therefore, the assignment output by the τ-decimation algorithm $\mathtt{DEC}_\tau$ on input $\mathbf{\Phi}$ is also a random variable. The outcomes of the output assignment depend on the random instance $\mathbf{\Phi}$, the order of variables being chosen, and the value selection based on the output from the local rule τ. Now we introduce two random variables to explicitly represent the order of variables and the value selection so that we can have a more concrete language to discuss how the randomness from both the instance and the algorithm affects the output assignment. We adopt the notation from [20] in the following discussion.

The order of variables can be represented by a random vector $\mathbf{Z} = (\mathbf{Z}_1, \mathbf{Z}_2, \cdots, \mathbf{Z}_n)$ whose entries are n i.i.d. random variables with uniform distribution over the interval $[0,1] \subset \mathbb{R}$, independent of the random instance $\mathbf{\Phi}$. We call $\mathbf{Z}$ the **ordering vector** of the algorithm. For all $i \in [n]$, the variable x_i in the instance $\mathbf{\Phi}$ is associated with the random variable $\mathbf{Z}_i$. In each iteration of the algorithm, the unassigned variable x_i with the largest value $\mathbf{Z}_i$, among all other unassigned variables, is selected. In the other words, we can construct the permutation $s : [n] \to [n]$ such that $\mathbf{Z}_{s(1)} > \mathbf{Z}_{s(2)} > \cdots > \mathbf{Z}_{s(n)}$, and for all $t \in [n]$ the variable $x_{s(t)}$ is selected in the t-th iteration. The value selection based the output from the local rule τ can be represented by a random vector $\mathbf{U} = (\mathbf{U}_1, \mathbf{U}_2, ..., \mathbf{U}_n)$ whose entries are n i.i.d. random variables with uniform distribution over the interval $[0,1] \subset \mathbb{R}$. We call $\mathbf{U}$ the **internal vector** of the algorithm. In the t-th iteration of the algorithm, the value $\sigma(x_{s(t)})$ assigned to the selected variable $x_{s(t)}$ is set to be 1 if $\mathbf{U}_t < \tau(B_{\mathbf{\Phi}_t}(x_{s(t)}, R))$, and 0 otherwise. Conditioning on $\mathbf{\Phi}$, $\mathbf{Z}$ and $\mathbf{U}$, the output assignment σ can be uniquely determined. Therefore, we can view the τ-decimation algorithm $\mathtt{DEC}_\tau$ as a deterministic algorithm on random input $(\mathbf{\Phi}, \mathbf{Z}, \mathbf{U})$, and denote by $\sigma_{\mathbf{\Phi},\mathbf{Z},\mathbf{U}}$ the output of the algorithm.

With this notion of the deterministic algorithm, we can construct a sequence of output assignments which will be used in the argument of the overlap gap property method. The sequence of output assignments is generated by applying the τ-decimation algorithm $\mathtt{DEC}_\tau$ on a random k-XORSAT instance $\mathbf{\Phi}$ multiple times in the following way: First, given a random k-XORSAT instance $\mathbf{\Phi}$, we sample an ordering vector $\mathbf{Z}$ and an internal vector $\mathbf{U}$. Then, we run the τ-decimation algorithm $\mathtt{DEC}_\tau$ on input $\mathbf{\Phi}$ with the ordering vector $\mathbf{Z}$ and the internal vector $\mathbf{U}$ to get the first output assignment σ_0. After that, we re-randomize (i.e. sample again) the entries of the internal vector $\mathbf{U}$ one by one from $\mathbf{U}_1$ to $\mathbf{U}_n$. Right after each re-randomization we run the algorithm again to get a new output assignment. By doing this, we obtain a sequence of $n+1$ output assignments for the instance $\mathbf{\Phi}$ in total. We denote by σ_i the output assignment generated after re-randomizing the first i entries of $\mathbf{U}$, for $i = 0, 1, 2, ..., n$. Precisely speaking, let $\mathbf{V} = (\mathbf{V}_1, \mathbf{V}_2, ..., \mathbf{V}_n)$ and $\mathbf{W} = (\mathbf{W}_1, \mathbf{W}_2, ..., \mathbf{W}_n)$ be two independent random internal vectors with the uniform distribution over $[0,1]^n$, and set $\mathbf{U^i} = (\mathbf{W}_1, ..., \mathbf{W}_i, \mathbf{V}_{i+1}, ..., \mathbf{V}_n)$ for $i = 0, 1, 2, ..., n$. Note that $\mathbf{U^0} = \mathbf{V}$ and $\mathbf{U^n} = \mathbf{W}$.

Then, the sequence of output assignments $\{\sigma_i\}_{i=0}^n$ can be written as $\{\sigma_{\mathbf{\Phi},\mathbf{Z},\mathbf{U^i}}\}_{i=0}^n$, which is equivalent to the sequence of output assignment obtained by running the τ-decimation algorithm $\mathtt{DEC}_\tau$ (for $n+1$ times in total) on input $(\mathbf{\Phi},\mathbf{Z},\mathbf{U^i})$ for all $i=0,1,...,n$.

Recall the projection π mapping assignments for the instance $\mathbf{\Phi}$ to assignments for the core instance $\mathbf{\Phi}_c$, by removing all variables not in the core instance. We can further obtain a sequence of assignments for the core instance $\mathbf{\Phi}_c$ by applying the projection on the output assignments σ_i, that is, we set $\{\sigma_i' = \pi(\sigma_{\mathbf{\Phi},\mathbf{Z},\mathbf{U^i}})\}_{i=0}^n$.

3.2 Insensitive to internal vector

In this section, we show that the τ-decimation algorithm $\mathtt{DEC}_\tau$ is *insensitive* to its internal vector. By *insensitive*, it means when the value of an entry in the internal vector $\mathbf{U}$ is changed, only a small portion of the assigned values in the output assignment $\sigma_{\mathbf{\Phi},\mathbf{Z},\mathbf{U}}$ change accordingly. If so, every two consecutive output assignments in the sequence $\{\sigma_i = \sigma_{\Phi,\mathbf{Z},\mathbf{U^i}}\}_{i=0}^n$ should only differ from each other in only a small portion of assigned values.

Consider the sequence of output assignment $\{\sigma_i = \sigma_{\mathbf{\Phi},\mathbf{Z},\mathbf{U^i}}\}_{i=0}^n$ from Section 3.1. Note that the i-th output assignment σ_i in the sequence is the output of the algorithm on input $(\mathbf{\Phi},\mathbf{Z},\mathbf{U^i})$. For any $i\in[n]$, the only difference between the input $(\mathbf{\Phi},\mathbf{Z},\mathbf{U^{i-1}})$ and the input $(\mathbf{\Phi},\mathbf{Z},\mathbf{U^i})$ is the i-th entries of the internal vectors $\mathbf{U^{i-1}}$ and $\mathbf{U^i}$. We can immediately see that the insensitivity of the algorithm implies that every two consecutive output assignments in the sequence are close to each other. Gamarnik and Sudan [20] proved the insensitivity of the τ-decimation algorithm in their works, using the notion of *influence range*. Although their works [20] focused on the random NAE-k-SAT problem, the proof for the insensitivity of the τ-decimation algorithm is independent of the type of clauses in the random constraint satisfaction framework. So, we can directly use the result here.

▶ **Definition 10.** *Given a random instance $\mathbf{\Phi}$ and a random ordering vector $\mathbf{Z}$, we say that x_i **influences** x_j if either $x_i = x_j$ or in the variable-to-variable graph of the instance $\mathbf{\Phi}$ there exists a sequence of variable nodes $y_0, y_1, ..., y_t \in \{x_1, x_2, ..., x_n\}$ such that the following statements hold.*

1. *$y_0 = x_i$ and $y_t = x_j$.*
2. *There exists a path from y_l to y_{l+1}, of length at most r, in the variable-to-variable graph G, for $l = 0, 1, ..., t-1$.*
3. *$\mathbf{Z}_{y_{l-1}} > \mathbf{Z}_{y_l}$ for $l = 1, 2, ..., t$. In particular, $\mathbf{Z}_{x_i} > \mathbf{Z}_{x_j}$.*

*We define the **influence range** of x_i to be the set of all variables x_j influenced by x_i, denoted by $\mathcal{IR}_{x_i}$.*

▶ **Lemma 11.** *Given an instance Φ, a vector $Z\in[0,1]^n$, and two vectors $U, U'\in[0,1]^n$, we assume there exists $i\in\{1,2,...,n\}$ such that $U_i \neq U_i'$ and $U_j = U_j'$ for all $j\neq i$. Then, $\sigma_{\Phi,Z,U}(x) = \sigma_{\Phi,Z,U'}(x)$ for all variables $x_j \notin \mathcal{IR}_{x_i}$.*

▶ **Lemma 12.** *For any $\xi\in(0,1)$ and sufficiently large n,*

$$\Pr\left[\max_{1\le i\le n} |\mathcal{IR}_{x_i}| \ge n^{1/6}\right] \le \exp\left(-\ln n(\ln\ln n)^{\xi/4}\right).$$

They first showed that changing the value of only one entry, say U_i, in the internal vector U only affects the values assigned to the variables in the influence range of the variable x_i (Lemma 11). They further showed that w.h.p. the size of the influence range of variables is sublinear for all variables (Lemma 12). Note that in the original statement of Lemma 12 in [20], the index 1/6 in the inequality above can be any real number between 0 and 1/5.

Here, we pick a fixed value $1/6$ for simplicity. Combining these two lemmas, we can show that w.h.p. the differences between $\sigma_{\mathbf{\Phi},\mathbf{Z},\mathbf{U^{i-1}}}$ and $\sigma_{\mathbf{\Phi},\mathbf{Z},\mathbf{U^{i}}}$ is upper bounded by $n^{1/6}$ for all $i \in [n]$.

▶ **Lemma 13.** *For any $\xi \in (0,1)$ and sufficiently large n,*

$$\Pr\left[\, d(\sigma_{\mathbf{\Phi},\mathbf{Z},\mathbf{U^{i-1}}}, \sigma_{\mathbf{\Phi},\mathbf{Z},\mathbf{U^{i}}}) \geq n^{1/6} \text{ for some } i \in [n] \,\right] \leq \exp\left(-\ln n (\ln\ln n)^{\xi/4}\right).$$

Proof. Fix an arbitrary $i \in [n]$. We know that $\mathbf{U_j^{i-1}} = \mathbf{U_j^{i}}$ for all $j \neq i$, and $\mathbf{U_i^{i-1}} \neq \mathbf{U_i^{i}}$. By Lemma 11, we have $\sigma_{\mathbf{\Phi},\mathbf{Z},\mathbf{U^{i-1}}}(x_j) = \sigma_{\mathbf{\Phi},\mathbf{Z},\mathbf{U^{i}}}(x_j)$ for all variables $x_j \notin \mathcal{IR}_{x_i}$. If $d(\sigma_{\mathbf{\Phi},\mathbf{Z},\mathbf{U^{i-1}}}, \sigma_{\mathbf{\Phi},\mathbf{Z},\mathbf{U^{i}}}) \geq n^{1/6}$ for some $i \in [n]$, we have $|\mathcal{IR}_{x_i}| \geq n^{1/6}$. Hence, by Lemma 12, the result follows. ◀

3.3 Freeness

Recall the definition of *free steps*. An iteration of the τ-decimation algorithm $\mathtt{DEC}_\tau$ is called a free step if the local rule τ gives the value $1/2$ in that iteration. In this case, the value chosen by the τ-decimation algorithm for the selected variable is either 0 or 1 with even probability. Intuitively, it means that the local rule τ cannot capture useful information from the local structure to guide the τ-decimation algorithm choosing value for the selected variable, and thus the τ-decimation algorithm simply make a random guess for the assigned value. We also recall the definition of a τ-decimation algorithm being δ-free. A τ-decimation algorithm $\mathtt{DEC}_\tau$ is δ-free on the random k-XORSAT instance $\mathbf{\Phi}$ if w.h.p. the algorithm has at least δn free steps, on input $\mathbf{\Phi}$. Informal speaking, the more free the τ-decimation algorithm, the less the information captured by the local rule.

By using the Wormald's method of differential equations, we can calculate the degree profile of the remaining factor graph after t steps of the τ-decimation algorithm, for all $0 \leq t \leq n$. With the degree profiles, we can calculate the probability of each step being free, and thus approximate how free the τ-decimation algorithm is. The probability of having free steps depends on the choice of the local rules. Lemma 14 shows the freeness of the $\mathtt{UC}$-decimation algorithm.

▶ **Lemma 14.** *For $k \geq 3$ and $r > 0$, the $\mathtt{UC}$-decimation algorithm $\mathtt{DEC}_{\mathtt{UC}}$ is $w_1(k,r)$-free on the random k-XORSAT instance $\mathbf{\Phi} \sim \mathbf{\Phi}_k(n, rn)$, where*

$$w_1(k,r) = \frac{(kr)^{\frac{1}{1-k}}}{k-1}\,\gamma\left(\frac{1}{k-1}, kr\right)$$

and γ is the lower incomplete gamma function given by $\gamma(a,x) \equiv \int_0^x t^{a-1}e^{-t}dt$.

The role of the local rules is to approximate the marginal probability of the selected variable over a randomly chosen solution for the sub-instance induced by the local neighborhood of the selected variable. Interestingly, even we have a local rule τ that is capable to give the exact marginals when the factor graph is a tree, it still cannot provide enough useful information to guide the τ-decimation algorithm making good decision for the assigned value. With such a local rule, the τ-decimation algorithm still has a certain level of freeness.

▶ **Lemma 15.** *Assume the local rule τ can give the exact marginal probabilities of variables on any factor graph that is a tree. For $k \geq 3$ and $r > 0$, the τ-decimation algorithm $\mathtt{DEC}_\tau$ is $w_e(k,r)$-free on the random k-XORSAT instance $\mathbf{\Phi} \sim \mathbf{\Phi}_k(n, rn)$, where*

$$w_e(k,r) = \int_0^1 S_R(x)dx,$$

$S_0(x) = 1$ and $S_l(x) = \exp\left(-kr[(1-x)(1-S_{l-1}(x)) + x]^{k-1}\right)$ for any $l \geq 1$ and $x \in \mathbb{R}$.

4 Proof of main theorems

We denote by α_n the *success probability* of the τ-decimation algorithm $\texttt{DEC}_\tau$, namely, α_n is the probability that the assignment output by the τ-decimation algorithm $\texttt{DEC}_\tau$ on the random k-XORSAT instance $\mathbf{\Phi} \sim \mathbf{\Phi}_k(n, rn)$ with n variables and rn clauses is a solution for $\mathbf{\Phi}$. Formally, we define α_n by the following expression $\alpha_n \equiv \Pr\left[\, \sigma_{\mathbf{\Phi}\sim\mathbf{\Phi}_k(n,rn),\mathbf{Z},\mathbf{U}} \in \mathcal{S}(\mathbf{\Phi}) \,\right]$, where $\mathbf{\Phi} \sim \mathbf{\Phi}_k(n, rn)$ is the random k-XORSAT instance, $\mathbf{Z}$ is the random ordering vector, and $\mathbf{U}$ is the random internal vector, as mentioned in Section 3.1. Now, we consider the sequence of output assignments $\{\sigma_i = \sigma_{\mathbf{\Phi},\mathbf{Z},\mathbf{U^i}}\}_{i=0}^n$ generated by the procedure in Section 3.1. We first prove that if the algorithm $\texttt{DEC}_\tau$ is δ-free, then the expected distance $\mathbb{E}\left[d(\sigma_0, \sigma_n)\right]$ between the first and the last assignments in the sequence is at least $(\delta/2)n + o(n)$.

▶ **Lemma 16.** *If the τ-decimation algorithm $\texttt{DEC}_\tau$ is δ-free on the random k-XORSAT instance $\mathbf{\Phi} \sim \mathbf{\Phi}_k(n, rn)$ for some $\delta > 0$, then we have $\mathbb{E}\left[d(\sigma_0, \sigma_n)\right] \geq (\delta/2)n + o(n)$.*

Next, we will show that, if the τ-decimation algorithm is "free enough", namely, strictly $2\mu(k, r)$-free, then we can pick a pair of output assignments and project them to the core instance $\mathbf{\Phi}_c$ so that the distance between the two corresponding core assignments falls in the *forbidden range* from the overlap gap property of the core instance $\mathbf{\Phi}_c$.

▶ **Lemma 17.** *For any $k \geq 3$ and $r \in (r_{core}(k), r_{sat}(k))$, if the τ-decimation algorithm $\texttt{DEC}_\tau$ is strictly $2\mu(k, r)$-free on the random k-XORSAT instance $\mathbf{\Phi} \sim \mathbf{\Phi}_k(n, rn)$, then there exist $0 \leq i_0 \leq n$ and $0 < \epsilon' < \epsilon(k, r)$ such that w.h.p. we have $\left|d(\pi(\sigma_0), \pi(\sigma_{i_0})) - \frac{1}{2}\epsilon' n\right| < \frac{1}{4}\epsilon' n$, where $\epsilon(k, r)$ is given in Lemma 8.*

The following lemma shows that the probability of both the output assignments $\sigma_{\mathbf{\Phi},\mathbf{Z},\mathbf{U^o}}$ and $\sigma_{\mathbf{\Phi},\mathbf{Z},\mathbf{U^{i_o}}}$ being solutions for the instance $\mathbf{\Phi}$ is lower bounded by α_n^2.

▶ **Lemma 18.** *For any $i \in [n]$, we have $\Pr\left[\, \sigma_0 \in \mathcal{S}(\Phi) \text{ and } \sigma_i \in \mathcal{S}(\Phi) \,\right] \geq \alpha_n^2$.*

Finally, we can combine all above lemmas in this section to give the proof of Theorem 1.

Proof of Theorem 1. We denote by $\mathcal{A}$ the event of $\left|d(\pi(\sigma_0), \pi(\sigma_{i_0})) - \frac{1}{2}\epsilon' n\right| < \frac{1}{4}\epsilon' n$, and we have $\Pr\left[\,\mathcal{A}\,\right] = 1 - o(1)$ by Lemma 17. On the other hand, we pick $i = i_0$ for the inequality in Lemma 18. We denote by $\mathcal{B}$ the event of $\sigma_0 \in \mathcal{S}(\Phi)$ and $\sigma_{i_0} \in \mathcal{S}(\Phi)$, and $\Pr\left[\,\mathcal{B}\,\right] \geq \alpha_n^2$. Note that we have $\Pr\left[\,\mathcal{A} \cap \mathcal{B}\,\right] \geq 1 - \Pr\left[\,\text{Not } \mathcal{A}\,\right] - \Pr\left[\,\text{Not } \mathcal{B}\,\right] \geq 1 - o(1) - (1 - \alpha_n^2) = \alpha_n^2 - o(1)$. Thus, we have $\alpha_n \leq \Pr\left[\,\mathcal{A} \cap \mathcal{B}\,\right]^{1/2} + o(1)$.

Now assume both $\mathcal{A}$ and $\mathcal{B}$ take places. Since both σ_0 and σ_{i_0} are solutions for the random instance $\mathbf{\Phi}$, both $\pi(\sigma_0)$ and $\pi(\sigma_{i_0})$ are solutions for the core instance $\mathbf{\Phi}_c$. Moreover, the distance $d(\pi(\sigma_0), \pi(\sigma_{i_0}))$ falls in the interval $((1/4)\epsilon' n, (3/4)\epsilon' n) \subsetneq (o(n), \epsilon n)$, which takes place with probability at most $o(1)$ by Lemma 8. So, we have $\Pr\left[\,\mathcal{A} \cap \mathcal{B}\,\right] \leq o(1)$, and thus $\alpha_n \leq o(1)$. ◀

To prove Theorem 2 and 3, all we need to do is to show that $\texttt{DEC}_{\texttt{UC}}$ and $\texttt{DEC}_\tau$ with the exact marginal assumption are strictly $2\mu(k, r)$-free. The results immediately follow by applying Theorem 1. From Lemma 14 and 15, we know that $\texttt{DEC}_{\texttt{UC}}$ and $\texttt{DEC}_\tau$ are $w_1(k, r)$-free and $w_e(k, r)$-free, respectively. So, we only need to show that $w_1(k, r) > 2\mu(k, r)$ and $w_e(k, r) > 2\mu(k, r)$. It can be done with the following lemmas, which give an upper bound of $\mu(k, r)$ in Lemma 19, a lower bound of $w_1(k, r)$ in Lemma 20, and a lower bound of and $w_e(k, r)$ in Lemma 21.

▶ **Lemma 19.** *For any $k \geq 4$ and $r \in (r_{core}(k), r_{sat}(k))$, we have $\mu(k, r) < \mu_u(k)$, where*

$$\mu_u(k) = (1 - e^{-1/k}) - (1 - e^{-1/k}) \ln(1 - e^{-1/k}).$$

▶ **Lemma 20.** *For any $k \geq k_0 \geq 3$ and $r \in (r_{core}(k), r_{sat}(k))$, $w_1(k,r) \geq w_1^*(k_0)$, where*

$$w_1^*(k) = \frac{k^{\frac{1}{1-k}}}{k-1}\gamma\left(\frac{1}{k-1}, k\left(\frac{k}{k+1}\right)^{k-1}\right)$$

▶ **Lemma 21.** *For any $k \geq k_0 \geq 3$ and $r \in (r_{core}(k), r_{sat}(k))$, we have $w_e(k,r) \geq w_e^*(k_0, r_{sat}(k_0))$, where $w_e^*(k,r) = x^-(k,r) - kr^2(x^-(k,r))^k$ and*

$$x^{\pm}(k,r) = \left(\frac{1 \pm \sqrt{1 - 4(kr)^{-2}[(kr)^{\frac{1}{k-1}} - 1]}}{2}\right)^{\frac{1}{k-2}}. \tag{1}$$

Proof of Theorem 2. Let $k \geq 9$ and $r \in (r_{core}(k), r_{sat}(k))$. By Lemma 19 and 20, we have $2\mu(k,r) < 2\mu_u(9) \leq 0.3420 < 0.3575 \leq w_1^*(9) \leq w_1(k,r)$. Then, by Lemma 14, $\texttt{DEC}_{\texttt{UC}}$ is strictly $2\mu(k,r)$-free. The result follows. ◀

Proof of Theorem 3. Let $k \geq 13$ and $r \in (r_{core}(k), r_{sat}(k))$. By Lemma 19 and 21, we have $2\mu(k,r) < 2\mu_u(13) \leq 0.2668 < 0.2725 \leq w_e^*(13) \leq w_e(k,r)$. Then, by Lemma 15, $\texttt{DEC}_{\tau}$ is strictly $2\mu(k,r)$-free. The result follows. ◀

References

1. Emmanuel Abbe, Shuangping Li, and Allan Sly. Binary perceptron: Efficient algorithms can find solutions in a rare well-connected cluster. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 860–873, New York, NY, USA, June 2022. Association for Computing Machinery. doi:10.1145/3519935.3519975.
2. Emmanuel Abbe, Shuangping Li, and Allan Sly. Proof of the Contiguity Conjecture and Lognormal Limit for the Symmetric Perceptron. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 327–338, February 2022. doi:10.1109/FOCS52979.2021.00041.
3. Dimitris Achlioptas and Amin Coja-Oghlan. Algorithmic Barriers from Phase Transitions. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 793–802, Philadelphia, PA, USA, October 2008. IEEE. doi:10.1109/FOCS.2008.11.
4. Dimitris Achlioptas, Amin Coja-Oghlan, and Federico Ricci-Tersenghi. On the solution-space geometry of random constraint satisfaction problems. *Random Structures & Algorithms*, 38(3):251–268, May 2011. doi:10.1002/rsa.20323.
5. Dimitris Achlioptas, Jeong Han Kim, Michael Krivelevich, and Prasad Tetali. Two-coloring random hypergraphs. *Random Structures and Algorithms*, 20(2):249–259, March 2002. doi:10.1002/rsa.997.
6. Dimitris Achlioptas and Michael Molloy. The solution space geometry of random linear equations. *Random Structures & Algorithms*, 46(2):197–231, March 2015. doi:10.1002/rsa.20494.
7. A. Braunstein, M. Mézard, and R. Zecchina. Survey propagation: An algorithm for satisfiability. *Random Structures and Algorithms*, 27(2):201–226, September 2005. doi:10.1002/rsa.20057.
8. Alfredo Braunstein and Riccardo Zecchina. Survey propagation as local equilibrium equations. *Journal of Statistical Mechanics: Theory and Experiment*, 2004(06):P06007, June 2004. doi:10.1088/1742-5468/2004/06/P06007.
9. Guy Bresler and Brice Huang. The Algorithmic Phase Transition of Random k-SAT for Low Degree Polynomials. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 298–309, Denver, CO, USA, February 2022. IEEE. doi:10.1109/FOCS52979.2021.00038.

10 S. Cocco, O. Dubois, J. Mandler, and R. Monasson. Rigorous Decimation-Based Construction of Ground Pure States for Spin-Glass Models on Random Lattices. *Physical Review Letters*, 90(4):047205, January 2003. doi:10.1103/PhysRevLett.90.047205.

11 Amin Coja-Oghlan. A Better Algorithm for Random k -SAT. *SIAM Journal on Computing*, 39(7):2823–2864, January 2010. doi:10.1137/09076516X.

12 Amin Coja-Oghlan. Belief Propagation Guided Decimation Fails on Random Formulas. *Journal of the ACM*, 63(6):1–55, February 2017. doi:10.1145/3005398.

13 Martin Dietzfelbinger, Andreas Goerdt, Michael Mitzenmacher, Andrea Montanari, Rasmus Pagh, and Michael Rink. Tight Thresholds for Cuckoo Hashing via XORSAT. In *Automata, Languages and Programming*, volume 6198, pages 213–225. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. doi:10.1007/978-3-642-14165-2_19.

14 Olivier Dubois and Jacques Mandler. The 3-XORSAT threshold. *Comptes Rendus Mathematique*, 335(11):963–966, December 2002. doi:10.1016/S1631-073X(02)02563-3.

15 David Gamarnik. The overlap gap property: A topological barrier to optimizing over random structures. *Proceedings of the National Academy of Sciences*, 118(41):e2108492118, October 2021. doi:10.1073/pnas.2108492118.

16 David Gamarnik and Eren C. Kızıldağ. Algorithmic obstructions in the random number partitioning problem. *The Annals of Applied Probability*, 33(6B), December 2023. doi:10.1214/23-AAP1953.

17 David Gamarnik, Eren C. Kizildağ, Will Perkins, and Changji Xu. Geometric Barriers for Stable and Online Algorithms for Discrepancy Minimization. In *Proceedings of Thirty Sixth Conference on Learning Theory*, pages 3231–3263. PMLR, July 2023.

18 David Gamarnik and Quan Li. Finding a large submatrix of a Gaussian random matrix. *The Annals of Statistics*, 46(6A), December 2018. doi:10.1214/17-AOS1628.

19 David Gamarnik and Madhu Sudan. Limits of local algorithms over sparse random graphs. *The Annals of Probability*, 45(4), July 2017. doi:10.1214/16-AOP1114.

20 David Gamarnik and Madhu Sudan. Performance of Sequential Local Algorithms for the Random NAE-K-SAT Problem. *SIAM Journal on Computing*, 46(2):590–619, January 2017. doi:10.1137/140989728.

21 Carla P. Gomes and Bart Selman. Satisfied with Physics. *Science*, 297(5582):784–785, August 2002. doi:10.1126/science.1074599.

22 Marco Guidetti and A. P. Young. Complexity of several constraint-satisfaction problems using the heuristic classical algorithm WalkSAT. *Physical Review E*, 84(1):011102, July 2011. doi:10.1103/PhysRevE.84.011102.

23 Samuel Hetterich. Analysing Survey Propagation Guided Decimation on Random Formulas, February 2016. arXiv:1602.08519.

24 Brice Huang and Mark Sellke. Tight Lipschitz Hardness for optimizing Mean Field Spin Glasses. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 312–322, Denver, CO, USA, October 2022. IEEE. doi:10.1109/FOCS54457.2022.00037.

25 Morteza Ibrahimi, Yashodhan Kanoria, Matt Kraning, and Andrea Montanari. The Set of Solutions of Random XORSAT Formulae. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 760–779. Society for Industrial and Applied Mathematics, January 2012. doi:10.1137/1.9781611973099.62.

26 Jeong Han Kim. The Poisson Cloning Model for Random Graphs, Random Directed Graphs and Random k-SAT Problems. In *Computing and Combinatorics*, volume 3106, pages 2–2. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. doi:10.1007/978-3-540-27798-9_2.

27 Lukas Kroc, Ashish Sabharwal, and Bart Selman. Message-passing and local heuristics as decimation strategies for satisfiability. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, pages 1408–1414, Honolulu Hawaii, March 2009. ACM. doi:10.1145/1529282.1529596.

28 Elitza Maneva, Elchanan Mossel, and Martin J. Wainwright. A new look at survey propagation and its generalizations. *Journal of the ACM*, 54(4):17, July 2007. doi:10.1145/1255443.1255445.

29 M. Mézard, G. Parisi, and R. Zecchina. Analytic and Algorithmic Solution of Random Satisfiability Problems. *Science*, 297(5582):812–815, August 2002. doi:10.1126/science.1073287.

30 M. Mézard, F. Ricci-Tersenghi, and R. Zecchina. Two Solutions to Diluted p-Spin Models and XORSAT Problems. *Journal of Statistical Physics*, 111(3/4):505–533, 2003. doi:10.1023/A:1022886412117.

31 Marc Mézard and Andrea Montanari. *Information, Physics, and Computation*. Oxford Graduate Texts. Oxford University Press, Oxford ; New York, 2009.

32 Michael Molloy. Cores in random hypergraphs and Boolean formulas. *Random Structures and Algorithms*, 27(1):124–135, August 2005. doi:10.1002/rsa.20061.

33 Will Perkins and Changji Xu. Frozen 1-RSB structure of the symmetric Ising perceptron. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 1579–1588, New York, NY, USA, June 2021. Association for Computing Machinery. doi:10.1145/3406325.3451119.

34 Boris Pittel and Gregory B. Sorkin. The Satisfiability Threshold for k -XORSAT. *Combinatorics, Probability and Computing*, 25(2):236–268, March 2016. doi:10.1017/S0963548315000097.

35 Boris Pittel, Joel Spencer, and Nicholas Wormald. Sudden Emergence of a Giant k-Core in a Random Graph. *Journal of Combinatorial Theory, Series B*, 67(1):111–151, May 1996. doi:10.1006/jctb.1996.0036.

36 Federico Ricci-Tersenghi and Guilhem Semerjian. On the cavity method for decimated random constraint satisfaction problems and the analysis of belief propagation guided decimation algorithms. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(09):P09001, September 2009. doi:10.1088/1742-5468/2009/09/P09001.

37 Alexander S. Wein. Optimal low-degree hardness of maximum independent set. *Mathematical Statistics and Learning*, 4(3):221–251, January 2022. doi:10.4171/msl/25.

Lookahead Games and Efficient Determinisation of History-Deterministic Büchi Automata

Rohan Acharya
University of Warwick, Coventry, UK

Marcin Jurdziński
University of Warwick, Coventry, UK

Aditya Prakash
University of Warwick, Coventry, UK

Abstract

Our main technical contribution is a polynomial-time determinisation procedure for history-deterministic Büchi automata, which settles an open question of Kuperberg and Skrzypczak, 2015. A key conceptual contribution is the lookahead game, which is a variant of Bagnol and Kuperberg's token game, in which Adam is given a fixed lookahead. We prove that the lookahead game is equivalent to the 1-token game. This allows us to show that the 1-token game characterises history-determinism for semantically-deterministic Büchi automata, which paves the way to our polynomial-time determinisation procedure.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases History determinism, Good-for-games, Automata over infinite words, Games

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.124

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2404.17530` [2]

Funding We acknowledge the Centre for Discrete Mathematics and Its Applications (DIMAP) at the University of Warwick for partial support.
Rohan Acharya: Undergraduate Research Support Scheme and the Department of Computer Science, University of Warwick.
Aditya Prakash: Chancellors' International Scholarship from the University of Warwick.

Acknowledgements We thank Udi Boker, Denis Kuperberg, and Karoliina Lehtinen for several insightful exchanges. We are grateful to the reviewers for their feedback and suggestions on how to improve the paper.

1 Introduction

History-deterministic (HD) automata are non-deterministic automata in which the non-determinism can be resolved "on the fly", based only on the prefix of the word read so far [6, 15]. This concept can be formalised using the history-determinism game (HD game), in which two players Adam and Eve make alternating moves choosing letters and transitions, thus constructing a word and a run of the automaton on it, respectively. Eve wins if the run is accepting or if the word is not in the language, and hence Eve's winning strategy will successfully resolve non-determinism by constructing an accepting run on the fly, for all words in the language. An automaton is then defined to be history-deterministic if Eve has a winning strategy in the game.

Henzinger and Piterman [11] introduced HD automata because of their potential to speed up key algorithmic tasks in verification and synthesis, such as language containment and strategy synthesis. In language containment, we ask whether all executions of an

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 124; pp. 124:1–124:18


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

implementation $\mathcal{A}$ satisfy a specification $\mathcal{H}$. If $\mathcal{H}$ is non-deterministic then the problem is **PSPACE**-hard, but if $\mathcal{H}$ is HD then it is more tractable, because it amounts to checking that $\mathcal{H}$ simulates $\mathcal{A}$. This can be done in polynomial time if the parity index of $\mathcal{H}$ is fixed [8, Theorem 3] and in quasi-polynomial time otherwise [18, Theorem 20]. Henzinger and Piterman had originally dubbed HD automata as good-for-games automata in their work because games whose winning conditions are represented by an HD automaton can be solved efficiently without automaton determinisation [11, Theorem 3.1], a well-known computational bottleneck in synthesis.

1.1 Related work

Key questions studied for HD parity automata include recognising them, their succinctness relative to deterministic automata, minimisation, and determinisation.

Recognising History-Deterministic Automata via Games. Kuperberg and Skrzypczak [14] gave a polynomial time algorithm to recognise HD co-Büchi automata, and Bagnol and Kuperberg [3] gave a polynomial time algorithm to recognise HD Büchi automata. These algorithms have been conceptually unified by Boker, Kuperberg, Lehtinen, and Skrzypczak [5] to be based on the 2-token game introduced by Bagnol and Kuperberg [3], leading to the 2-token conjecture.

▶ **Conjecture 1** (The 2-token conjecture [3, 5]). *A parity automaton is HD if and only if Eve wins the 2-token game on it.*

Proving the 2-token conjecture would imply that recognising HD parity automata of fixed parity index can be done in polynomial time. In contrast, the best upper bound currently known for the problem is **EXPTIME**, dating back to Henzinger and Piterman [11]. In the general case, when the parity index is not fixed, a lower bound of **NP**-hardness has been achieved only very recently [18].

Bagnol and Kuperberg [3] introduced the k-token game as a tool to characterise the conceptually more complex HD game. Like in the HD game, in the k-token game for $k \geq 1$, two players Adam and Even make alternating moves choosing letters and transitions, but in addition, in every round, after Eve chooses a transition, Adam also chooses k transitions. As a result, Adam constructs a word and k runs, Eve constructs a run, and Eve wins if her run is accepting or all of Adam's k runs are rejecting. A key insight in Bagnol and Kuperberg's work [3] is that the 2-token game is equivalent to the k-token game for all $k \geq 2$.

▶ **Theorem 2** ([3]). *Eve wins the 2-token game on a parity automaton if and only if for all $k \geq 2$, Eve wins the k-token game on it.*

Bagnol and Kuperberg's proof that the 2-token game characterises history-determinism for Büchi automata exploits this insight, by showing that if Adam wins the HD game on a Büchi automaton then he can win the k-token game for some k that is doubly-exponential in the size of the automaton, and hence also the 2-token game.

Boker et al. [5] have used an analogous, but more involved, argument to show that the 2-token game also characterises history-determinism for co-Büchi automata, combining Theorem 2 with the algorithm of Kuperberg and Skrzypczak [14] to recognise HD co-Büchi automata efficiently, which was based on the so-called Joker game. The Joker game is similar to the 1-token game but, additionally, Adam has the power to (finitely many times) "play Joker" by choosing a transition from Eve's token instead of a transition from his token, and Eve wins if her run is accepting, or Adam's run is rejecting, or Adam has played Joker infinitely many times.

Kuperberg and Skrzypczak's algorithm uses Joker games in their polynomial time algorithm to recognise HD co-Büchi automata, but to date, it was not known if Joker games characterise history-determinism on co-Büchi or Büchi automata, or on parity automata in general.

Succinctness and minimisation of HD automata. Kuperberg and Skrzypczak [14] proved that HD co-Büchi automata are exponentially more succinct than deterministic co-Büchi automata [14], which is tight [15, Theorem 4.1]. Abu Radi and Kupferman [1] showed that transition-based HD co-Büchi automata can be minimised in polynomial time and that they have canonicity. In contrast, minimisation of state-based HD Büchi or HD co-Büchi automata is **NP**-complete [22, Theorem 1]. Minimisation of transition-based Büchi automata is easily seen to be in **NP**, but the exact complexity is open.

Determinisation of HD Büchi automata. Kuperberg and Skrzypczak [14] also proved that every HD Büchi automaton with n states has an equivalent deterministic Büchi automaton with at most n^2 states. However, it is not known if HD Büchi automata are strictly more succinct than deterministic Büchi automata.

The determinisation procedure of Kuperberg and Skrzypczak for an HD Büchi automaton $\mathcal{H}$ involves carefully analysing the simulation game between $\mathcal{H}$ and an equivalent deterministic Büchi automaton of exponential size. At a high level, the procedure iteratively modifies the simulation game and the automaton $\mathcal{H}$, eventually yielding an equivalent game of quadratic size, from which a deterministic Büchi automaton of quadratic size can be extracted, but the procedure itself runs in exponential time.

Kuperberg and Skrzypczak also gave a non-deterministic polynomial-time procedure for determinisation of HD Büchi automata, which guesses a deterministic Büchi automaton of quadratic size and then checks for language equivalence [14, Theorem 10]. They left the exact complexity of determinisation for HD Büchi automata open, in particular, the question of whether HD Büchi automata can be determinised in polynomial time.

1.2 Our Contributions

- We introduce the k-lookahead game, a variant of the 1-token game, in which Adam's transition on his token is delayed by k steps, thus giving him a lookahead of k. We prove that the 1-token game is equivalent to the k-lookahead game.

  ▶ **Theorem A.** *For every parity automaton $\mathcal{A}$, Eve wins the 1-lookahead game on $\mathcal{A}$ if and only if she wins the k-lookahead game on $\mathcal{A}$.*

  The 1-token game is syntactically equivalent to the 1-lookahead game. Theorem A thus demonstrates that the 1-token game is already quite powerful, and it is analogous to Theorem 2 of Bagnol and Kuperberg.
- With Theorem A as a key tool, we show that the 1-token game characterises history-determinism on semantically-deterministic Büchi automata. These are automata in which, for every state, all transitions labelled by the same letter lead to language-equivalent states [20].

  ▶ **Theorem B.** *A semantically-deterministic Büchi automaton is history-determinsitic if and only if Eve wins the 1-token game on it.*

  A consequence of Theorem B is that Joker games characterise history-determinism on Büchi automata (Theorem 19). Since Joker games have smaller arenas than 2-token games, this leads to a more efficient algorithm for recognising HD Büchi automata (Lemma 20).

- We give a parity automaton with priorities 1, 2, and 3 on which Eve wins the Joker game but that is not HD (Theorem 23). This implies that the Joker game does not characterise history-determinism for parity automata and that Theorem B does not extend to parity automata.
- We give a polynomial time determinisation procedure for HD Büchi automata, thus resolving an open question of Kuperberg and Skrzypczak [14].
  ▶ **Theorem C.** *There is a polynomial-time procedure that converts every HD Büchi automaton with n states into an equivalent deterministic Büchi automaton with n^2 states.*
  Our determinisation procedure is inspired by that of Kuperberg and Skrzypczak [14], but rather than working with the simulation game between the automaton and a deterministic automaton of exponential size, thanks to Theorem B, we can work with the 1-token game instead. This results in an algorithm that is conceptually simpler and that runs in polynomial time.
- We also give a technique to reduce game-based characterisations of history-determinism to universal automata (automata that accept all words). Hence to prove the 1-token game characterisation of history-determinism for semantically-deterministic (SD) Büchi automata, it suffices to prove it for universal SD Büchi automata (Theorem 11). Likewise, to prove the 2-token conjecture for parity automata, it suffices to prove it for universal parity automata (Theorem 13).

2 Preliminaries

We let $\mathbb{N} = \{0, 1, 2, \cdots\}$ to be the set of natural numbers. For two natural numbers i, j such that $i < j$, we write $[i, j]$ to denote the set of integers $\{i, i+1, \ldots, j\}$, and $[j]$ to denote $[0, j]$. An *alphabet* Σ is a finite set of *letters*. We use Σ^* and Σ^ω to denote the set of words of finite and countably infinite length over Σ, respectively. We also let ε be the unique word of length 0. A language $\mathcal{L} \in \Sigma^\omega$ is a set of infinite words. For a finite word $u \in \Sigma^*$ and a language $\mathcal{L}$, we define $u^{-1}\mathcal{L}$ to be $\{w \mid uw \in \mathcal{L}\}$.

2.1 Games

Game arenas. An *arena* is a directed graph $G = (V, E)$ with vertices partitioned into $V_\forall$ and $V_\exists$ between two players Adam and Eve, respectively. Additionally, a vertex $v_0 \in V_\forall$ is designated as the initial vertex. We say that vertices in $V_\exists$ are owned by Eve and those in $V_\forall$ are owned by Adam.

A *play* on this arena is an infinite path starting at v_0 and it is formed as follows. A play starts with a token at v_0 and it proceeds for infinitely many rounds. At each round, the player who owns the vertex on which the token is currently placed chooses an outgoing edge, and the token is moved along this edge to the next vertex for another round of play. This creates an infinite path in the arena, which we call a play of G.

A *game* $\mathcal{G}$ consists of an arena $G = (V, E)$ and a winning condition given by a language $L \subseteq E^\omega$. We say that Eve *wins a play* ρ in G if $\rho \in L$, and Adam wins otherwise. A *strategy* for Eve in such a game $\mathcal{G}$ is a function from the set of plays that end at an Eve's vertex to an outgoing edge from that vertex. Eve's strategy is said to be winning if any play produced while she plays according to this strategy is winning for her. We say that Eve *wins the game* if she has a winning strategy. Winning strategies are defined for Adam analogously, and we say that Adam wins the game if he has a winning strategy. In this paper we deal with ω-regular games, which are known to be determined [17, 10], i.e., each game has a winner. Two games are *equivalent* if they have the same winner.

Parity games. An $[i,j]$-*parity game* $\mathcal{G}$ is played over a finite game arena $G = (V, E)$, with the edges of G labelled by a priority function $\chi : E \to [i,j]$ for some $i, j \in \mathbb{N}$ with $i < j$, and $i = 0$ or $i = 1$. A play ρ in the arena of $\mathcal{G}$ is winning for Eve if and only if the highest edge priority that occurs infinitely often is even. It is well known that parity games can be solved in polynomial time when the interval $[i,j]$ is fixed, and in quasi-polynomial time otherwise [7, 13, 16].

2.2 Automata

Parity automata. An $[i,j]$-*non-deterministic parity automaton* $\mathcal{A} = (Q, \Sigma, q_0, \Delta)$ consists of a finite directed graph with edges labelled by letters in Σ and *priorities* in $[i,j]$ for some $i, j \in \mathbb{N}$ with $i < j$. These edges are called *transitions*, which are elements of the set $\Delta \subseteq Q \times \Sigma \times [i,j] \times Q$, and the vertices of this graph are called *states*, which are elements of the set Q. Each automaton has a designated *initial state* $q_0 \in Q$. For states $p, q \in Q$ and a letter $a \in \Sigma$, we use $p \xrightarrow{a:c} q$ to denote a transition from p to q on the letter a that has the priority c.

A *run* on an infinite word w in Σ^ω is an infinite path in the automaton, starting at the initial state and following transitions that correspond to the letters of w in sequence. We write that such a run is *accepting* if it *satisfies the parity condition*, i.e., the highest priority occurring infinitely often amongst the transitions of the run is even, and a word w in Σ^ω is accepting if the automaton has an accepting run on w. The *language* of an automaton $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, is the set of words that it accepts. We write that the automaton $\mathcal{A}$ *recognises* a language $\mathcal{L}$ if $\mathcal{L}(\mathcal{A}) = \mathcal{L}$. A language $\mathcal{L} \subseteq \Sigma^\omega$ is *ω-regular* if it is recognised by some parity automaton. A parity automaton $\mathcal{A}$ is *deterministic* if for any given state in $\mathcal{A}$ and any given letter in Σ, there is at most one outgoing transition from that state on that letter.

We write that $[i,j]$, with $i = 0$ or 1, is the parity index of $\mathcal{A}$. A *Büchi* (resp. co-Büchi) automaton is a $[1,2]$ (resp. $[0,1]$) parity automaton. A *safety automaton* is one where all transitions have priority 0.

We write $(\mathcal{A}, q)$ to denote the automaton $\mathcal{A}$ with q as its initial state, and $\mathcal{L}(\mathcal{A}, q)$ to denote the language it recognises. Two states p and q in $\mathcal{A}$ are *equivalent* if $\mathcal{L}(\mathcal{A}, p) = \mathcal{L}(\mathcal{A}, q)$.

History-determinism. The (HD game) is a two player turn-based game between Adam and Eve, who take alternating turns to select a letter and a transition in the automaton (on that letter), respectively. After the game ends, the sequence of Adam's choices of letters is an infinite word, and the sequence of Eve's choices of transitions is a run on that word. Eve wins the game if her run is accepting or Adam's word is rejecting, and we say that an automaton is HD if Eve has a winning strategy in the history-determinism game.

▶ **Definition 3** (History-determinism game). *Given a parity automaton $\mathcal{A} = (Q, \Sigma, q_0, \Delta)$, the* history-determinism game *of $\mathcal{A}$ is defined between the players Adam and Eve as follows, with positions in Q. The game starts at q_0 and proceeds for infinitely many rounds. For each $i \in \mathbb{N}$, round i starts at a position $q_i \in Q$, and proceeds as follows:*

1. *Adam selects a letter $a_i \in \Sigma$;*
2. *Eve selects a transition $q_i \xrightarrow{a_i:c_i} q_{i+1} \in \Delta$.*

The new position is q_{i+1} from where round $(i+1)$ is played. Thus, the play of a history-determinism game can be seen as Adam constructing a word letter-by-letter, and Eve constructing a run transition-by-transition on the same word. Eve wins such a play if the following holds: if Adam's word is in $\mathcal{L}(\mathcal{A})$, then Eve's run is accepting.

If Eve wins the history-determinism game on $\mathcal{A}$, then we say that $\mathcal{A}$ is *history-deterministic*.

Semantic-determinism. Let $\mathcal{A}$ be a parity automaton. A transition δ from p to q on a letter a in $\mathcal{A}$ is called *language-preserving* if $\mathcal{L}(\mathcal{A}, q) = a^{-1}\mathcal{L}(\mathcal{A}, p)$. We say that a parity automaton is *semantically-deterministic*, SD for short, if all transitions in it are language-preserving. The following lemma can be shown by a simple inductive argument on the length of words.

▶ **Lemma 4.** *If a parity automaton is SD then all states in the automaton that can be reached from a fixed state q upon reading a finite word u accept the language $u^{-1}\mathcal{L}(\mathcal{A}, q)$.*

SD automata were introduced by Kuperberg and Skrzypczak as residual automata [14]. We follow Abu Radi, Kupferman, and Leshkowitz [21] by calling them SD automata instead.

2.3 Games on Automata

Simulation and simulation-like games (such as token games [3]) are fundamental amongst the techniques we use in this paper. We define these games below.

Simulation and stepahead simulation.

▶ **Definition 5** (Simulation game). *Given two parity automata $\mathcal{A} = (P, \Sigma, p_0, \Delta_A)$ and $\mathcal{B} = (Q, \Sigma, q_0, \Delta_B)$, the* simulation game *between $\mathcal{B}$ and $\mathcal{A}$ is a two player game played between Eve and Adam as follows, with positions in $P \times Q$. The game starts at (p_0, q_0), and proceeds for infinitely many rounds. For each $i \geq 0$, round i starts at position (p_i, q_i) and proceeds as follows:*

1. *Adam selects a letter $a_i \in \Sigma$;*
2. *Adam selects a transition $p_i \xrightarrow{a_i:c'} p_{i+1}$ in $\mathcal{A}$;*
3. *Eve selects a transition $q_i \xrightarrow{a_i:c} q_{i+1}$ in $\mathcal{B}$.*

At the end of a play of the simulation game, the letters selected by Adam in sequence form a word, while the sequence of his selected transitions and the sequence of Eve's selected transitions form runs on that word in $\mathcal{A}$ and $\mathcal{B}$, respectively. We say Eve wins the game if her run in $\mathcal{B}$ is accepting or Adam's run in $\mathcal{A}$ is rejecting. If Eve has a strategy to win the simulation game, then we say that $\mathcal{B}$ simulates *$\mathcal{A}$.*

The *stepahead simulation* game between $\mathcal{B}$ and $\mathcal{A}$ is defined similarly to the simulation game, except the orders of move in each round are changed as follows: Adam selects a letter first, then Eve selects a transition on $\mathcal{B}$, and then Adam selects a transition on $\mathcal{A}$. The winning condition is identical, which is that Eve's run on $\mathcal{B}$ is accepting if Adam's run on $\mathcal{A}$ is accepting. If Eve wins the stepahead simulation game between $\mathcal{B}$ and $\mathcal{A}$, then we say that $\mathcal{B}$ *step-ahead simulates* $\mathcal{A}$.

Token games. k-token games are similar to stepahead simulation games, but are played on a single automaton, and Adam constructs k runs instead of one. The winning objective of Eve requires her to construct an accepting run if one of k Adam's runs is accepting.

▶ **Definition 6** (k-token game). *The* k-token game *on a non-deterministic parity automaton $\mathcal{A} = (Q, \Sigma, q_0, \Delta)$ is defined between the players Adam and Eve as follows, with positions in $Q \times Q^k$. The game starts at $(q_0, (q_0)^k)$ and proceeds in ω many rounds. For each $i \in \mathbb{N}$, the round i starts at a position $(q_i, (p_i^1, p_i^2, \cdots, p_i^k)) \in Q \times Q^k$, and proceeds as follows.*

1. *Adam selects a letter $a_i \in \Sigma$.*
2. *Eve selects a transition $q_i \xrightarrow{a_i:c} q_{i+1} \in \Delta$.*
3. *Adam selects k transitions $p_i^1 \xrightarrow{a_i:c'_1} p_{i+1}^1, p_i^2 \xrightarrow{a_i:c'_2} p_{i+1}^2, \cdots p_i^k \xrightarrow{a_i:c'_k} p_{i+1}^k \in \Delta$.*

The new position is $(q_{i+1}, (p_{i+1}^1, p_{i+1}^2, \cdots, p_{i+1}^k))$, from where round $(i+1)$ begins.

Thus, in a play of the k-token game on $\mathcal{A}$, Eve constructs a run and Adam k runs, all on the same word. Eve wins such a play if the following holds: if one of Adam's k runs is accepting, then Eve's run is accepting.

Observe that the stepahead-simulation game between $\mathcal{A}$ and itself is equivalent to the 1-token game on $\mathcal{A}$.

Joker games. *Joker games* are defined similar to 1-token games, but additionally in each round, Adam can choose to play *Joker* and choose a transition from Eve's position instead of a transition from his position. The winning condition for Eve is the following: If Adam's sequence of transitions satisfies the parity conditions and Adam has played finitely many Jokers, then Eve's run is accepting as well.

▶ **Definition 7** (Joker games). *The* Joker game *on a non-deterministic parity automaton $\mathcal{A} = (Q, \Sigma, q_0, \Delta)$ is defined between the players Adam and Eve as follows, with positions in $Q \times Q$. The game starts at (q_0, q_0) and proceeds in ω many rounds. For each $i \in \mathbb{N}$, the round i starts at a position $(q_i, p_i) \in Q \times Q$, and proceeds as follows.*

1. *Adam selects a letter $a_i \in \Sigma$.*
2. *Eve selects a transition $q_i \xrightarrow{a_i:c_i} q_{i+1} \in \Delta$*
3. *Adam either selects a transitions $p_i \xrightarrow{a_i:c'_i} p_{i+1}$, or plays Joker and selects a transition $q_i \xrightarrow{a_i:c'_i} p_{i+1}$.*

The new position is (q_{i+1}, p_{i+1}), from where round $(i+1)$ begins.

Eve wins such a play if the following holds: if Adam plays finitely many Jokers and his sequence of transitions satisfies the parity condition, then Eve's run is accepting.

The following observations are easy to see.

▶ **Lemma 8.** *If $\mathcal{A}$ is an HD parity automaton, and if $\mathcal{B}$ is a parity automaton such that $\mathcal{L}(\mathcal{B}) \subseteq \mathcal{L}(\mathcal{A})$, then we have:*

1. *Eve wins the Joker game on $\mathcal{A}$ and the k-token game on $\mathcal{A}$, for all $k \geq 1$;*
2. *$\mathcal{A}$ simulates and step-ahead simulates $\mathcal{B}$.*

Proof. Fix a winning strategy σ for Eve in the HD game of $\mathcal{A}$. Consider the strategy for Eve in the above games, in which she follows σ based on the letters Adam chooses, ignoring the rest of his moves. Then Eve constructs an accepting run whenever the word constructed by Adam is accepting. In particular, if Adam constructs an accepting run in the k-token game or the (stepahead) simulation game, then his word must be in $\mathcal{L}(\mathcal{A})$, implying Eve's run is accepting as well.

Similarly, in a play of the Joker game, suppose Adam plays finitely many Jokers and his sequence of transitions satisfies the parity condition. Let i be the last round where Adam played a Joker. Then there is an accepting run on Adam's word, which can be obtained by concatenating Eve's run until round $(i-1)$ with Adam's run from round i. This implies that Adam's word is in $\mathcal{L}(\mathcal{A})$, once again implying that Eve's run is accepting. ◀

2.4 History-Deterministic Automata and Simulation

Let $\mathcal{L}$ be an ω-regular language and let $\mathcal{F}_{\mathcal{L}}$ be the set of automata whose recognized languages are subsets of $\mathcal{L}$, that is $\mathcal{F}_{\mathcal{L}} = \{\mathcal{A} \mid \mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}\}$. For $\mathcal{A}, \mathcal{B} \in \mathcal{F}_{\mathcal{L}}$, we define $\mathcal{A} \preceq \mathcal{B}$ to hold if $\mathcal{B}$ simulates $\mathcal{A}$. This relation $\preceq$ is called the *simulation preorder* because it is reflexive and transitive. Lemma 8 implies that every HD automaton $\mathcal{H}$ that recognises $\mathcal{L}$ is a *greatest* element in $\mathcal{F}_{\mathcal{L}}$ with respect to the simulation preorder, that is, we have $\mathcal{A} \preceq \mathcal{H}$ for all $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{H})$. We show that the converse also holds.

▶ **Lemma 9.** *Let $\mathcal{L}$ be an ω-regular language and let $\mathcal{F}_{\mathcal{L}} = \{\mathcal{A} \mid \mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}\}$. Then, an automaton $\mathcal{H}$ in $\mathcal{F}_{\mathcal{L}}$ is greatest w.r.t. the simulation preorder if and only if $\mathcal{H}$ recognises $\mathcal{L}$ and it is history-deterministic.*

Proof. We only need to prove the forward implication, since the backward implication follows from Lemma 8. Let $\mathcal{H} \in \mathcal{F}_{\mathcal{L}}$ be such that $\mathcal{A} \preceq \mathcal{H}$ for all automata $\mathcal{A}$ that satisfy $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}$. Fix $\mathcal{D}$ to be a deterministic parity automaton that recognises $\mathcal{L}$ (such a $\mathcal{D}$ always exists, see [9, Theorem 1.19 and 3.11]). Then, in particular, we have $\mathcal{D} \preceq \mathcal{H}$. Observe that this implies $\mathcal{L}(\mathcal{H}) \supseteq \mathcal{L}(\mathcal{D}) = \mathcal{L}$, and since $\mathcal{L}(\mathcal{H}) \subseteq \mathcal{L}$, we get that $\mathcal{L}(\mathcal{H}) = \mathcal{L}$.

We proceed to show that Eve wins the HD game on $\mathcal{H}$. Fix σ to be a winning strategy for Eve in the simulation game between $\mathcal{H}$ and $\mathcal{D}$. We use σ to construct a winning strategy in the HD game on $\mathcal{H}$ as follows. During the letter game on $\mathcal{H}$, Eve keeps a corresponding play of the simulation game between $\mathcal{H}$ and $\mathcal{D}$, where Adam is playing the same letters as the HD game and choosing the unique transitions available to him. Then Eve chooses transitions according to σ in the HD game and in the simulation game in her memory. This way, whenever Adam's word w in the HD game is in $\mathcal{L}(\mathcal{H}) = \mathcal{L}$, then the unique run in $\mathcal{D}$ on w is accepting, and hence, Eve's run in the HD game on $\mathcal{H}$ must be accepting as well. ◀

We make explicit a corollary of the above lemma.

▶ **Corollary 10** ([11, Theorem 4])**.** *If a nondeterministic parity automaton $\mathcal{A}$ simulates a language-equivalent history-deterministic automaton $\mathcal{H}$ then $\mathcal{A}$ is history-deterministic.*

Proof. Let $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{H}) = \mathcal{L}$. From Lemma 9, we know that $\mathcal{H}$ is a greatest element in $\mathcal{F}_{\mathcal{L}}$, and $\mathcal{H} \preceq \mathcal{A}$ implies that $\mathcal{A}$ is a greatest element in $\mathcal{F}_{\mathcal{L}}$ as well. It follows from Lemma 9 that $\mathcal{A}$ is history-deterministic. ◀

3 Sufficient to think about Universal Automata

In the next section, we will show that 1-token games characterise history-determinism on semantically-deterministic Büchi automata (Theorem B). In order to show this, we start by reducing this result to the restriction where our automata are universal (Theorem 11), i.e., recognise all words in the language. A very similar reduction also shows that proving the 2-token conjecture for universal parity is sufficient to conclude the 2-token conjecture for parity automata (Theorem 19).

▶ **Theorem 11.** *The following statements are equivalent:*

1. *For any semantically-deterministic Büchi automaton $\mathcal{A}$, Eve wins the 1-token game on $\mathcal{A}$ if and only if $\mathcal{A}$ is history-deterministic.*
2. *For any semantically-deterministic Büchi automaton $\mathcal{U}$ with $\mathcal{L}(\mathcal{U}) = \Sigma^\omega$, Eve wins the 1-token game on $\mathcal{U}$ if and only if $\mathcal{U}$ is history-deterministic.*

We shall use the following fact shown by Boker, Henzinger, Lehtinen, and Prakash to prove Theorem 11 [4].

▶ **Lemma 12** ([4])**.** *A non-deterministic parity automaton $\mathcal{A}$ is history-deterministic if and only if it simulates all deterministic safety automata $\mathcal{S}$ with $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{A})$.*

A proof of Lemma 12 can be found in the full version [2]. We now show Theorem 11.

Proof sketch for Theorem 11. It is clear that 1 implies 2. For the other direction, suppose 2 holds. Let $\mathcal{A}$ be a semantically-deterministic automaton that is not HD. We will show that Adam wins the 1-token game on $\mathcal{A}$.

From Lemma 12, we know that there is a deterministic safety automaton $\mathcal{S}$ such that $\mathcal{A}$ does not simulate $\mathcal{S}$ and $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{A})$. Consider the product safety automaton $\mathcal{P}$ of $\mathcal{S}$ and $\mathcal{A}$ which recognises the language $\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{S})$. We then complete $\mathcal{P}$ by adding an accepting sink state f, and transitions to f from all states q on letters a such that q did not have an outgoing transition on a in $\mathcal{P}$. We call this automaton $\mathcal{U}$. It is clear that $\mathcal{L}(\mathcal{U}, p) = \Sigma^\omega$ for all states p in $\mathcal{U}$, and hence $\mathcal{U}$ is SD. We show that Adam wins the HD game on $\mathcal{U}$, by using his winning strategy in the simulation game between $\mathcal{A}$ and $\mathcal{S}$ (recall that $\mathcal{P}$ was constructed by taking product of $\mathcal{S}$ and $\mathcal{A}$). The hypothesis implies that Adam wins the 1-token game on $\mathcal{U}$. We then show that we can adapt a winning strategy for Adam on 1-token game of $\mathcal{U}$ to one for the 1-token game on $\mathcal{A}$, by simply "projecting" his strategy to the $\mathcal{A}$ component: note that since $\mathcal{S}$ is deterministic, in plays of the 1-token game on $\mathcal{U}$, Eve's and Adam's states have the same $\mathcal{S}$-component at the start of each round. ◀

An almost word-by-word identical proof to above also shows that the 2-token conjecture can be reduced to the case where the automata are universal.

▶ **Theorem 13.** *The following statements are equivalent:*

1. *For any non-deterministic parity automaton $\mathcal{A}$, Eve wins the 2-token game on $\mathcal{A}$ if and only if $\mathcal{A}$ is history-deterministic.*
2. *For any non-deterministic parity automaton $\mathcal{U}$ with $\mathcal{L}(\mathcal{U}) = \Sigma^\omega$, Eve wins the 2-token game on $\mathcal{U}$ if and only if $\mathcal{U}$ is history-deterministic.*

4 When 1-Token Game is Enough

In this section, we will show the following result.

▶ **Theorem B.** *A semantically-deterministic Büchi automaton is history-determinsitic if and only if Eve wins the* 1*-token game on it.*

Towards this, we first introduce k-lookahead games, which are variants of 1-token games where Adam is given a lookahead of k.

4.1 Lookahead Games

Let us briefly recall how a round of the 1-token game on a parity automaton $\mathcal{A}$ works. In each round, Adam selects a letter, then Eve selects a transition on that letter on her token, and then Adam selects a transition on that letter on his token. The winning condition for Eve is that at the end of the play, either Eve's run is accepting or Adam's run is rejecting. This is very close to the simulation game between $\mathcal{A}$ and itself, except that the order of the moves in which Eve and Adam select transitions has been reversed. One can, however, see the 1-token game as a simulation game, where Adam picks the transition for round i in round $(i+1)$. Or equivalently, we can construct an automaton $\mathsf{Delay}(\mathcal{A})$ such that any non-determinism on $\mathcal{A}$ is "delayed" by one step, and then the 1-token game on $\mathcal{A}$ is equivalent to the simulation game between $\mathcal{A}$ and $\mathsf{Delay}(\mathcal{A})$. This insight was used by Prakash and Thejaswini to give an algorithm for deciding history-determinism of one-counter nets, by reducing the 1-token game to a simulation game [19]. Below we give a construction Delay on parity automata that delays the non-determinism by one-step, inspired by a similar construction for one-counter nets [19, Lemma 11].

▶ **Definition 14.** *For any non-deterministic parity automaton $\mathcal{A} = (Q, \Sigma, q_0, \Delta)$, the automaton* $\mathsf{Delay}(\mathcal{A})$ *is constructed so that it runs "one letter behind" $\mathcal{A}$, by storing a letter in its state space. More formally,* $\mathsf{Delay}(\mathcal{A}) = (Q', \Sigma, s, \Delta')$*, where $Q' = Q \times \Sigma \cup \{s\}$, and s is the initial state. The set of transitions Δ' is the union of the following sets of transitions.*

1. $\{(s \xrightarrow{a:0} (q_0, a)) \mid a \in \Sigma\}$.
2. $\{((p, a) \xrightarrow{a':c} (q, a')) \mid \ (p \xrightarrow{a:c} q) \in \Delta\}$.

Observe that $\mathsf{Delay}(\mathcal{A})$ accepts the same language as $\mathcal{A}$. The following lemma is easy to prove, since the expanded game arenas of the 1-token game on $\mathcal{A}$, and the simulation game between $\mathcal{A}$ and $\mathsf{Delay}(\mathcal{A})$ are equivalent, with identical winning conditions.

▶ **Lemma 15.** *For every non-deterministic parity automaton $\mathcal{A}$, Eve wins the 1-token game on $\mathcal{A}$ if and only if $\mathcal{A}$ simulates* $\mathsf{Delay}(\mathcal{A})$*.*

Furthermore, we can also show that Eve wins the 1-token game on $\mathsf{Delay}(\mathcal{A})$ if Eve wins the 1-token game on $\mathcal{A}$, by simply "delaying" her winning strategy in the 1-token game on $\mathcal{A}$.

▶ **Lemma 16.** *If Eve wins the 1-token game on an automaton $\mathcal{A}$, then Eve wins the 1-token game on* $\mathsf{Delay}(\mathcal{A})$*.*

An iterative application of Lemmas 15 and 16 gives us the following corollary.

▶ **Corollary 17.** *If Eve wins the 1-token game on a parity automaton $\mathcal{A}$, then $\mathcal{A}$ simulates* $\mathsf{Delay}^k(\mathcal{A})$ *for all $k \in \mathbb{N}$.*

Proof. Note that simulation relation is transitive, i.e., if $\mathcal{A}_0$ simulates $\mathcal{A}_1$ and $\mathcal{A}_1$ simulates $\mathcal{A}_2$, then $\mathcal{A}_0$ simulates $\mathcal{A}_2$. Suppose Eve wins the 1-token game on $\mathcal{A}$. From Lemma 16, induction gives us that Eve wins the 1-token game on $\mathsf{Delay}^k(\mathcal{A})$ for all $k \in \mathbb{N}$. From Lemma 15, we see that $\mathsf{Delay}^k(\mathcal{A})$ simulates $\mathsf{Delay}^{k+1}(\mathcal{A})$ for all $k \in \mathbb{N}$. Combining this with transitivity of simulation, we get that $\mathcal{A}$ simulates $\mathsf{Delay}^k(\mathcal{A})$ for all $k \in \mathbb{N}$. ◀

Call the simulation game between $\mathcal{A}$ and $\mathsf{Delay}^k(\mathcal{A})$ as the *k-lookahead game* on $\mathcal{A}$. Note that the 1-token game of $\mathcal{A}$ is then equivalent to the 1-lookahead game of $\mathcal{A}$. Corollary 17 can thus be restated as the following theorem.

▶ **Theorem A.** *For every parity automaton $\mathcal{A}$, Eve wins the 1-lookahead game on $\mathcal{A}$ if and only if she wins the k-lookahead game on $\mathcal{A}$.*

4.2 Games to Characterise History-Determinism

We now proceed to show Theorem B. From Theorem 11, we know that it suffices to only consider SD Büchi automata that are universal. The following lemma shows that every universal SD Büchi automaton is history-deterministic with sufficient lookahead.

▶ **Lemma 18.** *Let $\mathcal{U}$ be a semantically-deterministic Büchi automata such that $\mathcal{L}(\mathcal{U}) = \Sigma^\omega$. Then, there is a K such that* $\mathsf{Delay}^K(\mathcal{U})$ *is history-deterministic.*

Proof sketch. We let $K = 2^n$, where n is the number of states of $\mathcal{A}$. The crucial observation is that since $\mathcal{L}(\mathcal{U}, q) = \Sigma^\omega$ for any state q in $\mathcal{U}$, any finite word u that has length at least 2^n must have a run from q that passes through an accepting transition. Eve thus wins the history-determinism game on $\mathsf{Delay}^K(\mathcal{U})$ by exploiting the lookahead of K to take at least one accepting transition every K steps. The run of Eve's token then has infinitely many accepting transitions and hence is accepting, as desired. ◀

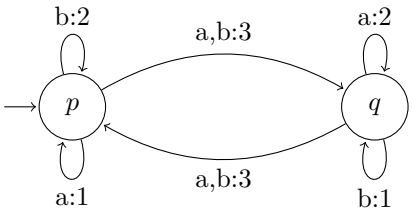


Figure 1 An [1,3]-automaton $\mathcal{A}$ that is not HD but on which Eve wins the Joker game.

We can now prove Theorem B.

Proof of Theorem B. The forward implication is clear by Lemma 8. For the backward direction, suppose that Eve wins the 1-token game on $\mathcal{A}$. Due to Theorem 11, we may assume that $\mathcal{A}$ is universal. Then, from Lemma 18, we know that there is a K such that $\mathsf{Delay}^K(\mathcal{A})$ is history-deterministic. If Eve wins the 1-token game on $\mathcal{A}$, then from Lemma 15, we know that $\mathcal{A}$ simulates $\mathsf{Delay}^K(\mathcal{A})$. But since $\mathsf{Delay}^K(\mathcal{A})$ is language equivalent to $\mathcal{A}$, we get from Corollary 10 that $\mathcal{A}$ is HD as well. ◀

Having shown that the 1-token game on a semantically-deterministic Büchi automaton $\mathcal{A}$ is equivalent to the HD game on $\mathcal{A}$, we are able to show that Joker games characterise history-determinism on Büchi automata. We also get an alternate proof of Bagnol and Kuperberg's result of 2-token games characterising history-determinism of Büchi automata as a corollary.

▶ **Theorem 19.** *For every Büchi automaton $\mathcal{A}$, the following statements are equivalent.*

1. *$\mathcal{A}$ is history-deterministic.*
2. *Eve wins the Joker game on $\mathcal{A}$.*
3. *Eve wins the 2-token game on $\mathcal{A}$.*

We prove Theorem 19 by reducing it to SD automata [2, Lemma 33], similar to Bagnol and Kuperberg [3, Lemma 16].

Joker games on a Büchi automaton have smaller arenas than 2-token games. As a result, we get a more efficient algorithm to recognise HD Büchi automata.

▶ **Lemma 20.** *Given a non-deterministic Büchi automaton $\mathcal{A} = (Q, \Sigma, q_0, \Delta)$, we can decide whether it is history-deterministic in time $\mathcal{O}(|\Sigma|^2|Q|^3|\Delta|)$.*

▶ Remark 21. While we have proven Theorem B by reducing it to universal automata for the sake of simplicity of arguments, one can also give a more direct proof. Such a direct proof involves arguing that if an automaton $\mathcal{A}$ is not history-deterministic, then there is a K exponential in the size of $\mathcal{A}$ such that $\mathcal{A}$ does not simulate $\mathsf{Delay}^K(\mathcal{A})$. It then follows from Lemma 15 that Eve loses the 1-token game on $\mathcal{A}$, as desired.

To argue that $\mathcal{A}$ does not simulate $\mathsf{Delay}^K(\mathcal{A})$, we reason based on the size of Adam's strategy in the history-determinism game on $\mathcal{A}$. Adam, in the simulation game between $\mathcal{A}$ and $\mathsf{Delay}^K(\mathcal{A})$, can pick letters according to his strategy in the history-determinism game on $\mathcal{A}$, thus ensuring Eve produces a rejecting run on her token. At the same time, Adam can exploit the lookahead in $\mathsf{Delay}^K(\mathcal{A})$ to construct an accepting run on his token, thus winning the simulation game between $\mathcal{A}$ and $\mathsf{Delay}^K(\mathcal{A})$.

We end this section by showing that unlike Büchi automata, Joker games do not characterise history-determinism on parity automata.

▶ **Example 22.** Consider the $[1,3]$-automaton $\mathcal{A}$ shown in Figure 1. Note that $\mathcal{A}$ accepts all words in $\{a,b\}^\omega$, and is SD. It is easy to see that the automaton $\mathcal{A}$ is not history-deterministic, since Adam can win the history-determinism game on $\mathcal{A}$ by choosing the letter a when Eve's token is at p, and b when her token is at q. This forces Eve to never see an even transition, causing her run to be rejecting.

Eve wins the Joker game on $\mathcal{A}$, however. Consider the strategy of Eve where she switches states if her and Adam's tokens are at different states, and otherwise stays at the same state. For Adam to win, Adam must play only finitely many Jokers, and construct an accepting run, which requires him to eventually stay in the same state. But then, Eve's strategy will ensure that Eve's and Adam's run are eventually identical, thus ensuring Adam cannot win the Joker game of $\mathcal{A}$.

▶ **Theorem 23.** *Joker games do not characterise history-determinism on (semantically-deterministic) parity automata.*

5 Determinising HD Büchi Automata in Polynomial Time

In this section, we present a polynomial time determinisation procedure for HD Büchi automata with only a quadratic state-space blowup. Our procedure combines ideas from the exponential-time procedure given by Kuperberg and Skrzypczak [14] with our 1-token game characterisation of history-determinism on SD Büchi automata.

▶ **Theorem C.** *There is a polynomial-time procedure that converts every HD Büchi automaton with n states into an equivalent deterministic Büchi automaton with n^2 states.*

Let $\mathcal{H}$ be an HD Büchi automaton. Kuperberg and Skrzypczak's procedure relies on carefully analysing the simulation game between $\mathcal{H}$ and an equivalent deterministic Büchi automaton $\mathcal{D}$ whose states are the subsets of $\mathcal{H}$. Their procedure iteratively makes modifications to $\mathcal{H}$ and $\mathcal{D}$ based on the structure of Eve's winning strategies in this game or, more precisely, the progress measures [12] or the signatures [23] of the vertices in this game, which they call ranks [14]. The end result of their procedure is a relation between states of $\mathcal{H}$ and states of $\mathcal{D}$ that correspond to sets containing a singleton in $\mathcal{H}$. A product construction based on this relation allows them to construct an equivalent determinsitic Büchi automaton whose number of states is quadratic in the number of states of $\mathcal{A}$.

The reason Kuperberg and Skrzypczak work with the simulation game between $\mathcal{H}$ and $\mathcal{D}$ is that it characterises the history-determinism of $\mathcal{H}$. A non-deterministic automaton is HD if and only if it simulates an equivalent deterministic one, as shown by Hezinger and Piterman [11, Theorem 4]. Our result on 1-token games characterising history-determinism (Theorem B) allows us to instead work with the 1-token game on $\mathcal{H}$. This results in a conceptually simpler procedure that works in polynomial time.

We present our algorithm in two steps. First, we introduce HD Büchi automata that have a sprint self-simulation, and we give a polynomial time determinisation procedure for them. The procedure involves a quadratic state-space blowup. Then, we give a polynomial time iterative procedure to transform $\mathcal{H}$ into an equivalent HD Büchi automaton of the same size that has a sprint self-simulation. Overall, this gives us a polynomial time procedure to determinise HD Büchi automata with a quadratic state-space blowup.

5.1 Determinising Automata with Sprint Self-Simulation

In this subsection, we describe what automata with sprint self-simulation are and give a polynomial time determinisation procedure for such HD Büchi automata. The key concept here is the *sprint simulation* relation between two Büchi automata, characterised by the *sprint*

step-ahead simulation game. This game is similar to the step-ahead simulation game, but the winning condition for Eve is that she must see an accepting transition before Adam does, i.e., she is in a sprint with Adam to see an accepting transition in the step-ahead simulation game first. Sprint step-ahead simulation (game) would be a more accurate phrase, but for brevity, we shorten it to just sprint simulation (game).

▶ **Definition 24.** *For two non-deterministic Büchi automata* $\mathcal{A} = (Q, \Sigma, q_0, \Delta_A)$ *and* $\mathcal{B} = (P, \Sigma, p_0, \Delta_B)$*, the* sprint simulation game *between* $\mathcal{A}$ *and* $\mathcal{B}$ *is played on the set of positions* $Q \times P$ *and it proceeds in rounds. In each round* $i = 0, 1, 2, \ldots$*, from position* (q_i, p_i)*, the two players Adam and Eve make the following choices:*

1. *Adam chooses a letter* $a_i \in \Sigma$*;*
2. *Eve chooses a transition* $\delta_i = (q_i \xrightarrow{a_i:c_i} q_{i+1}) \in \Delta_A$*;*
3. *Adam chooses a transition* $\delta'_i = (p_i \xrightarrow{a_i:c'_i} p_{i+1}) \in \Delta_B$*.*

The new position is (q_{i+1}, p_{i+1})*. At every round* i*, if transition* δ_i *is accepting then Eve wins the game, and otherwise, if the transition* δ'_i *is accepting then Adam wins the game. If neither* δ_i *nor* δ'_i *are accepting transitions, then the game continues for another round. Eve wins every infinite play.*

Observe that if Eve wins the sprint simulation game then she can do so by a positional strategy, because the objective for Eve is a disjunction of a safety and a reachability objective, which can be seen as a $[0, 1]$-parity game. If Eve has a winning strategy in the above game, we say that $\mathcal{A}$ sprint simulates $\mathcal{B}$. The sprint simulation relation is transitive, i.e., if $\mathcal{A}$ sprint simulates $\mathcal{B}$ and $\mathcal{B}$ sprint simulates $\mathcal{C}$, then $\mathcal{A}$ sprint simulates $\mathcal{C}$ [2, Lemma 36].

▶ Remark 25. The sprint simulation relation is similar to the 'dependency' relation introduced by Kuperberg and Skrzypczak [14, Definition 30]. While the sprint simulation relation is between two Büchi automata, dependency relation is derived from the sprint simulation game between a Büchi automaton and an equivalent exponential-sized deterministic Büchi automaton.

We say that an HD Büchi automaton $\mathcal{H}$ has a *sprint self-simulation* if it is semantically-deterministic and for every state p in $\mathcal{H}$, there is a language-equivalent state q, such that $(\mathcal{H}, p)$ sprint simulates $(\mathcal{H}, q)$. When $\mathcal{H}$ is clear from the context, we will just say that p sprint simulates q. For the rest of this subsection, fix $\mathcal{H} = (Q, \Sigma, q_0, \Delta)$ to be an HD Büchi automaton that has a sprint self-simulation. The following lemma follows from transitivity of sprint simulation.

▶ **Lemma 26.** *For every state* p *in* $\mathcal{H}$*, there is a language-equivalent state* q *such that* p *sprint simulates* q *and* q *sprint simulates itself.*

Proof. Fix a state p in $\mathcal{H}$. Then, there is a language-equivalent state q_0 in $\mathcal{H}$ such that p sprint simulates q_0. If q_0 does not sprint simulates itself, there is another language-equivalent q_1 such that q_0 sprint simulates q_1. Repeating this argument, we get a sequence of states $q_0, q_1, q_2, \ldots$, such that q_i sprint simulates q_{i+1}. Since, there are finitely many states in $\mathcal{H}$, there are two natural numbers $i < j$ such that $q_i = q_j$. Due to transitivity of sprint simulation [2, Lemma 36], it follows that p sprint simulates q_i and q_i sprint simulates itself, as desired. ◀

Let us call a state q *sprint deterministic* if the automaton $\mathcal{H}$ can be determinised by deleting transitions to get a deterministic subautomaton $\mathcal{F}_q$ so that the following holds: for all finite words w that have a run in $\mathcal{H}$ starting at q going through an accepting transition,

the unique run on w in $\mathcal{F}_q$ from q also sees an accepting transition. Thus, for states q that are sprint deterministic, there is a uniform strategy that achieves the objective of seeing an accepting transition as soon as possible on all words. We say that the automaton $\mathcal{F}_q$ as above is a witness for sprint determinism of q.

▶ **Lemma 27.** *A state q in $\mathcal{H}$ is sprint deterministic if and only if q sprint simulates itself. Moreover, there is a deterministic subautomaton $\mathcal{F}$ that can be computed in polynomial time and is a witness for all sprint deterministic states.*

Lemmas 26 and 27 above tell us that every state in $\mathcal{H}$ is either sprint deterministic, or it sprint simulates a state that is sprint deterministic. Fix a subautomaton $\mathcal{F}$ from Lemma 27, and a positional Eve strategy τ in the step-ahead simulation game from (p, q) for all pairs of states (p, q) such that p sprint simulates q.

The deterministic automaton $\mathcal{D}$ that is language-equivalent to $\mathcal{H}$ is then constructed to consist of pairs of states (p, q) such that p and q are language equivalent, p sprint simulates q, and q is sprint deterministic. Note that the pair of such states can be found in polynomial time, since checking for language containment on HD Büchi automata [18, Corollary 17] and deciding the winner of sprint simulation game can be done in polynomial time. Furthermore, for each state p in $\mathcal{H}$, we know from Lemmas 26 and 27 that there is a state q in $\mathcal{H}$ such that (p, q) is a state in $\mathcal{D}$. We let the initial state d_0 be (q_0, r_0) for some r_0 such that $(q_0, r_0) \in \mathcal{D}$.

At a state (q, p), the transitions in $\mathcal{D}$ from the second component p are chosen according to transitions from $\mathcal{F}$, while transitions from q are chosen via the positional Eve strategy τ. When an accepting transition $q \xrightarrow{a:2} q'$ is taken on the first component, we update the second component deterministically to be p' such that (q', p') is a state in $\mathcal{D}$. Or, equivalently, q' and p' are language equivalent, q' sprint simulates p', and p' is sprint deterministic. The priorities of transitions in $\mathcal{D}$ are the priorities of transitions of the first component.

We show the correctness of our construction using the definition of sprint simulation game and the fact that $\mathcal{H}$ is semantically-deterministic.

▶ **Lemma 28.** *The automaton $\mathcal{D}$ accepts the same language as $\mathcal{H}$.*

Proof. $\mathcal{L}(\mathcal{D}) \subseteq \mathcal{L}(\mathcal{H})$: If ρ is an accepting run of a word w in $\mathcal{D}$, then the projection of ρ on the first component is an accepting run in $\mathcal{H}$ as well.

$\mathcal{L}(\mathcal{H}) \subseteq \mathcal{L}(\mathcal{D})$: We show that for each state (p, q) in $\mathcal{D}$, if $w \in \mathcal{L}(\mathcal{H}, p)$ then the run from (p, q) on w in $\mathcal{D}$ sees an accepting transition eventually. We can then conclude by induction and the semantic determinism of $\mathcal{H}$ that runs of $\mathcal{D}$ on all accepting words in $\mathcal{L}(\mathcal{H})$ contain infinitely many accepting transitions each, and hence are accepting. To see this, let $w \in \mathcal{L}(\mathcal{H}, p)$ and let ρ_D be the run of $\mathcal{D}$ on w from (p, q). By construction of $\mathcal{D}$, we know that q is language-equivalent to p. Since q is sprint deterministic, the second component of ρ_D in $\mathcal{D}$ must contain an accepting transition on w eventually. But since p sprint simulates q, the run on the first component of ρ_D contains an accepting transition as well, as desired. ◀

5.2 Towards Automata with Sprint Self-Simulation

We now present a polynomial time algorithm to convert an HD Büchi automaton into an equivalent HD Büchi automaton that has a sprint self-simulation. Throughout this subsection, let $\mathcal{H} = (Q, \Sigma, q_0, \Delta)$ be an HD Büchi automaton.

We say that $\mathcal{H}$ is *good* if $\mathcal{H}$ is semantically-deterministic and Eve wins the Joker game from all states in $\mathcal{H}$. Every HD Büchi automaton $\mathcal{H}$ can be converted to an equivalent good HD Büchi automaton in polynomial time: we fix a winning strategy τ for Eve on the Joker game on $\mathcal{H}$, and consider the subautomaton $\mathcal{H}_N$ consisting of transitions that Eve takes according to τ [2, Lemma 38]. We thus assume without loss of generality that $\mathcal{H}$ is good.

To get an HD Büchi automaton equivalent to $\mathcal{H}$ and that has a sprint self-simulation, we iteratively make modifications to $\mathcal{H}$ based on the ranks of the 1-token game on $\mathcal{H}$. We first give a description of the 1-token game on a Büchi automaton as a $[0,2]$-parity game, and briefly recall the properties of ranks that we need on such games.

▶ **Definition 29.** *For a semantically-deterministic Büchi automaton $\mathcal{B} = (Q, \Sigma, q_0, \Delta)$, define the $[0,2]$-parity game $G_1(\mathcal{B}) = (V, E)$ as follows:*

- *The set of vertices V consists of the set $V = V_1 \cup V_2 \cup V_3$, where:*
  1. $V_1 = \{(p,q) \mid$ *p, q are states reachable from q_0 upon reading the same word $w\}$*
  2. $V_2 = \{(p,a,q) \mid (p,q) \in V_1\}$
  3. $V_3 = \{(p',q,a) \mid (p,a,q) \in V_2$ *and* $p \xrightarrow{a} p' \in \Delta\}$

  Eve's vertices are $V_\exists = V_2$, while Adam's vertices are $V_\forall = V_1 \cup V_3$
- *The set of edges E is the union of following sets:*
  1. $E_1 = \{(p,q) \to (p,a,q) \mid a \in \Sigma\}$ *(Adam chooses a letter)*
  2. $E_2 = \{(p,a,q) \to (p',q,a) \mid p \xrightarrow{a:c} p' \in \Delta\}$ *(Eve chooses a transition on her token)*
  3. $E_3 = \{(p',q,a) \to (p',q') \mid q \xrightarrow{a:c} q' \in \Delta\}$ *(Adam chooses a transition on his token)*
- *The priority function Ω is defined as follows. All elements in E_1 are assigned priority 0, while edges $(p,a,q) \to (p',q,a)$ in E_2 are assigned priority 2 if the transition $\delta = p \xrightarrow{a:c} p'$ in $\mathcal{B}$ is accepting (or equivalently, $c = 2$), and 0 otherwise. The edge $(p',q,a) \to (p',q')$ in E_3 is assigned priority 1 if the transition $q \xrightarrow{a:c} q'$ is accepting, and 0 otherwise.*

Observe that since $\mathcal{H}$ is SD, we have $\mathcal{L}(\mathcal{H}, p) = \mathcal{L}(\mathcal{H}, q)$ if (p,q) is a vertex in $G_1(\mathcal{H})$.

Ranks. We now define the ranks of a $[0,2]$-parity game $\mathcal{G}$. For each vertex v in $\mathcal{G}$, $\mathrm{rank}(v)$ is the largest number of 1's that Adam can guarantee Eve will see before seeing a 2 in the play (or only 0's) starting from v.

Observe that Eve wins such a parity game from every position if and only if the ranks of all the vertices are bounded. If this is the case, then there is a positional winning strategy τ, using which Eve can guarantee that she sees at most $\mathrm{rank}(v)$ many 1's before seeing a 2 (or seeing 0's forever) [23, Lemma 8] in every play. We will call such a strategy *optimal*. For $[0,2]$-parity games with n vertices and m edges, an optimal strategy can be computed in time $\mathcal{O}(mn)$ [12, Theorem 11].

The following property of ranks follows from their definition.

▶ **Lemma 30.** *Let $\mathcal{G}$ be a $[0,2]$-parity game, and let $v \xrightarrow{e} u$ be an edge in $\mathcal{G}$, such that either v belongs to Adam, or the edge e is prescribed by Eve's optimal strategy τ. Then the edge e has priority 2 or $\mathrm{rank}(v) \geq \mathrm{rank}(u)$. Furthermore, this inequality is strict if e has priority 1.*

Consider the 1-token game $G_1(\mathcal{H})$, and the ranks of its vertices. Note that for a vertex (q,p) in $G_1(\mathcal{H})$, we have $\mathrm{rank}(q,p) = 0$ if and only if q sprint simulates p. Define, for each state $q \in \mathcal{H}$, its optimal rank $\mathrm{opt}(q)$ to be the minimum rank of a vertex of the form (q,p) in $G_1(\mathcal{H})$. Note that if $\mathrm{opt}(q) = 0$ for all states $q \in \mathcal{H}$, then $\mathcal{H}$ has a sprint self-simulation. Thus, our iterative procedure focuses on reducing the optimal ranks for all states until they are all 0. We describe this procedure below.

Iterating towards a sprint self-simulation. Set $\mathcal{H}_0 = \mathcal{H}$. For each $i \geq 0$, perform the following three steps on $\mathcal{H}_i$ until $\mathcal{H}_{i+1} = \mathcal{H}_i$.

1. For all vertices $(p,q) \in G_1(\mathcal{H}_i)$, compute the optimal ranks $\mathrm{opt}_i(p)$ in $G_1(\mathcal{H}_i)$.
2. Obtain $\mathcal{H}'_i$ from $\mathcal{H}_i$ by removing all transitions $q \xrightarrow{a:1} q'$ with $\mathrm{opt}_i(q) < \mathrm{opt}_i(q')$.
3. Obtain $\mathcal{H}_{i+1}$ from $\mathcal{H}'_i$, by making all transitions $q \xrightarrow{a:1} q'$ with $\mathrm{opt}_i(q) > \mathrm{opt}_i(q')$ accepting.

We show that for each i, both $\mathcal{H}'_i$ and $\mathcal{H}_{i+1}$ are good HD Büchi automata that are equivalent to $\mathcal{H}_i$ [2, Lemmas 39 and 40]. By a simple induction, we get that each $\mathcal{H}_i$ for $i \geq 0$ is a good HD Büchi automaton equivalent to $\mathcal{H}$.

Note that in steps 2 and 3, we are either removing rejecting transitions or making rejecting transitions accepting, and hence this procedure terminates after at most $|\Delta|$ iterations. Let $\mathcal{H}^*$ be the automaton obtained after the procedure terminates. Since $\mathcal{H}^*$ is good, we know that it is SD. The next lemma thus shows that $\mathcal{H}^*$ has a sprint self-simulation.

▶ **Lemma 31.** *For all states p in $\mathcal{H}^*$, there is a language-equivalent state q in $\mathcal{H}^*$ such that p sprint simulates q.*

Proof. Assume, to the contrary, that there exists a state p such that $\text{opt}^*(p) = \text{rank}^*(p, q) > 0$. Fix an optimal winning strategy τ for Eve in $G_1(\mathcal{H}^*)$. Consider a finite play ρ of $G_1(\mathcal{H}^*)$ from (p, q) where Eve is playing according to τ, Adam chooses an accepting transition in his token at some point, while Eve is unable to. Then by monotonicity of ranks (Lemma 30), we know that rank^* strictly decreases across ρ at some point. Then, there must be a rejecting transition across which the quantity opt^* decreases as well. But since such transitions would have been made accepting in step 3 of the iteration, we get a contradiction. ◀

From the polynomial time determinisation construction for HD Büchi automata that have a sprint self-simulation presented in Section 5.1, we get a polynomial time determinisation procedure for HD Büchi automata. This concludes the proof of Theorem C.

6 Discussion

Our paper has shown two key results on HD Büchi automata: a 1-token game based characterisation of history-determinism for semantically-deterministic Büchi automata, and a polynomial time determinisation procedure. In the process of obtaining these results, we developed several novel techniques that we believe to be equally exciting and insightful. We finish by remarking some implications of our results and techniques, and natural future directions that our work points to.

Our first technique, presented in Section 3, reduces game based characterisations of history-determinism on parity automata to parity universal automata. But the history-determinism game on such an automaton is just a parity game, since Adam's word is always accepting. The 2-token conjecture thus reduces, by Theorem 13, to showing that this parity game is equivalent to the 2-token game. This seems easy enough to show at first glance, but it proves to be (unsurprisingly) difficult. This result also shows that the difficulty in proving or disproving the 2-token conjecture arises not from the language an automaton recognises, but rather from the structure of the automaton.

We also introduced lookahead games, and showed that k-lookahead games are equivalent to 1-token games for all $k \geq 1$ (Theorem A). This shows that the 1-token games are quite powerful, in the same sense that 2-token games are powerful due to them being equivalent to k-token games for all $k \geq 2$. While our 1-token game characterisation of history-determinism on SD Büchi automata does not extend to parity automata (Theorem 23), one can combine the two different approaches to give the 2-token game more power, both in the form of lookahead and more tokens. It would be interesting to consider such games to try extending the 2-token conjecture beyond Büchi and co-Büchi automata.

Our algorithm to determinise HD Büchi automata involves a quadratic blowup. However, we do not know whether this is tight. In fact, it is still open if HD Büchi automata are strictly more succinct than determinstic Büchi automata. Nevertheless, we are hopeful that our algorithm can offer some insights on how to make progress on this problem.

Let us end with a problem highlighted by Boker and Lehtinen in their recent survey [6, Section 6.3.2]. In all the existing game-based characterisations which are used to recognise HD automata efficiently, including ours (Theorem B), it is not clear how we can naturally convert a winning strategy for Eve from the 2-token game or the Joker game to a winning strategy in the HD game. Our algorithm to determinise HD Büchi automata, however, can be seen as one: starting with a winning strategy for Eve in the Joker game, we construct a strategy in the HD game that requires linear memory, thus obtaining a deterministic automaton of quadratic size. But the proof of correctness of our algorithm relies on Theorem B. Towards a more pure strategy-transfer argument, where ideally an algorithm for strategy transfer also proves a game-based characterisation of history-determinism, our algorithm comes tantalisingly close. Indeed, proving that the automaton constructed in Step 2 preserves the relevant invariants [2, Lemma 39] is the only place where we use the fact that we started with an HD automaton. We believe that trying to get rid of this assumption, in order to give an alternative strategy-transfer proof for the 1-token game characterisation of history-determinism on SD Büchi automata, could lead to crucial insights towards better understanding HD (Büchi) automata and token games.

References

1 Bader Abu Radi and Orna Kupferman. Minimization and Canonization of GFG Transition-Based Automata. *Logical Methods in Computer Science*, 18(3), 2022. doi:10.46298/lmcs-18(3:16)2022.

2 Rohan Acharya, Marcin Jurdziński, and Aditya Prakash. Lookahead Games and Efficient Determinisation of History-Deterministic Büchi Automata. *CoRR*, abs/2310.13498, 2024. doi:10.48550/arXiv.2310.13498.

3 Marc Bagnol and Denis Kuperberg. Büchi Good-for-Games Automata Are Efficiently Recognizable. In *Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018*, volume 122 of *LIPIcs*, pages 16:1–16:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.16.

4 Udi Boker, Thomas Henzinger, Karoliina Lehtinen, and Aditya Prakash. History-Determinism vs. Fair Simulation, To appear.

5 Udi Boker, Denis Kuperberg, Karoliina Lehtinen, and Michal Skrzypczak. On the Succinctness of Alternating Parity Good-for-Games Automata. *CoRR*, abs/2009.14437, 2020. arXiv:2009.14437.

6 Udi Boker and Karoliina Lehtinen. When a Little Nondeterminism Goes a Long Way: An Introduction to History-Determinism. *ACM SIGLOG News*, 10(1):24–51, 2023. doi:10.1145/3584676.3584682.

7 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding Parity Games in Quasi-polynomial Time. *SIAM J. Comput.*, 51(2):17–152, 2022. doi:10.1137/17m1145288.

8 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Generalized Parity Games. In *Foundations of Software Science and Computational Structures, 10th International Conference, FOSSACS 2007*, volume 4423 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2007. doi:10.1007/978-3-540-71389-0_12.

9 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.

10 Yuri Gurevich and Leo Harrington. Trees, Automata, and Games. In *Symposium on Theory of Computing, STOC 1982*, pages 60–65. ACM, 1982. doi:10.1145/800070.802177.

11 Thomas A. Henzinger and Nir Piterman. Solving Games Without Determinization. In *Computer Science Logic, CSL 2006*, volume 4207 of *Lecture Notes in Computer Science*, pages 395–410. Springer, 2006. doi:10.1007/11874683_26.

12 Marcin Jurdzinski. Small Progress Measures for Solving Parity Games. In *STACS 2000*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2000. doi:10.1007/3-540-46541-3_24.

13 Marcin Jurdzinski and Ranko Lazic. Succinct Progress Measures for Solving Parity Games. In *Symposium on Logic in Computer Science, LICS 2017*, pages 1–9. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005092.

14 Denis Kuperberg and Michal Skrzypczak. On Determinisation of Good-for-Games Automata. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015*, volume 9135 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2015. doi:10.1007/978-3-662-47666-6_24.

15 Orna Kupferman. Using the Past for Resolving the Future. *Frontiers Comput. Sci.*, 4, 2022. doi:10.3389/fcomp.2022.1114625.

16 Karoliina Lehtinen, Pawel Parys, Sven Schewe, and Dominik Wojtczak. A Recursive Approach to Solving Parity Games in Quasipolynomial Time. *Log. Methods Comput. Sci.*, 18(1), 2022. doi:10.46298/lmcs-18(1:8)2022.

17 Donald A. Martin. Borel Determinacy. *Annals of Mathematics*, 102(2):363–371, 1975. URL: http://www.jstor.org/stable/1971035.

18 Aditya Prakash. Checking History-Determinism is NP-hard for Parity Automata. In *Foundations of Software Science and Computation Structures - 27th International Conference, FoSSaCS 2024*, volume 14574 of *Lecture Notes in Computer Science*, pages 212–233. Springer, 2024. doi:10.1007/978-3-031-57228-9_11.

19 Aditya Prakash and K. S. Thejaswini. On History-Deterministic One-Counter Nets. In *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023*, volume 13992 of *Lecture Notes in Computer Science*, pages 218–239. Springer, 2023. doi:10.1007/978-3-031-30829-1_11.

20 Bader Abu Radi and Orna Kupferman. On Semantically-Deterministic Automata. In *International Colloquium on Automata, Languages, and Programming, ICALP 2023*, volume 261 of *LIPIcs*, pages 109:1–109:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.109.

21 Bader Abu Radi, Orna Kupferman, and Ofer Leshkowitz. A Hierarchy of Nondeterminism. In *Mathematical Foundations of Computer Science, MFCS 2021*, volume 202 of *LIPIcs*, pages 85:1–85:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.85.

22 Sven Schewe. Minimising Good-For-Games Automata Is NP-Complete. In *Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020*, volume 182 of *LIPIcs*, pages 56:1–56:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.FSTTCS.2020.56.

23 Igor Walukiewicz. Monadic Second-Order Logic on Tree-like Structures. *Theor. Comput. Sci.*, 275(1-2):311–346, 2002. doi:10.1016/S0304-3975(01)00185-2.

Edit Distance of Finite State Transducers

C. Aiswarya
Chennai Mathematical Institute, India
CNRS, ReLaX, IRL 2000, Chennai, India

Amaldev Manuel
Indian Institute of Technology Goa, India

Saina Sunny
Indian Institute of Technology Goa, India

Abstract

We lift metrics over words to metrics over word-to-word transductions, by defining the distance between two transductions as the supremum of the distances of their respective outputs over all inputs. This allows to compare transducers beyond equivalence.

Two transducers are *close* (*resp.* k-close) with respect to a metric if their distance is finite (*resp.* at most k). Over integer-valued metrics computing the distance between transducers is equivalent to deciding the closeness and k-closeness problems. For common integer-valued edit distances such as, Hamming, transposition, conjugacy and Levenshtein family of distances, we show that the closeness and the k-closeness problems are decidable for functional transducers. Hence, the distance with respect to these metrics is also computable.

Finally, we relate the notion of distance between functions to the notions of diameter of a relation and index of a relation in another. We show that computing edit distance between functional transducers is equivalent to computing diameter of a rational relation and both are a specific instance of the index problem of rational relations.

2012 ACM Subject Classification Theory of computation → Transducers; Theory of computation → Quantitative automata

Keywords and phrases transducers, edit distance, conjugacy

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.125

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: http://arxiv.org/abs/2404.16518

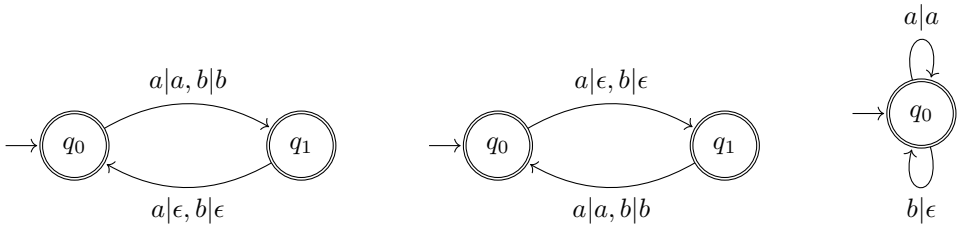
Funding *Amaldev Manuel*: Supported by the DST SERB MATRICS grant for the project *Deciding closeness of finite state transducers* [MTR/2022/000628].

1 Introduction

For meaningfully comparing two words (or sequences, vectors, functions, etc.), it is often necessary to have a measure that quantifies their (dis)similarity. It usually consists of associating a nonnegative integer to two words that indicates how different they are from each other. This usually defines a distance between words, the most popular of which are edit distances. It is the minimum number of edit operations required to transform one word into another. These operations typically include inserting or deleting a letter, substituting a letter with another, swapping adjacent letters (transpositions), and cyclic shifts. Edit distances are studied in coding [29, 41], parsing [2], speech recognition [33, 1], molecular biology [18, 24] etc. Interesting combinatorial problems on words such as the computation of longest common subsequences can be reduced to computing edit distances [6]. For a detailed overview of the history and applications of edit distances, see [27].

The notion of distance between two words can be lifted naturally to distance between a word and a set of words, or between two sets of words, and so on. There is a long line of research of this kind: computing the edit distance between two languages – usually defined as the smallest distance between any two pairs from the respective sets. It could be between a word and a regular language [42, 4], two regular languages [31], a regular language and itself [25], or a regular language and a context-free language [21]. In all these settings there are efficient algorithms for computing the edit distances.

In this paper we study the distance between two word-to-word functions (transductions) given by finite state transducers, i.e., automata with output. Finite state transducers are used in a variety of software and hardware systems such as encoders, decoders, demuxers, spell checkers, text normalizers, schema translators, template code generators, etc.



(a) $\mathcal{T}_1$. **(b)** $\mathcal{T}_2$. **(c)** $\mathcal{T}_3$.

Figure 1 $\mathcal{T}_1$ outputs letters at the odd positions, $\mathcal{T}_2$ outputs letters at the even positions and $\mathcal{T}_3$ outputs only a's.

Our aim is to develop a framework to meaningfully compare two transductions beyond equivalence. Consider the functions given by the transducers in Figure 1. The transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ output the letters at the odd and even positions respectively, while the transducer $\mathcal{T}_3$ erases b's in the input. If we were to find the odd one among these three functions, arguably $\mathcal{T}_3$ will be picked, with the length of the respective output on any input deviating significantly from that of the others. Our aim is to define a measure that quantifies such distances.

If we have a metric to compare the output words, we can extend it to transductions as follows. The distance between two transductions is the least upper bound of the distances between their respective outputs on any input word. We assume that their domains are the same, and we set the distance to infinity if this is not the case. We say that two transductions are close if their distance is finite, and they are k-close if their distance is at most k. We may simply say that two transducers are close (or k-close) instead, to mean that the transductions defined by these transducers are close (or k-close).

We are interested in the following question: *Given two finite state transducers, are the transductions defined by them close (or simply are the transducers close)?* Clearly, deciding closeness is a boundedness problem. We show closeness as well as k-closeness are decidable for various edit distance metrics, in particular Hamming (letter-to-letter substitutions), transposition (swapping adjacent letters), conjugacy (only cyclic shifts) and Levenshtein family of distances – Longest common subsequence (insertion and deletion), Levenshtein (insertion, deletion and substitution), and Damerau-Levenshtein (insertion, deletion, substitution and adjacent transposition). It turns out that computing distance between transducers is equivalent to deciding closeness and k-closeness over integer-valued metrics (see Proposition 3.6). Hence for the edit distances mentioned above, the distance between transducers is computable.

A related notion is that of *diameter* of a relation. We define it to be the supremum of the distance of every pair in the relation. We are interested in computing the diameter of rational relations over words, that is those given by (not necessarily functional) finite state transducers. A rational relation is said to have *bounded diameter* (*resp.* k-bounded diameter) if the diameter of the relation is finite (*resp.* at most k). It turns out that for every pair of transductions $\mathcal{T}_1$ and $\mathcal{T}_2$ there is a rational relation R such that for every metric, the diameter of R is same as the distance between $\mathcal{T}_1$ and $\mathcal{T}_2$. In fact, the converse is also true by virtue of Nivat's theorem (see Theorem 3.23).

Another related notion is that of the *index of a rational relation in the composition closure of another*. Let R, S be a rational relation over words. The index of R in the composition closure of S is defined to be the smallest integer k such that the relation R is contained in the k-fold composition of S. If such a k exists we say that R has the *finite index property* in the composition closure of S. We show that the finite index property is undecidable for arbitrary rational relations. However, if S is a metrizable relation (see Definition 3.18) w.r.t. the edit distances mentioned above, the index of R in the composition closure of S is computable.

Our decision procedure for k-closeness involves designing a weighted automaton that counts the number of edit operations for transforming one output to the other. We need to check whether there are input instances for which the weight is more than k. We extract a finite state automaton of size exponential in k that achieves this (see Proposition 3.11). This is a generic approach independent of the particular edit operations. However for Hamming and transposition distances, we have a direct polynomial time procedure for deciding k-closeness (see full version).

Recall that deciding closeness of transductions is same as deciding whether the diameter of a rational relation R is bounded. For the latter, consider a transducer recognising R. It turns out that if there are loops in this transducer that produce nonconjugate words (that are not cyclic shifts of each other) then such loops can be iterated to get unbounded diameter/distance. Thus a crucial ingredient in our decision procedure is checking for conjugacy of loops, which is decidable [3]. For boundedness w.r.t. Levenshtein distances, we show that this is also a sufficient condition (see Claim 4.9). For conjugacy distance, we show that the diameter of a rational relation R is bounded if and only if every pair in R is conjugate (see Proposition 4.6). Notice that this is not the case for arbitrary relations. In the case of Hamming distance, which only includes substitutions, we show that it is sufficient to check if the pairs of words generated by the loops after some shifted delay are identical (see Claim 4.11). This also holds true for transposition distance, but additionally, we also need to check if the words are permutations of each other (see Claim 4.12).

1.1 Related Work

The *adjacent functions* in [34] is an analogous definition for closeness between transductions with respect to prefix distance. Two functions $f, g : A^* \to B^*$ are *adjacent* if $\sup\{ d_p(f(w), g(w)) \mid w \in dom(f) \cap dom(g) \} < \infty$. Here, $d_p(u, v) = |u| + |v| - 2\max\{|z| \mid u, v \in zA^*\}$ denotes the *prefix distance* between two words u and v. The adjacency of two rational functions is used in deciding the sequentiality of a function. It is decidable to check if two given rational functions are adjacent or not (Proposition 1 of [34]).

Another problem that is similar in spirit is the *robustness problem*. We say a transducer $\mathcal{T}$ is *robust* w.r.t. a distance d if there is a nontrivial relation R between the distance between two input words (say $d(u, v)$) and distance between their corresponding outputs on $\mathcal{T}$ (say $d(\mathcal{T}(u), \mathcal{T}(v))$). For instance, R could be *Lipschitz continuity* – there is some $k > 0$ such

that $d(\mathcal{T}(u), \mathcal{T}(v)) \leq k \cdot d(u, v)$, or *locally Lipschitz continuity* – there exists $b, k > 0$ such that if $d(u, v) < b$ then $d(\mathcal{T}(u), \mathcal{T}(v)) \leq k \cdot d(u, v)$, etc. Sometimes, weaker notions of distance are considered (for instance by dropping the triangle inequality), and respective distances are called *cost* or *similarity* functions. The work [35] solves the locally Lipschitz continuity problem for sequential and unambiguous transducers using reversal bounded counter automata. The problem is shown to be undecidable for Lipschitz continuity even for deterministic transducers and the decidability is shown for the class that has a bound on the delay between input and output words [23].

Frougny and Sakarovitch studied rational relations with bounded delay [20], which is actually our diameter problem for rational relations when the distance over words is measured by their length difference. A problem related to the diameter of a rational relation is *almost reflexivity* of rational relations studied in [11]. A relation $R \subseteq A^* \times A^*$ is k-reflexive, for some integer $k \leq \infty$, if every element u of the domain is at a distance at most k from some element of the range v, with $(u, v) \in R$, and vice versa. The relation R is almost reflexive if $k < \infty$. It is shown undecidable to check if a deterministic rational relation is almost reflexive, or k-reflexive, for any given integer k, with respect to the following – Hamming, prefix, suffix, subword and Levenshtein edit distances. It is shown decidable for synchronized rational relation w.r.t. Hamming distance.

In 1966, Brzozowski raised the question of *finite power property* on regular languages – it takes a regular language L as input and asks whether there exists some positive integer n such that $(L + \epsilon)^n = L^*$. It was solved in 1979 by Hashiguchi [22] and Simon [37], independently. We study the *finite index property* of a rational relation in the iterative composition of another relation. Notice that the finite index property is different from the finite power property in two respects. One, it is over relations and not languages, and secondly and more importantly, the iteration is obtained by relation composition and not concatenation.

1.2 Organisation of the Paper

In § 2, we recall the definitions of finite state transducers, metrics on words and edit distances. In § 3, we define the notion of distance between transducers, the diameter of a rational relation, and the index of a rational relation in another. We also establish the relation between these notions and state our results in this section. In § 4, we give the connections with conjugacy and the proof arguments remaining from § 3. Finally, we conclude in § 5 with a short discussion on future directions. Proofs omitted are provided in the full version.

2 Preliminaries

Let A^* denote the set of all finite words over the alphabet A. We use $|w|$ to denote the length of the word w. Let $w[i\,..\,j]$ denote the factor of w from index i to j where $1 \leq i \leq j \leq |w|$. A transduction is a function from words to words.

2.1 Finite State Transducers

The simplest form of a transducer is a deterministic finite state machine whose each transition and each final state is labelled by a possibly empty *output word*. Formally, a *sequential transducer* $\mathcal{T} = \langle \mathcal{A}, \lambda, o \rangle$ with input alphabet A and output alphabet B is a *deterministic* finite state automaton $\mathcal{A}$ with two associated output functions $\lambda : \Delta \to B^*$ and $o : F \to B^*$ where Δ and F are the set of transitions and the set of accepting states of $\mathcal{A}$ respectively.

On an input word that is accepted by the automaton, we concatenate the output words produced by the transitions in the unique run of the machine and finally append the end-of-input word of the final state to obtain the output of the machine. That is to say, if $\rho = \delta_1 \cdots \delta_n$ is the successful run of $\mathcal{A}$ on a word $w \in A^*$, the *output* of $\mathcal{T}$ on w, denoted by $\mathcal{T}(w)$, is the word $\lambda(\rho) \cdot o(q)$ where $\lambda(\rho) = \lambda(\delta_1) \cdots \lambda(\delta_n)$ and q is accepting state reached by the run. Let $L(\mathcal{A})$ denote the set of words accepted by $\mathcal{A}$, called the *language* of $\mathcal{A}$ or the *domain* of $\mathcal{T}$ (denoted as $dom(\mathcal{T})$). We can see that $\mathcal{T}$ defines a function from $dom(\mathcal{T})$ to B^*. Functions defined by sequential transducers are called *sequential*. In the literature, they are known as *subsequential functions*, introduced by Schützenberger [36]. Transducers given in Figure 1 are sequential.

If we allow the finite state automaton $\mathcal{A}$ to be nondeterministic, then $\mathcal{T}$ no longer defines a function, but a binary relation on $A^* \times B^*$. Such relations are called *rational*. If the relation is a function, then the transducer is called *functional*, and the corresponding functions are called *rational functions*. We can restrict the nondeterminism and still compute all rational functions. A finite state automaton is *unambiguous* if on each input word the machine has at most one run. It is a well-known fact in the theory of transducers that all *rational functions* are computed by finite state transducers whose underlying automata are unambiguous [10]. Such transducers are called *unambiguous transducers*. Clearly sequential functions are a strict subset of rational functions. For instance, the function "output the input word if the last letter of the input is an a, otherwise the empty word" is rational but not sequential.

There exist generalisations of rational functions where the underlying automaton is a two-way finite state automaton or equivalently a finite state automaton with registers (corresponding functions are called *regular* [17, 5]), or two-way finite state automaton with pebbles (polyregular functions [8, 9]). An overview of the classical theory of transducers is given in [19]. In this paper, we restrict our attention to one-way functional transducers.

2.2 Metric on Words, Edit Distances

Simply put, a metric on a set is used to measure distance between any two elements of the set. A *metric on words* over the alphabet A is a function $d : A^* \times A^* \to [0, \infty]$ such that for any words u, v and w in A^*, $d(u, v) = 0 \iff u = v$ (*separation*), $d(u, v) = d(v, u)$ (*symmetry*), and $d(u, v) \leq d(u, w) + d(w, v)$ (*triangle inequality*).

A metric is *integer-valued* if it has range $\mathbb{N} \cup \{\infty\}$. A trivial metric on words is the *discrete metric* – distance between words u and v, denoted by $d_\infty(u, v)$, is 0 if $u = v$ and ∞ otherwise. Another straightforward distance on words is the absolute difference of their lengths (denoted as d_{len}). This is a *pseudo-metric* since the distance between two distinct words can be zero, i.e., does not satisfy the separation property of a metric.

An important class of metrics in the context of word transducers is *edit distances*. Loosely speaking, *edits* are operations that transform words, such as *inserting a letter*, *deleting a letter*, *substitutions (letter-to-letter)*, *adjacent transpositions* (swapping adjacent letters), *left and right shifts* etc. For a fixed set of edit operations C, the edit distance with respect to C between words u and v, is the minimum number of edits in C required to transform u to v if it is possible, and ∞ otherwise. The common edit distances and their corresponding operations are recalled in Table 1. Since many of these operations are obtained by combinations of the others, we can relate these metrics. The notation $d_1 \leq d_2$ is an abbreviation for $d_1(u, v) \leq d_2(u, v)$ for all words u, v. We can also relate the metrics up to boundedness (See [14] for a detailed introduction). Let $\alpha : \mathbb{N} \to \mathbb{N}$ be a *correction* function. Usual examples are increments (e.g. $x \mapsto x + 2$), scaling (e.g. $x \mapsto 2 \cdot x$) etc. We extend α to the domain $\mathbb{N} \cup \{\infty\}$ by letting $\alpha(\infty) = \infty$. We write $d_1 \lesssim d_2$ to mean that there is some α such that

Table 1 Edit Distances.

Edit Distance	Denotation	Allowed Operations
Hamming distance	d_h	letter-to-letter substitutions
Transposition distance	d_t	swapping adjacent letters
Conjugacy distance	d_c	left and right cyclic shifts
Levenshtein edit distance	d_l	insertions, deletions, and substitutions
Longest Common Subsequence	d_{lcs}	insertions and deletions
Damerau-Levenshtein distance	d_{dl}	insertions, deletions, substitutions and adjacent transpositions

$d_1 \leq \alpha \circ d_2$. Clearly, if $d_1 \leq d_2$ then $d_1 \lesssim d_2$. If $d_1 \lesssim d_2$ and $d_2 \lesssim d_1$, we write $d_1 \approx d_2$ (this is known as the cost equivalence or the boundedness equivalence). If two functions f and g are cost-equivalent then f and g are bounded over precisely the same family of subsets (See Proposition 1 of [14]).

▶ **Lemma 2.1.** *The metrics defined in Table 1 are related as follows:*

1. $d_{len} \leq d \leq d_\infty$, *for each edit distance metric* $d \in \{d_l, d_h, d_t, d_c, d_{lcs}, d_{dl}\}$
2. $d_l \approx d_{lcs} \approx d_{dl}$
3. $d_l \leq d_h \lesssim d_t$
4. $d_l \lesssim d_c$
5. d_c *and* d_t *as well as* d_c *and* d_h *are incomparable, i.e.,* $d_h \not\lesssim d_c, d_c \not\lesssim d_h$ *and* $d_t \not\lesssim d_c, d_t \not\lesssim d_c$

3 Distance between Transductions

In this section we define the notion of distance between two rational functions, diameter of a rational relation, and index of a rational relation in another. We establish the relation between these notions and state our results.

3.1 Comparing Transducers

We lift a metric over words to the class of word-to-word functions as follows.

▶ **Definition 3.1** (Metric on transductions). *Let d be a metric on words over the alphabet B. Given two partial functions $\mathcal{T}, \mathcal{S} : A^* \to B^*$, we define*

$$d(\mathcal{T}, \mathcal{S}) = \begin{cases} \sup\{\, d(\mathcal{T}(w), \mathcal{S}(w)) \mid w \in dom(\mathcal{T})\} & \text{if } dom(\mathcal{T}) = dom(\mathcal{S}) \\ \infty & \text{otherwise} \end{cases}$$

▶ **Proposition 3.2.** *d is a metric on transductions.*

▶ Remark 3.3. We can define a notion of distance between word-to-word relations in the above manner, however this distance will not be a metric. In particular $d(R, R)$ will not be 0 for a relation R that is not a (partial) function.

▶ **Example 3.4.** Consider the sequential transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ in Figure 1. The transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ output the letters at the odd and even positions respectively. For any input word u, $||\mathcal{T}_1(u)| - |\mathcal{T}_2(u)|| \leq 1$. Hence $d_{len}(\mathcal{T}_1, \mathcal{T}_2) = 1$. For input word $(ab)^n$ where $n > 1$, the outputs produced by $\mathcal{T}_1$ and $\mathcal{T}_2$ are a^n and b^n respectively. Since n substitutions are required to convert a^n to b^n, $d_l(a^n, b^n) = n$. Therefore, $d_h(\mathcal{T}_1, \mathcal{T}_2) = \infty$ as well as $d_l(\mathcal{T}_1, \mathcal{T}_2) = \infty$.

Table 2 Problems about distance between two transducers w.r.t. the metric d.

Problem	Input	Question
Distance Problem	transducers $\mathcal{T}, \mathcal{S}$	$d(\mathcal{T}, \mathcal{S})$?
Closeness Problem	transducers $\mathcal{T}, \mathcal{S}$	Is $d(\mathcal{T}, \mathcal{S}) < \infty$?
k-Closeness Problem	integer k, transducers $\mathcal{T}, \mathcal{S}$	Is $d(\mathcal{T}, \mathcal{S}) \leq k$?

▶ **Example 3.5.** The sequential transducer $\mathcal{T}_4$ in Figure 2 replaces each block of 0's by a single 0 and each block of 1's by a single 1. Similarly, $\mathcal{T}_5$ substitutes a block of 0's by a single 1 and a block of 1's by a single 0. The output words produced by the transducers on any input word is an alternate sequence of 0's and 1's. If $\mathcal{T}_4$ outputs 010, then $\mathcal{T}_5$ produces its complement, i.e., 101. The Hamming distance between $\mathcal{T}_4$ and $\mathcal{T}_5$ is ∞, but the Levenshtein distance is 2.

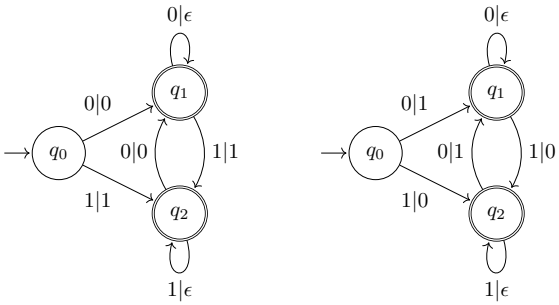


Figure 2 $\mathcal{T}_4$ (left) outputs 0 & 1 for each block of 0's & 1's *resp.* whereas $\mathcal{T}_5$ (right) outputs 1 & 0 for each block of 0's & 1's *resp.*

Let d be a distance on words. The value $d(\mathcal{T}, \mathcal{S})$ is an upper bound on how dissimilar the outputs of transducers $\mathcal{T}$ and $\mathcal{S}$ can be on any input. It is natural to ask the computational and boundedness problems given in Table 2.

Closeness and k-closeness are respectively a boundedness and an upper bound problem on distance.

▶ **Proposition 3.6.** *Let d be an integer-valued metric. The distance problem w.r.t. d is computable if and only if k-closeness and closeness problems w.r.t. d are decidable.*

Proof. Clearly, if we can compute the distance w.r.t. d then we can decide k-closeness as well as closeness. For the other direction, given two transducers, we first check if they are close and if it is we perform an exponential search – check if they are k-close for $k = 2^0, 2^1, 2^2, \ldots$ till it fails and subsequently perform a binary search on the interval $[2^n, 2^{n+1}]$, $n \in \mathbb{N}$ that contains the distance. ◀

We say two transducers $\mathcal{T}$ and $\mathcal{S}$ are *close* (*resp.* k*-close*, for $k \geq 0$) w.r.t. d if $d(\mathcal{T}, \mathcal{S}) < \infty$ (*resp.* $d(\mathcal{T}, \mathcal{S}) \leq k$). Closeness with respect to the discrete metric d_∞ is precisely the equivalence problem. Closeness w.r.t. the length metric d_{len} can be characterised in terms of delay as follows.

▶ **Proposition 3.7.** *Given two transducers $\mathcal{T}_1, \mathcal{T}_2$ with identical domain, $d_{len}(\mathcal{T}_1, \mathcal{T}_2)$ is finite iff there exists a $k \in \mathbb{N}$ such that on any input word w, the difference in lengths of the partial outputs of $\mathcal{T}_1, \mathcal{T}_2$ on any prefix of w is bounded by k.*

In the case of edit distances, closeness means that the output of $\mathcal{T}_1$ can be converted to the output of $\mathcal{T}_2$ by doing a bounded number of edits.

▶ Remark 3.8. From Definition 3.1, it is easy to verify that Lemma 2.1 holds for transducers as well. If $d_1 \lesssim d_2$, then it is easy to see that if transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ are *not* close w.r.t. d_1, then they are not close w.r.t. d_2 either.

The problems in Table 2 for unambiguous transducers with identical domains can be reduced to that for sequential transducers by considering the cartesian product of the unambiguous transducers. Given two unambiguous transducers $\mathcal{T}_1$ and $\mathcal{T}_2$, we obtain the sequential transducers $\mathcal{T}_1'$ and $\mathcal{T}_2'$ as follows. The input automata for $\mathcal{T}_1'$ and $\mathcal{T}_2'$ are the same, call it $\mathcal{A}$, which is the cartesian product of the input automata of $\mathcal{T}_1$ and $\mathcal{T}_2$. By treating the transitions of the cartesian product as the input alphabet, we get input determinism. The output functions of $\mathcal{T}_1'$ and $\mathcal{T}_2'$ are lifted from T_1 and T_2 respectively.

▶ **Proposition 3.9.** *Let d be a distance on words. For each pair of unambiguous transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ with identical domain, there exist a DFA $\mathcal{A}$ and output functions λ_1', o_1' and λ_2', o_2' such that $d(\mathcal{T}_1, \mathcal{T}_2) = d(\mathcal{T}_1', \mathcal{T}_2')$ where the sequential transducer $\mathcal{T}_i' = \langle \mathcal{A}, \lambda_i', o_i' \rangle$, $i \in \{1, 2\}$. Furthermore, the size of the automaton $\mathcal{A}$ is polynomial in the size of $\mathcal{T}_1$ and $\mathcal{T}_2$.*

Given two transductions $\mathcal{T}$ and $\mathcal{S}$, we define a distance function that maps each word w to the distance between their outputs on w.

▶ **Definition 3.10** (Distance function)**.** *The* distance function $f^d_{\mathcal{T},\mathcal{S}} : A^* \to \mathbb{N} \cup \{\infty\}$ *of $\mathcal{T}$ and $\mathcal{S}$ is $f^d_{\mathcal{T},\mathcal{S}}(w) = d(\mathcal{T}(w), \mathcal{S}(w))$ if $w \in dom(\mathcal{T}) \cap dom(\mathcal{S})$; otherwise $f^d_{\mathcal{T},\mathcal{S}}(w) = \infty$.*

Transducers $\mathcal{T}$ and $\mathcal{S}$ are close w.r.t. a metric d if their domains are the same and their distance function $f^d_{\mathcal{T},\mathcal{S}}$ is *limited* (i.e.,$< \infty$ on its domain). Similarly k-closeness w.r.t. d of $\mathcal{T}$ and $\mathcal{S}$ reduces to k-limitedness of $f^d_{\mathcal{T},\mathcal{S}}$. Limitedness problems are well-studied in the context of weighted automata [28, 12]. Therefore, when the distance function $f^d_{\mathcal{T},\mathcal{S}}$ is computable by a $(\min, +)$-automaton, the distance between $\mathcal{T}$ and $\mathcal{S}$ is computable due to Proposition 3.6.

However, there are distance functions that are not computable by weighted automata. Let $A = \{a, b\}$. Consider the sequential transducers $\mathcal{T}_1, \mathcal{T}_2 : A^* \to A^*$ with the domain a^*b^* defining the functions $a^pb^q \mapsto a^p$, $a^pb^q \mapsto a^q$ respectively ($\mathcal{T}_1$ outputs the a's and erases the b's, $\mathcal{T}_2$ erases a's and renames the b's as a's). It is easily checked that their distance function w.r.t. the Levenshtein family ($d \in \{d_l, d_{lcs}, d_{dl}\}$) is $f^d_{\mathcal{T}_1,\mathcal{T}_2} : a^pb^q \mapsto |p - q|$.

If $f : A^* \to \mathbb{N} \cup \{\infty\}$ is a function computed by weighted automata ($(\min, +)$ or $(\max, +)$ or B-automata [13]), then $L_{f \leq k} = \{w \in A^* \mid f(w) \leq k\}$ is regular for each $k \leq \mathbb{N}$. Hence the function $f^d_{\mathcal{T}_1,\mathcal{T}_2}$ is not realised by any of them (consider the language $L_{f^d_{\mathcal{T}_1,\mathcal{T}_2} \leq k}$). In fact, it can be shown that the function $f^d_{\mathcal{T}_1,\mathcal{T}_2}$ is not computed even upto boundedness [15].

To compute k-closeness w.r.t. any of the edit distances, it is not necessary to compute the distance function precisely. The k-approximation of the distance function $f^d_{\mathcal{T},\mathcal{S}}$ is the function $\lceil f^d_{\mathcal{T},\mathcal{S}} \rceil^{\leq k} : w \mapsto f^d_{\mathcal{T},\mathcal{S}}(w)$ if $f^d_{\mathcal{T},\mathcal{S}}(w) \leq k$ and ∞ otherwise.

▶ **Proposition 3.11.** *If $\mathcal{T}$ and $\mathcal{S}$ are close w.r.t. the length metric, then the approximation $\lceil f^d_{\mathcal{T},\mathcal{S}} \rceil^{\leq k}$ for a metric $d \in \{d_l, d_{lcs}, d_{dl}, d_h, d_t, d_c\}$ is computed by a distance automaton for each $k \in \mathbb{N}$.*

To check if $\mathcal{T}$ and $\mathcal{S}$ are k-close, we check if they have the same domain and they are close w.r.t. the length metric (otherwise they are neither close nor k-close). If so, we check if the domain of T is same as the domain of $\lceil f^d_{\mathcal{T},\mathcal{S}} \rceil^{\leq k}$. Thus we get:

▶ **Corollary 3.12.** *Let d be any metric from Table 1, and $\mathcal{T}$ and $\mathcal{S}$ be any functional transducers. It is decidable if $\mathcal{T}$ and $\mathcal{S}$ are k-close w.r.t. d.*

Table 3 Problems about diameter of a rational relation w.r.t. the metric d.

Problem	Input	Question
Diameter Problem	rational relation R	$dia_d(R)$?
Bounded Diameter Problem	rational relation R	Is $dia_d(R) < \infty$?
k-Bounded Diameter Problem	integer k, rational relation R	Is $dia_d(R) \leq k$?

3.2 Diameter of a Rational Relation

▶ **Definition 3.13** (Diameter of a Rational Relation w.r.t. a distance d). *The diameter of a rational relation R with respect to a distance d, denoted by $dia_d(R)$, is the supremum of the distance of the related words in R.*

$$dia_d(R) = \sup \{ d(u, v) \mid (u, v) \in R \}$$

Similar to the questions asked in Table 2, we can ask the questions given in Table 3 about diameter of a rational w.r.t. a metric d. We say a rational relation has *bounded (resp. k-bounded) diameter w.r.t. a distance d* if the diameter of the relation w.r.t. d is finite (*resp.* $\leq k$). A rational relation with *bounded delay* is precisely those relations with bounded diameter w.r.t. a length metric. Relations with 0-delay are called *length-preserving relations* [16] where any two related words are of equal length. It is decidable to check if a rational relation has bounded delay or 0-delay [20].

Relations bounded w.r.t. the discrete metric are simply those with only identical pairs. It is decidable to determine if a rational relation R is identity. First, check if R is length-preserving. If so, we can construct a letter-to-letter transducer for R based on Eilenberg and Schützenberger's theorem [16] stating that a length-preserving rational relation over $A^* \times B^*$ is a rational subset of $(A \times B)^*$, or equivalently, it can be realised by a letter-to-letter transducer whose transitions are labelled with elements of $A \times B$. Finally, validate this transducer for identity by examining the labels of each transition.

3.3 Index of a Rational Relation in a Composition Closure

Consider two rational relations R over $A^* \times B^*$ and S over $B^* \times C^*$. The composition $S \circ R$ over $A^* \times C^*$ is defined by $(S \circ R)(u) = S(R(u)) = \bigcup_{v \in R(u)} S(v)$.

▶ **Definition 3.14** (Composition closure of a Rational Relation). *Let S be a rational relation over $A^* \times A^*$. Let $S^{(n)}$ denote the composition of S with itself $n \geq 0$ times ($S^{(0)}$ is taken to be the identity relation), and let $S^{\leq (n)}$ denotes the composition of S with itself at most n times, i.e., $S^{\leq (n)} = S^{(0)} \cup S^{(1)} \cup \cdots S^{(n)}$.*

The composition closure *of S, denoted as $S^{(*)}$, is defined as $S^{(*)} = \bigcup_{i \geq 0} S^{(i)}$.*

Notice that we use parenthesis around the superscript to indicate that the base operation is composition, and not concatenation.

▶ **Definition 3.15** (Index of a Rational Relation in a Composition Closure). *Let S be a rational relation over $A^* \times A^*$. An* index *of a rational relation R in the composition closure of S, denoted as Index(R, S), is the smallest integer k such that R is contained in $S^{\leq (k)}$.*

▶ **Example 3.16.** Consider a relation S over $\{a, b\}^* \times \{a, b\}^*$ that deletes the first a if exists on any input. Fix an integer $k > 0$ and let R be the relation that deletes the first k a's from the input if exists. The index of R in $S^{(*)}$ is k since for any input word $u \in A^*$, $R(u) \in S^{\leq (k)}(u)$.

Table 4 Problems about the index of a rational relation in the composition closure of another.

Problem	Input	Question
Index Problem	rational relation R, S	Index(R, S)?
Bounded (or Finite) Index Problem	rational relation R, S	Is Index$(R, S) < \infty$?
k-Bounded Index Problem	integer k, rational relation R, S	Is Index$(R, S) \leq k$?

Consider another relation R' that deletes all a's from the input. Since $R'(a^{k+1}) \notin S^{\leq(k)}(a^{k+1})$ for any $k > 0$, the index of R' in $S^{(*)}$ is ∞.

As seen in the case of the distance and diameter problem, we can ask questions in Table 4 about the index of a rational relation in the composition closure of a relation. We say a rational relation R has *bounded (resp. k-bounded)* index in the composition closure of a rational relation S if the index of R in $S^{(*)}$ is finite (*resp.* $\leq k$).

Deciding the boundedness of the index problem for an arbitrary rational relation is difficult.

▶ **Lemma 3.17.** *It is undecidable to check if a rational relation has a bounded index in the composition closure of an arbitrary rational relation.*

However, we show that the index problem is decidable w.r.t. a large class of rational relations defined below.

▶ **Definition 3.18** (Metrizable Relation)**.** *Let S be a rational relation over $A^* \times A^*$. Let $d_S : A^* \times A^* \to \mathbb{N} \cup \{\infty\}$ be the distance between two vertices in the graph of S, i.e., for any two words u and v, $d_S(u, v)$ is the smallest i such that $v \in S^{(i)}(u)$, and ∞ otherwise.*

We say S is a d-metrizable relation *for a metric d if $d_S \approx d$.*

▶ **Proposition 3.19.** *Let R be a rational relation and S be a d-metrizable relation for an integer-valued metric d for which $d_{len} \lesssim d$. If boundedness of diameter w.r.t. d is decidable for a rational relation, then Index(R, S) is computable.*

Proof. Similar to distance problem, the index problem is computable iff bounded index and k-bounded index problems are decidable. For a rational relation R and d-metrizable relation S, we show that Index$(R, S) < \infty$ iff $dia_d(R) < \infty$ as follows.

$$
\begin{aligned}
dia_d(R) < \infty &\iff \exists k \in \mathbb{N} \text{ s.t. } \forall (u, v) \in R, d(u, v) \leq k \\
&\iff \exists k' \in \mathbb{N} \text{ s.t. } \forall (u, v) \in R, d_S(u, v) \leq k' && \text{(Since } d_S \approx d\text{)} \\
&\iff \forall (u, v) \in R, v \in S^{\leq(k')}(u) && \text{(Definition of } d_S\text{)} \\
&\iff \text{Index}(R, S) < \infty
\end{aligned}
$$

Therefore, if the boundedness of diameter w.r.t. d is decidable for a rational relation, then we can decide if Index$(R, S) < \infty$. If so, then it suffices to decide if Index$(R, S) \leq k$ for $k = 0, 1, \ldots$ and output the smallest k as the index of R in the composition closure of S.

Since $dia_d(R) < \infty$ and $d_{len} \lesssim d$, the rational relation R has a bounded delay. Similarly, S also has a bounded delay since for all $(u, v) \in S$, $d_S(u, v) = 1 \Rightarrow \exists k \in \mathbb{N}$ s.t. $d(u, v) \leq k$ (since $d_S \approx d$) $\Rightarrow \exists k' \in \mathbb{N}$ s.t. $d_{len}(u, v) \leq k'$ (since $d_{len} \lesssim d$). Since S has bounded delay, for any $k \in \mathbb{N}$, $S^{(k)}$ also has bounded delay. It is known that emptinesss and set difference of two rational relations with bounded delay is decidable (Corollary 2 of [20]). For any $k \in \mathbb{N}$, deciding Index$(R, S) \leq k$ reduces to checking if $R \subseteq S^{\leq(k)}$ (or equivalently, $R \setminus S^{\leq(k)} = \emptyset$), and hence decidable. ◀

A close and (almost) dual notion is that of a metric that defines a rational relation.

▶ **Definition 3.20** (Rationalizable Distance). *A distance d on words is* rationalizable *if the relation $S_d = \{(u, v) \mid d(u, v) = 1\}$, called the* distance relation *of d, is rational.*

▶ **Example 3.21.** Consider the hamming distance d_h. We can construct a rational relation $S_h = \{(u, v) \mid u \text{ and } v \text{ differ only in exactly one position}\}$. For example, let $A = \{a, b\}$ and $S_h(aba) = \{bba, aaa, abb\}$. For this, construct a transducer that nondeterministically chooses a position and replaces the input letter with other letters in the alphabet. Similarly, the distance relation of the length metric $S_{len} = \{(u, v) \mid ||u| - |v|| = 1\}$ is also rational.

In fact, we have the following result about the rationalizability of edit distances referred in Table 1.

▶ **Proposition 3.22.** *Every edit distance $d \in \{d_l, d_h, d_t, d_c, d_{lcs}, d_{dl}\}$ is rationalizable.*

3.4 Reductions between Distance, Diameter and Index Problems

We show that the distance problem of two rational functions is mutually reducible to the diameter problem of a rational relation, which in turn is mutually reducible to the index problem of a rational relation in the composition closure of a metrizable relation. Thus, their closeness and boundedness problems are also interreducible.

The correspondence between distance and diameter follows from Nivat's theorem:

▶ **Theorem 3.23** ([32]). *Let A and B be alphabets. The following conditions are equivalent.*

1. *R is a rational relation over $A^* \times B^*$.*
2. *There exist an alphabet C, two alphabetic morphisms $\phi : C^* \to A^*$ and $\psi : C^* \to B^*$ and a regular language $L \subset C^*$ such that $R = \{(\phi(w), \psi(w)) \mid w \in L\}$*

From Proposition 3.9 and (2) ⇒ (1) in the above theorem, it follows that distance of two rational functions reduces to the diameter of a rational relation. Now, given a rational relation R, we can create two functional transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ in the following way. The domain for these transducers corresponds to the set L in Theorem 3.23. For each transition in $\mathcal{T}_1$ and $\mathcal{T}_2$ that involves an input alphabet symbol σ, we set the outputs to be $\phi(\sigma)$ and $\psi(\sigma)$ in Theorem 3.23, respectively. Consequently, $\mathcal{T}_1$ and $\mathcal{T}_2$ consist of the sets $\{\phi(w) \mid w \in L\}$ and $\{\psi(w) \mid w \in L\}$ respectively. Since the domain of these transducers is identical, the distance between $\mathcal{T}_1$ and $\mathcal{T}_2$ with respect to any distance d, $d(\mathcal{T}_1, \mathcal{T}_2) = \sup\{\, d(\phi(w), \psi(w)) \mid w \in L \,\}$, that is equivalent to the diameter of R w.r.t. the distance d.

The correspondence between diameter and index for rationalizable distances is stated in the following proposition.

▶ **Proposition 3.24.** *The diameter of a rational relation R w.r.t. a rationalizable distance d is equal to the index of the rational relation R in the composition closure of the distance relation of d.*

Proof. Assume that the diameter of a relation R w.r.t. a distance d is ∞. We claim that the index of R in $S_d^{(*)}$ is also ∞ where S_d is the distance relation of d. Suppose not, i.e., let $k < \infty$ be the index of R in $S_d^{(*)}$. Thus, $\forall (u, v) \in R$, $v \in S_d^{\leq (k)}(u)$. Since S_d is the distance relation of d, $\forall (u, v) \in R$, $d(u, v) \leq k$. However, this contradicts the fact that $dia_d(R) = \infty$. Hence, the index of R in $S_d^{(*)}$ is infinite. Similarly, we can prove the other direction. Now, suppose the diameter of R w.r.t. d is finite, i.e.,

$$\begin{aligned}
\text{diameter of } R \text{ w.r.t. } d \text{ is } k < \infty &\iff \forall (u, v) \in R, d(u, v) \leq k \\
&\iff \forall (u, v) \in R, v \in S_d^{\leq (k)}(u) \\
&\iff \text{index of } R \text{ in } S_d^{(*)} \text{ is } k.
\end{aligned}$$

◀

3.5 Decidability Results

We study the problems stated in Tables 2, 3 and 4 and show that they are decidable for the metrics in Table 1. The index problems stated in Table 4 are undecidable in general (see Lemma 3.17), but is decidable for d-metrizable relations for metrics d given in Table 1.

Recall that, w.r.t. a metric d, distance problem is computable if and only if both closeness and k-closeness are decidable (see Proposition 3.6). We have shown that the k-closeness is decidable for all the metrics in Table 2 (Corollary 3.12). Hence to show the decidability of all the problems in Table 2, it suffices to show the decidability of the closeness problem. Furthermore, thanks to the inter-reductions described above (see § 3.4), the decidability of Table 3 follows as well as the decidability for the problems in Table 4 for the rationalizable distance. Moreover, the index of a rational relation in the composition closure of a d-metrizable relation for a metric d given in Table 2 is computable by Proposition 3.19.

It only remains to prove that closeness is decidable for edit distances in Table 1. This is stated below, and proved in the following section.

▶ **Theorem 3.25.** *Let d be any metric from Table 1, and $\mathcal{T}$ and $\mathcal{S}$ be any functional transducers. It is decidable if $\mathcal{T}$ and $\mathcal{S}$ are close w.r.t. d.*

4 Closeness for Edit Distances

In this section, we show that closeness is decidable for all the edit distances in Table 1. The first step is to check if the domain of the transducers are the same. This reduces to checking the equivalence of the underlying automata. For sequential transducers, the underlying automaton is a DFA, while for unambiguous transducers, the underlying automaton is an unambiguous NFA. Checking the equivalence of two unambiguous automata can be done in polynomial time [39], while it is PSPACE in the case of ambiguous automata [40]. Therefore, from now on, we assume that the domains of the transducers given as input to the closeness problem are identical.

Proposition 3.9 allows us to state the distance and closeness problems more abstractly in terms of an automaton over pairs of words. The proposition asserts that distance and closeness problems of two given sequential or unambiguous transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ can be reduced to the corresponding problem for a DFA $\mathcal{A}$ with two sets of output functions λ_1, o_1 and λ_2, o_2. We can combine the output functions to output a pair of words. That is to say, let $\lambda : \Delta \to B^* \times B^*$ be defined as $\lambda(\delta) = (\lambda_1(\delta), \lambda_2(\delta))$, where $\delta \in \Delta$ and Δ is the set of transitions of $\mathcal{A}$. Similarly let $o(p) = (o_1(p), o_2(p))$, where $p \in F$ and F is the set of accepting states of $\mathcal{A}$. Henceforth, we can assume that we are given a DFA $\mathcal{A}$ with the output functions λ and o, denoted as the sequential transducer $\mathcal{T}$. Since the input words are inconsequential for computing the distance, we can convert the transducer $\mathcal{T}$ to an automaton $\mathcal{A}$ that accepts a set of pairs of output words over $B^* \times B^*$, i.e., $L(\mathcal{A}) = \{(u, v) \in B^* \times B^* \mid (u, v) = \mathcal{T}(w), w \in dom(\mathcal{T})\}$. Clearly, transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ are close w.r.t. d if and only if there exist an integer $k \geq 0$ such that $\forall (u, v) \in L(\mathcal{A})$, $d(u, v) \leq k$.

Conjugacy of words plays an important role in closeness problems. A pair of words (u, v) is *conjugate* if there exist words x and y (possibly empty) such that $u = xy$ and $v = yx$ or equivalently, u and v are cyclic shifts of one another. For example, $(aaab, aaba)$ is a conjugate pair where $x = a$ and $y = aab$. Conjugacy relation is an equivalence relation on the set of words. A set of pairs is *conjugate* if each pair in the set is conjugate.

▶ **Lemma 4.1.** *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two sequential transducers that define a function from A^* to B^*. If $\mathcal{T}_1$ and $\mathcal{T}_2$ are close w.r.t. a metric $d \in \{d_l, d_h, d_t, d_c, d_{lcs}, d_{dl}\}$, then every loop in the trim automaton over $B^* \times B^*$, that accepts set of all pairs of output words of $\mathcal{T}_1$ and $\mathcal{T}_2$ on any input, generates only conjugate pair of words.*

Proof. This proof is adaptation of a related result in [3]. Let $\mathcal{A}$ be a trim automaton that realises the pair of output words of transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ on any input. Since $\mathcal{T}_1$ and $\mathcal{T}_2$ are close w.r.t. d, there exist an integer $k \geq 0$ such that $\forall (u, v) \in L(\mathcal{A})$, $d(u, v) \leq k$. Let (u, v) be a pair labelled in a loop rooted at some state q. Hence (u^ℓ, v^ℓ) for each $\ell \geq 0$ is also a pair in a loop rooted at q. We can safely assume that $|u| = |v|$, otherwise the edit distance will be unbounded as each iteration will increase the edit distance by a difference in length of u and v (Item 1 of Lemma 2.1).

Since $\mathcal{A}$ is trimmed, there exists a path from an initial state q_0 to q and from q to a final state q_f. Let (α_0, β_0) be a pair labelled in a path from q_0 to q, and let (α_1, β_1) be a pair labelled in a path from q to q_f. Thus, pair $(\alpha_0, \beta_0)(u^\ell, v^\ell)(\alpha_1, \beta_1)$ belongs to $L(\mathcal{A})$ where $\ell = 2^k$ (some value much larger than k). Since ℓ is much larger than k and $d(\alpha_0 u^\ell \alpha_1, \beta_0 v^\ell \beta_1) \leq k$, there exist large portions of u's and v's that match. Therefore, we can infer that u is a factor of vv, and v is a factor of uu.

Since v is an infix of uu, the following holds. There exist words x, y, p and q such that $v = xy$ and $u = px = yq$. Since $|u| = |v|$, length of p and length of y are the same, that implies $p = y$ (since $u = px = yq$). Therefore, $u = yx$. Hence u and v are conjugate words. Since the pair (u, v) was arbitrary, any pair generated by a loop in $\mathcal{A}$ is conjugate. ◀

4.1 Closeness w.r.t. Levenshtein distances and Conjugacy

In this subsection, we decide closeness w.r.t. Levenshtein family of distances – Levenshtein, Damerau-Levenshtein, and LCS distances – and conjugacy distance. Levenshtein family of distances are all equivalent with respect to closeness problems by Lemma 2.1 and Remark 3.8.

We have already seen that given two unambiguous transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ with identical domains, there exists an automaton $\mathcal{A}$ over $B^* \times B^*$ that accepts a set of all pairs of output words of $\mathcal{T}_1$ and $\mathcal{T}_2$ on any input. Thus, we can state the distance and closeness problems in terms of rational expressions over $B^* \times B^*$.

We define *pairs over the alphabet* B to be the set $B^* \times B^*$ with the pointwise concatenation $(u, v) \cdot (u', v') = (u \cdot u', v \cdot v')$. A *rational expression of pairs* over the alphabet B is a rational expression over the alphabet $\{(b, b') \mid b, b' \in (B \cup \{\epsilon\})\}$ that generates a subset of pairs over B. From the automaton $\mathcal{A}$ over $B^* \times B^*$, using state elimination method ([26], Lecture 9), we can construct the rational expression of pairs E for the output pairs generated by the transducer $\mathcal{T}_1$ and $\mathcal{T}_2$ on any input. We can lift the metric d to expressions by letting $d(E) = \sup\{d(u, v) \mid (u, v) \in L(E)\}$. Clearly $d(E) = d(\mathcal{T}_1, \mathcal{T}_2)$. Thus, the distance and closeness problems of sequential and unambiguous transducers reduce to the corresponding problems for a rational expression of pairs. Henceforth we assume that we are given a rational expression of pairs.

In the context of conjugacy distance, the closeness of a rational expression necessarily implies that every pair in the expression is conjugate. Otherwise, if there exists a pair $(u, v) \in L(E)$ such that u is not conjugate to v, then $d_c(u, v) = \infty$, thus $d_c(E) = \infty$. In fact, this is also a sufficient condition. The proof relies on the results from [3] that studies the conjugacy of rational expression over pairs of words. It crucially uses the notion of a *common witness* of a set of pairs.

▶ **Definition 4.2** (Common Witness of a Set of Pairs). *A* witness *of pair of conjugate words* (u, v) *is a word* z *such that either* $uz = zv$ *(called an* inner witness*) or* $zu = vz$ *(called an* outer witness*). A* common witness *of a set of pairs is a word* z *such that either* z *is an inner witness of every pair in the set, or* z *is an outer witness of every pair in the set.*

Lyndon and Schützenberger gave a characterisation of conjugacy of a pair of words, stated as a pair of words is conjugate if and only if it has both inner and outer witness (Proposition 1.3.4 of [30]). In [3], it is generalised to a set of pairs as follows.

▶ **Theorem 4.3** ([3]). *Let $M = (\alpha_0, \beta_0)G_1^*(\alpha_1, \beta_1)\cdots G_k^*(\alpha_k, \beta_k)$ be a set of pairs where $G_1, \ldots, G_k$, $k > 0$ are arbitrary sets of pairs of words, and $(\alpha_0, \beta_0), \ldots, (\alpha_k, \beta_k)$ are arbitrary pairs of words. The set M is conjugate iff M has a common witness.*

Existence of a common witness bounds the conjugacy distance of an expression as follows.

▷ Claim 4.4. If a rational expression over pairs E has a common witness z, then $d_c(E) \leq |z|$.

Proof. Since E has a common witness, either $\forall(u, v) \in L(E)$, $uz = zv$, or $\forall(u, v) \in L(E)$, $zu = vz$. WLOG, assume that $\forall(u, v) \in L(E)$, $uz = zv$. Now, for any pair $(u, v) \in L(E)$:

1. If $|u| > |z|$, then z is a prefix of u and suffix of v and hence $(u, v) = (zp, pz)$ for some word $p \in A^*$. Therefore $d_c(u, v) \leq |z|$ since v can be obtained by $|z|$ left cyclic shifts of u.
2. Otherwise, when $|u| \leq |z|$, the number of cyclic shifts required to transform u to v (note that u and v are conjugate since they have a witness) is less than $|u| \leq |z|$. ◁

A rational expression is *sumfree* if it does not use sum (i.e., $+$). In [3], it is shown that if a common witness exists, it is computable for a sumfree rational expression over pairs of words. It is folklore that every rational expression is equivalent to a sum of sumfree expressions [3]. The proposition below implies that to show closeness for a sum of sumfree expressions, it suffices to show closeness for each of its constituent sumfree expressions.

▶ **Proposition 4.5.** *Let $E = E_1 + \cdots + E_k, k \geq 1$ be a rational expression of pairs. Then $d(E) = \max(d(E_1), \ldots, d(E_k))$ for all word metrics d.*

An expression is conjugate if every pair generated by the expression is conjugate. The following proposition characterises closeness w.r.t. conjugacy distance.

▶ **Proposition 4.6.** *A rational expression over pairs of words is close w.r.t. conjugacy distance if and only if the expression is conjugate. Furthermore, the closeness w.r.t. conjugacy distance is decidable.*

Proof. One direction is trivial. Assume E to be an arbitrary rational expression of pairs and is conjugate. Let $E = E_1 + E_2 + \cdots + E_k$ where $E_1, E_2, \ldots, E_k$ are sumfree expressions. Since E is conjugate, each of its sumfree constituents E_i for $1 \leq i \leq k$ is also conjugate. Using Theorem 4.3, each E_i has a common witness, say z_i. From Claim 4.4, $d_c(E_i) \leq z_i$. Therefore, $d_c(E)$ is close w.r.t. conjugacy distance by Proposition 4.5. Hence, to decide closeness of E w.r.t. conjugacy distance, it suffices to check if E is conjugate. This reduces to checking if a common witness exists for each sumfree constituent. It is shown to be decidable in [3]. ◀

Now consider the case of Levenshtein distances. From Lemma 4.1, if an expression is close w.r.t. Levenshtein distances, it is necessary that every pair generated by a Kleene star in the expression needs to be conjugate. Using common witness, we show that it is also a sufficient condition.

▷ Claim 4.7. If a rational expression of pairs E has a common witness z, then $d_l(E) \leq 2|z|$.

Proof. The proof is similar to Claim 4.4. Since E has a common witness, either $\forall(u, v) \in L(E)$, $uz = zv$, or $\forall(u, v) \in L(E)$, $zu = vz$. WLOG, assume that $\forall(u, v) \in L(E)$, $uz = zv$. For any pair $(u, v) \in L(E)$, $|u| = |v|$ since $uz = zv$. There are two cases, either $|u| > |z|$ or $|u| \leq |z|$. If $|u| > |z|$, then z is a prefix of u and suffix of v and hence $(u, v) = (zp, pz)$ for some word p. Therefore, $d_l(E) \leq 2|z|$ by deleting z in the beginning and insert z at the end of u. Suppose $|u| \leq |z|$, the number of edits required to transform u to v is less than $|u| + |v| \leq 2|u| \leq 2|z|$. ◁

▶ **Proposition 4.8.** *Closeness of a rational expression w.r.t. Levenshtein distance is decidable.*

Proof. Given an arbitrary rational expression, there is an equivalent sum of sumfree expression. From Proposition 4.5, to show closeness for a sum of sumfree expressions, it suffices to show closeness for each of its constituent sumfree expressions. The general form of a sumfree expression $E = (\alpha_0, \beta_0)E_1^*(\alpha_1, \beta_1) \cdots E_k^*(\alpha_k, \beta_k)$ where $k \in \mathbb{N}$, for $0 \leq j \leq k$, (α_j, β_j) is a (possibly empty) pair of words, and for each $1 \leq i \leq k$, E_i is a sumfree expression.

▷ Claim 4.9. A sumfree expression $E = (\alpha_0, \beta_0)E_1^*(\alpha_1, \beta_1) \cdots E_k^*(\alpha_k, \beta_k)$ is close w.r.t. Levenshtein distance if and only if each E_i^* for $1 \leq i \leq k$ is conjugate.

Proof. From Lemma 4.1, if E is close w.r.t. Levenshtein edit distance then each E_i^* is conjugate. For the other direction, if each E_i^* is conjugate, then each E_i^* has a common witness, say z_i, by Theorem 4.3. From Claim 4.7, $d_l(E_i^*) \leq 2|z_i|$. Further, $d_l(E) \leq \sum_{j\in\{0\ldots k\}} d_l(\alpha_j, \beta_j) + \sum_{i\in\{1\ldots k\}} d_l(E_i^*) = \sum_{j\in\{0\ldots k\}} d_l(\alpha_j, \beta_j) + 2\sum_{i\in\{1\ldots k\}} z_i$, hence finite. This implies that if each E_i^* in E is conjugate, then $d_l(E)$ is finite. ◁

Therefore, checking the closeness of a rational expression w.r.t. Levenshtein distances reduces to checking the existence of a common witness of each Kleene star in its sumfree constituents, and thus decidable. ◀

For a sumfree rational expression, a witness, if exists, can be computed in polynomial time [3], and thus closeness w.r.t. Levenshtein and conjugacy distances are decidable in polynomial time. However, converting a rational expression to a sum of sumfree rational expressions can cause an exponential blow-up both in the number of summands and the size of each summand [3].

4.2 Closeness w.r.t. Hamming and Transposition distances

▶ **Theorem 4.10.** *Closeness w.r.t. Hamming and Transposition distance are decidable for functional transducers.*

Given two functional transducers, check if their domains are the same. If not, the distance is ∞ hence they are not close. Assume they have an identical domain. By Proposition 3.9, it suffices to consider two sequential transducers with a common underlying DFA. Let $\mathcal{T}_1 = \langle \mathcal{A}, \lambda_1, o_1 \rangle$ and $\mathcal{T}_2 = \langle \mathcal{A}, \lambda_2, o_2 \rangle$ be two sequential transducers. WLOG, we make the following assumptions.

1. (Property $\star$) Automaton $\mathcal{A}$ is trimmed, i.e., all states are accessible (reachable from the initial state) and coaccessible (from each state there is a path to some final state).
2. (Property ‡) $\mathcal{T}_1$ and $\mathcal{T}_2$ produce output words of identical length; otherwise the Hamming as well as transposition distance will be ∞. We can check this property: rename all the output letters in $\mathcal{T}_1$ and $\mathcal{T}_2$ to a and check their equivalence.
3. The delay between partial outputs of $\mathcal{T}_1$ and $\mathcal{T}_2$ is at most $k \in \mathbb{N}$ (By Proposition 3.7).

Let Q and $F \subseteq Q$ be the set of states and final states of $\mathcal{A}$ respectively, and let $q_0 \in Q$ be the initial state. For states $p, q \in Q$, Let $M_{p,q}$ be the set of pairs (u, v) such that there is a run ρ from p to q and $u = \lambda_1(\rho)$ and $v = \lambda_2(\rho)$. Extending this notation, for a state $q_f \in F$, let M'_{q,q_f} be the set of pairs (u, v) such that $u = u' \cdot o_1(q_f), v = v' \cdot o_2(q_f)$ and $(u', v') \in M_{q,q_f}$.

Let q be a state of the automaton. If (α, β) and (α', β') are two pairs in $M_{q_0,q}$, then $|\alpha| - |\beta| = |\alpha'| - |\beta'|$, or else one of the pairs in $\{(\alpha\alpha'', \beta\beta''), (\alpha'\alpha'', \beta'\beta'')\}$ will have different lengths, where (α'', β'') is some pair in M_{q,q_f}, for some $q_f \in F$, guaranteed by Property $(\star)$.

Therefore with each state q, we can associate the delay of a run reaching it, called the *delay at* q, denoted by ∂_q, as $|\alpha| - |\beta|$. Clearly $\partial_q \leq k$. By a symmetric argument, if (α, β) and (α', β') are two pairs in M_{q,q_f}, where q_f is some final state, then $|\alpha| - |\beta| = |\alpha'| - |\beta'| = -\partial_q$. This also implies that for all $(u, v) \in M_{q,q}$, $|u| = |v|$.

For each state q, either $M_{q,q} = \{(\epsilon, \epsilon)\}$, or $M_{q,q}$ is infinite. Let q be a state for which $M_{q,q}$ is nonempty. For a delay $\partial \in \mathbb{Z}$, a pair $(u, v) \in M_{q,q}$ where $n = |u| > \partial$, we define the interior of the pair (u, v) as

$$\mathit{interior}_\partial(u, v) = \begin{cases} (u[1 \dots n - \partial], v[\partial + 1 \dots n]) & \text{if } \partial \geq 0 \\ (u[\partial + 1 \dots n], v[1 \dots n - \partial]) & \text{if } \partial < 0 \end{cases}$$

For example, $\mathit{interior}_1(abc, def) = (ab, ef)$ and $\mathit{interior}_{-1}(abc, def) = (bc, de)$. We also define the *Left-Border* and *Right-Border* of the pair (u, v) as

$$\mathit{lborder}_\partial(u, v) = \begin{cases} v[1 \dots \partial] & \text{if } \partial \geq 0 \\ u[1 \dots \partial] & \text{if } \partial < 0 \end{cases} \qquad \mathit{rborder}_\partial(u, v) = \begin{cases} u[n - \partial + 1 \dots n] & \text{if } \partial \geq 0 \\ v[n - \partial + 1 \dots n] & \text{if } \partial < 0 \end{cases}$$

▷ Claim 4.11. Hamming distance between $\mathcal{T}_1$ and $\mathcal{T}_2$ is unbounded if and only if there exists a state $q \in Q$ and $(u, v) \in M_{q,q}$ such that $|u| = |v| > \partial_q$, and $u' \neq v'$ where $(u', v') = \mathit{interior}_{\partial_q}(u, v)$.

Proof. The Figure 3 depicts the situation described by (2).

($\leftarrow$): Assume there exists a state $q \in Q$ and $(u, v) \in M_{q,q}$ such that $|u| = |v| > \partial_q$, and $u' \neq v'$ where $(u', v') = \mathit{interior}_{\partial_q}(u, v)$. Let $(\alpha_0, \beta_0) \in M_{q_0,q}$ and $(\alpha_1, \beta_1) \in M_{q,q_f}$. Consider the pair $(u_i = \alpha_0 u^i \alpha_1, v_i = \beta_0 v^i \beta_1)$, $i \geq 1$ (shown in Figure 4). Since $u' \neq v'$, we can deduce that $d_h(u_i, v_i) \geq i$. Hence $d_h(\mathcal{T}_1, \mathcal{T}_2) = \infty$.

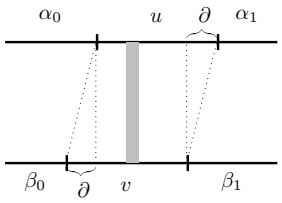


Figure 3 An edit in the interior of u and v.

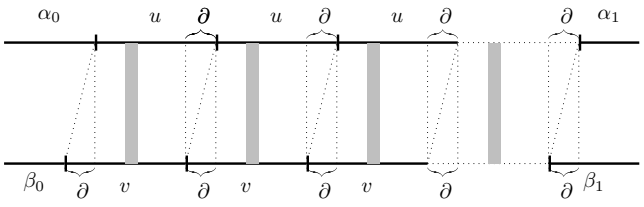


Figure 4 Words that require an arbitrarily large number of edits.

($\rightarrow$): Assume $d_h(\mathcal{T}_1, \mathcal{T}_2) = \infty$. Assume $\mathcal{A}$ has n states and the maximum length of an output produced on any transition or at the end-of-input is ℓ. Choose a run ρ of $\mathcal{A}$ such that the distance between the outputs produced on $\rho = \delta_1 \cdots \delta_m$, $m > 0$ is at least $((k+2)n+1)\ell$. We can associate each edit in $\lambda_1(\rho)$ with the transition δ_i such that the edit happens in

$\lambda_1(\delta_i)$. Since there are $((k+2)n+1)\ell$ edits, there are at least $(k+2)n+1$ transitions in ρ whose output words are edited. Associate each transition with its source state. By pigeonhole principle, there is a state q such that $\rho = \rho_1 \cdot \rho_2 \cdot \rho_3$ where

1. ρ_1 is a run from the initial state q_0 to q,
2. ρ_2 is a run from q to itself,
3. ρ_3 is a run from q to a final state q_f, and
4. there are at least $(k+1)$ edits in the factor $\lambda_1(\rho_2)$.

Let $u = \lambda_1(\rho_2)$ and $v = \lambda_2(\rho_2)$. Clearly $|u| = |v|$ and $|u| \geq (k+1)$. Since the edits in u are at least $k+1$, there is a position on which the pair $\mathit{interior}_{\partial_q}(u,v)$ differ. ◁

Next we show closeness w.r.t. transposition distance. We write $u \equiv v$ to denote that words u and v are permutations of each other. The *alphabetic vector* of a word over the alphabet A, denoted by $\vec{u}$, is the sequence $(|w|_{a_i})_{a_i \in A}$ for some fixed ordering of A. It is easy to observe that two words are permutations of each other if their alphabetic vectors are the same.

▷ Claim 4.12. Transposition distance between $\mathcal{T}_1$ and $\mathcal{T}_2$ is unbounded if and only if one of the following holds

1. There is a pair $(u,v) \in M'_{q_0,q_f}$, $q_f \in F$ such that $u \not\equiv v$.
2. There exists a state $q \in Q$ and $(u,v) \in M_{q,q}$ such that $|u| = |v| > \partial_q$, and $u' \neq v'$ where $(u',v') = \mathit{interior}_{\partial_q}(u,v)$.
3. There exists a state $q \in Q$ such that $M_{q,q}$ is infinite, and for each pair $(u,v) \in M_{q,q}$ of length at least $|\partial_q|$, $\mathit{interior}_{\partial}(u,v)$ is identical. Further, there are pairs $(u,v) \in M_{q,q}$ and $(\alpha,\beta) \in M_{q_0,q}$ (*resp.* M_{q,q_f}) such that: If $\partial_q \geq 0$, then $\alpha \not\equiv \beta \cdot \mathit{lborder}(u,v)$ (*resp.* $\mathit{rborder}(u,v) \cdot \alpha \not\equiv \beta$), and if $\partial_q < 0$, then $\alpha \cdot \mathit{lborder}(u,v) \not\equiv \beta$ (*resp.* $\alpha \not\equiv \mathit{rborder}(u,v) \cdot \beta$).

Proof. ($\leftarrow$): It is obvious that if Item 1 is true, then the transposition distance between $\mathcal{T}_1$ and $\mathcal{T}_2$ is unbounded. Therefore we assume that the output pairs of the transducers are permutations of each other. For Item 2, the proof is the same as in Claim 4.11. Next we consider Item 3. The cases are symmetric. Assume that there exist a pair $(u,v) \in M_{q,q}$, $(\alpha,\beta) \in M_{q_0,q}$, and WLOG $\partial_q \geq 0$ such that $\alpha \not\equiv \beta \cdot \mathit{lborder}(u,v)$. Let (α',β') be some pair in M_{q,q_f}. Consider the pair $(u_i = \alpha u^i \alpha', v_i = \beta v^i \beta')$, $i \geq 1$.

Let $(x,x) = \mathit{interior}_{\partial_q}(u,v)$, $z_1 = \mathit{lborder}(u,v)$, $z_2 = \mathit{rborder}(u,v)$. By assumption $\alpha \not\equiv \beta z_1$, and hence $z_2\alpha' \not\equiv \beta'$. Since interior of (u,v) is (x,x), we can deduce that $\alpha z_2 \alpha' \equiv \beta z_1 \beta'$. Therefore $\vec{\alpha} - \beta\vec{z_1} = z_2\vec{\alpha'} - \vec{\beta'}$. This means that the transpositions have to cancel out the differences in the vectors at each end of the word. We can prove by induction that it requires at least $|x|$ transpositions to mitigate a difference of 1, while keeping the alphabetic vector of the middle portion the same. Hence we deduce that $d_t(u_i,v_i) \geq i$.

($\rightarrow$): If $d_t(\mathcal{T}_1,\mathcal{T}_2) \in \infty$, either there is a pair of outputs (u,v) such that $d_t(u,v) = \infty$ (This is Item 1), or all the output pairs are permutations of each other and there is an infinite set of pairs $S = \{(u_i,v_i) \mid i > 0\}$ such that $d_t(u_i,v_i) \geq i$.

In the latter case, we show that either Item 2 or Item 3 holds. We say the set S is *error-bounded* if there is an $r > 0$ such that u_i and v_i differ in at most r positions. Clearly, there are sets with bounded errors on which d_t is infinite. We do case analysis.

If there is an infinite set of pairs $S = \{(u_i,v_i) \mid i > 0\}$ such that $d_t(u_i,v_i) \geq i$ that is *not* error-bounded, we proceed as in the proof of Claim 4.11 and obtain Item 2 by pigeonhole principle.

If the set of all output pairs is error-bounded, then clearly for states q such that $M_{q,q}$ is infinite, the interior of all the sufficiently large pairs in $M_{q,q}$ are identical. Moreover since the output pairs are permutations of each other there is a state q such that $|M_{q,q}| = \infty$ and there is a partial run from q_0 to q (or a partial run from q to q_f) whose output words are not permutations of each other. ◁

Claim 4.11 and Claim 4.12 can be verified for $\mathcal{T}_1$ and $\mathcal{T}_2$ in polynomial time. Thus, closeness of sequential and unambiguous transducers w.r.t. hamming and transposition distance is decidable in polynomial time.

5 Discussion and Conclusion

It is shown that distance between two rational functions w.r.t. common edit distances is computable. The related notions of diameter of a rational relation, and the index of a rational relation in the composition closure of another are also computable. We leave open the question of finding the precise computational complexity of the problems in Tables 2, 3 and 4.

The current decision procedure for closeness w.r.t. conjugacy and Levenshtein family of distances proceeds through the analysis of rational expressions. One could directly work on automata, but it is not enough to check for the conjugacy of simple cycles, as there can be complex strongly connected components. In such cases, a decidability proof for conjugacy can be achieved by utilizing Simon's factorization forests [38] and checking the conjugacy of the factorization trees inductively. Sumfree expressions are doing this in essence, circumventing the need to construct the transition monoids.

Lifting these notions to infinite words, and two-way transducers is an immediate next step. Distance between one-way transducers could be seen as the diameter of a rational relation obtained by the cartesian product. However, when the transducers $\mathcal{T}, \mathcal{S}$ are two-way or polyregular, the relation $\{(\mathcal{T}(w), \mathcal{S}(w)) \mid w \in dom(\mathcal{T})\}$ need not be rational. It remains to develop techniques for checking the conjugacy of non-rational relations.

An interesting question is: given two functional transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ with bounded distance, does there exist a transducer $\mathcal{T}$ such that $\mathcal{T}_2$ is equivalent to a cascading composition of $\mathcal{T}_1$ and $\mathcal{T}$? This is often called the *repair problem* and is well-studied between two regular languages [7].

References

1 M Ackroyd. Isolated word recognition using the weighted Levenshtein distance. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(2):243–244, 1980. doi:10.1109/TASSP.1980.1163382.

2 Alfred V Aho and Thomas G Peterson. A minimum distance error-correcting parser for context-free languages. *SIAM Journal on Computing*, 1(4):305–312, 1972. doi:10.1137/0201022.

3 C. Aiswarya, Amaldev Manuel, and Saina Sunny. Deciding conjugacy of a rational relation. *CoRR*, abs/2307.06777, 2023. doi:10.48550/arXiv.2307.06777.

4 Cyril Allauzen and Mehryar Mohri. Linear-space computation of the edit-distance between a string and a finite automaton. *CoRR*, abs/0904.4686, 2009. doi:10.48550/arXiv.0904.4686.

5 Rajeev Alur and Pavol Černý. Expressiveness of streaming string transducers. In *FSTTCS 2010*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.1.

6 Alberto Apostolico and Concettina Guerra. The longest common subsequence problem revisited. *Algorithmica*, 2:315–336, 1987. doi:10.1007/BF01840365.

7 Michael Benedikt, Gabriele Puppis, and Cristian Riveros. Regular repair of specifications. In *LICS 2011*, pages 335–344. IEEE Computer Society, 2011. doi:10.1109/LICS.2011.43.

8 Mikolaj Bojanczyk. Transducers of polynomial growth. In *LICS 2022*, pages 1–27. ACM, 2022. doi:10.1145/3531130.3533326.

9 Mikolaj Bojanczyk, Sandra Kiefer, and Nathan Lhote. String-to-string interpretations with polynomial-size output. In *ICALP 2019*, volume 132 of *LIPIcs*, pages 106:1–106:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.106.

10 Christian Choffrut and Serge Grigorieff. Uniformization of rational relations. In *Jewels are Forever: Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 59–71. Springer, 1999. doi:10.1007/978-3-642-60207-8_6.

11 Christian Choffrut and Giovanni Pighizzini. Distances between languages and reflexivity of relations. *Theoretical Computer Science*, 286(1):117–138, 2002. doi:10.1016/S0304-3975(01)00238-9.

12 Thomas Colcombet. On factorisation forests. *CoRR*, abs/cs/0701113, 2007. doi:10.48550/arXiv.cs/0701113.

13 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *ICALP 2009*, volume 5556 of *Lecture Notes in Computer Science*, pages 139–150. Springer, 2009. doi:10.1007/978-3-642-02930-1_12.

14 Thomas Colcombet. Regular cost functions, part I: logic and algebra over words. *Log. Methods Comput. Sci.*, 9(3), 2013. doi:10.2168/LMCS-9(3:3)2013.

15 Thomas Colcombet, Denis Kuperberg, Amaldev Manuel, and Szymon Torunczyk. Cost functions definable by min/max automata. In *STACS 2016*, volume 47 of *LIPIcs*, pages 29:1–29:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.29.

16 Samuel Eilenberg. *Automata, languages, and machines. A*. Pure and applied mathematics. Academic Press, 1974. URL: https://www.worldcat.org/oclc/310535248.

17 Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Log.*, 2(2):216–254, 2001. doi:10.1145/371316.371512.

18 David Eppstein, Zvi Galil, and Raffaele Giancarlo. Efficient algorithms with applications to molecular biology. In *Sequences: Combinatorics, Compression, Security, and Transmission*, pages 59–74. Springer, 1990. doi:10.1007/978-1-4612-3352-7_5.

19 Emmanuel Filiot and Pierre-Alain Reynier. Transducers, logic and algebra for functions of finite words. *ACM SIGLOG News*, 3(3):4–19, 2016. doi:10.1145/2984450.2984453.

20 Christiane Frougny and Jacques Sakarovitch. Rational relations with bounded delay. In *STACS 1991*, volume 480 of *Lecture Notes in Computer Science*, pages 50–63. Springer, 1991. doi:10.1007/BFb0020787.

21 Yo-Sub Han, Sang-Ki Ko, and Kai Salomaa. Computing the edit-distance between a regular language and a context-free language. In *DLT 2012*, volume 7410, pages 85–96. Springer, 2012. doi:10.1007/978-3-642-31653-1_9.

22 Kosaburo Hashiguchi. A decision procedure for the order of regular events. *Theoretical Computer Science*, 8(1):69–72, 1979. doi:10.1016/0304-3975(79)90057-4.

23 Thomas A. Henzinger, Jan Otop, and Roopsha Samanta. Lipschitz robustness of finite-state transducers. In *FSTTCS 2014*, volume 29 of *LIPIcs*, pages 431–443. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.431.

24 Richard M Karp. Mapping the genome: some combinatorial problems arising in molecular biology. In *STOC 1993*, number 8 in STOC '93, pages 278–285. ACM, 1993. doi:10.1145/167088.167170.

25 Stavros Konstantinidis. Computing the edit distance of a regular language. *Inf. Comput.*, 205(9):1307–1316, 2007. doi:10.1016/j.ic.2007.06.001.

26 Dexter Kozen. *Automata and computability*. Undergraduate texts in computer science. Springer, 1997. doi:10.1007/978-3-642-85706-5.

27 Joseph B Kruskal. An overview of sequence comparison: Time warps, string edits, and macromolecules. *SIAM review*, 25(2):201–237, 1983. doi:10.1137/1025045.

28 Hing Leung and Viktor Podolskiy. The limitedness problem on distance automata: Hashiguchi's method revisited. *Theoretical Computer Science*, 310(1-3):147–158, 2004. doi:10.1016/S0304-3975(03)00377-3.

29 Vladimir I Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics. Doklady*, volume 10, pages 707–710. Soviet Union, 1966.

30 Roger C Lyndon, Marcel-Paul Schützenberger, et al. The equation $a^M = b^N c^P$ in a free group. *Michigan Math. J*, 9(4):289–298, 1962. doi:10.1307/mmj/1028998766.

31 Mehryar Mohri. Edit-distance of weighted automata: General definitions and algorithms. *Int. J. Found. Comput. Sci.*, 14(06):957–982, 2003. doi:10.1142/S0129054103002114.

32 Maurice Nivat. *Transduction des langages de Chomsky*. PhD thesis, Annales de l'Institut Fourier, 1968. doi:10.5802/aif.287.

33 Teruo Okuda, Eiichi Tanaka, and Tamotsu Kasai. A method for the correction of garbled words based on the levenshtein metric. *IEEE Transactions on Computers*, 100(2):172–178, 1976. doi:10.1109/TC.1976.5009232.

34 Christophe Reutenauer and Marcel-Paul Schutzenberger. Minimization of rational word functions. *SIAM Journal on Computing*, 20(4):669–685, August 1991. doi:10.1137/0220042.

35 Roopsha Samanta, Jyotirmoy V. Deshmukh, and Swarat Chaudhuri. Robustness analysis of string transducers. In *ATVA 2013*, volume 8172, pages 427–441. Springer, 2013. doi:10.1007/978-3-319-02444-8_30.

36 Marcel Paul Schuetzenberger et al. Sur une variante des fonctions séquentielles. *Theoretical Computer Science*, 4(1):47–57, 1977. doi:10.1016/0304-3975(77)90055-X.

37 Imre Simon. Limited subsets of a free monoid. In *SFCS 1978*, pages 143–150. IEEE, 1978. doi:10.1109/SFCS.1978.21.

38 Imre Simon. Factorization forests of finite height. *Theoretical Computer Science*, 72(1):65–94, 1990. doi:10.1016/0304-3975(90)90047-L.

39 Richard Edwin Stearns and Harry B. Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM Journal on Computing*, 14(3):598–611, 1985. doi:10.1137/0214044.

40 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC 1973*, pages 1–9. ACM, 1973. doi:10.1145/800125.804029.

41 J Ullman. Near-optimal, single-synchronization-error-correcting code. *IEEE Transactions on Information Theory*, 12(4):418–424, 1966. doi:10.1109/TIT.1966.1053920.

42 Robert A. Wagner. Order-n correction for regular languages. *Commun. ACM*, 17(5):265–268, 1974. doi:10.1145/360980.360995.

Separability in Büchi VASS and Singly Non-Linear Systems of Inequalities

Pascal Baumann
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Eren Keskin
TU Braunschweig, Germany

Roland Meyer
TU Braunschweig, Germany

Georg Zetzsche
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Abstract

The ω-regular separability problem for Büchi VASS coverability languages has recently been shown to be decidable, but with an EXPSPACE lower and a non-primitive recursive upper bound – the exact complexity remained open. We close this gap and show that the problem is EXPSPACE-complete. A careful analysis of our complexity bounds additionally yields a PSPACE procedure in the case of fixed dimension ≥ 1, which matches a pre-established lower bound of PSPACE for one dimensional Büchi VASS. Our algorithm is a non-deterministic search for a witness whose size, as we show, can be suitably bounded. Part of the procedure is to decide the existence of runs in VASS that satisfy certain non-linear properties. Therefore, a key technical ingredient is to analyze a class of systems of inequalities where one variable may occur in non-linear (polynomial) expressions.

These so-called singly non-linear systems (SNLS) take the form $\boldsymbol{A}(x) \cdot \boldsymbol{y} \geq \boldsymbol{b}(x)$, where $\boldsymbol{A}(x)$ and $\boldsymbol{b}(x)$ are a matrix resp. a vector whose entries are polynomials in x, and $\boldsymbol{y}$ ranges over vectors in the rationals. Our main contribution on SNLS is an exponential upper bound on the size of rational solutions to singly non-linear systems. The proof consists of three steps. First, we give a tailor-made quantifier elimination to characterize all real solutions to x. Second, using the root separation theorem about the distance of real roots of polynomials, we show that if a rational solution exists, then there is one with at most polynomially many bits. Third, we insert the solution for x into the SNLS, making it linear and allowing us to invoke standard solution bounds from convex geometry.

Finally, we combine the results about SNLS with several techniques from the area of VASS to devise an EXPSPACE decision procedure for ω-regular separability of Büchi VASS.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Formal languages and automata theory

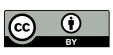
Keywords and phrases Vector addition systems, infinite words, separability, inequalities, quantifier elimination, rational, polynomials

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.126

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/2406.01008

Funding Funded by the European Union (ERC, FINABIS, 101077902). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.



1 Introduction

Vector addition systems with states (VASS) are one of the most popular and well-studied models of concurrent systems. A d-dimensional VASS consists of finitely many control states and d counters. Transitions between control states can increment or decrement the d counters, but importantly, one can only take a transition if the new counter values remain non-negative.

Separability problems. In recent years, a strong focus of the research on VASS was on *separability problems* [2, 5, 6, 8–11, 15–17, 31]. Here, we label the transitions of the input VASS $\mathcal{V}_1$, $\mathcal{V}_2$ by letters, which gives rise to languages L_1 and L_2. Then, we ask whether there exists a language S, from some class $\mathcal{S}$ of allowed separators, such that $L_1 \subseteq S$ and $L_2 \cap S = \emptyset$. Here, $\mathcal{S}$ is typically the class of regular languages.

An important motivation for studying separability problems is that separators can be viewed as certificates for disjointness, and thus the non-existence of a run in the product of $\mathcal{V}_1$ and $\mathcal{V}_2$. Such certificates are crucial for understanding safety verification for infinite-state systems, where the difficult part is to prove the non-existence of a run (the existence of a run is usually easy to show). In particular, certificates for non-existence are often the ingredient that is conceptually hardest to come by. For example, in the case of reachability in VASS, the KLM decomposition [18, 19, 22, 26] and Leroux's Presburger-definable inductive invariants [21] can be viewed as such certificates. Regular separators could play a similar role in alternative approaches to reachability.

In addition to understanding certificates, the recent attention on separability has led to other applications. For example, work on separability by bounded languages has led to a general framework to address unboundedness problems for VASS [8]. Moreover, separability results were used in an algorithm for deciding inclusion between unambiguous VASS [7].

With the recent contribution by Keskin and Meyer [16] (together with earlier decidability results for subclasses and variants [2, 5, 6, 8–11]), proving regular separability decidable for (finite-word) VASS, the *decidability* status of regular separability has largely been settled. However, concerning *complexity*, regular separability is far from understood, with few results: So far, the only exact complexity results are PSPACE-completeness for (succintly represented) one-dimensional VASS [9], EXPSPACE-completeness for VASS coverability languages [10], and Ackermann-completeness for VASS reachability languages [16].

Büchi VASS. A particularly challenging problem is (ω-)regular separability in Büchi VASS [2]. In a Büchi VASS $\mathcal{V}$, the language $L(\mathcal{V})$ consists of *infinite words* induced by runs that visit some final state infinitely often. As demonstrated by Baumann, Meyer, and Zetzsche [2], Büchi VASS behave quite differently in terms of regular separability from their finite-word counterpart, coverability languages of VASS [10]. Nevertheless, Baumann, Meyer, and Zetzsche proved decidability of regular separability for Büchi VASS [2]. However, the complexity remained open: Their algorithm requires at least Ackermannian time (because it constructs Karp-Miller graphs), and the only known lower bound is EXPSPACE.

Challenge: Non-linear constraints. Improving the complexity established in [2] is challenging due to the characterization of inseparability there: Inseparability is equivalent to the existence of a constellation of runs, called an *inseparability flower*, that must satisfy a *non-linear constraint*, meaning a constraint that is not expressible in linear arithmetic (i.e. first-order logic of $(\mathbb{Z}; +, <, 0, 1)$ or $(\mathbb{Q}; +, <, 0, 1)$). Essentially, such a flower is a triple

(α, β, γ) of cyclic runs such that (among other linear inequalities) the counter effect of the combined run $\alpha\beta\gamma$ is a scalar multiple of the counter effect of just α. In other words, we are looking for runs with effects $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{Z}^n$ such that

$$\exists x \in \mathbb{Q}\colon \boldsymbol{v} = x \cdot \boldsymbol{u} \ . \tag{1}$$

Detecting runs with such constraints is difficult: There are powerful generic EXPSPACE algorithms for detecting runs that satisfy unboundedness conditions [12], linear constraints [1], or variants of computation tree logic (CTL) [4]. However, condition (1) falls in neither of those categories.

In fact, we are not aware of any algorithmic approach to solving systems of linear inequalities with constraints of type (1) (let alone inside algorithms for VASS). There is a result by Gurari & Ibarra [13] showing that integral feasibility of systems of equalities $\boldsymbol{A} \cdot \boldsymbol{y} = \boldsymbol{b}(x)$ can be decided in NP, where $\boldsymbol{b}(x)$ is a vector containing in each component a quotient of polynomials in x. However, these do not seem to capture (1): By moving the denominators from $\boldsymbol{b}(x)$ to the left-hand side, one obtains equations where every variable from $\boldsymbol{y}$ is multiplied with the same polynomial over x. However, for (1), we need to multiply a *subset* of the linear variables (namely, those in $\boldsymbol{u}$) with a polynomial (namely, x). The same is true for the logic of *almost linear arithmetic* due to Weispfenning [32], whose existential fragment is also solvable in NP. Here, the definable sets are finite unions of solution sets of Gurari & Ibarra.

Furthermore, it is not even clear how to detect inseparability flowers by invoking reachability in VASS (even though this would only yield an Ackermann upper bound): To some extent, algorithms for reachability permit non-linear constraints – for example, using standard tricks, it is decidable whether one can reach a configuration with counter values (m, n) such that $n \leq 2^m$. However, the condition in (1) does not even seem to be captured by such methods.

Contribution. Our main result is that regular separability in Büchi VASS is EXPSPACE-complete, and PSPACE-complete in fixed dimension ≥ 1. The key technical ingredient is a method that we expect to be of independent interest: We develop a procedure for solving systems of linear inequalities with a single non-linear variable, which we call *singly non-linear systems of inequalities* (SNLS). We use our results about SNLS to show that if an inseparability witness exists, then there is one where all runs have at most doubly exponential length, yielding an EXPSPACE procedure. In fixed dimension, we obtain singly exponential bounds, leading to a PSPACE procedure.

Step I: Singly non-linear systems of inequalities. Intuitively, a singly non-linear system of inequalities (SNLS) is a system of inequalities that is linear in all but one variable. This means, there is one variable x that may appear in arbitrary polynomials, but all others can only occur linearly. More precisely, an SNLS is a system of inequalities of the form

$$\boldsymbol{A}(x) \cdot \boldsymbol{y} \geq \boldsymbol{b}(x), \quad \boldsymbol{y} \geq \boldsymbol{0} \ , \tag{2}$$

where $\boldsymbol{A}(x) \in \mathbb{Z}[x]^{m \times n}$ is an $m \times n$ matrix over the ring $\mathbb{Z}[x]$ of integer polynomials in x, $\boldsymbol{b}(x) \in \mathbb{Z}[x]^m$ is a vector of polynomials from $\mathbb{Z}[x]$, and $\boldsymbol{y}$ ranges over $\mathbb{Q}^n$. Notice that here indeed, x can be freely multiplied with itself and other variables, whereas the expression on the left-hand side must be linear in each component of $\boldsymbol{y}$.

Our main result about SNLS is that if a system as in (2) has a solution $(x, \boldsymbol{y}) \in \mathbb{Q} \times \mathbb{Q}^n$, then it has a solution where all numbers (numerators and denominators) are bounded exponentially in the description size of $\boldsymbol{A}(x)$ and $\boldsymbol{b}(x)$, even if numbers in the description are encoded in binary. This implies in particular that feasibility of SNLS is in NP.

In the proof, we first show that the set of all $x \in \mathbb{Q}$ for which there is a solution $(x, \boldsymbol{y})$ can be described by a Boolean combination Φ of polynomial constraints of the form $p(x) \geq 0$, for polynomials $p \in \mathbb{Z}[x]$. This amounts to a quantifier elimination procedure for a class of first-order formulas in the ordered field $(\mathbb{Q}; +, \cdot, <, 0, 1)$. This is perhaps surprising, since this structure does not admit quantifier elimination in general [24, Theorem 2].

Let us give a geometric explanation how we arrive at the constraints $\Phi(x)$: For each choice of x, the SNLS $\boldsymbol{A}(x) \cdot \boldsymbol{y} \geq \boldsymbol{b}(x),\ \boldsymbol{y} \geq \boldsymbol{0}$ defines a polyhedron. It is a standard fact in convex geometry that such a polyhedron has a point on a minimal face, and moreover this point can be expressed as the inverse of a submatrix of $\boldsymbol{A}(x)$ multiplied with $\boldsymbol{b}(x)$. This expression can then be plugged back into $\boldsymbol{A}(x) \cdot \boldsymbol{y} \geq \boldsymbol{b}(x)$ to obtain a set of polynomial constraints on x, subject to a particular determinant being non-zero. The latter non-zero condition can as well be expressed as a polynomial constraint.

We then show that Φ has a small solution: In one case, a rational root of one of the polynomials p is a solution – these can be bounded by the *Rational Root Theorem*. The other case is that the solution x lies strictly between two roots $r_1 < r_2$ of participating polynomials. But then one can observe that any rational number between those roots is a solution (if no other root lies between r_1 and r_2). Using the *Root Separation Theorem* (specifically, Rump's Bound [27]), which lower-bounds the size of such intervals (r_1, r_2), we can then conclude that such an interval must contain a rational number with small numerator and denominator.

Once we exhibit a small x, we can plug it into $\boldsymbol{A}(x) \cdot \boldsymbol{y} \geq \boldsymbol{b}(x)$ to obtain a system of linear inequalities. Then we use standard bounds to obtain a small (i.e. exponential) solution $\boldsymbol{y} \in \mathbb{Q}^n$. It should be noted that while our result about SNLS concerns rational solutions, we apply it in the case where $\boldsymbol{b}(x) \geq \boldsymbol{0}$, which means a rational solution can be turned into an integral solution by multiplying a common denominator.

Step II: Rackoff-like bounds. After establishing the solution bound for SNLS, we use this result in the context of Büchi VASS to show the existence of inseparability witnesses that are small, i.e. consist of runs that are at most doubly exponential in length. Here, we use an adaptation of the Rackoff technique [28] similar to the proofs of Habermehl [14] and Atig & Habermehl [1]. In [1], it is shown that runs satisfying (restricted) linear inequalities can be detected in EXPSPACE. For this, they use a Rackoff-style induction to bound the length of such runs. We devise a similar Rackoff-style induction to work with SNLS instead of ordinary linear inequalities. Different compared to the earlier works is the fact that our witnesses contain ω-counters, which may change when invoking the induction hypothesis. Moreover, we need to use a result of Demri on selective unboundedness [12, Theorem 4.6] (in EXPSPACE in the general case and PSPACE in fixed dimension) to check the coverability of our witnesses.

2 Preliminaries

Büchi VASS. A *Büchi vector addition system with states (Büchi VASS)* of dimension $d \in \mathbb{N}$ over an alphabet Σ is a tuple $\mathcal{V} = (Q, q_0, \Sigma, T, F)$. It consists of a finite set of states Q, an initial state $q_0 \in Q$, a set of final states $F \subseteq Q$, and a finite set of transitions $T \subseteq Q \times \Sigma^* \times \mathbb{Z}^d \times Q$. The size of the Büchi VASS is $|\mathcal{V}| := |Q| + |F| + \sum_{(q,w,\delta,q') \in T} \big(|w| + \|\delta\|_2\big)$. By $\|\delta\|_2$, we mean the size of the binary encoding of δ. Since we only consider Büchi VASS in this paper, we often simply call them VASS. If $d = 0$, we call $\mathcal{V}$ a *Büchi automaton*.

The semantics of the Büchi VASS is defined over its *configurations*, which are elements of $Q \times \mathbb{N}^d$. The *initial configuration* of $\mathcal{V}$ is $(q_0, \boldsymbol{0})$. We lift the transitions of the Büchi VASS to a relation over configurations $\to \subseteq Q \times \mathbb{N}^d \times \Sigma^* \times Q \times \mathbb{N}^d$ as follows: $(q, \boldsymbol{m}) \xrightarrow{w} (q', \boldsymbol{m}')$ if there is $(q, w, \delta, q') \in T$ such that $\boldsymbol{m}' = \boldsymbol{m} + \delta$. A *run* of the Büchi VASS is a (possibly infinite) sequence of configurations of the form $\sigma = (p_0, \boldsymbol{m}_0) \xrightarrow{w_1} (p_1, \boldsymbol{m}_1) \xrightarrow{w_2} \cdots$.

A run σ is *accepting* if it starts from the initial configuration and visits final states infinitely often, meaning there are infinitely many configurations $(q, \boldsymbol{m})$ in σ with $q \in F$. The run is said to be *labeled* by the word $w = w_0 w_1 \cdots$ in Σ^ω. The *language* $L(\mathcal{V})$ of the Büchi VASS consists of all infinite words that label an accepting run.

An infinite-word language $L \subseteq \Sigma^\omega$ is called *regular* if it is accepted by a Büchi automaton. As we only consider infinite-word languages, we just call them languages.

Arithmetic. Our approach to regular separability in Büchi VASS rests on a result about solutions to singly non-linear systems of inequalities. This also requires some terminology.

We define the integers, rationals, polynomials, and matrices together with the operations we need to perform on them. Let $a \in \mathbb{Z}$ be an integer. Its size $\|a\|_2 = |bin(a)|$ is the length of its binary encoding. We also use $\|a\|_1$ to denote the size of the unary encoding. This is the absolute value plus an extra bit for the sign. A polynomial with integer coefficients $p \in \mathbb{Z}[x]$ is a sum $\sum_{i=0}^{k} a_i x^i$ with $a_0, \ldots, a_k \in \mathbb{Z}$ and $a_k \neq 0$ if $k > 0$. We define $\|p\|_1 = \sum_{i=0}^{k} \|a_i\|_1$ and similar for $\|p\|_2$. The *degree* of the polynomial is $\deg(p) = k$, its *maximal coefficient* is $\mathrm{maxc}(p) = \max_{i \in [0,k]} \|a_i\|_1$. Note that $\|p\|_1 \leq (\deg(p) + 1) \cdot \mathrm{maxc}(p)$. A real number $r \in \mathbb{R}$ with $p(r) = 0$ is called a *root* of the polynomial. Let S be a set with a size function $\| - \|$ defined on it. We consider matrices $\boldsymbol{A} \in S^{m \times n}$ over S, and define their size $\|\boldsymbol{A}\|$ by summing up the sizes of the entries. We use $row(\boldsymbol{A}) = m$ and $col(\boldsymbol{A}) = n$. When $S = \mathbb{Z}[x]$, we also use $\deg(\boldsymbol{A})$ for the highest degree of a polynomial in $\boldsymbol{A}$ and $\mathrm{maxc}(\boldsymbol{A})$ for the maximal coefficient of a polynomial in $\boldsymbol{A}$. Pairs $(s_1, s_2) \in S \times S$ form a special case with size $\|(s_1, s_2)\| = \|s_1\| + \|s_2\|$. In particular, a rational number $t \in \mathbb{Q}$ is a pair $t = \frac{a}{b}$ of integers $a, b \in \mathbb{Z}$ with $\|t\|_1 = \|a\|_1 + \|b\|_1$, and similar for $\|t\|_2$.

We perform addition $a + b$ and multiplication $a \cdot b$ among integers, rationals, polynomials, and matrices. These operations can be executed in time polynomial in $\|a\|_2 + \|b\|_2$. The same holds for the comparison $a \geq b$ among integers and rationals. We also add, multiply, and compare integers and rationals with $-\infty$ and ∞. The definitions are as expected.

3 Main results

A language $R \subseteq \Sigma^\omega$ is said to *separate* languages $L_1, L_2 \subseteq \Sigma^\omega$, if $L_1 \subseteq R$ and $R \cap L_2 = \emptyset$. We call L_1 and L_2 *regular separable*, denoted by $L_1 \mid L_2$, if there is a separator R that is a regular language. The problem we address is the *regular separability problem* for Büchi VASS:

Given Two VASS $\mathcal{V}_1$ and $\mathcal{V}_2$ over some alphabet Σ.

Question Does $L(\mathcal{V}_1) \mid L(\mathcal{V}_2)$ hold?

We also consider the variants of this problem where the inputs are of fixed dimension: For a fixed number $d \in \mathbb{N} \setminus \{0\}$, the *$d$-dimensional regular separability problem* for Büchi VASS is the same problem as above, except that the input VASS $\mathcal{V}_1$ and $\mathcal{V}_2$ are restricted to be of dimension at most d. Our first main result is the following:

▶ **Theorem 3.1.** *The regular separability problem for Büchi VASS is* EXPSPACE*-complete. Moreover, the d-dimensional regular separability problem is* PSPACE*-complete for all $d \geq 1$.*

As mentioned above, the proof is based on a small model property for what we call singly non-linear systems of inequalities. We expect this result to be of independent interest. Formally, a *singly non-linear system (SNLS)* is a system of inequalities of the form

$$\boldsymbol{A}(x) \cdot \boldsymbol{y} \geq \boldsymbol{b}(x) \wedge \boldsymbol{y} \geq \boldsymbol{0} \ .$$

Here, $\boldsymbol{A}(x) \in \mathbb{Z}[x]^{m \times n}$ is an $m \times n$ matrix over the set of polynomials with integer coefficients in variable x, and $\boldsymbol{b} \in \mathbb{Z}[x]^m$ is a vector of polynomials. We also write an SNLS as $\mathcal{S} = (\boldsymbol{A}(x), \boldsymbol{b}(x))$, or $\mathcal{S}(x, \boldsymbol{y})$ to emphasize the variables. A *solution* to $\mathcal{S}$ is a pair $(t, \boldsymbol{s}) \in \mathbb{Q} \times \mathbb{Q}^n$ that satisfies $\boldsymbol{A}(t) \cdot \boldsymbol{s} \geq \boldsymbol{b}(t) \wedge \boldsymbol{s} \geq \boldsymbol{0}$. If a solution exists, we call the system *feasible*.

Our second main result is a bound on the size of least solutions.

▶ **Theorem 3.2.** *If the SNLS $\mathcal{S}$ is feasible, then it has a solution $(t, \boldsymbol{s})$, where all components of $\boldsymbol{s}$ have the same denominator, with $\|t\|_1, \|\boldsymbol{s}\|_1 \in (col(\mathcal{S}) \cdot \deg(\mathcal{S}) \cdot \mathrm{maxc}(\mathcal{S}))^{\mathcal{O}(\deg(\mathcal{S})^2 \cdot row(\mathcal{S})^4)}$.*

Theorem 3.2 implies that a feasible system $\mathcal{S}$ always has a solution of size at most singly exponential in $\|\mathcal{S}\|_1$. This gives an upper bound on the complexity of feasibility.

▶ **Corollary 3.3.** *Feasibility of SNLS is in* NP.

The reader may have noted that SNLS are more general than the non-linear systems we are confronted with when checking separability. There are at least two arguments in support of the generalization. First, non-linearity is not well-understood, and we believe a class of systems that admits an efficient algorithm for checking feasibility will find its applications. Second, the generalization only adds little complexity to the proof or, phrased differently, the special case already needs most considerations.

Organization. The remainder of the paper is organized as follows. In Section 4, we prove Theorem 3.2 and in Section 5, we show Theorem 3.1.

4 Singly Non-Linear Systems

In this section, we prove Theorem 3.2.

Some notation. By $\boldsymbol{A}(t)$ or $eval(\boldsymbol{A}(x), t)$ we mean the matrix with rational entries that results from $\boldsymbol{A}(x)$ by evaluating all polynomials at t. Let $\boldsymbol{A} \in R^{n \times n}$ be a square matrix over some ring R. In our exposition, we will consider matrices over the rings $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{Z}[x]$. We write $\det(\boldsymbol{A})$ for the determinant, and recall that if R is a field (such as $\mathbb{Q}$), then $\boldsymbol{A}$ is invertible if and only if $\det(\boldsymbol{A}) \neq 0$. The *adjugate* (also called *classical adjoint*) of $\boldsymbol{A}$ is the matrix $\mathrm{adj}(\boldsymbol{A}) \in R^{n \times n}$ with $\mathrm{adj}(\boldsymbol{A})[j, i] = (-1)^{i+j} \det(\boldsymbol{A}_{ij})$, where $\boldsymbol{A}_{ij}$ is the matrix obtained from $\boldsymbol{A}$ by removing the i-th row and the j-th column. It is well-known that then $\boldsymbol{A} \cdot \mathrm{adj}(\boldsymbol{A}) = \det(\boldsymbol{A}) \cdot \boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix in dimension n. In particular, if $\boldsymbol{A}$ is invertible, its inverse can be computed as $\boldsymbol{A}^{-1} = \frac{\mathrm{adj}(\boldsymbol{A})}{\det(\boldsymbol{A})}$ [20, Chapter XIII, Prop. 4.16].

In upper bound arguments, we will use the well-known Leibniz formula for determinants, which says $\det(\boldsymbol{A}) = \sum_{\sigma \in S_n} \mathrm{sgn}(\sigma) \cdot \prod_{i=1}^{n} \boldsymbol{A}[i, \sigma(i)]$ [20, Chapter XIII, Prop. 4.6]. Here, S_n is the set of all permutations of $[1, n]$ and $\mathrm{sgn}(\sigma) \in \{-1, 1\}$ is the sign of $\sigma \in S_n$.

Bounding solutions. For the proof of Theorem 3.2, we proceed in two steps. We first show that if an SNLS $\mathcal{S}(x, \boldsymbol{y})$ is feasible, then we find a small rational t for x such that the system $\mathcal{S}(t, \boldsymbol{y})$ is feasible. This system is the result of evaluating all polynomials in $\mathcal{S}$ at t, and thus having only $\boldsymbol{y}$ as the variables.

▶ **Lemma 4.1.** *If the SNLS $\mathcal{S}(x, \boldsymbol{y})$ is feasible, then there is a number $t \in \mathbb{Q}$ with $\|t\|_1 \in (col(\mathcal{S}) \cdot \deg(\mathcal{S}) \cdot \mathrm{maxc}(\mathcal{S}))^{\mathcal{O}(\deg(\mathcal{S}) \cdot row(\mathcal{S})^3)}$ such that $\mathcal{S}(t, \boldsymbol{y})$ is feasible.*

Lemma 4.1 is non-trivial and will occupy almost this entire section. To explain our approach, note that the feasibility of $\mathcal{S}(x, \boldsymbol{y})$ is equivalent to the feasibility of $\exists \boldsymbol{y}.\mathcal{S}(x, \boldsymbol{y})$. Our first step is to eliminate the quantifier and determine a new formula $\Phi(x)$ in which $\boldsymbol{y}$ no longer occurs and that is equivalent to the previous one over the rationals, $\exists \boldsymbol{y}.\mathcal{S}(x, \boldsymbol{y}) \models|_{\mathbb{Q}} \Phi(x)$. The equivalence says that for every $t \in \mathbb{Q}$, we have $t \models \exists \boldsymbol{y}.\mathcal{S}(x, \boldsymbol{y})$ if and only if $t \models \Phi(x)$.

The second step for Lemma 4.1 is to show that if the new formula holds, then we find a small solution for t. To this end, we will apply the Root Separation Theorem, which provides a lower bound on the distance between distinct real roots of polynomials. After establishing Lemma 4.1, we obtain Theorem 3.2 (at the end of this section) by taking the t provided by Lemma 4.1, and pair it with the $\boldsymbol{s} \in \mathbb{Q}^n$, which must exist according to the quantifier elimination done in the first step of Lemma 4.1.

4.1 Quantifier Elimination

We show how to remove the quantifier from $\exists \boldsymbol{y}.\mathcal{S}(x, \boldsymbol{y})$ with a tailor-made quantifier elimination algorithm. The fact that quantifier elimination is possible in this setting came as a surprise to us, given the non-linear nature and the setting of rationals. For example, the real closed field $(\mathbb{R}; +, \cdot, <, 0, 1)$ admits quantifier elimination by a classical result of Tarski [25, Theorem 3.3.15], but this is not true for the ordered field $(\mathbb{Q}; +, \cdot, <, 0, 1)$ of rationals [24, Theorem 2] (see [25, p. 71–72] for a simple example). This means, there are first-order formulas over $(\mathbb{Q}; +, \cdot, <, 0, 1)$ that have no quantifier-free equivalent. However, we show that if we existentially quantify the linear variables in the formulas induced by SNLS, then those quantifiers can be eliminated.

The precise formulation of the result needs some notation. A *lower bound constraint* has the form $p(x) \geq 0$ or $p(x) > 0$ with $p \in \mathbb{Z}[x]$ a polynomial with integer coefficients. The formula $\Phi(x)$ that we want to obtain takes the form $\bigvee_{i \in I} \bigwedge_{j \in J_i} \Phi_{i,j}(x)$, where the formulae $\Phi_{i,j}(x)$ are lower bound constraints. We call it a DNFLB, short for *disjunctive normal form with lower bound constraints as the literals*. We may also omit x and write Φ. We use $\deg(\Phi)$ and $\operatorname{maxc}(\Phi)$ for the maximal degree resp. coefficient of a polynomial in Φ.

▶ **Theorem 4.2.** *For every SNLS $\mathcal{S}(x, \boldsymbol{y})$, there is a DNFLB $\Phi(x)$ with $\exists \boldsymbol{y}.\mathcal{S}(x, \boldsymbol{y}) \models|_{\mathbb{Q}} \Phi(x)$, $\deg(\Phi) \in \mathcal{O}(row(\mathcal{S}) \cdot \deg(\mathcal{S}))$, and $\operatorname{maxc}(\Phi) \in (col(\mathcal{S}) \cdot \deg(\mathcal{S}) \cdot \operatorname{maxc}(\mathcal{S}))^{\mathcal{O}(row(\mathcal{S})^2)}$.*

Since our intention is to bound the solutions t to variable x, the given estimations on the degree and the maximal coefficient suffice for us. The proof actually gives an algorithm to compute Φ which runs in time exponential in the dimension of $\mathcal{S}$, but we do not need the effectiveness here. In the proof of Theorem 4.2, we will use a standard fact about polyhedra:

▶ **Lemma 4.3.** *Suppose $\boldsymbol{D} \in \mathbb{Q}^{m \times n}$ and $\boldsymbol{c} \in \mathbb{Q}^m$. If the system $\boldsymbol{D} \cdot \boldsymbol{x} \geq \boldsymbol{c}$ has a solution in $\mathbb{Q}^n$, then there is a solution $\boldsymbol{s} \in \mathbb{Q}^n$ that also satisfies $\boldsymbol{D}' \cdot \boldsymbol{s} = \boldsymbol{c}'$, where $(\boldsymbol{D}', \boldsymbol{c}')$ is a subset of the rows of $(\boldsymbol{D}, \boldsymbol{c})$ such that $\operatorname{rank}(\boldsymbol{D}') = \operatorname{rank}(\boldsymbol{D})$.*

Proof. By well-known decomposition theorems about polyhedra, a polyhedron $P = \{\boldsymbol{s} \in \mathbb{Q}^n \mid \boldsymbol{D} \cdot \boldsymbol{s} \geq \boldsymbol{c}\}$ is non-empty if and only if it has a non-empty minimal face [30, Theorem 8.5]. Moreover, minimal faces can be characterized as exactly the sets of the form $\{\boldsymbol{s} \in \mathbb{Q}^n \mid \boldsymbol{D}' \cdot \boldsymbol{s} = \boldsymbol{c}'\}$, where $(\boldsymbol{D}', \boldsymbol{c}')$ is a subset of the rows of $(\boldsymbol{D}, \boldsymbol{c})$ such that $\boldsymbol{D}'$ has the same rank as $\boldsymbol{D}$ [30, Theorem 8.4]. ◀

We are ready to prove Theorem 4.2:

Proof of Theorem 4.2. Let $\mathcal{S}(x, \boldsymbol{y}) = \boldsymbol{A}(x) \cdot \boldsymbol{y} \geq \boldsymbol{b}(x) \wedge \boldsymbol{y} \geq \boldsymbol{0}$. To fix the dimension, let $\boldsymbol{A} \in \mathbb{Z}[x]^{m \times n}$. We can equivalently write the SNLS as $\mathcal{S}'(x, \boldsymbol{y}) = \boldsymbol{D}(x) \cdot \boldsymbol{y} \geq \boldsymbol{c}(x)$ with

$$\boldsymbol{D}(x) \;=\; \begin{pmatrix} \boldsymbol{A}(x) \\ \boldsymbol{I}_n \end{pmatrix} \in \mathbb{Z}[x]^{(m+n)\times n} \qquad \boldsymbol{c}(x) \;=\; \begin{pmatrix} \boldsymbol{b}(x) \\ \boldsymbol{0}_n \end{pmatrix} \in \mathbb{Z}[x]^{m+n} \;,$$

i.e. we glue the $n \times n$ identity matrix $\boldsymbol{I}_n$ to the bottom of $\boldsymbol{A}(x)$ and extend $\boldsymbol{b}(x)$ by n zeros.

Assume $\mathcal{S}'$ is feasible and the solution for x is $t \in \mathbb{Q}$. By Lemma 4.3 and since $\boldsymbol{D}(t)$ has rank n, we can select a subset of n rows of $\boldsymbol{D}(t)$ and of $\boldsymbol{c}(t)$ such that the smaller system has a solution, even with equality. More formally, for any subset $R \subseteq [1, m+n]$, denote by $\boldsymbol{D}_R(t)$ (resp. $\boldsymbol{c}_R(t)$) the matrix (resp. vector) obtained by selecting only the rows in R from $\boldsymbol{D}(t)$ (resp. $\boldsymbol{c}(t)$). Then Lemma 4.3 tells us that there is an $\boldsymbol{s} \in \mathbb{Q}^n$ with $\boldsymbol{D}_R(t) \cdot \boldsymbol{s} = \boldsymbol{c}_R(t)$, where $\boldsymbol{D}_R(t)$ has rank n. In particular, $\boldsymbol{D}_R(t)$ is invertible and thus $\det(\boldsymbol{D}_R(t)) \neq 0$. The fact that $\boldsymbol{D}_R(t)$ is invertible allows us to write $\boldsymbol{s} = \boldsymbol{D}_R(t)^{-1} \cdot \boldsymbol{c}_R(t)$, which will be key for our quantifier elimination. The argumentation shows that for every $t \in \mathbb{Q}$, $\exists \boldsymbol{y}.\mathcal{S}'(t, \boldsymbol{y})$ is equivalent to the condition

$$\bigvee_{\substack{R \subseteq [1, m+n] \\ |R| = n}} \det(\boldsymbol{D}_R(t)) \neq 0 \;\wedge\; \boldsymbol{D}(t) \cdot \boldsymbol{D}_R(t)^{-1} \cdot \boldsymbol{c}_R(t) \geq \boldsymbol{c}(t) \;. \tag{3}$$

Here, of course, we only know that $\boldsymbol{D}_R(t)^{-1}$ exists when $\det(\boldsymbol{D}_R(t)) \neq 0$. To express (3) using polynomials, we employ the identity $\boldsymbol{D}_R(t)^{-1} = \frac{\operatorname{adj}(\boldsymbol{D}_R(t))}{\det(\boldsymbol{D}_R(t))}$ whenever $\boldsymbol{D}_R(t)$ is invertible (equivalently, whenever $\det(\boldsymbol{D}_R(t)) \neq 0$). Thus, the set of all t with $\exists \boldsymbol{y}\colon \mathcal{S}'(t, \boldsymbol{y})$ can be defined by the following DNFLB Φ:

$$\begin{aligned} \bigvee_{\substack{R \subseteq [1, m+n] \\ |R| = n}} & \Big(\det(\boldsymbol{D}_R(x)) > 0 \;\wedge\; \boldsymbol{D}(x) \cdot \operatorname{adj}(\boldsymbol{D}_R(x)) \cdot \boldsymbol{c}_R(x) \geq \det(\boldsymbol{D}_R(x)) \cdot \boldsymbol{c}(x) \Big) \\ & \vee \Big(\det(\boldsymbol{D}_R(x)) < 0 \;\wedge\; \boldsymbol{D}(x) \cdot \operatorname{adj}(\boldsymbol{D}_R(x)) \cdot \boldsymbol{c}_R(x) \leq \det(\boldsymbol{D}_R(x)) \cdot \boldsymbol{c}(x) \Big) \;, \end{aligned} \tag{4}$$

where indeed all conditions are of the form $p(x) \geq 0$ or $p(x) > 0$ for some polynomials p. Note that here, we distinguish the cases $\det(\boldsymbol{D}_R(x)) < 0$ and $\det(\boldsymbol{D}_R(x)) > 0$ because moving a negative $\det(\boldsymbol{D}_R(x))$ to the other side of the inequality changes $\geq$ to $\leq$. Moreover, note that (in contrast to (3)) in the formulation (4), all terms are well-defined, independenly of whether the current choice of R makes $\boldsymbol{D}_R(x)$ invertible or not. To be precise, we obtain the DNFLB by subtracting the right-hand sides of the inequalities from the left-hand sides and multiplying the result by -1 to invert the inequality where necessary. The above form will suffice to give an estimate on the maximal degree and the maximal coefficient.

It is now clear that the coefficients (resp. degrees) appearing in Φ are exponential (resp. polynomial) in the bitsize of $\mathcal{S}$. The precise bounds promised in the Theorem are straightforward to deduce from standard bounds on determinants, see the full version for details. ◀

4.2 Root Separation

To show Lemma 4.1, it remains to be shown that any feasible DNFLB $\Phi(x)$ has a solution that is exponentially bounded. The key observation is that if r and r' are adjacent roots of a polynomial $p(x) \in \mathbb{Z}[x]$ and a constraint $p(x) \geq 0$ or $p(x) > 0$ is satisfied for some t for x with $r < t < r'$, then any number t' in the open interval (r, r') will also satisfy the constraint: The polynomial does not change its sign between r and r'. Thus, we can think of $\mathbb{R}$ as being split into (i) roots of p and (ii) intervals between roots of p (and the infinite intervals below

the smallest and above the largest root). Then whether $t \in \mathbb{Q}$ satisfies $p(x) \geq 0$ or $p(x) > 0$ only depends on which of those parts of $\mathbb{R}$ the number t belongs to. This remains true if we refine this decomposition of $\mathbb{R}$ according to *all* polynomials occurring in Φ.

In order to construct rational numbers with small numerator and denominator in intervals (r, r'), we will rely on a Root Separation Theorem, saying that polynomial roots are not too close. More specifically, we use Rump's Bound [27, Theorem 8.5.5]:

▶ **Theorem 4.4** (Rump's Bound [27, Theorem 8.5.5]). *Suppose $r, r' \in \mathbb{R}$ are distinct roots of a polynomial $p(x) \in \mathbb{Z}[x]$ with degree $d \in \mathbb{N}$. Then $|r - r'| > (d^{d+1}(1 + \|p(x)\|_1)^{2d})^{-1}$.*

We will also use an elementary fact about rational roots of integral polynomials. It is known as the Rational Root Theorem or Integral Root Test [20, Chapter IV, Prop. 3.3]:

▶ **Lemma 4.5** (Rational Root Theorem [20, Chapter IV, Prop. 3.3]). *Let $p(x) = c_n x^n + \cdots + c_0 \in \mathbb{Z}[x]$ be a polynomial. If $r = a/b$ is a root of p with a, b co-prime, then a divides c_0 and b divides c_n. In particular, $|a|, |b| \leq \mathrm{maxc}(p)$.*

Finally, we need a standard bound on all real roots of a polynomial [27, Corollary 8.3.2]. This is known as Cauchy's bound.

▶ **Lemma 4.6** (Cauchy's Bound [27, Corollary 8.3.2]). *If $r \in \mathbb{R}$ is a root of a polynomial $p \in \mathbb{Z}[x]$, then $|r| \leq 1 + \|p\|_1$.*

Let $r_1 < \cdots < r_k \in \mathbb{R}$ be all the real roots of polynomials occurring in Φ. Observe that if $t \in \mathbb{Q}$ satisfies $\Phi(x)$ and $t \in (r_i, r_{i+1})$, then any rational number in (r_i, r_{i+1}) must satisfy $\Phi(x)$, because none of the polynomials in Φ changes its sign between r_i and r_{i+1}. This allows us to bound a rational solution, by distinguishing the following cases:

1. Suppose $\Phi(x)$ is satisfied by some rational root r_i of p in Φ. Write $r_i = \frac{a}{b}$ with a, b co-prime. Then the Rational Root Theorem (Lemma 4.5) implies $|a|, |b| \leq \mathrm{maxc}(p)$.
2. Suppose $\Phi(x)$ has a rational solution in some interval (r_i, r_{i+1}). Since r_i and r_{i+1} are the roots of some polynomials p, q in $\Phi(x)$, as observed above, any rational number in (r_i, r_{i+1}) is also a solution to $\Phi(x)$. Note that r_i, r_{i+1} are roots of $p(x) \cdot q(x)$ and thus by Theorem 4.4, we have $|r_i - r_{i+1}| > \frac{1}{b}$ for some $b \in \mathbb{Z}$ that is exponentially bounded. Thus, there is an integer $a \in (br_i, br_{i+1})$. Note that then $\frac{a}{b}$ belongs to the interval (r_i, r_{i+1}) and thus satisfies $\Phi(x)$. Moreover, by the Cauchy Bound (Lemma 4.6), we also have an exponential bound $U \in \mathbb{R}$ on $|r_i|, |r_{i+1}|$ and thus on $|a| \leq |b|U$.
3. Suppose $\Phi(x)$ has a rational solution t outside of $[r_1, r_k]$. If $t > r_k$ then every rational number in $[r_k, \infty)$ is also a solution. Moreover, by Lemma 4.6, any rational number t' with $t' > 1 + \|p\|_1$ for every polynomial p occurring in Φ can be chosen, e.g. $t' = 2 + c$, where $c = \max\{\|p\|_1 \mid p \text{ polynomial in } \Phi\}$. On the other hand, if $t < r_1$, then $t' = -(2 + c)$ is a solution by an analogous argument.

This proves that any feasible $\Phi(x)$ has a rational solution that is exponentially bounded, which is what we will use in our application to Büchi VASS. The precise bounds of Lemma 4.1 are shown in the full version.

Proof sketch for Theorem 3.2. For showing Theorem 3.2, we can now use the fact that if $t \in \mathbb{Q}$ admits a solution $(t, \boldsymbol{s})$, then by our argument in the proof of Theorem 4.2, $\boldsymbol{s}^* := \frac{\mathrm{adj}(D_R(t))}{\det(D_R(t))} \cdot \boldsymbol{c}_R(t)$ is also a solution, for some subset $R \subseteq [1, m+n]$. This means that we can apply the bound on t and the bounds on $\mathrm{adj}(\boldsymbol{D}_R(x))$ and $\det(\boldsymbol{D}_R(x))$ established in the proof of Theorem 4.2 to bound the solution $\boldsymbol{s}^*$. We can ensure that all components of $\boldsymbol{c}_R(t)$ have the same denominator by increasing the bit size at most $\deg(\mathcal{S})$-fold. If we compute $\boldsymbol{s}^*$ starting from such a vector, we get an $\boldsymbol{s}^*$ where all components have the same denominator.

Since the entries in $\boldsymbol{D}_R$ all appear in $\mathcal{S}$ and it is well-known that the determinant has polynomial bit size in the bit size of a matrix, it follows that there exists a solution $(t, \boldsymbol{s})$ of polynomial bit size. The precise bounds promised in Theorem 3.2 are derived in the full version.

5 ω-Regular Separability

We use the results from Section 4 to prove Theorem 3.1. Note that the lower bounds in Theorem 3.1 easily follow from [2]. First, that paper already shows PSPACE-completeness of regular separability for one-dimensional Büchi VASS, which yields the PSPACE lower bound for fixed dimension ≥ 1. In fact, their argument also yields EXPSPACE-hardness in the general case: The full version [3, Appendix E.1] describes a simple reduction from intersection emptiness of one-dimensional VASS that accept by final state to regular separability of Büchi VASS, and the construction is the same in higher dimension. This yields EXPSPACE-hardness of regular separability of Büchi VASS, since intersection emptiness of VASS of arbitrary dimension that accept by final state is EXPSPACE-hard [23].

It remains to prove the upper bounds in Theorem 3.1. To adequately formulate our proofs, we need to introduce additional VASS-related concepts.

More on Büchi VASS. Let $\mathcal{V} = (Q, q_0, \Sigma, T, F)$ be a Büchi VASS. Consider a (possibly infinite) run $\sigma = (p_0, \boldsymbol{m}_0) \xrightarrow{w_1} (p_1, \boldsymbol{m}_1) \xrightarrow{w_2} \cdots$ of $\mathcal{V}$. The sequence of transitions in σ is called a *path* and has the form $\rho = (p_0, w_1, \delta_1, p_1)(p_1, w_2, \delta_1, p_2)\ldots$. If a path is finite and the source state of its first transition coincides with the target state of its last transition, then we call it a *loop*. Since a run is uniquely determined by the start configuration and its sequence of transitions, we also denote a run by $\sigma = (p_0, \boldsymbol{m}_0).\rho$. If σ is finite and $(p_\ell, \boldsymbol{m}_\ell)$ is its last configuration, then we sometimes write $\sigma = (p_0, \boldsymbol{m}_0).\rho.(p_\ell, \boldsymbol{m}_\ell)$ to emphasize this. The *effect* $\delta(\rho)$ of some finite path $\rho = (p_0, w_1, \delta_1, p_1)\ldots(p_{\ell-1}, w_\ell, \delta_\ell, p_\ell)$ is the sum of all induced counter changes, formally $\delta(\rho) = \sum_{1 \leq i \leq \ell} \delta_i$.

Recall that configurations of the Büchi VASS $\mathcal{V}$ are elements of the set $Q \times \mathbb{N}^d$. We call the second component in a configuration the *counter valuation* and refer to the i-th entry as the *value of counter* i. For a configuration cf and a set of counters $I \subseteq [1, d]$ we also use $cf[I]$ to denote the counter valuation of cf restricted to the counters in I. A configuration $(q, \boldsymbol{m})$ is *coverable* in $\mathcal{V}$ if there is a run starting in the initial configuration $(q_0, \boldsymbol{0})$ and reaching a configuration $(q, \boldsymbol{m}')$ with $\boldsymbol{m}' \geq \boldsymbol{m}$. Here, $\geq$ is defined component-wise.

Moreover we also consider a set of *extended configurations* $Q \times \mathbb{N}_\omega^d$, where $\mathbb{N}_\omega = \mathbb{N} \cup \{\omega\}$. Here ω is used to represent a counter value that has become unbounded. For an extended configuration $(q, \boldsymbol{m})$ we use $\omega(q, \boldsymbol{m}) \subseteq [1, d]$ to denote the set of counters valued ω in $\boldsymbol{m}$. Comparisons and arithmetic operations between integer values and ω behave as expected, treating ω as ∞. Formally, $\omega \geq \omega$, $\omega \geq z$, and $\omega + z = \omega$ for all $z \in \mathbb{Z}$. The *size* of an extended configuration is $|(q, \boldsymbol{m})| = \log_2 |Q| + \|\boldsymbol{m}\|_2 + d$, where the extra bit per counter encodes whether it has value ω or not. We also use the size of a unary encoding $\|(q, \boldsymbol{m})\|_1 = |Q| + \|\boldsymbol{m}\|_1 + d$.

The transition relation is also lifted to extended configurations in the expected manner. Formally, for $(q, \boldsymbol{m})$, $(q', \boldsymbol{m}') \in Q \times \mathbb{N}_\omega^d$ we have $(q, \boldsymbol{m}) \xrightarrow{w} (q', \boldsymbol{m}')$ if there is a transition $(q, w, \delta, q') \in T$ such that $\boldsymbol{m}' = \boldsymbol{m} + \delta$, where addition between elements of $\mathbb{N}_\omega^d$ and $\mathbb{Z}^d$ is defined component-wise. Furthermore, our definitions of runs, paths, loops, etc. carry over to *extended* versions over the set of extended configurations in a straightforward way. More precisely, an *extended run* is a sequence of extended configurations $cf_1 \xrightarrow{w_1} cf_2 \xrightarrow{w_2} \cdots$, an *extended path* is the underlying sequence of transitions of an extended run, and an *extended*

loop is a finite extended path starting and ending in the same state. To cover an extended configuration, intuitively, the ω-counters need to become unbounded, and the remaining counters need to be covered. Formally, an extended configuration $(q, \boldsymbol{m})$ is *coverable* in $\mathcal{V}$ if for every $k \in \mathbb{N}$ there is a run starting in the initial configuration $(q_0, \mathbf{0})$ and reaching a configuration $(q, \boldsymbol{m}_k) \in Q \times \mathbb{N}^d$ such that $\boldsymbol{m}_k[j] \geq k$ for every counter $j \in \omega(q, \boldsymbol{m})$ and $\boldsymbol{m}_k[i] \geq \boldsymbol{m}[i]$ for every counter $i \in [1, d] \setminus \omega(q, \boldsymbol{m})$.

Finally, we sometimes want to restrict only some counters of the VASS to stay non-negative. In this case, we consider extended configurations in $Q \times \mathbb{Z}_\omega^d$, where $\mathbb{Z}_\omega = \mathbb{Z} \cup \{\omega\}$. We say an extended run $\sigma = cf.\rho$ *remains non-negative* on counters $I \subseteq [1, d]$ if $cf'[I] \subseteq \mathbb{N}^{|I|}$ for all extended configurations cf' on σ.

Dyck Language. Towards the EXPSPACE upper bound of Theorem 3.1, a first step is to reduce the separability problem to a variant where one language is fixed to the Dyck language. The Dyck language D_n with n-letters is defined over the alphabet $\Sigma_n = \{a_i, \bar{a}_i \mid i \in [1, n]\}$. It contains those words w where, for every prefix v with $w = v.u$, we have at least as many letters a_i as $\bar{a}_i$. Thus, the letters behave like VASS counters and, indeed, the Dyck language is accepted by a single-state VASS $\mathcal{D}_n$ with n counters that increments the i-th counter upon seeing letter a_i and decrements the i-th counter upon seeing $\bar{a}_i$. If a VASS is defined over the Dyck alphabet Σ_n, we also call it n-visible. We will sometimes treat an n-visible VASS of dimension d as a $(d+n)$-dimensional VASS, and refer to the additional n counters as external. Note that this amounts to forming the product with $\mathcal{D}_n$. Given a path ρ, we use $\varphi(\rho)$ for the effect on the external counters in this product construction. Moreover, we write $\delta\varphi(\rho)$ to denote the combined effect on both internal and external counters, i.e. the $(d+n)$-dimensional vector $(\delta(\rho), \varphi(\rho))$.

To avoid an exponential blow-up, our reduction uses a variant of VASS whose transitions are labeled by compressed words. Essentially, the reduction takes $\mathcal{V}_1$ and $\mathcal{V}_2$ and produces a VASS $\mathcal{V}$ that is a product of $\mathcal{V}_1$ and $\mathcal{V}_2$. Moreover, it acts on its counters like $\mathcal{V}_1$; the input labels of $\mathcal{V}$ correspond to the counter updates of $\mathcal{V}_2$. Since the latter are binary-encoded, the new VASS will have binary encoded input words. Let us make this precise. A *label-compressed VASS* (lcVASS) $\mathcal{V}$ is a VASS, where the transitions are of the form $(p, a^m, \delta, q) \in Q \times \Sigma^* \times \mathbb{Z}^d \times Q$, where $a \in \Sigma$ and $m \in \mathbb{N}$ is given in binary. Thus, for an lcVASS $\mathcal{V}$, we define its *size* as $|\mathcal{V}| = |Q| + |F| + \sum_{(q,a^m,\delta,q') \in T} (\log_2(m) + \|\delta\|_2)$. The reduction that fixes the Dyck language is captured by the following lemma.

▶ **Lemma 5.1** ([2, Lemma 3.4]). *Given $\mathcal{V}_1$ and $\mathcal{V}_2$ over Σ, we can compute in time polynomial in $|\mathcal{V}_1| + |\mathcal{V}_2|$ an n-visible lcVASS $\mathcal{V}$ so that $L(\mathcal{V}_1) \,|\, L(\mathcal{V}_2)$ if and only if $L(\mathcal{V}) \,|\, D_n$. Here, n is the dimension of $\mathcal{V}_2$.*

The polynomial time bound is not mentioned in [2], but the simple construction they use (from [11]) clearly implies this bound. The latter separability problem $L(\mathcal{V}) \,|\, L(D_n)$ has been studied closely in [2] as well. They first show that $\mathcal{V}$ can be transformed so as to make the language $L(\mathcal{V})$ pumpable. For the resulting VASS, they show that $L(\mathcal{V}) \,|\, L(D_n)$ holds if and only if the so called Karp-Miller graph of $\mathsf{KM}(\mathcal{V})$ does not contain an inseparability witness. Unfortunately, the transformation required for pumpability involves another Karp-Miller graph construction, and therefore does not fit into the space bound we aim for (said graph can be of Ackermannian size in the worst case). Instead, we reformulate the witness.

▶ **Definition 5.2.** *Let $\mathcal{V}$ be an n-visible d-dimensional VASS. An* inseparability bloom *for $\mathcal{V}$ is a tuple $❀ = (q_f, I, \alpha, \beta, \gamma)$ with q_f a final state, loops α, β, γ starting and ending in q_f, and a partition of the counters $\Omega \uplus I = [1, d+n]$ so that*

(i) *for all* $\rho = \alpha, \beta, \gamma$*, we have* $\delta\varphi(\rho)[I] \geq 0$*,*
(ii) $\delta(\alpha) + \delta(\beta) + \delta(\gamma) \geq 0$
(iii) $\varphi(\alpha) + \varphi(\beta) \geq 0$*,*
(iv) *there is a* $t \in \mathbb{Q}$ *with* $\varphi(\alpha) + \varphi(\beta) + \varphi(\gamma) = t \cdot \varphi(\alpha)$*.*

The size *of the bloom is* $|❀| = \log_2 |Q| + |I| + |\alpha| + |\beta| + |\gamma|$*.*

A stem for ❀ *is an extended run* $cf.\sigma$ *of the product VASS* $\mathcal{V} \times \mathcal{D}_n$ *with the following properties: It ends in an extended configuration* $cf' = (q_f, \boldsymbol{m})$ *for some* $\boldsymbol{m} \in \mathbb{N}_\omega^{d+n}$ *with* $\Omega \subseteq \omega(cf')$*, and the counters in* $I \setminus \omega(cf)$ *remain non-negative when executing* σ *from* cf *resp.* α*,* β*, and* γ *from* cf'*.*

An inseparability flower $—❀ = (cf.\sigma, ❀)$ *consists of an inseparability bloom and a suitable stem. The size is* $|—❀| = \|cf\|_1 + |\sigma| + |❀|$*. The flower is* coverable *if the extended configuration* cf *is coverable in the product VASS* $\mathcal{V} \times \mathcal{D}_n$*.*

The following can be derived from the results in [2], refer to the full version for the details.

▶ **Lemma 5.3.** *Let* $\mathcal{V}$ *be* n*-visible. We have* $L(\mathcal{V}) \not| D_n$ *if and only if some inseparability flower is coverable.*

Our main result is the following. Note: the unary counter encoding strengthens the bound.

▶ **Theorem 5.4.** *If an inseparability flower is coverable in an* n*-visible lcVASS* $\mathcal{V}$ *of dimension* d*, then there is one of size at most* $|—❀| = 2^{|\mathcal{V}|^{\mathcal{O}((d+n)^2)}}$*.*

Main algortihm. We now have all ingredients to formulate the algorithm that proves the upper bounds in Theorem 3.1. We first describe the EXPSPACE upper bound. Given $\mathcal{V}_1$ and $\mathcal{V}_2$ whose languages we wish to separate, we first compute the lcVASS $\mathcal{V}$ using Lemma 5.1. This takes poly time. The task is to check $L(\mathcal{V}) \,|\, D_n$, where n is the dimension of $\mathcal{V}_2$. Using Lemma 5.3, we have to find an inseparability flower for $\mathcal{V}$ that is coverable. Theorem 5.4 bounds the size of the flowers we have to consider. We thus use non-determinism to find a flower of bounded size followed by a somewhat involved coverability check. Savitch's theorem [29] turns the non-deterministic algorithm into a deterministic one.

We detect a flower of bounded size as follows. We first guess the final state $q_f \in F$ and the partitioning of the counters $I \uplus \Omega$. With this information, we can guess the stem.

Towards obtaining a suitable stem, we start by guessing an extended configuration cf, whose non-ω-entries are at most doubly exponentially large. We can store such configurations in exponential space. If $\Omega \subseteq \omega(cf)$ fails, we abort. We now guess a path σ of doubly exponential length from cf to a configuration cf'. As we proceed, we store the length of the path, which only needs exponential space. We abort, if one of the following happens while guessing σ: a counter from $I \setminus \omega(cf)$ becomes negative, σ becomes too long, or the last state on σ is different from q_f. If we have not aborted until now, we have determined $cf.\sigma.cf'$ that may serve as a stem for a bloom with final state q_f and partition $I \uplus \Omega$.

Given the stem, we can finish the construction of the bloom by guessing the cycles $\rho = \alpha, \beta, \gamma$. The reason we proceed in this order is the following. The cycles are too long to be stored in exponential space. Instead, we check the non-negativity required by a stem on-the-fly, while constructing the cycles. To do so, we need the configuration cf', which we can store upon finishing the guess of σ above. While guessing the cycles, we store their length and the numbers $\delta(\rho)$ and $\varphi(\rho)$. It is readily checked that these numbers are bounded by

$$2^{|\mathcal{V}|} \cdot 2^{|\mathcal{V}|^{\mathcal{O}((d+n)^2)}} = 2^{|\mathcal{V}|^{\mathcal{O}((d+n)^2)}}.$$

This means we can store them in exponential space. We abort, if one of the following applies: an intermediate valuation becomes negative on $I \setminus \omega(cf)$, the path becomes too long, or the last state is different from q_f. We compute the operations and comparisons required by (i) to (iv) in Definition 5.2. For (iv), we start with counter $d+1$ and store the quotient $\frac{\varphi(\alpha)[d+1]+\varphi(\beta)[d+1]+\varphi(\gamma)[d+1]}{\varphi(\alpha)[d+1]}$ as the rational number t. As the numerator and denominator will be at most doubly exponential, we can store the number in exponential space. For the remaining entries, we only perform the required comparison. As noted above, they can be executed in polynomial time. If a comparison fails, we reject. If we have not rejected up to now, we have determined an inseparability flower while using space $|\mathcal{V}|^{\mathcal{O}((d+n)^2)} \leq 2^{|\mathcal{V}|\cdot\mathcal{O}((d+n)^2)} = 2^{poly(|\mathcal{V}_1|+|\mathcal{V}_2|)}$.

It remains to check whether this flower is coverable. There are two challenges. First, since cf is extended, we have a combination of a simultaneous unboundedness and a coverability problem. Second, the non-ω-entries in cf may be doubly exponentially large. We reduce the problem to simultaneous unboundedness, which is in EXPSPACE as shown by Demri [12, Theorem 4.6(I)]. The reduction uses a simple gadget that subtracts the counter valuation to be covered and, if succcessful, makes a new target counter j unbounded. In the end we check simultaneous unboundedness of $\omega(cf) \cup \{j\}$. To handle the large values, we utilize Lipton's construction [23], which allows a VASS to simulate EXPSPACE-computations. As a remark, simultaneous unboundedness cannot be expressed by a polynomial-sized formula in Yen's logic [33], a fact that was first observed in [12], which means we cannot just invoke the bounds in [1].

For the PSPACE upper bound, we merely need to observe that in fixed dimension, all our bounds on counter values become singly exponential. Moreover, for the gadget that subtracts counter values, we do not need the Lipton construction, as we can subtract exponentially bounded values directly using transitions. Finally, in fixed dimension, the simultaneous unboundedness check is also possible in PSPACE, as shown by Demri [12, Theorem 4.6(II)].

The remainder of the paper proves Theorem 5.4. SNLS help us deal with Constraint (iv).

6 Proof of Theorem 5.4

In this section we fix an n-visible lcVASS $\mathcal{V} = (Q, q_0, \Sigma_n, T, F)$. Note that since $\mathcal{V}$ is label-compressed, the effect on the external counters $\varphi(\rho)$ for a path ρ is also compressed. This matches the effect $\delta(\rho)$, which is anyway encoded in binary.

The proof is Rackoff-like, and we explain the analogy as we proceed. Like in Rackoff's upper bound for coverability [28], we reason over all (extended) configurations. Unlike Rackoff, however, we do not look at short covering sequences from a given configuration, but rather at small flowers rooted in said configuration. To bound the size while maximizing over all configurations, we measure each flower's size without considering the configuration it is rooted in. We therefore define the *flower bound*

$$\mathcal{B}_{\mathcal{V}} \;=\; \max_{cf} \min\{|\sigma| + |❀| \mid (cf.\sigma, ❀) \text{ is an inseparability flower}\}\ .$$

Theorem 5.4 is an immediate consequence of the following.

▶ **Lemma 6.1.** $\mathcal{B}_{\mathcal{V}} \leq 2^{|\mathcal{V}|^{\mathcal{O}((d+n)^2)}}$.

Step I: From length bounds to flower size bounds. Before we prove Lemma 6.1, let us see how it implies Theorem 5.4. Notice that Lemma 6.1 makes a statement about the lengths of the runs σ, α, β, γ, whereas Theorem 5.4 also promises a small starting configuration cf. Thus, it remains to construct a small starting configuration.

Proof of Theorem 5.4. To begin, assume $—❀ = (cf.\sigma, ❀)$ is coverable. By Lemma 6.1, there is another flower $(cf.\sigma', ❀')$ that is rooted in the same extended configuration and satisfies $|\sigma'| + |❀'| \leq \mathcal{B}_\mathcal{V}$. The extended configuration cf may not obey the desired bound. Thus, we replace it by cf' defined by $\omega(cf') = \omega(cf)$ and $cf'[j] = \min\{cf[j], \mathcal{B}_\mathcal{V} \cdot 2^{|\mathcal{V}|}\}$ for all $j \in I \setminus \omega(cf)$. Here, we keep the ω-entries, and for each non-ω-counter, we take the value of cf unless it is larger than $\mathcal{B}_\mathcal{V} \cdot 2^{|\mathcal{V}|}$, in which case we truncate to this value. We now claim that $—❀' = (cf'.\sigma', ❀')$ is still a coverable inseparability flower, and its size is at most doubly exponential (satisfying the desired bound). Coverability is immediate by $cf' \leq cf$. We also have $\|cf'\|_1 \leq 2^{2^{\mathcal{O}(|\mathcal{V}|)}}$. To be an inseparability flower, we have to check that the counters whose values we truncated when moving from cf to cf' remain non-negative while executing σ, and also $\sigma.\rho$ with $\rho = \alpha, \beta, \gamma$. This, however, is clear by the fact that $\sigma.\rho$ has length at most $\mathcal{B}_\mathcal{V}$, and each transition can subtract at most $2^{|\mathcal{V}|}$ tokens. As we have this budget available in cf', the run remains non-negative. ◀

In the remainder of the section, we prove Lemma 6.1. Similar to Rackoff's proof for coverability, we generalize the notion of flowers to admit negative counter values. Then we use an induction on the number of non-negative counters to establish a bound on the length of shortest generalized flowers. Let $i \in [0, d+n]$, $❀ = (q_f, I, \alpha, \beta, \gamma)$ a bloom, and $\Omega = [1, d+n] \setminus I$. We call an extended run $cf.\sigma$ of $\mathcal{V} \times \mathcal{D}_n$ an *i-stem for* $❀$, if it ends in an extended configuration cf' with $cf' = (q_f, \boldsymbol{m}')$ for some $\boldsymbol{m}'$, $\Omega \subseteq \omega(cf)$, and the counters in $([1, i] \cap I) \setminus \omega(cf)$ remain non-negative when executing σ from cf resp. α, β, and γ from cf' in $\mathcal{V} \times \mathcal{D}_n$. Here, we use $[1, 0] = \emptyset$. We call a pair $(cf.\sigma, ❀)$ consisting of an i-stem and an inseparability bloom an *i-inseparability flower*. Note that a $(d+n)$-inseparability flower is an inseparability flower as in Definition 5.2. We say the flower is *b-bounded* with $b \in \mathbb{N} \setminus \{0\}$, if the counters in $([1, i] \cap I) \setminus \omega(cf)$ are bounded by b along $cf.\sigma.\rho$ for all $\rho = \alpha, \beta, \gamma$. We wish to establish an estimate on the following function f, note that $f(d+n) = \mathcal{B}_\mathcal{V}$:

$$f(i) = \max_{cf} \min\{|\sigma| + |❀| \mid (cf.\sigma, ❀) \text{ is an } i\text{-inseparability flower}\} .$$

Step II: From value bounds to length bounds. Similar to Rackoff's proof, we will bound $f(i+1)$ in terms of $f(i)$, which will then yield the bound on $\mathcal{B}_\mathcal{V} = f(n+d)$. However, there is a key difference to Rackoff's proof: When constructing runs that, in the first $i+1$ coordinates stay non-negative, Rackoff argues that if such a run only uses counter values in $[0, b]$ on the first $i+1$ coordinates, then there is such a run of length at most $|Q| \cdot (b+1)^{i+1}$: whenever a combination of values in $[0, b]$ (and a control state) repeats, we can cut out the infix in between. Hence, *value bounds yield length bounds*. Our setting requires a different argument: Simply cutting out infixes in α, β, γ might spoil the properties (ii), (iii), and (iv) of i-inseparability flowers. Instead, we use our results on SNLS to obtain length bounds from value bounds. The technique is similar to some other generalizations of Rackoff's result by Yen [33, Lemma 3.5] Habermehl [14, Lemma 3.2] and Atig and Habermehl [1, Lemma 5]. However, the following proof also needs to work with non-linear constraints (which are also not present in Demri's extension of Rackoff's result [12]).

▶ **Lemma 6.2.** *Let $—❀ = (cf.\sigma, ❀)$ be a b-bounded i-inseparability flower. Then there is an i-inseparability flower $—❀' = (cf.\sigma', ❀')$ with $|\sigma'| + |❀'| \leq (2^{|\mathcal{V}|} \cdot |Q| \cdot b)^{\mathcal{O}((d+n)^6)}$.*

Proof. Let $—❀ = (cf.\sigma, ❀)$ be a b-bounded i-inseparability flower with $❀ = (q_f, I, \alpha, \beta, \gamma)$. Let Ω be the complement of I. Let $\widehat{cf}$ be the extended configuration reached by $cf.\sigma$.

Let $D = ([1, i] \cap I) \setminus \omega(cf)$ be the counters we wish to keep non-negative. By a *D-loop*, we mean an extended run $cf_1.\tau.cf_2$ of $\mathcal{V} \times \mathcal{D}_n$ where τ is a loop, $cf_1[D] = cf_2[D]$, and $cf_1[\Omega] = cf_2[\Omega]$. The D-loop is called *irreducible* if it does not contain further D-loops. By

the pigeonhole principle, any run that is longer than $u_{len} = |Q| \cdot (b+1)^{|D \cup \Omega|}$ contains a D-loop. This means the length of an irreducible D-loop is at most u_{len}. Moreover, these loops can have at most $2 \cdot 2^{|\mathcal{V}|} \cdot u_{len}$ distinct effects on each counter, where the leading 2 is for the distinction between positive and negative values. This analysis yields an upper bound of $u_{num} = (2^{|\mathcal{V}|+1} \cdot u_{len})^{d+n}$ on the number of irreducible D-loops with distinct effects. We show how to construct a flower as promised in the lemma.

We begin by cutting out irreducible D-loops from σ, yielding σ' of length at most u_{len}. The extended configuration cf' reached by $cf.\sigma'$ coincides with $\widehat{cf}$ on the ω-entries and on the non-ω-entries for D.

Next, we want to shorten $\rho = \alpha, \beta, \gamma$. To this end, we first decompose them into irreducible D-loops. We assume all such D-loops have distinct effects, otherwise we pick a representative. With the previous analysis, the result is $\rho_0, \rho_1, \ldots, \rho_v$ with $v \leq u_{num}$. Here, ρ_0 is the loop on the final state into which the irreducible D-loops $\rho_1, \ldots, \rho_v$ are inserted. Note that ρ_0 is not necessarily a D-loop. What we know, however, is that ρ_0 has a non-negative effect on the counters in I, due to Condition (i) in the definition of inseparability blooms. Since we want to be able to insert all the D-loops directly into ρ_0, the latter should still contain the same starting configurations for such loops as ρ, at least when only considering the counters in $D \cup \Omega$. Therefore we cannot guarantee that ρ_0 contains no D-loops, because cutting all of them out might reduce the number of such configurations visited by ρ_0. However, ρ_0 still has length at most u_{len}^2 by the following argument. If we mark in ρ_0 the first occurrence of each element from $Q \times [0,b]^{D \cup \Omega}$, then each infix leading from one such marker to the next has at most length u_{len}, because longer infixes still contain a D-loop that has no marked configuration and can therefore be removed. Since there are at most $|Q \times [0,b]^{D \cup \Omega}| = |Q| \cdot (b+1)^{|D \cup \Omega|} = u_{len}$ markers, we obtain the stated length bound of u_{len}^2 for ρ_0.

A vector $\boldsymbol{x} \in \mathbb{N}^v$ with $\boldsymbol{x}[0] > 0$ and $\boldsymbol{x}[j] \geq 0$ for $1 \leq j \leq v$ can now be turned into a run $\rho_{\boldsymbol{x}}$ by glueing together $\boldsymbol{x}[j]$-many instances of ρ_j. Note that also the base loop ρ_0 may be repeated. As the order of the transitions does not influence the effect of the run, $\delta\varphi(\rho) = \delta\varphi(\rho_{\boldsymbol{y}})$ holds, where $\boldsymbol{y} = \Psi(\rho)$ is the so-called Parikh vector that counts the occurrences of irreducible loops in ρ. As a consequence, we can directly define the effect on the vector $\boldsymbol{x}$, namely $\delta\varphi(\boldsymbol{x}) = \sum_{i=0}^{v} \boldsymbol{x}[j] \cdot \delta\varphi(\rho_j) \in \mathbb{Z}^{d+n}$.

With irreducible D-loops at hand, we can formulate the search for a small bloom as an SNLS. We define the vectors $\boldsymbol{x}_\alpha, \boldsymbol{x}_\beta, \boldsymbol{x}_\gamma$, and the following constraints:

$$\begin{aligned}
\boldsymbol{x}_\alpha[0], \boldsymbol{x}_\beta[0], \boldsymbol{x}_\gamma[0] &\geq 1 & \delta(\boldsymbol{x}_\alpha + \boldsymbol{x}_\beta + \boldsymbol{x}_\gamma)[1,d] &\geq 0 \\
\boldsymbol{x}_\alpha, \boldsymbol{x}_\beta, \boldsymbol{x}_\gamma &\geq 0 & \varphi(\boldsymbol{x}_\alpha + \boldsymbol{x}_\beta)[d+1, d+n] &\geq 0 \\
\delta\varphi(\boldsymbol{x}_\alpha)[I], \delta\varphi(\boldsymbol{x}_\beta)[I], \delta\varphi(\boldsymbol{x}_\gamma)[I] &\geq 0 & (\varphi(\boldsymbol{x}_\alpha + \boldsymbol{x}_\beta + \boldsymbol{x}_\gamma) - t \cdot \varphi(\boldsymbol{x}_\alpha))[d+1, d+n] &= 0
\end{aligned}$$

The constraints on the left say that we repeat the base loops at least once, and the remaining loops a non-negative number of times. The last constraint is Condition (i) in the definition of blooms. The constraints on the right correspond to the Conditions (ii) to (iv).

It is readily checked that $(\Psi(\alpha), \Psi(\beta), \Psi(\gamma))$ solves the SNLS. This means Theorem 3.2 applies and yields a small rational solution. To turn it into an integer solution, we observe that our SNLS is monotonic in the sense that if $(\boldsymbol{x}_\alpha, \boldsymbol{x}_\beta, \boldsymbol{x}_\gamma)$ is a solution, so is $(k\boldsymbol{x}_\alpha, k\boldsymbol{x}_\beta, k\boldsymbol{x}_\gamma)$ for any $k \geq 1$. We multiply the rational solution by the common denominator to obtain the integer solution $(\boldsymbol{x}'_\alpha, \boldsymbol{x}'_\beta, \boldsymbol{x}'_\gamma)$ with the associated loops α', β', γ'.

We argue that $\multimap^{\text{❀}'} = (cf.\sigma', \text{❀}')$ with $\text{❀}' = (q_f, I, \alpha', \beta', \gamma')$ is an i-inseparability flower. Remember that cf' is the configuration reached by $cf.\sigma'$. As cf and I have not changed, the only thing we have to show is the non-negativity for the counters in D. For σ', this follows from the non-negativity of σ, and the fact that we only cut-out D-loops. For $\rho' = \alpha', \beta', \gamma'$,

which are executed from cf', we argue as follows. The base loop ρ_0 has the same effect on the D-counters as ρ, because we obtained ρ_0 by cutting-out D-loops from ρ. The effect of ρ on even the entire set I is non-negative due to Condition (i) for blooms. This means if one repetition of ρ_0 stays non-negative from cf', arbitrarily many repetitions will. The one repetition stays non-negative, because ρ was non-negative from $\widehat{cf}$ (the extended configuration reached by $cf.\sigma$), and cf' coincides with $\widehat{cf}$ on D. Since the D-loops that we glue into ρ_0 come with a valuation of the counters in D, and this valuation keeps them non-negative in ρ, the entire ρ' stays non-negative.

We analyze the complexity of our system $\mathcal{S}$. It has at most $row(\mathcal{S}) \in \mathcal{O}(d+n)$ many rows, and note that the non-negativity constraints do not count towards the rows. There are at most $col(\mathcal{S}) \in \mathcal{O}(u_{num})$ many columns. The degree is $\deg(\mathcal{S}) = 1$. The maximal coefficient is bounded from above by the largest possible loop effect, $\max(\mathcal{S}) \leq 2^{|\mathcal{V}|} \cdot u_{len}^2$. Then, Theorem 3.2 gives us rational solutions $\boldsymbol{x}_\alpha, \boldsymbol{x}_\beta, \boldsymbol{x}_\gamma$ of the form $\boldsymbol{x}_\rho[i] = \frac{a}{K}$, meaning K is the common denominator of all entries, with

$$\begin{aligned}
\|\boldsymbol{x}_\rho\|_1 &\in (col(\mathcal{S}) \cdot \deg(\mathcal{S}) \cdot \max(\mathcal{S}))^{\mathcal{O}(\deg(\mathcal{S})^2 \cdot row(\mathcal{S})^4)} \\
&= (\mathcal{O}(u_{num}) \cdot 2^{|\mathcal{V}|} \cdot u_{len}^2)^{\mathcal{O}((d+n)^4)} \\
&= (\mathcal{O}((2^{|\mathcal{V}|+1} \cdot u_{len})^{d+n}) \cdot 2^{|\mathcal{V}|} \cdot u_{len}^2)^{\mathcal{O}((d+n)^4)} \\
&= (2^{|\mathcal{V}|} \cdot u_{len})^{\mathcal{O}((d+n)^5)} = (2^{|\mathcal{V}|} \cdot |Q| \cdot (b+1))^{\mathcal{O}((d+n)^6)} \\
&= (2^{|\mathcal{V}|} \cdot |Q| \cdot b)^{\mathcal{O}((d+n)^6)} .
\end{aligned}$$

We already argued that the integer vectors $K\boldsymbol{x}_\alpha$, $K\boldsymbol{x}_\beta$, $K\boldsymbol{x}_\gamma$ are also solutions with runs α', β', γ'. Since each entry of these vectors is smaller than $(2^{|\mathcal{V}|} \cdot |Q| \cdot b)^{\mathcal{O}((d+n)^6)}$, and we have at most $u_{num} = (2^{|\mathcal{V}|} \cdot |Q| \cdot (b+1))^{\mathcal{O}((d+n)^2)}$ many loops with maximal size $u_{len}^2 = |Q|^2 \cdot (b+1)^{2(d+n)}$, we get

$$\begin{aligned}
|\rho'| &\leq (2^{|\mathcal{V}|} \cdot |Q| \cdot b)^{\mathcal{O}((d+n)^6)} \cdot (2^{|\mathcal{V}|} \cdot |Q| \cdot (b+1))^{\mathcal{O}((d+n)^2)} \cdot |Q|^2 \cdot (b+1)^{2(d+n)} \\
&= (2^{|\mathcal{V}|} \cdot |Q| \cdot b)^{\mathcal{O}((d+n)^6)} .
\end{aligned}$$

◀

Step III: Rackoff-style induction. We now give the bound on $f(i)$ that we need to prove Lemma 6.1. In the base case, no counter has to remain non-negative and so we have a 1-bounded 0-inseparability flower. We employ the bound from Lemma 6.2.

▶ **Lemma 6.3.** $f(0) = (2^{|\mathcal{V}|}|Q|)^{\mathcal{O}((d+n)^6)}$.

In the induction step, and as in Rackoff's result, the bound takes the form of a recurrence.

▶ **Lemma 6.4.** $f(i+1) \leq (2^{|\mathcal{V}|} \cdot f(i))^{\mathcal{O}((d+n)^6)}$.

Proof. Consider an $(i+1)$-inseparability flower $—❀ = (cf.\sigma, ❀)$ with $❀ = (q_f, I, \alpha, \beta, \gamma)$ and $\Omega = [1, d+n] \setminus I$. Let $r(i) = 2^{|\mathcal{V}|} \cdot f(i)$ serve as an abbreviation. We proceed by a case distinction. If $—❀$ is $r(i)$-bounded, then Lemma 6.2 provides another $(i+1)$-inseparability flower $(cf.\sigma', ❀')$ with $|\sigma'| + |❀'| \leq (2^{|\mathcal{V}|} \cdot |Q| \cdot r(i))^{\mathcal{O}((d+n)^6)} = (2^{|\mathcal{V}|} \cdot f(i))^{\mathcal{O}((d+n)^6)}$. This satisfies the bound stated in the lemma.

If $—❀$ is not $r(i)$-bounded, then $\sigma.\rho$ with $\rho = \alpha, \beta, \gamma$ exceeds $r(i)$. We identify the first moment when this happens, say after ρ_1 and for the $(i+1)$-th counter. The case where the run exceeds the bound already in σ is simpler. The run decomposes into

$$cf \xrightarrow{\sigma} cf_1 \xrightarrow{\rho_1} cf' \xrightarrow{\rho_2} cf_2 .$$

We argue that also $(cf'.\rho_2, ❀)$ is an $(i+1)$-flower, which means $cf'.\rho_2$ is an $(i+1)$-stem for ❀. Since ρ is a loop, it returns to q_f. We have $\omega(cf) = \omega(cf')$ and since $\Omega \subseteq \omega(cf)$, we get $\Omega \subseteq \omega(cf')$. As ❀ is a bloom, ρ has a non-negative effect on the counters in I. This means $cf_1[I] \leq cf_2[I]$, and so the counters in $([1, i+1] \cap I) \setminus \omega(cf')$ remain non-negative when executing α, β, γ from cf_2 as they did from cf_1. Also ρ_2 remains non-negative from cf', because ρ remained non-negative from cf_1.

Since $(cf'.\rho_2, ❀)$ is an $(i+1)$-flower, it is an i-flower. The induction hypothesis yields another i-flower $—❀' = (cf'.\sigma', ❀')$ with $|\sigma'| + |❀'| \leq f(i)$. Let $❀' = (q'_f, I', \alpha', \beta', \gamma')$ and let Ω' be the complement of I'.

We argue that $—❀'$ is actually an $(i+1)$-flower. If the counter $i+1$ that exceeds the bound $r(i)$ does not belong to I', there is nothing to show. Otherwise, even $\sigma'.\alpha'.\beta'.\gamma'$ in succession could subtract at most $f(i) \cdot 2^{|\mathcal{V}|} = r(i)$ tokens from counter $i+1$. Since this counter carries more than $r(i)$ tokens, this leaves us with a positive balance.

We show that also $(cf.\sigma.\rho_1.\sigma', ❀')$ is an $(i+1)$-flower, meaning $cf.\sigma.\rho_1.\sigma'$ is an $(i+1)$-stem for ❀'. We have $\omega(cf) = \omega(cf')$ and so $\Omega' \subseteq \omega(cf')$ implies $\Omega' \subseteq \omega(cf)$. It remains to show that the counters in $([1, i+1] \cap I') \setminus \omega(cf)$ remain non-negative. Consider the prefix $\sigma.\rho_1$ executed from cf. For the counters that also belong to I, non-negativity holds as $—❀$ is an $(i+1)$-flower. Assume there was a counter in $([1, i+1] \cap I') \setminus \omega(cf)$ that did not belong to I. Then it belonged to Ω. But as $\Omega \subseteq \omega(cf)$, we had a contradiction. For the suffix σ' executed from cf', and for α', β', γ', non-negativity holds as $—❀'$ is an $(i+1)$-flower.

To estimate the size of the newly constructed flower rooted in cf, we assume $\sigma.\rho_1$ does not repeat configurations on the first $(i+1)$-counters. If it does, we cut out the infix and adapt the values of the counters that are allowed to fall below zero. Then the length of $\sigma.\rho_1$ is bounded by $|Q| \cdot r(i)^{i+1}$, and we have

$$|\sigma.\rho_1.\sigma'| + |❀'| \leq |Q| \cdot r(i)^{i+1} + f(i) \leq \ (2^{|\mathcal{V}|} \cdot f(i))^{\mathcal{O}(i+1)} \leq (2^{|\mathcal{V}|} \cdot f(i))^{\mathcal{O}((d+n)^6)} \ . \qquad ◀$$

It remains to solve the recurrence. Let $a = 2^{|\mathcal{V}|}$ and $b = \mathcal{O}((d+n)^6)$. We have

$$f(d+n) = (a \dots (a \cdot f(0))^b \dots)^b \leq (a^{d+n} \cdot f(0))^{b^{d+n}} \ .$$

Since $f(0) = (2^{|\mathcal{V}|} \cdot |Q|)^{\mathcal{O}((d+n)^6)}$, we obtain the promised $\mathcal{B}_\mathcal{V} = f(d+n) \leq 2^{|\mathcal{V}|^{\mathcal{O}((d+n)^2)}}$.

References

1. M. F. Atig and P. Habermehl. On Yen's path logic for Petri nets. *Int. J. Found. Comput. Sci.*, 22(4):783–799, 2011. doi:10.1142/S0129054111008428.
2. P. Baumann, R. Meyer, and G. Zetzsche. Regular separability in Büchi VASS. In *Proc. STACS*, volume 254 of *LIPIcs*, pages 9:1–9:19. Schloss Dagstuhl, 2023. doi:10.4230/LIPICS.STACS.2023.9.
3. Pascal Baumann, Roland Meyer, and Georg Zetzsche. Regular separability in Büchi VASS. *CoRR*, abs/2301.11242, 2023. doi:10.48550/arXiv.2301.11242.
4. M. Blockelet and S. Schmitz. Model checking coverability graphs of vector addition systems. In *Proc. MFCS*, volume 6907 of *LNCS*, pages 108–119. Springer, 2011. doi:10.1007/978-3-642-22993-0_13.
5. L. Clemente, W. Czerwiński, S. Lasota, and C. Paperman. Regular separability of parikh automata. In *Proc. ICALP*, volume 80 of *LIPIcs*, pages 117:1–117:13. Schloss Dagstuhl, 2017. doi:10.4230/LIPICS.ICALP.2017.117.
6. L. Clemente, W. Czerwiński, S. Lasota, and C. Paperman. Separability of reachability sets of vector addition systems. In H. Vollmer and B. Vallée, editors, *Proc. STACS*, volume 66 of *LIPIcs*, pages 24:1–24:14. Schloss Dagstuhl, 2017. doi:10.4230/LIPICS.STACS.2017.24.

7 W. Czerwiński and P. Hofman. Language inclusion for boundedly-ambiguous vector addition systems is decidable. In *Proc. CONCUR*, volume 243 of *LIPIcs*, pages 16:1–16:22. Schloss Dagstuhl, 2022. doi:10.4230/LIPICS.CONCUR.2022.16.

8 W. Czerwiński, P. Hofman, and G. Zetzsche. Unboundedness problems for languages of vector addition systems. In *Proc. ICALP*, volume 107 of *LIPIcs*, pages 119:1–119:15. Schloss Dagstuhl, 2018. doi:10.4230/LIPICS.ICALP.2018.119.

9 W. Czerwiński and S. Lasota. Regular separability of one counter automata. *Log. Methods Comput. Sci.*, 15(2), 2019. doi:10.23638/LMCS-15(2:20)2019.

10 W. Czerwiński, S. Lasota, R. Meyer, S. Muskalla, K. N. Kumar, and P. Saivasan. Regular separability of well-structured transition systems. In *Proc. CONCUR*, volume 118 of *LIPIcs*, pages 35:1–35:18. Schloss Dagstuhl, 2018. doi:10.4230/LIPICS.CONCUR.2018.35.

11 W. Czerwiński and G. Zetzsche. An approach to regular separability in vector addition systems. In *Proc. LICS*, pages 341–354. ACM, 2020. doi:10.1145/3373718.3394776.

12 S. Demri. On selective unboundedness of VASS. *JCSS*, 79(5):689–713, 2013. doi:10.1016/J.JCSS.2013.01.014.

13 E. M. Gurari and O. H. Ibarra. An NP-complete number-theoretic problem. *J. ACM*, 26(3):567–581, 1979. doi:10.1145/322139.322152.

14 P. Habermehl. On the complexity of the linear-time μ-calculus for Petri Nets. In *Proc. ICATPN*, volume 1248 of *LNCS*, pages 102–116. Springer, 1997. doi:10.1007/3-540-63139-9_32.

15 E. Keskin and R. Meyer. Separability and non-determinizability of WSTS. In *Proc. CONCUR*, volume 279 of *LIPIcs*, pages 8:1–8:17. Schloss Dagstuhl, 2023. doi:10.4230/LIPICS.CONCUR.2023.8.

16 E. Keskin and R. Meyer. On the separability problem of VASS reachability languages. In *To appear in Proc. of LICS*, 2024.

17 C. Köcher and G. Zetzsche. Regular separators for VASS coverability languages. In *Proc. FSTTCS*, volume 284 of *LIPIcs*, pages 15:1–15:19. Schloss Dagstuhl, 2023. doi:10.4230/LIPICS.FSTTCS.2023.15.

18 S. R. Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *Proc. STOC*, pages 267–281. ACM, 1982. doi:10.1145/800070.802201.

19 J.-L. Lambert. A structure to decide reachability in Petri nets. *Theor. Comput. Sci.*, 99(1):79–104, 1992. doi:10.1016/0304-3975(92)90173-D.

20 S. Lang. *Algebra, Rev. 3rd Ed.* Springer, New York, 2002.

21 J. Leroux. Vector addition system reachability problem: a short self-contained proof. In *Proc. POPL*, pages 307–316. ACM, 2011. doi:10.1145/1926385.1926421.

22 J. Leroux and S. Schmitz. Demystifying reachability in vector addition systems. In *Proc. LICS*, pages 56–67. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.16.

23 R. J. Lipton. The reachability problem requires exponential space. Technical Report 63, Yale University, 1976.

24 Angus Macintyre, Kenneth McKenna, and Lou van den Dries. Elimination of quantifiers in algebraic structures. *Advances in Mathematics*, 47(1):74–87, 1983. doi:10.1016/0001-8708(83)90055-5.

25 D. Marker. *Model Theory: An Introduction.* Springer, New York, 2002.

26 E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984. doi:10.1137/0213029.

27 B. Mishra. *Algorithmic Algebra.* Texts and Monographs in Computer Science. Springer, 1993. doi:10.1007/978-1-4612-4344-1.

28 C. Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6:223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

29 W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *JCSS*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.

30 A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1986.

31 R. S. Thinniyam and G. Zetzsche. Regular separability and intersection emptiness are independent problems. In *Proc. FSTTCS*, volume 150 of *LIPIcs*, pages 51:1–51:15. Schloss Dagstuhl, 2019. doi:10.4230/LIPICS.FSTTCS.2019.51.

32 V. Weispfenning. The complexity of almost linear diophantine problems. *J. Symb. Comput.*, 10(5):395–404, 1990. doi:10.1016/S0747-7171(08)80051-X.

33 H.-C. Yen. A unified approach for deciding the existence of certain Petri net paths. *Information and Computation*, 96(1):119–137, 1992. doi:10.1016/0890-5401(92)90059-O.

Decidability of Graph Neural Networks via Logical Characterizations

Michael Benedikt
University of Oxford, UK

Chia-Hsuan Lu
University of Oxford, UK

Boris Motik
University of Oxford, UK

Tony Tan
University of Liverpool, UK

Abstract

We present results concerning the expressiveness and decidability of a popular graph learning formalism, graph neural networks (GNNs), exploiting connections with logic. We use a family of recently-discovered decidable logics involving "Presburger quantifiers". We show how to use these logics to measure the expressiveness of classes of GNNs, in some cases getting exact correspondences between the expressiveness of logics and GNNs. We also employ the logics, and the techniques used to analyze them, to obtain decision procedures for verification problems over GNNs. We complement this with undecidability results for static analysis problems involving the logics, as well as for GNN verification problems.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Logic, Graph Neural Networks

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.127

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/2404.18151

Funding *Michael Benedikt*: This research was funded by EPSRC grant EP/T022124/1.

1 Introduction

Graph Neural Networks (GNNs) have become the most common model for learning functions that work on graph data. Like traditional neural networks, GNNs consist of a layered architecture where layer $k+1$ takes as input the output of layer k. Each layer computes a function from graph vertices to a vector of numerical values – the *feature vector*. Computation of the feature vector at layer $k+1$ for a node u is based on aggregating vectors for layer k of nodes v that are related to u in the source graph: for example aggregating vectors associated to nodes adjacent to u in the graph. In an aggregation, the vectors of the previous vectors may be transformed using linear functions. A layer can perform multiple aggregations – corresponding to different linear functions – and then combine them to get the feature vector for the next layer. The use of graph structure ensures that the computation of the network is *invariant*: depending only on the input graph and the node up to isomorphism. There are many variations of GNN. One key design choice is the kind of aggregation used - one can use "local aggregation", over the neighbors of a node, or aggregation over all nodes in the graph. A second design choice is the kind of numerical functions that can be applied to vector components, in particular the kind of *activation functions* that can be applied at each layer: e.g. ReLU, sigmoid, piecewise linear functions.

An important issue in the study of graph learning is the *expressiveness* of a learning model. What kinds of computations can a given type of GNN express? The first results in this line were about the *separating power of a graph learning model*: what pairs of nodes can be distinguished using GNNs within a certain class. For example, it is known that the separating power of standard GNN models is limited by the Weisfeiler-Leman (WL) test [16].

A finer-grained classification would characterize the functions computed by GNNs within a certain class, in terms of some formalism that is easier to analyze. Such characterizations are referred to as *uniform expressiveness results* and there has been much less work in this area. [1] provides a classification of a class of GNNs in terms of *modal logic*. The main result in [1] is a characterization of the classifiers expressible in first-order logic that can be performed with a GNN having only *local aggregation* and *truncated* ReLU *activations* over *undirected graphs*. They also provide a lower bound on the expressiveness of GNNs having in addition a "global aggregator", that sums over all nodes in the graph.

In this work we continue the line of work on uniform expressiveness. Our work improves on the state of the art in a number of directions:

- *From first order expressiveness to general expressiveness* In contrast to [1], we provide logical characterizations of *all* the functions that can be computed by certain GNN formalisms, not just the intersection with first-order logic. To do this we utilize logics that go beyond first order, but which are still amenable to analysis.
- *From expressiveness to verification* While we deal with GNNs that go beyond first-order logic, we can still obtain characterizations in a logic where the basic satisfiability problems are decidable. This provides us with decidability of a number of natural verification problems related to GNNs. In doing this, we show a surprising link between GNNs and recently-devised decidable logics going beyond first-order logic, so-called *Presburger logics*.
- *From undirected graphs to directed graphs* While prior work focused on undirected graphs, we explore how the expressiveness characterizations vary with GNNs that can recognize directionality of graph edges. The aim is to show that the logical characterizations and decidability are often independent of the restriction to undirected graphs.
- *From bounded to unbounded activations* We explore the impact of the activation functions. We begin with the case of *bounded activation functions*, like the truncated ReLU of [1], and establish characterizations and decidability results for GNNs using this function. We show both some contrasts and some similarity to the case of *unbounded activation functions*, including the standard ReLU. Here some, but not all, of the corresponding decidability results fail.

Related work. Logics have been used to characterized the separating power of GNN languages ("non-uniform expressiveness") for a number of years: see [8] for an overview. The recent [9] provides logical characterizations of GNNs with piecewise linear activations. The logic is not decidable; indeed our undecidability results imply that one cannot capture such GNNs with a decidable logic.

We employ logical characterizations to gain insight on two basic verification problems – whether a given classification can be achieved on some nodes or on all nodes. There is prior work on verification of GNNs, but it focuses on more complex (but arguably more realistic) problems, adversarial robustness. The closest paper to ours is the recent [14], which formalizes a broad set of problems related to verifying that the output is in a certain region in Euclidean space. [14] provides both decidability and undecidability theorems, but they are incomparable to ours both in the results and in the techniques. For example Theorem 1 of [14] shows undecidability of a satisfiability problem where we verify that certain nodes

output a particular value, over GNNs which always distinguish a node from its neighbor. Theorem 2 of [14] shows a decidability result with a different kind of specification, where the degree of input graphs is bounded.

Recently, logics that combine uninterpreted relations with Presburger arithmetic have been applied to the analysis of transformers – transducers that process strings [4, 2]. Since this is outside of the context of general graphs, the details of the logics that are employed are a bit different than those we consider, and the focus is not on the decidability border.

Organization. We formalize our GNN model and the basic logics we study in Section 2. We present results on logical characterizations of GNNs with "bounded activation functions" – like the truncated ReLU of [1]. We apply these characterizations to get decidability results. Section 4 turns to the case of unbounded activation functions, which includes the traditional ReLU function. Here we provide lower bounds for expressiveness, and then turn to the implications for decidability. Section 5 gives conclusions and discusses several open issues and directions for future work. For space reasons, *many details of the constructions and proofs are deferred to the full version, which is available on arxiv.*

2 Preliminaries

Let $\mathbb{N}$, $\mathbb{N}^+$, $\mathbb{Z}$, and $\mathbb{Q}$ be the set of natural numbers, positive natural numbers, integers, and rational numbers, respectively. For $p, q \in \mathbb{Z}$ and $p \leq q$, $[p\,, q]$ is the set of integers between p and q, including p and q. For $r \in \mathbb{Q}$, $\lceil r \rceil$ is the smallest integer greater than or equal to r.

For a function f mapping from $\mathbb{Q}$ to $\mathbb{Q}$ and a vector $b \in \mathbb{Q}^m$, $f(b)$ denotes that f is applied to each entry of b.

▶ **Definition 1.** *An* n-graph *is a tuple* $\left\langle V, E, \{U_c\}_{1 \leq c \leq n} \right\rangle$*, where* $n \in \mathbb{N}$ *is the number of vertex colors;* V *is a nonempty finite set of vertices;* $E \subseteq V \times V$ *is a set of edges; each* $U_c \subseteq V$ *is the set of* c*-colored vertices.*

Note that we allow self-loops in graphs, and a graph is by default a *directed graph.* For a graph $\mathcal{G}$, we say that $\mathcal{G}$ is a *undirected graph* if for all $v, u \in V$, $(v, u) \in E$ if and only if $(u, v) \in E$. For a vertex $v \in V$, we let $\mathcal{N}_{\text{out},\mathcal{G}}(v) := \{u |\ (v, u) \in E\}$ and refer to this as the set of *out-neighbors of* v. The set of *in-neighbors of* v, denoted $\mathcal{N}_{\text{in},\mathcal{G}}(v)$ are defined analogously.

Graph Neural Networks. We use a standard notion of "aggregate-combine" graph neural networks with rational coefficients. The only distinction from the usual presentation is that we allow GNNs to work over directed graphs, with separate aggregations over incoming and outgoing edges, while traditional GNNs work on undirected graphs.

▶ **Definition 2.** *An* n-graph neural network (GNN) *is a tuple*

$$\left\langle \{d_\ell\}_{0 \leq \ell \leq L}, \{f^\ell\}_{1 \leq \ell \leq L}, \{C^\ell\}_{1 \leq \ell \leq L}, \{A^\ell_x\}_{\substack{1 \leq \ell \leq L \\ x \in \{out,in\}}}, \{R^\ell\}_{1 \leq \ell \leq L}, \{b^\ell\}_{1 \leq \ell \leq L} \right\rangle,$$

where $L \in \mathbb{N}^+$ *is the number of layers; each* $d_\ell \in \mathbb{N}^+$*, called the* dimension *of the* ℓ^{th} *layer, requiring* $d_0 := n$*, the number of colors; each* $f^\ell : \mathbb{Q} \to \mathbb{Q}$*, the* activation function *of the* ℓ^{th} *layer; each* $C^\ell, A^\ell_x, R^\ell \in \mathbb{Q}^{d_\ell \times d_{\ell-1}}$*, the* coefficient matrices *of the* ℓ^{th} *layer; and* $b^\ell \in \mathbb{Q}^{d_\ell}$*, the* bias vector *of the* ℓ^{th} *layer.*

All the coefficients are rational. In order to have an effective representation of a GNN, we will also *assume that the activation functions are computable.*

▶ **Definition 3.** *For an n-GNN $\mathcal{A}$ and an n-graph $\mathcal{G}$, the* computation of $\mathcal{A}$ on $\mathcal{G}$ *is a sequence of* derived feature functions $\left\{\xi^{\ell}_{\mathcal{G}} : V \to \mathbb{Q}^{d_\ell}\right\}_{0\leq\ell\leq L}$ *defined inductively: for $\ell = 0$, if $v \in U_c$, then the c^{th} entry of $\xi^0_{\mathcal{G}}(v)$ is 1; otherwise, the entry is 0. For $1 \leq \ell \leq L$,*

$$\xi^{\ell}_{\mathcal{G}}(v) \;:=\; f^{\ell}\left(C^{\ell}\xi^{\ell-1}_{\mathcal{G}}(v) + \sum_{x\in\{out,in\}}\left(A^{\ell}_{x}\sum_{u\in\mathcal{N}_{x,\mathcal{G}}(v)}\xi^{\ell-1}_{\mathcal{G}}(u)\right) + R^{\ell}\sum_{u\in V}\xi^{\ell-1}_{\mathcal{G}}(u) + b^{\ell}\right).$$

For $v \in V$, $\xi^{\ell}_{\mathcal{G}}(v)$ is called the ℓ-feature vector of v, *and $\xi^{\ell}_{\mathcal{G},i}(v)$ is the i^{th} entry of $\xi^{\ell}_{\mathcal{G}}(v)$.*

That is, we compute the feature values of a node v at layer $\ell + 1$ by adding several components. One component aggregates over the ℓ-layer feature vector from the outgoing neighbors of v, and applies a linear transformation. Another component does the same for the incoming neighbors of v, a third does this for every node in the graph, while another applies a transformation to the ℓ-layer feature vector of v itself. The linear transformation can be different for each component, and in particular can be a zero matrix that just drops that component. The final component of the sum is the bias vector.

When the graph $\mathcal{G}$ is clear from the context, we omit it and simply write $\xi^{\ell}(v)$ and $\xi^{\ell}_{i}(v)$, and similarly when the graph $\mathcal{G}$ is clear from the context, write $\mathcal{N}_{\text{out}}(v)$ and $\mathcal{N}_{\text{in}}(v)$ for the in-neighbors and out-neighbors.

Note that in most presentations of GNNs, one deals with only undirected edges. The above definition degenerates in that setting to two aggregations per layer, with the aggregation over all nodes often referred to in the literature as the *global readout*.

In some presentations of GNNs, a *classification function*, which associates a final Boolean decision to a node, is included in the definition. In our case, we have separated out the classification function as an independent component in defining the expressiveness: see the last part of the preliminaries.

Classes of activation functions. Following prior work on analysis of GNNs, some of our results will deal with activation functions that are bounded in value:

▶ **Definition 4.** *We say that the function $f : \mathbb{Q} \to \mathbb{Q}$ is* eventually constant, *if there exists $t_{\mathsf{left}}, t_{\mathsf{right}} \in \mathbb{Q}$ satisfying $t_{\mathsf{left}} < t_{\mathsf{right}}$, called the* left and right thresholds of f, *such that for every $x \leq t_{\mathsf{left}}$, $f(x) = f(t_{\mathsf{left}})$; for every $x \geq t_{\mathsf{right}}$, $f(x) = f(t_{\mathsf{right}})$.*

A standard eventually constant function is the *truncated* ReLU *function*, which is 0 for negative reals, 1 for x greater than 1, and x otherwise [1]. There are other eventually constant functions that are used in practice: for example, the linear approximation of standard bounded functions used in graph learning, like the Sigmoid activation function. We will be interested in functions that are defined on the reals, but which preserve the rationals. The definition of eventually constant extends to such a function in the obvious way.

For a GNN with eventually constant activation functions, we use $\left\{t^{\ell}_{\mathsf{left}}\right\}_{1\leq\ell\leq L}$ and $\left\{t^{\ell}_{\mathsf{right}}\right\}_{1\leq\ell\leq L}$ to denote the left and right thresholds of the GNN's activation functions.

We also consider *unbounded activation functions*, such as the *standard* ReLU *function*, which is x for non-negative reals and 0 for negative reals.

Flavors of GNN. For a GNN $\mathcal{A}$, we say that $\mathcal{A}$ is *outgoing-only*, denoted by $\mathcal{O}$, if for every $1 \leq \ell \leq L$, A^{ℓ}_{in} is a zero matrix. $\mathcal{A}$ is *bidirectional*, denoted by $\mathcal{B}$, if there is no restriction on A^{ℓ}_{in}. $\mathcal{A}$ is *local*, denoted by $\mathcal{L}$, if for every $1 \leq \ell \leq L$, R^{ℓ} is a zero matrix. In the usual GNN terminology, this would mean that *there is no global readout*. $\mathcal{A}$ is

global, denoted by $\mathcal{G}$, if global readout is allowed. $\mathcal{A}$ is *eventually constant*, denoted by $\mathcal{C}$, if for $1 \leq \ell \leq L$, f^ℓ is an eventually constant function. Our results outside of eventually constant will deal with either *piecewise linear* activations, denoted $\mathcal{PW}$, truncated ReLU activations, denoted TrReLU, or standard ReLU activations. We use the following naming, $(\mathcal{O}|\mathcal{B})(\mathcal{L}|\mathcal{G})(\mathcal{C}|\mathcal{PW}|\mathsf{TrReLU}|\mathsf{ReLU})$-GNN, for the set of GNNs satisfying constraints given by the prefix. For example, $\mathcal{OLC}$-GNN is the set of outgoing-only, local, and eventually constant GNNs; $\mathcal{BGPW}$-GNN is the set of GNNs allowing both incoming and outgoing, global readout, and piecewise linear activations.

Classifiers and Boolean semantics. Our GNNs define vector-valued classification functions on nodes. But for comparing with expressiveness and in defining verification problems, we will often use a derived function from nodes to Booleans. We do this by thresholding at the end – below we use .5 for convenience, but other choices do not impact the results.

▶ **Definition 5.** *For a L-layer n-GNN $\mathcal{A}$, an n-graph $\mathcal{G}$, and a vertex $v \in V$, we say that $\mathcal{A}$* accepts *the tuple $\langle \mathcal{G}, v \rangle$, if $\xi^L_{\mathcal{G},1}(v) \geq 0.5$.*

Note that the global readout component can interact with the activation functions f^ℓ, which can behave very differently on translated values due to non-linearity – think of a typical f^ℓ as a piecewise linear function. Global readout can also interact with the classification threshold, pushing some values above the threshold while leaving others below.

Two-variable Modal Logic with Presburger Quantifiers. We review logic with Presburger quantifiers. The basic idea is to combine a decidable logic on uninterpreted structures, like two-variable logic or guarded logic, with the ability to perform some arithmetic on the number of elements. There are several formalisms in the literature that combine Presburger arithmetic with a decidable uninterpreted logic, some originating many years ago [11]. We will rely on a recent logic from [3], but we will need several variations of the underlying idea here.

▶ **Definition 6.** *A* Presburger quantifier *is of the form:*

$$\mathcal{P}(x) \;:=\; \sum_{i=1}^{k} \lambda_i \cdot \#_y[\varphi_i(x,y)] \;\circledast\; \delta,$$

where $\delta \in \mathbb{Z}$; each $\lambda_i \in \mathbb{Z}$; each $\varphi_i(x,y)$ is a formula with free variables x and y; $\circledast$ is one of $=$, $\neq$, $\leq$, $\geq$, $<$, or $>$. Note that $\mathcal{P}(x)$ has one free variable x.

We give the semantics of these quantifiers inductively, assuming a semantics for $\varphi_i(x,y)$. Given a graph $\mathcal{G}$ and a vertex $v \in V$, we say that $\mathcal{P}(x)$ holds in $\mathcal{G}, x/v$, denoted by $\mathcal{G} \models \mathcal{P}(v)$, if the following (in)equality holds in $\mathbb{Z}$.

$$\sum_{i=1}^{k} \lambda_i \cdot |\{u \in V |\ \mathcal{G} \models \varphi_i(v,u)\}| \;\circledast\; \delta$$

▶ Remark 7. Note that each Presburger quantifier can be rewritten as a Boolean combination of expressions which *only use the inequality symbol $\geq$ as $\circledast$*. For example, $(\#_y[\varphi(x,y)] = \delta)$ and $(\#_y[\varphi(x,y)] \geq \delta) \wedge \neg(\#_y[\varphi(x,y)] \geq \delta + 1)$ are semantically equivalent. Therefore it is sufficient to consider Presburger quantifiers which only use the inequality symbol $\geq$.

▶ Remark 8. We will make use of Presburger quantifiers that allow for rational coefficients of the form:

$$\widetilde{\mathcal{P}}(x) \;:=\; \kappa_0 + \sum_{i=1}^{k} \kappa_i \cdot \#_y[\varphi_i(x,y)] \;\circledast\; \lambda_0 + \sum_{i=1}^{\ell} \lambda_i \cdot \#_y[\psi_i(x,y)],$$

where each $\kappa_i, \lambda_i \in \mathbb{Q}$. This is a shorthand for the Presburger quantifier:

$$\mathcal{P}(x) \;:=\; \sum_{i=1}^{k} (D\kappa_i) \cdot \#_y[\varphi_i(x,y)] + \sum_{i=1}^{\ell} (-D\lambda_i) \cdot \#_y[\psi_i(x,y)] \;\circledast\; D(\lambda_0 - \kappa_0),$$

where D is the least common multiplier of the denominators of the coefficients in $\widetilde{\mathcal{P}}(x)$.

▶ **Definition 9.** *We give the syntax of* two-variable modal logic with Presburger quantifiers (MP^2) *over vocabulary τ. Formulas will have exactly one free variable, denoted x below:*

- *$\top$ is an MP^2 formula.*
- *for a unary predicate $U \in \tau$, $U(x)$ is an MP^2 formula.*
- *if $\varphi(x)$ is an MP^2 formula, then so is $\neg\varphi(x)$.*
- *if $\varphi_1(x)$ and $\varphi_2(x)$ are MP^2 formulas, then so is $\varphi_1(x) \wedge \varphi_2(x)$.*
- *if $\{\varphi_i(x)\}_{1\le i\le k}$ is a set of MP^2 formulas and $\{\epsilon_i(x,y)\}_{1\le i\le k}$ is a set of* guard atoms*, of form $E(x,y)$, $E(y,x)$, or $\top$, then $\left(\sum_{i=1}^{k} \lambda_i \cdot \#_y[\epsilon_i(x,y) \wedge \varphi_i(y)] \;\circledast\; \delta\right)$ is also an MP^2 formula. $\{\epsilon_i(x,y)\}_{1\le i\le k}$ are the* guards *of the formula. Consistent with the restriction we announced on the logic, we consider the result as a formula with free variable x: if all ϵ_i are $\top$ it returns either every node or no node.*

The semantics of the Boolean connectives is as usual, while the semantics of the Presburger quantifiers is given by Definition 6.

An MP^2 formula $\varphi(x)$ is an n-formula if its vocabulary consists of n unary predicates. We use abbreviations $\vee$ and $\to$ as usual. Note that the guarded universal quantifier $\forall y\, E(x,y) \to \varphi(y)$ can be expressed as $(1 \cdot \#_y[E(x,y) \wedge \neg\varphi(y)] = 0)$, and the guarded existential quantifier $\exists y\, E(x,y) \wedge \varphi(y)$ can be expressed as $(1 \cdot \#_y[E(x,y) \wedge \varphi(y)] \ge 1)$.

The logic MP^2 combines Presburger arithmetic and quantification over the model. Thus one might worry that it has an undecidable satisfiability problem. And indeed, we will show this: see Theorem 29. An idea to gain decidability is to impose that the quantification is *guarded* – again, the underlying idea is from [3]. The logic $\mathcal{L}$-MP^2 (or "*local* MP^2") is obtained by excluding $\top$ as a guard. Analogously to what we did with GNNs, we use $\mathcal{L}$ to indicate that quantification is "local".

The logic $\mathcal{L}$-MP^2 is contained in the following logic, defined in [3]:

▶ **Definition 10.** *The syntax of the* guarded fragment of two-variable logic with Presburger quantifiers (GP^2) *over colored graph vocabulary τ starts with arbitrary atoms over the vocabulary, with the usual connective closure and the following rules for quantifiers:*

- *if $\varphi(x)$ is a GP^2 formula, then so are $\forall x\, \epsilon(x) \to \varphi(x)$ and $\exists x\, \epsilon(x) \wedge \varphi(x)$, where ϵ is either $U(x)$ or $x = x$ for some unary predicate $U \in \tau$.*
- *if $\varphi(x,y)$ is a GP^2 formula, then so are $\forall x\, \epsilon(x,y) \to \varphi(x,y)$ and $\exists x\, \epsilon(x,y) \wedge \varphi(x,y)$, where $\epsilon(x,y)$ is one of $E(x,y)$ or $E(y,x)$.*
- *if $\{\varphi_i(x,y)\}_{1\le i\le k}$ is a set of GP^2 formulas and $\{\epsilon_i(x,y)\}_{1\le i\le k}$ is a set of formulas, each of form $E(x,y)$ or $E(y,x)$, then $\left(\sum_{i=1}^{k} \lambda_i \cdot \#_y[\epsilon_i(x,y) \wedge \varphi_i(x,y)] \;\circledast\; \delta\right)$ is also a GP^2 formula.*

The main difference between the logic $\mathcal{L}$-MP^2 and the logic above is that the former is "modal", restricting to one-variable formulas, and allowing two variables only in the guards. While in the logic above we can build up more interesting two variable formulas, for example conjoining two guards.

We will make use of the following prior decidability result:

▶ **Theorem 11** ([3], Theorem 10). *The finite satisfiability problem of* GP^2 *is decidable.*

From this we easily derive the decidability of $\mathcal{L}$-MP^2:

▶ **Corollary 12.** *The finite satisfiability problem of* $\mathcal{L}$-MP^2 *is decidable.*

Notions of expressiveness for GNNs and MP^2 Formulas. Recalling that we have a node-to-Boolean semantics available for both logical formulas and GNNs (via thresholding), we use the term *n-specification* for either a n-GNN or a n-MP^2 formula.

▶ **Definition 13.** *If S_1, S_2 are n-GNNs, they are said to be* equivalent *if they accept the same nodes within n-graphs. If S_1 is a GNN and S_2 a node formula in some logic, we say S_1 and S_2 are equivalent if for every n-graph $\mathcal{G}$ and vertex $v \in V$, S_1 accepts $\langle \mathcal{G}, v \rangle$ if and only if $\mathcal{G}, v$ satisfies S_2.*

The notions of two languages of specifications being equally expressive, or equally expressive over undirected graphs, is defined in the obvious way:

Verification Problems for GNNs. We focus on two verification problems. The first is the most obvious analog of satisfiability for GNNs, whether it accepts some node of some graph:

▶ **Definition 14.** *For an n-GNN $\mathcal{A}$, we say that $\mathcal{A}$ is* satisfiable*, if there exist an n-graph $\mathcal{G}$ and a vertex $v \in V$, such that $\mathcal{A}$ accepts $\langle \mathcal{G}, v \rangle$.*

We will also consider a variation of the problem which asks whether a GNN accepts every node of some graph:

▶ **Definition 15.** *For an n-GNN $\mathcal{A}$, we say that $\mathcal{A}$ is* universally satisfiable*, if there exist an n-graph $\mathcal{G}$, such that for every vertex $v \in V$, $\mathcal{A}$ accepts $\langle \mathcal{G}, v \rangle$.*

Two GNNs $\mathcal{A}$ and $\mathcal{B}$ are equivalent if they accept the same tuples. Note that, like satisfiability and unlike universal satisfiability, this does not require a quantifier alternation. For brevity we will not state results for equivalence, but *it can easily be seen that both our positive and negative results on satisfiability also apply to equivalence.*

3 Characterization and decidability of GNNs with eventually constant activation functions

In this section, we only consider GNNs with eventually constant activations. In Section 3.1, we establish a key tool to analyzing these GNNs: we show that the set of possible activation values is finite, and one can compute an overapproximation of this set. We use this for two purposes. First we give a decidability result for GNNs with eventually constant activations and only local aggregation, and then we show that even with global aggregation we get an equivalence of the GNNs in expressiveness with a logic.

In Section 3.2, we show that the finite satisfiability of MP^2 is undecidable. Using the expressiveness characterization, this will imply that satisfiability problems for global GNNs are undecidable. These results were presented for GNNs and logics on directed graphs. In Section 3.3 we use the logical characterizations to show that they also apply to the standard setting for GNNs of undirected graphs.

3.1 Decidability of satisfiability problems for GNNs with eventually constant functions, via logic

We now come to one of the crucial definitions in the paper, the spectrum of a GNN.

▶ **Definition 16.** *For a $\mathcal{BGC}$-GNN $\mathcal{A}$ and $0 \leq \ell \leq L$, the* ℓ-spectrum *of $\mathcal{A}$, denoted by $\mathcal{S}^\ell$, is the set $\left\{\xi^\ell(v) \middle| \text{ for every } n\text{-graph } \mathcal{G} \text{ and vertex } v \in V\right\}$.*

That is, the ℓ-spectrum is the range of the feature vectors computed at layer ℓ, as we range over all input graphs and nodes. We show that the spectrum is actually finite, and a finite superset is computable:

▶ **Theorem 17.** *For every $\mathcal{BGC}$-GNN $\mathcal{A}$ and $0 \leq \ell \leq L$, the ℓ-spectrum of $\mathcal{A}$ is finite. We can compute a finite superset of the ℓ-spectrum from the specification of $\mathcal{A}$.*

We give some intuition for the proof. Our effective overapproximation of the spectrum will simulate the computation of the GNN, and will be defined inductively on the layers. Recall that a $\mathcal{BLC}$-GNN is given by dimensions $\{d_\ell\}_{0\leq\ell\leq L}$, activation functions $\left\{f^\ell\right\}_{1\leq\ell\leq L}$, coefficient matrices for transforming the prior node value $\left\{C^\ell\right\}_{1\leq\ell\leq L}$, coefficient matrices for local aggregation $\left\{A^\ell_x\right\}_{\substack{1\leq\ell\leq L\\ x\in\{\text{out},\text{in}\}}}$, coefficient matrices for global readout $\left\{R^\ell\right\}_{1\leq\ell\leq L}$, and bias vectors $\left\{b^\ell\right\}_{1\leq\ell\leq L}$.

▶ **Definition 18.** *For a $\mathcal{BLC}$-GNN $\mathcal{A}$ and $0 \leq \ell \leq L$, the set $\uparrow\mathcal{S}^\ell$ is defined as follows:*

$$
\begin{aligned}
\uparrow\mathcal{S}^0 &:= \{0,1\}^{d_0}\\
\uparrow\mathcal{S}^\ell_s &:= \left\{ f^\ell\left(C^\ell s + \sum_{x\in\{out,in\}} A^\ell_x \sum_{s'\in\uparrow\mathcal{S}^{\ell-1}} s' n^{A,s'}_x + R^\ell \sum_{s'\in\uparrow\mathcal{S}^{\ell-1}} s' n^{R,s'} + b^\ell\right) \middle| \; n^{A,s'}_x, n^{R,s'} \in \mathbb{N} \right\}\\
\uparrow\mathcal{S}^\ell &:= \bigcup_{s\in\uparrow\mathcal{S}^{\ell-1}} \uparrow\mathcal{S}^\ell_s
\end{aligned}
$$

We show that the set $\uparrow\mathcal{S}^\ell$ overapproximates the ℓ-spectrum:

▶ **Lemma 19.** *For every n-$\mathcal{BLC}$-GNN $\mathcal{A}$ and $0 \leq \ell \leq L$, for every n-graph $\mathcal{G}$ and vertex $v \in V$, there exists $s \in \uparrow\mathcal{S}^\ell$, such that $\xi^\ell(v) = s$.*

It is quite straightforward to see that every element of the spectrum is captured. It is an overapproximation because different integers that we sum in an inductive step may not be realized in the same graph.

We can show by induction on the number of the layers that the set is finite – regardless of computability of the activation functions!

▶ **Lemma 20.** *For every n-$\mathcal{BLC}$-GNN $\mathcal{A}$ and $0 \leq \ell \leq L$, $\uparrow\mathcal{S}^\ell$ has finite size and can be computed.*

In the inductive step, we have a finite set of rationals, thus some fixed precision. We take some integer linear combinations and we will obtain an infinite set of values, but only finitely many between the left and right thresholds of the eventually constant activations. Thus when we apply the activation functions to these values, we will get a finite set of rational values – since the activation functions map rationals to rationals.

▶ Remark 21. The restriction to rational coefficients is crucial in the argument. Consider the following 1-layer 1-$\mathcal{BL}$TrReLU-GNN. The dimensions are $d_0 = d_1 = 1$; the coefficient matrix C^1 is a zero matrix; $\left(A^1_{\text{out}}\right)_{1,1} = \sqrt{2}$; $\left(A^1_{\text{in}}\right)_{1,1} = -1$; the bias vector b^1 is a zero vector. It is not difficult to see that its 1-spectrum is $\left\{\mathsf{TrReLU}\left(\sqrt{2}k_1 - k_2\right) \middle| \; k_1, k_2 \in \mathbb{N}\right\}$, whose size is infinite since $\sqrt{2}$ is irrational.

▶ Remark 22. Even simple GNNs may have exponential size spectra. For example, let $\mathcal{A}_k$ be a 1-layer 1-$\mathcal{BL}$TrReLU-GNN defined as follows: the dimensions are $d_0 = d_1 = 1$; the coefficient matrices C^1 and A^1_{in} are zero matrices; $\left(A^1_{\text{out}}\right)_{1,1} = k^{-1}$; the bias vector b^1 is a zero vector. By definition, its 1-spectrum is $\left\{ik^{-1} \middle| \; i \in [0\,, k]\right\}$, whose size is $k+1$. But the description of $\mathcal{A}_k$ is only linear in $\log k$.

For GNNs with truncated ReLU and only local aggregation, there is a matching upper bound, as discussed after Theorem 25.

We now give several applications of the spectrum result. First we can use the finiteness of the spectrum to get a characterization of the expressiveness of $\mathcal{BGC}$-GNN and logic:

▶ **Theorem 23.** *For every n-$\mathcal{BGC}$-GNN $\mathcal{A}$, there exists an n-MP^2 formula $\Psi_{\mathcal{A}}(x)$, effectively computable from the description of $\mathcal{A}$, such that $\mathcal{A}$ and $\Psi_{\mathcal{A}}(x)$ are equivalent. In the case we start with an n-$\mathcal{BLC}$-GNN, the formula we obtain is in n-$\mathcal{L}$-MP^2.*

This expressiveness equivalence will be useful in getting further decidability results, as well as separations in expressiveness, for GNNs. The idea of the proof of the theorem is that we have only finitely many elements in the overapproximation set to worry about, so we can fix each in turn and write a formula for each.

Recall that finite satisfiability of $\mathcal{L}$-MP^2 is decidable by Corollary 12. Combining this with Theorem 23 we get decidability of satisfiability for $\mathcal{BLC}$-GNN:

▶ **Theorem 24.** *The satisfiability problem for $\mathcal{BLC}$-GNNs is decidable.*

A more realistic analysis of complexity requires stronger assumptions on the activation functions. For now we note only one special case, where everything is a truncated ReLU:

▶ **Theorem 25.** *For $\mathcal{BLC}$-GNNs with truncated* ReLU *activations, the satisfiability problem is* PSPACE*-complete. It is* NP*-complete when the number of layers is fixed.*

We briefly discuss the PSPACE upper bound argument. We can show that for an arbitrary input graph, there are only exponentially many activation values, each representable with a polynomial number of bits. We also show, via an "unravelling construction", a common technique use in analysis of modal and guarded logics [12, 7], that a satisfying model can be taken to be a tree of polynomial depth and branching. These two facts immediately give an elementary bound, since we could guess the tree and the activation values. We can improve to PSPACE by exploring a satisfying tree-like model on-the-fly: again, this is in line with the PSPACE algorithm for modal logic [12].

The PSPACE lower bound is established by embedding the description logic $\mathcal{ALC}$ into $\mathcal{L}$-MP^2. PSPACE-hardness will follow from this, since concept satisfiability problem of $\mathcal{ALC}$ with one role is PSPACE-hard [15]. The NP upper bound will use the same on-the-fly algorithm as in the PSPACE case, just observing that for fixed depth it can be implemented in NP. A direct encoding of SAT gives the lower bound.

The following converse to Theorem 23 shows that the logic is equally expressive as the GNN model:

▶ **Theorem 26.** *For every n-MP^2 formula $\Psi(x)$, there exists an n-$\mathcal{BGC}$TrReLU-GNN $\mathcal{A}_\Psi$, such that $\Psi(x)$ and $\mathcal{A}_\Psi$ are equivalent. If we start with an n-$\mathcal{L}$-MP^2 formula, we obtain an n-$\mathcal{BLC}$TrReLU-GNN.*

The idea of the proof is induction on the formula structure. For each subformula there will be an entry of a feature vector for the GNN which represents the subformula, in the sense that – for the final layer – its value is 1 if the subformula holds, or 0 otherwise. We will have an entry for each subformula at every iteration, but as we progress to later layers of the GNN, more of these entries will be correct with respect to the corresponding subformula. In an inductive case for a Presburger quantifier that uses some coefficients λ_i, the corresponding matrix will be multiplying certain quantifies by λ_i. Note that this translation is polynomial time, thus the size of the corresponding GNN is polynomial in the formula.

Putting together the two translation results, we have:

▶ **Corollary 27.** *The logic MP^2 and $\mathcal{BGC}$-GNNs are expressively equivalent, as are $\mathcal{L}$-MP^2 and $\mathcal{BLC}$-GNNs.*

The translations also tell us that *the expressiveness of GNNs with truncated* ReLU *is the same as that of GNNs with arbitrary eventually constant activations* – provided we use the Boolean semantics based on thresholds.

Recall from Corollary 12 that finite satisfiability for the richer logic GP^2, allowing unguarded unary quantification and containing $\mathcal{L}$-MP^2, is decidable. Using this and the expressiveness characterization gives decidability of universal satisfiability for these GNNs:

▶ **Theorem 28.** *The universal satisfiability problem of $\mathcal{BLC}$-GNNs is decidable.*

3.2 Undecidability of MP^2, and of GNNs with truncated ReLU and global readout

Note that we claimed that the spectrum is finite for GNNs with eventually constant activations, even when they have global readout. And we could compute a finite overapproximation of the spectrum. But in our decidability argument for $\mathcal{BLC}$-GNN, we required further the ability to decide membership in the spectrum for any fixed rational, and for this we utilized decidability of the logic. So what happens to decidability of the GNNs – or the corresponding logic – when global readout is allowed? We show undecidability of finite satisfiability for the logic MP^2, and of the corresponding GNN satisfiability problem. First for the logic:

▶ **Theorem 29.** *The finite satisfiability problem of MP^2 is undecidable.*

For the proof we apply an approach based on ideas in [3], using a reduction from Hilbert's tenth problem.

▶ **Definition 30.** *A* simple equation system *ε (with n variables and m equations) is a set of m equations of one of the forms $v_{i_1} = 1$, $v_{i_1} = v_{i_2} + v_{i_3}$, or $v_{i_1} = v_{i_2} \cdot v_{i_3}$, where $1 \leq i_1, i_2, i_3 \leq n$. We say the system ε is solvable if it has a solution in $\mathbb{N}$.*

▶ **Lemma 31.** *For every simple equation system ε with n variables and m equations, there exists an $(n+m)$- MP^2 formula $\Psi_\varepsilon(x)$ such that ε has a solution in $\mathbb{N}$ if and only if $\Psi_\varepsilon(x)$ is finitely satisfiable.*

Since the solvability (over $\mathbb{N}$) of simple equation systems is undecidable, Theorem 29 follows. From the theorem and Corollary 27 we obtain undecidability of static analysis for GNNs with global readout:

▶ **Theorem 32.** *The satisfiability problem of $\mathcal{BG}$TrReLU-GNNs is undecidable.*

We give the reduction used in Lemma 31, leaving the verification for the reader. The vocabulary of the constructed formulas $\Psi_\varepsilon(x)$, where ϵ is the simple equation system, consists of unary predicates P_i and U_j, where $1 \leq i \leq m$ and $1 \leq j \leq n$. For $1 \leq i \leq m$, we define $\varphi_i(x)$ depending on the i^{th} equation in ε.

- If the equation is $v_j = 1$, then $\varphi_i(x) := (\#_y[P_i(y) \wedge U_j(y)] = 1)$.
- If the equation is $v_{j_1} = v_{j_2} + v_{j_3}$, then $\varphi_i(x) := (\#_y[\psi_i(y)] - \#_y[\top] = 0)$ where

$$\begin{aligned}\psi_i(y) \;:=\; & (P_i(y) \wedge (U_{j_2}(y) \vee U_{j_3}(y)) \rightarrow (\#_x[E(y,x) \wedge P_i(x) \wedge U_{j_1}(x)] = 1)) \;\wedge \\ & (P_i(y) \wedge U_{j_1}(y) \rightarrow (\#_x[E(x,y) \wedge P_i(x) \wedge (U_{j_2}(x) \vee U_{j_3}(x))] = 1))\,.\end{aligned}$$

- If the equation is $v_{j_1} = v_{j_2} \cdot v_{j_3}$, then $\varphi_i(x) := (\#_y[\psi_i(y)] - \#_y[\top] = 0)$ where

$$\begin{aligned}\psi_i(y) \;:=\; & (P_i(y) \wedge U_{j_2}(y) \rightarrow (\#_x[E(y,x) \wedge P_i(x) \wedge U_{j_1}(x)] - \#_x[P_i(x) \wedge U_{j_3}(x)] = 0)) \;\wedge \\ & (P_i(y) \wedge U_{j_1}(y) \rightarrow (\#_x[E(x,y) \wedge P_i(x) \wedge U_{j_2}(x)] = 1))\,.\end{aligned}$$

We now define $\Psi_\varepsilon(x)$:

$$\begin{aligned}\psi^{disj}(x) \;&:=\; \bigwedge_{1 \leq i_1 < i_2 \leq m} (\#_y[P_{i_1}(y) \wedge P_{i_2}(y)] = 0) \;\wedge \bigwedge_{1 \leq j_1 < j_2 \leq n} (\#_y[U_{j_1}(y) \wedge U_{j_1}(y)] = 0) \\ \psi^{eq}(x) \;&:=\; \bigwedge_{\substack{1 \leq i_1 < i_2 \leq m \\ 1 \leq j \leq n}} (\#_y[P_{i_1}(y) \wedge U_j(y)] - \#_y[P_{i_2}(y) \wedge U_j(y)] = 0) \\ \Psi_\varepsilon(x) \;&:=\; \psi^{disj}(x) \wedge \psi^{eq}(x) \wedge \bigwedge_{1 \leq i \leq m} \varphi_i(x).\end{aligned}$$

Using a similar reduction, we obtain undecidability for universal satisfiability:

▶ **Theorem 33.** *The universal satisfiability problem of $\mathcal{BG}$TrReLU-GNNs is undecidable.*

3.3 Variations for the undirected case

Thus far we have been dealing with both logics and GNNs that work over directed graphs. We now show that all of the prior results apply to undirected graphs, the standard setting for GNNs.

We can enforce undirectedness within the larger decidable logic GP^2 to obtain decidability:

▶ **Corollary 34.** *The finite satisfiability problem of $\mathcal{L}$-MP^2 over undirected graphs is decidable.*

By reducing to decidability in the logic $\mathcal{L}$-MP^2, we can show that the satisfiability problem for GNNs on undirected graphs – that is, the standard notion of GNN – is decidable.

▶ **Theorem 35.** *The satisfiability problem of $\mathcal{BLC}$-GNNs over undirected graphs is decidable.*

▶ **Theorem 36.** *The universal satisfiability problem of $\mathcal{BLC}$-GNNs over undirected graphs is decidable.*

We can also revise our undecidability results for global GNNs to the undirected case, thus giving undecidability for the usual notion of GNN with global readout. This is done with the same reduction from solvability of simple equation systems to the finite satisfiability of MP^2, which we can show works over undirected graphs.

▶ **Theorem 37.** *The finite satisfiability problem of* MP^2 *over undirected graphs is undecidable.*

▶ **Theorem 38.** *The satisfiability problem of* $\mathcal{BG}$TrReLU-GNN*s over undirected graphs is undecidable.*

▶ **Theorem 39.** *The universal satisfiability problem of* $\mathcal{BG}$TrReLU-GNN*s over undirected graphs is undecidable.*

4 GNNs with unbounded activation functions

In this section, we consider GNNs with unbounded activations, such as the standard ReLU. Since we have already shown that global aggregation leads to undecidability even in the bounded case, in this section *we will only deal with GNNs having only local aggregation*. In Section 4.1 we show that the universal satisfiability problem of $\mathcal{BL}$ReLU-GNN is undecidable, a contrast to the case with eventually constant activation functions. In the process, we introduce a logic that also helps with understanding expressiveness of this class of GNNs.

In Section 4.2, we turn to the satisfiability problem, and give a partial positive result about decidability. Here we will not use the logic directly, but rather use components from decidability proofs for Presburger logics [3]. We will use the idea of representing the possible values of activations which was also used in the case of decidability for eventually constant activations. But in this case we will be representing an infinite set of values, using Presburger formulas.

4.1 (Un)decidability of GNNs with unbounded activation functions

We prove the undecidability of the universal satisfiability problem of $\mathcal{BL}$ReLU-GNN. Here we will use logic again. We will not obtain an expressiveness characterization, but merely a logic that embeds in $\mathcal{BL}$ReLU-GNNs: local two-variable modal logic with two-hop Presburger quantifiers ($\mathcal{L}$-M2P^2), which is the extension of MP^2 where the guards are conjunctions of at most two binary predicates.

▶ **Definition 40.** *The syntax of* local two-variable modal logic with two-hop Presburger quantifiers ($\mathcal{L}$-M2P^2) *over vocabulary* τ *is defined inductively:*

- $\top$ *is a* $\mathcal{L}$-M2P^2 *formula.*
- *for a unary predicate* $U \in \tau$, $U(x)$ *is a* $\mathcal{L}$-M2P^2 *formula.*
- *if* $\varphi(x)$ *is a* $\mathcal{L}$-M2P^2 *formula, then so is* $\neg\varphi(x)$.
- *if* $\varphi_1(x)$ *and* $\varphi_2(x)$ *are* $\mathcal{L}$-M2P^2 *formulas, then so is* $\varphi_1(x) \wedge \varphi_2(x)$.
- *if* $\{\varphi_i(x)\}_{1\leq i\leq k} \cup \{\varphi'_i(x)\}_{1\leq i\leq k'}$ *is a set of* $\mathcal{L}$-M2P^2 *formulas,* $\{\epsilon_i(x,z,y)\}_{1\leq i\leq k}$ *is a set of guard formulas, each of form* $E(x,z) \wedge E(z,y)$, $E(x,z) \wedge E(y,z)$, $E(z,x) \wedge E(z,y)$, *or* $E(z,x) \wedge E(y,z)$, *and* $\{\epsilon'_i(x,y)\}_{1\leq i\leq k'}$ *is another set of guard formulas, each of form* $E(x,y)$ *or* $E(y,x)$, *then*
  $\left(\sum_{i=1}^{k} \lambda_i \cdot \#_{z,y}[\epsilon_i(x,z,y) \wedge \varphi_i(y)] + \sum_{i=1}^{k'} \lambda'_i \cdot \#_y[\epsilon'_i(x,y) \wedge \varphi'_i(y)] \circledast \delta\right)$
  is also a $\mathcal{L}$-M2P^2 *formula. The numbers* $\delta, \lambda_i, \lambda'_i$, *and the comparison* $\circledast$ *are as in the standard Presburger quantifier definition.*

The idea is that we can still count a linear combination of cardinalities of the number of nodes satisfying a given lower-level formula that are one-hop away from the current node – as in $\mathcal{L}$-MP^2. Optionally, we can add on a linear combination of the number of two-hop paths that lead to a node satisfying other lower-level formulas.

The semantics of the formulas is given inductively, with the only step that is different from the usual cases being for the quantification, which is the obvious one. We call these *two-hop Presburger quantifiers*. We can apply a similar proof technique as in Theorem 26 to show that the $\mathcal{L}$-M2P^2 are expressible using $\mathcal{BL}$ReLU-GNNs.

▶ **Theorem 41.** *For every n-$\mathcal{L}$-M2P^2 formula $\Psi(x)$, there exists an n-$\mathcal{BL}$ReLU-GNN $\mathcal{A}_\Psi$, such that $\Psi(x)$ and $\mathcal{A}_\Psi$ are equivalent.*

Note that we do not claim an expressive equivalence here. Nevertheless this containment of the logic in the GNN class is useful, since we can show undecidability of the logic by reduction from the halting problem of two-counter machines, which is known to be undecidable [13].

▶ **Definition 42.** *A* two-counter machine *$\mathcal{M}$ is a finite list $d_1 \dots d_n$ of instructions having one of the forms* `INC` (c_i), `IF` $(c_i = 0)$ `GOTO` (j), *or* `HALT`, *where $i \in \{0, 1\}$ and $1 \leq j \leq n$.*

A configuration *is a tuple $\langle q, c_0, c_1\rangle$, where $1 \leq q \leq n$ and $c_0, c_1 \in \mathbb{N}$. We say $\langle q', c_0', c_1'\rangle$ is the* successor configuration *of $\langle q, c_0, c_1\rangle$ if, letting d_q be the q^{th} instruction of the machine:*

- *If d_q is* `INC` (c_i), *then $q' = q + 1$, $c_i' = c_i + 1$, and $c_{1-i}' = c_{1-i}$.*
- *If d_q is* `IF` $(c_i = 0)$ `GOTO` (j), *if $c_i = 0$, then $q' = j$, $c_0' = c_0$, and $c_1' = c_1$; otherwise, $q' = q + 1$, $c_i' = c_i - 1$, and $c_{1-i}' = c_{1-i}$.*

Note that if d_q is `HALT`, *there is no successor. This configuration is called a halting configuration.*

The computation *of the machine is a (possibly infinite) sequence of configurations where the first is $\langle 1, 0, 0\rangle$, consecutive pairs are in the succcessor relationship above, the last configuration is a halting configuration. The machine* halts *if its computation is a finite sequence.*

The reduction is by encoding the computation of a two-counter machine into the graph directly. We have illustrated it in Figure 1. Each configuration is encoded as a height 1 tree, which is denoted by a dashed box. Its line number is represented by the unary predicate Q_i realized by the root vertex, and the values of the counters are represented by the number of "labeled leaves" – those with predicate C_0 or C_1 being true. There are edges connected to the roots of each configuration, which encode the computation sequence. Then it is possible to assert the (in)equality between the number of leaves of some root and the root of the successor tree, which encodes the condition of a valid transition.

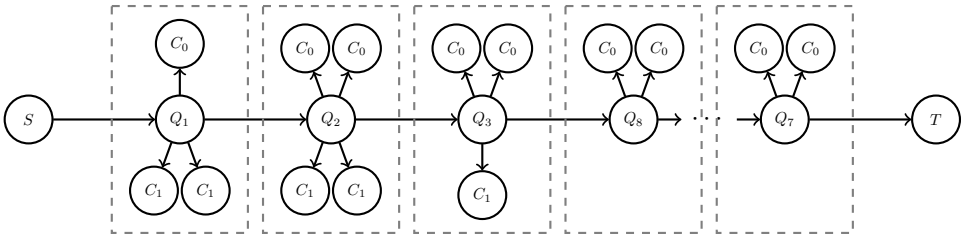


Figure 1 An example of the encoding of the computation of the two-counter machines to directed graphs.

▶ **Lemma 43.** *For every two-counter machine $\mathcal{M}$ with n instructions, there exists an $(n + 5)$-$\mathcal{L}$-M2P^2 formula $\Psi_\mathcal{M}(x)$ such that $\mathcal{M}$ halts if and only if $\forall x\ \Psi_\mathcal{M}(x)$ is finitely satisfiable.*

Since the halting problem of two-counter machines is undecidable, and $\mathcal{L}$-M2P^2 formulas can be translted to $\mathcal{BL}$ReLU-GNNs, we obtain the undecidability of the universal satisfibility problem of $\mathcal{BL}$ReLU-GNN, by reduction from $\mathcal{L}$-M2P^2.

▶ **Theorem 44.** *The universal satisfiability problem of* $\mathcal{BL}$ReLU-GNN*s is undecidable.*

We will later show that this holds also for undirected graphs: see Theorem 52 below.

We can also use the logic to get an expressiveness separation for GNNs: By Theorem 41 to show that $\mathcal{BL}$ReLU-GNNs can do more than $\mathcal{BLC}$-GNNs, it is sufficient to show that there is a $\mathcal{L}$-M2P^2 formula that is not given by a $\mathcal{BLC}$-GNN:

▶ **Lemma 45.** $\mathcal{L}$-M2P^2 *is strictly more expressive than* $\mathcal{BLC}$-GNN*.*

The following results are direct consequences of the lemma above and the logical characterization in the prior section:

▶ **Corollary 46.** $\mathcal{L}$-M2P^2 *is strictly more expressive than* $\mathcal{L}$-MP^2*.*

▶ **Corollary 47.** $\mathcal{BL}$ReLU-GNN *is strictly more expressive than* $\mathcal{BLC}$-GNN*.*

We comment on the proof of Lemma 45. We claim that the property "the number of two-hop paths from the vertex v to the green vertices is the same as the number of two-hop paths from v to the blue vertices" gives the separation. It is easy to express in the two-hop logic. To show that no $\mathcal{BLC}$-GNN can express it, we construct a sequence of pairs of graphs, each with a special node, such that the property holds in the special node of the first graph and fails in the special node of the second, while for every $\mathcal{BLC}$-GNN $\mathcal{A}$, for any sufficiently large pairs of graphs in this sequence, the special nodes are indistinguishable by $\mathcal{A}$. The graphs are as follows:

▶ **Definition 48.** *For* $n_1, n_2 \in \mathbb{N}$*, the* (n_1, n_2)-bipolar graph $\langle V, E, U_1, U_2 \rangle$ *is an undirected* 2*-graph defined as follows.*

$$
\begin{aligned}
U_1 &:= \{v_{1,i} |\ 1 \leq i \leq n_1\} \\
U_2 &:= \{v_{2,i} |\ 1 \leq i \leq n_2\} \\
V &:= U_1 \cup U_2 \cup \{v_0, v_1, v_2\} \\
\widetilde{E} &:= \{(v_0, v_1), (v_0, v_2)\} \cup \{(v_1, v_{1,i}) |\ 1 \leq i \leq n_1\} \cup \{(v_2, v_{2,i}) |\ 1 \leq i \leq n_2\} \\
E &:= \widetilde{E} \cup \left\{ (u, v) \,\middle|\, (v, u) \in \widetilde{E} \right\}
\end{aligned}
$$

See Figure 2.

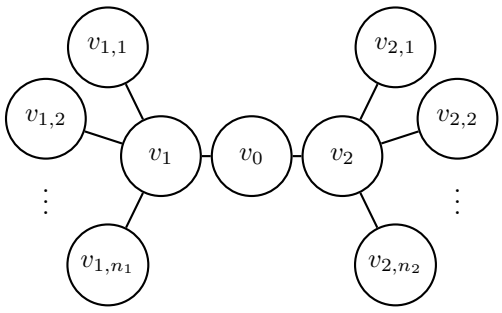


Figure 2 (n_1, n_2)-bipolar graph.

It is easy to see that these graphs with the distinguished nodes v_0 and v_0' disagree on the property, and we can also show that eventually no $\mathcal{BLC}$-GNN can distinguish them:

▶ **Lemma 49.** *For each 2-$\mathcal{BLC}$-GNN $\mathcal{A}$, there exist a threshold $n_{\mathcal{A}} \in \mathbb{N}$, such that for every $n_1, n_2 \geq n_{\mathcal{A}}$, the following properties hold. Let $\mathcal{G}$ be the $(n_{\mathcal{A}}, n_{\mathcal{A}})$-bipolar graph and $\mathcal{G}'$ be the (n_1, n_2)-bipolar graph. For every $0 \leq \ell \leq L$,*

- $\xi^{\ell}_{\mathcal{G}}(v_0) = \xi^{\ell}_{\mathcal{G}'}(v'_0)$, $\xi^{\ell}_{\mathcal{G}}(v_1) = \xi^{\ell}_{\mathcal{G}'}(v'_1)$, *and* $\xi^{\ell}_{\mathcal{G}}(v_2) = \xi^{\ell}_{\mathcal{G}'}(v'_2)$.
- *for* $1 \leq i \leq n_{\mathcal{A}}$ *and* $1 \leq j \leq n_1$, $\xi^{\ell}_{\mathcal{G}}(v_{1,1}) = \xi^{\ell}_{\mathcal{G}}(v_{1,i}) = \xi^{\ell}_{\mathcal{G}'}(v'_{1,j})$.
- *for* $1 \leq i \leq n_{\mathcal{A}}$ *and* $1 \leq j \leq n_2$, $\xi^{\ell}_{\mathcal{G}}(v_{2,1}) = \xi^{\ell}_{\mathcal{G}}(v_{2,i}) = \xi^{\ell}_{\mathcal{G}'}(v'_{2,j})$.

Above $\xi^{\ell}_{\mathcal{G}}$ refers to the ℓ^{th} derived feature function of the GNN $\mathcal{A}$ over the graph $\mathcal{G}$.

Thus far the results in this section are stated for directed graphs. We explain briefly why the undecidability and expressiveness separation results on GNNs with unbounded activation functions apply also to undirected graphs. For the expressiveness results, note that the graphs that we constructed in the proof of Lemma 45 are undirected. Hence the expressiveness gap between $\mathcal{BL}$ReLU-GNN and $\mathcal{BLC}$-GNN still exists for the undirected case.

▶ **Theorem 50.** *$\mathcal{BL}$ReLU-GNN is strictly more expressive over undirected graphs than $\mathcal{BLC}$-GNN.*

To obtain the undecidability of the universal satisfiability problem over undirected graphs of $\mathcal{BL}$ReLU-GNN, we again reduce from two-counter machines, but now with a modification to guarantee the direction of the transition.

▶ **Lemma 51.** *For every two-counter machine $\mathcal{M}$ with n instructions, there exists an $(n+8)$-$\mathcal{L}$-M2P^2 formula $\Psi_{\mathcal{M}}(x)$ such that $\mathcal{M}$ halts if and only if $\forall x\ \Psi_{\mathcal{M}}(x)$ is finitely satisfiable over undirected graphs.*

We cannot apply the exact encoding from Figure 1 and Lemma 43 here, because in that encoding we distinguished the predecessor and successor configurations by the direction of edges. Here, we sketch the trick that overcomes the lack of direction in the edges. We will utilize the predicates from the proof of Lemma 43: in particular we will have a predicate Q and an associated notion of Q vertex as in that proof.

We introduce three fresh unary predicates I_0, I_1, and I_2 to label the configuration's index modulo 3. We add an extra clause to the formula to guarantee that each element has exactly one of these three index labels. The elements in each 1-level tree will have the same index, in the sense of satisfying the same index predicates. Finally, for each Q vertex v with index i, there exists at most one Q vertex v' with index $(i+1 \bmod 3)$, such that v and v' are connected; there exists at most one Q vertex v'' with index $(i-1 \bmod 3)$, such that v and v' are connected. Therefore we can modify the formula which identifies the successor and predecessor based on the index, rather than the direction of the edges, and show that the two-counter machine halts if and only if the modified formula is finitely satisfiable over undirected graphs.

▶ **Theorem 52.** *The universal satisfiability problem of $\mathcal{BL}$ReLU-GNNs over undirected graphs is undecidable.*

4.2 Decidability of satisfiability for "modal" GNNs with unbounded activation functions

Thus the situation for universal satisfiability contrasts with the eventually constant case. What about the *satisfiability problem*? We do not know whether it is decidable for GNNs with piecewise linear activations, or even with just ReLU. We can see that even simple unbounded activation functions produce unbounded spectra, so the proof technique in the truncated case certainly will not work. For example, consider the following 1-layer 1-$\mathcal{BL}$ReLU-GNN.

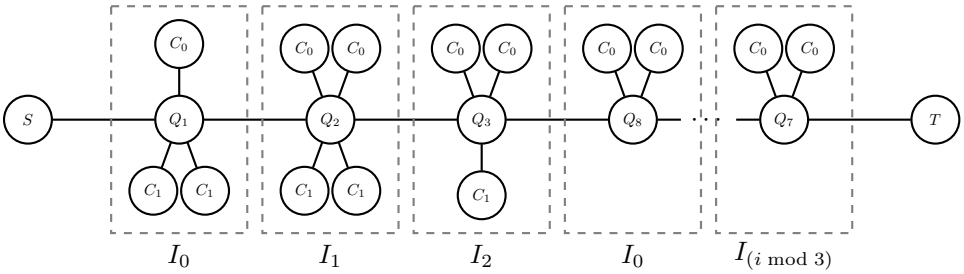


Figure 3 An example of the encoding of the computation of a two-counter machines in undirected graphs.

The dimensions are $d_0 = d_1 = 1$; the coefficient matrices C^1 and A^1_{in} are zero matrices; $\left(A^1_{\text{out}}\right)_{1,1} = 1$; the bias vector b^1 is a zero vector. It is not difficult to see that the value of $\xi^1_1(v)$ is the number of out-neighbors of v. Hence, the 1-spectrum of this GNN is the set of natural numbers.

We present a decidability result for the "modal version": aggregation over nodes connected by outgoing edges only, within a directed graph:

▶ **Theorem 53.** *The satisfiability problem of $\mathcal{OLPW}$-GNNs is decidable.*

Analogously to what we did in the eventually constant case, we describe all the possible values of a given activation function. Unlike in the eventually constant case, this will not be a finite set, but it will be *semi-linear*: that is, describable using a formula of Presburger arithmetic. We will first review the notion of semi-linear set that we use, where we modify the standard notion to deal with rational numbers. We then show that the set of all possible values output by a GNN is a semi-linear set.

For $a_0 \in \mathbb{Q}^k$ and $A = \{a_1, a_2, \ldots, a_m\}$ a finite subset of $\mathbb{Q}^k$, we define:

$$\mathbb{N}\text{-}\mathsf{Span}(a_0, A) := \left\{ a_0 + \sum_{1 \le i \le m} n_i a_i \;\middle|\; n_i \in \mathbb{N} \right\}.$$

A set $S \subseteq \mathbb{Q}^k$ is a *linear set*, if there is $a_0 \in \mathbb{Q}^k$ and a finite set $A \subseteq \mathbb{Q}^k$, such that S is $\mathbb{N}\text{-}\mathsf{Span}(a_0, A)$. The pair (a_0, A) is called *the basis of* S. A *semi-linear set* is a finite union of linear sets. A basis of a semi-linear set $\bigcup_{1 \le i \le k} \mathbb{N}\text{-}\mathsf{Span}\left(a^k_0, A^k\right)$ is the set $\left\{\left(a^1_0, A^1\right), \left(a^2_0, A^2\right), \ldots, \left(a^k_0, A^k\right)\right\}$.

For semi-linear sets $S_1, S_2, S \subseteq \mathbb{Q}^k$, we use the following operators:

$$T(S) := \{T(a) |\ a \in S\} \qquad \text{where } T : \mathbb{Q}^k \to \mathbb{Q}^m \text{ is an affine transformation}$$

$$\mathsf{KleeneStar}(S) := \left\{ \sum_{s \in S'} s \;\middle|\; \text{For every finite multi-subset } S' \text{ of } S \right\}$$

We recall that in the context of integers, both operators are known to preserve semi-linearity and the basis of the resulting semi-linear set can be computed. See, e.g., [5, 10, 6]. The arguments adapt easily to our rational setting, thus *we assume below that we have an algorithm for pushing semi-linear representations through these operators.*

We consider piecewise linear functions, defined by a sequence $((I_1, f_1), \ldots, (I_p, f_p))$ where $I_1 \cup \cdots \cup I_p$ is a partition of $\mathbb{Q}$ into p intervals and each $f_i : \mathbb{Q} \to \mathbb{Q}$ is an affine function. The sequence $((I_1, f_1), \ldots, (I_p, f_p))$ defines a function where x is mapped to $f_i(x)$ if x is in

the interval I_i. We apply a piecewise linear function on some fixed components of a vector, which is captured by the following notation. For a piecewise linear function $f : \mathbb{Q} \to \mathbb{Q}$, a rational vector $a \in \mathbb{Q}^k$ and $K \subseteq [1, k]$, we write $f_K(a)$ to denote the vector $b \in \mathbb{Q}^k$ where $b_i = f(a_i)$ for every $i \in K$ and $b_i = a_i$ for every $i \notin K$. In other words, $f_K(a)$ only applies the function f on the components in K and the identity function on the components outside K. Similar to affine transformation and Kleene star, *piecewise linear functions also preserve semi-linearity and the basis of the resulting semi-linear sets can be computed.* Again, we can easily adapt the argument in [6] to our rational setting.

To prove Theorem 53, we need two more definitions. Let $\mathcal{A}$ be a L-layer $\mathcal{OLPW}$-GNN. Let $d_0, d_1, \ldots, d_L$ be the dimension of the layers. We denote by $\mathbb{Q}^{[d_0,d_1,\ldots,d_{\ell-1}]}$ the Cartesian product $\mathbb{Q}^{d_0} \times \mathbb{Q}^{d_1} \ldots \times \mathbb{Q}^{d_{\ell-1}}$. Given an element m of this product, the i^{th} component of m, denoted by $m[i]$, is the projection of m to $\mathbb{Q}^{d_i}$.

For a graph $\mathcal{G}$, vertex $v \in V$, and $0 \leq \ell \leq L$, the *ℓ-history of v in $\mathcal{G}$* (w.r.t. $\mathcal{A}$), denoted by $\mathsf{hist}^{\ell}_{\mathcal{G}}(v) \in \mathbb{Q}^{[d_0,d_1,\ldots,d_\ell]}$, is the tuple that collects the first $(\ell+1)$ feature vectors of v. Formally, for $0 \leq i \leq \ell$, $\left(\mathsf{hist}^{\ell}_{\mathcal{G}}(v)\right)[i] = \xi^i_{\mathcal{G}}(v)$. When the graph $\mathcal{G}$ is clear from the context, we omit it and simply write $\mathsf{hist}^{\ell}(v)$. The *ℓ-history-space* of $\mathcal{A}$ is the set of all possible histories.

We now state our representation theorem, which immediately implies Theorem 53:

▶ **Theorem 54.** *For every $\mathcal{OLPW}$-GNN $\mathcal{A}$ and $0 \leq \ell \leq L$, the ℓ-history-space is semi-linear, and its basis can be effectively computed.*

We contrast the theorem with Theorem 17. There we could only overapproximate the spectrum, because we could not determine which numbers from previously layers were simultaneously realizable. By inductively maintaining the entire history at each node, we have enough information to resolve these questions of consistency, and compute an *exact* representation of the semantic object, not just an overapproximation.

The rest of this section is devoted to the proof of Theorem 54. We will first explain the intuition behind it. Let $\mathcal{A}$ be a L-layer $\mathcal{OLPW}$-GNN, as in Definition 2. Let $\mathcal{G}$ be a graph and v be a vertex. Recall that for every $1 \leq \ell \leq L$, the ℓ-feature vector of v is:

$$\xi^{\ell}(v) \;:=\; f^{\ell}\left(C^{\ell}\xi^{\ell-1}(v) + A^{\ell}_{\text{out}} \sum_{u \in \mathcal{N}_{\text{out}}(v)} \xi^{\ell-1}(u) + b^{\ell}\right).$$

We can rewrite it in terms of history:

$$\mathsf{hist}^{\ell}(v)[0] \;= \xi^0(v), \tag{1}$$

and for each $1 \leq i \leq \ell$:

$$\mathsf{hist}^{\ell}(v)[i] \;= f^i\left(C^i \cdot \mathsf{hist}^{\ell}(v)[i-1] \;+\; A^i_{\text{out}} \cdot \left(\sum_{u \in \mathcal{N}_{\text{out}}(v)} \mathsf{hist}^{\ell-1}(u)\right)[i-1] \;+\; b^i\right). \tag{2}$$

Thus, the ℓ-history of v can be computed by applications of sum, affine transformations, and piecewise linear functions on the sum of the history of its out-neighbors.

We formalise this intuition in the following paragraphs. For each $0 \leq \ell \leq L$, we define the set $\mathcal{H}^{\ell}$:

$$\mathcal{H}^0 \;:=\; \{0,1\}^{d_0}$$
$$\mathcal{H}^{\ell} \;:=\; \bigcup_{e \in \{0,1\}^{d_0}} \mathsf{proj}_{\ell} \circ T_{\ell} \circ T_{\ell-1} \circ \ldots \circ T_{0,e} \circ \mathsf{KleeneStar}\left(\mathcal{H}^{\ell-1}\right),$$

where the definition and intuition of each $\mathsf{proj}_{\ell}, T_{\ell}, \ldots, T_1, T_{0,e}$ is as follows.

- Intuitively $\mathsf{KleeneStar}\left(\mathcal{H}^{\ell-1}\right)$ captures the term $\sum_{u\in\mathcal{N}_{\text{out}}(v)} \mathsf{hist}^{\ell-1}(u)$ in Equation (2).
- $T_{0,e} : \mathbb{Q}^{[d_0,\ldots,d_{\ell-1}]} \to \mathbb{Q}^{[d_0,\ldots,d_{\ell-1},d_0]}$ is an affine transformation that maps a to (a, e), i.e., it simply "pads" e into a.
- For each $1 \leq i \leq \ell$, the transformation $T_i : \mathbb{Q}^{[d_0,\ldots,d_{\ell-1},d_0,\ldots,d_{i-1}]} \to \mathbb{Q}^{[d_0,\ldots,d_{\ell-1},d_0,\ldots,d_{i-1},d_i]}$ computes the vector $\mathsf{hist}^{\ell}(v)[i]$ defined in Equation (2) and pads it at the end. Formally, T_i maps a to (a, c) where $c = f^i\left(C^i a[\ell+i-1] + A^i_{\text{out}} a[i-1] + b^i\right)$.
- Finally, $\mathsf{proj}_\ell : \mathbb{Q}^{[d_0,\ldots,d_{\ell-1},d_0,\ldots,d_\ell]} \to \mathbb{Q}^{[d_0,\ldots,d_\ell]}$ is a projection that projects out the first ℓ components.

We can show that $\mathcal{H}^\ell$ is a semi-linear set, and this captures the ℓ-history-space, as stated formally in Lemma 55. Note that Theorem 53 follows easily from the lemma and the computability of the basis of $\mathcal{H}^\ell$.

▶ **Lemma 55.** *For every $\mathcal{OLPW}$-GNN $\mathcal{A}$ and $0 \leq \ell \leq L$,*

1. *$\mathcal{H}^\ell$ is a semi-linear set.*
2. *For every $s \in \mathbb{Q}^{[d_0,d_1,\ldots,d_\ell]}$, the following are equivalent.*
   - *$h \in \mathcal{H}^\ell$*
   - *There exists a graph $\mathcal{G}$ and vertex $v \in V$ such that $\mathsf{hist}^{\ell}(v) = h$.*

Proof. The first item follows immediately from the fact that $\mathcal{H}^0$ is semi-linear and the operators Kleene star, affine transformations and piecewise linear functions all preserve semi-linearity.

We now prove the second item by induction on ℓ. The base case $\ell = 0$ is trivial.

For the induction hypothesis, we assume that the lemma holds for $\ell - 1$. We show that $h \in \mathcal{H}^\ell$ if and only if there is a graph $\mathcal{G}$ and a vertex v such that $\mathsf{hist}^{\ell}(v) = h$.

We start with the "only if" direction. Suppose $h \in \mathcal{H}^\ell$. By definition, there is $e \in \{0,1\}^{d_0}$ and a finite multi-subset $\{\{h_1, h_2, \ldots, h_k\}\}$ of $\mathcal{H}^{\ell-1}$ such that:

$$h = \mathsf{proj}_\ell \circ T_\ell \circ T_{\ell-1} \circ \ldots \circ T_{0,e}\left(h_1 + h_2 + \cdots + h_k\right)$$

By the induction hypothesis, there exist graphs $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k$ and vertices $v_1, v_2, \ldots, v_k$ such that $\mathsf{hist}^{\ell-1}_{\mathcal{G}_i}(v_i) = h_i$ for every $1 \leq i \leq k$.

Let $\mathcal{G}$ be the graph obtained by taking the disjoint union of $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k$ and adding a fresh vertex v. Recalling that $\xi^0_{\mathcal{G}}(v)$ can achieve an arbitrary combination of $\{0,1\}$ vectors, based on the colors of v, we set the colors so that $\xi^0_{\mathcal{G}}(v) = e$. We have an outgoing edge from v to v_i for each $1 \leq i \leq k$. It is routine to verify that the ℓ-history of v is precisely h. Note that *because $\mathcal{A}$ is outgoing-only, the edge from v to v_i has no effect on the $(\ell-1)$-history of v_i*. Thus $\mathsf{hist}^{\ell-1}_{\mathcal{G}}(v_i) = \mathsf{hist}^{\ell-1}_{\mathcal{G}_i}(v_i)$

For the "if" direction, let $\mathcal{G}$ be a graph and v be a vertex. Let $v_1, \ldots, v_k$ be the out-neighbors of v. By definition, for each $1 \leq i \leq \ell$:

$$\mathsf{hist}^{\ell}_{\mathcal{G}}(v)[i] \;=\; f^i\left(C^i \cdot \mathsf{hist}^{\ell}_{\mathcal{G}}(v)[i-1] \;+\; A^i_{\text{out}} \cdot \left(\sum_{u\in\mathcal{N}_{\text{out}}(v)} \mathsf{hist}^{\ell-1}_{\mathcal{G}}(u)\right)[i-1] \;+\; b^i\right).$$

It is routine to verify that:

$$\mathsf{hist}^{\ell}_{\mathcal{G}}(v) \;=\; \mathsf{proj}_\ell \circ T_\ell \circ T_{\ell-1} \circ \ldots \circ T_{0,e}\left(\mathsf{hist}^{\ell-1}_{\mathcal{G}}(v_1) + \mathsf{hist}^{\ell-1}_{\mathcal{G}}(v_2) + \cdots + \mathsf{hist}^{\ell-1}_{\mathcal{G}}(v_k)\right),$$

where $e = \xi^0_{\mathcal{G}}(v)$. Therefore, $\mathsf{hist}^{\ell}_{\mathcal{G}}(v) \in \mathcal{H}^\ell$. ◀

5 Discussion

This work extends the exploration of the relationship between aggregate-combine GNNs and logic, with exact characterizations of expressiveness for GNNs with eventually constant activation functions, and embedding a logic into the GNNs with standard ReLU activations. We also obtain both decidability and undecidability results, some using the logical characterizations and some by porting the techniques used for decidability of the logics to apply directly on the GNNs. Perhaps the main take-away, echoing the theme of [1], is that Presburger logics and the techniques for analyzing them can be relevant to GNNs.

We have left open one major technical problem: the decidability of satisfiability for standard GNNs using the ReLU activation function. Here we have proven decidability only for the "outgoing-only" variant. We also do not know whether the undecidability results we have proven – e.g. for standard GNNs with global readout – still hold for the variants with outgoing-only aggregation. Thus, for all we know, the most crucial dividing line for decidability could revolve around outgoing-only vs bidirectional aggregation, rather than (e.g.) local vs global aggregation or truncation vs non-truncation in the activation function.

Looking at broader open issues, we focused here on some very basic verification problems on GNNs: can a certain classification be achieved? But it is clear that our techniques apply to many other logic-based verification problems; for example, it can be applied to determine whether a GNN can achieve a certain classification on a graph satisfying a certain sentence – provided that the sentence is also in one of our decidable logics.

We have not focused on complexity in this paper. Of course, for the broad class of GNNs with eventually constant activation functions, it is difficult to talk about complexity bounds. For GNNs based on truncated ReLU and local aggregation, we have shown satisfiability is PSPACE-complete, and is NP-complete for a fixed number of layers. The finer-grained complexity analysis for other decidability results is left for future work.

Our work provides motivation for exploring the properties of Presburger logics over relational structures and their connections with GNNs beyond the setting here, which considers only graphs with discrete feature values from a fixed set. In our ongoing work we are adapting our techniques to deal with GNNs whose feature values are *unbounded integers*, specified by an initial semi-linear set.

References

1 Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan L. Reutter, and Juan Pablo Silva. The Logical Expressiveness of Graph Neural Networks. In *ICLR*, 2020.

2 Pablo Barceló, Alexander Kozachinskiy, Anthony Wijdada Lin, and Vladamir Podolskii. Logical languages accepted by transformer encoders with hard attention. In *ICLR*, 2024.

3 Bartosz Bednarczyk, Maja Orlowska, Anna Pacanowska, and Tony Tan. On classical decidable logics extended with percentage quantifiers and arithmetics. In *FSTTCS*, 2021.

4 David Chiang, Peter Cholak, and Anand Pillay. Tighter bounds on the expressivity of transformer encoders. In *ICML*, 2023.

5 Dmitry Chistikov and Christoph Haase. The taming of the semi-linear set. In *ICALP*, 2016.

6 Seymour Ginsburg and Edwin H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific J. Math.*, 16(2):285–296, 1966.

7 Valentin Goranko and Martin Otto. Model theory of modal logic. In Blackburn, van Benthem, and Wolter, editors, *Handbook of Modal Logic*. North-Holland, 2007.

8 Martin Grohe. The logic of graph neural networks. In *LICS*, 2021.

9 Martin Grohe. The descriptive complexity of graph neural networks. In *LICS*, 2023.

10 C. Haase and Georg Zetzsche. Presburger arithmetic with stars, rational subsets of graph groups, and nested zero tests. *LICS*, 2019.

11 Viktor Kuncak, Huu Hai Nguyen, and Martin Rinard. An Algorithm for Deciding BAPA: Boolean Algebra with Presburger Arithmetic. In *CADE*, 2005.

12 Richard E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comput.*, 6(3):467–480, 1977.

13 Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.

14 Marco Sälzer and Martin Lange. Fundamental limits in formal verification of message-passing neural networks. In *ICLR*, 2023.

15 Manfred Schmidt-Schaubß and Gert Smolka. Attributive concept descriptions with complements. *Artif. Intell.*, 48(1):1–26, 1991.

16 Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks. In *ICLR*, 2019.

Automata-Theoretic Characterisations of Branching-Time Temporal Logics

Massimo Benerecetti
Università di Napoli Federico II, Italy

Laura Bozzelli
Università di Napoli Federico II, Italy

Fabio Mogavero
Università di Napoli Federico II, Italy

Adriano Peron
Università di Trieste, Italy

Abstract

Characterisations theorems serve as important tools in model theory and can be used to assess and compare the expressive power of temporal languages used for the specification and verification of properties in formal methods. While complete connections have been established for the linear-time case between temporal logics, predicate logics, algebraic models, and automata, the situation in the branching-time case remains considerably more fragmented. In this work, we provide an *automata-theoretic characterisation* of some important branching-time temporal logics, namely CTL* and ECTL* interpreted on arbitrary-branching trees, by identifying two variants of *Hesitant Tree Automata* that are proved equivalent to those logics. The characterisations also apply to *Monadic Path Logic* and the bisimulation-invariant fragment of *Monadic Chain Logic*, again interpreted over trees. These results widen the characterisation landscape of the branching-time case and solve a forty-year-old open question.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Modal and temporal logics; Theory of computation → Tree languages

Keywords and phrases Branching-Time Temporal Logics, Monadic Second-Order Logics, Tree Automata

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.128

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/2404.17421

Funding M. Benerecetti, F. Mogavero, and A. Peron are members of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM). This work has been partially supported by the GNCS 2024 project "Certificazione, monitoraggio, ed interpretabilità in sistemi di intelligenza artificiale".

1 Introduction

Temporal logics [49] play a pivotal role in the *formal verification* of complex systems [50]. Serving as *specification languages*, they provide a framework to express and reason about time-dependent properties, capturing the intricate behaviours and interactions of system components over time. Commonly, these languages are classified into two categories: *linear-time logics*, which emphasise properties spanning the entirety of a computation, and *branching-time logics*, specifically tailored to address the non-deterministic and concurrent nature of behaviours. Well-established representatives of the former include *Linear-Time Temporal Logic* (LTL) [61, 62], its full ω-regular extension ELTL [84], and the finite-horizon variant

$\mathrm{LTL_f}$ [32]. Important members of the second category, instead, belong to the families of *Dynamic Logics* [30, 35] and *Computation Tree Logics*, including CTL [21, 16, 24, 17, 22], CTL* [23, 25], ECTL* [80], $\mathrm{CTL^*_f}$ [79], and $\mathrm{ECTL^*_f}$ [74]. Additionally, more expressive but lower-level languages, like μ-Calculus [42], have been considered, which suitably extend classic modal logic with monadic fix-point operators, contributing to the rich tapestry of specification languages in the field of formal verification and synthesis.

The semantics of these temporal logics are typically formalised, at the meta-level, through various flavour of *predicate logic*, frequently *First-Order Logic* (FO) or *Second-Order Logic* (SO), interpreted over either *linearly-ordered structures*, such as finite and infinite words [60], or *branching structures*, like Kripke structures [43], labelled transition systems [40], and their tree unwindings. In tandem with this, the rich body of literature on automata-theoretic techniques [75] for words and trees, originated from [41, 56, 57, 66], has proven invaluable to provide effective technical tools for the solution of related *model-checking* [18, 2, 81, 47, 27], *satisfiability* [81, 26, 78, 6, 47], and *synthesis* [15, 63, 69] decision problems. Predicate logics and automata theory offer, in addition, a rich and coherent arsenal of tools to evaluate and compare the expressive power, as well as the computational properties, of temporal languages, as witnessed by numerous *characterisation theorems*. These results provide a dual perspective on the topic, which enhance our ability to navigate the intricate landscape of language fragments and allow us to assess their pros (*e.g.*, elementary complexity of decision problems) and cons (*e.g.*, limitations on the expressive power).

The initial seminal result in this context is Kamp's theorem [39, 31, 67, 68], which establishes the equivalence of LTL and FO over infinite words. The result also extends to $\mathrm{LTL_f}$ and FO on finite words [19]. A direct link has been drawn between FO-*definability* and *recognition* by *counter-free finite-state automata*, in both the finite [52] and infinite [48, 71, 72, 59] cases, by means of the notions of *star-free language*, *aperiodic language*, and *aperiodic syntactic monoid* (see [70], for finite words, and [58], for the infinite ones). Together these results provide a complete characterisation of the expressive power of LTL and $\mathrm{LTL_f}$ in terms of predicate logics and automata. A parallel correspondence exists between ELTL and $\mathrm{ELTL_f}$, the *Monadic Second-Order Logic* (MSO) and its *weak (finite-quantification) fragment* (WMSO), and regular automata on infinite and finite words. Notably, the equivalence between WMSO and regular automata [8, 20, 76], followed by the equivalence between MSO and ω-regular automata [9, 10, 51, 14], stands among the first results connecting the two fields of model theory and automata theory.

The landscape for branching-time temporal logics is considerably more intricate, due to the complex topology of the models and additional factors, such as *bisimulation invariance* [77] and *counting quantifiers* [29], and it is not as clear and complete as the linear-time counterpart. A significant milestone in this setting is the full correspondence between μ-Calculus, the *bisimulation-invariant* fragment of MSO interpreted over trees, and *(Symmetric) Alternating Parity Tree Automata* [38]. This result generalises the already known connection between the latter two formalisms [64]. Another noteworthy connection has been shown to exist between the *alternation-free* fragment of μ-Calculus (AFμ-Calculus), the bisimulation-invariant fragments of WMSO over bounded-branching trees, and *(Symmetric) Alternating Weak Tree Automata* [1, 37] (see [28, 11, 12, 13], for the unbounded-branching case), which extends previous partial results [45, 65]. The above equivalences lift also to the general case, by incorporating counting quantifiers into the temporal logics [37, 36]. The scenario in other cases appears significantly more fragmented. In recent developments, the equivalence between CTL and *(Symmetric) Hesitant Linear Tree Automata* [7] was proved. Nonetheless, as of today, no corresponding fragment of MSO has been identified. By contrast, several variants

of CTL* have been linked to the *path* and *chain* fragments of MSO since the eighties, although no automata characterisation has been provided thus far. For instance, it was shown in [34] that, on binary trees, CTL* is equivalent to *Monadic Path Logic* (MPL) [33]. Similar correspondences have been established in [74] for CTL^*_{f}, ECTL*, and ECTL^*_{f}, which equate, respectively, to FO, *Monadic Chain Logic* (MCL), and its weak fragment (WMCL). The result concerning CTL* was later extended to arbitrary-branching trees, addressing both bisimulation-invariance [54] and counting quantifiers [55]. As far as we know, no similar results are available for the other three logics. Finally, the recently introduced *Monadic Tree Logic* (MTL) [3] together with its variants have yet to find a correspondence either with temporal logics or with automata.

The objective of this work is to provide an *automata-theoretic characterisation* of CTL* and ECTL*, by identifying two specific classes of alternating tree automata that are expressively equivalent to those logics (the used technique extends seamlessly to the finite-horizon variants). A first result is the proof of the equivalence of the *symmetric variant* of classic ranked *Hesitant Tree Automata* (HTA) [47] with both ECTL* and the bisimulation-invariant fragment of MCL. To this end, for technical convenience, we employ two intermediate formalisms. On the one hand, to prove the equivalence between HTA and ECTL*, we use a *syntactic variant* of ECTL*, called *Computation Dynamic Logic* (CDL), alongside its counting version (CCDL). In ECTL* temporal operators are specified by means of right-linear grammars, while CDL uses finite automata on finite words for the same purpose incorporated into the dynamic modalities. Moreover, while the path subformulae in ECTL* are part of the alphabet of the grammar, in CDL they are specified by means of a testing function over the set of states of the automaton. It is straightforward to move from one formalism to the other by means of a linear-time translation. This logic essentially lifts to the branching-time realm the *Linear Dynamic Logic* (LDL) proposed in [32, 83]. On the other hand, we consider a *first-order extension* [82] of HTAs (HFTA) and show them equivalent to MCL by proving a closure property under *chain projections*. The final result, then, follows from the equivalence between HTAs and the bisimulation-invariant fragment of HFTA. As a second result, we first identify the *graded extension* of HTAs (HGTA), together with its counter-free restriction (HGTA_{cf}), and then prove their equivalence with CCDL and CCTL*, respectively. While for the definition of HGTA the standard notion of counting modalities smoothly applies, introducing HGTA_{cf} proves quite more intricate. We show, indeed, that a naive application of counter-freeness in the context of tree-automata leads to a class of languages that are not CTL* definable. To overcome this problem, we identify the crucial *mutual-exclusion* property of a HGTA that constrains the automaton branching-behaviours. This property, together with counter-freeness of the automaton linear behaviours, provides an apt definition of HGTA_{cf}, something that was previously only hypothesised in [54, 55]. The above characterisations holds also under bisimulation-invariance assumptions. Specifically, HTA_{cf} is equivalent to both CTL* and the bisimulation-invariant fragment of MPL. All these results, coupled with the algebraic characterisation of tree languages provided in [74], brings the expressiveness landscape for branching-time temporal logics to the same level as their linear-time counterpart, thus closing a forty-year-old open question posed in [73, 74].

2 Preliminaries

Let $\mathbb{N}$ be the set of natural numbers. For $i, j \in \mathbb{N}$ with $i \leq j$, $[i, j]$ denotes the set of natural numbers k such that $i \leq k \leq j$. For a finite or infinite word ρ over some alphabet, $|\rho|$ is the length of ρ ($|\rho| = \omega$ if ρ is infinite) and for all $0 \leq i < |\rho|$, $\rho(i)$ is the $(i+1)$-th letter of ρ.

Kripke Trees and Tree Languages. Given a non-empty set of directions D, a tree T (with set of directions in D) is a non-empty subset of D^* which is prefix closed (i.e., for each $w \cdot d \in T$ with $d \in D$, $w \in T$). Elements of T are called nodes and the empty word ε is the root of T. For $w \in T$, a *child* of w in T is a node in T of the form $w \cdot d$ for some $d \in D$. For $w \in T$, the *subtree of* T *rooted at node* w is the tree consisting of the nodes of the form w' such that $w \cdot w' \in T$. A *subtree of* T is a tree T' such that for some $w \in T$, T' is a subset of the subtree of T rooted at w. A *path* of T is a subtree π of T which is totally ordered by the child-relation (i.e., each node of π has at most one child in π). In the following, a path π of T is also seen as a word over T in accordance to the total ordering in π induced by the child relation. A *chain* of T is a subset of a path of T. A tree is *non-blocking* if each node has some child. A non-blocking tree T is infinite, and maximal paths in T are infinite as well.

For an alphabet Σ, a Σ-labelled tree is a pair (T, Lab) consisting of a tree and a labelling $Lab : T \mapsto \Sigma$ assigning to each node in T a symbol in Σ. A *tree-language* over Σ is a set of Σ-labeled trees. In this paper, we consider formalisms whose specifications denote *tree-languages* over a given alphabet Σ. For the easy of presentation, we assume that the labeled trees of a tree-language are non-blocking. All the results of this paper can be easily adapted to the general case, where the non-blocking assumption is relaxed. For a finite set AP of atomic propositions, a *Kripke tree* over AP is a non-blocking 2^{AP}-labelled tree.

Automata over Infinite and Finite Words. We first recall the class of parity nondeterministic automata on infinite words (parity NWA for short) which are tuples $\mathcal{A} = \langle \Sigma, Q, \delta, q_I, \Omega \rangle$, where Σ is a finite input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \mapsto 2^Q$ is the transition function, $q_I \in Q$ is an initial state, and $\Omega : Q \mapsto \mathbb{N}$ is a parity acceptance condition over Q assigning to each state a natural number (color). Given a word ρ over Σ, a *path* of $\mathcal{A}$ over ρ is a word π over Q of length $|\rho| + 1$ ($|\rho| + 1$ is ω if ρ is infinite) such that $\pi(i+1) \in \delta(\pi(i), \rho(i))$ for all $0 \leq i < |\rho|$. A *run* over ρ is a path over ρ starting at the initial state. The NWA $\mathcal{A}$ is *counter-free* if for all $n > 0$, states $q \in Q$ and finite words ρ over Σ, the following holds: if there is a path from q to q over ρ^n, then there is also a path from q to q over ρ.

A run π of $\mathcal{A}$ over an infinite word ρ is *accepting* if the highest color of the states appearing infinitely often along π is even. The ω-language $\mathsf{L}(\mathcal{A})$ accepted by $\mathcal{A}$ is the set of infinite words ρ over Σ such that there is an accepting run π of $\mathcal{A}$ over ρ.

A parity acceptance condition $\Omega : Q \mapsto \mathbb{N}$ is a *Büchi* (resp., *coBüchi*) condition if there is an even (resp., odd) color $n \in \mathbb{N}$ such that $\Omega(Q) \subseteq \{n-1, n\}$. A *Büchi* (resp., *coBüchi*) NWA is a parity NWA whose acceptance condition is Büchi (resp., coBüchi).

We also consider NWA over finite words (NWA_f for short) which are defined as parity NWA but the parity condition Ω is replaced with a set $F \subseteq Q$ of accepting states. A run π over a finite word is *accepting* if its last state is accepting.

3 Branching-Time Temporal Logics

In this section, we recall syntax and semantics of Counting-CTL* (CCTL* for short) [55], which extends the classic branching-time temporal logic CTL* [25] with counting operators. Moreover, we introduce a novel branching-time temporal logic more expressive than CCTL*, called *Counting Computation Dynamic Logic* (CCDL for short). CCDL can be viewed as a branching-time extension of Linear Dynamic Logic (LDL) [32]. However, unlike LDL, we consider NWA_f over finite words, instead of regular expressions, as the building blocks of formulae. This approach is similar to the one adopted in [83] for Visibly Linear Dynamic Logic, a context-free extension of LTL.

The Logic CCTL*. The syntax of CCTL* is given by specifying inductively the set of *state formulae* φ and the set of *path formulae* ψ over a given finite set AP of atomic propositions:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathtt{E}\psi \mid \mathtt{D}^n\varphi$$
$$\psi ::= \varphi \mid \neg\psi \mid \psi \wedge \psi \mid \mathtt{X}\,\psi \mid \psi\,\mathtt{U}\,\psi$$

where $p \in \mathrm{AP}$, $\mathtt{X}$ and $\mathtt{U}$ are the standard "next" and "until" temporal modalities, $\mathtt{E}$ is the existential path quantifier, and $\mathtt{D}^n$ is the counting operator with $n \in \mathbb{N} \setminus \{0\}$. The language of CCTL* consists of the state formulae of CCTL*. Standard CTL* is the fragment of CCTL* where the counting operators $\mathtt{D}^n$ with $n > 1$ are disallowed, and standard LTL [61] corresponds to the set of path formulae of CTL* where the path quantifiers are disallowed.

Given a Kripke tree $\mathcal{T} = (\mathrm{T}, \mathit{Lab})$ (over AP), a node w of T, an infinite path π of T, and $0 \leq i < |\pi|$, the satisfaction relations $(\mathcal{T}, w) \models \varphi$, for a state formula φ, (meaning that φ holds at node w of $\mathcal{T}$), and $(\mathcal{T}, \pi, i) \models \psi$, for a path formula ψ, (meaning that ψ holds at position i of the path π in $\mathcal{T}$) are defined as usual:

$$\begin{array}{ll}
(\mathcal{T}, w) \models p & \Leftrightarrow p \in \mathit{Lab}(w);\\
(\mathcal{T}, w) \models \mathtt{E}\psi & \Leftrightarrow (\mathcal{T}, \pi, 0) \models \psi \text{ for some infinite path } \pi \text{ of } T \text{ starting at node } w;\\
(\mathcal{T}, w) \models \mathtt{D}^n\varphi & \Leftrightarrow \text{there are at least } n \text{ distinct children } w' \text{ of } w \text{ in T s.t. } (\mathcal{T}, w') \models \varphi;\\
(\mathcal{T}, \pi, i) \models \varphi & \Leftrightarrow (\mathcal{T}, \pi(i)) \models \varphi;\\
(\mathcal{T}, \pi, i) \models \mathtt{X}\,\psi & \Leftrightarrow (\mathcal{T}, \pi, i+1) \models \psi;\\
(\mathcal{T}, \pi, i) \models \psi_1\,\mathtt{U}\,\psi_2 & \Leftrightarrow \text{for some } j \geq i\text{: } (\mathcal{T}, \pi, j) \models \psi_2 \text{ and } (\mathcal{T}, \pi, k) \models \psi_1 \text{ for all } i \leq k < j.
\end{array}$$

Note that $\mathtt{D}^1\varphi$ corresponds to $\mathtt{EX}\,\varphi$. A Kripke tree $\mathcal{T}$ satisfies (or is a model of) a state formula φ, written $\mathcal{T} \models \varphi$, if $\mathcal{T}, \varepsilon \models \varphi$. The tree-language $\mathsf{L}(\varphi)$ of φ is the set of models of φ. For an LTL formula ψ and an infinite word ρ over 2^{AP}, ρ satisfies ψ, written $\rho \models \psi$, if $\mathcal{T}_\rho \models \mathtt{E}\psi$, where $\mathcal{T}_\rho$ is a trivial tree-encoding of ρ. For an LTL formula ψ, $\mathsf{L}(\psi)$ denotes the set of infinite words over 2^{AP} satisfying ψ.

The New Logic CCDL. Like CCTL*, the syntax of CCDL is composed of *state formulae* φ and *path formulae* ψ over a given finite set AP of atomic propositions, defined as follows:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathtt{E}\psi \mid \mathtt{D}^n\varphi$$
$$\psi ::= \varphi \mid \neg\psi \mid \psi \wedge \psi \mid \langle\mathcal{A}\rangle\psi$$

where $p \in \mathrm{AP}$ and $\langle\mathcal{A}\rangle$ is the *existential sequencing* modality applied to a *testing* $\mathrm{NWA_f}$ $\mathcal{A}$. We define a *testing* $\mathrm{NWA_f}$ $\mathcal{A} = \langle 2^{\mathrm{AP}}, \mathrm{Q}, \delta, q_I, \mathrm{F}, \tau\rangle$ as consisting of an $\mathrm{NWA_f}$ $\langle 2^{\mathrm{AP}}, \mathrm{Q}, \delta, q_I, \mathrm{F}\rangle$ over finite words over 2^{AP} and a test function τ mapping states in Q to CCDL path formulae. Intuitively, along an infinite path π of a Kripke tree, the testing automaton accepts the labeling of a (possibly empty) infix $\pi(i)\dots\pi(j-1)$ of π if the embedded $\mathrm{NWA_f}$ has an accepting run $q_i \dots q_j$ over the labeling of such an infix so that, for each position $k \in [i, j]$, the formula $\tau(q_k)$ holds at position k along π. A test function τ is *trivial* if it maps each state to $\top$. We also use the shorthand $[\mathcal{A}]\psi \triangleq \neg\langle\mathcal{A}\rangle\neg\psi$ (*universal sequencing* modality). The language of CCDL consists of the state formulae of CCDL. We also consider the *bisimulation-invariant* fragment CDL of CCDL where the counting operators $\mathtt{D}^n$ with $n > 1$ are disallowed. Given a Kripke tree $\mathcal{T} = (\mathrm{T}, \mathit{Lab})$, an infinite path π of T, and $0 \leq i < |\pi|$, the semantics of modality $\langle\mathcal{A}\rangle$ is defined as follows, where $\mathcal{A} = \langle 2^{\mathrm{AP}}, \mathrm{Q}, \delta, q_I, \mathrm{F}, \tau\rangle$:

$(\mathcal{T}, \pi, i) \models \langle\mathcal{A}\rangle\psi \quad \Leftrightarrow$ for some $j \geq i$, $(i, j) \in \mathrm{R}_{\mathcal{A}}(\mathcal{T}, \pi)$ and $(\mathcal{T}, \pi, j) \models \psi$

where $\mathrm{R}_{\mathcal{A}}(\mathcal{T}, \pi)$ is the set of pairs (i, j) with $j \geq i$ such that there is an accepting run $q_i \dots q_j$ of the $\mathrm{NWA_f}$ embedded in $\mathcal{A}$ over $\mathit{Lab}(\pi(i)) \dots \mathit{Lab}(\pi(j-1))$ and, for all $k \in [i, j]$, it holds that $(\mathcal{T}, \pi, k) \models \tau(q_k)$. The notions of a model and tree-language of a CCDL formula are defined as for CCTL*.

Embedding of CCTL* into CCDL. The logic CCTL* can be easily embedded into CCDL as follows. Let $\mathcal{A}$ be the testing $\mathrm{NWA_f}$ having trivial tests and accepting all and only the words of length 1, and for CCDL path formulae ψ_1, ψ_2, let $\mathcal{A}_{\psi_1,\psi_2} = \langle 2^{\mathrm{AP}}, \{q_1, q_2\}, \delta, q_1, \{q_2\}, \tau \rangle$ be the testing $\mathrm{NWA_f}$ where, for all $a \in 2^{\mathrm{AP}}$, $\delta(q_1, a) = \{q_1, q_2\}$, $\delta(q_2, a) = \emptyset$, $\tau(q_1) = \psi_1$, and $\tau(q_2) = \psi_2$. Then, the next and until formulae $\mathtt{X}\,\psi_1$ and $\psi_1\,\mathtt{U}\,\psi_2$ can be expressed as follows: $\mathtt{X}\,\psi_1 \equiv \langle \mathcal{A} \rangle \psi_1$ and $\psi_1\,\mathtt{U}\,\psi_2 \equiv \psi_2 \vee \langle \mathcal{A}_{\psi_1,\psi_2} \rangle \top$.

4 Alternating Tree Automata

In this section, we recall the class of parity *alternating tree automata with first-order constraints* (FTA for short), introduced in [82] to provide an automata-theoretic characterization of MSO interpreted on arbitrary labeled trees. Moreover, we also recall the class of *graded alternating tree automata* (GTA for short), a subclass of FTA, which was introduced in [44] and allows for expressing counting modal requirements on the child relation of an input tree. The transition relation of both FTA and GTA is based on constraints on the set of states Q written as formulae in a suitable language, called *one-step logic*. The *one-step interpretations* of such formulae over Q are pairs (S, I), where S is an arbitrary (possibly infinite) non-empty set and I is a mapping $\mathrm{I} : \mathrm{S} \mapsto 2^{\mathrm{Q}}$, assigning to each element of S a subset of Q. Intuitively, the pair (S, I) describes the local behaviour of the automaton on reading a node w of the input tree. The set S corresponds to the set of children of the current input node w and, for each $w' \in \mathrm{S}$, $\mathrm{I}(w')$ is the set of states associated with the copies of the automaton which are sent to the child w' of w.

One-Step Logic for GTA. The one-step relation of GTA is specified by means of formulae θ of one-step positive graded modal logic over Q, we call *graded* Q-*constraints*, defined as:

$\theta ::= \top \mid \bot \mid \theta \vee \theta \mid \theta \wedge \theta \mid \Diamond_k \alpha \mid \Box_k \alpha$

where $k \in \mathbb{N} \setminus \{0\}$ and α is a *positive* Boolean formula over Q. The atomic formulae $\Diamond_k \alpha$ and $\Box_k \alpha$ are called Q-*atoms*. The atom $\Diamond_1 \alpha$ (resp., $\Box_1 \alpha$) is also denoted by $\Diamond \alpha$ (resp., $\Box \alpha$). A formula θ is *symmetric* if the atoms occurring in θ are of the form $\Diamond \alpha$ or $\Box \alpha$.

The satisfaction relation $(\mathrm{S}, \mathrm{I}) \models \theta$ for a one-step interpretation (S, I) over Q is inductively defined as follows (we omit the clauses for positive Boolean connectives which are standard):

- $(\mathrm{S}, \mathrm{I}) \models \Diamond_k \alpha$ if $|\{s \in \mathrm{S} \mid \mathrm{I}(s) \models \alpha\}| \geq k$;
- $(\mathrm{S}, \mathrm{I}) \models \Box_k \alpha$ if $|\{s \in \mathrm{S} \mid \mathrm{I}(s) \not\models \alpha\}| < k$.

If $(\mathrm{S}, \mathrm{I}) \models \theta$, we say that (S, I) is a model of θ. Intuitively, for an alternating automaton $\mathcal{A}$ with set of states Q, the atom $\Diamond_k \alpha$ requires that at the current input node w, there are at least k children of w and, for each of such nodes w', (**) there is a subset $\mathrm{Q}' \subseteq \mathrm{Q}$ satisfying α such that a copy of $\mathcal{A}$ is sent to node w' in state q, for each $q \in \mathrm{Q}'$. For an atom $\Box_k \alpha$, the previous condition (**) is required to hold for all but at most $k - 1$ children w' of w.

One-Step Logic for FTA. The one-step language $\mathrm{FOE}_1^+(\mathrm{Q})$ of positive first-order formulae with equality and monadic predicates over Q and first-order variables in Vr_1 is given by the sentences (formulae without free variables) generated by the following grammar:

$\theta ::= \top \mid \bot \mid q(x) \mid x = y \mid x \neq y \mid \theta \vee \theta \mid \theta \wedge \theta \mid \exists x.\, \theta \mid \forall x.\, \theta$

where $q \in \mathrm{Q}$ and $x, y \in \mathrm{Vr}_1$. An $\mathrm{FOE}_1^+(\mathrm{Q})$-sentence θ is called *first-order* Q-*constraint*; θ is *symmetric* if it does not contain equality and inequality atomic formulae. In FTA, these constraints allow formulae that refer to the children of a node of a tree by means of explicit first-order variables.

Given a one-step interpretation (S, I) over Q and an assignment $V : Vr_1 \to S$ of the first-order variables, the satisfaction relation $(S, I), V \models \theta$ is defined in a standard way. For sentences θ, this relation is independent of V, and we simply write $(S, I) \models \theta$. Note that graded Q-constraints can be trivially expressed in $FOE_1^+(Q)$, and first-order Q-constraints θ are *monotonic*, i.e., for all one-step interpretations (S, I) and (S, I') such that $I(s) \subseteq I'(s)$ for each $s \in S$, it holds that $(S, I) \models \theta$ entails $(S, I') \models \theta$. A *minimal model* of θ is a model (S, I) of θ such that there is no model (S, I') of θ with $I' \neq I$ and $I'(s) \subseteq I(s)$ for each $s \in S$.

Parity GTA and Parity FTA. A *parity* GTA $\mathcal{A}$ is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, q_I, \Omega \rangle$, where Σ, Q, q_I, and Ω are defined as for parity NWA, while the transition function δ is a mapping from $Q \times \Sigma$ to the set of graded Q-constraints. The set $\mathsf{Atoms}(\mathcal{A})$ is the set of Q-atoms occurring in the transition function of $\mathcal{A}$. Parity FTA $\mathcal{A} = \langle \Sigma, Q, \delta, q_I, \Omega \rangle$ are defined similarly but the transition function δ is of the form $\delta : Q \times \Sigma \mapsto FOE_1^+(Q)$. A GTA (resp., FTA) $\mathcal{A} = \langle \Sigma, Q, \delta, q_I, \Omega \rangle$ is *symmetric* if for all $(q, a) \in Q \times \Sigma$, the constraint $\delta(q, a)$ is symmetric. GTA (resp., FTA) $\mathcal{A}$ operate over non-blocking Σ-labeled trees (T, Lab). A run of $\mathcal{A}$ over (T, Lab) is a $(Q \times T)$-labeled tree $\mathsf{r} = (T_\mathsf{r}, Lab_\mathsf{r})$, where each node of T_r labelled by (q, w) describes a copy of $\mathcal{A}$ that is in state q reading the node w of T. Moreover, we require that:

- $Lab_\mathsf{r}(\varepsilon) = (q_I, \varepsilon)$ (initially, the automaton is in state q_I reading the root of the input T);
- for each node $y \in T_\mathsf{r}$ with $Lab_\mathsf{r}(y) = (q, w)$ and denoted by S_w the set of children of node w in the input T, there is a one-step interpretation (S_w, I) over Q satisfying $\delta(q, Lab(w))$ such that the set of labels associated with the children of y in T_r consists of the pairs (q', w') with $w' \in S_w$ and $q' \in I(w')$.

The run r is accepting if, for all infinite paths π starting from the root, the infinite sequence of states in $Lab_\mathsf{r}(\pi(0))Lab_\mathsf{r}(\pi(1))\ldots$ satisfies the parity acceptance condition Ω. The language $\mathsf{L}(\mathcal{A})$ accepted by $\mathcal{A}$ is the tree-language over Σ consisting of the non-blocking Σ-labeled trees (T, Lab) such that there is an accepting run of $\mathcal{A}$ over (T, Lab).

Dualization. For a graded Q-constraint θ, the *dual* $\widetilde{\theta}$ of θ is obtained from θ by exchanging $\vee$ with $\wedge$, $\top$ with $\bot$, and Q-atoms $\Diamond_k \alpha$ with $\Box_k \widetilde{\alpha}$, and vice versa, where $\widetilde{\alpha}$ is obtained from α by exchanging $\vee$ with $\wedge$. For example, the dual of $\Diamond_{k_1}(q_0 \vee q_1) \wedge \Box_{k_2} q_2$ is $\Box_{k_1}(q_0 \wedge q_1) \vee \Diamond_{k_2} q_2$. Similarly, the dual $\widetilde{\theta}$ of a first-order Q-constraint θ is obtained from θ by exchanging $\vee$ with $\wedge$, $\top$ with $\bot$, $x = y$ with $x \neq y$, and existential quantification $\exists x$ with universal quantification $\forall x$. For a parity GTA (resp., parity FTA) $\mathcal{A} = \langle \Sigma, Q, \delta, q_I, \Omega \rangle$, the *dual automaton* of $\mathcal{A}$ is the parity GTA (resp., parity FTA) $\widetilde{\mathcal{A}} = \left\langle \Sigma, Q, \widetilde{\delta}, q_I, \widetilde{\Omega} \right\rangle$, where for all $(q, a) \in Q \times \Sigma$, $\widetilde{\Omega}(q) = \Omega(q) + 1$ and $\widetilde{\delta}(q, a)$ is the dual of $\delta(q, a)$. By [82, 12], the following holds.

▶ **Proposition 4.1** ([82, 12]). *Let $\mathcal{A}$ be a parity* GTA *(resp., parity* FTA*). Then, the dual automaton of $\mathcal{A}$ is a parity* GTA *(resp., parity* FTA*) accepting the complement of* $\mathsf{L}(\mathcal{A})$.

5 Automata Characterisations of CDL and CCTL*

In this section, we provide effective automata-theoretic characterisations of the logics CCDL and CCTL*. We first consider the graded version of the class of *hesitant alternating tree automata* (HTA, for short), the latter being a well-known formalism introduced in [47] as an optimal automata-theoretic framework for model checking and synthesis of CTL*. We show that the graded version of HTA (HGTA for short) characterises the logic CCDL. In order to capture the logic CCTL*, we consider a subclass of HGTA obtained by enforcing a counter-freeness requirement on the linear-time behaviour of the automaton along an existential component together with an additional condition (we call *mutual-exclusion property*) on the alphabet of the linearization word automaton.

In the following, for a GTA $\mathcal{A}$ and a set $\mathrm{A} \subseteq \mathsf{Atoms}(\mathcal{A})$, we denote by $\mathsf{Con}(\mathrm{A})$ (resp., $\mathsf{Dis}(\mathrm{A})$) the conjunction (resp., disjunction) of the atoms occurring in A. As usual, the empty conjunction is $\top$ and the empty disjunction is $\bot$.

Hesitant GTA. An *hesitant* GTA (HGTA for short) is a tuple $\mathcal{A} = \langle \Sigma, \mathrm{Q}, \delta, q_I, \mathsf{H}, \mathsf{H}_\exists, \Omega \rangle$, where $\langle \Sigma, \mathrm{Q}, \delta, q_I, \Omega \rangle$ is a parity GTA, $\mathsf{H} = \langle \mathrm{Q}_1, \ldots, \mathrm{Q}_n \rangle$ is an *ordered* tuple of non-empty pairwise disjunct subsets Q_i of Q (called *components* of $\mathcal{A}$) which form a partition of Q, and $\mathsf{H}_\exists$ is a subset of the components in H (the so called *existential components*). Thus, like HTA [47], there is an ordered partition of Q into disjoint sets $\mathrm{Q}_1, \ldots, \mathrm{Q}_n$. Moreover, each component Q_i is classified as *transient*, *existential*, or *universal*, and the following holds:

- *transient requirement*: for each transient component Q_i and $(q, a) \in \mathrm{Q}_i \times \Sigma$, $\delta(q, a)$ only refers to states in components Q_j such that $j < i$;
- *existential requirement*: for each existential component Q_i and $(q, a) \in \mathrm{Q}_i \times \Sigma$, $\delta(q, a)$ can be rewritten as a disjunction of conjunctions of the form $\Diamond q' \wedge \mathsf{Con}(\mathrm{A})$, where $q' \in \mathrm{Q}_i$ and the atoms in A only refer to states in components Q_j such that $j < i$;
- *universal requirement*: for each universal component Q_i and $(q, a) \in \mathrm{Q}_i \times \Sigma$, $\delta(q, a)$ can be rewritten as a conjunction of disjunctions of the form $\Box q' \vee \mathsf{Dis}(\mathrm{A})$, where $q' \in \mathrm{Q}_i$ and the atoms in A only refer to states in components Q_j such that $j < i$;
- *hesitant acceptance requirement*: for each existential (resp., universal) component Q_i, the restriction Ω_{Q_i} of Ω to the set Q_i is a Büchi condition (resp., coBüchi condition).

The first three requirements ensure that every infinite path of a run of $\mathcal{A}$ gets trapped within some existential or universal component Q_i. The existential requirement establishes that from each existential state $q \in \mathrm{Q}_i$, exactly one copy is sent to a child of the current input node in component Q_i (all the other copies move to states with order lower than i). The universal requirement corresponds to the dual of the existential requirement. Finally, the hesitant acceptance requirement ensures that for each infinite path π of a run that gets trapped into an existential (resp., universal) component, π is accepting iff π visits infinitely many times states with even color (resp., π visits finitely many times states with odd color).

▶ **Example 5.1.** Let $\mathrm{AP} = \{p\}$ and φ_p be the CTL* formula $\mathtt{EX}\, p$ asserting that the root of the given Kripke tree has a child where p holds. We consider the tree-language L_2 consisting of the Kripke trees $\mathcal{T}$ such that there is an infinite path π from the root so that p never holds along π and at the even positions $2i$, φ_p holds at node $\pi(2i)$. L_2 requires counting modulo 2 and cannot be expressed in CCTL*. The language L_2 is recognised by the HGTA $\mathcal{A} = \langle \Sigma, \mathrm{Q}, \delta, q_I, \langle \mathrm{Q}_1, \mathrm{Q}_2 \rangle, \{\mathrm{Q}_2\}, \Omega \rangle$ consisting of three states having colour 0: the existential states q_I and q having the same and highest order ($\mathrm{Q}_2 = \{q_I, q\}$) and the transient state q_p ($\mathrm{Q}_1 = \{q_p\}$). Moreover, (i) $\delta(q_p, \{p\}) = \top$ and $\delta(q_p, \emptyset) = \bot$, (ii) $\delta(q_I, \emptyset) = \Diamond q \wedge \Diamond q_p$ and $\delta(q_I, \{p\}) = \bot$, and (iii) $\delta(q, \emptyset) = \Diamond q_I$ and $\delta(q, \{p\}) = \bot$.

Linearization. Fix an HGTA $\mathcal{A} = \langle \Sigma, \mathrm{Q}, \delta, q_I, \mathsf{H}, \mathsf{H}_\exists, \Omega \rangle$. Given a component Q_i of $\mathcal{A}$ and $\mathrm{A} \subseteq \mathsf{Atoms}(\mathcal{A})$, the set A *is lower than* Q_i if the atoms in A only refer to states with order $j < i$. For each existential (resp., universal) component Q_i and $q \in \mathrm{Q}_i$, we introduce a Büchi (resp., coBüchi) NWA $\mathcal{A}_{\mathrm{Q}_i, q}$ over the alphabet $\Sigma \times \mathsf{Atoms}(\mathcal{A})$. Intuitively, $\mathcal{A}_{\mathrm{Q}_i, q}$ encodes the *modular* behaviour of $\mathcal{A}$ starting at state q, which is composed of the behaviour along Q_i (which is linear-time when Q_i is existential), plus additional moves that lead to states with order lower than i: the input alphabet $\Sigma \times \mathsf{Atoms}(\mathcal{A})$ keeps track of these additional moves. When Q_i is universal, then $\mathcal{A}_{\mathrm{Q}_i, q}$ can be viewed as a universal tree automaton.

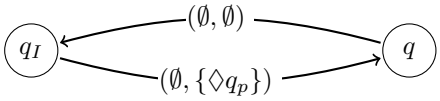
▶ **Definition 5.2** (Linearization word automata). *For each non-transient component* Q_i *of* $\mathcal{A}$ *and* $q \in \mathrm{Q}_i$*, we denote by* $\mathcal{A}_{\mathrm{Q}_i,q}$ *the parity* NWA $\mathcal{A}_{\mathrm{Q}_i,q} = \langle \Sigma \times 2^{\mathsf{Atoms}(\mathcal{A})}, \mathrm{Q}_i, \delta_{\mathrm{Q}_i}, q, \Omega_{\mathrm{Q}_i} \rangle$ *where for all* $q' \in \mathrm{Q}_i$*,* $a \in \Sigma$*, and* $\mathrm{A} \subseteq \mathsf{Atoms}(\mathcal{A})$*,* $\delta_{\mathrm{Q}_i}(q', (a, \mathrm{A}))$ *is defined as follow:*

- *Case* Q_i *is existential:* $q'' \in \delta_{\mathrm{Q}_i}(q', (a, \mathrm{A}))$ *if there is conjunction* ξ *in the disjunctive normal form of* $\delta(q', a)$ *such that* $\xi = \Diamond q'' \wedge \mathsf{Con}(\mathrm{A})$ *(note that* A *is lower than* Q_i*).*
- *Case* Q_i *is universal:* $q'' \in \delta_{\mathrm{Q}_i}(q', (a, \mathrm{A}))$ *if there is disjunction* ξ *in the conjunctive normal form of* $\delta(q', a)$ *such that* $\xi = \Box q'' \vee \mathsf{Dis}(\mathrm{A})$ *(note that* A *is lower than* Q_i*).*

Let Υ_{Q_i} *be the set of elements* $\mathrm{A} \subseteq \mathsf{Atoms}(\mathcal{A})$ *s.t.* $\delta_{\mathrm{Q}_i}(q', (a, \mathrm{A})) \neq \emptyset$ *for some* $(q', a) \in \mathrm{Q}_i \times \Sigma$*.*

▶ Remark 5.3. Note that the transition function of $\mathcal{A}_{\mathrm{Q}_i,q}$ is independent of q, and $\mathcal{A}_{\mathrm{Q}_i,q}$ is a Büchi (resp., coBüchi) NWA if Q_i is existential (resp., universal). We can equate the parity NWA $\mathcal{A}_{\mathrm{Q}_i,q}$ to the parity NWA over the alphabet $\Sigma \times \Upsilon_{\mathrm{Q}_i}$ which is obtained from $\mathcal{A}_{\mathrm{Q}_i,q}$ by restricting the transition function to the alphabet $\Sigma \times \Upsilon_{\mathrm{Q}_i}$. In the following, we write $\mathcal{A}_{\mathrm{Q}_i,q}$ to denote this automaton. Observe that each set of atoms $\mathrm{A} \in \Upsilon_{\mathrm{Q}_i}$ is lower than Q_i.

If we consider the HGTA $\mathcal{A}$ of Example 5.1, the Büchi NWA $\mathcal{A}_{\mathrm{Q}_2,q_I}$ associated with the existential component Q_2 is illustrated below. Note that $\Upsilon_{\mathrm{Q}_2} = \{\emptyset, \{\Diamond q_p\}\}$.



Let us fix an HGTA $\mathcal{A} = \langle \Sigma, \mathrm{Q}, \delta, q_I, \mathsf{H}, \mathsf{H}_\exists, \Omega \rangle$ with $\mathsf{H} = \langle \mathrm{Q}_1, \ldots, \mathrm{Q}_n \rangle$. For each graded Q-constraint θ, we denote by $\mathcal{A}^\theta$ the HGTA $\langle \Sigma, \mathrm{Q} \cup \{\theta\}, \delta_\theta, \theta, \mathsf{H}_\theta, \mathsf{H}_\exists, \Omega \cup (\theta \to 0) \rangle$ where for the states in Q, δ_θ agrees with δ, for the initial state θ, $\delta_\theta(\theta, a) = \theta$ for all $a \in \Sigma$, and $\mathsf{H}_\theta = \langle \mathrm{Q}_1, \ldots, \mathrm{Q}_n, \{\theta\} \rangle$. Note that $\{\theta\}$ is a transient component with highest order. Thus, from the root of the input tree, $\mathcal{A}^\theta$ send copies of $\mathcal{A}$ to the children of the root according to the constraint θ. By construction, for each existential state q of an HGTA $\mathcal{A}$, we obtain the following characterisation of the language $\mathsf{L}(\mathcal{A}^q)$, where $\mathcal{A}^q$ is the HGTA obtained from $\mathcal{A}$ by setting q as initial state instead of q_I, in terms of the linearization of $\mathcal{A}$.

▶ **Proposition 5.4.** *Let* $\mathcal{A}$ *be an* HGTA*,* Q_i *be an existential component of* $\mathcal{A}$*, and* $q \in \mathrm{Q}_i$*. Then, for each input* $\mathcal{T} = (\mathrm{T}, Lab)$*,* $\mathcal{T} \in \mathsf{L}(\mathcal{A}^q)$ if and only if *there is an infinite path* π *of* $\mathcal{T}$ *starting at the root and an infinite word* $\rho \in \mathsf{L}(\mathcal{A}_{\mathrm{Q}_i,q})$ *such that* ρ *is of the form* $\rho = (Lab(\pi(0)), \mathrm{A}_0)(Lab(\pi(1)), \mathrm{A}_1) \ldots$ *and for each* $i \geq 0$*,* $\mathcal{T}_{\pi(i)} \in \mathsf{L}(\mathcal{A}^{\mathsf{Con}(\mathrm{A}(i))})$*, where* $\mathcal{T}_{\pi(i)}$ *is the labelled subtree of* $\mathcal{T}$ *rooted at node* $\pi(i)$*.*

Counter-free HGTA**.** In order to capture CCTL*, we introduce a subclass of HGTA obtained by enforcing additional conditions. By Proposition 5.4 and the equivalence of LTL and Büchi *counter-free* NWA [19], a natural condition consists in requiring that for each non-transient component Q_i of the HGTA and state $q \in \mathrm{Q}_i$, the NWA $\mathcal{A}_{\mathrm{Q}_i,q}$ is counter-free (*counter-freeness requirement*).[1] However, this condition is not sufficient for characterising the logic CCTL*. A counterexample is the HGTA $\mathcal{A}$ of Example 5.1 which clearly satisfies the counter-freeness requirement but recognises a tree-language which is not expressible in CCTL*. We introduce an additional condition (*mutual-exclusion property*) on the alphabets of the linearization automata (see Definition 5.5 below). A *Counter-free* HGTA ($\mathrm{HGTA}_{\mathrm{cf}}$ for short) is an HGTA satisfying both the counter-free requirement and the mutual-exclusion condition.

[1] Note that the property of an NWA to be counter-free is independent of the initial state.

▶ **Definition 5.5.** *An* HGTA $\mathcal{A}$ *satisfies the* mutual-exclusion property *if for each non-transient component* Q_i *and for all* $\mathrm{A}, \mathrm{A}' \in \Upsilon_{\mathrm{Q}_i}$ *such that* $\mathrm{A} \neq \mathrm{A}'$*, it holds that there exists an atom* $\mathsf{atom} \in \mathrm{A}$ *and an atom* $\mathsf{atom}' \in \mathrm{A}'$ *such that* $\mathsf{L}(\mathcal{A}^{\mathsf{atom}})$ *is the complement of* $\mathsf{L}(\mathcal{A}^{\mathsf{atom}'})$*. Note that if* Υ_{Q_i} *is a singleton, then the previous property is fulfilled.*

Evidently, if $\mathcal{A}$ satisfies the mutual-exclusion condition, then for each non-transient component Q_i and for all $\mathrm{A}, \mathrm{A}' \in \Upsilon_{\mathrm{Q}_i}$ such that $\mathrm{A} \neq \mathrm{A}'$, it holds that $\mathsf{L}(\mathcal{A}^{\mathsf{Con}(\mathrm{A})}) \cap \mathsf{L}(\mathcal{A}^{\mathsf{Con}(\mathrm{A}')}) = \emptyset$. Intuitively, the mutual-exclusion condition requires that along a non-transient component Q_i, the distinct moves $\mathrm{A} \in \Upsilon_{\mathrm{Q}_i}$ (these moves lead to components with order lower than i) are mutually exclusive. Let us consider again the HGTA $\mathcal{A}$ of Example 5.1. Since $\Upsilon_{\mathrm{Q}_2} = \{\emptyset, \{\Diamond q_p\}\}$, by Definition 5.5, $\mathcal{A}$ does not satisfy the mutual-exclusion condition. Note that $\mathsf{Con}(\emptyset) = \top$ and $\mathsf{Con}(\{\Diamond q_p\}) = \Diamond q_p$. Hence, $\mathsf{L}(\mathcal{A}^{\top}) \cap \mathsf{L}(\mathcal{A}^{\mathsf{Con}(\{\Diamond q_p\})}) = \mathsf{L}(\mathcal{A}^{\mathsf{Con}(\{\Diamond q_p\})}) = \mathsf{L}(\mathtt{EX}\, p) \neq \emptyset$.

The dual $\widetilde{\mathcal{A}}$ of an HGTA $\mathcal{A} = \langle \Sigma, \mathrm{Q}, \delta, q_I, \mathsf{H}, \mathsf{H}_\exists, \Omega \rangle$ is the tuple $\left\langle \Sigma, \mathrm{Q}, \widetilde{\delta}, q_I, \mathsf{H}, \widetilde{\mathsf{H}_\exists}, \widetilde{\Omega} \right\rangle$, where $\widetilde{\delta}$ and $\widetilde{\Omega}$ are defined as for the dual of an arbitrary parity GTA and $\widetilde{\mathsf{H}_\exists}$ consists of the universal components of $\mathcal{A}$. By construction and Proposition 4.1, the considered subclasses of GTA are closed under Boolean language operations.

▶ **Proposition 5.6.** HGTA *(resp.,* $\mathrm{HGTA}_{\mathrm{cf}}$*) and* HGTA *satisfying the mutual-exclusion property are effectively closed under Boolean language operations.*

Enforcing the Mutual-exclusion Property. By exploiting dualization, an HGTA $\mathcal{A}$ can be converted into an equivalent HGTA $\mathcal{A}_s$ satisfying the mutual-exclusion condition. Intuitively, $\mathcal{A}_s$ is obtained by merging in a syntactical and *modular* way $\mathcal{A}$ with a renaming of the dual HGTA $\widetilde{\mathcal{A}}$.

▶ **Proposition 5.7.** *Given an* HGTA $\mathcal{A}$*, one can construct an* HGTA $\mathcal{A}_s$ *such that* $\mathcal{A}_s$ *satisfies the mutual-exclusion condition and* $\mathsf{L}(\mathcal{A}_s) = \mathsf{L}(\mathcal{A})$*.*

Note that the translation in Proposition 5.7 changes the second component Υ_{Q_i} of the alphabets of the linearization automata. Since counter-free NWA are not closed under inverse projection, the construction does not preserve the counter-freeness property. For example, for the HGTA of Example 5.1, the translation replaces the edge from q to q_I with label $(\emptyset, \emptyset)$ of the NWA $\mathcal{A}_{\mathrm{Q}_2, q_I}$ with two edges from q to q_I: one with label $(\emptyset, \{\Diamond q_p\})$ and the other one with label $(\emptyset, \{\Box q'_p\})$ where $\mathsf{L}(\mathcal{A}^{\mathsf{Con}(\{\Box q'_p\})}) = \mathsf{L}(\neg \mathtt{EX}\, p)$. The resulting NWA is not counter-free.

5.1 From Automata to Logics and Back

In this section, we show that the class of HGTA and the logic CCDL are effectively equivalent, and the class of $\mathrm{HGTA}_{\mathrm{cf}}$ effectively characterizes CCTL*. We start with the translations from automata to logics.

▶ **Theorem 5.8.** *Let* $\mathcal{A}$ *be an* HGTA *(resp., an* $\mathrm{HGTA}_{\mathrm{cf}}$*) over* 2^{AP}*. Then, one can construct a* CCDL *(resp.,* CCTL**) formula* $\varphi_{\mathcal{A}}$ *such that* $\mathsf{L}(\varphi_{\mathcal{A}}) = \mathsf{L}(\mathcal{A})$*. Moreover,* $\varphi_{\mathcal{A}}$ *is a* CDL *(resp. a* CTL**) formula if* $\mathcal{A}$ *is symmetric.*

Proof. We focus on the translation from $\mathrm{HGTA}_{\mathrm{cf}}$ $\mathcal{A} = \langle \Sigma, \mathrm{Q}, \delta, q_I, \mathsf{H}, \mathsf{H}_\exists, \Omega \rangle$ to CCTL*. For each $q \in \mathrm{Q}$, we construct a CCTL* formula φ_q such that $\mathsf{L}(\varphi_q) = \mathsf{L}(\mathcal{A}^q)$ and φ_q is a CTL* formula if $\mathcal{A}$ is symmetric. Thus, by setting $\varphi_{\mathcal{A}} \triangleq \varphi_{q_I}$, Theorem 5.8 directly follows. The proof is by induction on the order ℓ of the component Q_ℓ such that $q \in \mathrm{Q}_\ell$. We distinguish

the cases where q is transient, existential, or universal. The transient case is easy and the universal case follows from the existential case by a dualization argument. Now, assume that q is existential. Let us consider the Büchi NWA $\mathcal{A}_{\mathrm{Q}_\ell,q}$ over $2^{\mathrm{AP}} \times \Upsilon_{\mathrm{Q}_\ell}$ as defined in Definition 5.2. Recall that $\mathcal{A}_{\mathrm{Q}_\ell,q}$ is counter-free. Moreover, $\Upsilon_{\mathrm{Q}_\ell} \subseteq 2^{\mathsf{Atoms}(\mathcal{A})}$ contains only elements A such that states occurring in the atoms of A have order j lower than ℓ. Thus, by the induction hypothesis, for each $\mathrm{A} \in \Upsilon_{\mathrm{Q}_\ell}$, one can build a CCTL* formula φ_{A} such that $\mathsf{L}(\mathcal{A}^{\mathsf{Con}(\mathrm{A})}) = \mathsf{L}(\varphi_{\mathrm{A}})$. Hence, since $\mathcal{A}$ satisfies the mutual-exclusion condition, the following holds:

Claim 1. For all $\mathrm{A}, \mathrm{A}' \in \Upsilon_{\mathrm{Q}_\ell}$ such that $\mathrm{A} \neq \mathrm{A}'$, $\mathsf{L}(\varphi_{\mathrm{A}}) \cap \mathsf{L}(\varphi_{\mathrm{A}'}) = \emptyset$.

For each $\mathrm{A} \in \Upsilon_{\mathrm{Q}_\ell}$, let p_{A} be a fresh atomic proposition. We denote by $\mathrm{AP}_{\mathtt{ex}}$ the extension of AP with these fresh propositions. Moreover, let $\mathcal{A}_{\mathtt{ex},\mathrm{Q}_\ell,q}$ be the Büchi NWA over $2^{\mathrm{AP}_{\mathtt{ex}}}$ having the same set of states, initial state, acceptance condition as $\mathcal{A}_{\mathrm{Q}_\ell,q}$ and whose transition function $\delta_{\mathtt{ex},\mathrm{Q}_\ell}$ is obtained from the transition function δ_{Q_ℓ} of $\mathcal{A}_{\mathrm{Q}_\ell,q}$ as follows: for all $q' \in \mathrm{Q}_\ell$ and $a_{\mathtt{ex}} \in 2^{\mathrm{AP}_{\mathtt{ex}}}$, if $a_{\mathtt{ex}}$ is of the form $a \cup \{p_{\mathrm{A}}\}$, for some $a \in 2^{\mathrm{AP}}$ and $\mathrm{A} \in \Upsilon_{\mathrm{Q}_\ell}$, (i.e., $a_{\mathtt{ex}}$ contains a unique proposition in $\mathrm{AP}_{\mathtt{ex}} \setminus \mathrm{AP}$), then $\delta_{\mathtt{ex},\mathrm{Q}_\ell}(q', a_{\mathtt{ex}}) = \delta_{\mathrm{Q}_\ell}(q', (a, \mathrm{A}))$; otherwise, $\delta_{\mathtt{ex},\mathrm{Q}_\ell}(q', a_{\mathtt{ex}}) = \emptyset$. Being $\mathcal{A}_{\mathrm{Q}_\ell,q}$ counter-free, $\mathcal{A}_{\mathtt{ex},\mathrm{Q}_\ell,q}$ is clearly counter-free as well. Thus, by [19], one can construct an LTL formula ψ over $\mathrm{AP}_{\mathtt{ex}}$ such that $\mathsf{L}(\psi) = \mathsf{L}(\mathcal{A}_{\mathtt{ex},\mathrm{Q}_\ell,q})$. Since $\mathsf{L}(\mathcal{A}^{\mathsf{Con}(\mathrm{A})}) = \mathsf{L}(\varphi_{\mathrm{A}})$ for all $\mathrm{A} \in \Upsilon_{\mathrm{Q}_\ell}$, by construction and Proposition 5.4, we obtain the following characterization of the tree-language $\mathsf{L}(\mathcal{A}^q)$.

Claim 2. For each Kripke tree $\mathcal{T} = (\mathrm{T}, \mathit{Lab})$, $\mathcal{T} \in \mathsf{L}(\mathcal{A}^q)$ iff there is an infinite path π of $\mathcal{T}$ from the root and an infinite word ρ over $2^{\mathrm{AP}_{\mathtt{ex}}}$ such that $\rho \models \psi$ and, for all $j \geq 0$, (i) $\rho(j) \cap \mathrm{AP} = \mathit{Lab}(\pi(j))$, (ii) for all $p_{\mathrm{A}} \in \rho(j)$, $(\mathcal{T}, \pi(j)) \models \varphi_{\mathrm{A}}$, and (iii) there is a unique $\mathrm{A} \in \Upsilon_{\mathrm{Q}_\ell}$ such that $p_{\mathrm{A}} \in \rho(j)$.

Note that since $\mathsf{L}(\psi) = \mathsf{L}(\mathcal{A}_{\mathtt{ex},\mathrm{Q}_\ell,q})$, by construction, point (iii) in Claim 2 follows from the fact that $\rho \models \psi$. By exploiting the always modality $\mathtt{G}$ ($\mathtt{G}\,\xi$ is a shorthand of $\neg(\top\,\mathtt{U}\,\neg\xi)$) and both conjunction and disjunction, w.l.o.g. we can assume that the LTL formula ψ is in negation normal form, i.e., negation is applied only to atomic propositions. Now, let $f(\psi)$ be the CCTL* path formula over AP obtained from ψ by replacing each literal of the form p_{A} (resp., $\neg p_{\mathrm{A}}$), where $\mathrm{A} \in \Upsilon_{\mathrm{Q}_\ell}$, with the CCTL* state formula φ_{A} (resp., $\bigvee_{\mathrm{A}' \in \Upsilon_{\mathrm{Q}_\ell} \setminus \{\mathrm{A}\}} \varphi_{\mathrm{A}'}$). Finally, let us consider the CCTL* state formula φ_q defined as follows:

$\varphi_q \triangleq \mathtt{E}(f(\psi) \wedge \mathtt{G} \bigvee_{\mathrm{A} \in \Upsilon_{\mathrm{Q}_\ell}} \varphi_{\mathrm{A}})$.

Note that the second conjunct in the state formula φ_q ensures that, for the infinite path π selected by the path quantifier $\mathtt{E}$ and for each $j \geq 0$, the state formula φ_{A} holds at node $\pi(j)$ for some $\mathrm{A} \in \Upsilon_{\mathrm{Q}_\ell}$. We show that a Kripke tree $\mathcal{T} = (\mathrm{T}, \mathit{Lab})$ satisfies φ_q iff the characterization of $\mathsf{L}(\mathcal{A}^q)$ in Claim 2 holds. Hence, the result follows.

We shall now focus on the left-right implication of the equivalence (the right-left implication is similar). Thus, assume that $\mathcal{T} \models \varphi_q$. Hence, there exists an infinite path π of $\mathcal{T}$ from the root and an infinite sequence $\nu = \mathrm{A}_0, \mathrm{A}_1, \ldots$ over $\Upsilon_{\mathrm{Q}_\ell}$ such that $(\mathcal{T}, \pi, 0) \models f(\psi)$ and for each $j \geq 0$, $(\mathcal{T}, \pi(j)) \models \varphi_{\mathrm{A}_j}$. Let $\mathit{Lab}(\pi) \otimes \nu$ be the infinite word over $2^{\mathrm{AP}_{\mathtt{ex}}}$ defined as follows for all $j \geq 0$: $(\mathit{Lab}(\pi) \otimes \nu)(j) = \mathit{Lab}(\pi(j)) \cup \{p_{\mathrm{A}_j}\}$. By Claim 2, it suffices to show that $\mathit{Lab}(\pi) \otimes \nu \models \psi$. To this purpose, we show by structural induction that for each $j \geq 0$ and subformula θ of ψ if $(\mathcal{T}, \pi, j) \models f(\theta)$, then $(\mathit{Lab}(\pi) \otimes \nu, j) \models \theta$. Since the formula ψ is in negation normal form, by the induction hypothesis, the unique non-trivial case is when θ is either of the form p_{A} or of the form $\neg p_{\mathrm{A}}$ for some $\mathrm{A} \in \Upsilon_{\mathrm{Q}_\ell}$.

- $\theta = p_{\mathrm{A}}$: hence, $f(\theta) = \varphi_{\mathrm{A}}$. Since $(\mathcal{T}, \pi, j) \models f(\theta)$ and $(\mathcal{T}, \pi(j)) \models \varphi_{\mathrm{A}_j}$, by Claim 1, it follows that $\mathrm{A} = \mathrm{A}_j$, i.e. $\theta = p_{\mathrm{A}_j}$. Hence, $(\mathit{Lab}(\pi) \otimes \nu, j) \models \theta$, and the result follows.
- $\theta = \neg p_{\mathrm{A}}$: hence $f(\theta) = \bigvee_{\mathrm{A}' \in \Upsilon_{\mathrm{Q}_\ell} \setminus \{\mathrm{A}\}} \varphi_{\mathrm{A}'}$. Since $(\mathcal{T}, \pi, j) \models f(\theta)$ and $(\mathcal{T}, \pi(j)) \models \varphi_{\mathrm{A}_j}$, by Claim 1, $\mathrm{A} \neq \mathrm{A}_j$. Hence, $(\mathit{Lab}(\pi) \otimes \nu, j) \models \theta$, and we are done. ◀

From Logics to Automata. As to the translation from CCTL* to $\mathrm{HGTA}_{\mathrm{cf}}$, in order to ensure the mutual-exclusion property of the resulting $\mathrm{HGTA}_{\mathrm{cf}}$, we need a restricted syntactic form of CCTL* formulae, which is still expressively complete. A CCTL* formula is in *simple form* if each occurrence of the path quantifier $\mathtt{E}$ is immediately preceded by the counter modality $\mathtt{D}^1$ (note that $\mathtt{D}^1$ corresponds to the standard $\mathtt{EX}$ modality of CTL*). Formally, the set of state formulae φ of CCTL* in simple form is defined according to the following syntax: $\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathtt{D}^1\mathtt{E}\psi \mid \mathtt{D}^n\varphi$. One can easily show that the simple form is indeed expressively complete.

▶ **Theorem 5.9.** *Given a* CCDL *(resp.,* CCTL*) formula φ, one can construct an equivalent* HGTA *(resp.,* $\mathrm{HGTA}_{\mathrm{cf}}$*) $\mathcal{A}_\varphi$ such that $\mathsf{L}(\mathcal{A}_\varphi) = \mathsf{L}(\varphi)$. Moreover, $\mathcal{A}_\varphi$ is symmetric if φ is a* CDL *(resp., a* CTL*) formula.*

Proof. We focus on the translation from CCTL* to $\mathrm{HGTA}_{\mathrm{cf}}$. Fix a CCTL* formula Φ. W.l.o.g., we can assume that Φ is in simple form. As in the case of the alternating hesitant automata for CTL* [47], we construct the automaton by induction on the structure of Φ. With each state subformula φ of Φ we associate an $\mathrm{HGTA}_{\mathrm{cf}}$ $\mathcal{A}_\varphi$ over $\Sigma = 2^{\mathrm{AP}}$ such that $\mathsf{L}(\mathcal{A}_\varphi) = \mathsf{L}(\varphi)$. The cases where φ is an atomic proposition, or the root operator of φ is the counting modality $\mathtt{D}^n$ are straightforward. The cases where the root operator of φ is a Boolean connective directly follow from Proposition 5.6. Now, assume that $\varphi = \mathtt{E}\psi$ for some path formula ψ. Let $\mathsf{max}(\psi)$ be the set of state subformulae of ψ of the form $\mathtt{E}\xi$ or $\mathtt{D}^n\xi$ which are not preceded by the modality $\mathtt{E}$ or the counting modality in the syntax tree of ψ. Since ψ is in simple form, $\mathsf{max}(\psi)$ is of the form $\{\mathtt{D}^{n_1}\varphi_1, \ldots, \mathtt{D}^{n_k}\varphi_k\}$ for some $k \geq 0$, where $\varphi_1, \ldots, \varphi_k$ are CCTL* formulae in simple form. Note that if ψ is a CTL* formula, then $n_1 = \ldots = n_k = 1$. By the induction hypothesis, for each $i \in [1,k]$, one can construct an $\mathrm{HGTA}_{\mathrm{cf}}$ $\mathcal{A}_i = \langle 2^{\mathrm{AP}}, \mathrm{Q}_i, \delta_i, q_{I\,i}, \mathsf{H}_i, \mathsf{H}_{\exists,i}, \Omega_i\rangle$ such that $\mathsf{L}(\mathcal{A}_i) = \mathsf{L}(\varphi_i)$. For each $i \in [1,k]$, let $\widetilde{\mathcal{A}}_i = \left\langle 2^{\mathrm{AP}}, \widetilde{\mathrm{Q}}_i, \widetilde{\delta}_i, \widetilde{q}_{I\,i}, \widetilde{\mathsf{H}}_i, \widetilde{\mathsf{H}}_{\exists,i}, \widetilde{\Omega}_i\right\rangle$ be a renaming of the dual automaton of $\mathcal{A}_i$.

Let $\mathrm{AP}_{\mathtt{ex}}$ be an extension of AP obtained by adding for each state formula $\mathtt{D}^{n_i}\varphi_i$ a fresh proposition p_i. Then, the path formula ψ can be viewed as an LTL formula $\psi_{\mathtt{ex}}$ over $\mathrm{AP}_{\mathtt{ex}}$. By [19], one can build a Büchi *counter-free* NWA $\mathcal{N}_\psi = \langle 2^{\mathrm{AP}_{\mathtt{ex}}}, \mathrm{Q}, \delta, q_I, \Omega\rangle$ s.t. $\mathsf{L}(\mathcal{N}_\psi) = \mathsf{L}(\psi_{\mathtt{ex}})$. By construction, we easily deduce the following characterization of $\mathsf{L}(\varphi) = \mathsf{L}(\mathtt{E}\psi)$:

Claim 1: for each Kripke tree $\mathcal{T} = (\mathrm{T}, Lab)$, $\mathcal{T} \in \mathsf{L}(\varphi)$ *iff* there exists an infinite path π of $\mathcal{T}$ from the root and an infinite word ρ over $2^{\mathrm{AP}_{\mathtt{ex}}}$ such that $\rho \in \mathsf{L}(\mathcal{N}_\psi)$ and the following holds for each $i \geq 0$: (i) $\rho(i) \cap \mathrm{AP} = Lab(\pi(i))$, (ii) for each $\ell \in [1,k]$ such that $p_\ell \in \rho(i)$, $(\mathcal{T}, \pi(i)) \models \mathtt{D}^{n_\ell}\varphi_\ell$, and (iii) for each $\ell \in [1,k]$ such that $p_\ell \notin \rho(i)$, $(\mathcal{T}, \pi(i)) \models \neg\mathtt{D}^{n_\ell}\varphi_\ell$.

We define $\mathcal{A}_\varphi$ as follows: $\mathcal{A}_\varphi$ simulates the Büchi NWA $\mathcal{N}_\psi$ along a guessed infinite path of the input tree from the root and starts additional copies of the $\mathrm{HGTA}_{\mathrm{cf}}$ $\mathcal{A}_1, \ldots, \mathcal{A}_k, \widetilde{\mathcal{A}}_1, \ldots, \widetilde{\mathcal{A}}_k$. According to Claim 1, these copies guarantee that whenever the NWA $\mathcal{N}_\psi$ assumes that proposition p_ℓ labels (resp., p_ℓ does not label) the current node along the guessed path, then $\mathtt{D}^{n_\ell}\varphi_\ell$ holds (resp., $\mathtt{D}^{n_\ell}\varphi_\ell$ does not hold) at this node. The components of $\mathcal{A}$ consist of the existential component Q (the set of states of the Büchi counter-free NWA $\mathcal{N}_\psi$) and the components of the $\mathrm{HGTA}_{\mathrm{cf}}$ automata $\mathcal{A}_i$ and $\widetilde{\mathcal{A}}_i$ for each $i \in [1,k]$. Moreover, the existential component Q has highest order and the ordering of the components of $\mathcal{A}_i$ (resp., $\widetilde{\mathcal{A}}_i$) is preserved for each $i \in [1,k]$. For the transition function δ_φ of $\mathcal{A}_\varphi$, we have that for states in Q_i (resp., $\widetilde{\mathrm{Q}}_i$), δ_φ agrees with the corresponding δ_i (resp., $\widetilde{\delta}_i$). For states $q \in \mathrm{Q}$ and $a \in 2^{\mathrm{AP}}$, $\delta_\varphi(q,a)$ is defined as follows, where for each $\mathrm{I} \subseteq [1,k]$, $\mathrm{I}(a)$ denotes the subset of $\mathrm{AP}_{\mathtt{ex}}$ given by $a \cup \bigcup_{\ell\in\mathrm{I}}\{p_\ell\}$:

$$\delta_\varphi(q,a) \triangleq \bigvee_{\mathrm{I}\subseteq[1,k]} \bigvee_{q'\in\delta(q,\mathrm{I}(a))} (\Diamond q' \wedge \bigwedge_{\ell\in\mathrm{I}} \Diamond_\ell q_{I_i} \wedge \bigwedge_{\ell\in[1,k]\setminus\mathrm{I}} \Box_\ell \widetilde{q}_{I_i})$$

By construction, the induction hypothesis, and Claim 1, $\mathcal{A}_\varphi$ is an HGTA satisfying the mutual-exclusion property such that $\mathsf{L}(\mathcal{A}_\varphi) = \mathsf{L}(\varphi)$. It remains to show that for each $q \in \mathrm{Q}$, the Büchi NWA $\mathcal{A}_{\mathrm{Q},q}$ over the alphabet $2^{\mathrm{AP}} \times \Upsilon_{\mathrm{Q}}$ (see Definition 5.2) driven by the existential component Q of $\mathcal{A}_\varphi$ is counter-free. Let us consider the mapping g assigning to each $a_{\mathtt{ex}} \in 2^{\mathrm{AP}_{\mathtt{ex}}}$ the pair $(a, \bigcup_{\ell\in\mathrm{I}}\{\Diamond_\ell q_{I_i}\} \cup \bigcup_{\ell\in[1,k]\setminus\mathrm{I}}\{\Box_\ell \widetilde{q}_{I_i}\})$, where $a = \mathrm{AP} \cap a_{\mathtt{ex}}$ and I is the set of indexes in $j \in [1,k]$ such that $p_j \in a_{\mathtt{ex}}$. Clearly, g is a bijection between $2^{\mathrm{AP}_{\mathtt{ex}}}$ and $2^{\mathrm{AP}} \times \Upsilon_{\mathrm{Q}}$. Moreover, for the transition functions δ_{Q} and δ of $\mathcal{A}_{\mathrm{Q},q}$ and $\mathcal{N}_\psi$, respectively, it holds that, for each $(a,\mathrm{A}) \in 2^{\mathrm{AP}} \times \Upsilon_{\mathrm{Q}}$ and $q' \in \mathrm{Q}$, $\delta_{\mathrm{Q}}(q',(a,\mathrm{A})) = \delta(q', g^{-1}(a,\mathrm{A}))$, where g^{-1} is the inverse of g. Thus, since $\mathcal{N}_\psi$ is counter free, $\mathcal{A}_{\mathrm{Q},q}$ is counter free as well, and the result follows. ◀

6 Automata Characterisation of Monadic Chain Logic (MCL)

Monadic Chain Logic (MCL) is the fragment of MSO over Kripke trees where monadic second-order quantification is restricted to sets of nodes which forms chains, *i.e.* a subset of a path. In this section, we provide an automata-theoretic characterisation of MCL in terms of a subclass of parity FTA, called *Hesitant* FTA (HFTA for short), which represents the FTA counterpart of hesitant GTA. Moreover, we show that the bisimulation-invariant fragment of MCL and CDL are expressively equivalent.

The class of HFTA**.** An HFTA is a tuple $\mathcal{A} = \langle \Sigma, \mathrm{Q}, \delta, q_I, \mathsf{H}, \mathsf{H}_\exists, \Omega\rangle$, where $\langle \Sigma, \mathrm{Q}, \delta, q_I, \Omega\rangle$ is an FTA and H and $\mathsf{H}_\exists$ are defined as for HGTA. Moreover, we require that $\mathcal{A}$ satisfies the transient requirement and the hesitant acceptance requirement of HGTA and the following variants of the existential and universal requirements of HGTA:

- for each existential component Q_i and $(q,a) \in \mathrm{Q}_i \times \Sigma$, $\delta(q,a)$ is a disjunction of formulae of the form $\exists x.\,(q'(x) \wedge \theta(x))$ where $q' \in \mathrm{Q}_i$ and $\theta(x)$ only refers to states in lower components Q_j with $j < i$ (*first-order existential requirement*);
- for each universal component Q_i and $(q,a) \in \mathrm{Q}_i \times \Sigma$, $\delta(q,a)$ is a conjunction of formulae of the form $\forall x.\,(q'(x) \vee \theta(x))$ where $q' \in \mathrm{Q}_i$ and $\theta(x)$ only refers to states in lower components Q_j with $j < i$ (*first-order universal requirement*).

HFTA can be easily translated into equivalent MCL sentences.

▶ **Theorem 6.1.** *Given an* HFTA $\mathcal{A}$ *over* 2^{AP}*, one can construct in polynomial time an* MCL *sentence* $\varphi_{\mathcal{A}}$ *over* AP *such that* $\mathsf{L}(\varphi_{\mathcal{A}}) = \mathsf{L}(\mathcal{A})$.

Chain Projection. Like HGTA, the tree-languages accepted by HFTA are closed under Boolean operations. Thus, in the translation of MCL formulae into equivalent parity HFTA, the only non-trivial part concerns the treatment of MCL existential quantification. For this purpose, we define an operation on tree languages that captures the semantics of MCL existential quantification. Let L be a tree language over 2^{AP} and $p \in \mathrm{AP}$. The *chain projection of* L *over* p, denoted by $\exists^{\mathsf{c}} p.\mathsf{L}$, is the language consisting of the Kripke trees (T, Lab) over $\mathrm{AP} \setminus \{p\}$ such that there is an infinite path π of T from the root and a Kripke tree $(\mathrm{T}, Lab') \in \mathsf{L}$ so that: $Lab'(w) = Lab(w)$, for each $w \in \mathrm{T} \setminus \pi$, and $Lab'(w) \setminus \{p\} = Lab(w)$, otherwise.

We show that HFTA are effectively closed under chain projection, i.e., for each HFTA $\mathcal{A}$ over 2^{AP} and $p \in \mathrm{AP}$, one can construct an HFTA accepting $\exists^{\mathsf{C}} p.\mathsf{L}(\mathcal{A})$. The proof is divided in two steps. In the first step, we define a subclass of HFTA, called HFTA that *are nondeterministic in one path* (see Definition 6.3), whose closure under chain projection can be easily established (see Proposition 6.4). Then, in the second step, we show that an HFTA can be converted into an equivalent HFTA that is nondeterministic in one path.

We now introduce this subclass of automata. By exploiting the known notion of *basic formula* [82, 12], we first define a fragment of the one-step language $\mathrm{FOE}_1^+(\mathrm{Q})$ for a given non-empty set Q. A Q-*type* is a (possibly empty) set $\mathrm{A} \subseteq \mathrm{Q}$. It defines the first-order constraint $\mathtt{t}(\mathrm{A})(x) \triangleq \bigwedge_{q \in \mathrm{A}} q(x)$. Note that $\mathtt{t}(\mathrm{A})(x)$ is $\top$ if A is empty. Let $\mathtt{T}_\exists$ and $\mathtt{T}_\forall$ be two sets of Q-type. The *basic formula for the pair* $(\mathtt{T}_\exists, \mathtt{T}_\forall)$, denoted $\theta^{=}(\mathtt{T}_\exists, \mathtt{T}_\forall)$, is the $\mathrm{FOE}_1^+(\mathrm{Q})$ sentence defined as follows, where $\mathtt{T}_\exists = \{\mathrm{A}_1, \ldots, \mathrm{A}_n\}$ for some $n \geq 0$ and for variables $z_1, \ldots, z_k$, $\mathtt{diff}(z_1, \ldots, z_k) \triangleq \bigwedge_{i \neq j} z_i \neq z_j$:

$$\exists x_1 \ldots \exists x_n. \Big(\mathtt{diff}(x_1, \ldots, x_n) \wedge \bigwedge_{i=1}^{n} \mathtt{t}(\mathrm{A}_i)(x_i) \wedge \forall y. \big(\mathtt{diff}(x_1, \ldots, x_n, y) \rightarrow \bigvee_{\mathrm{A} \in \mathtt{T}_\forall} \mathtt{t}(\mathrm{A})(y)\big)\Big).$$

Intuitively, $\theta^{=}(\mathtt{T}_\exists, \mathtt{T}_\forall)$ asserts that there are n-distinct elements $s_1, \ldots, s_n$ of the given domain S such that each s_i satisfies the Q-type A_i of the existential part $\mathtt{T}_\exists$, and every other element of the domain satisfies some Q-type in the universal part $\mathtt{T}_\forall$.

▶ **Definition 6.2.** *Let* $\mathrm{Q}' \subseteq \mathrm{Q}$ *with* $\mathrm{Q}' \neq \emptyset$. *A basic formula* $\theta^{=}(\mathtt{T}_\exists, \mathtt{T}_\forall)$ *is* Q'-functional in one direction *if there exists* $\mathrm{A} \in \mathtt{T}_\exists$ *such that* A *is a singleton consisting of an element in* Q', *and for each* $\mathrm{B} \in (\mathtt{T}_\exists \setminus \{\mathrm{A}\}) \cup \mathtt{T}_\forall$, B *does not contain elements in* Q'. *A first-order* Q-*constraint is* Q'-functional in one direction *if it is the disjunction of basic formulae which are* Q'-*functional in one direction.*

Intuitively, when the local behaviour of an HFTA $\mathcal{A}$ at the current input node w is driven by a constraint θ that is Q'-functional in one direction, then there is a child w' of w such that exactly one copy of $\mathcal{A}$ is sent to w'. Moreover, the state of this copy is in Q' and the states of the copies sent to the children of w distinct from w' are in $\mathrm{Q} \setminus \mathrm{Q}'$.

▶ **Definition 6.3.** *An* HFTA $\mathcal{A} = \langle \Sigma, \mathrm{Q}, \delta, q_I, \mathsf{H}, \mathsf{H}_\exists, \Omega \rangle$ *is* nondeterministic in one path *if the initial state* q_I *belongs to some existential component* Q_ℓ *of* $\mathcal{A}$ *and the following hold:*

1. *for each* $q \in \mathrm{Q}_\ell$ *and* $a \in \Sigma$, $\delta(q, a)$ *is* Q_ℓ-*functional in one direction;*
2. *for each* $\mathcal{T} \in \mathsf{L}(\mathcal{A})$ *and for each infinite path* π *of* $\mathcal{T}$ *from the root, there is an accepting run* $r = (\mathrm{T}_r, Lab_r)$ *of* $\mathcal{A}$ *over* $\mathcal{T}$ *s.t. for each input node* $w \in \pi$, *there is exactly one node* y *of* r *reading* w, *i.e., such that* $Lab_r(y) = (q, w)$ *for some state* q; *moreover,* $q \in \mathrm{Q}_\ell$.

Let $\Sigma = 2^{\mathrm{AP}}$, $p \in \mathrm{AP}$, $\mathcal{A} = \langle \Sigma, \mathrm{Q}, \delta, q_I, \mathsf{H}, \mathsf{H}_\exists, \Omega \rangle$ be an HFTA that is nondeterministic in one path, and Q_ℓ be the existential component such that $q_I \in \mathrm{Q}_\ell$. We consider the HFTA $\exists^{\mathsf{C}} p.\mathcal{A} = \langle 2^{\mathrm{AP} \setminus \{p\}}, \mathrm{Q}, \delta', q_I, \mathsf{H}, \mathsf{H}_\exists, \Omega \rangle$, where the transition function δ' is defined as follows for all $q \in \mathrm{Q}$ and $a \in 2^{\mathrm{AP} \setminus \{p\}}$: $\delta'(q, a) = \delta(q, a)$ if $q \notin \mathrm{Q}_\ell$, and $\delta'(q, a) = \delta(q, a) \vee \delta(q, a \cup \{p\})$ otherwise. Hence, on all the states which are not in the existential component Q_ℓ, $\exists^{\mathsf{C}} p.\mathcal{A}$ behaves as $\mathcal{A}$. On the states in Q_ℓ, the projection automaton guesses whether in the simulated run of $\mathcal{A}$, proposition p marks the current input node or not, and proceeds according to the guess and the transition function of $\mathcal{A}$. By Definition 6.3, we easily obtain the following result.

▶ **Proposition 6.4.** *Let* $\mathcal{A}$ *be an* HFTA *over* 2^{AP} *that is nondeterministic in one path and* $p \in \mathrm{AP}$. *Then,* $\mathsf{L}(\exists^{\mathsf{C}} p.\mathcal{A}) = \exists^{\mathsf{C}} p.\mathsf{L}(\mathcal{A})$.

From HFTA to HFTA that are nondeterministic in one path. We now show that HFTA can be effectively translated into equivalent HFTA that are nondeterministic in one path. We first establish a preliminary result on the one-step logic $\mathrm{FOE}_1^+(\mathrm{Q})$ for a given non-empty set Q.

▶ **Definition 6.5.** *Let θ be a first-order* Q*-constraint and θ_s be a first-order $(\mathrm{Q} \cup 2^{\mathrm{Q}})$-constraint which is 2^{Q}-functional in one direction. We say that θ_s simulates θ if the following hold:*

- *for each minimal model (S, I) of θ and for each $s \in \mathrm{S}$, $(\mathrm{S}, \mathrm{I}[s \to \{\mathrm{I}(s)\}])$ is a model of θ_s;*
- *for each minimal model (S, I) of θ_s, let $s \in \mathrm{S}$ be the unique element in S such that $\mathrm{I}(s)$ is of the form $\{\mathrm{Q}'\}$ for some $\mathrm{Q}' \in 2^{\mathrm{Q}}$. Then, the pair $(\mathrm{S}, \mathrm{I}[s \to \mathrm{Q}'])$ is a model of θ;*

where the mappings $\mathrm{I}[s \to \{\mathrm{I}(s)\}]$ and $\mathrm{I}[s \to \mathrm{Q}']$ are defined in the obvious way.

Since each first-order Q-constraint is effectively equivalent to a disjunction of basic formulae [12], we easily obtain the following result.

▶ **Proposition 6.6.** *Let θ be a first-order* Q*-constraint. Then, one can construct a first-order $(\mathrm{Q} \cup 2^{\mathrm{Q}})$-constraint θ_s which is 2^{Q}-functional in one direction and simulates θ.*

Fix an HFTA $\mathcal{A} = \langle \Sigma, \mathrm{Q}, \delta, q_I, \mathsf{H}, \mathsf{H}_\exists, \Omega \rangle$ with $\mathsf{H} = \langle \mathrm{Q}_1, \ldots, \mathrm{Q}_n \rangle$. We construct in two stages an equivalent HFTA $\mathtt{Sim}(\mathcal{A})$ that is nondeterministic in one path. First, by a kind of powerset construction, we construct an automaton $\mathcal{A}_{\mathtt{PATH}}$ that is nondeterministic in one path but the acceptance condition of the existential component $Pow_{\mathcal{A}}$ containing the initial state is not a Büchi condition but an ω-regular set over the infinite sequences on $Pow_{\mathcal{A}}$. In the second stage of the construction, we show how the ω-regular condition can be converted into a standard Büchi condition by equipping the "macro" states in $Pow_{\mathcal{A}}$ with additional information. Intuitively, given an input tree (T, Lab) accepted by $\mathcal{A}$, the automaton $\mathcal{A}_{\mathtt{PATH}}$ simulates the behaviour of $\mathcal{A}$ along an accepting run r over (T, Lab) by guessing an infinite path π of the input tree from the root and proceeding as follows:

- in the input nodes $w \notin \pi$, $\mathcal{A}_{\mathtt{PATH}}$ simply simulates the behaviour of $\mathcal{A}$ along r;
- in the input nodes $w \in \pi$, $\mathcal{A}_{\mathtt{PATH}}$ keeps track in its "macro" state (a state in the existential component $Pow_{\mathcal{A}}$) of the states of $\mathcal{A}$ associated with the copies of $\mathcal{A}$ that read w along r. Thus, in the run of $\mathcal{A}_{\mathtt{PATH}}$, there is a unique infinite path ν from the root associated with the guessed input path π, and ν "collects" the set of parallel paths ν_r of the simulated run of $\mathcal{A}$ associated with the input path π. In order to check the acceptance condition on the individual parallel paths ν_r, an infinite sequence of "macro" states ρ must allow to distinguish the individual infinite paths over Q grouped by ρ. Thus, like in [82], a "macro" state associated with an input node w is a set of pairs (q_p, q): the pair (q_p, q) represents a copy of $\mathcal{A}$ in state q at node w along the simulated run r which has been generated by a copy of $\mathcal{A}$ in state q_p reading the parent node of w in the input tree.

Formally, we denote by $Pow_{\mathcal{A}}$ the subset of $2^{\mathrm{Q} \times \mathrm{Q}}$ consisting of the sets of pairs (q, q') of $\mathcal{A}$-states such that the order of q' is equal or lower than the order of q (*order requirement*). A $Pow_{\mathcal{A}}$*-path* ν is an infinite word $\nu = \mathrm{P}_0\mathrm{P}_1 \ldots$ over $Pow_{\mathcal{A}}$ such that the following conditions are fulfilled: (i) $\mathrm{P}_0 = \{(q_I, q_I)\}$ (*initialisation*), and (ii) for all $i \geq 0$ and $(q_i, q_{i+1}) \in \mathrm{P}_{i+1}$, there is an element of P_i of the form (q_{i-1}, q_i) (*consecution*). An $\mathcal{A}$*-path* of ν is a maximal (possibly finite) non-empty sequence $q_0 q_1 \ldots$ of $\mathcal{A}$-states such that $(q_{i-1}, q_i) \in \mathrm{P}_i$ for all $1 \leq i < |\nu|$. The $Pow_{\mathcal{A}}$-path ν is $\mathcal{A}$-accepting if all infinite $\mathcal{A}$-paths of ν satisfy the parity condition Ω of $\mathcal{A}$. The automaton $\mathcal{A}_{\mathtt{PATH}}$ is then given by $\mathcal{A}_{\mathtt{PATH}} = \langle \Sigma, \mathrm{Q} \cup Pow_{\mathcal{A}}, \delta_{\mathtt{PATH}}, \{(q_I, q_I)\}, \mathsf{H}_{\mathtt{PATH}}, \mathsf{H}_\exists \cup \{Pow_{\mathcal{A}}\}, \Omega \rangle$ where $\mathsf{H}_{\mathtt{PATH}} = \langle \mathrm{Q}_1, \ldots, \mathrm{Q}_n, Pow_{\mathcal{A}} \rangle$ (the existential component $Pow_{\mathcal{A}}$ has highest order) and $\delta_{\mathtt{PATH}}$ is defined as follows:

- for all $q \in \mathrm{Q}$ and $a \in \Sigma$, $\delta_{\mathtt{PATH}}(q, a) = \delta(q, a)$;

- for all $\mathrm{P} \in \mathit{Pow}_{\mathcal{A}}$ and $a \in \Sigma$, if P is empty, then $\delta_{\mathtt{PATH}}(\mathrm{P}, a) = \exists x.\, \mathrm{P}(x)$; otherwise, let us consider the first-order $(\mathrm{Q} \times \mathrm{Q})$-constraint θ given by $\bigwedge_{(q_p,q)\in \mathrm{P}} \delta_q(q, a)$, where $\delta_q(q, a)$ is obtained from $\delta(q, a)$ by replacing each predicate $q'(y)$ occurring in $\delta(q, a)$ with $(q, q')(y)$. By Proposition 6.6, one can construct a first-order $((\mathrm{Q} \times \mathrm{Q}) \cup \mathit{Pow}_{\mathcal{A}})$-constraint θ_s which is $\mathit{Pow}_{\mathcal{A}}$-functional in one direction and simulates θ. Then, $\delta_{\mathtt{PATH}}(\mathrm{P}, a)$ is obtained from θ_s by replacing each predicate $(q, q')(y)$ occurring in θ_s associated with an element of $\mathrm{Q} \times \mathrm{Q}$ with q'. Note that $\delta_{\mathtt{PATH}}(\mathrm{P}, a)$ satisfies the first-order existential requirement and is $\mathit{Pow}_{\mathcal{A}}$-functional in one direction.

Note that in the definition of $\mathcal{A}_{\mathtt{PATH}}$, no acceptance condition is defined for the macro states in $\mathit{Pow}_{\mathcal{A}}$ (the parity condition Ω inherited by $\mathcal{A}$ is defined only on the states in Q). By construction and Proposition 6.6, for each run r of $\mathcal{A}_{\mathtt{PATH}}$ over an input $(\mathrm{T}, \mathit{Lab})$ and every infinite path π of r starting at the root, either π is associated with a $\mathit{Pow}_{\mathcal{A}}$-path ν (in this case, we say that π is accepting if ν is accepting) or π gets trapped into some non-transient component of $\mathcal{A}$ (in this case, acceptance of π is determined by the parity condition Ω). We denote by $\mathsf{L}(\mathcal{A}_{\mathtt{PATH}})$ the set of input trees $(\mathrm{T}, \mathit{Lab})$ such that there is a run of $\mathcal{A}_{\mathtt{PATH}}$ over $(\mathrm{T}, \mathit{Lab})$ whose infinite paths starting at the root are all accepting. By construction and Proposition 6.6, we easily deduce the following crucial result.

▶ **Lemma 6.7.** *$\mathcal{A}_{\mathtt{PATH}}$ is nondeterministic in one path and $\mathsf{L}(\mathcal{A}_{\mathtt{PATH}}) = \mathsf{L}(\mathcal{A})$.*

Construction of the Automaton $\mathtt{Sim}(\mathcal{A})$. Let F_{B} (resp., $\mathrm{F}_{\mathsf{coB}}$) be the set of states in the existential (resp., universal) components of $\mathcal{A}$ having even (resp., odd) color. Fix a $\mathit{Pow}_{\mathcal{A}}$-path ν. By the order requirement, each infinite $\mathcal{A}$-path of ν gets trapped into an existential or universal component of $\mathcal{A}$. Thus, by the hesitant acceptance requirement of HFTA, the $\mathit{Pow}_{\mathcal{A}}$-path ν is $\mathcal{A}$-accepting if and only if for each infinite $\mathcal{A}$-*path* π of ν, the following holds: if π gets trapped into an existential component, then π visits *infinitely* many times some state in F_{B} (*Büchi condition*); otherwise (i.e., π gets trapped into an universal component), π visits *finitely* many times all the states in $\mathrm{F}_{\mathsf{coB}}$ (*coBüchi condition*).

It is known that coBüchi alternating word automata (AWA) over infinite words can be converted in quadratic time into equivalent Büchi AWA by means of the so called *ranking construction* [46]. We adapt the ranking construction and the Miyano-Hayashi construction [53] (for converting a Büchi AWA into an equivalent Büchi NWA) for providing a characterisation of acceptance of $\mathit{Pow}_{\mathcal{A}}$-paths ν by a classical Büchi condition on an extension of ν obtained by adding to each macro-state visited by ν additional finite-state information. Hence, we obtain the following result.

▶ **Theorem 6.8.** *For the given* HFTA $\mathcal{A}$*, one can construct an* HFTA $\mathtt{Sim}(\mathcal{A})$ *that is nondeterministic in one path and such that* $\mathsf{L}(\mathtt{Sim}(\mathcal{A})) = \mathsf{L}(\mathcal{A})$.

By Theorem 6.8 and Proposition 6.4, we obtain the following result.

▶ **Corollary 6.9.** *The class of* HFTA *is effectively closed under chain projection.*

An HFTA with transition function δ is in *normal form* if over existential (resp., universal) components Q_ℓ, $\delta(q, a)$ (resp., the dual of $\delta(q, a)$) is Q_ℓ-functional in one direction for all $q \in \mathrm{Q}_\ell$ and $a \in \Sigma$. Since the constructions for the Boolean language operations and the construction for the closure under chain projection (Theorem 6.8 and Proposition 6.4) preserve the normal form, we deduce the following result.

▶ **Theorem 6.10.** *Given an* MCL *sentence φ, one can construct an* HFTA $\mathcal{A}_\varphi$ *in normal form such that* $\mathsf{L}(\mathcal{A}_\varphi) = \mathsf{L}(\varphi)$.

We exploit the normal form for showing that CDL (or, equivalently, the class of symmetric HGTA) provides a characterisation of the bisimulation-fragment of MCL. It is known [82, 12] that for each FTA $\mathcal{A}$, one can construct a symmetric FTA $\mathcal{A}_S$ such that if $\mathsf{L}(\mathcal{A})$ is bisimulation-closed, then $\mathcal{A}$ and $\mathcal{A}_S$ accept the same tree-language. By adapting the construction given in [82, 12], we can show that a similar result holds for HFTA in normal form versus symmetric HGTA. Hence, by Theorems 5.8 and 5.9 and Theorem 6.10, we deduce the following result.

▶ **Theorem 6.11.** *The bisimulation-invariant fragment of* MCL*,* CDL*, and the class of symmetric* HGTA *are expressively equivalent in a constructive way.*

7 Conclusion

This work provides automata-theoretic characterisations of branching-time temporal logics, mainly focusing on CTL* and CDL, the latter being a syntactic variant of the already known ECTL*. Specifically, we prove the equivalence between the symmetric variant of classic ranked Hesitant Tree Automata (HTA) and both CDL and the bisimulation-invariant fragment of Monadic Chain Logic (MCL). The full MCL, instead, is proved equivalent to a first-order variant of HTAs. In addition, we close a longstanding gap in the expressiveness landscape of branching-time logics, by providing an automata-theoretic characterisation of CTL*. This is obtained via a generalisation to tree-languages of the notion of counter-freeness, originally introduced in the context of word languages. The generalisation essentially decomposes an HTA into a number of counter-free word automata, one for each level of the state decomposition of the HTA. This decomposition, however, works correctly only when the HTA satisfies the additional property of mutual-exclusion. The property requires that different sets of automaton states, active at the same time on a given node of the input tree, must accept different subtrees. Both mutual-exclusion and counter-freeness seem to be essential to capture a meaningful notion of counter-freeness for tree automata. Together these results bring the expressiveness landscape for branching-time temporal logics to almost the same level as their linear-time counterparts.

There are few open questions remaining. In particular, while Theorem 6.11 establishes the equivalence between the bisimulation invariant fragment of MCL and CDL, the precise relationship between CCDL (hence, ECTL*) and full MCL still remains unsettled. In addition, techniques similar to those used in this work may also be applicable to obtain a characterisation of Monadic Tree Logic (MTL), a fragment of MSO where quantified variables range over subtrees [3], and of Substructure Temporal Logic (STL), a temporal logic where one can implicitly predicate over substructure/subtrees [4, 5]. The restriction that variables range over trees, indeed, seem to be tightly connected with the notion of counter-freeness. The difficulty in this case is that counter-free HTAs would not suffice, since both MTL and STL are strictly more expressive than CTL*, and a meaningful definition of decomposition into word automata of a non-hesitant tree automaton is not immediately obvious.

References

1 A. Arnold and D. Niwiński. Fixed Point Characterization of Weak Monadic Logic Definable Sets of Trees. In *Tree Automata and Languages*, pages 159–188. North-Holland, 1992.

2 C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

3 M. Benerecetti, L. Bozzelli, F. Mogavero, and A. Peron. Quantifying over Trees in Monadic Second-Order Logic. In *LICS'23*, pages 1–13. IEEECS, 2023.

4 M. Benerecetti, F. Mogavero, and A. Murano. Substructure Temporal Logic. In *LICS'13*, pages 368–377. IEEECS, 2013.
5 M. Benerecetti, F. Mogavero, and A. Murano. Reasoning About Substructures and Games. *TOCL*, 16(3):25:1–46, 2015.
6 M. Bojańczyk. The Finite Graph Problem for Two-Way Alternating Automata. *TCS*, 3(298):511–528, 2003.
7 U. Boker and Y. Shaulian. Automaton-Based Criteria for Membership in CTL. In *LICS'18*, pages 155–164. ACM, 2018.
8 J.R. Büchi. Weak Second-Order Arithmetic and Finite Automata. *MLQ*, 6(1-6):66–92, 1960.
9 J.R. Büchi. On a Decision Method in Restricted Second-Order Arithmetic. In *ICLMPS'62*, pages 1–11. Stanford University Press, 1962.
10 J.R. Büchi. On a Decision Method in Restricted Second Order Arithmetic. In *Studies in Logic and the Foundations of Mathematics*, volume 44, pages 1–11. Elsevier, 1966.
11 F. Carreiro, A. Facchini, Y. Venema, and F. Zanasi. Weak MSO: Automata and Expressiveness Modulo Bisimilarity. In *CSL'14 & LICS'14*, pages 27:1–27. ACM, 2014.
12 F. Carreiro, A. Facchini, Y. Venema, and F. Zanasi. The Power of the Weak. *TOCL*, 21(2):15:1–47, 2020.
13 F. Carreiro, A. Facchini, Y. Venema, and F. Zanasi. Model Theory of Monadic Predicate Logic with the Infinity Quantifier. *AML*, 61(3-4):465–502, 2022.
14 Y. Choueka. Theories of Automata on ω-Tapes: A Simplified Approach. *JCSS*, 8(2):117–141, 1974.
15 A. Church. Logic, Arithmetics, and Automata. In *ICM'62*, pages 23–35, 1963.
16 E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach. In *POPL'83*, pages 117–126. ACM, 1983.
17 E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *TOPLAS*, 8(2):244–263, 1986.
18 E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 2002.
19 V. Diekert and P. Gastin. First-Order Definable Languages. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 261–306. Amsterdam University Press, 2008.
20 C.C. Elgot. Decision Problems of Finite Automata Design and Related Arithmetics. *TAMS*, 98:21–51, 1961.
21 E.A. Emerson and E.M. Clarke. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *LP'81*, LNCS 131, pages 52–71. Springer, 1982.
22 E.A. Emerson and E.M. Clarke. Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons. *SCP*, 2(3):241–266, 1982.
23 E.A. Emerson and J.Y. Halpern. "Sometimes" and "Not Never" Revisited: On Branching Versus Linear Time. In *POPL'83*, pages 127–140. ACM, 1983.
24 E.A. Emerson and J.Y. Halpern. Decision Procedures and Expressiveness in the Temporal Logic of Branching Time. *JCSS*, 30(1):1–24, 1985.
25 E.A. Emerson and J.Y. Halpern. "Sometimes" and "Not Never" Revisited: On Branching Versus Linear Time. *JACM*, 33(1):151–178, 1986.
26 E.A. Emerson and C.S. Jutla. Tree Automata, muCalculus, and Determinacy. In *FOCS'91*, pages 368–377. IEEECS, 1991.
27 E.A. Emerson, C.S. Jutla, and A.P. Sistla. On Model Checking for the muCalculus and its Fragments. *TCS*, 258(1-2):491–522, 2001.
28 A. Facchini, Y. Venema, and F. Zanasi. A Characterization Theorem for the Alternation-Free Fragment of the Modal μ-Calculus. In *LICS'13*, pages 478–487. IEEECS, 2013.
29 K. Fine. In So Many Possible Worlds. *NDJFL*, 13:516–520, 1972.
30 M.J. Fischer and R.E. Ladner. Propositional Dynamic Logic of Regular Programs. *JCSS*, 18(2):194–211, 1979.

31 D.M. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the Temporal Basis of Fairness. In *POPL'80*, pages 163–173. ACM, 1980.

32 G. De Giacomo and M.Y. Vardi. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI'13*, pages 854–860. IJCAI' & AAAI Press, 2013.

33 Y. Gurevich and S. Shelah. The Decision Problem for Branching Time Logic. *JSL*, 50(3):668–681, 1985.

34 T. Hafer and W. Thomas. Computation Tree Logic CTL* and Path Quantifiers in the Monadic Theory of the Binary Tree. In *ICALP'87*, LNCS 267, pages 269–279. Springer, 1987.

35 D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.

36 D. Janin. *A Contribution to Formal Methods: Games, Logic and Automata*. Habilitation thesis, Université Bordeaux I, Bordeaux, France, 2005.

37 D. Janin and G. Lenzi. On the Relationship Between Monadic and Weak Monadic Second Order Logic on Arbitrary Trees, with Applications to the mu-Calculus. *FI*, 61(3-4):247–265, 2004.

38 D. Janin and I. Walukiewicz. On the Expressive Completeness of the Propositional mu-Calculus with Respect to Monadic Second Order Logic. In *CONCUR'96*, LNCS 1119, pages 263–277. Springer, 1996.

39 H.W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, CA, USA, 1968.

40 R.M. Keller. Formal Verification of Parallel Programs. *CACM*, 19(7):371–384, 1976.

41 S.C. Kleene. Representation of Events in Nerve Nets and Finite Automata. In *Automata Studies*, pages 3–42. Princeton University Press, 1956.

42 D. Kozen. Results on the Propositional muCalculus. *TCS*, 27(3):333–354, 1983.

43 S.A. Kripke. Semantical Considerations on Modal Logic. *APF*, 16:83–94, 1963.

44 O. Kupferman, U. Sattler, and M.Y. Vardi. The Complexity of the Graded muCalculus. In *CADE'02*, LNCS 2392, pages 423–437. Springer, 2002.

45 O. Kupferman and M.Y. Vardi. Freedom, Weakness, and Determinism: From Linear-Time to Branching-Time. In *LICS'98*, pages 81–92. IEEECS, 1998.

46 O. Kupferman and M.Y. Vardi. Weak Alternating Automata are not That Weak. *TOCL*, 2(3):408–429, 2001.

47 O. Kupferman, M.Y. Vardi, and P. Wolper. An Automata Theoretic Approach to Branching-Time Model Checking. *JACM*, 47(2):312–360, 2000.

48 R.E. Ladner. Application of Model Theoretic Games to Discrete Linear Orders and Finite Automata. *IC*, 33(4):281–303, 1977.

49 Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems - Specification*. Springer, 1992.

50 Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems - Safety*. Springer, 1995.

51 R. McNaughton. Testing and Generating Infinite Sequences by a Finite Automaton. *IC*, 9(5):521–530, 1966.

52 R. McNaughton and S. Papert. *Counter-Free Automata*. MIT Press, 1971.

53 S. Miyano and T. Hayashi. Alternating Finite Automata on ω-Words. *TCS*, 32(3):321–330, 1984.

54 F. Moller and A.M. Rabinovich. On the Expressive Power of CTL*. In *LICS'99*, pages 360–368. IEEECS, 1999.

55 F. Moller and A.M. Rabinovich. Counting on CTL*: On the Expressive Power of Monadic Path Logic. *IC*, 184(1):147–159, 2003.

56 E.F. Moore. Gedanken-Experiments on Sequential Machines. In *Automata Studies*, pages 129–154. Princeton University Press, 1956.

57 A. Nerode. Linear Automaton Transformations. *PAMS*, 9(4):541–544, 195.

58 D. Perrin. Recent Results on Automata and Infinite Words. In *MFCS'84*, LNCS 176, pages 134–148. Springer, 1984.

59 D. Perrin and J. Pin. First-Order Logic and Star-Free Sets. *JCSS*, 32(3):393–406, 1986.

60 D. Perrin and J. Pin. *Infinite Words*. Pure and Applied Mathematics. Elsevier, 2004.
61 A. Pnueli. The Temporal Logic of Programs. In *FOCS'77*, pages 46–57. IEEECS, 1977.
62 A. Pnueli. The Temporal Semantics of Concurrent Programs. *TCS*, 13:45–60, 1981.
63 A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module. In *POPL'89*, pages 179–190. ACM, 1989.
64 M.O. Rabin. Decidability of Second-Order Theories and Automata on Infinite Trees. *TAMS*, 141:1–35, 1969.
65 M.O. Rabin. Weakly Definable Relations and Special Automata. In *Studies in Logic and the Foundations of Mathematics*, volume 59, pages 1–23. Elsevier, 1970.
66 M.O. Rabin and D.S. Scott. Finite Automata and their Decision Problems. *IBMJRD*, 3:115–125, 1959.
67 A. Rabinovich. A Proof of Kamp's Theorem. In *CSL'12*, LIPIcs 16, pages 516–527. Leibniz-Zentrum fuer Informatik, 2012.
68 A. Rabinovich. A Proof of Kamp's Theorem. *LMCS*, 10(1):1–16, 2014.
69 R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1992.
70 M.P. Schützenberger. On Finite Monoids Having Only Trivial Subgroups. *IC*, 8(2):190–194, 1965.
71 W. Thomas. Star-Free Regular Sets of ω-Sequences. *IC*, 42(2):148–156, 1979.
72 W. Thomas. A Combinatorial Approach to the Theory of ω-Automata. *IC*, 48(3):261–283, 1981.
73 W. Thomas. Logical Aspects in the Study of Tree Languages. In *CAAP'84*, pages 31–50. CUP, 1984.
74 W. Thomas. On Chain Logic, Path Logic, and First-Order Logic over Infinite Trees. In *LICS'87*, pages 245–256. IEEECS, 1987.
75 W. Thomas. Automata on Infinite Objects. In *Handbook of Theoretical Computer Science (vol. B)*, pages 133–191. MIT Press, 1990.
76 B.A. Trakhtenbrot. Finite Automata and the Logic of One-Place Predicates. *AMST*, 59:23–55, 1966.
77 J. van Benthem. *Modal Correspondence Theory*. PhD thesis, University of Amsterdam, Amsterdam, Netherlands, 1977.
78 M.Y. Vardi. Reasoning about The Past with Two-Way Automata. In *ICALP'98*, LNCS 1443, pages 628–641. Springer, 1998.
79 M.Y. Vardi and L.J. Stockmeyer. Improved Upper and Lower Bounds for Modal Logics of Programs: Preliminary Report. In *STOC'85*, pages 240–251. ACM, 1985.
80 M.Y. Vardi and P. Wolper. Yet Another Process Logic. In *LP'83*, LNCS 164, pages 501–512. Springer, 1984.
81 M.Y. Vardi and P. Wolper. Automata-Theoretic Techniques for Modal Logics of Programs. *JCSS*, 32(2):183–221, 1986.
82 I. Walukiewicz. Monadic Second Order Logic on Tree-Like Structures. *TCS*, 275(1-2):311–346, 2002.
83 A. Weinert and M. Zimmermann. Visibly Linear Dynamic Logic. *TCS*, 747:100–117, 2018.
84 P. Wolper. Temporal Logic Can Be More Expressive. *IC*, 56(1-2):72–99, 1983.

The Complexity of Computing in Continuous Time: Space Complexity Is Precision

Manon Blanc
Institut Polytechnique de Paris, Ecole Polytechnique, LIX, 91128 Palaiseau Cedex, France
Université Paris-Saclay, LISN, 91190 Gif-sur-Yvette, France

Olivier Bournez
Institut Polytechnique de Paris, Ecole Polytechnique, LIX, 91128 Palaiseau Cedex, France

Abstract

Models of computations over the integers are equivalent from a computability and complexity theory point of view by the (effective) Church-Turing thesis. It is not possible to unify discrete-time models over the reals. The situation is unclear but simpler for continuous-time models, as there is a unifying mathematical model, provided by ordinary differential equations (ODEs). Each model corresponds to a particular class of ODEs. For example, the General Purpose Analog Computer model of Claude Shannon, introduced as a mathematical model of analogue machines (Differential Analyzers), is known to correspond to polynomial ODEs. However, the question of a robust complexity theory for such models and its relations to classical (discrete) computation theory is an old problem. There was some recent significant progress: it has been proved that (classical) time complexity corresponds to the length of the involved curves, i.e. to the length of the solutions of the corresponding polynomial ODEs. The question of whether there is a simple and robust way to measure space complexity remains. We argue that space complexity corresponds to precision and conversely.

Concretely, we propose and prove an algebraic characterisation of **FPSPACE**, using *continuous* ODEs. Recent papers proposed algebraic characterisations of polynomial-time and polynomial-space complexity classes over the reals, but with a discrete-time: those algebras rely on discrete ODE schemes. Here, we use classical (continuous) ODEs, with the classic definition of derivation and hence with the more natural context of continuous-time associated with ODEs. We characterise both the case of polynomial space functions over the integers and the reals. This is done by proving two inclusions. The first is obtained using some original polynomial space method for solving ODEs. For the other, we prove that Turing machines, with a proper representation of real numbers, can be simulated by continuous ODEs and not just discrete ODEs. A major consequence is that the associated space complexity is provably related to the numerical stability of involved schemas and the associated required precision. We obtain that a problem can be solved in polynomial space if and only if it can be simulated by some numerically stable ODE, using a polynomial precision.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Computability; Theory of computation → Complexity classes; Mathematics of computing → Ordinary differential equations

Keywords and phrases Models of computation, Ordinary differential equations, Real computations, Analog computations, Complexity theory, Implicit complexity, Recursion scheme

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.129

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding This work has been partially supported by ANR Project ∂IFFERENCE and Labex Digicosme.

1 Introduction

Recently, there has been a renewed interest in models of computations over the reals and their associated complexity classes. The fact that these models appear in complexity issues of deep learning models (a.k.a. neural networks) partially explains it. For example, various

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 129; pp. 129:1–129:22


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

problems, such as the training of fully connected neural networks, have been proved to be a $\exists\mathbb{R}$-complete problem [2, 3]. Complexity classes like **FIXP** were introduced to discuss the complexity of continuous functions' fixed points in various contexts, such as game theory [32]. These classes and statements are related to discrete-time models of computation over the reals.

For discrete-time models of computations over the reals, the most famous approaches are computable analysis, based on the Turing machine model in [55] and [59] and algebraic models such as the Blum Shub Smale (BSS) model of computation [9, 8]. The class $\exists\mathbb{R}$ corresponds to the (constant-free, equivalently uniform) non-deterministic time of the BSS model of computation. Numerous decision problems were proved recently to be in this class. Both models were tailored for very different applications and it is well-known we cannot unify existing models with the equivalent of a Church-Turing thesis. For example, computable functions in a computable analysis model need to be continuous, while the BSS model intends to consider functions and problems over the polynomials that are not. It is also explained by the fact that some models have not been introduced with the idea of corresponding to actual physical machines but also to discuss abstract complexity (lower and upper bounds) for associated problems. Notice that some characterisations of complexity classes corresponding to **PSPACE** have been obtained [10, 11] in the BSS model.

Among models of computation over the reals, we can also distinguish continuous-time models. This includes models of old, first-ever-built computers, such as the Differential Analysers [57]. A famous mathematical model of such machines is the General Purpose Analog Computer model of Claude Shannon [53]. It covers many historical machines and today's analogue devices [56, 58] too. It also includes various recent approaches and models from deep learning such as Neural ODEs [26, 44] with many variants. In the context of continuous-time, the situation is clearer than with discrete-time models, as there is a unifying way to describe these models, provided by Ordinary Differential Equations (ODEs). Each model corresponds to a particular class of ODEs. For example, the GPAC corresponds to polynomial ODEs [37], and Neural ODEs are made by selecting the best solution among a parameterised class of ODEs: see, e.g. [44].

Even if particular classes of ODEs can describe such models, defining a robust and well-defined computation theory for continuous-time computations is not an easy problem: see [21] for the most recent survey. In short, the problem with time complexity is that considering the time variable as a measure of time is not robust: a curve can always be re-parameterised using a change of variable. The problem with space complexity is similar: reparameterisation corresponds to a change of time variable, but also of space-variable, introducing space and time contractions: See e.g. [21, 17]. Furthermore, many problems for simple dynamical systems are known to be undecidable, hence forbid **PSPACE**-completeness: see [38] and [39].

There was a recent breakthrough in [19, 17], where the authors relate time with the length of the solution curve of an ODE. Polynomial ODEs and their projections are known to cover a very wide class of functions, including all common functions or functions that can be built from them [35]. As the length of a curve is preserved under reparameterization, considering the length solves the issue of a possible change of variable. The authors prove that for polynomial ODEs, this is polynomially related to the time required to solve an ODE, hence providing a robust notion of time for ODEs. These statements and underlying constructions, which allow the simulation of Turing machines, led to solving several open problems: the existence of a universal ODE [20], the proof of the Turing-completeness of chemical reactions [33], or statements about the hardness of several dynamical systems problems [39].

The question of whether we can give a simple equivalent defining *space complexity* remains. We argue here that space complexity is polynomially related and conversely to the numerical stability of ODEs and their associated precision. We prove that a problem can be solved in polynomial space iff it can be simulated by some numerically stable ODE, using a polynomial precision. We prove this holds both for classical complexity over the discrete (functions over the integers) and also for space complexity for real functions in the model of computable analysis.

▶ Remark 1. In the literature, there are two possible definitions for **FPSPACE**, according to whether functions with non-polynomial size values are allowed or not. In this article, when we talk about **FPSPACE**, we always assume the outputs remain of polynomial size. Otherwise, the class is not closed by composition: the issue is about the usual convention of not counting the input and output as part of the total space used. Given f computable in polynomial space and g in logarithmic space, $f \circ g$ (and $g \circ f$) is computable in polynomial space. But, if exponential size output is allowed, this is not true: the issue is that if we assumed only f and g to be computable in polynomial space, the first might give an output of exponential size.

These questions of providing characterisations of classical complexity using ODEs can also be seen from the so-called "implicit complexity" point of view. Having "simple" characterisations of computability and complexity classes is useful for various fundamental and applied science fields. We are interested here in "algebraic" characterisations of those classes: we want to define them as the smallest set $[f_1, \ldots, f_k; o_1, \ldots, o_l]$ where the f_i are functions, closed under the operators o_j. For example, the set of computable functions over the integers is well-known to be: $[0, 1, \pi^i_k;$ *composition, minimisation, primitive recursion*$]$. Implicit complexity aims at giving similar algebras for classes of complexity theory: a reference survey is [27, 28]. The main benefit is to avoid the use of the framework of Turing machines, which is rather heavy and not necessarily well-known outside fundamental computer science. Several characterisations for **PTIME** over the integers were proposed. The first is due to Cobham in [29], but relies on explicit *ad hoc* bounds. Other approaches have then been proposed, see surveys [27, 28]. Recently, Bournez and Durand in [13] suggested an algebra using the so-called "linear-length" discrete ODEs. Instead of having explicit bounds, the linearity of the involved discrete ODE guarantees polynomial time complexity.

Using a similar approach, Blanc and Bournez in [4] and in [5] extended the constructions to a characterisation of **PTIME** for function over the reals. The latter extended the result to **PSPACE**, defining *robust* ODEs. However, those models rely on discrete ODEs (a.k.a. finite differences), which are discrete-time and less natural than continuous ODEs. We review all those results in Section 3.2.

This paper can be related to [19, 17]: the authors of these articles provide a characterisation of **PTIME** with *continuous* ODEs, establishing that time complexity corresponds to the length of the involved curve, i.e. the motto **time complexity = length**. Here, we get a motto of the form **space complexity = precision**.

Some of our constructions have similarities with the statements in [7]. In the later paper, various robustness concepts are introduced and it is proven that they lead to tractability. See the references in [7] for similar robustness statements. Robustness can also be associated with a dual motivation: the authors of [40] introduced a concept of robust undecidability, while here, we want a concept of robustness leading to tractability.

This is not the first time **FPSPACE** is characterised using continuous ODEs. However, the existing characterisation [34, 14] is obtained with complicated conditions on ODEs, while we have a simpler statement linking complexity to precision in a direct manner. Notice that

the latter approach dealt with polynomial ODEs, while we do not restrict to polynomial ODEs. We obtain our statements by revisiting the approach of the latter papers but working over a compact domain and dealing with error correction more finely.

Intuitively, this can also be read as being in **PSPACE** for an ODE is consistent with having an attractor easily discretisable when there is one. We can also define the notion of robustness, as the insensitivity to "small" perturbations.

While discussing all these issues, we propose an algebraic characterisation of **PSPACE**, using *continuous* ODEs with the following algebra ($\mathbb{RCD}$ is for Robust Continuous Differential) (Schema *robust ODE* is formally defined in Definition 9):

$$\mathbb{RCD} = [0, 1, \pi_i^k, +, -, \times, \tanh, \cos, \pi, \frac{x}{2}, \frac{x}{3}; \textit{composition}, \textit{robust ODE}].$$

For a function $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^{d'}$ sending every integer $\mathbf{n} \in \mathbb{N}^d$ to the vicinity of some integer of $\mathbb{N}^d$, say at distance less than $1/4$, we write $\mathrm{DP}(f)$ for its discrete part: this is the function from $\mathbb{N}^d \to \mathbb{N}^{d'}$ mapping $\mathbf{n} \in \mathbb{N}^d$ to the integer rounding of $\mathbf{f}(n)$. For a class $\mathcal{C}$ of such functions, we write $\mathrm{DP}(\mathcal{C})$ for the class of the discrete parts of the functions of $\mathcal{C}$.

▶ **Theorem 2.** $\mathrm{DP}(\mathbb{RCD}) = \mathbf{FPSPACE} \cap \mathbb{N}^{\mathbb{N}}$.

We also provide a characterisation of functions over the reals computable in polynomial space, inspired by [5]. This is obtained by adding a limit schema *ELim* to $\mathbb{RCD}$. If we consider $\overline{\mathbb{RCD}} = [0, 1, \pi_i^k, +, -, \times, \tanh, \cos, \pi, \frac{x}{2}, \frac{x}{3}; \textit{composition}, \textit{robust ODE}, \textit{ELim}]$ then:

▶ **Theorem 3** (Generic functions over the reals). $\overline{\mathbb{RCD}} \cap \mathbb{R}^{\mathbb{R}} = \mathbf{FPSPACE} \cap \mathbb{R}^{\mathbb{R}}$
More generally: $\overline{\mathbb{RCD}} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}} = \mathbf{FPSPACE} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^d}$.

We organise the article as follows. In Section 2, we recall the concept of dynamical systems and discuss some associated complexity issues. We introduce the concept of robust ODE and prove that a robust ODE can be solved in polynomial space (Theorem 11). This is obtained, using an original method for solving ODE, optimising space, inspired by Savitch's theorem. This provides one direction of all the above theorems. The other direction is the object of the following sections, starting from Section 3. We first recall some previous results on discrete ODEs in Section 3. Using extensions of constructions from [5], we then prove that we can simulate a Turing machine using robust continuous ODEs in Section 4. This is obtained by simulating some discrete ODEs using continuous ODEs, dealing with error corrections, and using the fact that the functions are robust to a controlled error. The main result of Section 3 is Theorem 42. It states we can simulate Turing machines robustly with continuous ODEs when space remains polynomial. This theorem leads to the proof of Theorem 2 in Section 4. In Section 5, we prove Theorem 3. In Section 6, we conclude and discuss future works.

Some basic concepts

When we say that a function over the real is computable this is always in the sense of computable analysis: see e.g. [59, 46, 23]. A reference book for issues related to complexity theory in computable analysis is [46].

We assume some basic familiarities with dynamical systems. See [41] for a monography on the theory of dynamical systems from a mathematical point of view. Formally, a *discrete-time dynamical system* is given by a set D, called the *domain* and some (possibly partial) function $\mathbf{u}$ from D to D. A trajectory of the system is a sequence $\mathbf{f}(t)$ evolving according to $\mathbf{u}$: that is $\mathbf{f}(t+1) = \mathbf{u}(\mathbf{f}(t))$ for all t. A *continuous-time dynamical system* is given by a set $D \subseteq \mathbb{R}^d$ and some ODE of the form

$$\mathbf{f}' = \mathbf{u}(\mathbf{f}(t)) \tag{1}$$

on D. A trajectory starting from $\mathbf{f}_0$ is a solution of the associated Initial Value Problem (IVP), given by (1) and initial condition $\mathbf{f}(0) = \mathbf{f}_0$. A dynamical system can equivalently be described by its flow: $\Phi(\mathbf{f}_0, t)$ gives the position of the dynamics at time t, for an initial position $\mathbf{f}(0) = \mathbf{f}_0$. It satisfies the flow property

$$\Phi(\mathbf{f}_0, 0) = \mathbf{f}_0 \quad \Phi(\mathbf{f}_0, t + t') = \Phi(\Phi(\mathbf{f}_0, t), t'). \tag{2}$$

The dynamics or the flow can be parametrised by some $\mathbf{x}$: $\mathbf{u}$ is also some function of $\mathbf{x}$ and the flow function is $\Phi_{\mathbf{x}}(\mathbf{f}_0, t)$.

In the long run, dynamical systems may exhibit attractors. We refer to [48] for discussions of many possible ways of defining this concept, and to [51] for a characterisation of the hardness of computing attractors from a computable analysis point of view. Somehow, our coming results state that the uncomputability discussed in [51] is intrinsically due to the non-numerical stability of the considered dynamical systems there.

2 Dynamical systems and associated complexity issues

2.1 Some complexity results on graphs

We need to discuss the hardness of solving IVP, or equivalently of computing $\Phi(y, t)$. For pedagogical reasons, we first discuss the case of a simple setting, namely the case of a (deterministic) directed graph. Indeed, observe that a discrete-time dynamical system (D, u) can also be seen as a particular (deterministic) directed graph $G = (V, \to)$, where, in the general case, V is not necessarily finite: G corresponds to $V = D$ and $\to$ to the graph of the function u, i.e. $\mathbf{x}_t \to \mathbf{x}_{t+1}$ iff $\mathbf{x}_{t+1} = \mathbf{u}(\mathbf{x}_t)$. The obtained graph is deterministic because any vertex has an outdegree 1. Starting from some point $\mathbf{x}_0$, there is at most one possible path, and consequently, for a given time T, we can talk about its position at time t, i.e. $\Phi(\mathbf{x}_0, T)$ is Tth element of this path: (as usual in complexity theory, the length of some integer x is the length of its binary representation, denoted by $\ell(x)$).

▶ **Proposition 4** (The case of finite graphs). *Let $s(n) \geq \log(n)$ be space-constructible. Assume the vertices of $G = (V, \to)$ can be encoded in binary using words of length $s(n)$. Assume the relation $\to$ is decidable using a space polynomial in $s(n)$. Then,*

- *given the encoding of $\mathbf{u} \in V$ and of $\mathbf{v} \in V$, we can decide whether there is some path from $\mathbf{u}$ to $\mathbf{v}$, in a space polynomial in $s(n)$.*
- *given the encoding of $\mathbf{u} \in V$, and integer T in binary, we can compute $\Phi(\mathbf{u}, T)$, in a space polynomial in $s(n)$ and the length of T.*

The second item is even a characterisation of the complexity of the problem. Indeed, the converse is true: If, given the encoding of $\mathbf{u} \in V$, and integer T in binary, we can compute $\Phi(\mathbf{u}, T)$, in a space polynomial in $s(n)$ and the length of T, then as $\to$ is given by $\Phi(., 1)$, then $\to$ is decidable using a space polynomial in $s(n)$.

Proof. It is well-known that for finite graphs, given a directed graph $G = (V, \to)$ and some vertices $\mathbf{u}, \mathbf{v} \in V$, determine whether there is some path between $\mathbf{u}$ and $\mathbf{v}$ in G, denoted by $\mathbf{u} \xrightarrow{*} \mathbf{v}$ is in NLOGSPACE: the rough idea is to guess non-deterministically the intermediate nodes. The formal proof is detailed in [54]. The same algorithm, working over representations

of vertices, when vertices are encoded using words of length $s(n)$ will work in NSPACE($s(n)$) (with the addition of the binary encoding of T if for the second item, if it bigger than $s(n)$). We then observe that NSPACE($s(n)$) = SPACE($s(n)$) from Savitch's theorem, recalled below. ◀

▶ **Theorem 5** (Savitch's theorem, [54, Theorem 8.5]). *For any space-constructible*[1] *function* $s : \mathbb{N} \to \mathbb{N}$ *with* $s(n) \geq \log n$*, we have* NSPACE($s(n)$) $\subseteq$ SPACE($s^2(n)$)*.*

Recall that the key argument of the proof of Theorem 5 is to express the question as a recursive procedure (expressing reachability in less than 2^t steps, called $CANYIELD(\mathbf{x}, \mathbf{y}, t)$ in [54]) guaranteeing the required space complexity: we write that relation $CANYIELD(\mathbf{x}, \mathbf{y}, t)$ is relation $\mathbf{x} \to \mathbf{y}$ when $t = 1$, and is relation $\exists \mathbf{z}$ such that $CANYIELD(\mathbf{x}, \mathbf{z}, t/2)$ and $CANYIELD(\mathbf{z}, \mathbf{y}, t/2)$ otherwise. If one prefers, this can also be understood as "guessing" some intermediate node $\mathbf{z}$.

▶ Remark 6 (Attractor point of view). We presented the above statement in terms of computing the flow $\Phi(\mathbf{x}, T)$. This could alternatively be interpreted in terms of attractors. Indeed, when the above hypothesis holds, then the dynamics is captured by a graph. In the long run, in particular, if T is greater than the number of vertices, any trajectory loops (i.e. reaches an attractor). The above statement could then also be read as the fact that such an attractor is then polynomial space computable.

2.2 Solving efficiently ODEs: what is known

This idea leads to an original method for solving ODEs. At least, this is original for the numerical analysis literature, as far as we know. A recent survey about computability and complexity issues for solving ODEs is [39]. In short: First, it is important to distinguish the case where we want to solve the ODE on a bounded (hence a compact) domain, from the case of the full domain $\mathbb{R}$: in the latter case, we might ask questions about the evolution of the system on the long run, which is harder. Over a compact domain, it is known that there exists some polynomial-time computable function $u : [-1, 1] \times [0, 1] \to \mathbb{R}$ such that $f' = u(f, t)$ has no computable solution, even over $[0, \delta]$, for any $\delta > 0$: see [45, 50, 1]. The involved ODE has no unique solution. It is known over compact or non-compact domains that if unicity holds, then its solution is computable [30, 31, 52]. However, the complexity can be arbitrarily high [46, 47]. If we want to get to tractability, then some regularity hypotheses must be assumed. A classical hypothesis is to assume the ODE to be Lipschitz.

Over a compact domain, it has been observed in several references (see e.g. [46]) that a careful analysis of Euler's method proves that, if $u : B(0, 1) \times [0, 1] \to \mathbb{R}^n$, with $B(0, 1) \subseteq \mathbb{R}^n$, is a polynomial time computable (right-)Lipschitz function then any solution $f : [0, 1] \to B(0, 1)$ of $f' = u(f, t)$ must be **FPSPACE**: see the discussions around Theorem 3.2 in [39] with several references. Kawamura has proved in [42] that there exists a polynomial-time computable function $u : [-1, 1] \times [0, 1] \to \mathbb{R}$, which satisfies a Lipschitz condition, such that the unique solution $f : [0, 1] \to \mathbb{R}$ takes values in $[-1, 1]$ and computing it leads to a **PSPACE**-complete problem. Hence, the question of solving ODEs over a compact domain in polynomial time corresponds to the question **PTIME** = **PSPACE** [42], even for $\mathcal{C}^\infty$-functions [43].

[1] As proved in [54], this hypothesis can be avoided, at the price of a slightly more complicated proof.

However, all these results are over compact domains, and dealing with non-compact domains, i.e. in the long run, is harder. **PSPACE** membership is not true, as this is possible to simulate any Turing machine by some finite-dimensional polynomial ODE [38] over a non-compact domain. This led to many undecidability results for analytic, and even very simple ODEs: see e.g. [38]. A possible way to analyse efficiency is to analyse the complexity of the solution assuming a bound on the function's growth (i.e. using parameterised complexity). It was proved in [16] that one can solve a polynomial ODE in polynomial time assuming a bound on $\mathbf{Y}(T) = \max_{0\leq t\leq T} \|\mathbf{f}(t)\|$. The result for polynomial ODEs was later improved in [49], where it is proved that the time T and parameter $\mathbf{Y}$ can be replaced by a single parameter, namely the length of the curve for polynomial ODEs. Furthermore, this parameter does not need to be given as input to the algorithm. This is a key argument for one direction of the motto "time complexity = length" we mentioned above. To get polynomial-time complexity over a non-compact domain, it is also mandatory not to use most classical methods from numerical analysis.

The same happens when discussing space complexity: a non-classical method is required to guarantee polynomial space complexity in the long run. No such method has yet been proposed, and this is the purpose of the coming subsection. Actually, for space complexity, in addition to all the problems mentioned, in all the above space or time analyses, the problem is that the complexity is (possibly implicitly) dependent on the Lipschitz constant or the length of the solution. In a system as simple as linear dynamics, the state at time T depends in Lipchitz way from the state at time 0, and the number of additional bits required to guarantee some precision 2^{-n} growth linearly with T. But the problem is that in a space polynomial in the input size, T has no reason to remain polynomial (consider, for example, a system simulating a Turing machine, as we will consider soon). Hence, the required precision is possibly exponential in the input size.

▶ Remark 7. The above comments can be interpreted informally as the fact that "most" (this could be "generic" in the sense of [51], i.e. (effective) descriptive theory) dynamical systems are intrinsically unstable, and an error method introduced at some step can make the method unavoidably incorrect in the long run unless we have a means to "guess" what will happen.

▶ Remark 8 (Attractor point of view). We presented the above statement in terms of computing the flow $\Phi(\mathbf{f}_0, T)$. But, this could alternatively be interpreted in terms of attractors. The point is that computing the attractors of a given dynamical system is hard in general, as this involves long-run behaviours. This explains all the undecidability results obtained in [51], even for very simple dynamics. However, as we will see, this is also explained by the fact that the latter paper discusses numerically unstable systems.

2.3 Solving efficiently ODEs: a space efficient method

This leads to an alternative approach to optimize space complexity: this can be seen as either using a non-deterministic algorithm that "guesses" the correct intermediate positions of the dynamics or, from the proof of Savitch's theorem approach, as an original recursive method to solve ODEs. As far as we know, we have never seen such a method discussed in the literature for solving ODEs.

Concretely: from the flow property, a strategy to compute $\Phi(\mathbf{f}_0, T)$ is either to use a particular numerical method if T is small, says smaller than $\Delta > 0$. Otherwise, we know that $\Phi(\mathbf{f}_0, T) = \Phi(\mathbf{z}, T/2)$, where $\mathbf{z} = \Phi(\mathbf{f}_0, T/2)$. This always holds, so if we can compute both quantities, we will solve the problem. The difficulty is that we cannot precisely compute $\mathbf{z}$ in

practice, but some numerical approximation $\widetilde{\mathbf{z}}$. If the system is numerically stable, we may assume this strategy works. The case when this strategy will not work is if the trajectory starting from $\widetilde{\mathbf{z}}$, for the second half of the work from time $T/2$ to T, has a behaviour different from the one starting in $\mathbf{z}$: in other words, if there is a high instability somewhere, namely in $\mathbf{z}$.

This leads to the following concept: we write $\mathbf{a} =_\epsilon \mathbf{b}$ for $\|\mathbf{a} - \mathbf{b}\| \leq \epsilon$ for conciseness.

▶ **Definition 9** (Robust (continuous) ODE). *A function* $\mathbf{f} : \mathbb{R}^{d+1} \to \mathbb{R}^{d'}$ *is robustly ODE definable (from initial condition* $\mathbf{g}$*, and dynamic* $\mathbf{u}$*) if*

1. *it corresponds to the solution of the following continuous ODE:*

$$\mathbf{f}(0, \mathbf{x}) = \mathbf{g}(\mathbf{x}) \quad \text{and} \quad \frac{\partial \mathbf{f}(t, \mathbf{x})}{\partial t} = \mathbf{u}(\mathbf{f}(t, \mathbf{x}), t, \mathbf{x}), \tag{3}$$

2. *and there is some rational* $\Delta > 0$*, and some polynomial* p *such that the schema (3) is (polynomially) numerically stable on* $[0, \Delta]$*: for all integer* n*, considering* $\eta(n) = p(n + \ell(\lceil \mathbf{x} \rceil))$ *we can compute* $\mathbf{f}(t, \mathbf{x})$ *at precision* 2^{-n} *by working a precision* $2^{-\eta(n)}$*: if you consider any solution of* $\tilde{\mathbf{x}} =_{2^{-\eta(n)}} \mathbf{x}$*,* $\tilde{\mathbf{f}}(0, \tilde{\mathbf{x}}) =_{2^{-\eta(n)}} \mathbf{g}(\mathbf{x})$ *and* $\frac{\partial \tilde{\mathbf{f}}(t, \tilde{\mathbf{x}})}{\partial t} =_{2^{-\eta(n)}} \mathbf{u}(\tilde{\mathbf{f}}(t, \tilde{\mathbf{x}}), t, \tilde{\mathbf{x}})$ *then* $\tilde{\mathbf{f}}(t, \tilde{\mathbf{x}}) =_{2^{-n}} \mathbf{f}(t, \mathbf{x})$ *when* $0 \leq t \leq \Delta$*.*
3. *For* $t \geq \Delta$*, we can compute* $\mathbf{f}(t, \mathbf{x})$ *at precision* 2^{-n} *by computing some approximation* $\widetilde{\mathbf{f}(t/2, \mathbf{x})}$ *of* $\mathbf{f}(t/2, \mathbf{x})$ *at precision* $2^{-\eta(n)}$*, i.e. of* $\Phi(\mathbf{g}(x), t/2)$*, and then some approximation of* $\Phi(\widetilde{\mathbf{f}(t/2, \mathbf{y})}, t/2)$*, working at precision* $2^{-\eta(n)}$*.*

▶ Remark 10. For more clarity, and conciseness, we will assume in the proofs that $d = d' = 1$, as it can be easily extended to more general cases.

▶ **Theorem 11.** *Consider an IVP as in the previous definition. If* $\mathbf{g}$ *and* $\mathbf{u}$ *are computable in polynomial space, then the solution* $\mathbf{f}$ *can be computed in polynomial space.*

Proof. From definitions and above arguments, all bits of $\Phi(\mathbf{y}, t)$ can be computed non-deterministically with precision n (i.e. at 2^{-n}) using computations with precision $\eta(n)$, hence is in $\mathbf{NPSPACE} = \mathbf{PSPACE}$. From the argument of the proof of Savitch's theorem, this can also be turned into a deterministic polynomial space recursive algorithm. ◀

The above theorem is the key argument to obtain one direction of our main theorems. We now go in the reverse direction. This requires talking about discrete ODEs, and some previous constructions.

3 Discrete ODEs: some previous results and constructions

3.1 Preliminary

We will use the concept of discrete ODE defined as follows (notice that we will write $\frac{\delta \mathbf{f}}{\delta n}$ for discrete derivation, by opposition of the classical $\frac{\partial \mathbf{f}}{\partial n}$ to help to distinguish discrete vs continuous ODEs.)

▶ **Definition 12** (Discrete derivation, notation δ). *For* $\mathbf{f} : \mathbb{N} \to \mathbb{R}^d \to \mathbb{R}^{d'}$*, the* discrete derivation *of* $\mathbf{f}$ *is* $\frac{\delta \mathbf{f}}{\delta n}(n, \mathbf{x}) = \mathbf{f}(n + 1, \mathbf{x}) - \mathbf{f}(n, \mathbf{x})$*.*

▶ Remark 13. We use the terminology "discrete ODE", as in [12, 13]. This concept has various names in other communities: this is also called *finite differences*, *difference equations*, sometimes *discretized ODE*, and the associated theory is sometimes called *discrete calculus*,

umbral calculus in other communities . Sometimes, some of the statements seem to be rediscovered with other names, but as far as we know, the idea of computing with discrete ODEs can be associated with [12, 13], and we follow the terminology used there: we refer to the discussions and references in [13] for the references and some discussions about the various names used in literature for similar concepts.

3.2 Algebraic characterisation with discrete ODEs: state of the art

In this subsection, we review some of the results already obtained using discrete ODEs.

▶ Remark 14. Notice that we do not need any of these statements directly, even if we will sometimes reuse some of their constructions (and some of their ideas).

Characterising PTIME over the integers. The concept of derivation along the length was introduced in [12]. A characterisation of **FPTIME** for functions over the integers has then been obtained in [12]:

▶ **Theorem 15** (Functions over the integers [12]). $\mathbb{LDL} \cap \mathbb{N}^{\mathbb{N}} = \mathbf{FPTIME} \cap \mathbb{N}^{\mathbb{N}}$, *for* $\mathbb{LDL} = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \times, sg(x)$; *composition, linear length ODE*$]$, *with* π_i^k *the projection function, and* $sg(x)$ *is 0 for* $x < 0$ *and 1 for* $x > 0$.

Toward the real numbers: characterising real sequences. Later, we introduced in [4]

▶ **Definition 16** (Operation $ELim$). *Given* $\tilde{\mathbf{f}} : \mathbb{R}^d \times \mathbb{N} \to \mathbb{R}^{d'} \in \mathbb{LDL}^\bullet$ *such that for all* $\mathbf{x} \in \mathbb{R}^d$, $n \in \mathbb{N}$, $\|\tilde{\mathbf{f}}(\mathbf{x}, 2^n) - \mathbf{f}(\mathbf{x})\| \leq 2^{-n}$ *for some function* $\mathbf{f}$, *then* $ELim(\tilde{\mathbf{f}})$ *is the (uniquely defined) corresponding function* $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^{d'}$.

and then we considered the class

$$\overline{\mathbb{LDL}^\bullet} = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \times, \overline{\text{cond}}(x), \frac{x}{2}; \textit{composition, linear length ODE, ELim}],$$

with $\overline{\text{cond}}(x)$ a sigmoid valuing 0 when $x < \frac{1}{4}$ and 1 when $x > \frac{3}{4}$. We proved this provides a characterisation of functions from $\mathbb{N}$ to $\mathbb{R}$ computable in polynomial time.

▶ **Theorem 17** (Sequences of reals [4]). $\overline{\mathbb{LDL}^\bullet} = \mathbf{FPTIME} \cap \mathbb{R}^{\mathbb{N}}$.

Characterisation of PTIME and PSPACE for functions over the real with discrete ODEs. We later succeeded in obtaining a characterisation of functions over the real computable in polynomial time and even space.

▶ **Theorem 18** (**FPTIME**, Generic functions over the reals [5]).
$\overline{\mathbb{LDL}^\circ} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}} = \mathbf{FPTIME} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^d}$, *with*
$\overline{\mathbb{LDL}^\circ} = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \tanh, \frac{x}{2}, \frac{x}{3}; \textit{composition, linear length ODE, ELim}]$.

Consider the following schema:

▶ **Definition 19** (Robust Discrete ODE [5]). *A bounded function* $\mathbf{f}$ *is robustly ODE definable if:*

1. *it corresponds to the solution of the following discrete ODE:*

$$\mathbf{f}(0, \mathbf{x}) = \mathbf{g}(\mathbf{x}) \quad \textit{and} \quad \frac{\delta \mathbf{f}(t, \mathbf{x})}{\delta t} = \mathbf{u}(\mathbf{f}(t, \mathbf{x}), \mathbf{h}(t, \mathbf{x}), t, \mathbf{x}), \tag{4}$$

2. *where the schema* (4) *is (polynomially) numerically stable: there exists some polynomial* p *such that, for all integer* n*, writing* $\eta(n) = p(n + \ell(\mathbf{y}))$*, if you consider any solution of* $\tilde{\mathbf{y}} =_{2^{-\eta(n)}} \mathbf{y}$ *and* $\tilde{\mathbf{h}}(x, \tilde{\mathbf{y}}) =_{2^{-\eta(n)}} \mathbf{h}(x, \tilde{\mathbf{y}})$*, and* $\tilde{\mathbf{f}}(0, \tilde{\mathbf{y}}) =_{2^{-\eta(n)}} \mathbf{g}(\mathbf{y})$ *and* $\frac{\partial \tilde{\mathbf{f}}(x,\tilde{\mathbf{y}})}{\partial x} =_{2^{-\eta(n)}} \mathbf{u}(\tilde{\mathbf{f}}(x, \tilde{\mathbf{y}}), \tilde{\mathbf{h}}(x, \tilde{\mathbf{y}}), x, \tilde{\mathbf{y}})$ *then* $\tilde{\mathbf{f}}(x, \tilde{\mathbf{y}}) =_{2^{-n}} \mathbf{f}(x, \mathbf{y})$*.*

We recall the notion of *essential linearity*: The idea is to measure the degree, similarly to the classical notion of degree in polynomial expression, but considering all subterms that are within the scope of a tanh function contributes to 0 to the degree. Then, essential linearity corresponds to linearity with this concept of degree.

▶ **Definition 20** ([4])**.** *The degree* $\deg(x, P)$ *of a term* P *in* $x \in V$ *is defined inductively as follows:*

- $\deg(x, x) = 1$ *and for* $x' \in V \cup \mathbb{Z}$ *such that* $x' \neq x$*,* $\deg(x, x') = 0$*;*
- $\deg(x, P + Q) = \max\{\deg(x, P), \deg(x, Q)\}$*;*
- $\deg(x, P \times Q) = \deg(x, P) + \deg(x, Q)$*;*
- $\deg(x, \tanh(P)) = 0$*.*

A polynomial expression P *is essentially constant in* x *if* $\deg(x, P) = 0$*.*

A vectorial function (resp. a matrix or a vector) is said to be a polynomial expression if all its coordinates (resp. coefficients) are, and *essentially constant* if all its coefficients are.

▶ **Definition 21** ([4])**.** *A polynomial expression* $\mathbf{g}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})$ *is essentially linear in* $\mathbf{f}(x, \mathbf{y})$ *if it is of the form:* $\mathbf{A}[\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}] \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}[\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}]$ *where* $\mathbf{A}$ *and* $\mathbf{B}$ *are polynomial expressions essentially constant in* $\mathbf{f}(x, \mathbf{y})$*.*

▶ Remark 22. A robust discrete ODE is said to be *linear* if $\mathbf{u}$ is essentially linear in $\mathbf{f}$ and $\mathbf{h}$.

Consider
$\overline{\mathbb{RLD}^\circ} = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \tanh, \frac{x}{2}, \frac{x}{3}; \textit{composition}, \textit{robust linear ODE}, \textit{ELim}]$.

▶ **Theorem 23** (**FPSPACE**, Generic functions over the reals [5])**.**
$\overline{\mathbb{RLD}^\circ} \cap \mathbb{R}^{\mathbb{R}} = \mathbf{FPSPACE} \cap \mathbb{R}^{\mathbb{R}}$
More generally: $\overline{\mathbb{RLD}^\circ} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}} = \mathbf{FPSPACE} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^d}$.

Notice that previous classes mix functions with integer and real arguments. Furthermore, they all involve some various types of discrete ODEs. We need to avoid all these issues, as we consider only continuous ODEs.

3.3 Simulating a discrete ODE using a continuous ODE

We first prove that it is possible to simulate a discrete ODE with a continuous ODE. The underlying idea can be attributed to [22], and has been improved in many ways by several authors. We present here the basic ideas, reformulated in our context. A more precise analysis will come (Proposition 39).

▶ **Definition 24** ("Ideal iteration trick", [22])**.** *Consider the following initial value problem for a discrete ODE, given by functions* $\mathbf{g}$ *and* $\mathbf{u}$*:*

$$\begin{cases} \mathbf{f}(0, \mathbf{x}) = \mathbf{g}(\mathbf{x}) \\ \dfrac{\delta \mathbf{f}}{\delta t}(t, \mathbf{x}) = \mathbf{u}(\mathbf{f}(t, \mathbf{x}), t, \mathbf{x}) \end{cases} \tag{5}$$

Then, let $\mathbf{G}(\mathbf{v}, t, \mathbf{x}) = \mathbf{u}(\mathbf{v}, t, \mathbf{x}) + \mathbf{v}$*, and consider the (continuous) IVP:*

$$\begin{cases} \mathbf{y}_1(0, \mathbf{x}) = \mathbf{y}_2(0, \mathbf{x}) = \mathbf{g}(\mathbf{x}) \\ \mathbf{y}_1' = c(\mathbf{G}(r(\mathbf{y}_2), r(t), \mathbf{x}) - \mathbf{y}_1)^3 \theta(\sin(2\pi t)) \\ \mathbf{y}_2' = c(r(\mathbf{y}_1) - \mathbf{y}_2)^3 \theta(-\sin(2\pi t)) \end{cases} \tag{6}$$

where c *a constant,* $\theta(x) = 0$ *if* $x \leq 0$ *and* $\theta(x) > 0$ *if* $x > 0$*. We abusively write* $r(\mathbf{y})$ *for the application of function* $r : \mathbb{R} \to \mathbb{R}$ *componentwise on vector* $\mathbf{y}$*. Here,* r *is a rounding function: we mean, by construction,* G *preserves the integers, and* r *is a function that maps a real value close to some integer to this integer: assume, say, that for* $z \in [n - \frac{1}{4}, n + \frac{1}{2}]$*,* $r(z) = n$*, for any integer* $n \in \mathbb{Z}$*.*

▶ Remark 25. We do not need to specify what $r(z)$ values for a z not in such an interval: the following reasoning remains correct, whatever it is.

Then, the solution of continuous ODE (6) simulates in a continuous way the discrete ODE (5): Indeed, $\mathbf{y}_1$ corresponds to the actual computation of the iterates of $\mathbf{G}$ (and hence computes the successive values of $\mathbf{f}$) and $\mathbf{y}_2$ acts as a "memory" equation. Let us detail how it works.

▶ Remark 26. We describe here an "ideal" computation, as $\theta(x)$ is exacly 0 when $x \leq 0$, and $r(z)$ is exactly some integer on suitable domains. Later in the paper, we will deal with a not-so-ideal θ and r.

Initially, $\mathbf{f}(0, \mathbf{x}) = \mathbf{y}_1(0, \mathbf{x}) = \mathbf{y}_2(0, \mathbf{x}) = \mathbf{g}(\mathbf{x})$. For $t \in [0, 1/2]$, we have $\theta(-\sin(2\pi t)) = 0$, and hence $\mathbf{y}_2' = 0$, so $\mathbf{y}_2$ is fixed and kept at value $\mathbf{g}(\mathbf{x})$ for $t \in [0, \frac{1}{2}]$. Consequently, for $t \in [0, 1/2]$, $r(\mathbf{y}_2)$ is also fixed and kept at value $\mathbf{g}(\mathbf{x})$, and $r(t)$ is also fixed and kept at value 0. Consequently, on this interval, if we write $C(t) = c\theta(\sin(2\pi t))$, then the dynamics of $\mathbf{y}_1$ is given by

$$\mathbf{y}_1' = C(t)(\mathbf{G}(\mathbf{g}(\mathbf{x}), 0, \mathbf{x}) - \mathbf{y}_1)^3 \tag{7}$$

▶ **Lemma 27** (Analysis of ODE (7))**.** *The solution* $\mathbf{y}_1(t, \mathbf{x})$ *of ODE* (7) *is converging to* $G(\mathbf{g}(\mathbf{x}), 0, \mathbf{x})$ *for any initial condition. Furthermore, for any initial condition* $\mathbf{y}_1(0, \mathbf{x}) \neq G(\mathbf{g}(\mathbf{x}), 0, \mathbf{x})$*, we have* $\left\|\mathbf{y}_1(\frac{1}{2}, \mathbf{x}) - \mathbf{G}(\mathbf{g}(x), 0, \mathbf{x})\right\| \leq \frac{\sqrt{2}}{2\sqrt{\int_0^{\frac{1}{2}} C(z)dz}}$*. In particular, for any* $m \in \mathbb{N}$*, we can select constant* c *such that for any initial condition* $\mathbf{y}_1(0, \mathbf{x})$*,* $\left\|\mathbf{y}_1(\frac{1}{2}, \mathbf{x}) - \mathbf{G}(\mathbf{g}(x), 0, \mathbf{x})\right\| \leq 2^{-m}$*.*

Consequently, $\mathbf{y}_1(t, \mathbf{x})$ will approach $\mathbf{G}(\mathbf{g}(\mathbf{x}), 0, \mathbf{x}) = \mathbf{f}(1, \mathbf{x})$ on this interval. Thus, $\mathbf{y}_1(\frac{1}{2}, \mathbf{x}) =_\epsilon \mathbf{f}(1, \mathbf{x})$ and $\mathbf{y}_2(\frac{1}{2}, \mathbf{x}) = \mathbf{g}(\mathbf{x})$, for some $\epsilon > 0$, that we can consider less than $\frac{1}{4} = 2^{-2}$, by selecting a big enough constant c (just taking $m = 2$ above). At $t = \frac{1}{2}$, $\mathbf{y}_1$ will hence have simulated one step of discrete ODE (5).

Now, for $t \in [\frac{1}{2}, 1]$ the roles of $\mathbf{y}_1$ and $\mathbf{y}_2$ are exchanged : $\mathbf{y}_1'(t, \mathbf{x}) = 0$, so $\mathbf{y}_1$ is kept fixed, $\mathbf{y}_2$ approaches $r(\mathbf{y}_1) = \mathbf{f}(1, \mathbf{x})$, thus $\mathbf{y}_1(1, \mathbf{x}) =_\epsilon \mathbf{y}_2(1, x) =_\epsilon \mathbf{f}(1, \mathbf{x})$.

By induction, from the same reasoning, we obtain that, for all $n \in \mathbb{N}$, $\mathbf{y}_1(n, \mathbf{x}) =_\epsilon \mathbf{y}_2(n, \mathbf{x}) =_\epsilon \mathbf{f}(n, \mathbf{x})$, and actually, we also have $\mathbf{y}_1(t + \frac{1}{2}, \mathbf{x}) =_\epsilon \mathbf{y}_2(t, \mathbf{x}) =_\epsilon \mathbf{f}(n, \mathbf{x})$ for all $t \in [n, n + \frac{1}{2}]$, for any integer n.

To implement such an ODE, we have to fix a function $\theta(x)$ with the above property. Taking $\mathrm{ReLU}(x) = \max(0, x)$ would satisfy it, but it is not a derivable function, and hence would not lead to a (classical) ODE. We could then take $\theta(x) = 0$ for $x \leq 0$, and $\exp(-1/x)$ for $x > 0$. The point is that such a function is not real analytic. The base functions we

consider in our class $\mathbb{RCD}$ are all real analytic, and real analytic functions are preserved by composition, so we cannot get such a function by compositions from our base functions. Furthermore, it is known that a real analytic function that is constant on some interval (we assumed it is 0 for $x \leq 0$) is constant. Hence, the above-considered function $\theta(x)$ cannot be real analytic. So, implementing this trick cannot be done directly using our base functions, using only compositions.

In Proposition 39, we will do a similar construction, but dealing with errors and not exact functions $\theta(z)$ and $r(x)$. Furthermore, here the purpose of function r was to correct errors around integers, i.e. around $\mathbb{Z}$: this will be possibly around other $\mathbb{Z}\delta = \{n\delta | n \in \mathbb{Z}\}$ for some $\delta > 0$. We will then naturally assume that for $z \in [n\delta - \frac{1}{4}\delta, n\delta + \frac{1}{2}\delta]$, $r(z) = n$, for any integer $n \in \mathbb{Z}$, in order to have a similar reasoning as the one above where $\delta = 1$.

3.4 Encoding of Turing machines configurations

Our proofs rely on some constructions from [5]. Concretely, we need to simulate the execution of a Turing machine (TM) by some dynamical system over the reals. This requires to encode the configurations of a Turing machine into some real numbers. We recall some of the definitions and constructions from [5].

Consider a Turing machine defined by $\mathcal{M} = (\Sigma, Q, I, F, \delta)$, with Σ the working alphabet, Q the set of states, $I, F \subseteq Q$ respectively the sets of initial and final states, $\delta : Q \times \Sigma \to Q \times \Sigma \times \{\leftarrow, \rightarrow\}$ the transition function. For some practical reasons, similar to the ones in [5], we assume that the working alphabet is made of the symbols 1 and 3, and that the blank symbol is symbol 0.

We explicit the encoding we will use. We assume $Q = \{0, 1, \ldots, |Q| - 1\}$. Let

$$\ldots l_{-k} l_{-k+1} \ldots l_{-1} l_0 r_0 r_1 \ldots r_n \ldots.$$

denote the content of the tape of the Turing machine M. In this representation, the head is in front of symbol r_0, and $l_i, r_i \in \{0, 1, 3\}$ for all i. Furthermore, we assume that there are no non-blank symbols between two blank symbols, i.e. that blank symbols, i.e. symbol 0, can only be eventually on the right, or eventually on the left. Such a configuration C can be denoted by $C = (q, l, r)$, where $l, r \in \Sigma^\omega$ are words over alphabet $\Sigma = \{0, 1, 3\}$ and $q \in Q$ denotes the internal state of M.

Now, write $\gamma_{word} : \Sigma^\omega \to \mathbb{R}$ for the function that maps a word $w = w_0 w_1 w_2 \ldots$ to the dyadic (hence real) number $\gamma_{word}(w) = \sum_{n \geq 0} w_n 4^{-(n+1)}$.

The idea is that configuration C can also be encoded by some element $\overline{C} = (q, \overline{l}, \overline{r}) \in \mathbb{N} \times \mathbb{R}^2$, by considering $\overline{r} = \gamma_{word}(r)$ and $\overline{l} = \gamma_{word}(l)$. In other words, we encode the configuration of a bi-infinite tape Turing machine M by real numbers using their radix 4 encoding, but using only digits 1,3. Notice that this lives in $Q \times [0, 1]^2$. Denoting the image of $\gamma_{word} : \Sigma^\omega \to \mathbb{R}$ by $\mathcal{I}$, this even lives in $Q \times \mathcal{I}^2$.

In other words, we consider the following encodings:

$$\gamma_{config}(C) = (q, \overline{l}, \overline{r})$$

with

$$\begin{aligned} \overline{l} &= l_0 4^{-1} + l_{-1} 4^{-2} + \cdots + l_{-k} 4^{-(k+1)} + \ldots \\ \overline{r} &= r_0 4^{-1} + r_1 4^{-2} + \cdots + l_n 4^{-(n+1)} + \ldots \end{aligned}$$

3.5 Revisiting some previous constructions

We denote by $\mathbb{RCD}_*$ the algebra $[0, 1, \pi_i^k, +, -, \times, \tanh, \cos, \pi, \frac{x}{2}, \frac{x}{3}; \textit{composition}]$. This is close to the class $\mathbb{LDL}^\circ = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \tanh, \frac{x}{2}, \frac{x}{3}; \textit{composition}, \textit{linear length ODE}]$, considered in [6, 5], but without the function $\ell(x)$, and without the possiblity of defining functions using linear length ODE (and with multiplication added).

We will reuse some of the construction from [5] (some corrections and more details can be found in [6]) but avoid systematically any use of linear length ODE and the length function $\ell(x)$. Furthermore, the class considered in [5] is mixing functions from the integers to the reals, and from the reals to the reals, and we need to keep only functions over the reals.

The following was stated in [5, Lemma 19].

▶ **Lemma 28.** *We denote by $Y(x, 2^{m+2})$ the function $\frac{1+\tanh(2^{m+2}x)}{2}$. For all integer m, for all $x \in \mathbb{R}$, $|\operatorname{ReLU}(x) - xY(x, 2^{m+2})| \leq 2^{-m}$, where $\operatorname{ReLU}(x) = \max(0, x)$.*

First, we observe that considering $Y(x, z) = \frac{1+\tanh(4xz)}{2}$ would yield a function in $\mathbb{RCD}_*$ with the same property: we avoid the computation of 2^m by a substitution of a variable, and using a multiplication. We then write ReLU-$\mathfrak{s}(Y, x)$ for $xY(x, z)$: we have $|\operatorname{ReLU}\text{-}\mathfrak{s}(2^m, x) - \operatorname{ReLU}(x)| \leq 2^{-m}$.

In particular, this was used to prove we can uniformly approximate the continuous sigmoid functions (when $1/(b-a)$ is in $\mathbb{LDL}^\circ$) defined as $\mathfrak{s}(a, b, x) = 0$ whenever $w \leq a$, $\frac{x-a}{b-a}$ whenever $a \leq x \leq b$, and 1 whenever $b \leq x$. The above trick provides a new version of [5, Lemma 20].

▶ **Lemma 29** (Uniform approximation of any piecewise continuous sigmoid)**.** *Assume $a, b, \frac{1}{b-a}$ is in $\mathbb{RCD}_*$. Then there is some function $\mathcal{C}\text{-}\mathfrak{s}(z, a, b, x) \in \mathbb{RCD}_*$ such that for all integer m,*

$$|\mathcal{C}\text{-}\mathfrak{s}(2^m, a, b, x) - \mathfrak{s}(a, b, x)| \leq 2^{-m}.$$

Proof. Take $\mathcal{C}\text{-}\mathfrak{s}(z, a, b, x) = \frac{(x-a)Y(x-a, z2^{1+c}) - (x-b)Y(x-b, z2^{1+c})}{b-a}$. observing that $(b-a)\,\mathfrak{s}(a, b, x) = \operatorname{ReLU}(x-a) - \operatorname{ReLU}(x-b)$. From triangle inequality, it will hold, choosing c with $\frac{1}{b-a} \leq 2^c$. ◀

The authors of [5] proved the existence of some function corresponding to a continuous (controlled) approximation of the fractional part function: we write by $\{.\}$ the fractional part function.

▶ **Theorem 30** ([5, Lemma 28])**.** *There exists some function $\xi : \mathbb{N}^2 \to \mathbb{R}$ in $\mathbb{LDL}^\circ$ such that for all $n, m \in \mathbb{N}$ and $x \in [-2^n, 2^n]$, whenever $x \in [\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8}]$, $\left|\xi(2^m, 2^n, x) - \{x - \frac{1}{8}\}\right| \leq 2^{-m}$.*

We say that some real function is a real extension of a function over the integers if they coincide for integer arguments. It is not clear that we have an extension over the reals of ξ in our algebra $\mathbb{RCD}_*$, but if we add a real extension of such a function, from the proof of [5, Corollary 22], we obtain the bestiary of functions considered in [5, Corollary 22]: we write $\mathbb{RCD}_* + \xi$ for the algebra where some real extension of function ξ is added as a base function.

▶ **Corollary 31** (A bestiary of functions)**.** *There exist*

1. *$\xi_1, \xi_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto \mathbb{R} \in \mathbb{RCD}_* + \xi$ such that, for all $n, m \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever $x \in [\lfloor x \rfloor - \frac{1}{2}, \lfloor x \rfloor + \frac{1}{4}]$, $|\xi_1(2^m, 2^n, x) - \{x\}| \leq 2^{-m}$, and whenever $x \in [\lfloor x \rfloor, \lfloor x \rfloor + \frac{3}{4}]$, $|\xi_2(2^m, 2^n, x) - \{x\}| \leq 2^{-m}$.*
2. *$\sigma_1, \sigma_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto \mathbb{R} \in \mathbb{RCD}_* + \xi$ such that, for all $n, m \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever $x \in [\lfloor x \rfloor - \frac{1}{2}, \lfloor x \rfloor + \frac{1}{4}]$, $|\sigma_1(2^m, 2^n, x) - \lfloor x \rfloor| \leq 2^{-m}$, and whenever $x \in I_2 = [\lfloor x \rfloor, \lfloor x \rfloor + \frac{3}{4}]$, $|\sigma_2(2^m, 2^n, x) - \lfloor x \rfloor| \leq 2^{-m}$.*

3. $\lambda : \mathbb{N}^2 \times \mathbb{R} \mapsto [0,1] \in \mathbb{RCD}_* + \xi$ *such that for all* $m, n \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, *whenever* $x \in [\lfloor x \rfloor + \frac{1}{4}, \lfloor x \rfloor + \frac{1}{2}]$, $|\lambda(2^m, 2^n, x) - 0| \leq 2^{-m}$, *and whenever* $x \in [\lfloor x \rfloor + \frac{3}{4}, \lfloor x \rfloor + 1]$, $|\lambda(2^m, 2^n, x) - 1| \leq 2^{-m}$.
4. $\bmod_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto [0,1] \in \mathbb{RCD}_* + \xi$ *such that for all* $m, n \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, *whenever* $x \in [\lfloor x \rfloor - \frac{1}{4}, \lfloor x \rfloor + \frac{1}{4}]$, $|\bmod_2(2^m, 2^n, x) - \lfloor x \rfloor \bmod 2| \leq 2^{-m}$.
5. $\div_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto [0,1] \in \mathbb{RCD}_* + \xi$ *such that for all* $m, n \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, *whenever* $x \in [\lfloor x \rfloor - \frac{1}{4}, \lfloor x \rfloor + \frac{1}{4}]$, $|\div_2(2^m, 2^n, x) - \lfloor x \rfloor // 2| \leq 2^{-m}$, *with* // *the integer division.*

Similarly, the equivalent of [5, Lemmas 23,24 and 25] still hold in $\mathbb{RCD}_* + \xi$:

▶ **Lemma 32.** *There exists* $\mathcal{C}$-if $\in \mathbb{RCD}_* + \xi$ *such that,* $l \in [0,1]$, *if we take* $|d' - 0| \leq 1/4$, *then* $|\mathcal{C}\text{-if}(2^m, d', l) - 0| \leq 2^{-m}$, *and if we take* $|d' - 1| \leq 1/4$, *then* $|\mathcal{C}\text{-if}(2^m, d', l) - l| \leq 2^{-m}$.

▶ **Lemma 33.** *Let* $\alpha_1, \alpha_2, \ldots, \alpha_n$ *be some integers, and* $V_1, V_2, \ldots, V_n$ *some constants. We write* $\text{send}(\alpha_i \mapsto V_i)_{i \in \{1,\ldots,n\}}$ *for the function that maps any* $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ *to* V_i, *for all* $i \in \{1, \ldots, n\}$.

There is some function in $\mathbb{RCD}_* + \xi$, *that we write* $\mathcal{C}\text{-send}(2^m, \alpha_i \mapsto V_i)_{i \in \{1,\ldots,n\}}$, *that maps any* $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ *to a real at distance at most* 2^{-m} *of* V_i, *for all* $i \in \{1, \ldots, n\}$.

▶ **Lemma 34.** *Let* N *be some integer. Let* $\alpha_1, \alpha_2, \ldots, \alpha_n$ *be some integers, and* $V_{i,j}$ *for* $1 \leq i \leq n$ *some constants, with* $0 \leq j < N$. *We write* $\text{send}((\alpha_i, j) \mapsto V_{i,j})_{i \in \{1,\ldots,n\}, j \in \{0,\ldots,N-1\}}$ *for the function that maps any* $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ *and* $y \in [j - 1/4, j + 1/4]$ *to* $V_{i,j}$, *for all* $i \in \{1, \ldots, n\}$, $j \in \{0, \ldots, N-1\}$.

There is some function in $\mathbb{RCD}_* + \xi$, *that we write* $\mathcal{C}\text{-send}(2^m, (\alpha_i, j) \mapsto V_{i,j})_{i \in \{1,\ldots,n\}, j \in \{0,\ldots,N-1\}}$, *that maps any* $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ *and* $y \in [j - 1/4, j + 1/4]$ *to a real at distance at most* 2^{-m} *of* $V_{i,j}$, *for all* $i \in \{1, \ldots, n\}$, $j \in \{0, \ldots, N-1\}$.

Working with one step of a Turing machine

As the proof of [5, Lemmas 30] is done using all the functions provided by these lemmas, we obtain:

▶ **Lemma 35.** *We can construct some function* $\overline{Next}$ *in* $\mathbb{RCD}_* + \xi$ *that simulates one step of* M, *i.e. computing the Next function sending a configuration* $\overline{C}$ *of Turing machine* M *to* $\overline{C'}$, *where* C' *is the next one:* $\|Next(2^m, 2^S, \overline{C}) - \overline{C'}\| \leq 2^{-m}$. *Furthermore, it is robust to errors on its input, up to space* S: *considering* $\|\tilde{C} - \overline{C}\| \leq 4^{-(S+2)}$, $\|Next(2^m, 2^S, \tilde{C}) - \overline{C'}\| \leq 2^{-m}$ *remains true.*

Converting integers an dyadics to words and conversely

The article [5] also defined some functions for converting integers and dyadics to their encoding as words, and conversely. Namely, the following encoding is considered: every digit in the binary expansion of dyadic d is encoded by a pair of symbols in the radix 4 expansion of $\overline{d} \in \mathcal{I} \cap [0,1]$: digit 0 (respectively: 1) is encoded by 11 (resp. 13) if before the "decimal" point in d, and digit 0 (respectively: 1) is encoded by 31 (resp. 33) if after. For example, for $d = 101.1$ in base 2, $\overline{d} = 0.13111333$ in base 4. Conversely, given $\overline{d}$, the article provided a way to construct d. This corresponds to [5, Lemmas 33 and 34]:

▶ **Lemma 36** (From $\mathbb{N}$ to $\mathcal{I}$). *We can construct some function* $Decode : \mathbb{N}^2 \to \mathbb{R}$ *in* $\mathbb{LDL}^\circ$ *that maps* m *and* n *to some point at distance less than* 2^{-m} *from* $\gamma_{word}(\overline{n})$.

▶ **Lemma 37** (From $\mathcal{I}$ to $\mathbb{R}$, and multiplying in parallel)**.** *We can construct some function* $\mathit{EncodeMul} : \mathbb{N}^2 \times [0,1] \times \mathbb{R} \to \mathbb{R}$ *in* $\mathbb{LDL}^\circ$ *that maps* m, 2^S, $\gamma_{word}(\overline{d})$ *and (bounded)* λ *to some real at distance at most* 2^{-m} *from* λd*, whenever* $\overline{d}$ *is of length less than* S*.*

As for ξ, it is not clear that we have some real extensions of these functions in $\mathbb{RCD}_*$: we write $\mathbb{RCD}_* + \xi + \mathit{Decode} + \mathit{Encode}$ for the algebra where some real extension of these functions is added as a base function.

3.6 Constructing the missing functions

We need a way to construct some substitute of "missing functions" (ξ, *Decode* and *EncodeMul*). As all of them are defined using discrete ODEs, an idea is to use a continuous ODE to simulate the respective discrete ODEs: we hence revisit the construction of the ideal iteration trick of Section 3.3, dealing with errors and not exact functions $\theta(z)$ and $r(x)$.

The key is to revisit Lemma 27, and do a more detailed analysis of possible involved errors in dynamics of the form (7). This equation has been studied by various authors in several articles, including [24, 25, 36, 18, 34]. We use the following statement from [34, Lemma 4.5], [14, Lemma 5.2], obtained basically by a case analysis of error propagations in Lemma 27.

▶ **Lemma 38** (Improved error analysis of ODE (7), [34, Lemma 4.5] [14, 5.2])**.** *Consider a point* $b \in \mathbb{R}$*, some* $\gamma > 0$ *some reals* $t_0 < t_1$*, and a function* $\phi : \mathbb{R} \to \mathbb{R}$ *with the property that* $\phi(t) \geq 0$ *for all* $t \geq t_0$ *and* $\int_{t_0}^{t_1} \phi(t)dt > 0$*. Let* $\rho, \delta \geq 0$ *and let* $\overline{b}, E : \mathbb{R} \to \mathbb{R}$ *be functions such that that* $|\overline{b}(t) - b| \leq \rho$ *and* $|E(t)| \leq \delta$ *for all* $t \geq t_0$*. Then the IVP defined by*

$$z' = c(\overline{b}(t) - z)^3\phi(t) + E(t)$$

with the initial condition $z(t_0) = \overline{z}_0$*, where* $\gamma > 0$ *and* $c \geq \frac{1}{2\gamma^2 \int_{t_0}^{t_1} \phi(t)dt}$ *satisfies*

1. $|z(t_1) - b| < \rho + \gamma + \delta(t_1 - t_0)$*, independently of the initial condition* $\overline{z}_0 \in \mathbb{R}$
2. $\min(\overline{z}_0, b - \rho) - \delta(t_1 - t_0) \leq z(t) \leq \max(\overline{z}_0, b + \rho) + \delta(t_1 - t_0)$ *for all* $t \in [t_0, t_1]$.

▶ **Proposition 39** (Simulating a discrete ODE by a continuous ODE)**.** *Assume* $\mathbf{G}$ *is almost constant around* $\mathbb{N}\delta$ *and* r *is a rounding function around* $\mathbb{N}\delta$ *for some* $\delta > 0$*: for* $z \in [n\delta - \frac{1}{4}\delta, n\delta + \frac{1}{2}\delta]$*,* $r(z) = n$*, for any integer* $n \in \mathbb{Z}$*.*

Suppose that, in (6)*, we replace function* $\theta(z)$ *and function* $r(z)$ *by some suitable approximations: we take* $\theta(x) = \mathrm{ReLU}(x)$*,* $\theta_{\epsilon'}(x)$*,* $r_{\epsilon'(z)}$ *such that* $\theta(z) =_{\epsilon'} \theta(z)$*, and* $r_{\epsilon'}(x) =_{\epsilon'}$*, and take constant* c *big enough. Then the solution of the obtained ODE will continuously simulate the discrete ODE* (5)*, with the same bounds as in the analysis in Section 3.3, i.e. with error at most* ϵ *if* ϵ' *is taken sufficiently small. To guarantee* $\epsilon = 2^{-n}$*, it is sufficient to take* $\epsilon' = 2^{-p(n)}$ *and* $\theta_{\epsilon'}(x) = \mathrm{ReLU}\text{-}\mathfrak{s}(2^{p(n)}, x)$ *for some polynomial* p*.*

Proof. The key is that the involved errors additively propagate, from Lemma 38. Namely, they are in $\mathcal{O}(\epsilon')$, but they are then corrected from the reasoning in Section 3.3: rounding function corrects errors or order ϵ whenever its argument is at a distance less than $1/4\delta$ of some $n\delta$ exactly as in the reasoning in Section 3.3 (where $\delta = 1$, even if now it introduces some error ϵ' at every step; but the latter is corrected at the next step). Observe that the involved constant c, is of order 2^n.

More formally, we claim that for all $n \in \mathbb{N}$, $\mathbf{y}_1(n, \mathbf{x}) =_\epsilon \mathbf{y}_2(n, \mathbf{x}) =_\epsilon \mathbf{f}(n, \mathbf{x})$, and $\mathbf{y}_1(t + \frac{1}{2}, \mathbf{x}) =_\epsilon \mathbf{y}_2(t, \mathbf{x}) =_\epsilon \mathbf{f}(n, \mathbf{x})$ for all $t \in [n, n + \frac{1}{2}]$.

For $n = 0$, initially $\mathbf{f}(0, \mathbf{x}) = \mathbf{y}_1(0, \mathbf{x}) = \mathbf{y}_2(0, \mathbf{x}) = \mathbf{g}(\mathbf{x})$. For $t \in [n, n + 1/2]$, we then have $\theta(-\sin(2\pi t)) =_{\epsilon'} 0$, and hence $\mathbf{y}_2' =_{\epsilon'} 0$, so $\mathbf{y}_2$ is kept close to value $\mathbf{g}(\mathbf{x})$ for $t \in [0, \frac{1}{2}]$, with an error less than $\frac{1}{2}\epsilon'$.

Consequently, for $t \in [0, 1/2]$, $r(\mathbf{y}_2)$ is kept close to a constant value $\mathbf{g}(\mathbf{x})$, when an error less than ϵ', if we choose $\epsilon' < \frac{1}{4}\delta$. Meanwhile, $r(t)$ is also at a value close to n with an error lesser than ϵ'.

Consequently, on this interval, if we write $C(t) = c\theta(\sin(2\pi t))$, then the dynamics of $\mathbf{y}_1$ is given by a dynamic of the form of Lemma 38. This lemma states that $\mathbf{y}_1(t, \mathbf{x})$ will approach $\mathbf{G}(\mathbf{g}(\mathbf{x}), 0, \mathbf{x}) = \mathbf{f}(1, \mathbf{x})$ on this interval, with an error of order $\epsilon' + \epsilon' + \frac{1}{2}\epsilon'$.

Here the hypothesis that $\mathbf{G}$ is almost constant around $\mathbb{N}\delta$ means that its value is guaranteed to be at ϵ' from $\mathbf{G}(\mathbf{g}(\mathbf{x}), 0, \mathbf{x})$ on the interval.

Thus, $\mathbf{y}_1(\frac{1}{2}, \mathbf{x}) =_{\epsilon/2} \mathbf{f}(1, \mathbf{x})$, if we choose $\frac{5}{2}\epsilon' < \epsilon/2$. At $t = n + \frac{1}{2}$, $\mathbf{y}_1$ will hence have simulated one step of discrete ODE (5), with an error less than $\epsilon/2$, and $\mathbf{y}_2$ will be close to $\mathbf{g}(\mathbf{x})$ with an error less than $\epsilon' < \epsilon/2$.

Now, for $t \in [n + \frac{1}{2}, n + 1]$ the roles of $\mathbf{y}_1$ and $\mathbf{y}_2$ are exchanged: $\mathbf{y}_1'(t, \mathbf{x}) =_{\epsilon'} 0$, so $\mathbf{y}_1$ is kept almost fixed, with a new error less than $\frac{1}{2}\epsilon'$. In the same time $\mathbf{y}_2$ approaches $r(\mathbf{y}_1) = \mathbf{f}(1, \mathbf{x})$ by Lemma 38, with some new error of order less than $\frac{5}{2}\epsilon' < \epsilon/2$.

Consequently, we get the property at rank $n + 1$. ◀

▶ Remark 40. Observe that, somehow, the idea is that the constructions always replace every function with a function that does not change much locally (i.e. changes in a controlled way). This is the key that provides a robust ODE as in Definition 9, leading to polynomial space complexity by Theorem 11.

In other words, whenever we have some discrete ODE as in (5) defining some function $\mathbf{f}(t, \mathbf{x})$, we can construct some continuous ODE, using only functions from $\mathbb{RCD}_*$, such that one of its projection provides a function $\mathbf{f}(z, t, \mathbf{x})$, with the guarantee $\mathbf{f}(2^n, t, \mathbf{x})$ is 2^{-n} close to $\mathbf{f}(n, \mathbf{x})$, whenever t is close (at a distance less than $1/4$) to some integer n.

This works, as we can obtain such a $r_{\epsilon'}(x)$ from the functions from Corollary 31: Consider $r(x, 2^m) = \sigma_2(2^m, 2^n, x + \frac{1}{4})$ that works over $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, and observe that this is sufficient to apply the trick for the required functions, from the form of the considered discrete ODE in [5].

Except that we have a bootstrap problem: ξ was defined using a discrete ODE in [5], and as the functions from Corollary 31 are defined above using ξ, we cannot apply this reasoning to get function ξ. But the point is that for the special case of ξ, it is easy to construct a function in $\mathbb{RCD}$ that corresponds to some real extension of ξ, as we have functions such as $\sin(x) = \cos(\frac{\pi}{2} - x)$ and π.

▶ **Lemma 41.** *Function ξ has some real extension in $\mathbb{RCD}_*$.*

Proof. If we succeed to obtain a function $i(2^m, 2^n, x)$ that values $\lfloor x \rfloor$ whenever $x \in [\lfloor x \rfloor, \lfloor x \rfloor + \frac{3}{4}]$, we are done, as we can then obtain $\xi(2^m, 2^n, x)$ by considering $\xi(2^m, 2^n, x) = x + \frac{7}{8} - i(2^m, 2^n, x + \frac{7}{8})$.

A possible solution is then the following: consider function $R_e(x) := \mathfrak{s}(x, 0, e/2)$, and then $t_e(x) = (1 - R_e(\sin(2\pi x)))((1 - R_e(\sin(4\pi x)))$. If we put aside some interval of width $e/2$ around $\frac{1}{2}$ and $\frac{7}{8}$ where it takes values in $[0, 1]$, it values 0 on $[\lfloor x \rfloor, \lfloor x \rfloor + \frac{7}{8}]$, and then 1 on $[\lfloor x \rfloor + \frac{7}{8}, \lfloor x \rfloor + 1]$. We can then consider $I_e(t) = 8\int_0^t t_e(x)dx$ (i.e. the solution of ODE $l_e' = 8t_e$), and then $i(t) =_{e.t} l_e(t)$. It is then sufficient to replace $\mathfrak{s}$ by $\mathcal{C}$-$\mathfrak{s}$, in the above expressions, in order to control the error and make it smaller than 2^{-m}. ◀

Consequently, this is true that we can substitute a discrete ODE with a continuous ODE for the required functions *Decode* and *EncodeMul*: just replace ξ in the involved schemas by the above function. Notice that we can also easily get a real extension of the function that maps n to 2^n.

3.7 Working with all steps of a Turing machines

We can then go from one step of a Turing machine, to arbitrarily many steps. We are following the idea of [5], but replacing discrete ODEs with continuous ODEs.

▶ **Theorem 42.** *Consider some Turing machine M that computes some function $f : \Sigma^* \to \Sigma^*$ in some polynomial space $S(\ell(\omega))$ on input ω. One can construct some function $\tilde{f} : \mathbb{N}^2 \times \mathbb{R} \to \mathbb{R}$ in $\mathbb{RCD}$ that does the same: we have $\tilde{f}(2^m, 2^{S(\ell(\omega))}, \gamma_{word}(\omega))$ that is at most 2^{-m} far from $\gamma_{word}(f(\omega))$.*

Proof. We denote by $\mathcal{M}$ the Turing machine computing f. Similarly to the arguments in [5], we can state that there exists a function $Exec$ solution of a robust linear discrete ODE (E) that "computes" the execution of $\mathcal{M}$, with C_{init} the initial configuration :

$$(E) : \begin{cases} Exec(2^m, 0, 2^S, C_{init}) = C_{init} \\ \dfrac{\delta Exec(2^m, t, 2^S, C_{init})}{\delta t} = Next(2^m, 2^S, Exec(2^m, t, 2^S, C_{init})) - Exec(2^m, t, 2^S, C_{init}). \end{cases}$$

For any configuration $\overline{C}$ of $\mathcal{M}$, let write $F(\overline{C})) = F(2^m, 2^S, \overline{C}) = Next(2^m, 2^S, \overline{C}) + \overline{C}$, associated to the righthand side of the above discrete ODE. Denoting by $\tilde{C}$ the errorless encoding of the configuration C, from the constructions of [5] (Lemma 35), it is true that if $\left|\overline{C} - \tilde{C}\right| \leq 4^{-(S+2)}$, then $\left|F(\overline{C}) - F(\tilde{C})\right| \leq 4^{-(S+2)}$. F does not change much locally on the space of configuration. Denoting by S the space of $\mathcal{M}$, and replacing m by $m+2S+4$ as in [5], we have $\left|Next(2^m, 2^S, \overline{C}) - \overline{C}\right| \leq 4^{-(S+2)}$. So at each step of the TM, the error is fixed (and bounded). We can then apply the above arguments (Proposition 39) to simulate continuously (E), with some controlled error: all involved quantities have encoding polynomials in the size of the inputs. ◀

4 Proof of Theorem 2

Proof. $\subseteq$: In this direction, we just need to prove that $\mathbb{RCD}$ contains only functions over the reals that are computable in polynomial space. Indeed, then for a function $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^{d'}$ sending every integer $\mathbf{n} \in \mathbb{N}^d$ to the vicinity of some integer of $\mathbb{N}^d$, at a distance less than $1/4$, by approximating its value with precision $1/4$ on its input arguments, and taking the closest integer, we will get a function from the integers to the integers, that corresponds to $\mathrm{DP}(f)$, and that will be in $\mathbf{FPSPACE} \cap \mathbb{N}^{\mathbb{N}}$.

This is indeed the case, since i) all the base functions of $\overline{\mathbb{RCD}}$ are in **FPSPACE**: they are even in **FPTIME**, see [46] ii) $\mathbb{R}^{\mathbb{R}} \cap \mathbf{FPSPACE}$ is stable under *composition*. iii) stability under *robust ODE* follows from Theorem 11.

$\supseteq$: In the other direction, we use an argument similar to [5]: namely, as the function is polynomial space computable, this means that there is a polynomial space computable function $g : \mathbb{N}^{d''+1} \to \{1, 3\}^*$ so that on $\mathbf{m}, 2^n$, it provides the encoding $\overline{\phi(\mathbf{m}, n)}$ of some dyadic $\phi(\mathbf{m}, n)$ with $\|\phi(\mathbf{m}, n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}$ for all $\mathbf{m}$. The problem is then to decode, compute and encode the result to produce this dyadic. More precisely, from Theorem 42, we get $\tilde{g}$ with

$$|\tilde{g}(2^e, 2^{p(max(\mathbf{m},n))}, Decode(2^e, \mathbf{m}, n)) - \gamma_{word}(g(\mathbf{m}, n))| \leq 2^{-e}$$

for some polynomial p corresponding to the time required to compute g, and $e = \max(p(max(\mathbf{m}, n)), n)$. Then we need to transform the value to the correct dyadic: we mean

$$\tilde{\mathbf{f}}(\mathbf{m}, n) = EncodeMul(2^e, 2^t, \tilde{g}(2^e, 2^t, Decode(2^e, \mathbf{m}, n)), 1),$$

where $t = p(max(\mathbf{m}, n))$, $e = \max(p(max(\mathbf{m}, n)), n)$ provides a solution with $\|\tilde{\mathbf{f}}(\mathbf{m}, 2^n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}$. ◀

5 Proof of Theorem 3

Proof. $\subseteq$: To prove that $\overline{\mathbb{RCD}} \subseteq \mathbb{R}^{\mathbb{R}} \cap \mathbf{FPSPACE}$, we only need to add to the previous arguments that $\mathbb{R}^{\mathbb{R}} \cap \mathbf{FPSPACE}$ is also stable under $ELim$.
$\supseteq$: In this direction, we have the same issue as in [5]: the strategy of decoding, working with the Turing machine, and encoding is not guaranteed to work for all inputs. But, we can solve it by using an adaptative barycenter technique as in [5].

We recall the principle here for a function whose domain is $\mathbb{R}$, but it can be generalised to $\mathbb{R}^d$. The idea is to construct some function $\lambda : \mathbb{N}^2 \times \mathbb{R} \to [0,1]$ definable in $\mathbb{RCD}_*$ as in Corollary 31, but with a continuous ODE : Adapting the proof from [5] and using the simulation of ξ in our continuous framework, we can consider $\lambda(2^m, N, x) = \Psi(\Xi(2^{m+1}, N, x - 9/8))$ where $\Psi(x) = \mathcal{C}\text{-}\mathfrak{s}(2^{m+1}, 1/4, 1/2, x)$. In particular, by definition, $\lambda \in \mathbb{RCD}_*$. Thus, by Lemma 30, if $\lambda(2^m, N, x) =_{2^{-m}} 0$, then $\sigma_2(2^m, N, x) =_{2^{-m}} \lfloor x \rfloor$. If $\lambda(2^m, N, x) =_{2^{-m}} 1$, then $\sigma_1(2^m, N, x) =_{2^{-m}} \lfloor x \rfloor$ and if $\lambda(2^m, N, x) \in (0,1)$, then $\sigma_1(2^m, N, x) =_{2^{-m}} \lfloor x \rfloor + 1$ and furthermore $\sigma_2(2^m, N, x) =_{2^{-m}} \lfloor x \rfloor$. So, $\lambda(\cdot, 2^n, x)\mathit{Formula}_1(x, u, M, n) + (1 - \lambda(\cdot, 2^n, n))\mathit{Formula}_2(x, u, M, n)$ and we are sure to be close (up to some bounded error) to some 2^{-m} approximation of a function f. ◀

6 Conclusion

We characterised polynomial space using an algebraically defined class of functions, by using a finite set of basic functions, closure under composition, and a schema for defining functions from robust ODEs. We proposed a concept of robust ODEs solvable in polynomial space. As far as we know, this is an original method for solving ODEs optimising space. It is based on classical constructions such as Savitch's theorem. We extended existing characterisations to a characterisation of functions over the reals and not only over the integers.

The interesting message from our statements is that we provide a clear and simple concept associated with continuous ODEs for space: space corresponds to the precision for numerically stable systems. Hence, compiled with [17], we now know the length of solutions corresponds to time and precision to memory.

Considering future work: We have an algebraically defined class of functions. It remains to know whether this could be transferred at the level of polynomial ODE. We know that the solutions of polynomial ODEs define a very robust class of functions, stable by many operations: sum, products, division, ODE solving, etc: see [35, 15]. Hence, all the base functions we consider in our algebraic class can be turned into polynomial ODEs, by adding some variables. It would be interesting to understand if we could define space complexity directly at the level of polynomial ODEs, using precision.

Recently, another characterisation of **PSPACE** was obtained for polynomial ODEs using rather ad-hoc definitions in [34, 14] and working over a non-compact space. Could our characterisation be put at this simpler class of ODEs, but working with precision? The point is that this characterisation uses unbounded domains, so precision is harder to interpret in their constructions, where the schemas are somehow done to control errors.

Of course, from our statements, adding any **FPSPACE**-computable function over the reals among the base functions would not change the class. However, we did not intend to minimise the number of base functions. For example, $\tanh(t)$ is the solution of the ODE $f' = 1 + f^2$ and $\cos(t)$ can be obtained by the two-dimensional ODE $y_1' = -y_2$, $y_2' = y_1$. Minimising the number of base functions is also left for future work. We believe that even in this setting, proving space complexity corresponds to precision is already significant, independently of this question of a minimal set of base functions.

References

1 Oliver Aberth. The concect of effective method applied to computational problems of linear algebra. *J. Comput. Syst. Sci.*, 5(1):17–25, 1971. doi:10.1016/S0022-0000(71)80004-1.

2 Mikkel Abrahamsen, Linda Kleist, and Tillmann Miltzow. Training neural networks is er-complete. *Advances in Neural Information Processing Systems*, 34:18293–18306, 2021. URL: https://proceedings.neurips.cc/paper/2021/hash/9813b270ed0288e7c0388f0fd4ec68f5-Abstract.html.

3 Daniel Bertschinger, Christoph Hertrich, Paul Jungeblut, Tillmann Miltzow, and Simon Weber. Training fully connected neural networks is ∃r-complete. *CoRR*, abs/2204.01368, 2022. doi:10.48550/arXiv.2204.01368.

4 Manon Blanc and Olivier Bournez. A characterization of polynomial time computable functions from the integers to the reals using discrete ordinary differential equations. In Jérôme Durand-Lose and György Vaszil, editors, *Machines, Computations, and Universality - 9th International Conference, MCU 2022, Debrecen, Hungary, August 31 - September 2, 2022, Proceedings*, volume 13419 of *Lecture Notes in Computer Science*, pages 58–74. Springer, 2022. MCU'22 Best Student Paper Award. doi:10.1007/978-3-031-13502-6_4.

5 Manon Blanc and Olivier Bournez. A characterisation of functions computable in polynomial time and space over the reals with discrete ordinary differential equations: Simulation of turing machines with analytic discrete odes. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France*, volume 272 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.MFCS.2023.21.

6 Manon Blanc and Olivier Bournez. Simulation of turing machines with analytic discrete odes: FPTIME and FPSPACE over the reals characterised with discrete ordinary differential equations. *CoRR*, abs/2307.11747, 2023. doi:10.48550/arXiv.2307.11747.

7 Manon Blanc and Olivier Bournez. Quantifiying the robustness of dynamical systems. relating time and space to length and precision. In Aniello Murano and Alexandra Silva, editors, *32nd EACSL Annual Conference on Computer Science Logic, CSL 2024, February 19-23, 2024, Naples, Italy*, volume 288 of *LIPIcs*, pages 17:1–17:20, Naples, Italy, February 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2024.17.

8 Lenore Blum. *Complexity and real computation*. Springer, 1998. URL: https://www.worldcat.org/oclc/37004484.

9 Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers; NP completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, July 1989.

10 Olivier Bournez, Felipe Cucker, Paulin Jacobé de Naurois, and Jean-Yves Marion. Computability over an arbitrary structure. sequential and parallel polynomial time. In Andrew D. Gordon, editor, *Foundations of Software Science and Computational Structures, 6th International Conference (FOSSACS'2003)*, volume 2620 of *Lecture Notes in Computer Science*, pages 185–199, Warsaw, 2003. Springer. doi:10.1007/3-540-36576-1_12.

11 Olivier Bournez, Felipe Cucker, Paulin Jacobé de Naurois, and Jean-Yves Marion. Implicit complexity over an arbitrary structure: Sequential and parallel polynomial time. *Journal of Logic and Computation*, 15(1):41–58, 2005. doi:10.1093/logcom/exh036.

12 Olivier Bournez and Arnaud Durand. Recursion schemes, discrete differential equations and characterization of polynomial time computation. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th Int Symposium on Mathematical Foundations of Computer Science, MFCS*, volume 138 of *LIPIcs*, pages 23:1–23:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.23.

13 Olivier Bournez and Arnaud Durand. A characterization of functions over the integers computable in polynomial time using discrete ordinary differential equations. *Computational Complexity*, 32(2):7, 2023. doi:10.1007/s00037-023-00240-1.

14 Olivier Bournez, Riccardo Gozzi, Daniel S Graça, and Amaury Pouly. A continuous characterization of PSPACE using polynomial ordinary differential equations. *Journal of Complexity*, 77:101755, August 2023. doi:10.1016/j.jco.2023.101755.

15 Olivier Bournez, Daniel Graça, and Amaury Pouly. On the Functions Generated by the General Purpose Analog Computer. *Information and Computation*, 257:34–57, 2017. doi:10.1016/j.ic.2017.09.015.

16 Olivier Bournez, Daniel S Graça, and Amaury Pouly. On the complexity of solving initial value problems. In *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*, pages 115–121, 2012. doi:10.1145/2442829.2442849.

17 Olivier Bournez, Daniel S. Graça, and Amaury Pouly. Polynomial Time corresponds to Solutions of Polynomial Ordinary Differential Equations of Polynomial Length. *Journal of the ACM*, 64(6):38:1–38:76, 2017. doi:10.1145/3127496.

18 Olivier Bournez, Daniel Silva Graça, and Amaury Pouly. Computing with polynomial ordinary differential equations. *Journal of Complexity*, 36:106–140, 2016. doi:10.1016/j.jco.2016.05.002.

19 Olivier Bournez, Daniel Silva Graça, and Amaury Pouly. Polynomial time corresponds to solutions of polynomial ordinary differential equations of polynomial length: The general purpose analog computer and computable analysis are two efficiently equivalent models of computations. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 109:1–109:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.109.

20 Olivier Bournez and Amaury Pouly. A universal ordinary differential equation. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 116:1–116:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.116.

21 Olivier Bournez and Amaury Pouly. A survey on analog models of computation. In *Handbook of Computability and Complexity in Analysis*, pages 173–226. Springer, 2021.

22 M. S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, 138(1):67–100, 6 February 1995. doi:10.1016/0304-3975(94)00147-B.

23 Vasco Brattka, Peter Hertling, and Klaus Weihrauch. A tutorial on computable analysis. In *New computational paradigms*, pages 425–491. Springer, 2008.

24 Mark Braverman. Hyperbolic Julia sets are poly-time computable. *Electronic Notes in Theoretical Computer Science*, 120:17–30, 2005. doi:10.1016/j.entcs.2004.06.031.

25 Manuel L. Campagnolo, Cristopher Moore, and José Félix Costa. Iteration, inequalities, and differentiability in analog computers. *Journal of Complexity*, 16(4):642–660, 2000. doi:10.1006/jcom.2000.0559.

26 Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018. URL: https://proceedings.neurips.cc/paper/2018/hash/69386f6bb1dfed68692a24c8686939b9-Abstract.html.

27 P. Clote. Computational models and function algebras. In Edward R. Griffor, editor, *Handbook of Computability Theory*, pages 589–681. North-Holland, Amsterdam, 1998. doi:10.1016/s0049-237x(99)80033-0.

28 Peter Clote and Evangelos Kranakis. *Boolean Functions and Computation Models*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2002. doi:10.1007/978-3-662-04943-3.

29 Alan Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam, 1962.

30 Pieter Collins and Daniel S Graça. Effective computability of solutions of ordinary differential equations the thousand monkeys approach. *Electronic Notes in Theoretical Computer Science*, 221:103–114, 2008. doi:10.1016/j.entcs.2008.12.010.

31 Pieter Collins and Daniel S. Graça. Effective Computability of Solutions of Differential Inclusions The Ten Thousand Monkeys Approach. *Journal of Universal Computer Science*, 15(6):1162–1185, 2009. doi:10.3217/jucs-015-06-1162.

32 Kousha Etessami and Mihalis Yannakakis. On the complexity of Nash equilibria and other fixed points. *SIAM Journal on Computing*, 39(6):2531–2597, 2010. doi:10.1137/080720826.

33 François Fages, Guillaume Le Guludec, Olivier Bournez, and Amaury Pouly. Strong turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs. In Jérôme Feret and Heinz Koeppl, editors, *Computational Methods in Systems Biology - 15th International Conference, CMSB 2017, Darmstadt, Germany, September 27-29, 2017, Proceedings*, volume 10545 of *Lecture Notes in Computer Science*, pages 108–127. Springer, 2017. CMSB'2017 Best Paper Award. doi:10.1007/978-3-319-67471-1_7.

34 Riccardo Gozzi. *Analog Characterization of Complexity Classes*. PhD thesis, Instituto Superior Técnico, Lisbon, Portugal and University of Algarve, Faro, Portugal, 2022.

35 Daniel S. Graça. *Computability with Polynomial Differential Equations*. PhD thesis, Instituto Superior Técnico, 2007.

36 Daniel S. Graça, Manuel L. Campagnolo, and Jorge Buescu. Robust simulations of Turing machines with analytic maps and flows. In B. Cooper, B. Loewe, and L. Torenvliet, editors, *Proceedings of CiE'05, New Computational Paradigms*, volume 3526 of *Lecture Notes in Computer Science*, pages 169–179. Springer-Verlag, 2005. doi:10.1007/11494645_21.

37 Daniel S. Graça and José Félix Costa. Analog computers and recursive functions over the reals. *jcomp*, 19(5):644–664, 2003. doi:10.1016/S0885-064X(03)00034-7.

38 Daniel S. Graça, N. Zhong, and J. Buescu. Computability, noncomputability and undecidability of maximal intervals of IVPs. *Transactions of the American Mathematical Society*, 2006. To appear.

39 Daniel S. Graça and Ning Zhong. *Handbook of Computability and Complexity in Analysis*, chapter Computability of Differential Equations, pages 71–99. Springer, 2021.

40 Daniel Silva Graça and Ning Zhong. Robust non-computability and stability of dynamical systems. *CoRR*, abs/2305.14448, 2023. doi:10.48550/arXiv.2305.14448.

41 Morris W. Hirsch, Stephen Smale, and Robert Devaney. *Differential Equations, Dynamical Systems, and an Introduction to Chaos*. Elsevier Academic Press, 2003.

42 Akitoshi Kawamura. Lipschitz continuous ordinary differential equations are polynomial-space complete. In *2009 24th Annual IEEE Conference on Computational Complexity*, pages 149–160. IEEE, 2009. doi:10.1109/CCC.2009.34.

43 Akitoshi Kawamura, Hiroyuki Ota, Carsten Rösnick, and Martin Ziegler. Computational complexity of smooth differential equations. *Logical Methods in Computer Science*, 10, 2014. doi:10.2168/LMCS-10(1:6)2014.

44 Patrick Kidger. On neural differential equations. *CoRR*, abs/2202.02435, 2022. doi:10.48550/arXiv.2202.02435.

45 Ker-I Ko. On the computational complexity of ordinary differential equations. *Information and Control*, 58(1-3):157–194, July/August/September 1983. doi:10.1016/S0019-9958(83)80062-X.

46 Ker-I Ko. *Complexity theory of real functions*, volume 3 of *Progress in theoretical computer science*. Birkhäuser, Boston, 1991.

47 Webb Miller. Recursive function theory and numerical analysis. *Journal of Computer and System Sciences*, 4(5):465–472, October 1970. doi:10.1016/S0022-0000(70)80043-5.

48 John Milnor. On the concept of attractor. *Communications in Mathematical Physics*, 99:177–195, 1985.

49 Amaury Pouly and Daniel Silva Graça. Computational complexity of solving polynomial differential equations over unbounded domains. *Theoretical Computer Science*, 626:67–82, 2016. doi:10.1016/j.tcs.2016.02.002.

50 Marian Boykan Pour-El and J. Ian Richards. A computable ordinary differential equation which possesses no computable solution. *Annals of Mathematical Logic*, 17:61–90, 1979.

51 Cristobal Rojas and Mathieu Sablik. On the algorithmic descriptive complexity of attractors in topological dynamics. *CoRR*, abs/2311.15234, 2023. doi:10.48550/arXiv.2311.15234.

52 Keijo Ruohonen. An effective Cauchy-Peano existence theorem for unique solutions. *International Journal of Foundations of Computer Science*, 7(2):151–160, 1996. doi:10.1142/S0129054196000129.

53 Claude E. Shannon. Mathematical theory of the differential analyser. *Journal of Mathematics and Physics MIT*, 20:337–354, 1941.

54 Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.

55 Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937. Reprinted in Martin Davis. The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions. Raven Press, 1965. doi:10.1112/plms/s2-42.1.230.

56 Bernd Ulmann. *Analog computing*. Walter de Gruyter, 2013.

57 Bernd Ulmann. *Analog and hybrid computer programming*. De Gruyter Oldenbourg, 2020.

58 Veritasum. Future computers will be radically different (analog computing). Youtube video, 2022. URL: https://www.youtube.com/watch?v=GVsUOuSjvcg.

59 Klaus Weihrauch. *Computable Analysis - An Introduction*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2000. doi:10.1007/978-3-642-56999-9.

Function Spaces for Orbit-Finite Sets

Mikołaj Bojańczyk
University of Warsaw, Poland

Lê Thành Dũng (Tito) Nguyễn
École normale supérieure de Lyon, France

Rafał Stefański
University of Warsaw, Poland

Abstract

Orbit-finite sets are a generalisation of finite sets, and as such support many operations allowed for finite sets, such as pairing, quotienting, or taking subsets. However, they do not support function spaces, i.e. if X and Y are orbit-finite sets, then the space of finitely supported functions from X to Y is not orbit-finite. We propose a solution to this problem inspired by linear logic.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Orbit-finite sets, automata, linear types, game semantics

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.130

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version with Appendix*: https://arxiv.org/abs/2404.05265

Funding *Mikołaj Bojańczyk*: Supported by Polish NCN Maestro Grant 2022/46/A/ST6/00072.
Lê Thành Dũng (Tito) Nguyễn: Supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "France 2030" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).
Rafał Stefański: Supported by EPSRC project EP/V040944/1 "Resources in Computation".

Acknowledgements L. T. D. Nguyễn would like to thank Clovis Eberhart and Cécilia Pradic for their ongoing collaboration on "implicit automata for data words" (cf. [18, §1.4.4]), which inspired some ideas in this paper. R. Stefański would like to thank Samson Abramsky for introducing him to the beautiful world of game semantics and suggesting to study it in the context of this paper.

1 Introduction

The class of orbit-finite sets is a class of sets that contains all finite sets and some infinite sets, but still shares some properties with the class of finite sets. The idea, which dates back to Fraenkel–Mostowski models of set theory, is to begin with an infinite set $\mathbb{A}$ of *atoms* or *urelements*. We think of the atoms as being names, such as Eve or John, and atoms can only be compared with respect to equality. Intuitively speaking, an orbit-finite set is a set that can be constructed using the atoms, such as $\mathbb{A}^2$ or $\mathbb{A}^*$, subject to the constraint that there are finitely many elements up to renaming atoms. For example, $\mathbb{A}^2$ is orbit-finite because it has two elements up to renaming atoms, namely (John, John) and (John, Eve), while $\mathbb{A}^*$ is not orbit-finite, because the length of a sequence is invariant under renaming atoms, and there are infinitely many possible lengths. For a survey on orbit-finite sets, see [5].

The notion of orbit-finiteness can be seen as an attempt to find an appropriate notion of finiteness for the nominal sets of Gabbay and Pitts [20]. This attempt emerged from the study of computational models such as monoids [6] and automata [7] over infinite alphabets. Let us illustrate orbit-finiteness using an automaton example.

▶ **Example 1.1** (An orbit-finite automaton)**.** Let $L \subseteq \mathbb{A}^*$ be the language of words in which the letter from the first position does not appear again. This language contains John · Mark · Mark · Eve, because John does not reappear, but it does not contain John · Mark · John. To recognize this language, we can use a deterministic automaton, which uses its state to remember the first letter. In this automaton, the input alphabet is $\Sigma = \mathbb{A}$ and the state space is $Q = 1 + 1 + \mathbb{A}$. In this state space, there are two special states, namely the initial state and a rejecting error state, and furthermore there is one state for each atom $a \in \mathbb{A}$, which represents a situation where the first letter was a but it has not been seen again yet. This state space is infinite; but it is orbit-finite, since each of the three components in Q represents a single orbit. ⌟

Orbit-finite sets have many advantages, which ensure that they are a good setting for automata theory, and discrete mathematics in general. For example, an orbit-finite set can be represented in a finite way [5], which ensures that it becomes meaningful to talk about algorithms that input orbit-finite sets, such as an emptiness check for an automaton. Also, orbit-finite sets are closed under taking disjoint unions and products, which ensures that natural automata constructions, such as the union of two nondeterministic automata or the product of two deterministic automata can be performed.

However, orbit-finite sets do not have all the closure properties of finite sets. Notably missing is the powerset operation, and more generally taking function spaces. For example, if we look at the powerset of $\mathbb{A}$, then this powerset will not be orbit-finite, since already the finite subsets give infinitely many orbits (two finite subsets of different sizes will be in different orbits). The lack of powersets means that one cannot do the subset construction from automata theory, and in particular deterministic and nondeterministic automata are not equivalent. This non-equivalence was known from the early days of automata for infinite alphabets [16], and in fact, some decision problems, such as equivalence, are decidable for deterministic automata but undecidable for nondeterministic automata [17]. Another construction that fails is converting a deterministic automaton into a monoid [6, p. 221]; this is because function spaces on orbit-finite sets are no longer orbit-finite, as explained in the following example.

▶ **Example 1.2** (Failure of the monoid construction)**.** Let us show that the automaton from Example 1.1 cannot be converted into a monoid. The standard construction would be to define the monoid as the subset $M \subseteq Q \to Q$ of all state transformations, namely the subset generated by individual input letters. Unfortunately, this construction does not work. This is because in order for two input words to give the same state transformation, they need to have the same set of letters that appear in them. In particular, the corresponding set of set transformations is not orbit-finite, for the same reason as why the finite powerset is not orbit-finite. Not only does the standard construction not work, but also this language is not recognized by any orbit-finite monoid. ⌟

An attempt to address this problem was provided in [24, 9], based on *single-use* functions. The idea, which originates in linear types and linear logic, is to restrict the functions so that they use each argument at most once. For example, consider the following two functions that input atoms and output Booleans:

$$a \in \mathbb{A} \mapsto \begin{cases} \text{true} & \text{if } a = \text{John} \\ \text{false} & \text{otherwise} \end{cases} \qquad a \in \mathbb{A} \mapsto \begin{cases} \text{true} & \text{if } a = \text{John or } a = \text{Eve} \\ \text{false} & \text{otherwise} \end{cases}$$

Intuitively, the first function is single-use, since it compares the input atom to John only, while the second function is not single-use, since it requires two comparisons, with John and Eve. Here is another example, which shows that the problems with the monoid construction from Example 1.2 could be blamed on a violation of the single-use condition.

▶ **Example 1.3.** Consider the transition function of the automaton in Example 1.1, which inputs a state in $1 + 1 + \mathbb{A}$ together with an input letter from $\mathbb{A}$, and returns a new state. This function is not single use. Indeed, if the state is in $\mathbb{A}$, then the transition function compares it for equality with the input letter; but if the comparison returns true, another copy of the old state must be kept as the new state for future comparisons. ⌟

If one restricts attention to functions that are single-use, much of the usual robustness of automata theory is recovered, with deterministic automata being equivalent to monoids, and both being equivalent to two-way deterministic automata [9].

Despite the success of the single-use restriction in solving automata problems, one would ideally prefer a more principled approach, in which instead of defining single-use automata, we would define a more general object, namely single-use sets and functions. Then the definitions of automata and monoids, as well theorems speaking about them, should arise automatically as a result of suitable closure properties of the sets and functions.

This approach was pursued in [24], in which a *category* of orbit-finite sets with single-use functions was proposed. In this corresponding category, one can represent the set of all single-use functions between two orbit-finite sets X and Y as a new set, call it $X \Rightarrow Y$, which is also orbit-finite. However, as we will see later in this paper, this proposal is not entirely satisfactory, since it fails to account for standard operations that one would like to perform on function spaces, most importantly partial application (currying). In the language of category theory, the proposal from [24] failed to be a monoidal closed category.

Contributions of this paper. The main contribution of this paper is to propose a notion of single-use sets and functions, which extends the proposal from [24], but which is rich enough to be closed under taking function spaces. More formally, we propose a category of single-use functions between orbit-finite sets equipped with additional metadata, and we prove that a suitable quotient of this category is symmetric monoidal closed (Theorem 4.2).

The main idea is to follow the tradition of linear types, and extend the type system from [24] with a new type constructor &. This way we can distinguish between two kinds of products namely $X \otimes Y$ (in [24] denoted as $X \times Y$) and $X \& Y$. Thanks to this distinction, the function space can be built so that the appropriate operations on functions, namely application and currying, can be implemented in a single-use way.

Our proposed category is strongly inspired by linear types, and the proof that it admits function spaces uses a form of *game semantics* – a tool that we take from programming language theory. However, as far as we know, it is an original idea to have an infinite but orbit-finite base type $\mathbb{A}$, and to observe that all constructions in game semantics are consistent with orbit-finiteness. We believe that the resulting category deserves further study, and that it is an interesting and non-trivial example of a category representing "finite" objects.

Some further adjacent developments – such as an alternative solution to the problem of function spaces, using vector spaces of orbit-finite dimension – are presented in Section 7.

2 Sets with atoms

We begin with a brief introduction to orbit-finite sets. For a more detailed treatment, see [5].

Fix for the rest of this paper a countably infinite set $\mathbb{A}$, whose elements will be called atoms. We assume that $\mathbb{A}$ has no other structure except for equality; we will only be interested in notions which are *equivariant*, i.e. invariant under renaming atoms. For example, $\mathbb{A}$ has only two equivariant subsets, namely the empty and full subsets. On the other hand, the set $\mathbb{A}^2$ has four equivariant subsets ($\emptyset$, $\mathbb{A}^2$, the diagonal and its complement). In order to meaningfully speak about equivariant subsets, we must be able to have an action of atom

renamings on the set, as formalized in the following definition. The finite support condition is a technical condition that ensures that the action is well-behaved; it dates back to the work of Fraenkel and Mostowski, and is explained in the survey texts [20, 5].

▶ **Definition 2.1.** *A* set with atoms *is a set X, equipped with an action of the group of atom renamings, subject to the following* finite support condition: *for every $x \in X$ there is a finite set of atoms, such that if an atom renaming π fixes all atoms in the set, then it also fixes x.*

The idea is that a set with atoms is any kind of object that deals with atoms, such as the set $\mathbb{A}^*$ of all words over the alphabet $\mathbb{A}$, or the family of finite subsets of $\mathbb{A}$. Among such objects, we will be interested in those which are "finite". This will be formalized by saying that there are finitely many orbits, as described below. Define the *orbit* of an element x in a set with atoms to be the elements that can be obtained from x by applying some atom renaming. For example, in the set $\mathbb{A}^2$, the orbit of (John, Eve) contains (Mark, John), but it does not contain (John, John). The orbits form a partition of a set with atoms.

▶ **Definition 2.2.** *A set with atoms is called* orbit-finite *if it has finitely many orbits.*

Typical examples of orbit-finite sets are polynomial expressions such as $\mathbb{A}^4 + \mathbb{A}^3 + \mathbb{A}^3 + 1$. Here, 1 represents the set of zero-length sequences; this set has a unique element which is its own orbit. For example, $\mathbb{A}^3$ has five orbits, because there are five possible ways of choosing a pattern of equalities in a sequence of three names. On the other hand, $\mathbb{A}^*$ has infinitely many orbits, since sequences of different lengths are necessarily in different orbits. The family of finite subsets of $\mathbb{A}$ is also not orbit-finite, because subsets of different sizes are in different orbits. The full powerset $\mathsf{P}\mathbb{A}$ is not even a legitimate object in our setting, because some of its elements, i.e. some subsets of $\mathbb{A}$, violate the finite support condition.

2.1 Finiteness of function spaces

As mentioned above, orbit-finite sets can be seen as a certain generalization of finite sets. They allow some, but not all, operations that can usually be done on finite sets. For example, orbit-finite sets are closed under disjoint unions $X + Y$ and products $X \times Y$. Another good property is that an orbit-finite set has only finitely many equivariant subsets (an equivariant subset is one that is invariant under the action of atom permutations). This is because an equivariant subset is a union of some of the finitely many orbits. This accounts for some of the good computational properties of orbit-finite sets.

If X and Y are orbit-finite sets, the set of equivariant functions $X \to Y$ is always literally finite, because it is an equivariant subset of $X \times Y$. However, as we have seen in Example 1.2, when converting an automaton to a monoid, we want to use the *partial applications* $\delta(-, a)$ of the transition function $\delta : Q \times \Sigma \to Q$; while δ is equivariant, in general, $\delta(-, a)$ is not. But it is *finitely supported*, i.e. invariant under all atom renamings that fix some finite set of atoms that depends only on the function. The issue is that the *finitely supported function space* from X to Y is not orbit-finite.

▶ **Example 2.3.** The function $\mathbb{A} \to \mathbb{A}$ below is finitely supported, because it is invariant under all atom renamings that fix Mark, John, Eve and Bill:

$$f(a) = \begin{cases} \text{Mark} & \text{if } a \in \{\text{John, Eve, Bill}\} \\ a & \text{otherwise} \end{cases}$$

If π is the atom renaming that swaps Mark with Adam, then applying it to the function f defined above gives the function $\pi(f)$ that has the same definition (or source code, if a programming intuition is to be followed), except that Mark is used instead of Adam.

In the case of $\mathbb{A} \to \mathbb{A}$, the finitely supported function space is not orbit-finite. Indeed, the condition $a \in \{\text{John, Eve, Bill}\}$ can be replaced by $a \in X$ for any finite set $X \subset \mathbb{A} \setminus \{\text{Mark}\}$ of exceptional values, and two choices of X of different cardinalities will give us two functions in different orbits. ⌟

3 Single-use sets and functions

The lack of function spaces is the problem addressed in this paper. Our solution builds on the idea from [24, Section 2.2], which is to consider only functions that are single-use. This notion, already illustrated intuitively in the introduction, is formalized in Section 3.1, where we show how it almost, but not quite, achieves function spaces. Then, in the rest of this section, we show how function spaces can be recovered by using a more refined type system.

3.1 Single-use functions over polynomial orbit-finite sets

We do not define the single-use functions on all orbit-finite sets, but only a syntactically defined fragment, namely the *polynomial orbit-finite sets*, which are sets that can be generated from 1 and $\mathbb{A}$ using products $\times$ and disjoint unions $+$. Therefore, we will allow orbit-finite sets like $1 + \mathbb{A}^2$, but we will not allow orbit-finite sets like the set of non-repeating pairs $\{(a, b) \mid a \neq b \in \mathbb{A}\}$ or the set of unordered pairs $\{\{a, b\} \mid a \neq b \in \mathbb{A}\}$. It is an open problem to find a satisfactory definition of single-use functions on all orbit-finite sets. (A simple hack is to use a quotienting construction, similar to Section 4, but what we would really like to do is to identify some extra structure in a set, possibly an action of some yet unknown group or semigroup, which enables us to speak about single-use functions.)

Consider two polynomial orbit-finite sets X and Y. To define which functions $X \to Y$ are single-use, we use an inductive definition. We begin with certain functions that are considered single-use, such as the equality test of type $\mathbb{A} \times \mathbb{A} \to 1 + 1$. These functions are called the *prime functions*, and their full list is given in Figure 1. Next, we combine the prime functions into new ones using three combinators. The first, and most important, combinator is function composition. Then, we have two combinators for the two type constructors: if two functions $f_1 : X_1 \to Y_1$ and $f_2 : X_2 \to Y_2$ are single-use, then the same is true for:

$$\underset{(x_1,x_2)\mapsto(f_1(x_1),f_2(x_2))}{f_1 \times f_2 : X_1 \times X_2 \to Y_1 \times Y_2} \qquad \underset{\text{left}(x_1)\mapsto\text{left}(f_1(x_1))}{\underset{\text{right}(x_2)\mapsto\text{right}(f_2(x_2))}{f_1 + f_2 : X_1 + X_2 \to Y_1 + Y_2}}.$$

Crucially, the list of prime single-use functions does not contain the copying function $a \in \mathbb{A} \mapsto (a, a) \in \mathbb{A}^2$. Therefore, an alternative name for the single-use functions is *copyless*. If we added copying, then we would get all finitely supported functions [24, Lemma 23].

▶ **Example 3.1.** Consider the function of type $\mathbb{A}^3 \to \mathbb{A}$ which inputs a triple (a, b, c) of atoms and returns a if c is equal to Mark, and b otherwise. This function is a single-use function. It is obtained by composing the six functions listed below:

Function	Type after function
Append 1.	$\mathbb{A} \times \mathbb{A} \times \mathbb{A} \times 1$
Replace added 1 with Mark using the constant function.	$\mathbb{A} \times \mathbb{A} \times \mathbb{A} \times \mathbb{A}$
Apply the equality test to the last two components.	$\mathbb{A} \times \mathbb{A} \times (1 + 1)$
Distribute.	$\mathbb{A} \times \mathbb{A} \times 1 + \mathbb{A} \times \mathbb{A} \times 1$
Project to first and second components, respectively.	$\mathbb{A} + \mathbb{A}$
Co-diagonal	$\mathbb{A}$

Table 1 The prime single-use functions for polynomial orbit-finite sets X, Y and Z.

Function	Type	Definition
Functions about $\mathbb{A}$		
equality test	$\mathbb{A} \times \mathbb{A} \to 1 + 1$	$a, b \mapsto$ if $a = b$ then true else false
constant a	$1 \to \mathbb{A}$	$x \mapsto a$
identity	$\mathbb{A} \to \mathbb{A}$	$x \mapsto x$
Functions about $\times$		
commutativity of $\times$	$X \times Y \to Y \times X$	$x \times y \mapsto y \times x$
first projection	$X \times Y \to X$	$x \times y \mapsto x$
second projection	$X \times Y \to Y$	$x \times y \mapsto y$
append 1	$X \to X \times 1$	$x \mapsto x \times ()$
associativity of $\times$	$(X \times Y) \times Z \to X \times (Y \times Z)$	$(x \times y) \times z \mapsto x \times (y \times z)$
Functions about $+$		
first co-projection	$X \to X + Y$	$x \mapsto \mathrm{left}(x)$
second co-projection	$Y \to X + Y$	$y \mapsto \mathrm{right}(y)$
co-diagonal	$X + X \to X$	$\begin{cases} \mathrm{left}(x) \mapsto x \\ \mathrm{right}(x) \mapsto x \end{cases}$
commutativity of $+$	$X + Y \to Y + X$	$\begin{cases} \mathrm{left}(x) \mapsto \mathrm{right}(x) \\ \mathrm{right}(y) \mapsto \mathrm{left}(y) \end{cases}$
associativity of $+$	$(X + Y) + Z \to X + (Y + Z)$	$\begin{cases} \mathrm{left}(\mathrm{left}(x)) \mapsto \mathrm{left}(x) \\ \mathrm{left}(\mathrm{right}(y)) \mapsto \mathrm{right}(\mathrm{left}(y)) \\ \mathrm{right}(z) \mapsto \mathrm{right}(\mathrm{right}(z)) \end{cases}$
Distributivity		
$+$ distributes over $\times$	$X \times (Y + Z) \to (X \times Y) + (X \times Z)$	$\begin{cases} x \times (\mathrm{left}(y)) \mapsto \mathrm{left}(x \times y) \\ x \times (\mathrm{right}(z)) \mapsto \mathrm{right}(x \times z) \end{cases}$

To justify this description, one should also show that the six functions are single-use. Appending 1, distributivity and co-diagonal are prime functions. The other three are obtained by combining prime functions using the combinators. For example, the equality test is paired, using the combinator for $\times$, with the identity on the remaining two atoms. ⌟

The design goal of the single-use restriction is to have orbit-finite function spaces. The rough idea is that a single-use function can only use a bounded number of atoms in its source code, which guarantees orbit-finiteness of the function space.

▶ **Example 3.2.** Consider the functions of type $\mathbb{A} \to 1 + 1$, which can be seen as subsets of the atoms, with $1 + 1$ representing the Booleans.

- The finitely supported functions $\mathbb{A} \to 1 + 1$ correspond to the finite or co-finite subsets of $\mathbb{A}$. Therefore, the set of such functions admits an equivariant bijection with a disjoint union of two copies of the finite powerset $\mathsf{P}_{\mathrm{fin}}\mathbb{A}$, in particular it is not orbit-finite.
- There are four possible single-use functions $\mathbb{A} \to 1 + 1$: (a) always return true; (b) always return false; (c) check for equality with some fixed atom a; (d) check for inequality ($\neq$) with some fixed atom a. Therefore, the set of single-use functions admits an equivariant bijection with the orbit-finite set $1 + 1 + \mathbb{A} + \mathbb{A}$. ⌟

The above example shows that the space of single-use functions of some type $X \to Y$ is orbit-finite, and in fact it can be described using a polynomial orbit-finite set. This is true for every choice of polynomial orbit-finite sets X and Y, as proved in [24, Theorem 5], and illustrated in the following example.

▶ **Example 3.3.** Assume that the input type X is some power of the atoms $\mathbb{A}^k$, and the output type Y does not use atoms, e.g. it is $Y = 1 + 1$. The assumption on the input type can be made without loss of generality using distributivity, while the assumption on the output type is a proper restriction, but it will allow us to skip some technical details of the general construction while retaining the important intuitions. We describe below a type that represents all single-use functions from $\mathbb{A}^k$ to Y; we shall denote it by $\mathbb{A}^k \Rightarrow Y$. Note that $\Rightarrow$ is *not* a primitive type constructor in our grammar of types; it is a notation that stands for the inductive construction below.

This type is defined by induction on k. In the base case of $k = 0$ we simply need to give a value from the output type, and therefore $\mathbb{A}^0 \Rightarrow Y$ is the same as Y. Consider now the induction step of $k > 0$. We observe that a single-use function that inputs $\mathbb{A}^k$ must begin with some equality test, and then continue with one of two single-use functions that have fewer arguments (one for the case when the equality test returns true, and one for the other case). This observation leads to the following definition of the type $\mathbb{A}^k \Rightarrow Y$:

$$\underbrace{\coprod_{i \in \{1,\ldots,k\}} \mathbb{A} \times (\mathbb{A}^{k-1} \Rightarrow Y) \times (\mathbb{A}^{k-1} \Rightarrow Y)}_{\substack{\text{starts by comparing } i\text{-th} \\ \text{coordinate to some constant}}} + \underbrace{\coprod_{i,j \in \{1,\ldots,k\}} (\mathbb{A}^{k-2} \to Y) \times (\mathbb{A}^{k-2} \to Y)}_{\substack{\text{starts by comparing the} \\ i\text{-th and } j\text{-th coordinates}}}.$$

Note that the above representation of the function space is not necessarily unique, i.e. the same function can be represented in several different ways. For example, the order in which equality tests are performed will matter for the representation, but might not matter for the function. This is not something that we worry about; it is dealt with in Section 4. ⌟

Unfortunately, the proposal illustrated in Example 3.3 and described in more detail in [24] does not give a satisfactory solution to the problem of function spaces. The problem is that the set of representations $X \Rightarrow Y$ should also support operations on functions. More specifically, we should be able to indicate single-use operations which do the following:

evaluation: a single-use function from $(X \Rightarrow Y) \times X$ to Y which inputs a representation of a function and applies it to an argument;

composition: a single-use function from $(X \Rightarrow Y) \times (Y \Rightarrow Z)$ to $(X \Rightarrow Z)$ which inputs the representations of two functions and returns a representation of their composition;

currying (i.e. partial application): for each single-use function from $X \times Y$ to Z, there should be a single-use function from X to $Y \Rightarrow Z$ which inputs a first argument and returns a representation of the partially applied function.

Only in the presence of all of these operations can we speak of a function space, and the corresponding category can be called closed. (Composition can be obtained through evaluation and currying, so the essential operations are evaluation and currying.) The following example shows that the currying operation is not single-use, and therefore the space of single-use functions as defined in this section is not closed.

▶ **Example 3.4.** Consider the single-use function

$$f : \mathbb{A} \times \mathbb{A} \to 1 + 1 \qquad (a, b) \mapsto \begin{cases} \text{result of test } a = \text{Mark} & \text{if } b = \text{Eve} \\ \text{result of test } a = \text{John} & \text{otherwise.} \end{cases}$$

The currying of this function, is a new function which maps a first argument $a \in \mathbb{A}$ to the partially applied function $f(a, \_)$. This currying is

$$a \mapsto \begin{cases} b \mapsto b = \text{Eve} & \text{if } a = \text{Mark} \\ b \mapsto b \neq \text{Eve} & \text{if } a = \text{John} \\ b \mapsto \text{false} & \text{otherwise} \end{cases}$$

Recall that in Example 3.2 we showed that the space of single-use functions of type $\mathbb{A} \to 1+1$ can be represented as $1 + 1 + \mathbb{A} + \mathbb{A}$. If we use this representation, then the currying of the function f is not single-use, because we need to compare the input atom a to two constants, Mark and John. If we use the representation from Example 3.3, then the corresponding type will be $\mathbb{A} \otimes (1+1)^2$, but the problems with currying will persist. ⌟

For similar reasons, the function space we proposed above does not support function composition either; it cannot be used to convert automata into single-use monoids since the resulting monoid would need to use function composition as its monoid operation.[1]

3.2 Linear types and single-use functions on them

To solve the problems above, we introduce a more refined type system, which is based on linear types. The main idea is to pay more attention to types in Example 3.3, where we describe a single-use function by specifying the first equality test that it makes, and then giving two descriptions of the functions that will be used in each of the two possible outcomes of the equality test. The main observation is that these two outcomes are mutually exclusive, and therefore we intend to use only one of the two descriptions. For this reason, we will use a type constructor & that comes from linear logic. The intended meaning is that an object of type $X \& Y$ consists of two objects, but with the ability to use only one of them. Since linear logic uses $\otimes$ for the product that we have so far denoted by $\times$, we will also follow that convention. Using these two products, the appropriate type for Example 3.3 becomes:

$$\coprod_{i \in \{1,\ldots,k\}} \mathbb{A} \otimes ((\mathbb{A}^{k-1} \Rightarrow Y) \& (\mathbb{A}^{k-1} \Rightarrow Y)) \quad + \quad \coprod_{i,j \in \{1,\ldots,k\}} (\mathbb{A}^{k-2} \to Y) \& (\mathbb{A}^{k-2} \to Y).$$

Under this definition, the problems from Example 3.4 will be solved, at least for the particular type considered in that example. However, by introducing a new type constructor, we will have to redefine the single-use functions, and then we will have to give a representation of functions that allow this new type constructor, without incurring the need to add any other new type constructors. This is what we will do now.

▶ **Definition 3.5** (Linear types). *A* linear type *is any expression constructed from the atomic types* 1 *and* $\mathbb{A}$ *using three*[2] *binary type constructors* $+, \&$ *and* $\otimes$.

---

[1] This problem is solved in [9] and [24] in a different way, namely by showing that every orbit-finite monoid necessarily divides a single-use monoid, using a Krohn-Rhodes construction. However, this construction is difficult and delicate, in particular it does not work for atoms that have more structure than equality alone. In contrast, the proposal that we give in this paper works for other kinds of atoms, as discussed in Section 6.

[2] We set up our type system without using the multiplicative disjunction $\parr$ of linear logic. This is a common practice in linear type systems, as they are often based on intuitionistic linear logic, rather than classical linear logic (see e.g. [1]). It is also worth mentioning that our type system is in fact *affine* [13, §1.2.1], as we are going to allow discarding the unused values of $\mathbb{A}$. However, since there is no risk of confusion, we have decided to use the name *linear types* for the sake of simplicity.

In our linear types, it is only the products that are differentiated, while + comes in only one version. Here is the intuitive explanation of the difference between the two kinds of products, following Girard [13, §1.1.2]. Having a pair $x \otimes y$ is like having the ability of using both components x and y. On the other hand, having a pair $x\&y$ is like having the ability to use one of the two components, at our choice, but not both at once. For example, the input type of the equality test will be $\mathbb{A} \otimes \mathbb{A}$ not $\mathbb{A}\&\mathbb{A}$, since the test will need to consume both arguments. This intuition can only go so far; for example, it is not entirely clear what "our choice" means. We revisit this intuition in Section 5, where game semantics will be used to indicate who makes which choices.

We think of each linear type X as representing a set $[\![X]\!]$, as defined below:

$$[\![1]\!] = 1 \quad [\![\mathbb{A}]\!] = \mathbb{A} \quad [\![X + Y]\!] = [\![X]\!] + [\![Y]\!] \quad [\![X \otimes Y]\!] = [\![X\&Y]\!] = [\![X]\!] \times [\![Y]\!].$$

All sets that arise in this way will be polynomial orbit-finite sets. Note that the two kinds of product represent the same set, namely the set of pairs in the usual set-theoretic sense. However, the two type constructors will be different, because different functions will be allowed to operate on them. As the expression goes, "the proof of the pudding is in the eating"; in this case the pudding is the types and the eating is the functions.

As we did in Section 3.1, the single-use functions will be defined in terms of prime functions and combinators. The combinators are the same, except that instead of $f_1 \times f_2$ we now have two ways of pairing functions, using $\otimes$ and &. The prime functions are inherited from the previous system, with $\times$ understood as $\otimes$, together with a few new functions for &, as described in Table 2. This is summarized in the following definition.

▶ **Definition 3.6** (Single-use functions). *The class of single-use functions is the least class of functions with the following properties:*

1. *It contains the functions from Tables 1 and 2, with $\times$ in Table 1 understood as $\otimes$;*
2. *It is closed under composition, as well as under combining functions using $+$, $\otimes$ and &.*

Table 2 Prime single-use functions that involve &.

Function	Type	Definition
diagonal	$X \to X\&X$	$x \mapsto x\&x$
first projection	$X\&Y \to X$	$x\&y \mapsto x$
second projection	$X\&Y \to Y$	$x\&y \mapsto y$
& distributes over $\otimes$	$X \otimes (Y\&Z) \to (X \otimes Y)\&(X \otimes Z)$	$x \otimes (y\&z) \mapsto (x \otimes y)\&(x \otimes z)$
& distributes over $+$	$X + (Y\&Z) \to (X\&Y) + (X\&Z)$	$\begin{cases} x\&\mathrm{left}(y) \mapsto \mathrm{left}(x\&y) \\ x\&\mathrm{right}(z) \mapsto \mathrm{right}(x\&z) \end{cases}$

Formally speaking, a single-use function consists of an input linear type X, an output linear type Y, and a function between the sets $[\![X]\!]$ and $[\![Y]\!]$ that is generated using the prime functions and combinators from the above definition. As was the case in Section 3.1, all single-use functions are finitely supported. Therefore, one can think of the single-use functions of type $X \to Y$ as being a subset of the set of all finitely supported functions from $[\![X]\!]$ to $[\![Y]\!]$. This subset is strict: as we will see, the space of single-use functions will be orbit-finite, unlike the space of all finitely supported functions. We will be thinking of the single-use functions as a category.

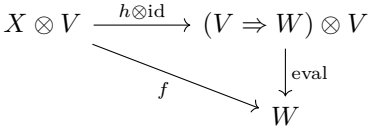
▶ **Definition 3.7** (Category of single-use sets). *The category of single-use sets is:*

1. *The objects are linear types, as per Definition 3.5.*
2. *The morphisms between types X and Y are single-use functions from $[\![X]\!]$ to $[\![Y]\!]$.*

In the very definition of the above category, there is a faithful functor to the category of orbit-finite sets with finitely supported functions. This functor maps objects X to their underlying sets $[\![X]\!]$, which are orbit-finite sets, and it maps morphisms to the corresponding functions. The functions seen to be finitely supported, and the functor is faithful by definition, since the morphisms in Definition 3.7 are defined to be single-use functions.

The main technical result of this paper is that the category of single-use sets has function spaces, as stated in the following theorem. The appropriate product will be $\otimes$, and not &. Since the Cartesian product in our category is & and not $\otimes$, this means that the result we are targeting is symmetric monoidal closed with respect to $\otimes$, and not Cartesian closed. For now, our theorem stops a bit short of saying that the category is monoidal closed, since several different elements of the function space might represent the same function; but we will come back to this in Section 4.

▶ **Theorem 3.8.** *Let V and W be objects (i.e. linear types). There exists an object, denoted by $V \Rightarrow W$, and a morphism (i.e. a single-use function)* $\mathrm{eval} : (V \Rightarrow W) \otimes V \to W$ *with the following property. For every morphism $f : X \otimes V \to W$ there is a (not necessarily unique) morphism $h : X \to (V \Rightarrow W)$ such that the following diagram commutes:*



The above theorem is the main technical contribution of this paper. The difficulty in its proof is finding a representation of the single-use functions that is rich enough to capture all functions, but simple enough to be described by a linear type (in particular, the corresponding set will be orbit-finite). In Section 3.1, when the types did not have "&", we could pull off a relatively simple construction, which was possible mainly due to the strong distributivity rules that allowed converting each type into a normal like $\mathbb{A}^{n_1} + \cdots + \mathbb{A}^{n_\ell}$. In the presence of "&", the distributivity rules are not as strong, and the way in which a single-use program can interact with its input is rather subtle. Our solution is presented in Section 5.

4 Quotienting by partial equivalence relations

A drawback of Theorem 3.8 is that the function space $V \Rightarrow W$ can contain different representations of the same function; this will mean that currying is not unique. To overcome this issue, we use a simple quotient construction, inspired by a classical technique used in realizability semantics of typed λ-calculi (see e.g. [3, Chapter 15]).

Define a *partial equivalence relation* to be a relation that is symmetric and transitive, but not necessarily reflexive. This is the same as (complete) equivalence relation on some subset, hence we may speak of the quotient of a set by a partial equivalence relation (which is an usual quotient of a subset). We will use a partial equivalence on the function space $X \Rightarrow Y$ to: (1) remove objects that do not represent any morphism; (2) identify two objects if they represent the same morphism. After such a quotient, the function space will have unique representations for functions.

▶ **Definition 4.1.** *The* quotiented single-use category *is:*

- *Objects are pairs (linear type X, equivariant partial equivalence relation on $[\![X]\!]$);*
- *The set of morphisms between objects $(X, \sim_X)$ and $(Y, \sim_Y)$ is the quotient of the set of single-use functions from X to Y by*

$$f \sim g \quad :\Longleftrightarrow \quad \forall x, x' \in [\![X]\!],\ x \sim_X x' \Rightarrow f(x) \sim_Y g(x')$$

We can then define the function space object from $(X, \sim_X)$ to $(Y, \sim_Y)$ as $(X \Rightarrow Y, \sim_{X \Rightarrow Y})$ where two elements of $X \Rightarrow Y$ are related by the partial equivalence relation $\sim_{X \Rightarrow Y}$ when the single-use functions they represent are related by the above-defined $\sim$. The quotiented single-use category is also equipped with a tensor product $\otimes$ on its objects.

▶ **Theorem 4.2.** *The quotiented single-use category, equipped with the tensor product $\otimes$, is a monoidal closed category, i.e. it satisfies the conclusions of Theorem 3.8, but, furthermore, the morphism h is unique.*

5 Game semantics

This section is devoted to the proof of Theorem 3.8. To construct the function space $X \Rightarrow Y$, we use game semantics to identify a certain normal form of programs that compute single-use functions. The presentation in this section is self-contained, and does not assume any knowledge of game semantics. We base our notation on [2].[3]

Let us begin with a brief motivation for why game semantics will be useful.

While it is intuitively clear which functions should be allowed as single-use for simple types such as $\mathbb{A} \to 1 + 1$ or $\mathbb{A} \otimes \mathbb{A} \to \mathbb{A} + \mathbb{A}$, these intuitions start to falter when considering more complex types. How can one show that a function is *not* single-use? If one were to use the definition of single-use functions alone, then one would need to rule out any possibility of constructing the function from the primes using the combinators, including constructions that use composition many times, and with unknown intermediate types.

This is the reason why we consider game semantics. It will allow us to give a more principled description of the intuition that pairs of type $X \otimes Y$ can be used on both coordinates, while pairs of type $X \& Y$ can be used on a chosen coordinate only. The idea behind game semantics is to give the description in terms of an interaction between two players:

1. System, who represents the function (we will identify with this player); and
2. Environment, who supplies inputs and requests outputs of the function.

One of the intuitions behind the setup is that if a type $X \& Y$ appears in the input of the function, then it is the System who can choose to use X or Y, while if the type appears in the output, then it is the Environment who makes the choice. (In this paper, we consider functions of first-order types of the form $X \to Y$, where X and Y are linear types that do not use $\to$, and therefore there will be a clear distinction between input and output values.) Before giving a formal definition of game semantics, we give simple example of the interaction.

▶ **Example 5.1.** Consider the two types $X \otimes (Y \& Z)$ and $(X \otimes Y) \& (X \otimes Z)$. Among the prime functions in Table 2, we find distributivity in the direction $\to$, but not in the direction $\leftarrow$. We explain this asymmetry using the interaction between two players System and Environment[4].

Let us first consider the interaction in the direction $\to$. The player Environment begins by requesting an output. Since this output is of type $(X \otimes Y) \& (X \otimes Z)$, this means that Environment can choose to request either of the two types $X \otimes Y$ and $X \otimes Z$. Suppose that Environment requests $X \otimes Y$. Now System needs to react, and produce two elements: one of type X and one of type Y. Both can be obtained from the input; for the second one player System can choose how to resolve the input $Y \& Z$ to get the appropriate value.

[3] Another standard reference for the category of "simple games" upon which we build is [15]. For a recent survey of modern game semantics, see [10].

[4] Observe that in Table 1 the $(+, \otimes)$-distributivity is also only given in one direction. In that case the inverse is, in fact, a single-use function, as it can be constructed from prime functions.

Consider now the interaction in the opposite direction $\leftarrow$. As we will see, System will be unable to react to the behavior of Environment, which demonstrates that there is no distributivity in this direction. The problem is that Environment can begin by requesting an element of type X, since the output type is $X \otimes (Y \& Z)$, while still reserving the possibility to request $Y \& Z$ in the future (because the tensor product $\otimes$ means that both output values need to be produced). To produce this element, System will need choose one of the two coordinates in the input type $(X \otimes Y) \& (X \otimes Z)$, and any of these two choices will be premature, since Environment can then request the opposite choice in the output type. ⌟

As illustrated in the above example, we will use a game to describe the possible behaviours of a function $X \to Y$. The game will be played in an arena, arising from the linear types X and Y, which will tell us what moves are possible for the two players. In each arena, we will be interested in strategies for System, telling us how System should react to Environment's moves. Morally, such a strategy will say how the function reacts to requests in the output type Y and values in the input type X.

Proof scheme for Theorem 3.8. In the remainder of this section, we define the arenas and strategies of our game semantics. Our main Theorem 3.8 will then follow from the two key properties below, that we establish in the technical appendix of the full version.

Representation of single-use functions by strategies: To a strategy in the arena for $X \to Y$, we will assign a single-use function of type $X \to Y$ that it represents. This mapping will be partial, i.e. some ill-behaved strategies will not represent any functions. We will show that the set of strategies in the arena is large enough to represent all single-use functions.

Representation of strategies by a linear type: We shall build a linear type $X \Rightarrow Y$ which can represent all well-formed strategies on the arena for $X \to Y$. In particular, this implies that such strategies form an orbit-finite set. Furthermore, this linear type $X \Rightarrow Y$ will be equipped with single-use evaluation and currying functions, as required.

5.1 Arenas and strategies without constants and equality tests

We begin with a simpler version of the game semantics, in which the arenas and strategies will describe functions that are not allowed to perform equality tests, nor to use constants. These strategies will model functions such as the identity function $\mathbb{A} \to \mathbb{A}$, which directly passes its input to its output, but they will not model the equality test $\mathbb{A} \otimes \mathbb{A} \to 1 + 1$, or the constant functions of type $1 \to \mathbb{A}$. The general idea is to use standard game semantics for linear logic, with an extra feature that we call *register operations*. The register operations will be used to model the way in which atoms are passed from the input to output. For example, in the identity function, Environment will write the input atom into the register, and then System will read the output atom from that register. The following definition of an arena is based on the definition from [2, p. 4], slightly adapted for the context of this paper:

▶ **Definition 5.2** (Arena). *An* arena *consists of:*

1. *A set of* moves*, with each move having an assigned owner, who is either "System" or "Environment", and one of three register operations, which are "none", "read", or "write".*
2. *A set of plays, which a set of finite sequences of moves that is closed under prefixes, and such that in every play, the owner of the first move is Environment, and then the owners alternate between the two players.*

▶ Remark 5.3. In all arenas that we consider, the "read" moves will be owned by System and the "write" moves will be owned by Environment. Therefore, we could simplify the register operations and have just one, called "read/write", whose status is determined by its owner.

In this paper, all the strategies that we consider shall be for player System.

▶ **Definition 5.4.** *A* strategy *in an arena is a subset of plays in the arena, which:*

1. *is closed under prefixes;*
2. *if the strategy contains a play p that ends with a move owned by System, then it also contains all possible plays in the arena that extend p with one move of Environment;*
3. *if the strategy contains a play p that ends with a move owned by Environment, then it contains exactly one play in the arena that extends p with one move of System;*
4. *there is some k such that all plays in the strategy have length at most k;*
5. *every "read" move is directly preceded by a "write" move (in particular a play cannot begin with "read"), and every "write" move is either the last move, or directly succeeded by a "read" move.*

Conditions 2 and 3, which are standard in game semantics (cf. [2, p. 5]), guarantee that the strategy is deterministic and only "ends" when Environment has no moves to play. Let us now comment on the last two conditions, which are not standard.

- The fourth condition is motivated by the idea that we study "finite" types, and there will be no need for unbounded computations[5]
- The last condition will be called the *immediate read condition*. It ensures that there is matching between "read" and "write" moves in plays that do not end with write. Since "write" will always be owned by Environment, the immediate read condition will ensure a matching between "write" and "read" moves.

We now show how to associate to each linear type a corresponding arena, and also how to associate an arena to a function type $X \to Y$ (which is not a linear type in the sense of Definition 3.5) between linear types X and Y. The arenas that we construct so far will not be our final proposal, since the corresponding strategies will not be able to use constants or perform equality tests; this will be fixed in Section 5.2. Before giving a formal definition, we begin with a simple example.

▶ **Example 5.5.** We define an arena for $\mathbb{A} \to \mathbb{A}$. It will be rather impoverished, since the only allowed strategy in it will correspond to the identity function. However, this is consistent with the stage that we are at, where we only consider functions that do not use constants or perform equality tests; for such functions of type $\mathbb{A} \to \mathbb{A}$ the only possibility is the identity.

The arena describes the following interaction between the two players: Environment requests an output, then System requests an input, then Environment grants the input, and finally System grants the output by forwarding the input that was granted. It has 4 moves:

move	owner	register operation
request output	Environment	none
request input	System	none
grant input	Environment	write
grant output	System	read

The plays are defined as all sequences that begin with a move of Environment, alternate between players, use each move only once, and have the following condition: "grant output" can only be played after "request output", and likewise for "grant input" and "request input".

[5] This condition is only meaningful, if there are arenas that admit plays of unbounded length. Since, in this section, all the arenas will be finite this condition will be vacuously true. It will only become relevant in Section 5.2, once we introduce arenas with constants and equality tests.

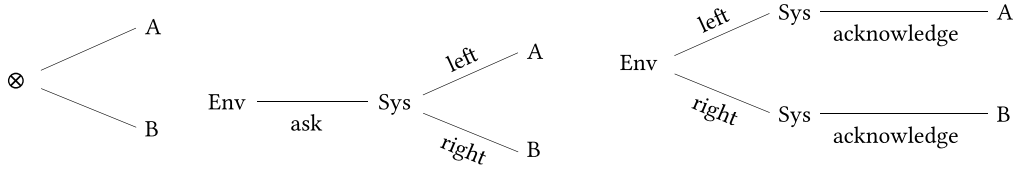


Figure 1 Pictures of the operations $A \otimes B$, $A + B$, $A\&B$ and $A \otimes B$ respectively on arenas. The picture for $\otimes$ is less useful than the next two, since the root node of the tree is not a player, but a node labelled by $\otimes$. The intuition is that the game is played in parallel on both arenas, and therefore a position in it can be visualized as a pair of positions in the two arenas.

A quick inspection of this definition reveals that there is a unique maximal play, where the moves are played in the order from the table, and all other plays are prefixes of this maximal play. Because of the uniqueness of responses, the set of plays is also a strategy. As mentioned at the beginning of this example, the strategy describes the identity function. ⌟

We hope that the above example explains some intuitions about how arenas describe types and strategies describe functions. We now give a formal definition which is compositional: we define arenas for the basic types 1 and $\mathbb{A}$, and then we define operations on arenas that correspond to the type constructors $+$, $\&$, and $\otimes$. We begin with the basic types.

▶ **Definition 5.6** (Arenas for 1 and $\mathbb{A}$)**.**

1. *The arena for the type* 1 *is empty: there are no moves, the only play is the empty sequence.*
2. *The arena for type* $\mathbb{A}$ *has two moves, which must be played one after the other: first player Environment makes a move called "request" that has no register operation, and then player System responds with a move called "grant" that has register operation "read".*

In the above definition, we only described the behaviour of $\mathbb{A}$ when viewed as an output type. To get the input type, where the players are swapped and read is swapped with write, we will use duality, which is another operation on arenas. This operation is defined below, together with other operations that correspond to the type constructors.

▶ **Definition 5.7.** *Let A and B be arenas. We define the following arenas (see also Figure 1):*

$\bar{A}$ *This is called the* dual arena *of A. The moves and plays are the same as in A, except the owners are swapped, and the "read" and "write" register operations are swapped.*

$A + B$ *The moves in this arena are the disjoint union of the moves of A and B, with inherited owners and register operations, plus three extra moves: "ask" owned by Environment, and "left", "right" owned by System. The plays are defined as follows. Player Environment begins with an ask move, then System responds with a left or right move, and the remaining sequence is a play in the arena A or B, depending on whether System chose left or right.*

$A\&B$ *The moves in this arena are the disjoint union of the moves of A and B, with inherited owners and register operations, plus three extra moves: "acknowledge" owned by System, and "left", "right" owned by Environment. The plays are defined as follows. Environment begins by choosing left or right, then System responds with an acknowledge move, and the remaining sequence is a play in either A or B, depending on Environment's choice. (This construction differs slightly from the one from [2, Exercise 1.10] – this is because we want to keep it analogous to the construction for $A + B$.)*

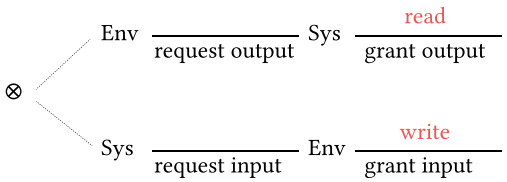
$A \otimes B$ *The moves in this arena are the disjoint union of the moves of A and B, with inherited owners and register operations. A play in this arena is any shuffle of plays in the two arenas A and B. (A shuffle of two words is any word obtained by interleaving them, e.g. shuffles of "abc" and "123" include "a1b23c" and "12ab3c"). By Definition 5.6, we require that the owners of the move alternate in the interleaved sequences. See [2, p.7].*

Equipped with the above definitions, we present our first attempt at assigning arenas to types. In the second item of the following definition, we use the name *library-less*, because our second and final definition of the arena for a function type, as presented in the next section, will be equipped with an extra feature that will be called a library.

▶ **Definition 5.8.** *Let X and Y be linear types.*

- *The* arena for X *is defined by inductively applying the constructions from Definition 5.6 and Definition 5.7 according to the structure of the type.*
- *The* library-less arena for $X \to Y$ *is defined to be (dual of arena of X) $\otimes$ (arena of Y).*

▶ **Example 5.9.** The library-less arena for the identity type $\mathbb{A} \to \mathbb{A}$ is the same as the arena that we explicitly defined in Example 5.5. It can be drawn as the following picture:



As discussed previously, our notion of arenas does not yet take into account some structure on the atoms. This will be fixed in the next section, by modifying the second item in Definition 5.8. On the other hand, the arenas from the first item, for linear types without function types, are already in their final form.

In principle the construction from the second item in Definition 5.8 can be nested, and thus used to assign arenas to higher-order types that can nest $\to$ with the other type constructors. This is how it is usually done in linear logic. However, the construction that we will describe in the next section will be less amenable to nesting, and will use it only to describe functions between types that do not use $\to$.

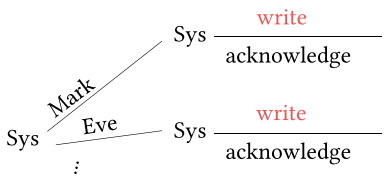
5.2 Arenas and strategies with constants and equality tests

In Section 5.1, we described arenas for functions that did not use the structure of the atoms, i.e. constants and equality tests. We now show how these arenas can be extended to cover this structure. The general idea is to equip the arenas with an extra part, which we call the *library*, that describes the allowed operations on the atoms. (The library as we present it here only contains functions for equality and constants, but in the proof of Theorem 6.2, we will use a library that has other relations beyond equality.)

▶ **Definition 5.10.** *The* library arena *and its parts are defined as follows (cf. Figure 2).*

1. *The* constant choice arena *is the following arena $\mathbb{A} + 1$ moves: first player System chooses an atom, then player Environment plays move with register operation "write".*
2. *The* equality test arena *is an arena which the plays are as follows:*
   a. *first System plays a move with register operation "read";*
   b. *then Environment plays an move with no register operation;*
   c. *then System plays a move with register operation "read";*
   d. *then Environment plays one of two moves, called $=$ and $\neq$, with no register operation.*
3. *The* library arena *is obtained by applying $\otimes$ to infinitely[6] many copies of the constant choice arena and infinitely many copies of the equality test arena.*

The constant choice arena:



The equality test arena:

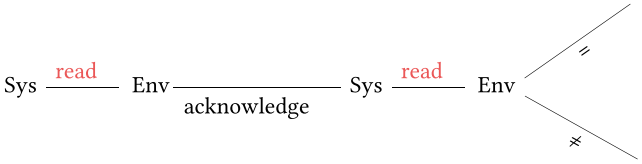


Figure 2 Pictures of the library arenas. We use the convention that the register operations are in red, and the names of the moves, which have no other role than to distinguish them, are in black. Note that the first move in this arena is owned by System, and we assume in Definition 5.2 that the first move is owned by Environment. This is because this arena, like all arenas in Definition 5.10, is not intended to be a stand-alone arena, but only as part of the bigger arena from Definition 5.11 where the first move is indeed owned by Environment.

The library arena is infinite. Taking the tensor product of infinitely many copies of the two arenas ensures that the library arena satisfies the following property, which corresponds to the ! operation from linear logic:

$$\text{library arena} \quad \equiv \quad (\text{constant choice arena}) \otimes (\text{equality test arena}) \otimes (\text{library arena}). \tag{1}$$

In the above, $\equiv$ refers to isomorphism of arenas, which is defined in the natural way: this is a bijection between the moves, which is consistent with the owners, register operations and plays in the expected way. Another property is that the library arena is isomorphic to a tensor product of itself:

$$\text{library arena} \quad \equiv \quad \text{library arena} \otimes \text{library arena}. \tag{2}$$

We are now ready to give the final definition of arenas for functions between linear types, which takes into account the structure of the atoms.

▶ **Definition 5.11** (Function arena). *For linear types X and Y, the arena of $X \to Y$ is*

(library arena) $\otimes$ (dual arena of X) $\otimes$ (arena of Y).

This completes the game semantics of linear types and functions between them. We do not intend to give game semantics for higher-order types, such as functions on functions etc. As a result, we will only be using the dual once, namely for the arena of the input type. Also, note that the read/write operations will be used in a restricted way, as announced in Remark 5.3,

[6] Observe that the shuffle operation from Definition 5.7 can also be used for infinite families of arenas.

namely that the "read" moves will be owned by System and the "write" moves will be owned by Environment. This is because the library arena has this property, the arena for $\mathbb{A}$ has this property, and all operations on arenas that we have defined preserve this property.

As said before, the rest of the proof of Theorem 3.8 takes place in the technical appendix.

6 Beyond equality

So far, we have studied the case when the atoms are equipped with equality only. Consider now a more general case: let $\mathbb{A}$ be any relational structure, i.e. any set equipped with relations (but not functions). For example, we could use the rational numbers with their linear order. Another example would be Presburger arithmetic, i.e. $(\mathbb{N}, +)$. Since we want to have relations only, we think of addition as a ternary relation $x + y = z$. The construction from Section 3 of the single-use category can be generalized to this case; and not only is the definition of the category generic, but the same is true for the proof that function spaces exist.

▶ **Definition 6.1** (Single-use functions over a relational structure). *Let $\mathbb{A}$ be a relational structure. The* single-use category over $\mathbb{A}$ *is defined in the same way as in Definition 3.7, except that the list of prime functions is extended with one prime function $\mathbb{A}^n \to 1 + 1$ for every n-ary relation in the vocabulary of $\mathbb{A}$. (Here, the power $\mathbb{A}^n$ uses $\otimes$.)*

▶ **Theorem 6.2.** *Consider a relational structure $\mathbb{A}$, in which for every $k \in \{0, 1, \ldots\}$ there are finitely many relations of arity k. Then the single-use category over $\mathbb{A}$ satisfies the weak universal property stated in Theorem 3.8.*

In the above statement, "weak" alludes to the non-uniqueness of currying; we could again get a symmetric monoidal closed category by performing the quotient construction of Section 4.

The theorem is proved in the same way as Theorem 3.8. The assumption on the vocabulary is used to ensure that in the corresponding game semantics (cf. Section 5), there are finitely available moves in any given moment, because the vocabulary can only be queried up to the maximal number of atoms in the input, due to the single-use restriction.

Note that this theorem can be applied to any relational structure, including undecidable ones. Clearly, there must be some benefit from assuming that the structure has a decidable first-order theory, which means that there is an algorithm which checks if a first-order sentence is true in the structure. The benefit is that we can check if two morphisms are equal, as expressed in the following theorem, whose assumption applies to structures such as Presburger arithmetic, or the real field $(\mathbb{R}, +, \cdot, \leq)$.

▶ **Theorem 6.3.** *Consider a relational structure $\mathbb{A}$, in which for every $k \in \{0, 1, \ldots\}$ there are finitely many relations in the vocabulary that have arity k. If this structure has a decidable first-order theory, then there is an algorithm for the following problem:*
Input: *Two morphisms, described by expressions using prime functions and combinators.*
Question: *Are they the same morphism?*

7 Further topics

Two-way automata. Theorem 6.3 gives us a reasonable category of single-use functions over a relational structure with a decidable first-order theory. However, the latter property is not the only one needed to ensure that the category is appropriate to automata. If we want to model automata and their decision procedures, we will also need to execute certain fixpoint algorithms, as explained in [5]. In order to allow it, we assume that $\mathbb{A}$ is an *oligomorphic* (see

[5, Definition 3.9]) structure. It is a standard assumption in the theory of sets with atoms, used to ensure that the notion of orbit-finite set is meaningful. Examples of oligomorphic structures include the atoms with equality only, the rational numbers with their linear order, and the Rado graph. Non-examples include Presburger arithmetic and the real field.

▶ **Theorem 7.1.** *Let* $\mathbb{A}$ *be an* oligomorphic *structure with a decidable first-order theory. Then the emptiness problem is decidable for single-use deterministic two-way automata over* $\mathbb{A}$*.*

In the proof of this theorem, we show that the single-use category over an oligomorphic structure $\mathbb{A}$ is *traced* with respect to the coproduct $+$, and use this to model deterministic two-way automata over $\mathbb{A}$. The idea that traced monoidal categories are a natural setting for two-way automata comes from [14].

Orbit-finite dimensional vector spaces. An alternative solution for the problem of function spaces – our central motivation in this paper – is to use vector spaces of orbit-finite dimension. The technical tools for this were developed already in [8]; our contribution here is mainly one of perspective, namely framing it as a symmetric monoidal closed category.

▶ **Theorem 7.2.** *The category of orbit-finitely spanned vector spaces [8, Section VI] is symmetric monoidal closed, with respect to the tensor product.*

▶ Remark 7.3. A similar result was observed in [22, Theorem 3.8], but using the smaller category of vector spaces that admit an orbit-finite basis.

An advantage of the vector space category is its simplicity, and the fact that it is "bigger" in the following sense: The two solutions for function spaces discussed in this paper, namely the single-use solution and the vector space solution, sit on both sides of the classical category of orbit-finite sets, as witnessed by two faithful functors, one from the single-use category to the orbit-finite category, and one from the orbit-finite category to the vector space category. However, there are two limitations of the orbit-finitely spanned vector spaces.

First, the existence of function spaces is dependent on the choice of atoms. Theorem 7.2 works when the atoms have equality only, and it also works when the atoms are equipped with a total order. This is because the dual spaces are orbit-finitely spanned in these cases, as proved in [8, Corollary VI.5]. However, this is no longer the case for other choices of atoms, such as the Rado graph, see [8, Example 9]. This is in contrast to the single-use category, where the existence of function spaces is independent of the choice of atoms.

A second limitation is that unlike the single-use category (cf. Theorem 7.1), the vector space category does not support two-way automata. This is because this category generalizes the orbit-finite category (i.e. it admits a faithful functor from it), and in the orbit-finite category emptiness is undecidable for deterministic two-way automata [17, Theorem 5.3]. This precludes the kind of traced construction that we did in the single-use category. This issue appears already without atoms: the category of finite-dimensional vector spaces is not traced with respect to the sum $\oplus$ of vector spaces.

Related work: categories and λ-calculus. There have been several works using category theory to generalize classical operations on automata, such as the coalgebraic "generalized powerset construction" [23]. The closest to the philosophy that this paper might be the work of Colcombet and Petrişan [11]: it introduces a setting where automata over different categories may be studied and compared (see e.g. [4] for applications). Within this setting, Pradic and the second author have investigated some properties of automata over symmetric monoidal closed categories [18, Sections 1.2.3 and 4.7–4.8].

The latter emerged as part of their research on "implicit automata" [19, 18, 21], which is about relating the expressive power of automata and typed λ-calculi. In [18, Chapter 4], a monoidal closed category of single-use assignments on string-valued registers is built and used to relate a register-based string transducer model to a λ-calculus with linear types. Indeed, symmetric monoidal closed categories are famous for providing denotational semantics for the linear λ-calculus. Similarly, our results here could serve to characterize the languages of words with atoms studied by the first and third author in [9] via some typed λ-calculus.

Conversely, our Theorem 3.8 might also be provable by representing single-use functions as λ-terms (or programs in some richly structured syntactic formalism) instead of strategies over games. Indeed, it is a classical fact that a simple type is inhabited by finitely many linear λ-terms up to β-conversion (when there are no primitive constants), and variations on this fact have been used in the literature to relate automata and λ-calculus [19, 12].

References

1 Samson Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111(1&2):3–57, 1993. doi:10.1016/0304-3975(93)90181-R.

2 Samson Abramsky. Semantics of interaction. In Peter Dybjer and Andrew Pitts, editors, *Proceedings of the 1996 CLiCS Summer School, Isaac Newton Institute*. Cambridge University Press, 1997. arXiv:1312.0121.

3 Roberto M. Amadio and Pierre-Louis Curien. *Domains and Lambda-Calculi*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1998. doi:10.1017/CBO9780511983504.

4 Quentin Aristote. Active learning of deterministic transducers with outputs in arbitrary monoids. In Aniello Murano and Alexandra Silva, editors, *32nd EACSL Annual Conference on Computer Science Logic, CSL 2024, February 19-23, 2024, Naples, Italy*, volume 288 of *LIPIcs*, pages 11:1–11:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.CSL.2024.11.

5 Mikołaj Bojańczyk. Slightly infinite sets. URL: https://www.mimuw.edu.pl/~bojan/paper/atom-book.

6 Mikołaj Bojańczyk. Nominal Monoids. *Theory of Computing Systems*, 53(2):194–222, 2013. doi:10.1007/S00224-013-9464-1.

7 Mikołaj Bojańczyk, Bartek Klin, and Slawomir Lasota. Automata theory in nominal sets. *Logical Methods in Computer Science*, 10(3), 2014. doi:10.2168/LMCS-10(3:4)2014.

8 Mikołaj Bojańczyk, Bartek Klin, and Joshua Moerman. Orbit-finite-dimensional vector spaces and weighted register automata. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470634.

9 Mikołaj Bojańczyk and Rafał Stefański. Single-Use Automata and Transducers for Infinite Alphabets. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 113:1–113:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.113.

10 Pierre Clairambault. *Causal Investigations in Interactive Semantics*. Habilitation à diriger des recherches, Université Aix-Marseille, February 2024. URL: https://theses.hal.science/tel-04523273.

11 Thomas Colcombet and Daniela Petrişan. Automata Minimization: a Functorial Approach. *Logical Methods in Computer Science*, 16(1), March 2020. doi:10.23638/LMCS-16(1:32)2020.

12 Paul Gallot, Aurélien Lemay, and Sylvain Salvati. Linear high-order deterministic tree transducers with regular look-ahead. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 38:1–38:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.38.

13 Jean-Yves Girard. Linear logic: its syntax and semantics. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Notes*, pages 1–42. Cambridge University Press, 1995. doi:10.1017/CBO9780511629150.002.

14 Peter Hines. A categorical framework for finite state machines. *Mathematical Structures in Computer Science*, 13(3):451–480, 2003. doi:10.1017/S0960129503003931.

15 Martin Hyland. Game semantics. In Andrew M. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*, Publications of the Newton Institute, pages 131–184. Cambridge University Press, 1997. doi:10.1017/CBO9780511526619.005.

16 Michael Kaminski and Nissim Francez. Finite-Memory Automata. *Theoretical Computer Science*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.

17 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic*, 5(3):403–435, 2004. doi:10.1145/1013560.1013562.

18 Lê Thành Dũng Nguyễn. *Implicit automata in linear logic and categorical transducer theory*. PhD thesis, Université Paris XIII (Sorbonne Paris Nord), December 2021. URL: https://theses.hal.science/tel-04132636.

19 Lê Thành Dũng Nguyễn and Cécilia Pradic. Implicit automata in typed λ-calculi I: aperiodicity in a non-commutative logic. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 135:1–135:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.135.

20 Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2013. doi:10.1017/CBO9781139084673.

21 Cécilia Pradic and Ian Price. Implicit automata in λ-calculi iii: affine planar string-to-string functions, 2024. arXiv:2404.03985.

22 Michał R. Przybyłek. A note on stone-Čech compactification in zfa, 2024. arXiv:2304.09986.

23 Alexandra Silva, Filippo Bonchi, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Generalizing determinization from automata to coalgebras. *Logical Methods in Computer Science*, 9(1), 2013. doi:10.2168/LMCS-9(1:9)2013.

24 Rafał Stefański. *The single-use restriction for register automata and transducers over infinite alphabets*. PhD thesis, University of Warsaw, 2023.

The Structure of Trees in the Pushdown Hierarchy

Arnaud Carayol
Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France

Lucien Charamond
Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France

Abstract

In this article, we investigate the structure of the trees in the pushdown hierarchy, a hierarchy of infinite graphs having a decidable MSO-theory. We show that a binary complete tree in the pushdown hierarchy must contain at least two different subtrees which are isomorphic. We extend this property to any tree with no leaves and with chains of unary vertices of bounded length. We provided two applications of this result. A first application in formal language theory, gives a simple argument to show that some languages are not deterministic higher-order indexed languages. A second application in number theory shows that the real numbers defined by deterministic higher-order pushdown automata are either rational or transcendental.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Verification by model checking

Keywords and phrases Pushdown hierarchy, Monadic second-order logic, Automatic numbers

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.131

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Acknowledgements The authors are indebted to Didier Caucal for pointing out the notion of (ℓ, b)-tree implicit in the work [3] and starting this work. The authors would like to thanks the reviewers for their work.

1 Introduction

The pushdown hierarchy (also called the Caucal hierarchy) is a robust hierarchy of infinite directed graphs for which monadic second-order logic (MSO) is decidable. These graphs have a countable set of vertices but their edges and nodes are labeled and colored by finite sets. Such infinite graphs with decidable MSO-theories play an important role in automated program verification as they provide a framework in which the model-checking problem for many relevant properties such as termination and safety is decidable. The robustness of the pushdown hierarchy is witnessed by its numerous characterizations and closure properties (we refer the reader to [21] for a survey).

A first characterization of the pushdown hierarchy is via graph transformations following the original idea of Caucal [10]. Every graph in the pushdown hierarchy can be constructed starting from a finite tree by combining two graph transformations that preserve the decidability of MSO-theories namely MSO-interpretations [12] and graph unfolding [13]. As shown in Figure 1, the pushdown hierarchy consists of two intertwined hierarchies: one of classes of trees $(\mathrm{Tree}_n)_{n\geqslant 0}$ and one of classes of graphs $(\mathrm{Graph}_n)_{n\geqslant 0}$. The class Tree_0 contains all finite trees and for $n \geqslant 0$, Graph_n contains all graphs that can be MSO-interpreted in a tree of Tree_n. The trees in Tree_{n+1} are the unfoldings of the graphs in Graph_n. In particular, Graph_0 contains all finite graphs, Tree_1 contains the regular trees and the graphs in Graph_1 are the prefix-recognizable graphs [9]. This hierarchy is closed under most if not all transformations known to preserve the decidability of MSO-theories [8]. It is in particular closed under MSO-transductions [13] and the Muchnik's iteration [22]. More recently, the pushdown hierarchy was shown to be closed under MSO+$\mathrm{U}^{\mathrm{fin}}$-interpretations in [20].

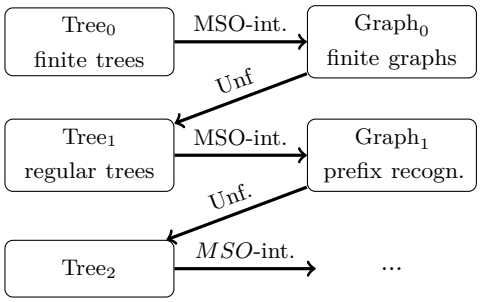


Figure 1 The construction of the pushdown hierarchy using MSO-interpretations and the unfolding operation.

The graphs and trees in the pushdown hierarchy admit several alternative characterizations. The graphs in Graph_n are (up-to isomorphism) the transition graphs of higher-order pushdown automata of order n (see [8]). Higher-order pushdown automata [16] are a generalization of the standard model of pushdown automata which manipulates stacks of stacks at order 2, stacks of stacks of stacks at order 3, ... The deterministic terms in Tree_{n+1} are the solutions of higher-order safe recursion schemes of order-n [10, 15].

These different characterizations make it easy to show that a graph belongs to the pushdown hierarchy. It is rather more complicated to show that a graph does not belong to the pushdown hierarchy. Still the question of characterizing the graphs in this hierarchy has received a lot of attention. There are two main approaches. One approach is to work with higher-order pushdown automata and to develop pumping lemmas for these automata [5, 6, 18, 19]. A second approach is to focus on structural properties of the graphs and to work by induction of the level in the hierarchy using graph transformations [1, 7]. The most involded separation result, namely the separation between trees produced by safe and unsafe recursion schemes was obtained with the first approach [19]. However this approach is much more technically involved and arguably the technical results obtained are less likely to be reusable. In this work, we follow the second approach.

Our starting point is a question asked to the first author by Wolfgang Thomas. He asked whether the pushdown hierarchy contains irrational algebraic numbers such as $\sqrt{2}$. Meaning, does there exists an infinite word (i.e., a unary tree) in the pushdown hierarchy that encodes the expansion of $\sqrt{2}$ in some base $\ell \geqslant 2$? Sadly with this precise statement, the question seems still far out of reach[1]. But the recent work of [3] shows that if we choose to represent a real number in $[0, 1]$ not by its expansion but by a particular tree encoding this expansion, the trees associated with irrational algebraic numbers have strong structural properties: they are deterministic, mostly-complete and all their subtrees are non-isomorphic. The trees representing expansions of real numbers in $[0, 1]$ are implicit in the work of [3] and generalize the definition of automatic real numbers [4]. Indeed they coincide with a generalization of automatic real numbers in which deterministic finite automata are replaced by deterministic higher-order pushdown automata [11].

[1] In this sense, the pushdown hierarchy is known to contain all the rational numbers and all the morphic numbers (and hence all the automatic numbers) in $[0, 1]$. Indeed morphic sequences have been shown to belong to Tree_2 in [10, Proposition 3.2]. Hence the pushdown hierarchy contains expansions of transcendental numbers (see [2]) but is not known to contain the expansions of any irrational algebraic number.

In Section 4 and Section 5, we show that any tree with no leaves and with chains of unary vertices of bounded length in the pushdown hierarchy must contain two different isomorphic subtrees. The main technical ingredient is a precise description of the MSO-interpretations constructing a complete binary tree from a complete binary tree. In Section 6, we give two applications of this result. First, we show how it can be used to show that certain languages such as the language $L_{ww} = \{ww \mid w \in \{0,1\}^*\}$ cannot be accepted by any deterministic higher-order pushdown automaton. Second, we show that a real number in $[0,1]$ represented by a tree in the pushdown hierarchy is either rational or transcendental (i.e., not algebraic). As the real numbers with a tree in Tree_1 are the automatic reals numbers, the result was proved for level 1 in [2]. It was also shown for level 2 in [3].

2 Preliminaries

Notations. Let Σ^* denote the set of all words over the alphabet Σ. We write $u \sqsubseteq v$ if u is a prefix of v and $u \sqsubsetneq v$ if u is a strict-prefix of v. If Σ is equipped with a total order relation, we denote by $<_{\mathrm{lex}}$ the resulting lexicographic ordering on words over Σ.

Infinite graphs and trees. In this article, we consider graphs with countably many vertices with labeled edges and colored vertices. Let Σ be a finite set of edge labels and Θ be a finite set of vertex colors, a graph G labeled by Σ and colored by Θ is a tuple (V, E, C) where V is a countable set of vertices, $E \subseteq V \times \Sigma \times V$ is a set of labeled edges, $C \subseteq \Theta \times V$ is the set of colors. A graph G is deterministic if for all $\sigma \in \Sigma$ and all vertices u, v and v', $u \xrightarrow[G]{\sigma} v$ and $u \xrightarrow[G]{\sigma} v'$ then $v = v'$.

A path π in a graph G from u to v is a sequence $u_0\sigma_0u_1 \cdots \sigma_{n-1}u_n \in V(\Sigma V)^*$ such that $u_0 = u$, $u_n = v$ and $(u_i, \sigma_i, u_{i+1}) \in E$ for $i \in [0, n-1]$. This path is labeled by the word $w = \sigma_0 \cdots \sigma_{n-1}$. We write $u \xrightarrow[G]{w} v$ (or simply $u \xrightarrow{w} v$ if G is clear from the context) to denote the existence of such a path. We extend this notation to a language L over Σ by taking $u \xrightarrow{L} v$ if and only if $u \xrightarrow{w} v$ for some $w \in L$. To improve readability, we write $\longrightarrow$ instead of $\xrightarrow{\Sigma}$, $\longrightarrow^*$ instead of $\xrightarrow{\Sigma^*}$ and $\longrightarrow^+$ instead of $\xrightarrow{\Sigma^+}$.

For two graphs G_1 and G_2, we write $G_1 \sim G_2$ to denote the existence of an isomorphism between G_1 and G_2.

A graph T is a tree if there exists a vertex r called the *root* of T such that there exists a unique path from the root to any vertex. Vertices in a tree are called nodes. A node v is a child of a node u if $u \xrightarrow{\Sigma} v$. In this case, we say that u is the parent of v. A node v is a descendant of u if $u \longrightarrow^* v$ which we also denote by $u \sqsubseteq_T v$. The subtree of a tree T rooted at a node u, denoted by $T_{|u}$, is the tree obtained by restricting T to u and its descendants.

Every node in a deterministic tree is uniquely identified by the label of the unique path from the root to this vertex. As a result, a deterministic tree T labeled by Σ and colored by Θ is determined up-to isomorphism by a mapping from a prefix-closed subset of Σ^* to the subsets of Θ. When reasoning up to isomorphism, we will not distinguish between a deterministic tree and the associated mapping. We always assume that Σ comes with a fixed arbitrary order and hence that the nodes of a deterministic tree can be compared using the lexicographic order.

A complete binary tree is a deterministic tree labeled by $\{0,1\}$ in which every node has two children. The 0-child is called the left-child and the 1-child is called the right-child. For a direction $\gamma \in \{\uparrow, \swarrow, \searrow\}$, we say that v is the γ-successor of u, if v is the parent of u and

$\gamma = \uparrow$ or if v is the left-child (resp. right-child) of u and $\gamma = \swarrow$ (resp. $\gamma = \searrow$). We say that v is in direction γ relative to u, if the γ-successor of u is on the minimal path (ignoring the orientations of the edges) from u to v.

Monadic-second order logic on graphs. We define *monadic-second order logic* (MSO) over graphs with labeled edges and colored nodes as usual. We use lowercase letters $x, y, z, \ldots$ for first order variables and uppercase letters $X, Y, Z, \ldots$ for second order variables. The atomic formulas are $x = y$, $x \in X$, $x \xrightarrow{\sigma} y$ and $\theta(x)$ for σ an edge label and θ a color. MSO-formulas are obtained by applying boolean operators ($\neg$ and $\vee$) and existential quantifiers ($\exists$) over both first and second order variables. To improve readability, we will freely use any definable notion as syntactic sugar: $\forall$, $\Rightarrow$, $X \subseteq Y$, ...

The notion of free variables is defined as usual. We write $\varphi(x_1, \ldots, x_n, X_1, \ldots, X_m)$ when the free variables of a formula φ are among $x_1, \ldots, x_n, X_1, \ldots, X_m$. A closed formula does not have any free variables. For a graph G and a formula $\varphi(x_1, \ldots, x_n, X_1, \ldots, X_m)$, we write $G \models \varphi[u_1, \ldots, u_n, U_1, \ldots, U_m]$ when the graph satisfies the formula if the free variables $x_1, \ldots, x_n, X_1, \ldots, X_m$ are respectively interpreted as $u_1, \ldots, u_n, U_1, \ldots, U_m$ where the u_i's are vertices and U_i's are sets of vertices. The MSO-theory of a graph G is a set of closed formulas satisfied by G. A vertex u of a graph G is MSO-definable in G if there exists a formula $\varphi(x)$ such that u is the only vertex such that $G \models \varphi[u]$. The notion of an MSO-definable set of vertices is defined similarly.

Graph transformations. An *MSO-interpretation* $\mathcal{I}$ (on graphs) is given by a tuple of MSO-formulas $(\delta(x), (\varphi_\sigma(x, y))_{\sigma \in \Sigma}, (\varphi_\theta(x))_{\theta \in \Theta})$ where Σ and Θ are finite sets of labels and colors respectively. An *MSO-recoloring* is a special case of MSO-interpretation which does not erase any vertices (i.e., $\delta(x) = \mathbf{true}$) and preserves all edges (i.e., $\varphi_\sigma(x, y) = x \xrightarrow{\sigma} y$ for $\sigma \in \Sigma$).

An *MSO-transduction* (see [12]) is the composition of a K-copying operation followed by an MSO-interpretation. For a finite set of labels $K = \{k_1, \ldots, k_n\}$, the K-copying operation adds for every vertex u of the graph, fresh vertices $u_1, \ldots, u_n$ as well as edges from u to u_i labeled by k_i for each $i \in [1, n]$.

The *unfolding* of a graph G from a vertex s is the tree denoted by $\mathrm{Unf}(G, s)$ whose vertices consists of all paths in G starting from s and with an edge labeled by a from a path π to a path π' if $\pi' = \pi a t$ for some vertex t. Furthermore a path π, ending at a vertex t of G, is colored in $\mathrm{Unf}(G, s)$ with the same colors as t in G.

3 The pushdown hierarchy

The pushdown hierarchy contains the (possibly infinite) graphs which can be constructed using MSO-interpretations combined with the unfolding operation starting from a finite tree. The pushdown hierarchy consists of two intertwined hierarchies of classes of graphs[2]: one containing trees $(\mathrm{Tree}_n)_{n \in \mathbb{N}}$ and one containing graphs $(\mathrm{Graph}_n)_{n \in \mathbb{N}}$ such that:

- Tree_0 is the class of all finite trees;
- for $n \geqslant 0$, Graph_n is the class of all graphs G such that there exists an MSO-interpretation $\mathcal{I}$ and a tree $T \in Tree_n$ with $G \sim \mathcal{I}(T)$;
- for $n \geqslant 1$, Tree_n is the class of trees such that there exists a graph $G \in \mathrm{Graph}_{n-1}$ and a vertex $u \in G$ such that $T \sim \mathrm{Unf}(G, u)$.

[2] All the graphs we consider are labeled and colored by finite sets: only the set of vertices is infinite.

As both MSO-interpretations and the unfolding (from an MSO-definable vertex) preserve the decidability of MSO-theories [12, 13], it follows that all graphs in the pushdown hierarchy have a decidable MSO-theory [8]. Our main contribution only uses the following closure properties which follow from [8]:

▶ **Theorem 1** ([8]). *The following properties hold:*

1. *For $n \geqslant 0$, for a deterministic tree T in Tree_n and an MSO-recoloring μ, the tree $\mu(T)$ belongs to Tree_n.*
2. *For $n \geqslant 0$, the class Graph_n is closed under MSO-transductions and under restriction to reachable vertices from a given vertex (not necessarily MSO-definable) and Tree_n is closed under taking subtrees.*
3. *For $n \geqslant 1$, every graph $G \in \mathrm{Graph}_n$ is isomorphic to $\mathcal{I}(T)$ for some complete binary tree $T \in \mathrm{Tree}_n$ and some MSO-interpretation $\mathcal{I}$. Furthermore, we can assume that for all nodes $s \neq s'$ of $\mathcal{I}(T)$, s and s' are incomparable for $\sqsubseteq_T$.*

3.1 MSO-interpretations and tree-walking automata

By Property 1 of Theorem 1, every graph in Graph_n can be MSO-interpreted in a complete binary tree. In [7, 8], it is shown that MSO-interpretations applied to deterministic trees can be described using only MSO-recolorings and tree walking automata. To simplify the presentation, we tailor our definitions to complete binary trees (cf. Remark 4).

A *tree-walking automaton* on complete binary trees colored by Θ is a tuple $A = (Q, q_A, F, \Delta)$ where Q is the finite set of states, $q_A \in Q$ is the initial state, $F \subseteq Q \times 2^\Theta$ is the set of accepting conditions and $\Delta \subseteq Q \times 2^\Theta \times \{\uparrow, \swarrow, \searrow\} \times Q$ is the set of transitions. Intuitively, a transition (p, c, γ, q) expresses that if the automaton is in state p on a node u colored by the colors of c, it can move in state q to the γ-successor of u.

A run of a tree-walking automaton A on a complete binary tree T starting from a node u_0 in state q is a finite sequence $q_0 u_0 q_1 u_1 \cdots q_n u_n \in (QV_T)^+$ with $q_0 = q$ and for $i \in [0, n-1]$, there exists a transition $\delta = (q_i, T(u_i), \gamma, q_{i+1}) \in \Delta$ with u_{i+1} the γ-successor of u_i. A run is accepting if $(q_n, T(u_n))$ belongs to F. A tree-walking automaton A *accepts a pair of nodes* (u, v) if there exists an accepting run of A on T from the initial state q_A starting at u and ending in v.

On complete binary trees, tree-walking automata can accept any MSO-definable binary relation provided that the trees are recolored with a suitable MSO-recoloring.

▶ **Proposition 2** ([7, 8]). *For every binary complete tree T and every MSO-formula $\varphi(x, y)$, there exists an MSO-recoloring μ and a tree-walking automaton A_φ which accepts on $\mu(T)$ the pairs of nodes (u, v) such that $T \models \varphi[u, v]$.*

In this article, we need a stronger result in the case when the MSO-formula $\varphi(x, y)$ defines a functional relation (i.e., for each node u, there exists at most one node v such that $T \models \varphi[u, v]$). Under this restriction, we will show that the tree-walking automaton can be chosen to be *deterministic* and *non-backtracking* on T.

A tree-walking automaton is said to be *deterministic* if for all state q and set of colors $c \in 2^\Theta$, $(q, c, \gamma, p) \in \Delta$ and $(q, c, \gamma', p') \in \Delta$ implies that $\gamma = \gamma'$ and $p = p'$. This notion of determinism guarantees that there is at most one run starting from a given node in a given state but it does not forbid the tree-walking automaton from visiting the same node twice. A tree-walking automaton is said to be *non-backtracking* on T if none of its runs on T visits the same node twice. In particular a non-backtracking automaton, when going from u to v, will always follow the shortest path from u to v (ignoring the orientations of the edges).

▶ **Proposition 3.** *For every complete binary tree T and every MSO-formula $\varphi(x, y)$ functional on T, there exist an MSO-recoloring μ and a deterministic and non-backtracking tree-walking automaton A_φ which accepts on $\mu(T)$ the pairs of nodes (u, v) such that $T \models \varphi[u, v]$.*

Proof Sketch. Let A be the non-deterministic tree-walking automaton and μ_A be the MSO-recoloring obtained for $\varphi(x, y)$ in Proposition 2. Our aim is to define a new coloring μ_B that captures the functional behavior of A on T. For this we define, for every node u and every state p of A, $\text{target}(u, p)$ to be the unique node v such that all accepting runs of A on $\mu_A(T)$ starting in state p at u end in v. If no such node v exists, $\text{target}(u, p)$ is undefined. As A accepts a functional relation, if A accepts a pair (u, v) then for the initial state q_A of A, $\text{target}(q_A, u)$ is defined and equal to v.

To define the coloring μ_B, we fix an arbitrary order on the states of A. For each state p of A, we color a node u by the tuple (p, γ, q) if $\text{target}(u, p)$ is defined and equal to v, γ is the direction in $\{\uparrow, \swarrow, \searrow\}$ of v relative to u and q is the smallest state of A such that $\text{target}(u_\gamma, q) = v$ with u_γ the γ-successor of u. Such a state q must exists, as every run from u to v must go through u_γ. This coloring can be defined in MSO as μ_A is MSO-definable.

The deterministic and non-backtracking automaton B has the same states, initial state and acceptance conditions as A. In a state p at a node u colored with a tuple (p, γ, q), the automaton moves in the direction γ to the state q. It is easy to show that B accepts (u, v) on $\mu_B(T)$ if and only if A accepts (u, v) on $\mu_A(T)$ which concludes the proof. ◀

▶ Remark 4. To ease the presentation, we only defined tree-walking automata on binary complete trees but they can be defined to work on general deterministic trees and the results of both Proposition 2 and Proposition 3 generalize to this setting.

As a spin-off result, we obtain a simple proof to an open question of [10, Question b] which asks (when reformulated in the setting of this article) if all deterministic trees in Tree_n can be obtained by replacing general MSO-interpretations by a restricted sub-class called deterministic rational inverse mapping. A *deterministic rational inverse mapping* is an MSO-interpretation in which edges are defined by deterministic tree-walking automata working on deterministic trees (cf. Remark 4) and the colors are obtained by renaming or erasing the existing colors. Only vertices that are source or target of an edge are kept (i.e. $\delta(x) := \exists y, \bigvee_{\sigma \in \Sigma} \varphi_\sigma(x, y) \vee \varphi_\sigma(y, x)$).

By a direct induction on the level of the pushdown hierarchy and using Proposition 3, we obtain the following proposition.

▶ **Proposition 5.** *Every deterministic tree in Tree_n is obtained by a n-fold application of a deterministic inverse rational mapping followed by an unfolding starting with a finite tree.*

4 Trees with no self-similarities in the pushdown hierarchy

A tree is said to have a self-similarity if it contains two different subtrees which are isomorphic. The main contribution of this article is the following theorem.

▶ **Theorem 6.** *Every complete binary tree in the pushdown hierarchy has self-similarities.*

Trees with no self-similarities are called *pure* in this article. To prove Theorem 6, we need to show that the pushdown hierarchy does not contain any pure binary complete tree. Assume toward a contradiction that it does. Let $n_0 \geqslant 0$ denote the smallest level such that either Graph_{n_0} or Tree_{n_0} contain such a tree.

Consider the case where Tree_{n_0} contains some pure complete binary tree T_{pure}. As T_{pure} is infinite, we must have $n_0 \geqslant 1$ and by definition of Tree_{n_0}, the tree T_{pure} is (up-to isomorphism) the unfolding of a graph $G \in \mathrm{Graph}_{n_0-1}$ from some vertex r_G. By Property 2 of Theorem 1, we can assume w.l.o.g. that all the vertices of G are reachable from r_G. As all subtrees of the unfolding of G from r_G are non-isomorphic, every vertex of G must be reachable by exactly one path from r_G. Indeed, if a node s was reachable by two different paths $\pi_1 \neq \pi_2$ from r_G, then the subtrees rooted at π_1 and at π_2 in $\mathrm{Unf}(G, r_G)$ would be isomorphic. This implies that G is a tree. As every tree is isomorphic to its unfolding from its root, the graph G must be isomorphic to T_{pure}. Hence $G \sim T_{\mathrm{pure}}$ is a pure tree in Graph_{n_0-1} which contradicts the definition of n_0.

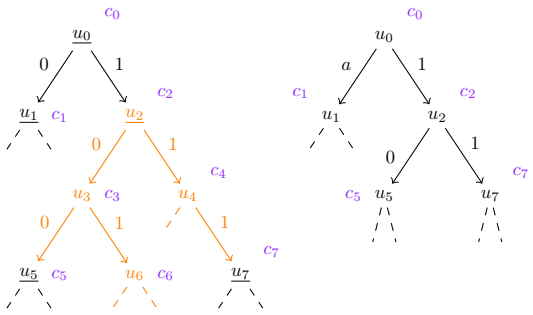


Figure 2 A tree T with a distinguished set of nodes $U = \{u_0, u_1, u_2, u_5, u_7\}$ (on the left) and the induced tree T_U (on the right). The chunk C_{u_2} is highlighted in orange.

It only remains to consider the case where Graph_{n_0} contains a pure binary tree T_{pure} but Tree_{n_0} does not contain any pure complete binary trees. By definition of Graph_{n_0} and by Property 3 of Theorem 1, there must exist a complete binary tree T and an MSO-interpretation $\mathcal{I}$ such that T_{pure} is isomorphic to $\mathcal{I}(T)$. We will show that if an MSO-interpretation is able to produce a pure complete binary tree when applied to some complete binary tree T, then the tree T must be a pure tree *in disguise*.

To formalize what we mean, we recall the notion of embedding in a tree which is illustrated in Figure 2. A set of nodes U in a tree T which contains a unique minimal element induces a tree denoted by T_U. Intuitively this tree is obtained by restricting the tree T to the vertices in U while preserving their colors and inheriting the ancestor relation from T. The label of an edge (u, v) in T_U is the label of the first edge in the unique path from u to v in T. Remark that we present the notion in its most general form but we will mainly work with embedding defining binary complete trees.

▶ **Definition 7.** *An* embedding *in a tree T labeled by Σ is a set of nodes U which contains a unique minimal element for the ancestor relation $\sqsubseteq_T$. This element is called the root of the embedding. This embedding* induces *the tree T_U whose nodes are the elements of U and such that $u \xrightarrow{x} v \in T_U$ if and only if $u \xrightarrow{xw} v \in T$ for some $w \in \Sigma^*$ and there are no $v' \in U$ such that $u \sqsubsetneq v' \sqsubsetneq v$. Moreover the nodes of U have the same colors in T and T_U.*

In Proposition 8, we show that, after applying a suitable MSO-recoloring to the tree T, the resulting tree embeds a pure complete binary tree. This is the main technical contribution of this paper.

▶ **Proposition 8.** *Let T_{pure} and T be two complete binary trees and let $\mathcal{I}$ be an MSO-interpretation such that $T = \mathcal{I}(T)$. If T_{pure} is pure then there exists an MSO-recoloring μ and an embedding $\mathcal{S}_{\mathcal{I}}$ MSO-definable in $\mu(T)$ which induces a pure complete binary tree.*

We defer the proof of this proposition to Section 5, to first show how it can be used to conclude the proof of Theorem 6. By applying Proposition 8, we obtain an MSO-recoloring μ and an embedding $\mathcal{S}_{\mathcal{I}}$ MSO-definable in $\mu(T)$ which induces a pure binary complete tree. The tree $\mu(T)$ belongs to Tree_{n_0} by Property 1 of Theorem 1. To reach a contradiction, we will show that the tree induced by $\mathcal{S}_{\mathcal{I}}$ in $\mu(T)$ belongs to Tree_{n_0}.

▶ **Proposition 9.** *Let $n \geqslant 0$, let T be a deterministic tree in Tree_n and let U be an MSO-definable embedding in T inducing a deterministic tree T_U. The tree T_U belongs to Tree_n.*

Proof Sketch. For $n = 0$, the result is immediate. Hence we assume that $n \geqslant 1$ and furthermore using the closure properties of Theorem 1, we can assume that the root of the embedding is the root of the tree T. Thanks to Property 1 of Theorem 1, we can color the nodes of U in T with a fresh color \$ to obtain a tree $T_{\$}$ also in Tree_n. Consider the following MSO-interpretation $\mathcal{I}$ that produces T_U from $T_{\$}$: it only keeps the vertices colored by \$ and for $x \in \Sigma$, it defines an x-labeled edge between two such vertices u and v if and only if v can be reached from u by a path labeled by a word in $x\Sigma^*$ which does not visit any vertices colored by \$ (except u or v). Furthermore, $\mathcal{I}$ erases the color \$ and preserves all other colors. Clearly, the tree induced by U on T is isomorphic to $\mathcal{I}(T_{\$})$. By definition of Tree_n, the tree $T_{\$}$ is the unfolding of some graph $G \in \mathrm{Graph}_{n-1}$ from some vertex r. The key ingredient is that because $\mathcal{I}(T_{\$})$ is deterministic, the MSO-interpretation $\mathcal{I}$ commutes with the unfolding. It follows that $T_U \sim \mathcal{I}(T_{\$}) \sim \mathrm{Unf}(\mathcal{I}(G), r)$. Hence T_U is isomorphic to the unfolding of the graph $\mathcal{I}(G)$ in Graph_{n-1} and belongs to Tree_n. ◀

By Proposition 9, the tree induced by $\mathcal{S}_{\mathcal{I}}$ in $\mu(T)$ belongs to Tree_{n_0} and by Proposition 8 it is pure which brings the contradiction and conclude the proof of Theorem 6.

Clearly, Theorem 6 does not hold for all trees in the pushdown hierarchy. For instance, the pushdown hierarchy contains infinite unary trees corresponding to non-ultimately-periodic infinite words[3] which are therefore pure. However, Theorem 6 can be extended to any tree that does not contain arbitrary long chains of unary vertices.

▶ **Corollary 10.** *Every tree in the pushdown hierarchy with no leaves and in which the length of all chains of unary vertices is bounded has self-similarities.*

Remark that if we simply ask that there is no infinite chains of unary vertices, Corollary 10 no longer holds. It is possible to construct a pure binary tree in Graph_2 in which all chains of unary vertices are finite. As illustrated in Figure 3, an example of such a tree can be obtained by starting with a copy of the complete binary tree and replacing each node $u = u_1 \cdots u_n \in \{0,1\}^+$ by the finite chain $\bullet \xrightarrow{u_1} \bullet \xrightarrow{u_2} \bullet \cdots \xrightarrow{u_n} \bullet$.

5 Proof of Proposition 8

This section is devoted to proving Proposition 8. For the rest of the section, we fix a complete binary tree T and an MSO-interpretation $\mathcal{I}$ such that $\mathcal{I}(T)$ is a pure complete binary tree. Furthermore for each label $x \in \{0,1\}$, we fix a deterministic and non-backtracking tree-walking automaton A_x with states in Q_x and an MSO-recoloring μ_x such that A_x accepts the pair (u,v) on $\mu_x(T)$ if and only if $u \xrightarrow{x} v$ in $\mathcal{I}(T)$. We assume Q_0 and Q_1 are disjoint and take $Q = Q_0 \uplus Q_1$.

[3] For example, the unary trees representing the morphic words belong to Tree_2 (see [10, Proposition 3.2]).

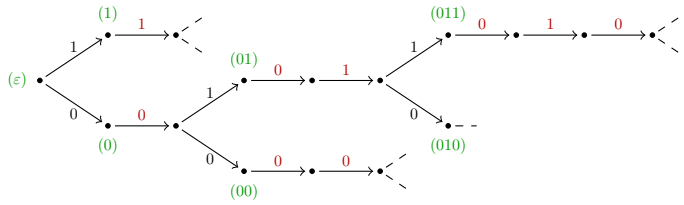


Figure 3 An example of a pure non-complete binary tree in Graph_2. This tree has its edges labeled by $\{0,1\}$. It is obtained by replacing each node $u = u_1 \cdots u_n \in \{0,1\}^+$ of the complete binary tree by a finite line of length $|u|$ in which the i-th edge is labeled by u_i. The corresponding node of the complete binary tree is in green between brackets.

In Subsection 5.1, we define an MSO-definable embedding $\mathcal{S}_\mathcal{I}$ inducing in T a complete binary tree. In Subsection 5.2, we define a MSO-recoloring μ of T. In Subsection 5.3, we show that the binary complete tree induced by $\mathcal{S}_\mathcal{I}$ in $\mu(T)$ is pure.

5.1 Definition of the embedding $\mathcal{S}_\mathcal{I}$

Let us start by remarking that any embedding S in a tree T induces a partition of the nodes of the tree T in regions, we call them chunks. A *chunk* of T is a set of nodes obtained by removing a finite number of subtrees from a subtree of T. Hence a chunk C is described by its *boundary* $(u_0, \ldots, u_n)$ which is a sequence of nodes of T with u_0 the root of the chunk and the $u_1, \ldots, u_n$ are descendants of u_0 which are the pairwise incomparable roots of the subtrees that are removed from $T_{|u_0}$. In other terms, $C = T_{|u_0} \setminus \bigcup_{i\in[1,n]} T_{|u_i}$. We call n the degree of the chunk C.

To every node s in an embedding S, we associate the chunk C_s rooted at s defined by the boundary $(s, s_1, \ldots, s_n)$ where $s_1, \ldots, s_n$ are the children of s in the tree T_S in lexicographical order. This notion is illustrated in Figure 2. The chunks $(C_s)_{s\in S}$ form a partition of the subtree of T rooted at r_S, the root of the embedding S. For our purpose, it is more convenient to obtain a partition of the whole tree T. Hence, we define C_{r_S} by the boundary $(\varepsilon, s_1, \ldots, s_n)$ instead of $(r_S, s_1, \ldots, s_n)$.

The content of a chunk C is the set of nodes kept by the interpretation $\mathcal{I}$ (i.e., $C \cap \mathcal{I}(T)$). We can leverage the fact that $\mathcal{I}(T)$ is a complete binary tree to show that for any chunk with a finite content, the size of the content is bounded by a constant that only depends on the degree of the chunk and on the interpretation $\mathcal{I}$.

▶ **Lemma 11.** *Under the assumptions of this section, for all $m \geqslant 0$, there exists a constant $d \geqslant 0$ depending only on m and on the MSO-interpretation $\mathcal{I}$ such that for each chunk C of degree m, if the content of C is finite, then the size of this content is bounded by d.*

Proof. Let C be a chunk of degree $m \geqslant 0$ with a boundary $(u_0, u_1, \ldots, u_m)$ and let $d := 2(m+1)\max(|Q_0|, |Q_1|)$.

Assume toward a contradiction that C has a finite content U of size $k > d$. As $\mathcal{I}(T)$ is a tree, $\mathcal{I}(T)$ restricted to the content U of C is a forest F with k vertices and hence at most $k-1$ edges. As $\mathcal{I}(T)$ is a complete binary tree, there are at least $2k-(k-1) = k+1$ edges of $\mathcal{I}(T)$ starting in U and ending outside of C. By the pigeonhole principle, at least $\ell > \frac{k+1}{2}$ edges share the same label $x \in \{0,1\}$. Let $(v_1, w_1), \ldots, (v_l, w_l)$ be an enumeration of these edges. As v_i belongs to C and w_i does not, the accepting run of A_x on $\mu_x(T)$ for the pair (v_i, w_i) must cross the boundary of C. By the pigeonhole principle, at least $\frac{\ell}{m+1}$ of these runs cross the boundary of C at the same u_{i_0} with $i_0 \in [0,n]$. As $\frac{\ell}{m+1} > \max(|Q_0|, |Q_1|)$,

there exists two different runs (corresponding to two different edges) reaching u_{i_0} in the same state. This implies that these two different edges must have the same target which contradicts the fact that $\mathcal{I}(T)$ is a tree. ◀

▶ **Definition 12.** *The embedding $\mathcal{S}_\mathcal{I}$ is composed of all nodes u for which both their left and right subtrees contain infinitely many nodes of $\mathcal{I}(T)$.*

Using Lemma 11, we can show that the content of the chunks associated with $\mathcal{S}_\mathcal{I}$ are finite and hence bounded.

▶ **Proposition 13.** *The set $\mathcal{S}_\mathcal{I}$ is an MSO-definable embedding in T defining a complete binary tree. Moreover, for all $s \in \mathcal{S}_\mathcal{I}$, the content of the chunk C_s has size at most d where d is a constant that only depends on $\mathcal{I}$.*

Proof Sketch. For a node u of T, we write $\mathrm{Inf}(u)$ if there are infinitely many nodes of $\mathcal{I}(T)$ below u.

We start by showing that for all node u satisfying $\mathrm{Inf}(u)$, there exist two incomparable descendants v and v' of u such that $\mathrm{Inf}(v)$ and $\mathrm{Inf}(v')$. Assume towards a contradiction that it is not the case for some $u \in T$. Let $v_1, v_2, \dots$ be an enumeration of the descendants v of u for which $\mathrm{Inf}(v)$ holds. For all $i \geqslant 1$, the content of the chunk C_i with boundary (u, v_i) must be finite otherwise it would contain a descendant v of u satisfying $\mathrm{Inf}(v)$ which is incomparable with v_i. By assumption, the v_i's belong to the same infinite branch and hence $T_{|u} = \bigcup_{i \geqslant 1} C_i$. Therefore the size of the content of the C_i's must be unbounded which contradicts Lemma 11.

As the least common ancestor of v and v' belongs to $\mathcal{S}_\mathcal{I}$, we have shown that below every node u satisfying $\mathrm{Inf}(u)$ there exists an element of $\mathcal{S}_\mathcal{I}$. It immediately follows that $\mathcal{S}_\mathcal{I}$ is non-empty and that below each of the two children of an element of $\mathcal{S}_\mathcal{I}$, there exists a unique minimal element in $\mathcal{S}_\mathcal{I}$ (as $\mathcal{S}_\mathcal{I}$ is closed under least common ancestor). This shows that $\mathcal{S}_\mathcal{I}$ induces a complete binary tree. As on deterministic trees the predicate $\mathrm{Inf}(u)$ is MSO-definable, the embedding $\mathcal{S}_\mathcal{I}$ is also MSO-definable.

Using Lemma 11, it only remains to show that content of every chunk C_s for $s \in \mathcal{S}_\mathcal{I}$ is finite. Toward a contradiction, assume that for some $s \in \mathcal{S}_\mathcal{I}$ which is not the root of the embedding, the chunk C_s with boundary (s, s_0, s_1) has an infinite content. There must exist some s' in C_s satisfying $\mathrm{Inf}(s')$ and which is incomparable with both s_0 and s_1. By symmetry, we can assume that s' is in the left subtree of s. In this case, the least common ancestor of s' and s_0 would belong to $\mathcal{S}_\mathcal{I}$ and would be strictly between s and s_0 which would contradict the definition of s_0. The case of the chunk of root of the embedding is treated with similar arguments. ◀

5.2 Definition of the MSO-recoloring μ of the nodes of the embedding

The MSO-recoloring μ will color each vertex $s \in \mathcal{S}_\mathcal{I}$ with a tuple (F_s, π_s, ψ_s) where F_s is the forest obtained by restricting $\mathcal{I}(T)$ to the chunk C_s and π_s and ψ_s are two finite functions describing how to reconnect the forest F_s to the other forests to obtain $\mathcal{I}(T)$.

The forest F_s. For a node $s \in \mathcal{S}_\mathcal{I}$, let $v_1, \dots, v_n$ with $0 \leqslant n \leqslant d$, be an enumeration in lexicographic order of the content of C_s. The nodes of the forest F_s are in $[1, n]$ and for $x \in \{0, 1\}$, $i \xrightarrow{x} j$ in F_s if and only if $v_i \xrightarrow{x} v_j$ in $\mathcal{I}(T)$ and for a color c, i is colored by c in F_s if and only if v_i is colored by c in $\mathcal{I}(T)$.

The partial function π_s. The partial function π_s provides information on the edges that connect the forest F_s to the rest of $\mathcal{I}(T)$. Consider a vertex $i \in [1,n]$ of F_s and a label $x \in \{0,1\}$ such that i has no outgoing x-labeled edge in F_s. Because $\mathcal{I}(T)$ is complete, there is an edge $v_i \xrightarrow{x} v$ in $\mathcal{I}(T)$. The node v is outside of the chunk C_s and hence the unique run of A_x accepting the pair (v_i, v) on $\mu_x(T)$ must exit the chunk through one of its boundaries[4] $\gamma \in \{\text{root}, \text{left}, \text{right}\}$ in a state $q_\gamma \in Q_x$. As A_x follows the minimal path from v_i to v, it crosses the boundary exactly once. The partial function $\pi_s : [1,d] \times \{0,1\} \to \{\text{root}, \text{left}, \text{right}\} \times Q$ associates (i,x) to (γ, q_γ) if the x-labeled edge outgoing from i is missing in F_s and is undefined otherwise.

The partial function ψ_s. The partial function ψ_s provides information on the behavior of the automata A_0 and A_1 when entering the chunk C_s from one of its boundaries. Let $\gamma \in \{\text{root}, \text{left}, \text{right}\}$ be a boundary symbol and let s_γ be the corresponding node in the boundary of C_s. The value of $\psi_s(\gamma, q)$ will provide information on the unique accepting run ρ of A_x on $\mu_x(T)$ starting in state q at s_γ (if it exists). If the run ρ enters C_s and ends at a vertex v_i of the content of C_s, then $\psi_s(\gamma, q)$ is defined to be i. If the run ρ enters C_s and exits through a boundary $\gamma \neq \gamma' \in \{\text{root}, \text{left}, \text{right}\}$ in some state q', then $\psi_s(\gamma, q)$ is defined to be (γ', q'). In all other cases, ψ_s is undefined. As a result ψ_s is a partial mapping $\{\text{root}, \text{left}, \text{right}\} \times Q \to [1,d] \cup (\{\text{root}, \text{left}, \text{right}\} \times Q)$.

▶ **Lemma 14.** *The recoloring μ which colors every $s \in \mathcal{S}_\mathcal{I}$ with the color (F_s, π_s, ψ_s) and leave all other nodes uncolored is MSO-definable.*

5.3 The tree induced by $\mathcal{S}_\mathcal{I}$ on $\mu(T)$ is pure

We start by showing how $\mathcal{I}(T)$ can be reconstructed from the complete binary tree induced by the embedding $\mathcal{S}_\mathcal{I}$ in $\mu(T)$. This tree, denoted E in the following, is essentially the mapping associating to every node $s \in \mathcal{S}_\mathcal{I}$ the tuple (F_s, ψ_s, π_s).

Every node in $\mathcal{I}(T)$ corresponds to a unique node in some forest coloring of E. More precisely, for a node $u \in \mathcal{I}(T)$, the address of u is the unique pair (s,i) such that u is the i-th node in lexicographic order of the content of the chunk C_s. Let $\text{Addr} = \{(s,i) \in \mathcal{S}_\mathcal{I} \times [1,d] \mid E(s) = (F_s, \pi_s, \psi_s) \text{ and } i \in F_s\}$ be the set of all valid addresses.

We now define a deterministic tree R whose nodes are the addresses in Addr and which we will prove to be isomorphic to $\mathcal{I}(T)$ in Proposition 16.

In the tree R, an address $(s,i) \in \text{Addr}$ inherits the colors of the corresponding vertex in F_s. Hence, we define $\mathsf{Colors}(s,i)$ to be $\{\theta \in \Theta \mid E(s) = (F_s, \pi_s, \psi_s) \text{ and } (\theta, i) \in F_s\}$. To define the edges of the tree R, we introduce, for each label $x \in \{0,1\}$, a function Target_x which defines the target of all the x-labeled outgoing edges in R. To define Target_x, we need two auxiliary functions Travel and Follow. These definitions are illustrated in Figure 4.

For $(s,\gamma) \in \mathcal{S}_\mathcal{I} \times \{\text{root}, \text{left}, \text{right}\}$, we define $\mathsf{Travel}(s,\gamma)$ to be equal to (s',γ') if upon leaving C_s by the boundary γ, we enter $C_{s'}$ by its boundary γ'.

For all $s \in \mathcal{S}_\mathcal{I}$, $\gamma \in \{\text{root}, \text{left}, \text{right}\}$ and $q \in Q_x$, $\mathsf{Follow}(s,\gamma,q)$ is recursively defined by (s,j) if $\psi_s(\gamma,q) = j \in [1,d]$ and by $\mathsf{Follow}(s',\gamma'',q')$ if $\psi_s(\gamma,q) = (\gamma',q')$ and $\mathsf{Travel}(s,\gamma') = (s',\gamma'')$. Intuitively, $\mathsf{Follow}(s,\gamma,q)$ gives the address of the end of the run of A_x starting in q at the boundary γ of C_s.

[4] If the boundary of C_s is (s, s_0, s_1), then s correspond to root, s_0 to left and s_1 to right.

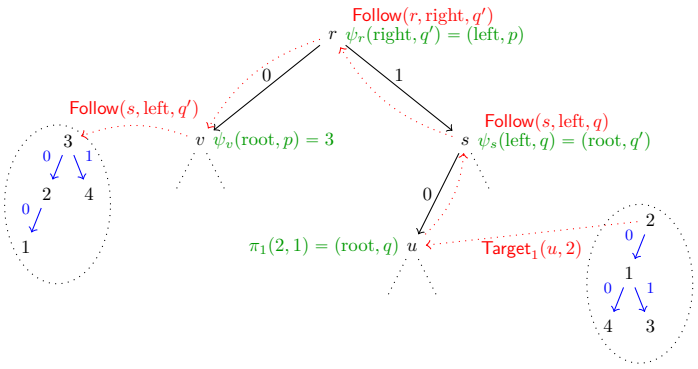


Figure 4 An example of the computation of the function Target. The red path represents the successive recursive calls made when computing $\mathsf{Target}_1(u, 2) = (v, 3)$. As there is no edge from 2 labeled by 1 in F_u, $\mathsf{Target}_1(u, 2)$ calls $\mathsf{Follow}(s, \text{left}, q)$, as $\mathsf{Travel}(u, \text{root}) = (s, \text{left})$ following the function $\pi_1(2, 1) = (\text{root}, q)$. In turn, $\mathsf{Follow}(s, \text{left}, q)$ calls $\mathsf{Follow}(r, \text{right}, q')$ as $\psi_s(\text{left}, q) = (\text{root}, q)$ and $\mathsf{Travel}(s, \text{root}) = (r, \text{right})$. It then proceeds to call $\mathsf{Follow}(s, \text{left}, q')$ as before, which follows $\psi_v(\text{root}, p) = 3$ ending the run on the vertex 3 of the forest F_v.

For $x \in \{0, 1\}$, $\mathsf{Target}_x(s, i)$ is defined to be (s, j) if $i \xrightarrow{x} j$ belongs to F_s otherwise, the x-labeled outgoing edge from i is missing from the forest F_s and $\pi_s(i, x)$ is defined and equal to some $(\gamma, q) \in \{\text{root}, \text{left}, \text{right}\} \times Q_x$ and $\mathsf{Target}_x(s, i)$ is taken to be $\mathsf{Follow}(s', \gamma', q)$ where $(s', \gamma') = \mathsf{Travel}(s, \gamma)$.

▶ **Lemma 15.** *For two addresses (s_u, i_u) and $(s_v, i_v) \in$ Addr respectively corresponding to nodes u and v in $\mathcal{I}(T)$, we have $\mathsf{Target}_x(s_u, i_u) = (s_v, i_v)$ if and only if $u \xrightarrow{x} v$ in $\mathcal{I}(T)$. Furthermore, as the tree-walking automaton A_x is non-backtracking in T, for all $s \in \mathcal{S}_\mathcal{I}$ such that $s \sqsubseteq s_u$ and $s \sqsubseteq s_v$, the recursive calls to* Follow *made when computing $\mathsf{Target}_x(s_u, i_u)$ will all stay below s.*

From this lemma, it follows that the tree R is isomorphic to $\mathcal{I}(T)$.

▶ **Proposition 16.** *The deterministic binary tree R with nodes in* Addr *and defined by* Colors, Target_0 *and* Target_1 *is isomorphic to $\mathcal{I}(T)$.*

We use this reconstruction of $\mathcal{I}(T)$ in E presented above to show that any self-similarity in E would lead to a self-similarity in $\mathcal{I}(T)$ which is impossible by assumption.

▶ **Proposition 17.** *Under the assumptions of this section, for the MSO-recoloring μ and the MSO-definable embedding $\mathcal{S}_\mathcal{I}$ defined previously, the deterministic binary tree E induced by $\mathcal{S}_\mathcal{I}$ on $\mu(T)$ does not have any self-similarities.*

Proof. Assume toward a contradiction that E has a self-similarity and let $s_1 \neq s_2$ be two nodes of E such that $E_{|s_1} \sim E_{|s_2}$. Recall that the nodes of E are the elements of $\mathcal{S}_\mathcal{I}$ and that the ancestor relation $\sqsubseteq_E$ coincide with the ancestor relation $\sqsubseteq_T$ of T on $\mathcal{S}_\mathcal{I} \times \mathcal{S}_\mathcal{I}$. In the following, we say that an address $(t, i) \in$ Addr is below a node s of E if t is descendant of s in E (or equivalently in T).

▷ Claim 18. There exists an address $(t, j) \in$ Addr below s_1 such that all the descendants of (t, j) in the tree R of Proposition 16 have an address below s_1.

Proof. For a label $x \in \{0,1\}$, consider the set X_x of addresses (s,i) below s_1 such that $\mathsf{Target}_x(s,i)$ is not below s_1. By definition of Target_x, there exists $\gamma \in \{\text{left}, \text{right}\}$ such that for all $(s,i) \in X_x$, $\mathsf{Target}_x(s,i)$ must be equal to $\mathsf{Follow}(s_1,\gamma,q)$ for some q. By the pigeonhole principle, X_x is of size at most $|Q_x|$ as otherwise we would have two addresses in X_x with the same value of Target_x which would contradict the fact that R is a tree.

The set Y of addresses (s,i) having a descendant (in R) in the finite set $X_0 \cup X_1$ is itself finite. Let k be the maximal depth of an $s \in E$ such that $(s,i) \in Y$ for some i. As s_1 belongs to $\mathcal{S}_{\mathcal{I}}$, we have by definition of $\mathcal{S}_{\mathcal{I}}$ that there are infinitely many vertices of $\mathcal{I}(T)$ below s_1 in T. Let Z be the corresponding set of addresses. Remark that all these addresses are necessarily below s_1 as if a node $r \in \mathcal{I}(T)$ has an address (s,i) then s is the greatest ancestor of r which belongs to $\mathcal{S}_{\mathcal{I}}$. Hence as Z is infinite, it must contain an element (t,j) with t at depth greater than k in T. By definition of the depth k, this implies that all descendants of (t,j) in R are below s_1. ◁

As $E_{|s_1}$ and $E_{|s_2}$ are isomorphic, there exists a (unique) bijection h that maps every node s of $E_{|s_1}$ to the corresponding node $h(s)$ in $E_{|s_2}$. Let $t' = h(t)$ be the node corresponding to t in $E_{|s_2}$. In particular, t and t' have the same color in R and hence (t',j) is an address in Addr. We claim that $T_1 = R_{|(t,j)}$ is isomorphic to $T_2 = R_{|(t',j)}$ which will bring the contradiction as $R \sim \mathcal{I}(T)$ is assumed to be pure. To see this, consider two descendants (t_1,j_1) and (t_2,j_2) of (t,j) such that $(t_1,j_1) \xrightarrow[R]{x} (t_2,j_2)$ and hence $\mathsf{Target}_x(t_1,j_1) = (t_2,j_2)$. By definition of (t,j), we have that both t_1 and t_2 are below s_1. The recursive calls to Follow made when computing $\mathsf{Target}_x(t_1,j_1)$ stay inside the subtree $E_{|s_1}$ (cf. Lemma 15). As $E_{|s_2}$ is isomorphic to $E_{|s_1}$, we have that $\mathsf{Target}_x(h(t_1),j_1) = (h(t_2),j_2)$. This implies our claim and conclude the proof. ◀

6 Applications

In this section, we leverage the well-known connection between the deterministic trees in the pushdown hierarchy and the trees defined by deterministic higher-order pushdown automata (presented in Subsection 6.1) to give two applications of our main results: one in formal language theory in Subsection 6.2 and one in number theory in Subsection 6.3.

6.1 Higher-order pushdown automata

Higher-order pushdown automata are a generalization of the standard model of pushdown automata. To simplify the presentation, we only define formally higher-order pushdown automata of order 2. We refer the reader to [15] for a definition at all orders.

An order-2 pushdown automaton works on a stack of stacks, called an order-2 stack. We start by defining order-1 and order-2 stacks and the operation to manipulate them. Let Γ be a stack alphabet and let $\perp \notin \Gamma$ be a distinguished bottom of stack symbol. An order-1 stack is a sequence $\perp\gamma_1 \dots \gamma_n \in \perp\Gamma^*$, denoted by $[\,\gamma_1 \dots \gamma_n\,]_1$. The symbol γ_n is the top-most symbol of the stack and $[\,]_1$ is called the empty order-1 stack. An order-2 stack is a non-empty sequence $s_1, \dots, s_n$ of order-1 stacks denoted by $[\,s_1, \dots, s_n\,]_2$. The order-1 stack s_n is the top-most order-1 stack and $[\,[\,]_1\,]_2$ is the empty order-2 stack.

We now define operations on order-2 stacks. For every symbol $\gamma \in \Gamma$, the operation push_γ pushes the symbol γ on the top-most order-1 stack (i.e., $\mathrm{push}_\gamma([\,s_1, \dots, [\,\gamma_1, \dots, \gamma_m\,]_1\,]_2) = [\,s_1, \dots, [\,\gamma_1, \dots, \gamma_m, \gamma\,]_1\,]_2$). The operation pop_1 removes the top-most symbol of the top-most order-1 stack provided that $m \geqslant 1$ (i.e., $\mathrm{pop}_1([\,s_1, \dots, [\,\gamma_1, \dots, \gamma_m\,]_1\,]_2)$ is undefined if

$m = 0$ and is equal to $[\, s_1, \ldots, [\, \gamma_1, \ldots, \gamma_{m-1} \,]_1 \,]_2$ if $m \geqslant 1$). The operation copy_2 copies the top-most order-1 stack (i.e., $\mathrm{copy}_2([\, s_1, \ldots, s_n \,]_2) = [\, s_1, \ldots, s_n, s_n \,]_2$). Finally, the operation pop_2 removes the top-most order-1 stack, if the order-2 stack is not reduced to one order-1 stack (i.e., $\mathrm{pop}_2([\, s_1, \ldots, s_n \,]_2) = [\, s_1, \ldots, s_{n-1} \,]_2$).

An order-2 pushdown automaton P (with ε-transitions) is defined by a tuple (Q, q_0, Σ, Δ) where Q is a finite set of states, $q_0 \in Q$ is the initial state, Σ is a finite set of input symbols, Δ is the set of transitions of the form $(p, \gamma, \sigma, \mathrm{op}, q)$ with $p, q \in Q$, $\gamma \in \Gamma \cup \{\bot\}$, $\sigma \in \Sigma \cup \{\varepsilon\}$ and op an operation in $\{\mathrm{pop}_1, \mathrm{push}_\gamma, \mathrm{pop}_2, \mathrm{copy}_2\}$. We furthermore assume that states of P can be partitioned into $Q_\Sigma \uplus Q_\varepsilon$ such that states in Q_Σ (resp. in Q_ε) are only the source of transitions labeled by Σ (resp. labeled by ε). The automaton is said to be deterministic if for all $q \in Q$, $\gamma \in \Gamma \cup \{\bot\}$ and $\sigma \in \Sigma \cup \{\varepsilon\}$, there exists at most one transition in Δ starting with (q, γ, σ) and if there exists a transition starting with (q, γ) labeled by ε then it is the only transition starting with (q, γ).

A configuration of an order-2 pushdown automaton is a pair (q, s) with $q \in Q$ and s an order-2 stack. For $\sigma \in \Sigma \cup \{\varepsilon\}$, the automaton P induces a relation $\xrightarrow[P]{\sigma}$ between the configurations of P defined by: $(p, s) \xrightarrow[P]{\sigma} (q, s')$ if there exists a transition $(p, \gamma, \sigma, \mathrm{op}, q)$ with $s' = \mathrm{op}(s)$ and γ is the top-most symbol of the top-most order-1 stack of s. From these relations, we can define the relation $\xRightarrow[P]{w}$ for each $w \in \Sigma^*$ in the usual way. If the automaton P is deterministic, we define for all $w \in \Sigma^*$, $\delta_P(w)$ to be the unique configuration (q, s) (if it exists) such that $(q_0, [\,[\,]_1\,]_2) \xRightarrow[P]{w} (q, s)$ with s a stack and $q \in Q_\Sigma$.

If we fix a set $F \subseteq Q_\Sigma$ of final states, the automaton P accepts the language $L(P)$ of words over Σ defined by $L(P) := \{w \in \Sigma^* \mid (q_0, [\,[\,]_1\,]_2) \xRightarrow[P]{w} (q, s) \wedge q \in F\}$. Figure 5 gives an example of a deterministic order-2 pushdown automaton accepting the language $\{1^n 0^n 1^n \mid n \geqslant 1\}$.

If we fix a finite set Θ of vertex colors and a mapping $\mathrm{Col} : Q_\Sigma \to 2^\Theta$, a deterministic order-2 pushdown automaton P defines a deterministic tree $T(P)$ with edges labeled by Σ and with nodes colored by Θ. This tree is given by the partial function T_P from Σ^* to 2^Θ such that for all $w \in \Sigma^*$, $T_P(w) = \mathrm{Col}(q)$ if $\delta_P(w)$ is defined and equal to (q, s) and is undefined otherwise.

The trees defined in this way are the deterministic trees in the pushdown hierarchy.

▶ **Theorem 19** ([10, 15]). *The deterministic trees in* Tree_n *are the trees defined by deterministic higher-order pushdown automata of order* $n - 1$.

6.2 Deterministic vs non-deterministic higher-order pushdown automata

Let Σ be a finite alphabet. For a language $L \subseteq \Sigma^*$ and a word $w \in \Sigma^*$, recall that the left-quotient of L by w, denoted by $w^{-1}L$, is the set $w^{-1}L := \{u \in \Sigma^* \mid wu \in L\}$. We can leverage Theorem 19 to show that if all the left-quotients of a language are different then this language cannot be accepted by any deterministic higher-order pushdown automaton.

▶ **Theorem 20.** *Let* Σ *be a finite alphabet with at least two symbols and* L *be a language over* Σ*. If for all* $w_1 \neq w_2 \in \Sigma^*$*, we have* $w_1^{-1}L \neq w_2^{-1}L$*, the language* L *cannot be accepted by a deterministic higher-order pushdown automaton of any order. In particular, this is the case for the languages:* $\{ww \mid w \in \Sigma^*\}$*,* $\{w \mid w$ *a palindrome in* $\Sigma^*\}$ *and* $\{w^{f(|w|)} \mid w \in \Sigma^*\}$ *for* $f : \mathbb{N} \to \mathbb{N}$ *strictly increasing.*

Proof Sketch. Let Σ be a finite alphabet with a least two symbols and L be a language over Σ. Assume that for all $w_1 \neq w_2 \in \Sigma^*$, $w_1^{-1}L \neq w_2^{-1}L$.

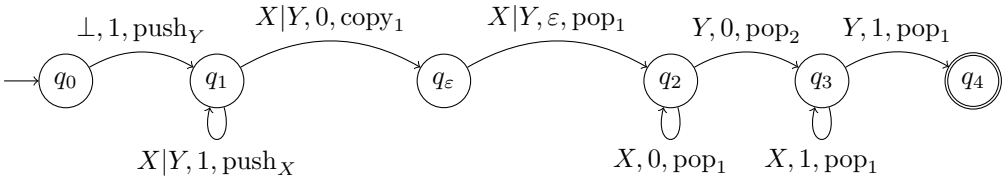


Figure 5 A deterministic higher-order pushdown automaton of order-2 recognizing the language $\{1^n 0^n 1^n \mid n \geqslant 1\}$ with q_0 its initial state and q_4 its final state. Note that the loop on the state q_1 labeled by $X|Y, 1, \text{push}_X$ is actually depicting two transitions: $(q_1, X, 1, \text{push}_X, q_1)$ and $(q_1, Y, 1, \text{push}_X, q_1)$ and similarly for the transitions going from q_1 to q_ε and from q_ε to q_2. For example, the accepting run for the word 110011 is the following $(q_0, [\,[\,]_1\,]_2) \xrightarrow{1} (q_1, [\,[\,Y\,]_1\,]_2) \xrightarrow{1} (q_1, [\,[\,YX\,]_1\,]_2) \xrightarrow{0} (q_\varepsilon, [\,[\,YX\,]_1[\,YX\,]_1\,]_2) \xrightarrow{\varepsilon} (q_2, [\,[\,YX\,]_1[\,Y\,]_1\,]_2) \xrightarrow{0} (q_3, [\,[\,YX\,]_1\,]_2) \xrightarrow{1} (q_3, [\,[\,Y\,]_1\,]_2) \xrightarrow{1} (q_4, [\,[\,]_1\,]_2)$.

Consider the deterministic complete tree $T_L : \Sigma^* \to \{0,1\}$ labeled by Σ and colored by $\{0,1\}$ defined by $T_L(w) = 1$ if and only if $w \in L$. By Theorem 19, T_L belongs to the pushdown hierarchy if and only if L is accepted by a deterministic higher-order pushdown automaton of some order. The assumption of the language L is equivalent to requiring all the subtrees of T_L to be non-isomorphic. Hence T_L does not belong to the pushdown hierarchy by Corollary 10 and our claim follows. ◀

We remark that the language $\{w \mid w \text{ a palindrome in } \Sigma^*\}$ is accepted by a non-deterministic pushdown automaton, the language $\{ww \mid w \in \Sigma^*\}$ is accepted by a non-deterministic order-2 pushdown automaton and $\{w^{|w|} \mid w \in \Sigma^*\}$ is accepted by a non-deterministic order-3 pushdown automaton.

6.3 Real numbers defined by deterministic higher-order pushdown automata

We introduce a generalization of the notion of automatic sequence [4] by replacing the deterministic finite automata used to generate automatic sequences by deterministic higher-order pushdown automata. This generalization follows an approach initiated in [3, 11] and is explicitely mentioned in the conclusion of [11].

Recall that an automatic sequence in base $b \geqslant 2$ is an infinite sequence $\lambda_1\lambda_2\cdots \in \Lambda^\omega$ which is defined using a deterministic finite automaton A over the alphabet $\Sigma_b := \{0, 1, \ldots, b-1\}$. For all $i \geqslant 1$, the automaton outputs the symbol λ_i after having read the decomposition $\langle i \rangle_b$ of i in base b. For the automaton A to output symbols in Λ, we simply associate a symbol in Λ to every state of A using a mapping $\text{Output} : Q \to \Lambda$.

Similarly, a deterministic order-k pushdown automaton P over $\Sigma_b := \{0, \ldots, b-1\}$ equipped with an output function $\text{Output} : Q_\Sigma \to \Lambda$ defines a sequence $\lambda_1\lambda_2\cdots\Lambda^\omega$ if for all $i \geqslant 1$, $\delta_P(\langle i \rangle_b) = (q, s)$ with $\text{Output}(q) = \lambda_i$. If such an automaton exists, the sequence $\lambda_1\lambda_2\cdots \in \Lambda^\omega$ is said to be automatic of order k in base b.

For two bases ℓ and $b \geqslant 2$, a real number in $[0,1]$ is said to be (ℓ, b)-automatic of order-k if it admits an expansion $0.\alpha_1\alpha_2\cdots$ in base ℓ and the sequence $\alpha_1\alpha_2\cdots \in \{0, \ell-1\}^\omega$ is automatic of order-k in base b.

For instance, consider the real number α whose binary expansion is $0.\alpha_1\alpha_2\cdots$ with $\alpha_m = 1$ if $m = 2^{3n} - 2^{2n} + 2^n - 1$ for some $n \geqslant 1$ and $\alpha_m = 0$ otherwise. This number $\alpha = \sum_{n \geqslant 1} 2^{-2^{3n}+2^{2n}-2^n+1}$ is $(2,2)$-automatic of order-2. To see this, remark that for all $n \geqslant 1$,

the binary decomposition of $2^{3n} - 2^{2n} + 2^n - 1$ is $1^n0^n1^n$. Hence an order-2 deterministic pushdown automaton describing the sequence $\alpha_1\alpha_2\dots$ can be obtained by making the deterministic order-2 pushdown automaton presented in Figure 5 complete.

An alternative definition of (ℓ, b)-automatic numbers of order-k can be achieved by considering what we call the (ℓ, b)-tree of the real number. For a real number α with a presentation $0.\alpha_1\alpha_2\cdots$ in base $\ell \geqslant 2$, we can associate its (ℓ, b)-tree T_α which is the deterministic tree whose edges are labeled by $\{0, \dots, b-1\}$ and whose nodes are colored by a unique color in $\{0, \dots, \ell - 1\}$. The domain of the tree T_α is $\{\varepsilon\} \cup [1, \ell - 1][0, \ell - 1]^*$, for all $i \geqslant 1$, $T_\alpha(\langle i \rangle_b) = \alpha_i$ and by convention, $T_\alpha(\varepsilon)$ is taken to be 0. Thanks to Theorem 19, a real number α is (ℓ, b)-automatic of order k if and only if it admits an (ℓ, b)-tree in Tree_{k+1}.

The notion of (ℓ, b)-tree is implicit in the work of [3] where the authors use it to give a sufficient condition for a irrational number to be transcendental.

▶ **Theorem 21** ([3]). *Let $\ell \geqslant 2$, $b \geqslant 2$, $0 \leqslant \alpha \leqslant 1$ a real number and let T_α be an (ℓ, b)-tree for α. If T_α has self-similarities then α is either rational or transcendental.*

By Corollary 10, it immediately follows that:

▶ **Corollary 22.** *For all $\ell \geqslant 2$ and all $b \geqslant 2$, the (ℓ, b)-automatic real number of order k are either rational or transcendental.*

7 Conclusion

In this article, we have shown that every tree in the pushdown hierarchy with no leaves and with chains of unary vertices of bounded length must contain self-similarities. This in particular implies that the trees produced by safe recursion schemes have this property. It is an ongoing work to generalize this property to general unsafe recursion schemes [17]. This would in particular prove that collapsible pushdown automata, a generalization of higher-order pushdown automata cannot be used to generate irrational algebraic numbers [14].

References

1 Luca Aceto, Arnaud Carayol, Zoltán Ésik, and Anna Ingólfsdóttir. Algebraic synchronization trees and processes. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 30–41. Springer, 2012. doi:10.1007/978-3-642-31585-5_7.

2 Boris Adamczewski and Yann Bugeaud. On the complexity of algebraic numbers I. Expansions in integer bases. *Annals of Mathematics*, 165(2):547–565, 2007.

3 Boris Adamczewski, Julien Cassaigne, and Marion Le Gonidec. On the computational complexity of algebraic numbers : the Hartmanis-Stearns problem revisited. *Transactions of the American Mathematical Society*, 373:3085–3115., 2020.

4 Jean-Paul Allouche and Jeffrey O. Shallit. *Automatic Sequences - Theory, Applications, Generalizations*. Cambridge University Press, 2003. URL: http://www.cambridge.org/gb/knowledge/isbn/item1170556/?site_locale=en_GB.

5 Achim Blumensath. On the structure of graphs in the Caucal hierarchy. *Theor. Comput. Sci.*, 400(1-3):19–45, 2008. doi:10.1016/J.TCS.2008.01.053.

6 Achim Blumensath. Erratum to "On the structure of graphs in the Caucal hierarchy" [Theoret. Comput. Sci 400 (2008) 19-45]. *Theor. Comput. Sci.*, 475:126–127, 2013. doi:10.1016/J.TCS.2012.12.044.

7 Laurent Braud and Arnaud Carayol. Linear orders in the pushdown hierarchy. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 88–99. Springer, 2010. doi:10.1007/978-3-642-14162-1_8.

8 Arnaud Carayol and Stefan Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *FSTTCS 2003: Foundations of Software Technology and Theoretical Computer Science, 23rd Conference, Mumbai, India, December 15-17, 2003, Proceedings*, volume 2914 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2003.

9 Didier Caucal. On infinite transition graphs having a decidable monadic theory. In Friedhelm Meyer auf der Heide and Burkhard Monien, editors, *Automata, Languages and Programming, 23rd International Colloquium, ICALP96, Paderborn, Germany, 8-12 July 1996, Proceedings*, volume 1099 of *Lecture Notes in Computer Science*, pages 194–205. Springer, 1996. doi:10.1007/3-540-61440-0_128.

10 Didier Caucal. On infinite terms having a decidable monadic theory. In Krzysztof Diks and Wojciech Rytter, editors, *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*, volume 2420 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 2002. doi:10.1007/3-540-45687-2_13.

11 Didier Caucal and Marion Le Gonidec. Context-free sequences. In Gabriel Ciobanu and Dominique Méry, editors, *Theoretical Aspects of Computing - ICTAC 2014 - 11th International Colloquium, Bucharest, Romania, September 17-19, 2014. Proceedings*, volume 8687 of *Lecture Notes in Computer Science*, pages 259–276. Springer, 2014. doi:10.1007/978-3-319-10882-7_16.

12 Bruno Courcelle. Monadic second-order definable graph transductions: A survey. *Theor. Comput. Sci.*, 126(1):53–75, 1994. doi:10.1016/0304-3975(94)90268-2.

13 Bruno Courcelle and Igor Walukiewicz. Monadic second-order logic, graph coverings and unfoldings of transition systems. *Ann. Pure Appl. Log.*, 92(1):35–62, 1998. doi:10.1016/S0168-0072(97)00048-1.

14 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. *ACM Trans. Comput. Log.*, 18(3):25:1–25:42, 2017. doi:10.1145/3091122.

15 Teodor Knapik, Damian Niwiński, and Paweł Urzyczyn. Higher-order pushdown trees are easy. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002. doi:10.1007/3-540-45931-6_15.

16 A. N. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 12:38–43, 1976.

17 C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 81–90. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.38.

18 Paweł Parys. A pumping lemma for pushdown graphs of any level. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 54–65. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPICS.STACS.2012.54.

19 Paweł Parys. On the expressive power of higher-order pushdown systems. *Log. Methods Comput. Sci.*, 16(3), 2020. URL: https://lmcs.episciences.org/6723.

20 Paweł Parys. The Caucal hierarchy: Interpretations in the (W)MSO+U logic. *Inf. Comput.*, 286:104782, 2022. doi:10.1016/J.IC.2021.104782.

21 Wolfgang Thomas. Constructing infinite graphs with a decidable mso-theory. In Branislav Rovan and Peter Vojtás, editors, *Mathematical Foundations of Computer Science 2003, 28th International Symposium, MFCS 2003, Bratislava, Slovakia, August 25-29, 2003, Proceedings*, volume 2747 of *Lecture Notes in Computer Science*, pages 113–124. Springer, 2003. doi:10.1007/978-3-540-45138-9_6.

22 Igor Walukiewicz. Monadic second-order logic on tree-like structures. *Theor. Comput. Sci.*, 275(1-2):311–346, 2002. doi:10.1016/S0304-3975(01)00185-2.

Integer Linear-Exponential Programming in NP by Quantifier Elimination

Dmitry Chistikov[1]
Centre for Discrete Mathematics and its Applications (DIMAP) &
Department of Computer Science, University of Warwick, Coventry, UK

Alessio Mansutti
IMDEA Software Institute, Madrid, Spain

Mikhail R. Starchak
St. Petersburg State University, Russia

Abstract

This paper provides an NP procedure that decides whether a *linear-exponential system* of constraints has an integer solution. Linear-exponential systems extend standard integer linear programs with exponential terms 2^x and remainder terms (x mod 2^y). Our result implies that the existential theory of the structure $(\mathbb{N}, 0, 1, +, 2^{(\cdot)}, V_2(\cdot,\cdot), \leq)$ has an NP-complete satisfiability problem, thus improving upon a recent ExpSpace upper bound. This theory extends the existential fragment of Presburger arithmetic with the exponentiation function $x \mapsto 2^x$ and the binary predicate $V_2(x, y)$ that is true whenever $y \geq 1$ is the largest power of 2 dividing x.

Our procedure for solving linear-exponential systems uses the method of quantifier elimination. As a by-product, we modify the classical Gaussian variable elimination into a non-deterministic polynomial-time procedure for integer linear programming (or: existential Presburger arithmetic).

2012 ACM Subject Classification Computing methodologies → Symbolic and algebraic algorithms; Theory of computation → Logic

Keywords and phrases decision procedures, integer programming, quantifier elimination

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.132

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding *Dmitry Chistikov*: Supported in part by the Engineering and Physical Sciences Research Council [EP/X03027X/1].
Alessio Mansutti: funded by the Madrid Regional Government (César Nombela grant 2023-T1/COM-29001), and by MCIN/AEI/10.13039/501100011033/FEDER, EU (grant PID2022-138072OB-I00).
Mikhail R. Starchak: Supported by the Russian Science Foundation, project 23-71-01041.

1 Introduction

Integer (linear) programming is the problem of deciding whether a system of linear inequalities has a solution over the integers ($\mathbb{Z}$). It is a textbook fact that this problem is NP-complete; however, proof of membership in NP is not trivial. It is established [3, 27] by showing that, if a given system has a solution over $\mathbb{Z}$, then it also has a *small* solution. The latter means that the bit size of all components can be bounded from above by a polynomial in the bit size of the system. Integer programming is an important language that can encode many combinatorial problems and constraints from multiple application domains; see, e.g., [20, 32].

[1] During the work on this paper, DC was a visitor to the Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern and Saarbrücken, Germany, a visiting fellow at St Catherine's College and a visitor to the Department of Computer Science at the University of Oxford, United Kingdom.

In this paper we consider more general systems of constraints, which may contain not only linear inequalities (as in integer programming) but also constraints of the form $y = 2^x$ (exponentiation base 2) and $z = (x \bmod 2^y)$ (remainder modulo powers of 2). Equivalently, and embedding both new operations into a uniform syntax, we look at a conjunction of inequalities of the form

$$\sum_{i=1}^{n} \left(a_i \cdot x_i + b_i \cdot 2^{x_i} + \sum_{j=1}^{n} c_{i,j} \cdot (x_i \bmod 2^{x_j}) \right) + d \leq 0\,, \tag{1}$$

referred to as an *(integer) linear-exponential system*. In fact, the linear-exponential systems that we consider can also feature equalities $=$ and strict inequalities $<$.[2]

Observe that a linear-exponential system of the form $x_1 = 1 \wedge \bigwedge_{i=1}^{n}(x_{i+1} = 2^{x_i})$ states that x_{n+1} is the tower of 2s of height n. This number is huge, and makes proving an analogue of the small solution property described above a hopeless task in our setting. This obstacle was recently shown avoidable [11], however, and an exponential-space procedure for linear-exponential programs was found, relying on automata-theoretic methods. Our main result is that, in fact, the problem belongs to NP.

▶ **Theorem 1.** *Deciding whether a linear-exponential system over $\mathbb{Z}$ has a solution is in* NP.

We highlight that the choice of the base 2 for the exponentials is for the convenience of exposition: our result holds for any positive integer base given in binary as part of the input.

As an example showcasing the power of integer linear-exponential systems, consider computation of discrete logarithm base 2: given non-negative integers $m, r \in \mathbb{N}$, producing an $x \in \mathbb{N}$ such that $2^x - r$ is divisible by m. As sketched in [14], this problem is reducible to checking feasibility (existence of solutions) of at most $\log m$ linear-exponential systems in two variables, by a binary search for a suitable exponent x. Hence, improving Theorem 1 from NP to PTime for the case of linear-exponential systems with a *fixed* number of variables would require a major breakthrough in number theory. In contrast, under this restriction, feasibility of standard integer linear programs can be determined in PTime [19].

For the authors of this paper, the main motivation for looking at linear-exponential systems stems from logic. Consider the first-order theory of the structure $(\mathbb{N}, 0, 1, +, 2^{(\cdot)}, V_2(\cdot,\cdot), \leq)$, which we refer to as the *Büchi–Semenov arithmetic*. In this structure, the signature $(0, 1, +, \leq)$ of linear arithmetic is extended with the function symbol $2^{(\cdot)}$, interpreted as the function $x \mapsto 2^x$, and the binary predicate symbol V_2, interpreted as $\{(x, y) \in \mathbb{N} \times \mathbb{N} : y$ is the largest power of 2 that divides $x\}$. Importantly, the predicate V_2 can be replaced in this definition with the function $x \bmod 2^y$, because the two are mutually expressible:

$$\begin{aligned} V_2(x, y) &\iff \exists v \left(2 \cdot y = 2^v \wedge 2 \cdot (x \bmod 2^v) = 2^v\right), \\ (x \bmod 2^y) = z &\iff z \leq 2^y - 1 \wedge \left(x = z \vee \exists u \left(V_2(x - z, 2^u) \wedge 2^y \leq 2^u\right)\right). \end{aligned}$$

Above, the subtraction symbol can be expressed in the theory in the obvious way (perhaps with the help of an auxiliary existential quantifier for expressing $x - z$).

Büchi–Semenov arithmetic subsumes logical theories known as Büchi arithmetic and Semenov arithmetic; see Section 2. As a consequence of Theorem 1, we show:

▶ **Theorem 2.** *The satisfiability problem of existential Büchi–Semenov arithmetic is in* NP.

[2] While equalities are considered for convenience only (they can be encoded with a pair of inequalities $\leq$), the addition of $<$ is of interest. Indeed, differently from standard integer programming, one cannot define $<$ in terms of $\leq$, since 2^y is not an integer for $y < 0$. Observe that $(x \bmod 2^y) = 0$ when $y < 0$, because over the reals $(a \bmod m) = a - m \left\lfloor \frac{a}{m} \right\rfloor$, where $\lfloor . \rfloor$ is the floor function.

Theorems 1 and 2 improve upon several results in the literature. The most recent such result is the exponential-space procedure [11] already mentioned above. In 2023, two elementary decision procedures were developed concurrently and independently for integer linear programs with exponentiation constraints ($y = 2^x$), or equivalently for the existential fragment of Semenov arithmetic: they run in non-deterministic exponential time [2] and in triply exponential time [41], respectively. Finally, another result subsumed by Theorem 2 is the membership in NP for the existential fragment of Büchi arithmetic [13].

Theorem 1 is established by designing a non-deterministic polynomial-time decision procedure, which, unlike those in papers [11, 13] but similarly to [2, 41], avoids automata-theoretic methods and instead relies on *quantifier elimination*. This is a powerful method (see, e.g., [9] as well as Section 2) that can be seen as a bridge between logic and integer programming. Presburger [30] used it to show decidability of linear integer arithmetic (and Tarski for real arithmetic with addition and multiplication). For systems of linear equations, quantifier elimination is essentially Gaussian elimination. As a little stepping stone, which was in fact one of the springboards for our paper, we extend the PTIME integer Gaussian elimination procedure by Bareiss [1, 39] into an NP procedure for solving systems of inequalities over $\mathbb{Z}$ (thus re-proving membership of integer linear programming in NP).

A look ahead. The following Section 2 recalls some relevant related work on logical theories of arithmetic. At the end of the paper (Section 9) this material is complemented by a discussion of future research directions, along with several more key references.

The NP procedure for integer programming is given as Algorithm 1 in Section 4. In this extended abstract, we do not provide a proof of correctness or analysis of the running time, but instead compare the algorithm with the classic Gauss–Jordan variable elimination and with Bareiss' method for systems of equations (that is, equalities). Necessary definitions and background information are provided in the Preliminaries (Section 3).

Our core result is an NP procedure for solving linear-exponential systems over $\mathbb{N}$. Its pseudocode is split into Algorithms 2–4. These are presented in the same imperative style with non-deterministic branching as Algorithm 1, and in fact Algorithm 3 relies on Algorithm 1. Section 5 provides a high-level overview of all three algorithms together. To this end, we introduce several new auxiliary concepts: quotient systems and quotient terms, delayed substitution, and primitive linear-exponential systems. After this, technical details of Algorithms 2–4 are given. Section 6 sketches key ideas behind the correctness argument, and the text within this section is thus to be read alongside the pseudocode of Algorithms. An overview of the analysis of the worst-case running time is presented in Section 7. The basic definitions are again those from Preliminaries, and of particular relevance are the subtleties of the action of term substitutions.

Building on the core procedure, in Section 8 we show how to solve in NP linear-exponential systems not only over $\mathbb{N}$ but also over $\mathbb{Z}$ (Theorem 1) and how to decide Büchi–Semenov arithmetic in NP (Theorem 2). The modifications to this procedure that enable proving both results for a different integer base $b > 2$ for the exponentials are given in Appendix A.

2 Arithmetic theories of Büchi, Semenov, and Presburger

In this section, we review results on arithmetic theories that are the most relevant to our study.

Büchi arithmetic is the first-order theory of the structure $(\mathbb{N}, 0, 1, +, V_2(\cdot,\cdot), \leq)$. By the celebrated Büchi–Bruyère theorem [4, 5], a set $S \subseteq \mathbb{N}^d$ is definable in $(\mathbb{N}, 0, 1, +, V_2(\cdot,\cdot), \leq)$ if and only if the representation of S as a language over the alphabet $\{0,1\}^d$ is recognisable

by a deterministic finite automaton (DFA). The theorem is effective, and implies that the satisfiability problem for Büchi arithmetic is in Tower (in fact, Tower-complete) [31, 36]. The situation is different for the existential fragment of this theory. The satisfiability problem is now NP-complete [13], but existential formulae are less expressive [15]. In particular, this fragment fails to capture the binary language $\{10, 01\}^*$. Decision procedures for Büchi arithmetic have been successfully implemented and used to automatically prove many results in combinatorics on words and number theory [35].

Semenov arithmetic is the first-order theory of the structure $(\mathbb{N}, 0, 1, +, 2^{(\cdot)}, \leq)$. Its decidability follows from the classical work of Semenov on sparse predicates [33, 34], and an explicit decision procedure was given by Cherlin and Point [6, 28]. Similarly to Büchi arithmetic, Semenov arithmetic is Tower-complete [8, 29]; however, its existential fragment has only been known to be in NExpTime [2]. The paper [41] provides applications of this fragment to solving systems of string constraints with string-to-integer conversion functions.

Büchi–Semenov arithmetic is a natural combination of these two theories. Differently from Büchi and Semenov arithmetics, the $\exists^*\forall^*$-fragment of this logic is undecidable [6]. In view of this, the recent result showing that the satisfiability problem of existential Büchi–Semenov arithmetic is in ExpSpace [11] is surprising. The proof technique, moreover, establishes the membership in ExpSpace of the extension of existential Büchi–Semenov arithmetic with arbitrary regular predicates given on input as DFAs. Since this extension can express the intersection non-emptiness problem for DFAs, its satisfiability problem is PSpace-hard [22]. The decision procedure of [11] was applied to give an algorithm for solving real-world instances of word equations with length constraints.

Both first-order theories of the structures $(\mathbb{N}, 0, 1, +, \leq)$ and $(\mathbb{Z}, 0, 1, +, \leq)$ are usually referred to as *Presburger arithmetic*, because the decision problems for these theories are logspace inter-reducible, meaning that each structure can be interpreted in the other. The procedures that we propose in this paper build upon a version of the quantifier-elimination procedure for the first-order theory of the structure $(\mathbb{Z}, 0, 1, +, \leq)$. Standard procedures for this theory [9, 26, 38] are known to be suboptimal when applied to the existential fragment: throughout these procedures, the bit size of the numbers in the formulae grow exponentially faster than in, e.g., geometric procedures for the theory [7]. A remedy to this well-known issue was proposed by Weispfenning [39, Corollary 4.3]. We develop his observation in Section 4.

3 Preliminaries

We usually write $a, b, c, \ldots$ for integers, $x, y, z, \ldots$ for integer variables, and $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \ldots$ and $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \ldots$ for vectors of those. By $\boldsymbol{x} \setminus y$ we denote the vector obtained by removing the variable y from $\boldsymbol{x}$. We denote linear-exponential systems and logical formulae with the letters $\varphi, \chi, \psi, \ldots$, and write $\varphi(\boldsymbol{x})$ when the (free) variables of φ are among $\boldsymbol{x}$.

For $a \in \mathbb{R}$, we write $|a|$, $\lceil a \rceil$, and $\log a$ for the *absolute value*, *ceiling*, and (if $a > 0$) the *binary logarithm* of a. All numbers encountered by our algorithm are encoded in binary; note that $n \in \mathbb{N}$ can be represented using $\lceil \log(n+1) \rceil$ bits. For $n, m \in \mathbb{Z}$, denote $[n, m] \coloneqq \{n, n+1, \ldots, m\}$. The set $\mathbb{N}$ of non-negative integers contains 0.

Terms. As in Equation (1), a *(linear-exponential) term* is an expression of the form

$$\sum_{i=1}^{n} \left(a_i \cdot x_i + b_i \cdot 2^{x_i} + \sum_{j=1}^{n} c_{i,j} \cdot (x_i \bmod 2^{x_j})\right) + d, \tag{2}$$

where $a_i, b_i, c_{i,j} \in \mathbb{Z}$ are the *coefficients* of the term and $d \in \mathbb{Z}$ is its *constant*. If all b_i and $c_{i,j}$ are zero then the term is said to be *linear*. We denote terms by the letters $\rho, \sigma, \tau, \ldots$, and write $\tau(\boldsymbol{x})$ if all variables of the term τ are in $\boldsymbol{x}$. For a term τ in Equation (2), its 1-norm is $\|\tau\|_1 \coloneqq \sum_{i=1}^{n}(|a_i| + |b_i| + \sum_{j=1}^{n} |c_{i,j}|) + |d|$.

We use the words "system" and "conjunction" of constraints interchangeably. While equalities and inequalities of a linear-exponential system are always of the form $\tau = 0$, $\tau \leq 0$, and $\tau < 0$, for the convenience of exposition we often rearrange left- and right-hand sides and write, e.g., $\tau_1 \leq \tau_2$. In our procedures, linear-exponential systems may contain equalities, inequalities, and also *divisibility constraints* $d \mid \tau$, where τ is a term as in Equation (2), $d \in \mathbb{Z}$ is non-zero, and $\mid$ is the *divisibility predicate*, $\{(d, n) \in \mathbb{Z} \times \mathbb{Z} : n = kd \text{ for some } k \in \mathbb{Z}\}$. We write $\mathit{mod}(\varphi)$ for the (positive) least common multiple of all divisors d appearing in divisibility constraints $d \mid \tau$ of a system φ. For purely syntactic reasons, it is sometimes convenient to see a divisibility constraint $d \mid \tau_1 - \tau_2$ as a *congruence* $\tau_1 \equiv_d \tau_2$, where $d \geq 1$ with no loss of generality. We use the term *divisibility constraint* also for these congruences.

Substitutions. Our procedure uses several special kinds of substitutions. Consider a linear-exponential system φ, a term τ, two variables x and y, and $a \in \mathbb{Z} \setminus \{0\}$.

- We write $\varphi[\tau \,/\, x]$ for the system obtained from φ by replacing every *linear occurrence* of x outside modulo operators with τ. To clarify, this substitution only modifies the "$a_i \cdot x_i$" parts of the term in Equation (2), but not the "$c_{i,j} \cdot (x_i \bmod 2^{x_j})$" parts.
- We write $\varphi[\tau \,/\, x \bmod 2^y]$ and $\varphi[\tau \,/\, 2^x]$ for the system obtained from φ by replacing with τ every occurrence of $(x \bmod 2^y)$ and 2^x, respectively.
- We write $\varphi[[\frac{\tau}{a} \,/\, x]]$ for the *vigorous substitution* of $\frac{\tau}{a}$ for x. This substitution works as follows. **1:** Multiply every equality and inequality by a, flipping the signs of inequalities if $a < 0$; this step also applies to inequalities in which x does not occur. **2:** Multiply both sides of divisibility constraints in which x occurs by a, i.e., $d \mid \tau$ becomes $a \cdot d \mid a \cdot \tau$. **3:** Replace with τ every linear occurrence of $a \cdot x$ outside modulo operators. Note that, thanks to step 1, each coefficient of x in the system can be factorised as $a \cdot b$ for some $b \in \mathbb{Z}$.

We sometimes see substitutions $[\tau \,/\, \tau']$ as first-class citizens: functions mapping systems to systems. When adopting this perspective, $\varphi[\tau \,/\, \tau']$ is seen as a function application.

4 Solving systems of linear inequalities over $\mathbb{Z}$

In this section we present Algorithm 1 (GaussQE), a non-deterministic polynomial time quantifier elimination (QE) procedure for solving systems of linear inequalities over $\mathbb{Z}$, or in other words for integer programming. A constraint (equality, inequality, or divisibility) is *linear* if it only contains linear terms, as defined in Section 3.

We already mentioned in Section 1 that Integer Programming $\in$ NP is a standard result. Intuitively, the range of each variable is infinite, which necessitates a proof that a suitable (and *small*) range suffices; see, e.g., [3, 27, 37]. Methods developed in these references, however, do not enjoy the flexibility of quantifier elimination: e.g., they either do not preserve formula equivalence or are not actually removing quantifiers.

▶ **Theorem 3.** *Algorithm 1 (GaussQE) runs in non-deterministic polynomial time and, given a linear system $\varphi(\boldsymbol{x}, \boldsymbol{z})$ and variables $\boldsymbol{x}$, produces in each non-deterministic branch β a linear system $\psi_\beta(\boldsymbol{z})$ such that $\bigvee_\beta \psi_\beta$ is equivalent to $\exists \boldsymbol{x}\, \varphi$.*

GaussQE is based on an observation by Weispfenning, who drew a parallel between a weak form of QE and Gaussian variable elimination [39]. Based on this observation and relying on an insight by Bareiss [1] (to be discussed below), Weispfenning sketched a non-deterministic procedure for deciding *closed* existential formulae of Presburger arithmetic in polynomial time. Although the idea of weak QE [39] has since been developed further [23], the NP observation has apparently remained not well known.

Algorithm 1 GaussQE: Gauss–Jordan elimination for integer programming.

Input: $\boldsymbol{x}$: sequence of variables; $\varphi(\boldsymbol{x}, \boldsymbol{z})$: system of linear constraints.
Output of each branch (β): system $\psi_\beta(\boldsymbol{z})$ of linear constraints.
Ensuring: $\bigvee_\beta \psi_\beta$ is equivalent to $\exists \boldsymbol{x}\, \varphi$.

1: replace each inequality $\tau \leq 0$ in φ with $\tau + y = 0$, where y is a fresh slack variable
2: $\ell \leftarrow 1$; $\quad s \leftarrow ()$ ▷ *s is an empty sequence of substitutions*
3: **foreach** x in $\boldsymbol{x}$ **do**
4: **if** no equality of φ contains x **then continue**
5: **guess** $ax + \tau = 0$ (with $a \neq 0$) $\leftarrow$ equality in φ that contains x
6: $p \leftarrow \ell$; $\quad \ell \leftarrow a$ ▷ *previous and current lead coefficients*
7: **if** τ contains a slack variable y not assigned by s **then**
8: **guess** $v \leftarrow$ integer in $[0, |a| \cdot mod(\varphi) - 1]$
9: append $[v \,/\, y]$ to s
10: $\varphi \leftarrow \varphi[[\frac{-\tau}{a} \,/\, x]]$
11: divide each constraint in φ by p ▷ *in divisibility constraints, both sides are affected*
12: $\varphi \leftarrow \varphi \wedge (a \mid \tau)$
13: **foreach** equality $\eta = 0$ of φ that contains some slack variable y not assigned by s **do**
14: replace $\eta = 0$ with $\eta[0 \,/\, y] \leq 0$ **if** the coefficient at y is positive **else** with $\eta[0 \,/\, y] \geq 0$
15: apply substitutions of s to φ
16: **foreach** x in $\boldsymbol{x}$ that occurs in φ **do**
17: **guess** $r \leftarrow$ integer in $[0, mod(\varphi) - 1]$
18: $\varphi \leftarrow \varphi[r \,/\, x]$
19: **return** φ

Due to space constraints, we omit the proof of Theorem 3, and explain instead only the key ideas. We first consider the specification of GaussQE, in particular non-deterministic branching. We then recall the main underlying mechanism: Gaussian variable elimination (thus retracing and expanding Weispfenning's observation). After that, we discuss extension of this mechanism to tackle inequalities over $\mathbb{Z}$.

Input, output, and non-determinism. The input to GaussQE is a system φ of linear constraints, as well as a sequence $\boldsymbol{x}$ of variables to eliminate. The algorithm makes non-deterministic guesses in lines 5, 8, and 17. Output of each branch (of the non-deterministic execution) is specified at the top: it is a system ψ_β of linear constraints, in which all variables x in $\boldsymbol{x}$ have been eliminated. For any specific non-deterministic branch, call it β, the output system ψ_β may not necessarily be equivalent to $\exists \boldsymbol{x}\, \varphi$, but the disjunction of all outputs across all branches must be: $\bigvee_\beta \psi_\beta$ has the same set of satisfying assignments as $\exists \boldsymbol{x}\, \varphi$.[3]

The number of non-deterministic branches (individual paths through the execution tree) is usually exponential, but each of them runs in polynomial time. (This is true for all algorithms presented in this paper.) If all variables of the input system φ are included in $\boldsymbol{x}$, then each branch returns a conjunction of numerical assertions that evaluates to true or false.

[3] Formally, an assignment is a map ν from (free) variables to $\mathbb{Z}$. It satisfies a constraint if replacing each z in the domain of ν with $\nu(z)$ makes the constraint a true numerical assertion.

Gaussian elimination and Bareiss' method. Consider a system φ of linear equations (i.e., equalities) over fields, e.g., $\mathbb{R}$ or $\mathbb{Q}$, and let $\boldsymbol{x}$ be a vector of variables that we wish to eliminate from φ. We recall the Gauss–Jordan variable elimination algorithm, proceeding as follows:

01: $\ell \leftarrow 1$
02: **foreach** x in $\boldsymbol{x}$ **do**
03: **if** no equality of φ contains x **then continue**
04: **let** $ax + \tau = 0$ (with $a \neq 0$) $\leftarrow$ an arbitrary equality in φ that contains x
05: $p \leftarrow \ell$; $\ell \leftarrow a$
06: $\varphi \leftarrow \varphi[[\frac{-\tau}{a} \,/\, x]]$
07: divide each constraint in φ by p
08: **return** φ

By removing from this code all lines involving p and ℓ (lines 01, 05 and 07), we obtain a naive version of the procedure: an equation is picked in line 04 and used to remove one of its occurring variables in line 06. Indeed, applying a vigorous substitution $[[\frac{-\tau}{a} \,/\, x]]$ to an equality $bx + \sigma = 0$ is equivalent to first multiplying this equality by the lead coefficient a and then subtracting $b \cdot (ax + \tau) = 0$. The result is $-b\tau + a\sigma = 0$, and x is eliminated.

An insightful observation due to Bareiss [1] is that, after multiple iterations, coefficients accumulate non-trivial common factors. Lines 01, 05, and 07 take advantage of this. Indeed, line 07 divides every equation by such a common factor. Importantly, if all numbers in the input system φ are integers, then the division is without remainder. To show this, Bareiss uses a linear-algebraic argument based on an application of the Desnanot–Jacobi identity (or, more generally, Sylvester's identity) for determinants [1, 10, 21]. Over $\mathbb{Q}$, this makes it possible to perform Gaussian elimination (its "fraction-free one-step" version) in PTime. (This is not the only polynomial-time method; cf. [32, Section 3.3].)

Gaussian elimination for systems of equations can be extended to solving over $\mathbb{Z}$, by introducing divisibility constraints: line 06 becomes $\varphi \leftarrow \varphi[[\frac{-\tau}{a} \,/\, x]] \wedge (a \mid \tau)$. However, while the running time of the procedure remains polynomial, its effect becomes more modest: the procedure reduces a system of linear equations over $\mathbb{Z}$ to an equivalent system of equations featuring variables not in $\boldsymbol{x}$ and multivariate linear congruences that may still contain variables from $\boldsymbol{x}$. To completely eliminate $\boldsymbol{x}$, further computation is required. For our purposes, non-deterministic guessing is a good enough solution to this problem; see the final **foreach** loop in lines 16–18 of GaussQE.

From equalities to inequalities. GaussQE extends Bareiss' method to systems of inequalities over $\mathbb{Z}$. As above, the method allows us to control the (otherwise exponential) growth of the bit size of numbers. Gaussian elimination is, of course, still at the heart of the algorithm (see lines 2–6, 10, and 11), and we now discuss two modifications:

- Line 1 introduces *slack variables* ranging over $\mathbb{N}$. These are internal to the procedure and are removed at the end (lines 13–15).
- In line 5 the equality $ax + \tau = 0$ is selected non-deterministically.

The latter modification is required for the correctness (more precisely: completeness) of GaussQE. Geometrically, for a satisfiable system of inequalities over $\mathbb{Z}$ consider the convex polyhedron of all solutions over $\mathbb{R}$ first. At least one of solutions over $\mathbb{Z}$ must lie in or near a facet of this polyhedron. Line 5 of Algorithm 1 attempts to guess this facet. The amount of slack guessed in line 8 corresponds to the distance from the facet. Observe that if $ax + \tau = 0$ corresponds to an equality of the original system φ, then every solution of φ needs to satisfy $ax + \tau = 0$ exactly, and so there is no slack (lines 8–9 are not taken).

The values chosen for the slack variables in line 8 have, in fact, a counterpart in the standard decision procedures for Presburger arithmetic. When the latter pick a term ρ to substitute, the substitutions in fact introduce $\rho + k$ for k ranging in some $[0, \ell]$, where ℓ depends on $mod(\varphi)$. The amount of slack considered in GaussQE corresponds to these values of k. (Because of this parallel, making the range of guesses in line 8 symmetric, i.e., $|v| \leq |a| \cdot mod(\varphi) - 1$, extends our procedure to the entire existential Presburger arithmetic.)

5 Solving linear-exponential systems over $\mathbb{N}$: an overview

In this section we give an overview of our non-deterministic procedure to solve linear-exponential systems over $\mathbb{N}$. The procedure is split into Algorithms 2–4. A more technical analysis of these algorithms is given later in Section 6.

Whenever non-deterministic Algorithms 1–4 call one another, the return value is always just the output of a single branch, rather than (say) the disjunction over all branches.

Algorithm 2 (LinExpSat). This is the main procedure. It takes as input a linear-exponential system φ without divisibility constraints and decides whether φ has a solution over $\mathbb{N}$. The procedure relies on first (non-deterministically) fixing a linear ordering θ on the exponential terms 2^x occurring in φ (line 2). For technical convenience, this ordering contains a term 2^{x_0}, with x_0 fresh variable, and sets $2^{x_0} = 1$. Variables are iteratively eliminated starting from the one corresponding to the leading exponential term in θ (i.e., the biggest one), until reaching x_0 (lines 3–16). The elimination of each variable is performed by first rewriting the system (in lines 8–14) into a form admissible for Algorithm 3 discussed below. This rewriting introduces new variables, which will never occur in exponentials throughout the entire procedure and are later eliminated when the procedure reaches x_0. Overall, the termination of the procedure is ensured by the decreasing number of exponentiated variables. After LinExpSat rewrites φ, it calls Algorithm 3 to eliminate the currently biggest variable.

Algorithm 3 (ElimMaxVar). This procedure takes as input an ordering θ, a *quotient system induced by* θ and a *delayed substitution*. Let us introduce these notions.

Quotient systems. Let $\theta(\boldsymbol{x})$ be the ordering $2^{x_n} \geq 2^{x_{n-1}} \geq \cdots \geq 2^{x_0} = 1$, where $n \geq 1$. A *quotient system induced by* θ is a system $\varphi(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{z}')$ of equalities, inequalities, and divisibility constraints $\tau \sim 0$, where $\sim \in \{<, \leq, =, \equiv_d \colon d \geq 1\}$ and τ is an *quotient term (induced by θ)*, that is, a term of the form

$$a \cdot 2^{x_n} + f(\boldsymbol{x}') \cdot 2^{x_{n-1}} + b \cdot x_{n-1} + \tau'(x_0, \ldots, x_{n-2}, \boldsymbol{z}')\,,$$

where $a, b \in \mathbb{Z}$, $f(\boldsymbol{x}')$ is a linear term, and τ' is a linear-exponential term in which the variables from $\boldsymbol{z}'$ do not occur exponentiated. Furthermore, for every variable z' in $\boldsymbol{z}'$, the quotient system φ features the inequalities $0 \leq z' < 2^{x_{n-1}}$. The variables in $\boldsymbol{x}$, $\boldsymbol{x}'$ and $\boldsymbol{z}'$ form three disjoint sets, which we call the *exponentiated variables*, the *quotient variables* and the *remainder variables* of the system φ, respectively. We also refer to the term $b \cdot x_{n-1} + \tau'(x_0, \ldots, x_{n-2}, \boldsymbol{z}')$ as the *least significant part* of the quotient term τ. Importantly, quotient terms are **not** linear-exponential terms.

Here is an example of a quotient system induced by $2^{x_3} \geq 2^{x_2} \geq 2^{x_1} \geq 2^{x_0} = 1$, and having quotient variables $\boldsymbol{x}' = (x_1', x_2')$ and remainder variables $\boldsymbol{z}' = (z_1', z_2')$

$$\begin{aligned} -2^{x_3} + (2 \cdot x_1' - x_2' - 1) \cdot 2^{x_2} + \big\{ -2 \cdot x_2 + 2^{x_1} - (z_1' \bmod 2^{x_1}) \big\} \leq 0, \quad & 0 \leq z_1' < 2^{x_2}, \\ x_1' \cdot 2^{x_2} + \big\{ x_1 + z_2' - 5 \big\} = 0, \quad & 0 \leq z_2' < 2^{x_2}. \end{aligned}$$

The curly brackets highlight the least significant parts of two terms of the system, the other parts being $\pm z'_1$ and $\pm z'_2$ stemming from the inequalities on the right.

Delayed substitution. This is a substitution of the form $[x' \cdot 2^{x_{n-1}} + z' \,/\, x_n]$, where 2^{x_n} is the leading exponential term of θ. Our procedure delays the application of this substitution until x_n occurs linearly in the system φ. One can think of this substitution as an equality $(x_n = x' \cdot 2^{x_{n-1}} + z')$ in φ that must not be manipulated for the time being.

Back to ElimMaxVar, given a quotient system $\varphi(\boldsymbol{x}, \boldsymbol{x'}, \boldsymbol{z'})$ induced by θ and the delayed substitution $[x' \cdot 2^{x_{n-1}} + z' \,/\, x_n]$, the goals of this procedure are to (i) eliminate the quotient variables $\boldsymbol{x'} \setminus x'$; (ii) eliminate all occurrences of the leading exponential term 2^{x_n} of θ and apply the delayed substitution to eliminate the variable x_n; (iii) finally, remove x'. Upon exit, ElimMaxVar gives back to LinExpSat a (non-quotient) linear-exponential system where x_n has been eliminated; i.e., a system with one fewer exponentiated variable.

For steps (i) and (iii), the procedure relies on the Algorithm 1 (GaussQE) for eliminating variables in systems of inequalities, from Section 4. This is where flexibility of QE is important: in line 22 some variables are eliminated and some are not. Step (ii) is instead implemented by Algorithm 4.

Algorithm 4 (SolvePrimitive). The goal of this procedure is to rewrite a system of constraints where x_n occurs exponentiated with another system where all constraints are linear. The specification of the procedure restricts the output further. At its core, SolvePrimitive tailors Semenov's proof of the decidability of the first-order theory of the structure $(\mathbb{N}, 0, 1, +, 2^{(\cdot)}, \leq)$ [34] to a small syntactic fragment, which we now define.

Primitive linear-exponential systems. Let u, v be two variables. A linear-exponential system is said to be (u, v)-*primitive* whenever all its (in)equalities and divisibility constraints are of the form $a \cdot 2^u + b \cdot v + c \sim 0$, with $a, b, c \in \mathbb{Z}$ and $\sim \in \{<, \leq, =, \equiv_d \colon d \geq 1\}$.

The input to SolvePrimitive is a (u, v)-primitive linear-exponential system. This procedure removes all occurrences of 2^u in favour of linear constraints, working under the assumption that $u \geq v$. This condition is ensured when ElimMaxVar invokes SolvePrimitive. The variable u of the primitive system in the input corresponds to the term $x_n - x_{n-1}$, and the variable v stands for the variable x' in the delayed substitution $[x' \cdot 2^{x_{n-1}} + z' \,/\, x_n]$. ElimMaxVar ensures that $x_n - x_{n-1} \geq x'$.

6 Algorithms 2, 3, 4: a walkthrough

Having outlined the interplay between Algorithms 2–4, we move to their technical description, and present the key ideas required to establish the correctness of our procedure for solving linear-exponential systems over $\mathbb{N}$.

6.1 Algorithm 2: the main loop

Let $\varphi(x_1, \ldots, x_n)$ be an input linear-exponential system (with no divisibility constraints). As explained in the summary above, LinExpSat starts by guessing an ordering $\theta(x_0, \ldots, x_n)$ of the form $t_1 \geq t_2 \geq \cdots \geq t_n \geq 2^{x_0} = 1$, where $(t_1, \ldots, t_n)$ is a permutation of the terms $2^{x_1}, \ldots, 2^{x_n}$, and x_0 is a fresh variable used as a placeholder for 0. Note that if φ is satisfiable (over $\mathbb{N}$), then θ can be guessed so that $\varphi \wedge \theta$ is satisfiable; and conversely no such θ exists if φ is unsatisfiable. For the sake of convenience, we assume in this section that the ordering $\theta(x_0, \ldots, x_n)$ guessed by the procedure is $2^{x_n} \geq 2^{x_{n-1}} \geq \cdots \geq 2^{x_1} \geq 2^{x_0} = 1$.

Algorithm 2 LinExpSat: A procedure to decide linear-exponential systems over $\mathbb{N}$.

Input: $\varphi(x_1, \ldots, x_n)$: linear-exponential system (without divisibility constraints).
Output: True ($\top$) if φ has a solution over $\mathbb{N}$, and otherwise false ($\bot$).

1: **let** x_0 be a fresh variable ▷ *placeholder for* 0
2: **guess** $\theta \leftarrow$ ordering of the form $t_1 \geq t_2 \geq \cdots \geq t_n \geq 2^{x_0} = 1$, where $(t_1, \ldots, t_n)$ is a permutation of the terms $2^{x_1}, \ldots, 2^{x_n}$
3: **while** θ is not the ordering $(2^{x_0} = 1)$ **do**
4: $\quad 2^x \leftarrow$ leading exponential term of θ ▷ *in the i-th iteration, 2^x is t_i*
5: $\quad 2^y \leftarrow$ successor of 2^x in θ ▷ *and 2^y is t_{i+1}*
6: $\quad \varphi \leftarrow \varphi[w \,/\, (w \bmod 2^x) : w \text{ is a variable}]$
7: $\quad \boldsymbol{z} \leftarrow$ all variables z in φ such that z is x or z does not appear in θ
8: $\quad$ **foreach** z in $\boldsymbol{z}$ **do** ▷ *form a quotient system induced by θ*
9: $\quad\quad$ **let** x' and z' be two fresh variables
10: $\quad\quad \varphi \leftarrow \varphi \wedge (0 \leq z' < 2^y)$
11: $\quad\quad \varphi \leftarrow \varphi[z' \,/\, (z \bmod 2^y)]$
12: $\quad\quad \varphi \leftarrow \varphi[(z' \bmod 2^w) \,/\, (z \bmod 2^w) : w \text{ is such that } \theta \text{ implies } 2^w \leq 2^y]$
13: $\quad\quad \varphi \leftarrow \varphi[(x' \cdot 2^y + z') \,/\, z]$ ▷ *replaces only the linear occurrences of z*
14: $\quad\quad$ **if** z is x **then** $(x'_0, z'_0) \leftarrow (x', z')$ ▷ *for delayed substitution, see next line*
15: $\quad \varphi \leftarrow$ ElimMaxVar$(\theta, \varphi, [x'_0 \cdot 2^y + z'_0 \,/\, x])$
16: $\quad$ remove 2^x from θ
17: **return** $\varphi(\boldsymbol{0})$ ▷ *evaluates to $\top$ or $\bot$*

The **while** loop starting in line 3 manipulates φ and θ, non-deterministically obtaining at the end of the ith iteration a system $\varphi_i(\boldsymbol{x}, \boldsymbol{z})$ and an ordering $\theta_i(\boldsymbol{x})$, where $\boldsymbol{x} = (x_0, \ldots, x_{n-i})$ and $\boldsymbol{z}$ is a vector of i fresh variables. The non-deterministic guesses performed by LinExpSat are such that the following properties (I1)–(I3) are loop invariants across all branches, whereas (I4) is an invariant for at least one branch (below, $i \in [0, n]$ and $(\varphi_0, \theta_0) \coloneqq (\varphi, \theta)$):

I1. All variables that occur exponentiated in φ_i are among $x_0, \ldots, x_{n-i}$.
I2. θ_i is the ordering $2^{x_{n-i}} \geq 2^{x_{n-i-1}} \geq \cdots \geq 2^{x_1} \geq 2^{x_0} = 1$.
I3. All variables z in $\boldsymbol{z}$ are such that $z < 2^{x_{n-i}}$ is an inequality in φ_i.
I4. $\varphi_i \wedge \theta_i$ is equisatisfiable with $\varphi \wedge \theta$ over $\mathbb{N}$.

More precisely, writing $\bigvee_\beta \psi_\beta$ for the disjunction of all the formulae $\varphi_i \wedge \theta_i$ obtained across all non-deterministic branches, we have that $\bigvee_\beta \psi_\beta$ and $\varphi \wedge \theta$ are equisatisfiable. Therefore, whenever $\varphi \wedge \theta$ is satisfiable, (I4) holds for at least one branch. If $\varphi \wedge \theta$ is instead unsatisfiable, then (I4) holds instead for all branches.

The invariant above is clearly true for φ_0 and θ_0, with $\boldsymbol{z}$ being the empty set of variables. Item (I2) implies that, after n iterations, θ_n is $2^{x_0} = 1$, which causes the **while** loop to exit. Given θ_n, properties (I1) and (I3) force the values of x_0 and of all variables in $\boldsymbol{z}$ to be zero, thus making $\varphi \wedge \theta$ equisatisfiable with $\varphi_n(\boldsymbol{0})$ in at least one branch of the algorithm, by (I4). In summary, this will enable us to conclude that the procedure is correct.

Let us now look at the body of the **while** loop. Its objective is simple: manipulate the current system, say φ_i, so that it becomes a quotient system induced by θ_i, and then call Algorithm 3 (ElimMaxVar). For these systems, note that 2^x and 2^y in lines 4–5 correspond to $2^{x_{n-i}}$ and $2^{x_{n-i-1}}$, respectively. Behind the notion of quotient system there are two goals. One of them is to make sure that 2^x and 2^y are not involved in modulo operations. (We will discuss the second goal in Section 6.2.) The **while** loop achieves this goal as follows:

- Since 2^x is greater than every variable in φ_i, every $(w \bmod 2^x)$ can be replaced with w.
- For 2^y instead, we "divide" every variable z that might be larger than it. Observe that z is either x or from the vector $\boldsymbol{z}$ in (I3) of the invariant. The procedure replaces every linear occurrence of z with $x' \cdot 2^y + z'$, where x' and z' are fresh variables and z' is a residue modulo 2^y, that is, $0 \leq z' < 2^y$.

The above-mentioned replacement simplifies all modulo operators where z appears: $(z \bmod 2^y)$ becomes z', and every $(z \bmod 2^w)$ such that θ_i entails $2^w \leq 2^y$ becomes $(z' \bmod 2^w)$. We obtain in this way a quotient system induced by θ_i, and pass it to ElimMaxVar.

Whilst the goal we just discussed is successfully achieved, we have not in fact eliminated the variable x completely. Recall that, according to our definition of substitution, occurrences of 2^x in the system φ are unaffected by the application of $[x' \cdot 2^y + z' \,/\, x]$ in line 13 of LinExpSat. Because of this, the procedure keeps this substitution as a *delayed substitution* for future use, to be applied (by ElimMaxVar) when x will finally occur only linearly.

6.2 Algorithm 3: elimination of leading variable and quotient variables

Let $\varphi(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{z}')$ be a quotient system induced by an ordering $\theta(\boldsymbol{x})$, with $\boldsymbol{x}$ exponentiated, $\boldsymbol{x}'$ quotient and $\boldsymbol{z}'$ remainder variables, and consider a delayed substitution $[x' \cdot 2^y + z' \,/\, x]$. ElimMaxVar removes $\boldsymbol{x}'$ and x, obtaining a linear-exponential system ψ that adheres to the loop invariant of LinExpSat. This is done by following the three steps described in the summary of the procedure, which we now expand.

Step (i): lines 3–22. This step aims at calling Algorithm 1 (GaussQE) to eliminate all variables in $\boldsymbol{x}' \setminus x'$. There is, however, an obstacle: these variables are multiplied by 2^y. Here is where the second goal behind the notion of quotient system comes into play: making sure that least significant parts of quotient terms can be bounded in terms of 2^y. To see what we mean by this and why it is helpful, consider below an inequality $\tau \leq 0$ from φ, where $\tau = a \cdot 2^x + f(\boldsymbol{x}') \cdot 2^y + \rho(\boldsymbol{x} \setminus x, \boldsymbol{z}')$ and ρ is the least significant part of τ.

Since φ is a quotient system induced by θ, all variables and exponential terms 2^w appearing in ρ are bounded by 2^y, and thus every solution of $\varphi \wedge \theta$ must also satisfy $|\rho| \leq \|\rho\|_1 \cdot 2^y$. More precisely, the value of ρ must lie in the interval $[(r-1) \cdot 2^y + 1, r \cdot 2^y]$ for some $r \in [-\|\rho\|_1, \|\rho\|_1]$. The procedure guesses one such value r (line 9). The inequality $\tau \leq 0$ can be rewritten as

$$\big(a \cdot 2^x + f(\boldsymbol{x}') \cdot 2^y + r \cdot 2^y \leq 0\big) \;\wedge\; \big((r-1) \cdot 2^y < \rho \leq r \cdot 2^y\big). \tag{3}$$

Fundamentally, $\tau \leq 0$ has been split into a "left part" and a "right part", shown with big brackets around. The "right part" $(r-1) \cdot 2^y < \rho \leq r \cdot 2^y$ is made of two linear-exponential inequalities featuring none of the variables we want to eliminate ($\boldsymbol{x}'$ and x). Following the same principle, the procedure produces similar splits for all strict inequalities, equalities, and divisibility constraints of φ. In the pseudocode, the "left parts" of the system are stored in the formula γ, and the "right parts" are stored in the formula ψ.

Let us focus on a "left part" $a \cdot 2^x + f(\boldsymbol{x}') \cdot 2^y + r \cdot 2^y \leq 0$ in γ. Since θ implies $2^x \geq 2^y$, we can factor out 2^y from this constraint, obtaining the inequality $a \cdot 2^{x-y} + f(\boldsymbol{x}') + r \leq 0$. There we have it: the variables $\boldsymbol{x}' \setminus x'$ occur now linearly in γ and can be eliminated thanks to GaussQE. For performing this elimination, the presence of 2^{x-y} is unproblematic. In fact, the procedure uses a placeholder variable u for 2^{x-y} (line 1), so that γ is in fact a linear system with, e.g., inequalities $a \cdot u + f(\boldsymbol{x}') + r \leq 0$. Observe that inequalities $\boldsymbol{x}' \geq \boldsymbol{0}$ are added to γ in line 22, since GaussQE works over $\mathbb{Z}$ instead of $\mathbb{N}$. This concludes Step (i).

Algorithm 3 ElimMaxVar: Variable elimination for quotient systems.

Input: $\theta(\boldsymbol{x})$: ordering of exponentiated variables;
$\varphi(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{z}')$: quotient system induced by θ, with $\boldsymbol{x}$ exponentiated, $\boldsymbol{x}'$ quotient, and $\boldsymbol{z}'$ remainder variables;
$[x' \cdot 2^y + z'/x]$: delayed substitution for φ.

Output of each branch (β): $\psi_\beta(\boldsymbol{x} \setminus x, \boldsymbol{z}')$: linear-exponential system such that for every z in $\boldsymbol{z}'$, z does not occur in exponentials and $0 \le z < 2^y$ occurs in ψ_β.

Ensuring: $(\exists x\, \theta) \wedge \bigvee_\beta \psi_\beta$ is equivalent to $\exists x \exists \boldsymbol{x}' (\theta \wedge \varphi \wedge x = x' \cdot 2^y + z')$ over $\mathbb{N}$.

let u be a fresh variable ▷ *u is an alias for 2^{x-y}*
$\gamma \leftarrow \top$; $\psi \leftarrow \top$
Elimination of $\boldsymbol{x}' \setminus x'$:
$\Delta \leftarrow \varnothing$ ▷ *map from linear-exponential terms to $\mathbb{Z}$*
foreach $(\tau \sim 0)$ in φ, where $\sim \in \{=, <, \le, \equiv_d \colon d \ge 1\}$ **do**
  let τ be $(a \cdot 2^x + f(\boldsymbol{x}') \cdot 2^y + \rho)$, where ρ is the least significant part of τ
  if $a = 0$ and $f(\boldsymbol{x}')$ is an integer **then** $\psi \leftarrow \psi \wedge (\tau \sim 0)$
  else if the symbol $\sim$ belongs to $\{=, <, \le\}$ **then**
    if $\Delta(\rho)$ is undefined **then**
      guess $r \leftarrow$ integer in $[-\|\rho\|_1, \|\rho\|_1]$
      $\psi \leftarrow \psi \wedge ((r-1) \cdot 2^y < \rho) \wedge (\rho \le r \cdot 2^y)$
      update Δ : add the key–value pair (ρ, r)
    $r \leftarrow \Delta(\rho)$
    if the symbol $\sim$ is $<$ **then**
      $\sim \leftarrow \le$
      $r \leftarrow r + 1$ ▷ *$(v < w)$ is equivalent to $(v + 1 \le w)$*
    $\gamma \leftarrow \gamma \wedge (a \cdot u + f(\boldsymbol{x}') + r \sim 0)$
    if the symbol $\sim$ is $=$ **then** $\psi \leftarrow \psi \wedge (r \cdot 2^y = \rho)$
  else ▷ *$\sim$ is $\equiv_d$ for some $d \in \mathbb{N}$*
    guess $r \leftarrow$ integer in $[1, mod(\varphi)]$
    $\gamma \leftarrow \gamma \wedge (a \cdot u + f(\boldsymbol{x}') - r \sim 0)$
    $\psi \leftarrow \psi \wedge (r \cdot 2^y + \rho \sim 0)$
$\gamma \leftarrow$ GaussQE $(\boldsymbol{x}' \setminus x', \gamma \wedge \boldsymbol{x}' \ge \mathbf{0})$
$\gamma \leftarrow \gamma[2^u \,/\, u]$ ▷ *u now is an alias for $x - y$*
$(\chi, \gamma) \leftarrow$ SolvePrimitive(u, x', γ)
$\chi \leftarrow \chi[x - y \,/\, u][x' \cdot 2^y + z' \,/\, x]$ ▷ *apply delayed substitution: x is eliminated*
Elimination of x':
if χ is $(-x' \cdot 2^y - z' + y + c = 0)$ for some $c \in \mathbb{N}$ **then**
  guess $b \leftarrow$ integer in $[0, c]$
  $\gamma \leftarrow \gamma \wedge (x' = b)$
  $\psi \leftarrow \psi \wedge (b \cdot 2^y = -z' + y + c)$
else
  let χ be $(-x' \cdot 2^y - z' + y + c \le 0) \wedge (d \mid x' \cdot 2^y + z' - y - r)$, with $d, r \in \mathbb{N}$, $c \ge 3$
  guess $(b, g) \leftarrow$ pair of integers in $[0, c] \times [1, d]$
  $\gamma \leftarrow \gamma \wedge (x' \ge b) \wedge (d \mid x' - g)$
  $\psi \leftarrow \psi \wedge ((b-1) \cdot 2^y < -z' + y + c) \wedge (-z' + y + c \le b \cdot 2^y) \wedge (d \mid g \cdot 2^y + z' - y - r)$
assert(GaussQE(x', γ) is equivalent to $\top$) ▷ *upon failure, Algorithm 2 returns $\perp$*
return ψ

Before moving on to Step (ii), we justify the use of the map Δ from line 3. If the procedure were to apply Equation (3) and replace every inequality $\tau \leq 0$ with three inequalities, then multiple calls to ElimMaxVar would produce a system with exponentially many constraints. A solution to this problem is to guess $r \in [-\|\rho\|_1, \|\rho\|_1]$ only once, and use it in all the "left parts" stemming from inequalities in φ having ρ as their least significant part. The "right part" $(r-1)\cdot 2^y < \rho \leq r \cdot 2^y$ is added to ψ only once. The map Δ implements this memoisation, avoiding the aforementioned exponential blow-up. Indeed, the number of least significant parts grows very slowly throughout LinExpSat, as we will see in Section 7.

Step (ii): lines 23–25. The goal of this step is to eliminate all occurrences of the term 2^{x-y}. For convenience, the procedure first reassigns u to now be a placeholder for $x-y$ (line 23). Because of this reassignment, the system γ returned by GaussQE at the end of Step (i) is a (u, x')-primitive linear-exponential system.

The procedure calls Algorithm 4 (SolvePrimitive), which constructs from γ a pair of systems $(\chi_\beta(u), \gamma_\beta(x'))$, which is assigned to (χ, γ). Both are linear systems, and thus all occurrences of 2^{x-y} (rather, 2^u) have been removed. At last, all promised substitutions can be realised (line 25): u is replaced with $x-y$, and the delayed substitution replaces x with $x' \cdot 2^y + z'$. This eliminates x. The only variable that is yet to be removed is x' (Step (iii)).

It is useful to recall at this stage that SolvePrimitive is only correct under the assumption that $u \geq x' \geq 0$. This assumption is guaranteed by the definition of θ, the delayed substitution, and the fact that u is a placeholder for $x-y$ (and we are working over $\mathbb{N}$). Indeed, if $x' = 0$, then the inequality $2^x \geq 2^y$ in θ ensures $u = x - y \geq 0 = x'$. If $x' \geq 1$,

$$\begin{aligned} u = x - y &= x' \cdot 2^y + z' - y && \text{delayed substitution} \\ &\geq x' \cdot (y+1) + z' - y && 2^y \geq y+1, \text{ for every } y \in \mathbb{N} \\ &= y \cdot (x'-1) + x' + z' \geq x'. && \text{since } x' \geq 1. \end{aligned}$$

Step (iii): lines 26–35. This step deals with eliminating the variable x' from the formula $\gamma(x') \wedge \chi(x', z', y) \wedge \psi(\boldsymbol{x} \setminus x, \boldsymbol{z}')$, where ψ contains the "right parts" of φ computed during Step (i). The strategy to eliminate x' follows closely what was done to eliminate the other quotient variables from $\boldsymbol{x}'$ during Step (i): the algorithm first splits the formula $\chi(x', z', y)$ into a "left part", which is added to γ and features the variable x', and a "right part", which is added to ψ and features the variables z' and y. It then eliminates x' by calling GaussQE on γ. To perform the split into "left part" and "right part", observe that χ is a system of the form either $-x' \cdot 2^y - z' + y + c = 0$ or $(-x' \cdot 2^y - z' + y + c \leq 0) \wedge (d \mid x' \cdot 2^y + z' - y - r)$ (see the spec of SolvePrimitive). By arguments similar to the ones used for ρ in Step (i), $-z' + y + c$ can be bounded in terms of 2^y. (Notice, e.g., the similarities between the inequalities in line 34 and the ones in line 10.) After the elimination of x', if GaussQE does not yield an unsatisfiable formula, ElimMaxVar returns the system ψ to LinExpSat.

Before moving on to the description of SolvePrimitive, let us clarify the semantics of the **assert** statement occurring in line 35. It is a standard semantics from programming languages. If an assertion b evaluates to true at runtime, **assert**(b) does nothing. If b evaluates to false instead, the execution aborts and the main procedure (LinExpSat) returns $\perp$. This semantics allows for assertions to query NP problems, as done in line 35 (and in line 11 of SolvePrimitive), without undermining the membership in NP of LinExpSat.

Algorithm 4 SolvePrimitive: A procedure to decompose and linearise primitive systems.

Input: u, v : two varaibles; φ : (u,v)-primitive linear-exponential system.
Output of each branch (β): a pair of linear systems $(\chi_\beta(u), \gamma_\beta(v))$ such that $\chi_\beta(u)$ is either of the form $(u = a)$ or of the form $(u \geq b) \wedge (d \mid u - r)$, where $a, d, r \in \mathbb{N}$ and $b \geq 3$.
Ensuring: $(u \geq v \geq 0)$ entails that $\bigvee_\beta (\chi_\beta \wedge \gamma_\beta)$ is equivalent to φ.

1: **let** φ be $(\chi \wedge \psi)$, where χ is the conjunction of all (in)equalities from φ containing 2^u
2: $(d, n) \leftarrow$ pair of non-negative integers such that $mod(\varphi) = d \cdot 2^n$ and d is odd
3: $C \leftarrow \max\left\{n, 3 + 2 \cdot \left\lceil \log(\frac{|b|+|c|+1}{|a|}) \right\rceil : (a \cdot 2^u + b \cdot v + c \sim 0) \text{ in } \chi, \text{ where } \sim \in \{=, <, \leq\}\right\}$
4: **guess** $c \leftarrow$ element of $[0, C-1] \cup \{\star\}$ ▷ $\star$ *signals* $u \geq C$
5: **if** c is not $\star$ **then**
6: $\quad \chi \leftarrow (u = c)$
7: $\quad \gamma \leftarrow \varphi[2^c \,/\, 2^u]$
8: **else** ▷ *assuming* $u \geq C$, *(in)equalities in* χ *simplify to* $\top$ *or* $\bot$
9: $\quad$ **assert**(χ has no equality, and in all its inequalities 2^u has a negative coefficient)
10: $\quad$ **guess** $r \leftarrow$ integer in $[0, d-1]$ ▷ *remainder of* 2^{u-n} *modulo* d *when* $u \geq C \geq n$
11: $\quad$ **assert**($d \mid 2^u - 2^n \cdot r$ is satisfiable)
12: $\quad r' \leftarrow$ discrete logarithm of $2^n \cdot r$ base 2, modulo d
13: $\quad d' \leftarrow$ multiplicative order of 2 modulo d
14: $\quad \chi \leftarrow (u \geq C) \wedge (d' \mid u - r')$
15: $\quad \gamma \leftarrow \psi[2^n \cdot r / 2^u]$ ▷ $2^n \cdot r$ *is a remainder of* 2^u *modulo* $mod(\psi) = d \cdot 2^n$
16: **return** (χ, γ)

6.3 Algorithm 4: from primitive systems to linear systems

Consider an input (u,v)-primitive linear-exponential system φ, and further assume we are searching for solutions over $\mathbb{N}$ where $u \geq v$. The goal of SolvePrimitive is to decompose φ (in the sense of monadic decomposition [24, 16]) into two *linear* systems: a system χ only featuring the variable u, and a system γ only featuring v.

To decompose φ, the key parameter to understand is the threshold C for the variable u (line 3). This positive integer depends on two quantities, one for "linearising" the divisibility constraints, and one for "linearising" the equalities and inequalities of φ. Below we first discuss the latter quantity. Throughout the discussion, we assume $u \geq C$, as otherwise the procedure simply replaces u with a value in $[0, C-1]$ (see lines 6 and 7).

Consider an inequality $a \cdot 2^u + b \cdot v + c \leq 0$. Regardless of the values of u and v, as long as $|a \cdot 2^u| > |b \cdot v + c|$ holds, the truth of this inequality will solely depend on the sign of the coefficient a. Since we are assuming $u \geq v$ and $u \geq C \geq 1$, $|a \cdot 2^u| > |b \cdot v + c|$ is implied by $|a| \cdot 2^u > (|b| + |c|) \cdot u$. In turn, this inequality is implied by $u \geq C$, because both sides of the inequalities are monotone functions, $|a| \cdot 2^u$ grows faster than $(|b| + |c|) \cdot u$, and, given $C' \coloneqq 3 + 2 \cdot \left\lceil \log(\frac{|b|+|c|+1}{|a|}) \right\rceil$ (which is at most C), we have

$$|a| \cdot 2^{C'} \geq |a| \cdot 2^3 \cdot \left(\frac{|b| + |c| + 1}{|a|}\right)^2 > \big(|b| + |c|\big) \cdot 2^{\left\lceil \log(\frac{|b|+|c|+1}{|a|}) \right\rceil + 2} > \big(|b| + |c|\big) \cdot C',$$

where to prove the last inequalities one uses the fact that $2^{x+1} > 2 \cdot x + 1$ for every $x \geq 0$. Hence, when $u \geq C$, every inequality in φ simplifies to either $\top$ or $\bot$, and this is also true for strict inequalities. The Boolean value $\top$ arises when a is negative. The Boolean $\bot$ arises when a is positive, or when instead of an inequality we consider an equality.

It remains to handle the divisibility constraints, again under the assumption $u \geq C$. This is where the second part of the definition of C plays a role. Because $u \geq C \geq n$ (see the definition of (d, n) in line 2), we can guess $r \in [0, d-1]$ such that $mod(\varphi) \mid 2^u - 2^n \cdot r$ (line 10). This constraint is equivalent to $d \mid 2^{u-n} - r$ and, since 2^n and d are coprime, it is also equivalent to $d \mid 2^u - 2^n \cdot r$. It might be an unsatisfiable constraint: the procedure checks for this eventuality in line 11, by solving a discrete logarithm problem (which can be done in NP, see [18]). Suppose a solution is found, say r' (as in line 12). We can then represent the set of solutions of $d \mid 2^u - 2^n \cdot r$ as an arithmetic progression: it suffices to compute the multiplicative order of 2 modulo d, i.e., the smallest positive integer d' such that $d \mid 2^{d'} - 1$. This is again a discrete logarithm problem, but differently from the previous case d' always exists since d and 2 are coprime. The set of solutions of $d \mid 2^u - 2^n \cdot r$ is given by $\{r' + \lambda \cdot d' : \lambda \in \mathbb{Z}\}$, that is, $mod(\varphi) \mid 2^u - 2^n \cdot r$ is equivalent to $d' \mid u - r'$. The procedure then returns $\chi(u) \coloneqq (u \geq C \wedge d' \mid u - r')$ and $\gamma(v) \coloneqq \psi[2^n \cdot r \,/\, 2^u]$ (see lines 14 and 15), where ψ (defined in line 1) is the system obtained from φ by removing all equalities and inequalities featuring 2^u.

Elaborating the arguments sketched in this section, we can prove that Algorithms 2–4 comply with their specifications.

▶ **Proposition 4.** *Algorithm 2 (LinExpSat) is a correct procedure for deciding the satisfiability of linear-exponential systems over* $\mathbb{N}$.

7 Complexity analysis

We analyse the procedure introduced in Sections 5 and 6 and show that it runs in non-deterministic polynomial time. This establishes Theorem 1 restricted to $\mathbb{N}$.

▶ **Proposition 5.** *Algorithm 2 (LinExpSat) runs in non-deterministic polynomial time.*

To simplify the analysis required to establish Proposition 5, we assume that Algorithms 2–4 store the divisibility constraints $d \mid \tau$ of a system φ in a way such that the coefficients and the constant of τ are always reduced modulo $mod(\varphi)$. For example, if $mod(\varphi) = 5$, the divisibility $5 \mid (7 \cdot x + 6 \cdot 2^x - 1)$ is stored as $5 \mid (2 \cdot x + 2^x + 4)$. Any divisibility can be updated in polynomial time to satisfy this requirement, so there is no loss of generality. Observe that Algorithm 1 (GaussQE) is an exception to this rule, as the divisibility constraints it introduces in line 12 must respect some structural properties throughout its execution. Thus, line 23 of Algorithm 3 (ElimMaxVar) implicitly reduces the output of GaussQE modulo $m = mod(\varphi)$ as appropriate. Since GaussQE runs in non-deterministic polynomial time, the reduction takes polynomial time too.

As is often the case for arithmetic theories, the complexity analysis of our algorithms requires tracking several parameters of linear-exponential systems. Below, we assume an ordering $\theta(\boldsymbol{x}) = (2^{x_n} \geq \cdots \geq 2^{x_0} = 1)$ and let φ be either a linear-exponential system or a quotient system induced by θ. Here are the parameters we track:

- The least common multiple of all divisors $mod(\varphi)$, defined as in Section 3.
- The number of equalities, inequalities and divisibility constraints in φ, denoted by $\#\varphi$. (Similarly, given a set T, we write $\#T$ for its cardinality.)
- The 1-norm $\|\varphi\|_1 \coloneqq \max\{\|\tau\|_1 : \tau \text{ is a term appearing in an (in)equality of } \varphi\}$. For linear-exponential terms, $\|\tau\|_1$ is defined in Section 3. For quotient terms τ induced by θ, the 1-norm $\|\tau\|_1$ is defined as the sum of the absolute values of all the coefficients and constants appearing in τ. The definition of $\|\varphi\|_1$ excludes integers appearing in divisibility constraints since, as explained above, those are already bounded by $mod(\varphi)$.

- The *linear norm* $\|\varphi\|_{\mathfrak{L}} \coloneqq \max\{\|\tau\|_{\mathfrak{L}} : \tau \text{ is a term appearing in an (in)equality of } \varphi\}$. For a linear-exponential term $\tau = \sum_{i=1}^{n} \left(a_i \cdot x_i + b_i \cdot 2^{x_i} + \sum_{j=1}^{n} c_{i,j} \cdot (x_i \bmod 2^{x_j})\right) + d$, we define $\|\tau\|_{\mathfrak{L}} \coloneqq \max\{|a_i|, |c_{i,j}| : i, j \in [1,n]\}$, that is, the maximum of all coefficients of x_i and $(x_i \bmod 2^{x_j})$, in absolute value. For a quotient term induced by θ, of the form $\tau = a \cdot 2^{x_n} + (c_1 \cdot x'_1 + \cdots + c_m \cdot x'_m + d) \cdot 2^{x_{n-1}} + b \cdot x_{n-1} + \rho(x_0, \ldots, x_{n-2}, \boldsymbol{z}')$, we define $\|\tau\|_{\mathfrak{L}} \coloneqq \max\big(|b|, \|\rho\|_{\mathfrak{L}}, \max\{|c_i| : i \in [1,m]\}\big)$, thus also taking into account the coefficients of the quotient variables $x'_1, \ldots, x'_m$.
- The set of the least significant terms $lst(\varphi, \theta)$ defined as $\big\{ \pm\rho : \rho \text{ is the least significant part of a term } \tau \text{ appearing in an (in)equality } \tau \sim 0 \text{ of } \varphi, \text{ with respect to } \theta \big\}$. We have already defined the notion of the least significant part for a quotient term induced by θ in Section 5. For a (non-quotient) linear-exponential system φ, the least significant part of a term $a \cdot 2^{x_n} + b \cdot x_n + \tau'(x_1, \ldots, x_{n-1}, \boldsymbol{z})$ is the term $b \cdot x_n + \tau'$.

Two observations are in order. First, the bit size of a system $\varphi(x_1, \ldots, x_n)$ (i.e., the number of bits required to write down φ) is in $O(\#\varphi \cdot n^2 \cdot \log(\max(\|\varphi\|_1, mod(\varphi), 2)))$. Second, together with the number of variables in the input, our parameters are enough to bound all guesses in the procedure. For instance, the value of $c \neq \star$ guessed in line 4 of Algorithm 4 (SolvePrimitive) can be bounded as $O(\log(\max(mod(\gamma), \|\chi\|_1)))$.

The analysis of the whole procedure is rather involved. Perhaps a good overall picture of this analysis is given by the evolution of the parameters at each iteration of the main **while** loop of LinExpSat, described in Lemma 6 below. This loop iterates at most n times, with n being the number of variables in the input system. Below, Φ stands for Euler's totient function, arising naturally because of the computation of multiplicative orders in SolvePrimitive.

▶ **Lemma 6.** *Consider the execution of* LinExpSat *on an input $\varphi(x_1, \ldots, x_n)$, with $n \geq 1$. For $i \in [0,n]$, let (φ_i, θ_i) be the pair of a system and ordering obtained after the ith iteration of the **while** loop of line 3, where $\varphi_0 = \varphi$ and θ_0 is the ordering guessed in line 2. Then, for every $i \in [0, n-1]$, φ_{i+1} has at most $n+1$ variables, and for every $\ell, s, a, c, d \geq 1$,*

$$\textit{if} \quad \begin{cases} \#lst(\varphi_i, \theta_i) & \leq \ell \\ \#\varphi_i & \leq s \\ \|\varphi_i\|_{\mathfrak{L}} & \leq a \\ \|\varphi_i\|_1 & \leq c \\ mod(\varphi_i) & \mid d \end{cases} \quad \textit{then} \quad \begin{cases} \#lst(\varphi_{i+1}, \theta_{i+1}) & \leq \ell + 2(i+2) \\ \#\varphi_{i+1} & \leq s + 6(i+2) + 2 \cdot \ell \\ \|\varphi_{i+1}\|_{\mathfrak{L}} & \leq 3 \cdot a \\ \|\varphi_{i+1}\|_1 & \leq 2^5 (i+3)^2 (c+2) + 4 \cdot \log(d) \\ mod(\varphi_{i+1}) & \mid \operatorname{lcm}(d, \Phi(\alpha_i \cdot d)) \end{cases}$$

*for some $\alpha_i \in [1, (3 \cdot a + 2)^{(i+3)^2}]$. The $(i+1)$st iteration of the **while** loop of line 3 runs in non-deterministic polynomial time in the bit size of φ_i.*

We iterate the bounds in Lemma 6 to show that, for every $i \in [0,n]$, the bit size of φ_i is polynomial in the bit size of the initial system φ. A challenge is to bound $mod(\varphi_i)$, which requires studying iterations of the map $x \mapsto \operatorname{lcm}(x, \Phi(\alpha \cdot x))$, where α is some positive integer. We show the following lemma:

▶ **Lemma 7.** *Let $\alpha \geq 1$ be in $\mathbb{N}$. Consider the integer sequence $b_0, b_1, \ldots$ given by the recurrence $b_0 \coloneqq 1$ and $b_{i+1} \coloneqq \operatorname{lcm}(b_i, \Phi(\alpha \cdot b_i))$. For every $i \in \mathbb{N}$, $b_i \leq \alpha^{2 \cdot i^2}$.*

Given Lemma 6, one can show $\alpha_j \leq (\|\varphi\|_{\mathfrak{L}} + 2)^{O(j^3)}$ for every $j \in [0, n-1]$. Then, since $mod(\varphi_0) = 1$, for a given $i \in [0, n-1]$ we apply Lemma 7 with $\alpha = \operatorname{lcm}(\alpha_0, \ldots, \alpha_i)$ to derive $mod(\varphi_{i+1}) \leq (\|\varphi\|_{\mathfrak{L}} + 2)^{O(i^6)}$. Once a polynomial bound for the bit size of every φ_i is established, Proposition 5 follows immediately from the last statement of Lemma 6.

8 Proofs of Theorem 1 and Theorem 2

In this section, we discuss how to reduce the task of solving linear-exponential systems over $\mathbb{Z}$ to the non-negative case, thus establishing Theorem 1. We also prove Theorem 2.

Solving linear-exponential systems over $\mathbb{Z}$ (proof of Theorem 1). Let $\varphi(x_1, \ldots, x_n)$ be a linear-exponential system $\varphi(x_1, \ldots, x_n)$ (without divisibility constraints). We can non-deterministically guess which variables will, in an integer solution $\boldsymbol{u} \in \mathbb{Z}^n$ of φ, assume a non-positive value. Let $I \subseteq [1, n]$ be the set of indices corresponding to these variables. Given $i \in I$, all occurrences of $(x \bmod 2^{x_i})$ in φ can be replaced with 0, by definition of the modulo operator. We can then replace each linear and exponentiated occurrence of x_i with $-x_i$. Let $\chi(\boldsymbol{x})$ be the system obtained from φ after these replacements.

The absolute value of all entries of $\boldsymbol{u}$ is a solution for χ over $\mathbb{N}$. However, χ might feature terms of the form 2^{-x_i} for some $i \in I$ and thus not be a linear-exponential system. We show how to remove such terms. Consider an inequality of the form $\tau \leq \sigma$, where the term τ contains no 2^{-x} and $\sigma \coloneqq \sum_{i \in I} a_i \cdot 2^{-x_i}$ with some a_i non-zero. Since each x_i is a non-negative integer, we have $\left|\sum_{i \in I} a_i \cdot 2^{-x_i}\right| \leq \sum_{i \in I} |a_i| =: B$. Therefore, in order to satisfy $\tau \leq \sigma$, any solution $\boldsymbol{v}$ of χ must be such that $\tau(\boldsymbol{v}) \leq B$. We can then non-deterministically add to χ either $\tau < -B$ or $\tau = g$, for some $g \in [-B, B]$.

Case $\tau < -B$. The inequality $\tau \leq \sigma$ is entailed by $\tau < -B$ and can thus be eliminated.

Case $\tau = g$ for some $g \in [-B, B]$. We replace $\tau \leq \sigma$ with $g \leq \sigma$, and multiply both sides of this inequality by $2^{\sum_{i \in I} x_i}$. The resulting inequality is rewritten as $g \cdot 2^z \leq \sum_{i \in I} a_i \cdot 2^{z_i}$, where z and all z_i are fresh variables (over $\mathbb{N}$) that are subject to the equalities $z = \sum_{i \in I} x_i$ and $z_i = \sum_{j \in I \setminus \{i\}} x_j$. We add these equalities to χ.

In the above cases we have removed from χ the inequality $\tau \leq \sigma$ in favour of inequalities and equalities only featuring linear-exponential terms. Strict inequalities $\tau < \sigma$ can be handled analogously; and for equalities $\tau = \sigma$ one can separately consider $\tau \leq \sigma$ and $-\tau \leq -\sigma$. The fresh variables z and z_i can be introduced once and reused for all inequalities.

Repeating the process above for each equality and inequality yields (in non-deterministic polynomial time) a linear-exponential system ψ that is satisfiable over $\mathbb{N}$ if and only if the input system φ is satisfiable over $\mathbb{Z}$. The satisfiability of ψ is then checked by calling LinExpSat. Hence, correctness and NP membership follow by Propositions 4 and 5, respectively. ◀

Deciding existential Büchi–Semenov arithmetic (proof of Theorem 2). Let φ be a formula in the existential theory of the structure $(\mathbb{N}, 0, 1, +, 2^{(\cdot)}, V_2(\cdot,\cdot), \leq)$ (i.e., Büchi–Semenov arithmetic). By De Morgan's laws, we can bring φ to negation normal form. Negated literals can then be replaced by positive formulae: $\neg V_2(\tau, \sigma)$ becomes $V_2(\tau, z) \wedge \neg(z = \sigma)$ where z is a fresh variable, $\neg(\tau = \sigma)$ becomes $(\tau < \sigma) \vee (\sigma < \tau)$, and $\neg(\tau \leq \sigma)$ becomes $\sigma < \tau$. Next, occurrences of $V_2(\cdot,\cdot)$ and $2^{(\cdot)}$ featuring arguments other than variables can be "flattened" by introducing extra (non-negative integer) variables: e.g., an occurrence of 2^τ can be replaced with 2^z, where z is fresh, subject to conjoining to the formula φ the constraint $z = \tau$. Lastly, recall that $V_2(x, y)$ can be rephrased in terms of the modulo operator via a linear-exponential system $2 \cdot y = 2^v \wedge 2 \cdot (x \bmod 2^v) = 2^v$, where v is a fresh variable.

After the above transformation, we obtain a formula ψ of size polynomial with respect to the original one. This formula is a positive Boolean combination of linear-exponential systems. A non-deterministic polynomial-time algorithm deciding ψ first (non-deterministically) rewrites each disjunction $\varphi_1 \vee \varphi_2$ occurring in ψ into either φ_1 or φ_2. After this step, each non-deterministic branch contains a linear-exponential system. The algorithm then calls LinExpSat. Correctness and NP membership then follow by Propositions 4 and 5. ◀

9 Future directions

We have presented a quantifier elimination procedure that decides in non-deterministic polynomial time whether a linear-exponential system has a solution over $\mathbb{Z}$. As a by-product, this result shows that satisfiability for existential Büchi–Semenov arithmetic belongs to NP. We now discuss further directions that, in view of our result, may be worth pursuing.

As mentioned in Section 2, the $\exists^*\forall^*$-fragment of Büchi–Semenov arithmetic is undecidable. Between the existential and the $\exists^*\forall^*$-fragments lies, in a certain sense, the optimisation problem: minimising or maximising a variable subject to a formula. It would be interesting to study whether the natural optimisation problem for linear-exponential systems lies within an optimisation counterpart of the class NP.

With motivation from verification questions, problems involving integer exponentiation have recently been approached with satisfiability modulo theories (SMT) solvers [12]. The algorithms developed in our paper may be useful to further the research in this direction.

Our work considers exponentiation with a single base. In a recent paper [17], Hieronymi and Schulz prove the first–order theory of $(\mathbb{N}, 0, 1, +, 2^{\mathbb{N}}, 3^{\mathbb{N}}, \leq)$ undecidable, where $k^{\mathbb{N}}$ is the predicate for the powers of k. Therefore, the first-order theories of the structures $(\mathbb{N}, 0, 1, +, V_2, V_3, \leq)$ and $(\mathbb{N}, 0, 1, +, 2^{(\cdot)}, 3^{(\cdot)}, \leq)$, which capture $2^{\mathbb{N}}$ and $3^{\mathbb{N}}$, are undecidable. Decidability for the existential fragments of all the theories in this paragraph is open.

Lastly, it is unclear whether there are interesting relaxed versions of linear-exponential systems, i.e., over $\mathbb{R}$ instead of $\mathbb{Z}$. Observe that, in the existential theory of the structure $(\mathbb{R}, 0, 1, +, 2^{(\cdot)}, \leq)$, the formula $x = 2^{y'+z'} \wedge y = 2^{y'} \wedge z = 2^{z'}$ defines the graph of the multiplication function $x = y \cdot z$ for positive reals. This "relaxation" seems then only to be decidable subject to (a slightly weaker version of) Schanuel's conjecture [25]. To have an unconditional result one may consider systems where only one variable occurs exponentiated. These are, in a sense, a relaxed version of (u, v)-primitive systems. Under this restriction, unconditional decidability was previously proved by Weispfenning [40].

References

1 Erwin H. Bareiss. Sylvester's identity and multistep integer-preserving Gaussian elimination. *Math. Comput.*, 22:565–578, 1968. doi:10.1090/S0025-5718-1968-0226829-0.

2 Michael Benedikt, Dmitry Chistikov, and Alessio Mansutti. The complexity of Presburger arithmetic with power or powers. In *ICALP*, pages 112:1–112:18, 2023. doi:10.4230/LIPICS.ICALP.2023.112.

3 Itshak Borosh and Leon Bruce Treybig. Bounds on positive integral solutions of linear Diophantine equations. *Proc. Am. Math. Soc.*, 55(2):299–304, 1976. doi:10.2307/2041711.

4 Véronique Bruyère, Georges Hansel, Christian Michaux, and Roger Villemaire. Logic and p-recognizable sets of integers. *Bull. Belgian Math. Soc.*, 1(2):191–238, 1994. Corrigendum: *Bull. Belgian Math. Soc.*, 1, 1994, 577. doi:10.36045/bbms/1103408547.

5 J. Richard Büchi. Weak second-order arithmetic and finite automata. *Math. Logic Quart.*, 6(1-6):66–92, 1960. doi:10.1002/malq.19600060105.

6 Gregory Cherlin and Françoise Point. On extensions of Presburger arithmetic. In *4th Easter Conference on Model Theory*, volume 86 of *Humboldt-Univ. Berlin Seminarberichte*, pages 17–34, 1986. URL: https://webusers.imj-prg.fr/~francoise.point/papiers/cherlin_point86.pdf.

7 Dmitry Chistikov, Christoph Haase, and Alessio Mansutti. Geometric decision procedures and the VC dimension of linear arithmetic theories. In *LICS*, pages 59:1–59:13, 2022. doi:10.1145/3531130.3533372.

8 Kevin J. Compton and C. Ward Henson. A uniform method for proving lower bounds on the computational complexity of logical theories. *Ann. Pure Appl. Log.*, 48(1):1–79, 1990. doi:10.1016/0168-0072(90)90080-L.

9 David C. Cooper. Theorem proving in arithmetic without multiplication. *Machine Intelligence*, 7(91-99):300, 1972.

10 Charles L. Dodgson. Condensation of determinants, being a new and brief method for computing their arithmetical values. *Proc. R. Soc. Lond.*, 15:150–155, 1867. doi:10.1098/rspl.1866.0037.

11 Andrei Draghici, Christoph Haase, and Florin Manea. Semënov arithmetic, affine VASS, and string constraints. In *STACS*, pages 29:1–29:19, 2024. doi:10.4230/LIPICS.STACS.2024.29.

12 Florian Frohn and Jürgen Giesl. Satisfiability modulo exponential integer arithmetic, 2024. To appear in IJCAR. arXiv:2402.01501.

13 Florent Guépin, Christoph Haase, and James Worrell. On the existential theories of Büchi arithmetic and linear p-adic fields. In *LICS*, pages 1–10, 2019. doi:10.1109/LICS.2019.8785681.

14 Christoph Haase. Approaching arithmetic theories with finite-state automata. In *LATA*, pages 33–43, 2020. doi:10.1007/978-3-030-40608-0_3.

15 Christoph Haase and Jakub Różycki. On the expressiveness of Büchi arithmetic. In *FoSSaCS*, pages 310–323, 2021. doi:10.1007/978-3-030-71995-1_16.

16 Matthew Hague, Anthony W. Lin, Philipp Rümmer, and Zhilin Wu. Monadic decomposition in integer linear arithmetic. In *IJCAR*, pages 122–140, 2020. doi:10.1007/978-3-030-51074-9_8.

17 Philipp Hieronymi and Christian Schulz. A strong version of Cobham's theorem. In *STOC*, pages 1–21, 2022. doi:10.1145/3519935.3519958.

18 Antoine Joux, Andrew Odlyzko, and Cécile Pierrot. The past, evolving present, and future of the discrete logarithm. In *Open Problems in Mathematics and Computational Science*, pages 5–36. Springer, 2014. doi:10.1007/978-3-319-10683-0_2.

19 Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/moor.8.4.538.

20 Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey. *50 years of integer programming 1958–2008: From the early years to the state-of-the-art*. Springer, 2009.

21 Anna Karapiperi, Michela Redivo-Zaglia, and Maria Rosaria Russo. Generalizations of Sylvester's determinantal identity, 2015. arXiv:1503.00519.

22 Dexter Kozen. Lower bounds for natural proof systems. In *FOCS*, pages 254–266, 1977. doi:10.1109/SFCS.1977.16.

23 Aless Lasaruk and Thomas Sturm. Weak integer quantifier elimination beyond the linear case. In *CASC*, pages 275–294, 2007. doi:10.1007/978-3-540-75187-8_22.

24 Leonid Libkin. Variable independence for first-order definable constraints. *ACM Trans. Comput. Log.*, 4(4):431–451, 2003. doi:10.1145/937555.937557.

25 Angus Macintyre and Alex J. Wilkie. On the decidability of the real exponential field. In Piergiorgio Odifreddi, editor, *Kreiseliana. About and Around Georg Kreisel*, pages 441–467. A K Peters, 1996.

26 Derek C. Oppen. Elementary bounds for Presburger arithmetic. In *STOC*, pages 34–37, 1973. doi:10.1145/800125.804033.

27 Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981. doi:10.1145/322276.322287.

28 Françoise Point. On decidable extensions of Presburger arithmetic: from A. Bertrand numeration sytems to Pisot numbers. *J. Symb. Log.*, 65(3):1347–1374, 2000. doi:10.2307/2586704.

29 Françoise Point. On the expansion $(\mathbb{N}; +, 2^x)$ of Presburger arithmetic, 2007. Preprint. URL: https://webusers.imj-prg.fr/~francoise.point/papiers/Pres.pdf.

30 Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du I Congrès des Mathématiciens des Pays Slaves*, pages 92–101. Warsaw, 1929.

31 Sylvain Schmitz. Complexity hierarchies beyond Elementary. *ACM Trans. Comput. Theory*, 8(1):3:1–3:36, 2016. doi:10.1145/2858784.

32 Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

33 Aleksei L. Semenov. On certain extensions of the arithmetic of addition of natural numbers. *Math. USSR Izv.*, 15(2):401–418, 1980. doi:10.1070/im1980v015n02abeh001252.

34 Aleksei L. Semenov. Logical theories of one-place functions on the set of natural numbers. *Math. USSR Izv.*, 22(3):587–618, 1984. doi:10.1070/im1984v022n03abeh001456.

35 Jeffrey Shallit. *The Logical Approach to Automatic Sequences: Exploring Combinatorics on Words with Walnut*. Cambridge University Press, 2022. doi:10.1017/9781108775267.

36 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9, 1973. doi:10.1145/800125.804029.

37 Joachim von zur Gathen and Malte Sieveking. A bound on solutions of linear integer equalities and inequalities. *P. Am. Math. Soc.*, 72(1):155–158, 1978. doi:10.1090/S0002-9939-1978-0500555-0.

38 Volker Weispfenning. The complexity of almost linear Diophantine problems. *J. Symb. Comput.*, 10(5):395–404, 1990. doi:10.1016/S0747-7171(08)80051-X.

39 Volker Weispfenning. Complexity and uniformity of elimination in Presburger arithmetic. In *ISSAC*, pages 48–53, 1997. doi:10.1145/258726.258746.

40 Volker Weispfenning. Deciding linear-exponential problems. *SIGSAM Bull.*, 34(1):30–31, 2000. doi:10.1145/373500.373513.

41 Hao Wu, Yu-Fang Chen, Zhilin Wu, Bican Xia, and Naijun Zhan. A decision procedure for string constraints with string-integer conversion and flat regular constraints. *Acta Inform.*, 2023. doi:10.1007/s00236-023-00446-4.

A Theorem 1 holds for any positive integer base given in binary

Algorithm 2 and Algorithm 3 are agnostic with regard to the choice of the base $k \geq 2$. They do not inspect k and see exponential terms k^x as purely syntactic objects. Their logic does not need to be updated to accommodate a different base. To add support for a base k given in input to these two algorithms, it suffices to replace in the pseudocode every 2 with k.

Algorithm 4 is different, as it uses properties of exponentiation. In that algorithm, line 2 must be updated as follows. The pair (d, n) is redefined to be such that d is the largest integer coprime with k dividing $mod(\gamma)$, and k^n is the smallest power of k divisible by $\frac{mod(\gamma)}{d}$. For example, in the case when $k = 6$ and $mod(\gamma) = 60$, we obtain $d = 5$ and $n = 2$, because 36 is the smallest power of 6 divisible by $\frac{60}{5} = 12$. It is clear that $n \leq \lceil \log(mod(\gamma)) \rceil$, and the pair (d, n) can be computed in deterministic polynomial time.

Apart from this update, it suffices to replace every occurrence of $2^n \cdot r$ with $\frac{mod(\varphi)}{d} \cdot r$, and every remaining occurrence of 2 with k (except for the constant 2 appearing in the expression $3 + 2 \cdot \lceil \log_k(\frac{|b|+|c|+1}{|a|}) \rceil$). This means that the discrete logarithm problems of lines 11–13 must be solved with respect to k instead of 2 (but this can still be done in non-deterministic polynomial time). No other change is necessary.

Finite-Memory Strategies for Almost-Sure Energy-MeanPayoff Objectives in MDPs

Mohan Dantam
School of Informatics, University of Edinburgh, UK

Richard Mayr
School of Informatics, University of Edinburgh, UK

Abstract

We consider finite-state Markov decision processes with the combined Energy-MeanPayoff objective. The controller tries to avoid running out of energy while simultaneously attaining a strictly positive mean payoff in a second dimension.

We show that *finite memory* suffices for almost surely winning strategies for the Energy-MeanPayoff objective. This is in contrast to the closely related Energy-Parity objective, where almost surely winning strategies require infinite memory in general.

We show that exponential memory is sufficient (even for deterministic strategies) and necessary (even for randomized strategies) for almost surely winning Energy-MeanPayoff. The upper bound holds even if the strictly positive mean payoff part of the objective is generalized to multidimensional strictly positive mean payoff.

Finally, it is decidable in pseudo-polynomial time whether an almost surely winning strategy exists.

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains; Mathematics of computing → Probability and statistics

Keywords and phrases Markov decision processes, energy, mean payoff, parity, strategy complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.133

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2404.14522` [18]

1 Introduction

Background. Markov decision processes (MDPs) are a standard model for dynamic systems that exhibit both stochastic and controlled behaviour [28]. MDPs play a prominent role in many domains, e.g., artificial intelligence and machine learning [32, 30], control theory [5, 1], operations research and finance [31, 23, 9, 29], and formal verification [2, 31, 20, 14, 3, 11].

An MDP is a directed graph where states are either controlled or random. If the current state is controlled then the controller can choose a distribution over all possible successor states. If the current state is random then the next state is chosen according to a fixed probability distribution. One assigns numeric rewards to transitions (and this can be generalized to multidimensional rewards). Moreover, priorities (aka colours), encoded by bounded non-negative numbers, are assigned to states. By fixing a strategy for the controller and an initial state, one obtains a probability space of runs of the MDP. The goal of the controller is to optimize the expected value of some objective function on the runs.

The *strategy complexity* of a given objective is the amount of memory (and randomization) needed for an optimal (resp. ε-optimal) strategy. Common cases include memoryless strategies, finite-memory strategies, Markov strategies (using a discrete clock, aka step counter), and general infinite-memory strategies.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 133; pp. 133:1–133:17

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related work. The Parity, MeanPayoff and Energy objectives have been extensively studied in the formal verification community. A run satisfies the (min-even) *Parity objective* iff the minimal priority that appears infinitely often in the run is even. It subsumes all ω-regular objectives, and in particular safety, liveness, fairness, etc. The *MeanPayoff objective* requires that the limit average reward per transition along a run is positive (resp. non-negative in some settings). MeanPayoff objectives go back to a 1957 paper by Gillette [21] and have been widely studied, due to their relevance for efficient control. The *Energy objective* [10] requires that the accumulated reward at any time in a run stays above some finite threshold (typically 0). The intuition is that a controlled system has some finite initial energy level that must never become depleted.

Combinations of these objectives have also been studied, where the runs need to satisfy several of the above conditions simultaneously.

The existence of almost surely winning strategies for *MeanPayoff-Parity* in MDPs is decidable in polynomial time [12]. These strategies require only finite memory for MeanPayoff > 0 [22], but infinite memory for MeanPayoff ≥ 0 [13].

The existence of almost surely winning strategies for *Energy-Parity* in MDPs is decidable in $\mathsf{NP} \cap \mathsf{coNP}$ and in pseudo-polynomial time [25]. (The $\mathsf{NP} \cap \mathsf{coNP}$ upper bound holds even for turn-based stochastic games [26].) Almost surely winning strategies in MDPs require only finite memory in the special case of Energy-Büchi [12], but infinite memory for Energy-co-Büchi and thus for Energy-Parity [25]. However, ε-optimal strategies for Energy-Parity require only finite (at most doubly exponential) memory, and the value can be effectively approximated in doubly exponential time (even for turn-based stochastic games) [17].

The *Energy-MeanPayoff* objective is similar to Energy-Parity, but replaces the Parity part by a MeanPayoff objective for a second reward dimension. I.e., one considers an MDP with 2-dimensional transition rewards, where the Energy condition applies to the first dimension and the MeanPayoff condition applies to the second dimension. (It can be generalized to higher dimensions d, where the MeanPayoff condition applies to all dimensions $2, 3, \ldots, d$.) This might look like a direct generalization of the Energy-Parity objective, since Parity games are reducible to MeanPayoff games [27, 24]. However, this reduction does not work in the context of these combined objectives when one considers stochastic systems like MDPs; see below. Non-stochastic Energy-MeanPayoff games have been studied in [8].

A sightly different objective has been studied in [16] who consider MDPs with d-dimensional rewards, where $d = d_1 + d_2$. The objective requires a strictly positive MeanPayoff *surely* in the first d_1 dimensions, and *almost surely* in the remaining d_2 dimensions. This objective is strictly stronger than Energy-MeanPayoff. E.g., a MeanPayoff of zero in the first dimension may or may not satisfy the Energy objective, but it never satisfies the objective in [16].

The objective studied in [6] aims to maximize the expected MeanPayoff (rather than the probability of it being strictly positive) while satisfying the energy constraint. However, unlike in our work, the reward function has a single dimension (i.e., both criteria apply to the same value) and ε-optimal strategies can require infinite memory.

Our contribution. We consider the Energy-MeanPayoff objective in MDPs with d-dimensional rewards. The first dimension needs to satisfy the Energy condition (never drop below 0), while each other dimension needs to have a *strictly* positive MeanPayoff. We show that

almost surely winning strategies for Energy-MeanPayoff require only *finite* memory. [1] This is in contrast to the Energy-Parity objective where almost surely winning strategies require infinite memory in general [25, Page 4] (even for the simpler Energy-co-Büchi objectives). This also shows that Energy-Parity is not reducible to Energy-MeanPayoff in MDPs, unlike the reduction from Parity to MeanPayoff in [27, 24].

We show that almost surely winning strategies for Energy-MeanPayoff, if they exist, can be chosen as deterministic strategies with an exponential number of memory modes. The crucial property is that it suffices to remember the stored energy only up to some exponential upper bound. A small counterexample shows the corresponding exponential lower bound. Even for randomized strategies, an exponential number of memory modes is required, and this holds even for the case of small transition rewards in $\{-1, 0, +1\}$.

Although almost surely winning strategies are "exponentially large" in this sense, their existence is still decidable in pseudo-polynomial time; cf. Section 7.

2 Preliminaries

A *probability distribution* over a countable set S is a function $f\colon S \to [0,1]$ with $\sum_{s\in S} f(s) = 1$. $\mathsf{supp}(f) \stackrel{\text{def}}{=} \{s \mid f(s) > 0\}$ denotes the support of f and $\mathcal{D}(S)$ is the set of all probability distributions over S. Given an alphabet Σ, let Σ^ω and Σ^* (Σ^+) denote the set of infinite and finite (non-empty) sequences over Σ, respectively. Elements of Σ^ω or Σ^* are called words.

MDPs and Markov chains. A *Markov Decision Process* (MDP) is a controlled stochastic directed graph $\mathcal{M} \stackrel{\text{def}}{=} (S, S_\Box, S_\bigcirc, E, P, \boldsymbol{r})$ where the set of vertices S (also called states) is partitioned into the states $S_\Box$ of the player $\Box$ (*Maximizer*), and chance vertices (aka random states) $S_\bigcirc$. Let $E \subseteq S \times S$ be the transition relation. We write $s \longrightarrow s'$ if $(s, s') \in E$ and assume that $\mathsf{Succ}(s) \stackrel{\text{def}}{=} \{s' \mid sEs'\} \neq \emptyset$ for every state s. The *probability function* P assigns each random state $s \in S_\bigcirc$ a distribution over its successor states, i.e., $P(s) \in \mathcal{D}(\mathsf{Succ}(s))$. We extend the domain of P to $S^*S_\bigcirc$ by $P(\rho s) \stackrel{\text{def}}{=} P(s)$ for all $\rho s \in S^+S_\bigcirc$. A *Markov chain* is an MDP with only random states, i.e., $S_\Box = \emptyset$. In this paper we consider finite-state MDPs, i.e., the set of states S is finite.

Strategies. A *run* is an infinite sequence $s_0 s_1 \ldots \in S^\omega$ such that $s_i \longrightarrow s_{i+1}$ for all $i \geq 0$. A *path* is a finite prefix of a run. Let $\mathit{Runs}(\mathcal{M}) \stackrel{\text{def}}{=} \left\{\rho = (q_i)_{i\in\mathbb{N}} \mid q_i \longrightarrow q_{i+1}\right\}$ denote the set of all possible runs. A strategy of the player $\Box$ is a function $\sigma\colon S^*S_\Box \to \mathcal{D}(S)$ that assigns to every path $ws \in S^*S_\Box$ a probability distribution over the successors of s. If these distributions are always Dirac then the strategy is called *deterministic* (aka pure), otherwise it is called *randomized* (aka mixed). The set of all strategies of player $\Box$ in $\mathcal{M}$ is denoted by $\Sigma^{\mathcal{M}}$. A run/path $s_0 s_1 \ldots$ is compatible with a strategy σ if $s_{i+1} \in \mathsf{supp}(\sigma(s_0 \ldots s_i))$ whenever $s_i \in S_\Box$. Finite-memory strategies are a subclass of strategies using a finite set M of memory modes. A function $\mathsf{nxt} : \mathsf{M} \times S_\Box \mapsto \mathcal{D}(S)$ chooses a (distribution over) successor states based on the current memory mode and state and $\mathsf{upd} : \mathsf{M} \times E \mapsto \mathcal{D}(\mathsf{M})$ updates the memory mode upon observing a transition. Let $\sigma[\mathsf{m}]$ denote the finite-memory strategy σ starting in

[1] Our results do *not* carry over to Energy-MeanPayoff objectives with *non-strict* inequalities where one just requires a MeanPayoff ≥ 0 almost surely. This needs infinite memory even for the case of $d = 2$, i.e., one energy-dimension and one MeanPayoff-dimension. It suffices to modify the counterexample for Energy-co-Büchi from [25, Page 4] such that a visit to a state with unfavourable colour incurs a reward of -1 in the MeanPayoff-dimension.

memory mode m. The set of all finite-memory strategies in $\mathcal{M}$ is denoted by $\Sigma_f^{\mathcal{M}}$. Strategies with memory $|\mathsf{M}| = 1$ are called *memoryless*. Memoryless deterministic (resp. randomized) strategies are called MD (resp. MR). By fixing some finite-memory strategy σ from some initial state in a finite-state MDP $\mathcal{M}$, we obtain a finite-state Markov chain, denoted by $\mathcal{M}^\sigma$.

Measure. An MDP $\mathcal{M}$ with initial state s_0 and strategy σ yields a probability space $(s_0 S^\omega, \mathcal{F}_{s_0}, \mathcal{P}^{\mathcal{M}}_{\sigma,s_0})$ where $\mathcal{F}_{s_0}$ is the σ-algebra generated by the cylinder sets $s_0 s_1 \dots s_n S^\omega$ for $n \geq 0$. The probability measure $\mathcal{P}^{\mathcal{M}}_{\sigma,s_0}$ is first defined on the cylinder sets. For $\rho = s_0 \dots s_n$, let $\mathcal{P}^{\mathcal{M}}_{\sigma,s_0}(\rho) \stackrel{\text{def}}{=} 0$ if ρ is not compatible with σ and otherwise $\mathcal{P}^{\mathcal{M}}_{\sigma,s_0}(\rho S^\omega) \stackrel{\text{def}}{=} \prod_{i=0}^{n-1} \tau(s_0 \dots s_i)(s_{i+1})$ where τ is σ or P depending on whether $s_i \in S_\Box$ or $S_\bigcirc$, respectively. If $\mathcal{M}$ is a Markov chain then there is only a single strategy, and we simply write $\mathcal{P}^{\mathcal{M}}_{s_0}$. By Carathéodory's extension theorem [4], this defines a unique probability measure on the σ-algebra. Given some reward function $v : s_0 S^\omega \to \mathbb{R}$, we write $\mathcal{E}(.)$ for the expectation w.r.t. $\mathcal{P}$ and v.

Objectives. General objectives are defined by real-valued measurable functions. However, we mostly consider indicator functions of measurable sets. Hence, our objectives can be described by measurable subsets $\mathtt{O} \subseteq S^\omega$ of runs starting at a given initial state. By $\mathcal{P}^{\mathcal{M}}_{\sigma,s}(\mathtt{O})$ we denote the payoff under σ, i.e., the probability that runs from s belong to $\mathtt{O}$. The value of a state is defined as $\mathtt{val}^{\mathcal{M}}_{\mathtt{O}}(s) \stackrel{\text{def}}{=} \sup_{\sigma \in \Sigma^{\mathcal{M}}} \mathcal{P}^{\mathcal{M}}_{\sigma,s}(\mathtt{O})$. For $\varepsilon > 0$ and state s, a strategy $\sigma \in \Sigma^{\mathcal{M}}$ is ε-optimal iff $\mathcal{P}^{\mathcal{M}}_{\sigma,s}(\mathtt{O}) \geq \mathtt{val}^{\mathcal{M}}_{\mathtt{O}}(s) - \varepsilon$. A 0-optimal strategy is called *optimal*. An MD/MR strategy is called *uniformly* ε-optimal (resp. uniformly optimal) if it is so from every start state. An optimal strategy from s is called *almost surely winning* if $\mathtt{val}^{\mathcal{M}}_{\mathtt{O}}(s) = 1$. By $\mathtt{AS}(\mathtt{O})$ (resp. $\mathtt{AS}_f(\mathtt{O})$) we denote the set of states that have an almost surely winning strategy (resp. an almost surely winning finite-memory strategy) for objective $\mathtt{O}$. For ease of presentation, we drop subscripts and superscripts wherever possible if they are clear from the context.

We use the syntax and semantics of the LTL operators [15] F (eventually), G (always) and X (next) to specify some conditions on runs. A reachability objective is defined by a set of target states $T \subseteq S$. A run $\rho = s_0 s_1 \dots$ belongs to $\mathsf{F}\, T$ iff $\exists i \in \mathbb{N}\, s_i \in T$. Similarly, ρ belongs to $\mathsf{F}^{\leq n} T$ (resp. $\mathsf{F}^{\geq n} T$) iff $\exists i \leq n$ (resp. $i \geq n$) such that $s_i \in T$. Dually, the safety objective $\mathsf{G}\, T$ consists of all runs which never leave T. We have $\mathsf{G}\, T = \neg \mathsf{F} \neg T$.

Energy/Reward/Counter-based objectives. Let $r : E \to \{-R, \dots, 0, \dots, R\}$ be a bounded function that assigns rewards to transitions. Depending on context, the sum of these rewards in a path can be viewed as energy, cost/profit or a counter. If $s \longrightarrow s'$ and $r((s,s')) = c$, we write $s \xrightarrow{c} s'$. Let $\rho = s_0 \xrightarrow{c_0} s_1 \xrightarrow{c_1} \dots$ be a run. We say that ρ satisfies

1. the *k-energy* objective $\mathsf{EN}(k)$ iff $\left(k + \sum_{i=0}^{n-1} c_i\right) \geq 0$ for all $n \geq 0$.
2. the *l-storage condition* $\mathtt{Infix}(l)$ if $l + \sum_{i=m}^{n-1} c_i \geq 0$ holds for every infix $s_m \xrightarrow{c_m} s_{m+1} \dots s_n$ of the run. Let $\mathsf{ST}(k,l)$ denote the set of runs that satisfy both the k-energy and the l-storage condition. Let $\mathsf{ST}(k) \stackrel{\text{def}}{=} \bigcup_l \mathsf{ST}(k,l)$. Clearly, $\mathsf{ST}(k) \subseteq \mathsf{EN}(k)$.
3. *Mean payoff* $\mathtt{MP}(\rhd c)$ for some constant $c \in \mathbb{R}$ iff $\left(\liminf_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} c_i\right) \rhd\ c$ for $\rhd \in \{<, \leq, =, \geq, >\}$.

A different way to consider the energy objective is to encode the energy level (the sum of the transition weights so far) into the state space and then consider the obtained infinite-state game with a safety objective.

An objective $\mathtt{O}$ is called *shift-invariant* iff for all finite paths ρ and plays $\rho' \in S^\omega$, we have $\rho\rho' \in \mathtt{O} \iff \rho' \in \mathtt{O}$. Mean payoff objectives are shift-invariant, but energy and storage/infix objectives are not.

Multidimensional reward-based objectives. Let $\mathbb{N}, \mathbb{Q}, \mathbb{R}$ denote the set of positive integers, rationals and reals respectively. For a d-dimensional real vector $\boldsymbol{\mu}$, let μ_i denote the i^{th} component of $\boldsymbol{\mu}$ for $1 \leq i \leq d$. Given two vectors $\boldsymbol{\mu}, \boldsymbol{\nu} \in \mathbb{R}^d$, $\sim \in \{<, \leq, >, \geq, =\}$ we say $\boldsymbol{\mu} \sim \boldsymbol{\nu}$ if $\mu_i \sim \nu_i$ for every i. In particular, $\boldsymbol{\mu} > \mathbf{0}$ means that *every* component of $\boldsymbol{\mu}$ is strictly greater than 0. For a multidimensional reward function $\boldsymbol{r} : E \to [-R, R]^d$, we can consider any boolean combination of reward based objectives using any components of $\boldsymbol{r}$. For instance, $\mathtt{O}_1 = \mathtt{EN}_1(k) \cap \mathtt{MP}_2(> 0)$ denotes the objective that contains all runs that satisfy $\mathtt{EN}(k)$ in the 1^{st} dimension and $\mathtt{MP}(> 0)$ in the 2^{nd} one. We denote conjunctions of the same objective across different dimensions in vectorized form, with the dimension information in the subscript. Therefore, $\mathtt{EN}_{[a,b]}(\boldsymbol{k}) \cap \mathtt{MP}_{[c,d]}(\boldsymbol{> x})$ denotes the runs where the $\mathtt{EN}_i(k_i)$ objective is satisfied for each $i \in [a, b]$ and the $\mathtt{MP}_j(> x_j)$ objective is satisfied for each $j \in [c, d]$. Given an infinite run $\rho = s_0 \xrightarrow{\boldsymbol{c_0}} s_1 \xrightarrow{\boldsymbol{c_1}} \ldots$, let $X_n(\rho) \stackrel{\text{def}}{=} s_n$ denote the n-th state. Let $\boldsymbol{Y_n}$ be the sum of the rewards in the first n steps, i.e., $\boldsymbol{Y_n}(\rho) \stackrel{\text{def}}{=} \sum_{i=0}^{n-1} \boldsymbol{c_i}$. These become random variables once an initial distribution and a strategy are fixed.

Size of an instance. Given an MDP $\mathcal{M} = (S, S_\Box, S_\bigcirc, E, P, \boldsymbol{r})$ with reward function $\boldsymbol{r} : E \to [-R, R]^d$, its size $|\mathcal{M}|$ is the number of bits used to describe it. Similarly for $|P|$. Transition probabilities and rewards can thus be stored in binary. We call a size pseudo-polynomial in $|\mathcal{M}|$ if it is polynomial for the case where R is "small", i.e., if R is given in unary.

3 The Main Result

▶ **Theorem 1.** *Let $\mathcal{M} = (S, S_\Box, S_\bigcirc, E, P, \boldsymbol{r})$ be an MDP with d-dimensional rewards on the edges $\boldsymbol{r} : E \to [-R, R]^d$. For the multidimensional Energy-MeanPayoff objective $\mathtt{EN}_1(k) \cap \mathtt{MP}_{[2,d]}(\boldsymbol{> 0})$ the following properties hold.*

1. *The existence of an almost-surely winning strategy implies the existence of an almost-surely winning finite-memory strategy.*
2. *Moreover, a deterministic strategy with an exponential number of memory modes is sufficient.*
3. *An exponential (in $|P|$) number of memory modes is necessary in general, even for randomized strategies, even for $|S| = 5$, $d = 2$ and $R = 1$.*

In the following three sections we prove items 1.,2.,3. of Theorem 1, respectively.

Here we sketch the main idea for the upper bound. Except in a special corner case where the energy fluctuates only in a bounded region, almost-surely winning strategies for Energy-MeanPayoff can be chosen among some particular strategies that alternate between two modes, playing two different memoryless strategies. This alternation keeps the balance between the Energy-part and the MeanPayoff-part of the objective. This is similar to almost-surely winning strategies for the Energy-Parity objective in [25]. In one mode, one plays a randomized memoryless strategy that almost surely yields a positive mean payoff in all dimensions (in case of Energy-Parity, instead of mean payoff it satisfies Parity almost surely). This is called the *Gain* phase. Whenever the energy level (the cumulative reward in dimension 1) gets dangerously close to zero, one switches to the other mode and plays a different memoryless strategy that focuses exclusively on getting the energy level up again, while temporarily neglecting the other part of the objective (Parity or Mean payoff, respectively). This is called a *Bailout*. Once the energy level is sufficiently high, one switches back to the Gain phase again. The crucial property is that, except in a null set, only finitely many Bailouts are required, and thus the temporary neglect of the second part of the objective does not matter in the long run. Such a strategy uses infinite memory,

because it needs to remember the unbounded energy level. For Energy-Parity (and even Energy-co-Büchi) this cannot be avoided and finite-memory strategies do not work [25]. However, for Energy-MeanPayoff one can relax the requirements somewhat. Suppose that one records the stored energy only up to a certain bound b, i.e., one forgets about potential excess energy above b. In that case, one might have to do infinitely many Bailouts with high probability, most of which are unnecessary (but one does not know which ones). However, for a sufficiently large bound b, these superfluous Bailouts occur so infrequently that they do not compromise the MeanPayoff-part of the objective. The critical part of the proof is to show this property and an upper bound on b. Once this is established, one obtains a finite-memory strategy, because it suffices to record the energy level only in the range $[0, b]$ (plus one extra bit of memory to record the current phase, Gain or Bailout).

Note that the argument above is different from the one that justifies finite-memory ε-optimal strategies for *Energy-Parity* in [17]. These also record the energy only in a bounded region, but stop doing Bailouts after the upper bound has been visited. I.e., they do too few Bailouts, and thus incur an ε-chance of losing. In contrast, our almost-surely winning strategies for Energy-MeanPayoff rather do too many Bailouts, but sufficiently infrequently such that they don't compromise the objective.

4 Proof of Item 1

W.l.o.g, we assume that every state in $\mathcal{M}$ has an almost surely winning strategy for Energy-MeanPayoff for some initial energy level. (Otherwise, consider a suitably restricted sub-MDP.) For conciseness, we denote the objective by $\mathtt{O}(k) \stackrel{\text{def}}{=} \mathtt{EN}_1(k) \cap \mathtt{MP}_{[\mathbf{2},d]}(\mathbf{> 0})$. Let

$$\mathtt{Win}(s) \stackrel{\text{def}}{=} \{k \mid s \in \mathtt{AS}(\mathtt{O}(k))\}, \quad i_s \stackrel{\text{def}}{=} \min(\mathtt{Win}(s))$$

denote the possible initial energy levels and the minimum initial energy level such that one can win almost surely from state s. In particular, i_s is well defined by our assumption on $\mathcal{M}$.

Towards a contradiction, assume that not all configurations are winnable with a finite-memory strategy. I.e., let $\mathtt{Win}_f(s) \stackrel{\text{def}}{=} \{k \mid s \in \mathtt{AS}_f(\mathtt{O}(k))\}$ denote the energy levels from which one can win almost surely with a *finite-memory* strategy from s, and assume that there is a state $s^\dagger$ such that $i_{s^\dagger} \notin \mathtt{Win}_f(s^\dagger)$. We then construct a finite-memory winning strategy from $s^\dagger$ for $\mathtt{O}(i_{s^\dagger})$, leading to a contradiction. Similar to i_s, let f_s denote the minimal k such that $k \in \mathtt{Win}_f(s)$ and ∞ if there is no such k.

▶ **Definition 2.** *We construct a new MDP $\mathcal{M}^*$ which abstracts away all the $\mathtt{Win}_f$ configurations. At every state s, the player gets the option to enter a winning sink state if the energy level is sufficiently large to win with finite memory, i.e., if the current energy level is at least f_s. The states of the MDP $\mathcal{M}^*$ will have two copies of each state s of $\mathcal{M}$, namely s and s'. Moreover, we add a new state s_{win}. All states s' are controlled by $\Box$ and every step $s_1 \longrightarrow s$ in the original MDP $\mathcal{M}$ is now mapped to a step $s_1 \longrightarrow s'$ with the same reward (and the same probability if s_1 was a random state). In s', the player has two choices: he can either go to s with reward $\mathbf{0}$ or go to s_{win} with reward $(-f_s, \mathbf{0})$. The latter choice is only available if $f_s < \infty$. s_{win} is a winning sink where $s_{win} \longrightarrow s_{win}$ with reward $\mathbf{1}$, i.e., reward $+1$ in all dimensions.*

The following lemma shows that the existence of almost surely winning (finite-memory) strategies coincides in $\mathcal{M}^*$ and $\mathcal{M}$.

▶ **Lemma 3.** *Let $s \in S$ and $k \in \mathbb{N}$, and consider the objective $\mathtt{O}(k)$. There exists an almost surely winning strategy σ^* from s in $\mathcal{M}^*$ if and only if there exists an almost surely winning strategy σ from s in $\mathcal{M}$. Moreover, if σ^* is finite-memory then σ can be chosen as finite-memory, and vice-versa.*

Proof. Towards the "only if" direction, let σ^* be a strategy from s in $\mathcal{M}^*$ that is almost surely winning for $\mathtt{O}(k)$. We define a strategy σ from s in $\mathcal{M}$ that plays as follows. First σ imitates the moves of σ^* until (if ever) σ^* chooses a move $s'_1 \to s_{\text{win}}$ with non-zero probability at some state s'_1. This is possible, since any finite path in $\mathcal{M}^*$ that does not contain s_{win} can be bijectively mapped to a path in $\mathcal{M}$. The only difference is that paths in $\mathcal{M}^*$ contain extra steps via primed states, which are skipped in the paths in $\mathcal{M}$. Moreover, the transition probabilities at random states coincide in $\mathcal{M}^*$ and $\mathcal{M}$. If σ^* chooses a move $s'_1 \to s_{\text{win}}$ with non-zero probability at some state s'_1 then the current energy level must be $\geq f_{s_1}$, because σ^* satisfies the energy objective almost surely (and thus even surely). Thus, in $\mathcal{M}$, there exists an almost surely winning finite-memory strategy $\hat{\sigma}$ for $\mathtt{O}(f_{s_1})$ from s_1. In this situation σ continues by playing $\hat{\sigma}$ from s_1. Therefore, σ satisfies the energy objective surely. Moreover, by shift invariance and the properties of $\hat{\sigma}$, it also satisfies the Mean payoff objective almost surely. Thus, σ satisfies $\mathtt{O}(k)$ almost surely. Finally, if σ^* is finite-memory then so is σ, because $\hat{\sigma}$ is also finite-memory.

Towards the "if direction, let σ be a strategy from s in $\mathcal{M}$ that is almost surely winning for $\mathtt{O}(k)$. We define a strategy σ^* from s in $\mathcal{M}^*$ that imitates the moves of σ. Moreover, at primed states q' it always goes to q (and never to s_{win}). Since the probabilities at random states coincide in $\mathcal{M}^*$ and $\mathcal{M}$, also the probabilities of the induced paths coincide. The only difference is that the runs in $\mathcal{M}^*$ contain extra steps via primed states and these extra steps carry reward zero. Thus, the mean payoff of a run in $\mathcal{M}^*$ is $1/2$ the mean payoff of the corresponding run in $\mathcal{M}$. However, this does not affect the property that the mean payoff is > 0 almost surely in either MDP. Thus, σ^* satisfies $\mathtt{O}(k)$ almost surely. Finally, if σ is finite-memory then so is σ^*. ◀

The next lemma shows that, in $\mathcal{M}^*$, it is impossible to satisfy Energy-MeanPayoff from s with arbitrarily high probability, unless one also allows arbitrarily large fluctuations in the energy level, or $f_s = i_s$. (Recall that f_s, i_s are defined relative to $\mathcal{M}$.)

▶ **Lemma 4.** *For every state s with $f_s > i_s$ and every $\ell \in \mathbb{N}$, there exists a $\delta_\ell > 0$ such that* $\mathtt{val}^{\mathcal{M}^*}_{\mathtt{O}(i_s) \cap \mathtt{Infix}_1(\ell)}(s) \leq 1 - \delta_\ell$.

Proof. Towards a contradiction, assume that $\mathtt{val}^{\mathcal{M}^*}_{\mathtt{O}(i_s) \cap \mathtt{Infix}_1(\ell)}(s) = 1$ for some ℓ.

$\mathtt{O}(i_s) \cap \mathtt{Infix}_1(\ell) = \mathtt{EN}_1(i_s) \cap \mathtt{MP}_{[\mathbf{2},d]}(\mathbf{> 0}) \cap \mathtt{Infix}_1(\ell) = \mathtt{ST}_1(i_s, \ell) \cap \mathtt{MP}_{[\mathbf{2},d]}(\mathbf{> 0})$. Therefore, we have $\mathtt{val}^{\mathcal{M}^*}_s\big(\mathtt{ST}_1(i_s, \ell) \cap \mathtt{MP}_{[\mathbf{2},d]}(\mathbf{> 0})\big) = 1$. Below we prove that this objective has a finite-memory almost-surely winning strategy σ in $\mathcal{M}^*$. Consider a modified MDP $\mathcal{M}^*_1$ that encodes the energy level up to $i_s + \ell$ in the states. A step exceeding the upper energy bound $i_s + \ell$ results in a truncation to $i_s + \ell$, while a step leading to a negative energy leads to a losing sink. There exists a memoryless randomized (MR) strategy σ_1 in $\mathcal{M}^*_1$ from state (s, i_s) that wins $\mathtt{MP}_{[\mathbf{2},d]}(\mathbf{> 0})$ almost surely, by Lemma 6. We can then carry σ_1 back to $\mathcal{M}^*$ as a finite-memory strategy σ with $i_s + \ell + 1$ memory modes such that $\mathcal{P}^{\mathcal{M}^*}_{\sigma,s}\big(\mathtt{ST}_1(i_s, \ell) \cap \mathtt{MP}_{[\mathbf{2},d]}(\mathbf{> 0})\big) = 1$. By set inclusion, $\mathcal{P}^{\mathcal{M}^*}_{\sigma,s}(\mathtt{O}(i_s)) = 1$. By Lemma 3, there also exists a finite-memory strategy from s in $\mathcal{M}$ that is almost surely winning for $\mathtt{O}(i_s)$. This implies $f_s = i_s$, a contradiction to our assumption $f_s > i_s$. Hence, we obtain $\delta_\ell \overset{\text{def}}{=} 1 - \mathtt{val}^{\mathcal{M}^*}_{\mathtt{O}(i_s) \cap \mathtt{Infix}_1(\ell)}(s) > 0$. ◀

The following three lemmas show that almost surely winning strategies for Energy-MeanPayoff can be found by combining two different memoryless strategies for the simpler `Bailout` and `Gain` objectives.

First, we define the objective $\mathtt{Bailout}(k) \overset{\text{def}}{=} \mathtt{EN}_1(k) \cap \mathtt{MP}_1(> 0)$. Let $i_s^{\mathtt{Bailout}}$ denote the minimal energy value k with which one can almost surely satisfy $\mathtt{Bailout}(k)$ when starting from state s (or ∞ if it does not exist).

▶ **Lemma 5** ([6, Lemma 3]). *Let $\mathcal{M}$ be an MDP. If $s \in \mathtt{AS}(\mathtt{Bailout}(k))$ for some $k \in \mathbb{N}$ then $i_s^{\mathtt{Bailout}} \leq 3 \cdot |\mathcal{M}| \cdot R$. Moreover, there exists a uniform MD strategy $\sigma^*_{\mathtt{Bailout}}$ which is almost surely winning $\mathtt{Bailout}(k)$ from every state $s \in \mathtt{AS}(\mathtt{Bailout}(k))$.*

We define the `Gain` objective as $\mathtt{MP}_{[1,d]}(> 0)$. The following lemma shows that an almost surely winning strategy $\sigma^*_{\mathtt{Gain}}$ for this objective can be chosen as memoryless randomized.

▶ **Lemma 6** ([7, Proposition 5.1]). *There is a uniform MR strategy $\sigma^*_{\mathtt{Gain}}$ which is almost surely winning for `Gain` (or any subset of dimensions) from all states $s \in \mathtt{AS}(\mathtt{Gain})$.*

A difference between $\mathcal{M}^*$ and $\mathcal{M}$ is that if one can almost surely win Energy-MeanPayoff in $\mathcal{M}^*$ then one can also push the energy level arbitrarily high. This does not always hold in $\mathcal{M}$. (Consider, e.g., a single-state Markov chain with a single loop with reward 0 in the 1^{st} dimension and $+1$ in all other dimensions.) The difference comes from the loop at state s_{win} in $\mathcal{M}^*$ which has a strictly positive reward in all dimensions. Thus, the following lemma only holds for $\mathcal{M}^*$.

▶ **Lemma 7.** *In $\mathcal{M}^*$, there are two uniform memoryless strategies $\sigma^*_{\mathtt{Bailout}}$ and $\sigma^*_{\mathtt{Gain}}$ which, starting from any state $s \in \mathtt{AS}(\mathtt{0}(k))$, almost surely satisfy $\mathtt{Bailout}(k)$ and `Gain`, respectively.*

Proof. Let $s \in \mathtt{AS}(\mathtt{0}(k))$. We show that $s \in \mathtt{AS}(\mathtt{Bailout}(k))$ and $s \in \mathtt{AS}(\mathtt{Gain})$. The existence of the memoryless strategies $\sigma^*_{\mathtt{Bailout}}$ and $\sigma^*_{\mathtt{Gain}}$ then follows from Lemma 5 and Lemma 6, respectively.

We assumed that all states s in $\mathcal{M}$ admit an almost surely winning strategy for Energy-MeanPayoff. By Lemma 3, this also holds for all states q in $\mathcal{M}^*$. Let $\sigma^\sharp_q$ denote an almost surely winning strategy from q for $\mathtt{0}(i_q)$ in $\mathcal{M}^*$ (without restrictions on memory).

Recall from Section 2 that the random variable X_t denotes the state at time t, and Y_t denotes the (d-dimensional) sum of the rewards until time t.

▷ Claim 8. For every state $q \in \mathcal{M}^*$ there exists some number of steps $n_q \in \mathbb{N}$ and a probability $p_q > 0$ such that

$$\mathcal{P}^{\mathcal{M}^*}_{\sigma^\sharp_q, q}\left(\bigcup_{j=0}^{n_q} ((Y_j)_1 > i_{X_j} - i_q) \cup ((Y_j)_1 \geq f_{X_j} - i_q)\right) \geq p_q.$$

Proof. Towards a contradiction, assume that for all m

$$\mathcal{P}^{\mathcal{M}^*}_{\sigma^\sharp_q, q}\left(\bigcup_{j=0}^{m} ((Y_j)_1 > i_{X_j} - i_q) \cup ((Y_j)_1 \geq f_{X_j} - i_q)\right) = 0.$$

Due to the second part of the union, this implies that never $(Y_j)_1 + i_q \geq f_{X_j}$. Since $\sigma^\sharp_q$ satisfies $\mathtt{EN}_1(i_q)$ almost surely, it can never choose the step to s_{win}. This implies $\mathcal{P}^{\mathcal{M}^*}_{\sigma^\sharp_q, q}(\mathsf{F} s_{\text{win}}) = 0$, i.e., X_j is always different from s_{win}. (The values f_s were initially defined with respect to states s of the original MDP $\mathcal{M}$, but the definition is naturally extended to the MDP $\mathcal{M}^*$, by giving the primed states the same value, i.e., $f_{s'} = f_s$. The state s_{win} does not appear in $\mathcal{M}$, but only in $\mathcal{M}^*$. We can extend the definition by having $f_{s_{\text{win}}} = 0$. However, this is not strictly required. The f_{X_j} is already defined, since X_j is always different from s_{win}.)

Since $\sigma^\sharp_q$ satisfies $\mathtt{EN}_1(i_q)$ almost surely, all runs always satisfy $(Y_j)_1 \geq i_{X_j} - i_q$ for all j. On the other hand, our assumption yields $\mathcal{P}^{\mathcal{M}^*}_{\sigma^\sharp_q, q}\left(\bigcup_{j=0}^{m} (Y_j)_1 > i_{X_j} - i_q\right) = 0$. This implies that $(Y_j)_1 = i_{X_j} - i_q$ for all j. Hence, in all runs the energy fluctuates by at most $\ell \stackrel{\text{def}}{=} 2 \max_q i_q$.

Thus, $\mathcal{P}^{\mathcal{M}^*}_{\sigma^\sharp_q,q}(\mathtt{O}(i_q) \cap \mathtt{Infix}_1(\ell)) = 1$. Then Lemma 4 implies that $f_q = i_q$. Since $X_0 = q$ we have $f_{X_0} = f_q$ and thus $(Y_0)_1 \geq f_{X_0} - i_q = 0$. This contradicts our assumption, since the second part of the union is surely satisfied. ◁

For any state q, let n_q, p_q denote the values from Claim 8.

Now we show that $s \in \mathtt{AS}(\mathtt{Bailout}(k))$. Define a strategy $\sigma_{\mathtt{Bailout}}$ which plays in phases, separated by resets. It remembers the number of steps $t \geq 0$ since last reset, the (under-approximated) sum of rewards Q_t and the current state X_t. The first phase starts at state s and $\sigma_{\mathtt{Bailout}}$ plays like $\sigma^\sharp_s$ until one of the following events occur.

1. There is enough energy such that it is safe to move to s_{win}, i.e., $(Q_t \geq f_{X_t} - i_s)$, or
2. The current energy level is strictly greater than the minimal required energy level of the current state, i.e., $(Q_t > i_{X_t} - i_s)$, or
3. n_s steps have elapsed, i.e., $(t = n_s)$.

If at any point Item 1 happens, then the strategy simply goes to s_{win}. If it is the case that Item 2 occurs before $t = n_s$, let's say at some time t', then the phase ends at t'. The sum of the rewards in the phase, between the last reset (where $t = 0$) and the current time is $\geq i_{X_{t'}} - i_s + 1$. If neither Item 1 nor Item 2 occurs before $t = n_s$, then the phase ends and we let $t' \stackrel{\text{def}}{=} t = n_s$. The sum of the rewards in this phase is then exactly $i_{X_{t'}} - i_s$. At the end of the phase $\sigma_{\mathtt{Bailout}}$ resets the number of steps ($t = 0$), and Q_t to 0. In the following phase it moves according to $\sigma^\sharp_{X_{t'}}$ until the next reset.

$\sigma_{\mathtt{Bailout}}$ clearly satisfies $\mathtt{EN}_1(k)$ as it is a mix of energy safe strategies $\left(\sigma^\sharp_q\right)_{q \in S^*}$ and since we are starting from a safe energy level. By Claim 8, there is a positive probability (lower-bounded by $\min_q p_q > 0$) that either Item 1 or Item 2 happens in each phase.

Hence, unless event Item 1 occurs, Item 2 occurs infinitely often almost surely. Moreover, since the length of phases is upper bounded by $\max_q n_q$, it occurs frequently. We obtain $\mathcal{P}^{\mathcal{M}^*}_{\sigma_{\mathtt{Bailout}},s}\left(\mathtt{MP}_1 \geq \min_q\left(\frac{p_q}{n_q}\right) > 0 \,|\, \neg \mathsf{F} s_{\text{win}}\right) = 1$. On the other hand, if s_{win} is reached, then $\mathtt{MP}_1$ holds by shift invariance and the definition of the positive rewards in the loop at s_{win}. Therefore, $\mathcal{P}^{\mathcal{M}^*}_{\sigma_{\mathtt{Bailout}},s}(\mathtt{EN}_1(i_s) \cap \mathtt{MP}_1(> 0)) = 1$.

Now we show that $s \in \mathtt{AS}(\mathtt{Gain})$. We make use of the following strategies.

- $\sigma^\sharp_q$ which satisfies $\mathtt{EN}_1(k) \cap \mathtt{MP}_{[\mathbf{2},\boldsymbol{d}]}(\mathbf{> 0})$ almost surely from q for every $k \geq i_q$.
- a uniform MD strategy $\sigma^*_{\mathtt{MP}_1}$ which satisfies $\mathtt{MP}_1(> 0)$ almost surely from every state. It exists since $\mathtt{AS}(\mathtt{MP}_1(> 0)) = S^*$ (where S^* is the set of states of $\mathcal{M}^*$), because $\mathcal{P}^{\mathcal{M}^*}_{\sigma_{\mathtt{Bailout}},s}(\mathtt{EN}_1(i_s) \cap \mathtt{MP}_1(> 0)) = 1$.

From the former, we get probabilistic bounds on the achievable mean payoff in all the dimensions, i.e., for all states s, and $0 \leq \varepsilon < 1$, there is a $d-1$ dimensional vector $\boldsymbol{\nu_\varepsilon} > \mathbf{0}$ such that $\mathcal{P}^{\mathcal{M}^*}_{\sigma^\sharp_s,s}\left(\mathtt{MP}_{[\mathbf{2},\boldsymbol{d}]} \geq \boldsymbol{\nu_\varepsilon}\right) \geq 1 - \frac{\varepsilon}{2}$. This follows from the fact that for any sequence of decreasing vectors $\boldsymbol{\nu_n} \to \mathbf{0}$ in $\mathbb{R}^{d-1}$, $\mathtt{MP}_{[\mathbf{2},\boldsymbol{d}]}(\mathbf{> 0}) = \bigcup_n \mathtt{MP}_{[\mathbf{2},\boldsymbol{d}]}(\mathbf{\geq \boldsymbol{\nu_n}})$ and continuity of measures. Furthermore, denoting by $\boldsymbol{Y_t}$ the sum of rewards in all dimensions until time t, there exists a sufficiently large bound $n_\varepsilon \in \mathbb{N}$ such that $\mathcal{P}^{\mathcal{M}^*}_{\sigma^\sharp_s,s}\left(\frac{(Y_t)_j}{t} \geq \frac{(\nu_\varepsilon)_j}{2}\right) \geq 1 - \varepsilon$ in each of the dimensions $j \in [2,d]$ for all $t \geq n_\varepsilon$ steps. This can be shown by observing that $\mathtt{MP}_j\left(\geq (\nu_\varepsilon)_j\right) = \bigcap_{k=1}^\infty \bigcup_{n=1}^\infty \bigcap_{t=n}^\infty \left(\frac{(Y_t)_j}{t} \geq (\nu_\varepsilon)_j \cdot \left(1 - \frac{1}{2^k}\right)\right)$ and using continuity of measures.

Similarly, there exists a bound $n^*_\varepsilon \in \mathbb{N}$ and value $\nu^*_\varepsilon > 0$ such that $\mathcal{P}_{\sigma^*_{\mathtt{MP}_1},s}\left(\frac{(Y_t)_1}{t} \geq \frac{\nu^*_\varepsilon}{2}\right) \geq 1 - \varepsilon$ after $t \geq n^*_\varepsilon$ steps for every state s.

Now consider the following strategy $\sigma_{\mathtt{Gain}}$, which switches between two phases.

Phase 1: If the current state is q, it moves according to $\sigma^\sharp_q$ for some number $\alpha > n_\varepsilon$ of steps. Then it switches to phase 2.

Phase 2: It moves according to $\sigma^*_{\mathtt{MP}_1}$ for some number $\beta > n^*_\varepsilon$ of steps, and then switches back to phase 1.

The strategy $\sigma_{\mathtt{Gain}}$ is a finite-memory strategy, since the lengths of the alternating phases are bounded by α and β, respectively. (Even if $\sigma^\sharp_q$ is an infinite-memory strategy, it can only use bounded memory in each phase.)

We fix $\sigma_{\mathtt{Gain}}$ from the start state s and obtain a finite-state Markov chain. In every BSCC of this Markov chain, the expected mean payoff in the 1^{st} dimension will be

$$\geq \frac{-i^\sharp + \beta \cdot (1-\varepsilon) \cdot \left(\frac{\nu^*_\varepsilon}{2}\right) - \beta \cdot \varepsilon \cdot R}{\alpha + \beta}.$$

where $i^\sharp = \max_s i_s$ denotes the maximum (over all states) minimal safe energy.

Similarly, in every BSCC, the expected mean payoff in the j^{th} dimension for $j \geq 2$ can be lower-bounded by

$$\geq \frac{\alpha \cdot \left((1-\varepsilon) \cdot \left(\frac{(\nu_\varepsilon)_j}{2}\right) - \varepsilon \cdot R\right) - \beta \cdot R}{\alpha + \beta}.$$

By choosing ε sufficiently small, β sufficiently large to make the first term positive and $\alpha \gg \beta$ sufficiently large to make the second term positive, we can get positive expected mean payoff in all dimensions. Since this holds in every BSCC of the induced finite Markov chain, the objective $\mathtt{Gain}$ is satisfied almost surely. ◀

The following lemma shows the converse of Lemma 7. In $\mathcal{M}^*$, it is always possible to win $\mathtt{O}(i_s)$ almost surely from s by playing a particular strategy $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g}}$ which combines the two uniform memoryless strategies $\sigma^*_{\mathtt{Bailout}}$ and $\sigma^*_{\mathtt{Gain}}$. Let Z_b denote the minimal universally safe energy level for $\mathtt{Bailout}$, i.e., $Z_b \stackrel{\text{def}}{=} \max_s \min\{k \mid s \in \mathtt{AS}(\mathtt{Bailout}(k))\}$. Moreover, let $Z_g > Z_b$ be a larger energy level at which our strategy switches from $\sigma^*_{\mathtt{Bailout}}$ to $\sigma^*_{\mathtt{Gain}}$.

Similarly to [25], we define an infinite-memory strategy $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g}}$ that always records the current energy level and operates by switching between two phases. It starts by playing $\sigma^*_{\mathtt{Gain}}$ (Gain-phase) if our starting energy level is sufficiently high ($\geq Z_b + R$), and otherwise starts by playing $\sigma^*_{\mathtt{Bailout}}$ (Bailout-phase). In the $\mathtt{Bailout}$-phase, the primary goal is to pump the energy level up until it is $\geq Z_g$, and then it switches to the $\mathtt{Gain}$-phase. It enters the $\mathtt{Bailout}$-phase again if the energy level drops below $Z_b + R$ (in which case it will still be $\geq Z_b$).

▶ **Lemma 9.** *There exists a $Z_g \in \mathbb{N}$ such that for every s in $\mathcal{M}^*$ the strategy $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g}}$ is almost surely winning for $\mathtt{O}(i_s)$ from s.*

Proof. The parameter Z_g is chosen sufficiently large such that there is a fixed non-zero probability that after every Bailout-phase one never needs another $\mathtt{Bailout}$. (Thus, except in a null set there are only finitely many Bailouts.) The existence of such a finite Z_g is guaranteed by the fact that $\lim_{k\to\infty} \mathcal{P}_{\sigma^*_{\mathtt{Gain}},s}(\mathtt{O}(k)) = 1$. ([18, Lemma 22]). Eventually, except in a null set, $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g}}$ plays $\mathtt{Gain}$ forever, thus satisfying $\mathtt{O}(i_s)$ almost surely from s. ◀

Some combined objectives like Energy-Parity really require infinite memory for almost surely winning strategies [25]. However, we show that a sufficiently large *finite* memory is enough to win Energy-MeanPayoff almost surely. The idea is to modify the strategy $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g}}$ such that it remembers the current energy only in the interval $[0, b]$, for some sufficiently large $b > Z_g$, and ignores any possible excess energy above b. This modified strategy is denoted by $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}$, and it has a finite set of memory modes $[0, b] \times \{0, 1\}$. The $\{0, 1\}$ part is used to remember the current phase ($\mathtt{Gain} = 0$ or $\mathtt{Bailout} = 1$). Then $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}[(u, x)]$ denotes the strategy $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}$ with current memory mode $(u, x) \in [0, b] \times \{0, 1\}$.

The finite bound b on the remembered energy has the effect that $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}$ can no longer guarantee a fixed positive probability of not needing another Bailout after each Bailout-phase. Thus, one might have infinitely many Bailouts with positive probability. (Most of these are unnecessary, but one cannot be sure which ones). Unlike for Energy-Parity, where using infinitely many `Bailout` phases can compromise the objective, the nature of the $\mathtt{MP}_{[\mathbf{2},\boldsymbol{d}]}(\mathbf{> 0})$ objective allows us to use infinitely many Bailouts with non-zero probability, provided that they happen sufficiently infrequently.

By its construction, the strategy $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}[(i_s, x)]$ is energy-safe from every state s, every initial energy $\geq i_s$ and $x \in \{0, 1\}$. It remains to show that it also satisfies $\mathtt{MP}_{[\mathbf{2},\boldsymbol{d}]}(\mathbf{> 0})$ almost surely. Since $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}$ is finite-memory, it suffices to consider the induced finite Markov chain $\mathcal{A}$ and show that the expected mean payoff is strictly positive in every BSCC. I.e., we prove that $\mathcal{E}_{\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}},s}\big(\mathtt{MP}_{[\mathbf{2},\boldsymbol{d}]}\big) > \mathbf{0}$ for a sufficiently large b. To this end, we consider the finite Markov chains $\mathcal{A}^{\mathtt{Gain}}$ and $\mathcal{A}^{\mathtt{Bailout}}$ obtained by fixing the memoryless strategies $\sigma^*_{\mathtt{Gain}}$ and $\sigma^*_{\mathtt{Bailout}}$ in $\mathcal{M}^*$, respectively. The application of $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}$ can then be seen as alternating between these two Markov chains based on hitting certain energy levels.

Let $T^{\mathtt{Gain}}$ denote the random variable that measures the length of a Gain-phase, when starting at energy level Z_g and assuming that the energy it truncated at b. Similarly, $T^{\mathtt{Bailout}}$ is the random variable that measures the length of a Bailout-phase when starting at energy level Z_b. (Here it does not matter that the energy is truncated at b, since the Bailout-phase ends when the energy reaches $Z_g < b$.) Since R can be > 1, the Bailout-phase might actually start at a slightly higher energy level $u \in [Z_b, Z_b + R - 1]$, and thus $T^{\mathtt{Bailout}}$ over-approximates the actual length of the Bailout-phase, which is conservative for our analysis. Similarly, the Gain phase might start with an energy slightly higher than Z_g, and $T^{\mathtt{Gain}}$ under-approximates the length of the Gain-phase, which is again conservative. The random variables $(Y_{T^{\mathtt{Gain}}})_i$ and $(Y_{T^{\mathtt{Bailout}}})_i$ then measure the sum of the rewards the i^{th} dimension obtained during the Gain and Bailout phases, respectively.

The following lemma shows that the strategy $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}$ can attain a strictly positive mean payoff in all dimensions $i \in [2, d]$, provided that the expected reward during the Gain-phase is sufficiently large (positive) and the expected reward during the Bailout-phase (though possibly negative) is not too small.

▶ **Lemma 10.** *If there are constants $v_i^1 > 0$ and v_i^2 such that, for all $i \in [2, d]$ and states q*

$$\mathcal{E}^{\mathcal{M}^*}_{\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}[(Z_g,0)],q}((Y_{T^{\mathtt{Gain}}})_i) \geq v_i^1$$
$$\mathcal{E}^{\mathcal{M}^*}_{\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}[(Z_b,1)],q}((Y_{T^{\mathtt{Bailout}}})_i) \geq v_i^2$$
$$v_i^1 + v_i^2 > 0$$

then $\mathcal{E}^{\mathcal{M}^*}_{\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}[\mathsf{m}],s}(\mathtt{MP}_i) > 0$ for all s and $\mathsf{m} \in [i_s, b] \times \{0, 1\}$.

Proof. By fixing the finite-memory strategy $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}$, we obtain a finite Markov chain. Consider any BSCC in this Markov chain. In this BSCC, except for a null set of runs, either no Bailouts happen or infinitely many. In the former case, this BSCC behaves like playing $\sigma^*_{\mathtt{Gain}}$ forever, which attains a strictly positive mean payoff in all dimensions almost surely, and thus a strictly positive expected mean payoff in each dimension i. In the second case, almost surely there happen infinitely many Bailouts, each starting at an every level $\geq Z_b$. Then, by the finiteness of the BSCC, we obtain that $\mathcal{E}(T^{\mathtt{Gain}}) < \infty$. Moreover, by the definition of $\sigma^*_{\mathtt{Bailout}}$, the expected duration of the Bailout-phase is always finite, i.e., $\mathcal{E}(T^{\mathtt{Bailout}}) < \infty$. Thus, by linearity of expectations, $\mathcal{E}^{\mathcal{M}^*}_{\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}},s}(\mathtt{MP}_i) \geq (v_i^1 + v_i^2)/(\mathcal{E}(T^{\mathtt{Gain}}) + \mathcal{E}(T^{\mathtt{Bailout}})) > 0$. ◀

The following technical Lemma 11 (proof in [18, Appendix B]) shows that the constants v_i^1, v_i^2 from Lemma 10 exist. Recall that the finite Markov chains $\mathcal{A}^{\mathtt{Gain}}$ and $\mathcal{A}^{\mathtt{Bailout}}$ are obtained by fixing the memoryless strategies $\sigma^*_{\mathtt{Gain}}$ and $\sigma^*_{\mathtt{Bailout}}$ in $\mathcal{M}^*$, respectively. Let $x_{\min,1}$ and $x_{\min,2}$ denote the minimal occurring non-zero probabilities in these two Markov chains, respectively. (They come from solutions of linear programs and can be chosen as only exponentially small, i.e., described by a polynomial number of bits; cf. [18, Appendix B]). The proof works by applying general results about expected first passage times in truncated Markov chains to the induced Markov chains $\mathcal{A}^{\mathtt{Gain}}$ and $\mathcal{A}^{\mathtt{Bailout}}$. The general idea is that in the Gain-phase one has a general up drift in all dimensions, and in particular in the first (energy) dimension. It is thus unlikely to go down very far in the energy dimension, even if the energy is truncated at b. Thus, for a sufficiently large truncation point b (actually $b = Z_g + 1$ suffices), the expected time spent in the Gain-phase is very large relative to the expected time spent in the Bailout phase. More exactly, the former increases exponentially in b, while the latter is polynomial in b. For a sufficiently large b (exponential in $|\mathcal{M}^*|$), the condition $v_i^1 + v_i^2 > 0$ is met.

▶ **Lemma 11.** *Let $\mu_i > 0$ denote the lower bound on the mean payoff in dimensioni in any BSCC in the Markov chain $\mathcal{A}^{\mathtt{Gain}}$ with corresponding computable constants c_i, $g_{\mathtt{Gain}}$, $h_{\mathtt{Gain}}$, and let μ denote the lower bound on the mean payoff in the 1^{st} dimension in any BSCC of $\mathcal{A}^{\mathtt{Bailout}}$ with the corresponding constants $g_{\mathtt{Bailout}}, h_{\mathtt{Bailout}}$. All the above constants, except c_i, can be chosen as at most exponential in $|\mathcal{M}^*|$ and $1/(1-c_i) \in \mathcal{O}\big(\exp\big(\exp\big(|\mathcal{M}^*|^{\mathcal{O}(1)}\big)\big)\big)$.*

Then there are constants $0 < C_1 < 1$, $C_2 > 0$, $C_3 > 0$, $C_4 > 0$, $C_5 > 0$, all exponential in $|\mathcal{M}^*|$ and dependent only on $\mathcal{M}$, such that for $k \stackrel{def}{=} \frac{2\cdot|S^*|}{x_{\min,1}^{|S^*|}} \in \mathcal{O}\big(\exp\big(|\mathcal{M}^*|^{\mathcal{O}(1)}\big)\big)$, any $\delta \in (0,1)$ sufficiently small such that $(|S^*|+1)\cdot\big(\frac{1}{\delta}-1\big) + \lceil\big(\log_{c_i}(\delta\cdot(1-c_i))\big)\rceil \geq \frac{h_{\mathtt{Gain}}}{\mu_i}$ for all $2 \leq i \leq d$, one can choose
$Z_g \stackrel{def}{=} Z_b + R + k{\cdot}R + \max_i\big(R\cdot\lceil\log_{c_i}(\delta\cdot(1-c_i))\rceil - R + 1, h_{\mathtt{Gain}}\big) \in \mathcal{O}\Big(e^{|\mathcal{M}^*|^{\mathcal{O}(1)}}\cdot\log(1/\delta)\Big)$
and $b \stackrel{def}{=} Z_g + 1$ so that

$$\mathcal{E}^{\mathcal{M}^*}_{\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}[(Z_g,0)],q}\big((Y_{T^{\mathtt{Gain}}})_i\big) \geq C_1\cdot\frac{1}{\delta} - C_2\log_2\left(\frac{1}{\delta}\right) - C_3 \qquad \stackrel{def}{=} v_i^1$$

$$\mathcal{E}^{\mathcal{M}^*}_{\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}[(Z_b,1)],q}\big((Y_{T^{\mathtt{Bailout}}})_i\big) \geq -C_4\log_2\left(\frac{1}{\delta}\right) - C_5 \qquad \stackrel{def}{=} v_i^2$$

In particular, in order to satisfy the condition $v_i^1 + v_i^2 > 0$, it suffices to choose $1/\delta \in \mathcal{O}\big(\max(1/C_1, \max_{2\leq j\leq 5} C_j)^{\mathcal{O}(1)}\big)$. Since the constants C_j are exponential in $|\mathcal{M}^*|$, and by the conditions on the other constants above, the value Z_g, and hence the overall bound $b = Z_g + 1$, can be chosen such that $b \in \mathcal{O}\big(\exp\big(|\mathcal{M}^*|^{\mathcal{O}(1)}\big)\big)$.

Now we can prove the first item of our main result.

Proof of Theorem 1 (Item 1). Towards a contradiction, we assume that there exists a state $s^\dagger$ such that there is no finite-memory almost surely winning strategy from $s^\dagger$ for $\mathtt{O}(i_{s^\dagger})$ in the MDP $\mathcal{M}$.

First we consider the MDP $\mathcal{M}^*$. The finite-memory strategy $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}[(i_{s^\dagger},1)]$ from $s^\dagger$ is energy-safe by construction and satisfies $\mathtt{EN}_1(i_{s^\dagger})$ surely. Now consider the finite Markov chain induced by fixing this finite-memory strategy in $\mathcal{M}^*$. By Lemma 10 and Lemma 11, for a sufficiently large (exponential) b it yields a strictly positive expected mean payoff $v_i^1 + v_i^2 > 0$ in every dimension $i \in [2,d]$ in every BSCC of this Markov chain. Since the Markov chain is finite, this implies that the mean payoff in every dimension $i \in [2,d]$ is strictly positive almost

surely. Hence, $\mathcal{P}^{\mathcal{M}^*}_{\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}[(i_{s^\dagger},1)],s^\dagger}\big(\mathtt{MP}_{[2,d]}(>0)\big) = 1$ and thus $\mathcal{P}^{\mathcal{M}^*}_{\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}[(i_{s^\dagger},1)],s^\dagger}(\mathtt{O}(i_{s^\dagger})) = 1$. So there exists an almost surely winning finite-memory strategy from $s^\dagger$ for $\mathtt{O}(i_{s^\dagger})$ in $\mathcal{M}^*$. However, Lemma 3 then implies that there also exists an almost surely winning finite-memory strategy from $s^\dagger$ for $\mathtt{O}(i_{s^\dagger})$ in $\mathcal{M}$. Contradiction. ◀

▶ Remark 12. If $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}$ satisfies $\mathtt{O}(i_s)$ almost surely from some state s then it also satisfies the stronger objective $\mathtt{O}(i_s) \cap \mathtt{Infix}(b)$ almost surely. Consider a winning run induced by $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}$. While the true energy might sometimes be higher than b, the energy remembered by $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}$ is always $\leq b$. Even with this conservative under-approximation of the energy, the run still satisfies the energy objective. Therefore, in any winning run induced by $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}$, the energy can never *decrease* by more than b. Thus, also $\mathtt{Infix}(b)$ is satisfied almost surely.

5 Proof of Item 2

Given some state s, let $\sigma = (\mathsf{M}, \mathsf{m}_0, \mathsf{upd}, \mathsf{nxt})$ be a finite-memory strategy that is almost surely winning for $\mathtt{O}(i_s)$ (which exists by Item 1). We show there exists an almost surely winning strategy σ' for $\mathtt{O}(i_s)$ such that the energy fluctuations are bounded by some constant which is exponential in $|\mathcal{M}|$.

First, inside any BSCC B of $\mathcal{M}^\sigma$, we construct an almost surely winning strategy σ_B and upper bound the minimal safe energy levels and energy fluctuation while following σ_B. Using this, we upper bound the energy fluctuations in paths before reaching a BSCC. We use the fact that the set of states and transitions that occur in any BSCC of a Markov chain induced by fixing some finite-memory strategy in an MDP is an end component of this MDP ([19, Theorem 3.2]).

▶ **Lemma 13.** *Let B be a BSCC of $\mathcal{M}^\sigma$ and let $\mathcal{M}(B)$ be the corresponding end component in $\mathcal{M}$ with states S_B and transitions E_B. Then there is a strategy σ_B, a bound $b_B \in \mathcal{O}\big(\exp\big(|\mathcal{M}(B)|^{\mathcal{O}(1)}\big)\big)$ such that for any state $q \in S_B$, there is a minimal safe energy level $j_q \overset{def}{=} i_q^{\mathcal{M}(B)} \leq 3 \cdot |S_B| \cdot R$ such that $\mathcal{P}^{\mathcal{M}(B)}_{\sigma_B,q}(\mathtt{O}(j_q) \cap \mathtt{Infix}(b_B)) = 1$.*

Proof Sketch. (Full proof in [18, Appendix C].). The idea is that for $\mathcal{M}(B)$ there are two cases. In the first case it behaves similar to $\mathcal{M}^*$ from Section 4, in the sense that it is possible to win `Gain` and `Bailout` almost surely, and thus Energy-MeanPayoff can be won almost surely by switching between the two strategies for `Gain` and `Bailout` like in the strategy $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g}}$. Then one can invoke Lemma 11 and Remark 12 on $\mathcal{M}(B)$ to get an exponential bound b_B such that $\mathcal{P}^{\mathcal{M}(B)}_{\sigma_B,q}(\mathtt{O}(j_q) \cap \mathtt{Infix}(b_B)) = 1$.

If the first case does not hold then $\mathcal{M}(B)$ is very restrictive, and one can show that the energy level fluctuations are bounded by a constant in $\mathcal{O}(|S_B| \cdot R)$. ◀

Since the minimal safe energy levels inside these end components are not too large, one can then bound the energy fluctuations in paths before they reach any such end component $\mathcal{M}(B)$. The following lemma is shown in [18, Appendix C].

▶ **Lemma 14.** *Let T denote the union of all S_B of every BSCC B of $\mathcal{M}^\sigma$, as in Lemma 13. Then one can almost surely reach any state in T with the corresponding minimal safe energy level with energy fluctuations of at most $5 \cdot |S| \cdot R$.*

Proof of Theorem 1 (Item 2). By Lemmas 13 and 14, for each state s, one can choose a strategy σ and some constant $b \in \mathcal{O}\big(\exp\big(|\mathcal{M}|^{\mathcal{O}(1)}\big)\big)$ such that $\mathcal{P}^{\mathcal{M}}_{\sigma,s}(\mathtt{O}(i_s) \cap \mathtt{Infix}(b)) = 1$. This means if one encodes the energy levels between $[0, b]$ into the state space by discarding

any excess energy above b and redirecting all the transitions which result in a negative energy to a losing sink (for $\mathtt{MP}_{[2,d]}(>0)$) and constructs this larger MDP $\mathcal{M}[0,b]$, then there is a strategy σ' such that $\mathcal{P}^{\mathcal{M}[0,b]}_{\sigma',(s,k)}\big(\mathtt{MP}_{[2,d]}(>0)\big) = 1$ for every $k \in [i_s, b]$. Then, by Lemma 6, there also exists a memoryless (MR) strategy σ^* in $\mathcal{M}[0,b]$ which is almost surely winning $\mathtt{MP}_{[2,d]}(>0)$ from (s,k).

We can carry the memoryless strategy σ^* in $\mathcal{M}[0,b]$ back to $\mathcal{M}$ as a finite-memory strategy $\sigma^*_{\mathcal{M}}$ with memory $[0,b]$. It stores the encoded under-approximated energy level from $\mathcal{M}[0,b]$ in its finite memory instead. Thus, $\sigma^*_{\mathcal{M}}$ is a finite-memory strategy from s that satisfies $\mathtt{0}(i_s)$ almost surely, and the size of its memory is bounded by $b \in \mathcal{O}\big(\exp\big(|\mathcal{M}|^{\mathcal{O}(1)}\big)\big)$.

The strategy $\sigma^*_{\mathcal{M}}$ uses randomization, because σ^* from Lemma 6 is MR. However, the MR strategy σ^* for the mean payoff objective could be replaced by a deterministic strategy with an exponential number of memory modes. Hence, the overall number of memory modes in the obtained deterministic version of $\sigma^*_{\mathcal{M}}$ is still only exponential. ◀

6 The Lower Bound (Proof of Item 3)

In the previous sections we have shown that finite memory suffices for almost surely winning strategies for the Energy-MeanPayoff objective. However, the required memory depends on the given MDP. We show that no fixed finite amount of memory is sufficient for all MDPs. In fact, the required memory is exponential in the transition probabilities even for an otherwise fixed 5-state MDP with just one controlled state, $R = 1$ and $d = 2$.

▶ **Definition 15.** *Let $1 > \delta > 0$ and $\mathcal{M}_\delta = (S, S_\Box, S_\bigcirc, E, P, \boldsymbol{r})$ be an MDP with 2-dimensional rewards. It has just one controlled state s with transitions $s \to s_l$ and $s \to s_r$. From s_l there are two transitions $e_1 = (s_l \to s_l^1)$ and $e_2 = (s_l \to s_l^2)$. Let $\mathcal{P}(e_1) = (1+\delta)/2$ and $\mathcal{P}(e_2) = (1-\delta)/2$ and $\boldsymbol{r}(e_1) = (+1,+1)$ and $\boldsymbol{r}(e_2) = (-1,-1)$. s_l^1 and s_l^2 are random states which each have just one transition back to s with probability 1 and reward $\mathbf{0}$. From s_r there is only one transition e_3 back to s with probability 1 and $\boldsymbol{r}(e_3) = (+1,-1)$.*

The following lemma directly implies the exponential lower bound on the number of memory modes in Theorem 1(Item 3).

▶ **Lemma 16.** *Consider the Energy-MeanPayoff objective. For every finite bound $m \in \mathbb{N}$ on the number of memory modes there exists a $\delta \stackrel{def}{=} 1/(6m) > 0$ such that the finite MDP $\mathcal{M}_\delta = (S, S_\Box, S_\bigcirc, E, P, \boldsymbol{r})$ from Definition 15 satisfies the following properties.*

1. *$\exists \sigma'\ \mathcal{P}^{\mathcal{M}_\delta}_{\sigma',s}(\mathtt{EN}_1(0) \cap \mathtt{MP}_{\mathbf{2}}(>0)) = 1$, i.e., it is possible to win almost surely from s in $\mathcal{M}_\delta$, even with initial energy 0.*
2. *For every finite-memory strategy σ with $\leq m$ memory modes we have $\mathcal{P}^{\mathcal{M}_\delta}_{\sigma,s}(\mathtt{EN}_1(k) \cap \mathtt{MP}_{\mathbf{2}}(>0)) = 0$ for every $k \in \mathbb{N}$, i.e., σ attains nothing in $\mathcal{M}_\delta$, regardless of the initial energy k.*
3. *For $\mathcal{M}_\delta$ we have $|S| = 5$, $d = 1$ and $R = 1$. The number of memory modes required for an almost-surely winning strategy in $\mathcal{M}_\delta$ is exponential in $|P|$ (and in $|\mathcal{M}_\delta|$).*

Proof. Towards item 1, consider a strategy σ' that plays as follows. It keeps a counter that records the current energy, which is initially 0. Whenever the current energy is 0, it plays $s \to s_r$, otherwise it plays $s \to s_l$. Thus, σ' satisfies $\mathtt{EN}_1(0)$ surely from s. Since $\delta > 0$ it follows from the classic Gambler's ruin problem (with strictly positive expected gain, here in the first reward dimension) that σ' plays $s \to s_r$ only finitely often, except in a null set of the runs. Therefore, the expected mean payoff (in the second dimension) under σ' is $(1+\delta)/2 - (1-\delta)/2 = \delta > 0$. Hence, $\mathcal{P}^{\mathcal{M}_\delta}_{\sigma',s}(\mathtt{MP}_{\mathbf{2}}(>0)) = 1$. Since the energy objective is satisfied surely, we obtain $\mathcal{P}^{\mathcal{M}_\delta}_{\sigma',s}(\mathtt{EN}_1(0) \cap \mathtt{MP}_{\mathbf{2}}(>0)) = 1$.

Towards item 2, let $\delta \stackrel{\text{def}}{=} 1/(6m) > 0$ and let σ be a finite-memory strategy with $\leq m$ memory modes. Consider the finite-state Markov chain $\mathcal{C}$ that is induced by playing σ from s in $\mathcal{M}_\delta$. This Markov chain has $\leq 5m$ states, since $\mathcal{M}$ has 5 states and σ has $\leq m$ memory modes. Let B be any BSCC of $\mathcal{C}$ that is reachable from s and the initial memory mode of σ. In particular, $|B| \leq 5m$. In B there must not exist any loop that does not contain s_r, because otherwise the energy objective cannot be satisfied almost surely. Thus, every path in B of length $\geq 5m$ must contain s_r (and hence a reward $(+1, -1)$) at least once. Therefore, the expected mean payoff in B (in the second reward dimension) is $\leq 5m\delta - 1 = -1/6 < 0$. Since this holds in every reachable BSCC, we obtain $\mathcal{P}^{\mathcal{M}_\delta}_{\sigma,s}(\mathtt{MP}_{\mathbf{2}}(> 0)) = 0$ and thus $\mathcal{P}^{\mathcal{M}_\delta}_{\sigma,s}(\mathtt{EN}_1(k) \cap \mathtt{MP}_{\mathbf{2}}(> 0)) = 0$.

Towards item 3, the size of $\mathcal{M}_\delta$ follows from Definition 15. By items 1 and 2, the required number of memory modes m for an almost-surely winning strategy satisfies $m > 1/(6\delta)$. Since $|P| = \Theta(\log(1/\delta))$ and $|\mathcal{M}_\delta| = \Theta(|P|)$, we obtain $m = \Omega(\exp(|P|))$ and $m = \Omega(\exp(|\mathcal{M}_\delta|))$. ◀

The exponential lower bound on the required memory does not require probabilities encoded in binary like in Lemma 16. One can construct an equivalent example with polynomially many states where all transition probabilities are 1/2. This is because one can encode exponentially small probabilities 2^{-k} with a chain of k extra states and transition probabilities 1/2.

7 Computational Complexity

We have shown that the existence of an almost surely winning strategy for the Energy-MeanPayoff objective for a given state and initial energy level in an MDP implies the existence of a deterministic such strategy with exponentially many memory modes (unlike for Energy-Parity which requires infinite memory in general [25]).

A related problem is the decidability of the question whether a given state in an MDP and a given initial energy level admit an almost surely winning strategy for Energy-MeanPayoff. This problem is decidable in *pseudo-polynomial* time, using an algorithm very similar to the one for Energy-Parity presented in [25]. I.e., the time is polynomial, provided that the bound R on the rewards is given in unary. Transition probabilities in the MDP can still be represented in binary. The crucial point is that it suffices to witness the mere *existence* of an almost surely winning strategy, regardless of its memory. Basically, it suffices that the algorithm proves that the infinite-memory strategy $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g}}$ wins almost surely (plus a small extra argument about a corner case where the energy fluctuates only in a bounded region). The algorithm does not need to compute the bound b or to explicitly construct the finite-memory strategy $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g},\mathtt{b}}$.

▶ **Proposition 17.** *Let $\mathcal{M} = (S, S_\Box, S_\bigcirc, E, P, \boldsymbol{r})$ be an MDP with d-dimensional rewards on the edges $\boldsymbol{r} : E \to [-R, R]^d$. For any state s and $k \in \mathbb{N}$, the existence of an almost surely winning strategy from s for the multidimensional Energy-MeanPayoff objective $\mathtt{EN}_1(k) \cap \mathtt{MP}_{[\mathbf{2},\boldsymbol{d}]}(> \mathbf{0})$ is decidable in pseudo-polynomial time (i.e., polynomial for R in unary).*

Proof. The proof is similar to the one for Energy-Parity presented in [25]. We outline the differences below. First, in the corner case where it is impossible to pump the energy up arbitrarily high almost surely from some state q, the only possible way to win Energy-MeanPayoff (resp. Energy-Parity) almost surely (if at all) is by using a non-null set of runs where the energy only ever fluctuates in a bounded region. In that case, the size of the energy fluctuations in these runs can safely be restricted to a region that is polynomial in

$|S| \cdot R$, and thus pseudo-polynomial in $|\mathcal{M}|$ [25]. It thus suffices to win multi-dimensional $\mathtt{MP}_{[2,d]}(> 0)$ almost surely in a derived MDP $\mathcal{M}'$ where the bounded energy is encoded into the states. Deciding this requires time polynomial in $|\mathcal{M}'|$ [12, 22] and thus pseudo-polynomial in $|\mathcal{M}|$. The winning situations of the corner case can then be encoded into $\mathcal{M}$, yielding a derived MDP $\mathcal{M}'$ of pseudo-polynomial size, where Energy-MeanPayoff can be won almost surely if and only if it can be won almost surely by a combination of `Gain` and `Bailout` strategies, i.e., by strategy $\sigma^*_{\mathtt{alt},\mathtt{Z_b},\mathtt{Z_g}}$. Therefore, it suffices to compute the states (and minimal initial energy levels k) where `Gain` and $\mathtt{Bailout}(k)$ can be won almost surely. The objective $\mathtt{Bailout}(k) \stackrel{\text{def}}{=} \mathtt{EN}_1(k) \cap \mathtt{MP}_1(> 0)$ is exactly the same as the `Bailout` objective analysed in [25], and winning it almost surely is decidable in pseudo-polynomial time. Our objective $\mathtt{Gain} \stackrel{\text{def}}{=} \mathtt{MP}_{[1,d]}(> 0)$ differs from the `Gain` objective considered in [25] (which was $\mathtt{MP}_1(> 0) \cap \mathtt{Parity}$), but winning it almost surely is still decidable in polynomial time [12, 22] by solving a linear program. So overall the algorithm runs in pseudo-polynomial time. ◀

References

1 Pieter Abbeel and Andrew Y. Ng. Learning first-order Markov models for control. In *Advances in Neural Information Processing Systems 17*, pages 1–8. MIT Press, 2004.

2 Galit Ashkenazi-Golan, János Flesch, Arkadi Predtetchinski, and Eilon Solan. Reachability and safety objectives in Markov decision processes on long but finite horizons. *Journal of Optimization Theory and Applications*, 185:945–965, 2020.

3 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

4 Patrick Billingsley. *Probability and measure*. John Wiley & Sons, 2008.

5 Vincent D. Blondel and John N. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36(9):1249–1274, 2000.

6 T. Brázdil, A. Kučera, and P. Novotný. Optimizing the expected mean payoff in energy Markov decision processes. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 9938 of *LNCS*, pages 32–49, 2016.

7 Tomáš Brázdil, Václav Brožek, Krishnendu Chatterjee, Vojtěch Forejt, and Antonín Kučera. Markov decision processes with multiple long-run average objectives. *Logical Methods in Computer Science*, 10, 2014.

8 Véronique Bruyère, Quentin Hautem, Mickael Randour, and Jean-François Raskin. Energy Mean-Payoff Games. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory (CONCUR 2019)*, volume 140 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:17, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CONCUR.2019.21.

9 Nicole Bäuerle and Ulrich Rieder. *Markov Decision Processes with Applications to Finance*. Springer-Verlag Berlin Heidelberg, 2011.

10 Arindam Chakrabarti, Luca De Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Resource interfaces. In *International Workshop on Embedded Software*, pages 117–133, 2003.

11 K. Chatterjee and T. Henzinger. A survey of stochastic ω-regular games. *Journal of Computer and System Sciences*, 78(2):394–413, 2012.

12 Krishnendu Chatterjee and Laurent Doyen. Energy and mean-payoff parity Markov decision processes. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 6907, pages 206–218, 2011.

13 Krishnendu Chatterjee, Thomas A. Henzinger, and Marcin Jurdziński. Mean-payoff parity games. In *Logic in Computer Science (LICS)*, pages 178–187, 2005.

14 Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018. doi:10.1007/978-3-319-10575-8.

15 E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, December 1999.

16 Lorenzo Clemente and Jean-Francois Raskin. Multidimensional beyond worst-case and almost-sure problems for mean-payoff objectives. In *Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 257–268, 2015.

17 Mohan Dantam and Richard Mayr. Approximating the value of energy-parity objectives in simple stochastic games. In *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*, pages 38:1–38:15, 2023. doi:10.4230/LIPIcs.MFCS.2023.38.

18 Mohan Dantam and Richard Mayr. Finite-memory Strategies for Almost-sure Energy-MeanPayoff Objectives in MDPs, 2024. arXiv:2404.14522.

19 Luca De Alfaro. *Formal verification of probabilistic systems*. PhD thesis, Stanford University, 1997.

20 János Flesch, Arkadi Predtetchinski, and William Sudderth. Simplifying optimal strategies in limsup and liminf stochastic games. *Discrete Applied Mathematics*, 251:40–56, 2018.

21 Dean Gillette. Stochastic games with zero stop probabilities. *Contributions to the Theory of Games*, 3:179–187, 1957.

22 Hugo Gimbert, Youssouf Oualhadj, and Soumya Paul. Computing optimal strategies for Markov decision processes with parity and positive-average conditions. Working paper or preprint, 2011.

23 T.P. Hill and V.C. Pestien. The existence of good Markov strategies for decision processes with general payoffs. *Stoch. Processes and Appl.*, 24:61–76, 1987.

24 M. Jurdziński. Deciding the winner in parity games is in UP ∩ co-UP. *Information Processing Letters*, 68:119–124, 1998.

25 Richard Mayr, Sven Schewe, Patrick Totzke, and Dominik Wojtczak. MDPs with Energy-Parity Objectives. In *Logic in Computer Science (LICS)*. IEEE, 2017.

26 Richard Mayr, Sven Schewe, Patrick Totzke, and Dominik Wojtczak. Simple stochastic games with almost-sure energy-parity objectives are in NP and coNP. In *Proc. of Fossacs*, volume 12650 of *LNCS*, 2021. Extended version on arXiv. arXiv:2101.06989.

27 A. Puri. *Theory of hybrid systems and discrete event structures*. PhD thesis, University of California, Berkeley, 1995.

28 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

29 Manfred Schäl. Markov decision processes in finance and dynamic options. In *Handbook of Markov Decision Processes*, pages 461–487. Springer, 2002.

30 Olivier Sigaud and Olivier Buffet. *Markov Decision Processes in Artificial Intelligence*. John Wiley & Sons, 2013.

31 William D. Sudderth. Optimal Markov strategies. *Decisions in Economics and Finance*, 43:43–54, 2020.

32 R.S. Sutton and A.G Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, 2018.

Functional Closure Properties of Finite $\mathbb{N}$-Weighted Automata

Julian Dörfler
Saarland Informatics Campus (SIC), Saarbrücken Graduate School of Computer Science, Saarland University, Germany

Christian Ikenmeyer
University of Warwick, Coventry, UK

Abstract

We determine all functional closure properties of finite $\mathbb{N}$-weighted automata, even all multivariate ones, and in particular all multivariate polynomials. We also determine all univariate closure properties in the promise setting, and all multivariate closure properties under certain assumptions on the promise, in particular we determine all multivariate closure properties where the output vector lies on a monotone algebraic graph variety.

2012 ACM Subject Classification Theory of computation → Automata extensions

Keywords and phrases Finite automata, weighted automata, counting, closure properties, algebraic varieties

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.134

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/2404.14245

Funding *Christian Ikenmeyer*: EPSRC EP/W014882/2

1 Finite $\mathbb{N}$-weighted automata and functional closure properties

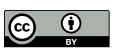
Let Σ be a finite set, for example $\Sigma = \{0, 1\}$. A finite $\mathbb{N}$-weighted automaton with all weights 1 is a nondeterministic finite automaton that on input $w \in \Sigma^\star$ outputs the number of accepting computation paths on input w, instead of just outputting whether or not an accepting computation path exists, see Def. 2.1 for the formal definition [1]. While every nondeterministic finite automaton determines a subset of $\Sigma^\star$, a finite $\mathbb{N}$-weighted automaton computes a function $\Sigma^\star \to \mathbb{N}$. A function $f : \Sigma^\star \to \mathbb{N}$ can be presented as the series $\sum_{w\in\Sigma^\star} f(w)w$, and the set of series is denoted by $\mathbb{N}\langle\langle\Sigma^\star\rangle\rangle$ in the automata literature, see e.g. [9][2]. The natural way of adding two functions $\Sigma^\star \to \mathbb{N}$ and adding two series in $\mathbb{N}\langle\langle\Sigma^\star\rangle\rangle$ coincides, but in both presentations we have a natural way of taking the product, and those do not coincide:

1. Pointwise product of functions $\Sigma^\star \to \mathbb{N}$. This is called the Hadamard product.
2. Convolution of series, called the Cauchy product.

A series f is called *recognizable* if there is a finite $\mathbb{N}$-weighted automaton that computes f. The set of recognizable series is denoted by $\mathbb{N}^{\mathrm{rec}}\langle\langle\Sigma^\star\rangle\rangle$ in [9], but we denote it by #FA, to emphasise that we undertake a study similar to #P in [11, Thm 3.13], [4, Thm 6], and recently [12], but instead of polynomial-time Turing machines we study finite automata.

[1] We use the equality of the number of accepting paths of an NFA and the output of the corresponding $\mathbb{N}$-weighted automaton, see [9, Exa. 2.2].

[2] A series with finite support is called a *polynomial*, but we will not be concerned with the support of series in this paper. Instead, we use the term *polynomial* as it is used in commutative algebra, and we mean multivariate polynomials with rational coefficients.



The Kleene-Schützenberger theorem states that #FA is the smallest set that contains all support 1 series and is closed under sums, Cauchy products, and Kleene-iterations (whenever well-defined, a Kleene-iteration is the sum of all Cauchy powers), see [9, §4], but we will not need this insight.

In this paper we study the functional closure properties[3] of #FA. A function $\varphi : \mathbb{N}^m \to \mathbb{N}$ is called a *functional closure property* of #FA if for all $f_1 \in \#\mathsf{FA}$, $f_2 \in \#\mathsf{FA}$, ..., $f_m \in \#\mathsf{FA}$ we have that $\varphi(f_1, \ldots, f_m) \in \#\mathsf{FA}$. By $\varphi(f_1, \ldots, f_m)$ we mean the function that on input $w \in \Sigma^\star$ outputs $\varphi(f_1(w), \ldots, f_m(w))$.

Classically, one of the simplest functional closure properties of #FA is $\varphi : \mathbb{N}^2 \to \mathbb{N}$, $\varphi(f_1, f_2) = f_1 + f_2$. This is a functional closure property of #FA, because given $f_1 \in \#\mathsf{FA}$ and $f_2 \in \#\mathsf{FA}$, we can show $\varphi(f_1, f_2) = f_1 + f_2 \in \#\mathsf{FA}$ by an easy construction: The new NFA consists of a copy of the NFA for f_1 and a copy of the NFA for f_2, and makes an initial nondeterministic choice as to which NFA to run, see Lemma 3.1 for the details.

Another classical simple functional closure property of #FA is $\varphi : \mathbb{N}^2 \to \mathbb{N}$, $\varphi(f_1, f_2) = f_1 \cdot f_2$. This corresponds to the Hadamard product. This is a functional closure property of #FA, because given $f_1 \in \#\mathsf{FA}$ and $f_2 \in \#\mathsf{FA}$, we can show $\varphi(f_1, f_2) = f_1 \cdot f_2 \in \#\mathsf{FA}$ by the following construction: The new NFA consists of the product NFA of the NFAs for f_1 and f_2, and the accepting states correspond to pairs of accepting states, see Lemma 3.2 for the details. This product construction corresponds to the Hadamard product.

The Cauchy product is also a product on the set #FA, but we explain now that the Cauchy product is not "functional", and hence it is out of scope for this type of studies. If $\varphi : \mathbb{N}^m \to \mathbb{N}$ is a functional closure property of #FA, then we can study the corresponding map $\widetilde{\varphi} : \underbrace{\#\mathsf{FA} \times \#\mathsf{FA} \times \cdots \times \#\mathsf{FA}}_{m \text{ times}} \to \#\mathsf{FA}$. Observe that if φ is a functional closure property of #FA, then by definition we have that for all pairs $(w, w') \in \Sigma^\star \times \Sigma^\star$:

if $(f_1(w), \ldots, f_m(w)) = (f_1(w'), \ldots, f_m(w'))$, then $\widetilde{\varphi}(f_1, \ldots, f_m)(w) = \widetilde{\varphi}(f_1, \ldots, f_m)(w')$.

Let $\zeta : \#\mathsf{FA} \to \#\mathsf{FA}$ denote the Cauchy square. We use the observation above to show that ζ is not equal to $\widetilde{\varphi}$ for any $\varphi : \mathbb{N} \to \mathbb{N}$. Let $m = 1$ and $f(w) = 1$ if $w = 1$, $f(w) = 0$ otherwise. Clearly, $f \in \#\mathsf{FA}$. Then $\zeta(f)(11) = 1$, and $\zeta(f)(w) = 0$ for all $w \neq 11$. In particular $\zeta(f)(0) \neq \zeta(f)(11)$, even though $f(0) = f(11)$. Hence, $\zeta \neq \widetilde{\varphi}$ for all $\varphi : \mathbb{N} \to \mathbb{N}$.

Numerous functional closure properties of #FA exist, for example the safe decrementation $\max\{0, f_1 - 1\}$, and the binomial coefficient $\binom{f_1}{2}$. But not all non-negative functions are functional closure properties of #FA, for example $(f_1 - f_2)^2$ is not, which can be shown using the Pumping Lemma. In this paper, we determine *all* functional closure properties of #FA, see §1.2 for the detailed statement.

1.1 Motivation

Functional closure properties can be studied for many different counting machine models (also for example with different types of oracle access) and different types of input sets. The first study of this type was done for nondeterministic polynomial-time Turing machines, i.e., the class #P, see [11], [4], and the recent [12]. Recall that the class #P is the class of functions $f : \Sigma^* \to \mathbb{N}$ for which a nondeterministic polynomial time Turing machine M exists such that for all $w \in \Sigma^*$ the number of accepting paths for the computation $M(w)$ is exactly $f(w)$. The papers mentioned above prove that the relativizing multivariate polynomial

[3] See [11, Sec. 1] for the naming *functional closure property*. A different reasonable name would be *pointwise closure property*.

closure properties are exactly those polynomials that have nonnegative integers in their expansion over the binomial basis, see [12]. A functional closure property $\varphi : \mathbb{N}^m \to \mathbb{N}$ of $\#\mathsf{P}$ is *relativizing* if φ is a closure property for all $\#\mathsf{P}^A$, where $A \subset \Sigma^*$ is some oracle. The hope is that for simpler models of computation no oracle access is required to determine the functional closure properties, and we show that this is true for $\#\mathsf{FA}$, see §1.2.

Functional closure properties can be used directly to construct combinatorial proofs of equalities and inequalities. For example, Fermat's little theorem states that p divides $a^p - a$. The quantity $\frac{1}{p}(a^p - a)$ has a combinatorial interpretation, which can be deduced from the fact that $\frac{1}{p}((f_1)^p - f_1)$ is a univariate functional closure property of $\#\mathsf{P}$, see [12, Prop. 7.3.1], which coincides with the original proof [18], see also [17, eq. (5)]). On the other hand, if a function is not a functional closure property, then this means in a very strong sense that there is no combinatorial interpretation for the quantity it describes. For example, the Hadamard inequality ([10, §2.13], [3, §2.11], [12, eq. (2)]) states that

$$\det \begin{pmatrix} a_{11} & \cdots & a_{1d} \\ \vdots & \ddots & \vdots \\ a_{d1} & \cdots & a_{dd} \end{pmatrix}^2 \leq \prod_{i=1}^{d} (a_{i1}^2 + \cdots + a_{id}^2).$$

One could try to prove this by finding a combinatorial interpretation of the difference $\mathcal{H} \geq 0$ of the right-hand side and the left-hand side, but even for $d = 3$ we have that

$$\varphi(f_1, \ldots, f_9) \;=\; (f_1^2 + f_2^2 + f_3^3) \cdot (f_4^2 + f_5^2 + f_6^3) \cdot (f_7^2 + f_8^2 + f_9^3) - \det \begin{pmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{pmatrix}$$

is not a 9-variate relativizing functional closure property of $\#\mathsf{P}$, see [12, §7.2]. In particular, if the function is not a closure property of $\#\mathsf{P}$, then there are instantiations $f_1, \ldots, f_9 \in \#\mathsf{P}$ such that $\varphi(f_1, \ldots, f_9)$ is not in $\#\mathsf{P}$, whereas a combinatorial interpretation of $\mathcal{H}$ should yield $\varphi(f_1, \ldots, f_9) \in \#\mathsf{P}$. This does not rule out a more indirect combinatorial proof for the inequality: For example, for proving combinatorially that $(a-1)^2 \geq 0$ one could try to interpret the quantity $(a-1)^2$ combinatorially, but $(f_1 - 1)^2$ is not a relativizing closure property of $\#\mathsf{P}$. However, $f_1 \cdot (f_1 - 1)^2 = 6\binom{f_1}{3} + 2\binom{f_1}{2}$ is a relativizing closure property of $\#\mathsf{P}$ (see [12, §2.4]). There is an obvious combinatorial interpretation of $6\binom{a}{3} + 2\binom{a}{2}$ as counting size 2 and 3 subsets with multiplicity 6 and 2, respectively. Hence this gives an indirect combinatorial proof for the inequality $(a-1)^2 \geq 0$ by providing a combinatorial interpretation for $a(a-1)^2$.

Some inequalities are only true if the inputs satisfy certain constraints. For example, the *Ahlswede Daykin inequality*, see [1], [2], [12, §1.2(3)]: If $a_0b_0 \geq c_0d_0$ and $a_0b_1 \geq c_0d_1$ and $a_1b_0 \geq c_0d_1$ and $a_1b_1 \geq c_1d_1$, then $(c_0+c_1)(d_0+d_1) \geq (a_0+a_1)(b_0+b_1)$. If all quantities are in $\#\mathsf{P}$, including the differences $c_0d_0 - a_0b_0$, can we conclude that $(c_0 + c_1)(d_0 + d_1) - (a_0 + a_1)(b_0 + b_1)$ is in $\#\mathsf{P}$? This is an example of a promise problem: We are given twelve $\#\mathsf{P}$ functions $a_0, a_1, b_0, b_1, c_0, c_1, d_0, d_1, h_1, h_2, h_3, h_4$ with the guarantee that $a_0b_0 + h_1 = c_0d_0$, $a_0b_1 + h_2 = c_0d_1$, $a_1b_0 + h_3 = c_0d_1$, $a_1b_1 + h_4 = c_1d_1$. In other words, the 12-dimensional output vector that we get for every $w \in \Sigma^*$ lies on a codimension 4 algebraic subvariety in $\mathbb{Q}^{12}$. Recall that an algebraic subvariety is defined as the simultaneous zero set of a set of polynomials. Since the 4 variables $h_1, \ldots, h_4$ are determined by the other 8, this variety is a so-called *graph* or *graph variety*. Numerous questions about combinatorial proofs for inequalities from different areas of mathematics can be phrased in the language of graph varieties, see [12]. The idea is to collect the equations for a set S (the variety) into what is called the *vanishing ideal* I, i.e., $I = I(S) = \{\varphi \in \mathbb{Q}[f_1, \ldots, f_m] \mid \forall (f_1, \ldots, f_m) \in S : \varphi(f_1, \ldots, f_m) = 0\}$; and define the *coordinate ring* $\mathbb{Q}[S]$ as the quotient ring $\mathbb{Q}[f_1, \ldots, f_m]/I(S)$, see [7]. An element in the quotient ring is a coset with respect to the vanishing ideal. If there exists a representative

φ' in a coset $\varphi + I$ that is a functional closure property of #P, then every function in $\varphi + I$ is a promise closure property of #P on the variety S. It is desirable to also have the opposite direction, but this only holds under some reasonable restrictions on S, in particular it holds for all graph varieties. This is used in [12, Prop. 2.5.1] to show that $c_0d_0 - a_0b_0$ is not a relativizing promise closure property of #P on this graph variety. We prove the same strong dichotomy for monotone graph varieties for #FA instead of #P, see Theorem 4.9.

The systematic study of combinatorial interpretations and combinatorial proofs via definitions from computational complexity theory is a very recent research direction [15, 16, 12, 17, 13, 5, 6]. The goal is to determine whether or not certain quantities admit a combinatorial description or not. Famous open questions of this type in algebraic combinatorics have been listed by Stanley in [22], for example his problems 9, 10, and 12. As many combinatorialists do, Stanley has phrased his questions in an informal way without mentioning counting classes.

The class #P is the correct class for some purposes, but for others it is too large. For example, the determinant of a skew-symmetric matrix with entries from $\{-1, 0, 1\}$ is always non-negative, but this quantity is trivially in #P, because the determinant can be computed in polynomial time. This gives no satisfying insight into whether or not this quantity has a combinatorial interpretation. Smaller counting classes are required (see also the discussion in [15, §1]), and we provide the first study of functional closure properties for the subclass #FA $\subset$ #P. Unlike the classification for #P, our results do not rely on oracle separations, i.e., our classification is entirely unconditional.

1.2 Our results

Let $n \operatorname{rem} p \in \{0, \ldots, p-1\}$ denote the smallest nonnegative r such that $n \equiv_p r$. A function $\varphi : \mathbb{N} \to \mathbb{N}$ is called *ultimately PORC* (Polynomial On Residue Classes[4]) if $\exists p, N \in \mathbb{N}$ and there exist polynomials $\varphi_0, \ldots, \varphi_{p-1} : \mathbb{N} \to \mathbb{Q}$ such that for every $n \geq N$ we have $\varphi(n) = \varphi_{n \operatorname{rem} p}(n)$ [5].

We first classify the univariate functional closure properties of #FA:

- **Theorem** (see Theorem 3.22). *A function $\varphi : \mathbb{N} \to \mathbb{N}$ is a functional closure property of #FA if and only if φ is an ultimately PORC function.*

More generally, we classify the multivariate functional closure properties of #FA:

- **Theorem** (see Theorem 3.23). *A function $\varphi : \mathbb{N}^m \to \mathbb{N}$ is a functional closure property of #FA if and only if φ can be written as a finite sum of finite products of univariate ultimately PORC functions.*

We analyze the special case of multivariate polynomials:

- **Theorem** (see Lemma 3.26). *A multivariate polynomial $\varphi : \mathbb{N}^m \to \mathbb{N}$ with rational coefficients is a functional closure property of #FA iff for every ψ that can be formed from φ by replacing any subset of variables – including the empty set – by constants from $\mathbb{N}$, then all dominating terms of ψ in the binomial basis have positive coefficients.*

---

[4] PORC functions are also known as quasipolynomials or pseudopolynomials, but we want to avoid those names for the potential confusion to quasipolynomial growth and pseudopolynomial running times.

[5] Note that each φ in this paper is defined on the natural numbers and maps to the natural numbers, which is a subtle restriction. For example, a univariate polynomial $\varphi : \mathbb{Q} \to \mathbb{Q}$ maps integers to integers if and only if its coefficients in the binomial basis are integers, see Section 2. However, non-negativity is not an algebraic property. Also note that for the case of φ being just a univariate polynomial, the corresponding linear recursive sequence can have negative entries in the matrix.

We lift this result to monotone graph varieties (and to more general sets, see Theorem 4.6), where we get exactly the desirable classification given by the vanishing ideal:

- **Theorem** (see Theorem 4.9). *Let S be a monotone graph variety and let $I = I(S)$ be its vanishing ideal. A multivariate polynomial $\varphi : S \to \mathbb{N}$ is a functional promise closure property of #FA with regard to S if and only if there exists $\psi \in I$ such that $\varphi + \psi$ is a multivariate functional closure property of #FA.*

2 Notation

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$. For a finite set Σ let $\Sigma^\star$ denote the set of all finite length sequences with elements from Σ. The vector space of multivariate polynomials $\mathbb{Q}[f_1, \ldots, f_m]$ in variables $f_1, \ldots, f_m$ has a basis given by products of binomial coefficients: $\left\{ \prod_{i=1}^m \binom{f_i}{c_i} \right\}_{c_1,\ldots,c_m}$, where each $c_i \in \mathbb{N}$. Here we used $\binom{x}{c} = \frac{1}{c!} x \cdot (x-1) \cdot \ldots \cdot (x-c+1)$ as a polynomial. This is called the *binomial basis*. A multivariate polynomial φ is called *integer valued* if $\varphi(\mathbb{Z}^m) \subseteq \mathbb{Z}$, which is equivalent to $\varphi(\mathbb{N}^m) \subseteq \mathbb{Z}$, and which is also equivalent to all coefficients in the binomial basis being integers, see for example [12, Prop. 4.2.1] for a short proof of this classical fact.

We now recall (see [9, Def. 2.1]) our main model of computation, the finite $\mathbb{N}$-weighted automaton, which we just call non-deterministic finite automaton (NFA) for brevity.

▶ **Definition 2.1.** *An NFA M is a tuple $(Q, \Sigma, \mathrm{wt}, \mathrm{in}, \mathrm{out})$ where the set of states Q and the alphabet Σ are finite sets and $\mathrm{wt} : Q \times \Sigma \times Q \to \mathbb{N}$ is the weighted transition function, $\mathrm{in} : Q \to \mathbb{N}$ are the weighted initial states and $\mathrm{out} : Q \to \mathbb{N}$ are the weighted accepting states*[6]*. A* computation *P for a word $w = w_1 \ldots w_n \in \Sigma^\star$ of length n is a sequence $q_0 q_1 \ldots q_n$ in Q^{n+1}. It has* multiplicity *or* weight[7] *$\mathbf{w}(P) = \mathrm{in}(q_0) \cdot \prod_{i=1}^n \mathrm{wt}(q_{i-1}, w_i, q_i) \cdot \mathrm{out}(q_n)$ and* partial weight *$\underline{\mathbf{w}}(P) = \mathrm{in}(q_0) \cdot \prod_{i=1}^n \mathrm{wt}(q_{i-1}, w_i, q_i)$. We say that M* computes *$f : \Sigma^\star \to \mathbb{N}$ where $f(w)$ is the sum of the weights over all computations of M on w. The class #FA is defined as the set of all functions $f : \Sigma^\star \to \mathbb{N}$ that are computed by NFAs.*

If needed to distinguish these for different automata, we use a corresponding subscript, for example the weights of computations in M_f would be denoted by $\mathbf{w}_f$, etc.

▶ **Definition 2.2** (Simple NFA). *We say an NFA $M = (Q, \Sigma, \mathrm{wt}, \mathrm{in}, \mathrm{out})$ is* simple *if $\mathrm{im\,wt}, \mathrm{im\,in}, \mathrm{im\,out} \subseteq \{0, 1\}$.*

The notion of a simple NFA also motivates our use of the term NFA opposed to $\mathbb{N}$-weighted automaton: We simply count the number of accepting paths of M on a word w. This is in line with #P counting the number of accepting paths on a polynomial time non-deterministic Turing machine.

▶ **Lemma 2.3** (Folklore). *For every NFA M there exists a simple NFA M' computing the same function.*

[6] Note that in definitions by other authors one can find simpler versions of NFAs, in particular unweighted initial and accepting states and unweighted edges while also restricting to a single initial state. We will see soon that working with unweighted NFAs is not a restriction, but additionally restricting the model to have a single initial state is strictly weaker, since this model could not compute any function f with $f(\varepsilon) > 1$. To obtain the same expressiveness one would have to additionally allow for ε-transitions, while disallowing cycles of ε-transitions to prevent infinite values for f.

[7] We will use both of these terms interchangeably. For a weighted automaton, calling this weight is more natural, while when looking at the underlying graph as a multigraph, multiplicity of paths and walks is more natural.

A proof of this simple fact can be found in the appendix of the full version, for the sake of completeness. We denote by $a \equiv_p b$ that $a \in \mathbb{N}$ and $b \in \mathbb{N}$ are congruent modulo $p \in \mathbb{N} \setminus \{0\}$. The indicator function $\mathbb{1}_{n=c} : \mathbb{N} \to \mathbb{N}$ is defined as $n \mapsto \begin{cases} 1 & \text{if } n = c \\ 0 & \text{otherwise} \end{cases}$ and analogously for different conditions. By abuse of notation, if we have a function $f : \Sigma^\star \to \mathbb{N}$ and an expression $\varphi : \mathbb{N} \to \mathbb{N}$ in n, we replace n by f in the expression to denote $\varphi \circ f$. For example we use $\mathbb{1}_{f=c}$ to denote the function $w \mapsto \mathbb{1}_{f(w)=c}$, similarly $\binom{f}{2}$ denotes the function $w \mapsto \binom{f(w)}{2}$. Furthermore we use the notation $[n]$ to denote the set $\{1, \ldots, n\}$ for any $n \in \mathbb{N}$.

3 Functional closure properties

3.1 Univariate functional closure properties

We say a function $\varphi : \mathbb{N} \to \mathbb{N}$ is a functional closure property of #FA if $\varphi(\#\mathsf{FA}) \subseteq \#\mathsf{FA}$, i.e. if for every function $f \in \#\mathsf{FA}$ the function $\varphi \circ f$ is also in #FA. Our goal in this section is to classify all functional closure properties of #FA. They will be precisely the ultimately PORC functions.

We call a function $\varphi : \mathbb{N} \to \mathbb{N}$ an *ultimately almost PORC function* if there is a *quasiperiod* p, an *offset* $N \in \mathbb{N}$ and *constituents* $\varphi_0, \ldots, \varphi_{p-1} : \mathbb{N} \to \mathbb{Q}$, where each φ_i is either a polynomial with rational coefficients or a function in $2^{\Theta(n)}$, and for every $n \geq N$ we have $\varphi(n) = \varphi_{n \operatorname{rem} p}(n)$. If all the constituents are polynomials, we call φ an *ultimately PORC function*. The smallest representative of each constituent and of the finite cases before the periodic behaviour is captured by the *shifted remainder* operator $n \operatorname{srem}_p N$, defined via

$$n \operatorname{srem}_p N = \begin{cases} n & \text{if } n < N \\ \min\{k \geq N \mid k \equiv_p n\} & \text{if } n \geq N \end{cases}$$

The first half of the section is dedicated to proving that every ultimately PORC function is a functional closure property of #FA, see Lemma 3.18. In order to prove this we show that #FA is closed under

▶ **Lemma 3.1** (Addition). *If $f, g \in \#\mathsf{FA}$, then $f + g \in \#\mathsf{FA}$.*

▶ **Lemma 3.2** (Multiplication). *If $f, g \in \#\mathsf{FA}$, then $f \cdot g \in \#\mathsf{FA}$.*

▶ **Lemma 3.9** (Subtraction of constants). *If $f \in \#\mathsf{FA}$, then $\forall c \in \mathbb{N} : \max(f - c, 0) \in \#\mathsf{FA}$.*

▶ **Lemma 3.10** (Clamping). *If $f \in \#\mathsf{FA}$, then $\min(f, c) \in \#\mathsf{FA}$ for any constant $c \in \mathbb{N}$.*

▶ **Lemma 3.11** (Comparison with constants). *If $f \in \#\mathsf{FA}$, then the functions $\mathbb{1}_{f=c}$, $\mathbb{1}_{f \leq c}$, $\mathbb{1}_{f \geq c}$ are in #FA for any constant $c \in \mathbb{N}$.*

▶ **Lemma 3.13** (Division by constants). *If $f \in \#\mathsf{FA}$, then $\forall c \in \mathbb{N} \setminus \{0\} : \lfloor f/c \rfloor \in \#\mathsf{FA}$.*

▶ **Lemma 3.14** (Modular arithmetic). *If $f \in \#\mathsf{FA}$, then the function $\mathbb{1}_{f \equiv_c d}$ is in #FA for any constants $c \in \mathbb{N} \setminus \{0\}$ and $d \in \mathbb{Z}_c$.*

▶ **Lemma 3.15** (Binomial coefficients). *If $f \in \#\mathsf{FA}$, then $\binom{f}{c} \in \#\mathsf{FA}$ for any constant $c \in \mathbb{N}$.*

While addition and multiplication are technically bivariate functional closure properties, we list them here already since they are abundantly used throughout the proofs of the univariate functional closure properties. Proofs of those two classical results can be found in [8, Ch. 4.1 and 4.2.2] and in the appendix of the full version, for the sake of completeness.

With the exception of binomial coefficients all the other closure properties need to be able to "remove" some of the possible computations. For example, consider decrementation, the special case of truncated subtraction by one, and consider some simple NFA M computing some strictly positive function f. We now want to construct an NFA M' that computes $f-1$, i.e. an NFA that has exactly one non-zero computation less than M (assuming computations of weights zero or one). For this we want a procedure to single out one non-zero computation of M to then change its weight to zero. For stronger models of computation – like polynomial time non-deterministic Turing machines – this approach seems hopeless. Already deciding the existence of one such computation is NP-hard. However for NFAs, deciding the existence of a non-zero computation can be decided by a deterministic finite automaton, namely the powerset automaton. Adjusting the powerset construction to filter out a single non-zero computation, namely the lexicographically minimal one can then be used to show that decrementation is a closure property of #FA.

Generalizing this approach to more general properties about the computations gives us the framework of stepwise computation properties:

▶ **Definition 3.3** (Stepwise computation property). *Let $M = (Q, \Sigma, \mathrm{wt}, \mathrm{in}, \mathrm{out})$ be an NFA. A* stepwise computation property prop *is defined as* $\mathrm{prop} = (S, \mathrm{init}, \mathrm{step}, \mathrm{cond})$ *where S is a finite set and* $\mathrm{init} : Q \to S$, $\mathrm{step} : Q \times \Sigma \times Q \times S \to S$ *and* $\mathrm{cond} : S \to \{0,1\}$ *are functions. For $w = w_1 \dots w_n \in \Sigma^\star$ and a computation $P = q_0 \dots q_n$ of M on w we define a* step sequence $s_0 := \mathrm{init}(q_0)$ *and* $s_i := \mathrm{step}(q_{i-1}, w_i, q_i, s_{i-1})$ *for $i \in [n]$. We also write* $\mathrm{prop}(w, P) := \mathrm{cond}(s_n)$ *to be the evaluation of the property.*

These stepwise computation properties now enable us, given a simple NFA, to construct NFAs computing both of the following:

▶ **Lemma 3.4.** *Let M_f be a simple NFA computing a function f and let* prop *be a stepwise computation property. Then there is an NFA M computing $g(w) = \sum_P \mathbf{w}_f(P) \cdot \mathrm{prop}(w, P)$, where the sum is over all computations P of M_f on w.*

▶ **Lemma 3.5.** *Let M_f be a simple NFA computing a function f and let* prop *be a stepwise computation property. Then there is an NFA M computing $g(w) = \sum_P \mathrm{prop}(w, P)$, where the sum is over all computations P of M_f on w.*

Proof sketch. For both of these lemmas, we construct a sort of product automaton of M_f and prop (represented by the set S), the details can be found in the appendix of the full version. ◀

In other words, stepwise computation properties allow us to either "disable" specific computations of M_f or they allow us to directly extract information about the computations of M_f. Note that the restriction on the finiteness of S is necessary, as the elements of S are hard-coded into the state space of the NFA in Lemmas 3.4 and 3.5. In particular, these lemmas do not hold for even countably infinite S, which can be seen with the example $S = \mathbb{N}$ when we define the stepwise computation property in such a way that $\mathrm{prop}(w, P) = 1$ iff $|w|$ is prime, leading to an NFA recognizing the language of all words of prime length, a well known contradiction.

Further note that these two constructions do not incur exponential blowups themselves, however for most of our applications the set S will be of exponential size in the number of states of M_f.

Returning to our decrementation example, to use Lemma 3.4 we want to construct a stepwise computation property prop with $\text{prop}(w, P) = 0$ iff P is the lexicographically smallest non-zero weight computation on w. For this we can set $S = \mathcal{P}(Q)$ to be the set of all subsets of states Q, denoting the set of states that currently are the endpoints of lexicographically smaller partial computations of non-zero weight than the partial computation P we are on. We need to store all such potential states, since some current partial non-zero weight computations might not be possible to be completed to a full non-zero weight computation. Initially this set contains all states that are smaller than the start state of P, the step function then checks which lexicographically smaller partial computations can be extended and whether any new partial computations that agreed with P up to this state can be lexicographically smaller than P. Finally the cond function then checks whether there are any such lexicographically smaller partial computations left that can be completed to a non-zero weight computation, i.e. that end on a state $q \in Q$ with $\text{out}(q) = 1$.

We want to generalize this idea to be able to generally create stepwise computation properties that argue about the number of non-zero computations, either in total or lexicographically smaller than a given computation. However doing this in general would require choosing the set S as the set of functions $Q \to \mathbb{N}$, which is infinite. As a result we embed the number of computations into finite semirings first to then extract the relevant information. For this purpose we only consider semirings with both additive and multiplicative identities. Homomorphisms h from a semiring $\mathcal{R}$ into another semiring $\mathcal{R}'$ need to fulfill $h(a + b) = h(a) + h(b)$, $h(a \cdot b) = h(a) \cdot h(b)$, $h(1_{\mathcal{R}}) = 1_{\mathcal{R}'}$, $h(0_{\mathcal{R}}) = 0_{\mathcal{R}'}$. Since every element of $\mathbb{N}$ is either 0 or can be formed by repeated addition of 1, any homomorphism from $\mathbb{N}$ into any other semiring is uniquely defined.

We can then use $\mathcal{R}$ to construct stepwise computation properties (the full proofs can be found in the appendix of the full version). Combined with Lemmas 3.4 and 3.5 these use similar ideas to [14].

▶ **Lemma 3.6.** *Let $N = (Q, \Sigma, \text{wt}, \text{in}, \text{out})$ be a simple NFA and let $\mathcal{R}$ be a finite semiring and let $\tau : \mathbb{N} \to \mathcal{R}$ be the unique homomorphism from $\mathbb{N}$ to $\mathcal{R}$. For any function $\pi : \mathcal{R} \to \{0, 1\}$ there is a stepwise computation property* prop *with* $\text{prop}(w, P) = \pi(\tau(\sum_{P'} \mathbf{w}(P')))$*, where the sum is over all computations P' of N on w, independent of P.*

Proof sketch. Construct $\text{prop} = (S, \text{init}, \text{step}, \text{cond})$ via: $S = Q \to \mathcal{R}$, $\text{init}(q) = r \mapsto \tau(\text{in}(r))$, $\text{step}(q, \sigma, q', s) = r \mapsto \sum_{r' \in Q} s(r') \cdot \tau(\text{wt}(r', \sigma, r))$, and $\text{cond}(s) = \pi\big(\sum_{r \in Q} s(r) \cdot \tau(\text{out}(r))\big)$. Let $s_0, \ldots, s_n$ be the step sequence of any computation P of w. Since τ is a homomorphism, we can pull out τ. Thus $s_i(q) = \tau(\sum_{\tilde{P}} \underline{\mathbf{w}}(\tilde{P}))$, where the sum is over all computations $\tilde{P}$ of $w_1, \ldots w_i$ ending in the state q. The condition cond then completes this to $\text{prop}(w, P) = \pi(\tau(\sum_{P'} \mathbf{w}(P')))$, where the sum is over all computations P' of N on w. ◀

▶ **Lemma 3.7.** *Let $M = (Q, \Sigma, \text{wt}, \text{in}, \text{out})$ be a simple NFA with some ordering $<$ of Q, let $\mathcal{R}$ be a finite semiring and let $\tau : \mathbb{N} \to \mathcal{R}$ be the unique homomorphism from $\mathbb{N}$ to $\mathcal{R}$. For any function $\pi : \mathcal{R} \to \{0, 1\}$ there is a stepwise computation property* prop *with* $\text{prop}(w, P) = \pi(\tau(\sum_{P'} \mathbf{w}(P')))$ *for all non-zero computations P of N on w, where the sum is over all computations P' of N on w that are lexicographically smaller than P.*

Proof sketch. Construct $\text{prop} = (S, \text{init}, \text{step}, \text{cond})$ via: $S = Q \to \mathcal{R}$, $\text{init}(q) = r \mapsto \tau(\mathbb{1}_{r<q} \cdot \text{in}(r))$, $\text{step}(q, \sigma, q', s) = r \mapsto \sum_{r' \in Q} s(r') \cdot \tau(\text{wt}(r', \sigma, r)) + \tau(\mathbb{1}_{r<q'} \cdot \text{wt}(q, \sigma, r))$, $\text{cond}(s) = \pi(\sum_{r \in Q} \tau(\text{out}(r)) \cdot s(r))$. Let $s_0, \ldots, s_n$ be the step sequence of any computation $P = q_0 \ldots q_n$ of w. Inductively we can show that $s_i(r) = \sum_{P'} \tau(\underline{\mathbf{w}}(P'))$ for all $i \in \{0, \ldots, n\}$ and $r \in Q$, where the sum is over all computations $P' = q'_0 \ldots q'_i$ of N on $w_1 \ldots w_i$ with

$q'_i = r$ that are lexicographically smaller than $q_0 \dots q_i$. Initially the only computations $P' = q'_0$ that are lexicographically smaller than the computation q_0 are the ones with $q'_0 < q_0$. For $i > 0$ for a computation $P' = q'_0 \dots q'_i$ to be lexicographically smaller than $q_0 \dots q_i$ there are two possibilities. Either $q'_0 \dots q'_{i-1}$ is already lexicographically smaller than $q_0 \dots q_{i-1}$ or $q'_0 \dots q'_{i-1} = q_0 \dots q_{i-1}$ and $q'_i < q_i$. In the second case the weight of P' is precisely $w(q_{i-1}, w_i, q'_i)$ since P is a computation of non-zero weight and thus weight exactly 1. Combining all of this, we can finish the proof of the claim with a similar argument to Lemma 3.6. ◀

Note that the previous lemma makes no statement about the value of $\text{prop}(w, P)$ for any computations P of weight zero. However, this is enough for our uses, since we only combine it with Lemma 3.4, i.e., $\text{prop}(w, P)$ gets weighted by $\mathbf{w}(P)$.

Most commonly, as is the case for decrementation, we want to be able to exactly distinguish the number of non-zero computations if it is less than k and otherwise be able to tell that the number is at least k. This is achieved by using the following capped semiring:

▶ **Definition 3.8** (Capped semiring). *For $k \in \mathbb{N}$ we call the semiring $\mathcal{R}_k = \{0, \dots, k\}$ with the operations $a +_{\mathcal{R}} b := \min(a + b, k)$ and $a \cdot_{\mathcal{R}} b := \min(a \cdot b, k)$ the* capped *semiring.*

We can now show that decrementation is a functional closure property by simply using Lemma 3.7 using the capped semiring $\mathcal{R}_1$ and $\pi(a) = a$ to construct our wanted stepwise computation property computing $\text{prop}(w, P) = 0$ iff P is the lexicographically smallest non-zero weight computation on w.

Most of the remaining closure properties are now proven by using the capped semiring of a specific size and choosing the function π accordingly, we will show this in detail for the example of subtraction, the remaining proofs can be found in the appendix of the full version.

▶ **Lemma 3.9** (Subtraction of constants). *If $f \in \#\mathsf{FA}$, then $\forall c \in \mathbb{N} : \max(f - c, 0) \in \#\mathsf{FA}$.*

Proof. Let $M_f = (Q_f, \Sigma, \text{wt}_f, \text{in}_f, \text{out}_f)$ be a simple NFA computing f with an arbitrary ordering $<$ on Q_f. Lemma 3.7 on the capped semiring $\mathcal{R}_c$ with $\pi(a) = \mathbb{1}_{a \geq c}$ for all $a \in \mathcal{R}$ constructs a stepwise computation property prop with

$$\text{prop}(w, P) = \begin{cases} 1 & \text{if the number of non-zero computations } P' \text{ on } w \text{ that are lex. smaller than } P \text{ is at least } c \\ 0 & \text{otherwise} \end{cases}$$

for all computations P on w of non-zero weight, i.e., $\text{prop}(w, P) = 0$ iff P is one of the c lexicographically smallest computations on w with non-zero weight. It follows that the NFA M constructed by Lemma 3.4 computes $g(w) = \max(f(w) - c, 0)$. ◀

If instead of rejecting the c lexicographically smallest computations, we accept only those computations, we compute the minimum of f and c.

▶ **Lemma 3.10** (Clamping). *If $f \in \#\mathsf{FA}$, then $\min(f, c) \in \#\mathsf{FA}$ for any constant $c \in \mathbb{N}$.*

By using the capped semiring $\mathcal{R}_{c+1}$ with $\pi_=(a) = \mathbb{1}_{a=c}$, $\pi_\leq(a) = \mathbb{1}_{a \leq c}$ and $\pi_\geq(a) = \mathbb{1}_{a \geq c}$, we can compute the indicator functions $\mathbb{1}_{f=c}$, $\mathbb{1}_{f \leq c}$ and $\mathbb{1}_{f \geq c}$ respectively.

▶ **Lemma 3.11** (Comparison with constants). *If $f \in \#\mathsf{FA}$, then the functions $\mathbb{1}_{f=c}$, $\mathbb{1}_{f \leq c}$, $\mathbb{1}_{f \geq c}$ are in $\#\mathsf{FA}$ for any constant $c \in \mathbb{N}$.*

The previous lemma in particular also implies the following:

▶ **Lemma 3.12.** *If $\varphi : \mathbb{N} \to \mathbb{N}$ is a functional closure property of #FA and $\psi : \mathbb{N} \to \mathbb{N}$ is an arbitrary function with $\varphi(n) = \psi(n)$ for all but finitely many $n \in \mathbb{N}$, then ψ is also a functional closure property of #FA.*

Proof. Let $f \in \#\mathsf{FA}$ be arbitrary. Further let $N \in \mathbb{N}$ be such that $\varphi(n) = \psi(n)$ for all $n \geq N$. Then $(\psi \circ f)(w) = \sum_{i=0}^{N-1} \mathbb{1}_{f(w)=i}\psi(i) + \mathbb{1}_{f(w)\geq N}\varphi(f(w))$ for all $w \in \Sigma^\star$. In particular $\psi \circ f \in \#\mathsf{FA}$ since the $\psi(i)$ are constants and $\varphi \circ f \in \#\mathsf{FA}$. ◀

Division and modular arithmetic however use a different semiring: they use the finite cyclic semiring $\mathbb{Z}_c = \{0, \ldots, c-1\}$ with $\pi(a) = \mathbb{1}_{a=c-1}$ and $\pi(a) = \mathbb{1}_{a=d}$ respectively.

▶ **Lemma 3.13** (Division by constants)**.** *If $f \in \#\mathsf{FA}$, then $\forall c \in \mathbb{N} \setminus \{0\} : \lfloor f/c \rfloor \in \#\mathsf{FA}$.*

▶ **Lemma 3.14** (Modular arithmetic)**.** *If $f \in \#\mathsf{FA}$, then the function $\mathbb{1}_{f \equiv_c d}$ is in #FA for any constants $c \in \mathbb{N} \setminus \{0\}$ and $d \in \mathbb{Z}_c$.*

The previous closure properties turn out to already be sufficient to generate all functional closure properties, so in particular they are sufficient to generate binomial coefficients by using subtraction of constants, multiplication and division by constants by using the definition of binomial coefficients as a polynomial: $\binom{x}{c} = \frac{1}{c!}x \cdot (x-1) \cdot \ldots \cdot (x-c+1)$. Nonetheless, we give an additional proof for binomial coefficients as a different interesting application of the stepwise computation property framework.

▶ **Lemma 3.15** (Binomial coefficients)**.** *If $f \in \#\mathsf{FA}$, then $\binom{f}{c} \in \#\mathsf{FA}$ for any constant $c \in \mathbb{N}$.*

Proof. Let $M_f = (Q_f, \Sigma, \mathrm{wt}_f, \mathrm{in}_f, \mathrm{out}_f)$ be a simple NFA computing f and let $c \in \mathbb{N}$. For $c < 2$ the statement of this lemma is trivially true, so assume $c \geq 2$.

We construct the c-fold product automaton $M_f^c = (Q_f^c, \Sigma, \mathrm{wt}_f^c, \mathrm{in}_f^c, \mathrm{out}_f^c)$ with

$$\begin{aligned}
\mathrm{wt}_f^c((q_1, \ldots, q_c), \sigma, (q_1', \ldots, q_c')) &= \textstyle\prod_{i=1}^c \mathrm{wt}_f(q_i, \sigma, q_i') \\
\mathrm{in}_f^c((q_1, \ldots, q_c)) &= \textstyle\prod_{i=1}^c \mathrm{in}_f(q_i) \\
\mathrm{out}_f^c((q_1, \ldots, q_c)) &= \textstyle\prod_{i=1}^c \mathrm{out}_f(q_i)
\end{aligned}$$

M_f^c is a simple NFA and every computation on M_f^c is the cartesian product of c computations on M_f. Our aim is to now construct a stepwise computation property $\mathrm{prop} = (S, \mathrm{init}, \mathrm{step}, \mathrm{cond})$ such that $\mathrm{prop}(w, P) = 1$ iff P is composed of c pairwise distinct computations[8] on N_f.

For this let S be the set of all equivalence relations on the set $[c]$. We define $\mathrm{init}((q_0, \ldots, q_c))$ to be the equivalence relation R_0 with $(a, b) \in R_0$ iff $q_a = q_b$. Additionally we define $\mathrm{cond}(R) = 1$ iff R is the equivalence relation where every element is only equivalent to itself, i.e. a computation gets accepted iff all its constituent computations are pairwise distinct. Finally we define $\mathrm{step}((q_1, \ldots, q_c), \sigma, (q_1', \ldots, q_c'), R)$ to be the equivalence relation R' defined via $(a, b) \in R'$ iff $(a, b) \in R$ and $q_a' = q_b'$. With a simple induction we can prove that for a computation $P = P_1 \times \ldots \times P_c$ and the step sequence $R_0, \ldots, R_n$ we have $(a, b) \in R_i$ iff the computations P_a and P_b are identical for the first i steps. It follows that the NFA M constructed by Lemma 3.4 computes $g(w) = \binom{f(w)}{c} \cdot c!$. Now, $\binom{f}{c} \in \#\mathsf{FA}$ by Lemma 3.13. ◀

[8] We could also require them to be sorted in lexicographical order by having S be the set of all total preorders, but since we can divide by $c!$ we are going with the easier exposition.

While the combination of the previous lemmas can be used to show that any polynomial written in the binomial basis with non-negative integer coefficients is a functional closure property of #FA, we can do better by considering a shifted binomial basis. For example, consider the polynomial $\varphi(x) = \frac{x^2}{2} - \frac{3x}{2} + 1$. This polynomial is non-negative for all $x \in \mathbb{N}$. Writing φ in the binomial basis we get $\varphi(x) = \binom{x}{2} - \binom{x}{1} + 1$. If however we allow the upper indices of the binomial basis to be shifted, we can write φ without the use of negative coefficients as $\varphi(x) = \binom{x-1}{2}$. While $x - 1$ itself is not a functional closure property of #FA the function $\max(x-1, 0)$ is a functional closure property of #FA and is different from $x - 1$ for only finitely many $x \in \mathbb{N}$. In the same way we see that $\varphi'(x) := \binom{\max(x-1,0)}{2}$ only differs from φ for finitely many $x \in \mathbb{N}$, namely $x = 0$. Using Lemma 3.12 to change those finitely many values, we see that φ is indeed a functional closure property of #FA.

Generalizing this idea we will show with the next two lemmas that this is possible for any φ with integer coefficients in the binomial basis, with a small restriction: We don't show that φ itself is a functional closure property of #FA, but rather that $x \mapsto \max(\varphi(x), 0)$ is one. Note that this restriction is the best we can hope for, since no computation in an NFA can ever have negative weight.

▶ **Lemma 3.16.** *Let $\varphi(x) = \sum_{i=0}^{r} a_i \cdot \binom{x}{i}$ with $a_i \in \mathbb{Z}$ and $a_r > 0$. Then there are $b_0, \ldots, b_r \in \mathbb{N}$ and $c_0, \ldots, c_r \in \mathbb{N}$ with $\varphi(x) = \sum_{i=0}^{r} b_i \cdot \binom{x-c_i}{i}$.*

Proof sketch. We inductively prove this claim by using the Chu-Vandermonde identity [21] on the term of highest degree. It allows us to replace the highest degree binomial via $\binom{x-c_r}{r} = \sum_{i=0}^{r} (-1)^{r-i} \binom{r-i+c_r-1}{r-i} \binom{x}{i}$. For sufficiently large $c_r \in \mathbb{N}$ this implies that the leading term of $\varphi(x) - a_r \cdot \binom{x-c_r}{r}$ again is positive and of smaller degree. ◀

A full proof of the previous lemma can be found in the appendix of the full version.

▶ **Lemma 3.17** (Integer-valued polynomials)**.** *Let $f \in$ #FA and let $\varphi : \mathbb{Q} \to \mathbb{Q}$ be an integer-valued polynomial, then $\max(\varphi \circ f, 0) \in$ #FA.*

Proof. We can assume the leading coefficient of φ to be positive. Otherwise $\max(\varphi \circ f, 0)$ can be directly written as a finite sum $\sum_i c_i \cdot \mathbb{1}_{f=i}$ which is in #FA by Lemmas 3.1, 3.2 and 3.11. Write φ in the binomial basis as $\varphi(x) = a_0 \cdot \binom{x}{0} + \ldots + a_r \cdot \binom{x}{r}$ with $a_r > 0$. Since φ is integer-valued, all of the a_i are integers, see [12, Prop. 4.2.1]. Using Lemma 3.16 we get a representation $\varphi(x) = \sum_{i=0}^{r} b_i \cdot \binom{x-c_i}{i}$ with $b_0, \ldots, b_r \in \mathbb{N}$ and $c_0, \ldots, c_r \in \mathbb{N}$. For $x \geq \max\{c_i \mid 0 \leq i \leq r\} =: N$ we have that $\binom{x-c_i}{i} = \binom{\max(x-c_i,0)}{i}$ and thus $\psi(x) := \sum_{i=0}^{r} b_i \cdot \binom{\max(x-c_i,0)}{i}$ only differs from $x \mapsto \max(\varphi(x), 0)$ on finitely many inputs and is a functional closure property of #FA by Lemmas 3.1, 3.2, 3.9 and 3.15. Lemma 3.12 then finishes off the claim. ◀

We now have the tools available to show our claim that every ultimately PORC function is a functional closure property of #FA.

▶ **Lemma 3.18.** *Every ultimately PORC function is a functional closure property of* #FA*.*

Proof. Let $\varphi : \mathbb{N} \to \mathbb{N}$ be an ultimately PORC function with period p comprised of the polynomial constituents $\varphi_0, \ldots, \varphi_{p-1} : \mathbb{N} \to \mathbb{Q}$ and $N \in \mathbb{N}$, s.t. for all $n \geq N$ we have $\varphi(n) = \varphi_{n \operatorname{rem} p}(n)$. Additionally let $f \in$ #FA. We can write

$$\varphi \circ f = \sum_{i=0}^{N-1} \mathbb{1}_{f=i} \cdot \varphi(i) + \mathbb{1}_{f \geq N} \cdot \left(\sum_{i=0}^{p-1} \mathbb{1}_{f \equiv_p i} \cdot \lfloor \max(\varphi_i \circ f, 0) \rfloor \right).$$

Figure 1 NFAs computing the functions $1^n \mapsto n$ and $1^n \mapsto 2^n$ respectively. Edges with a multiplicity of 2 are denoted by listing the edge label twice.

Combining Lemmas 3.1, 3.2, 3.11 and 3.14 this shows $\varphi \circ f \in \#\mathsf{FA}$, if we can show $\lfloor \max(\varphi_i \circ f, 0) \rfloor \in \#\mathsf{FA}$ for all $i \in \{0, \dots, p-1\}$. To show this let α_i be the common denominator of the coefficients of φ_i. Then $\alpha_i \cdot \varphi_i$ is a polynomial with integer coefficients, so it in particular is an integer-valued polynomial and by Lemma 3.17 we have that $\max(\alpha_i \cdot \varphi_i \circ f, 0) \in \#\mathsf{FA}$. Combining this with Lemma 3.13 we get that $\left\lfloor \frac{\max(\alpha_i \cdot \varphi_i \circ f, 0)}{\alpha_i} \right\rfloor = \lfloor \max(\varphi_i \circ f, 0) \rfloor \in \#\mathsf{FA}$. ◀

The remainder of this section is dedicated to showing that no other functional closure properties of $\#\mathsf{FA}$ exist. This will make use of the following well known algebraic interpretation of NFAs:

▶ **Lemma 3.19** (see [19]). *If $M = (Q, \Sigma, \mathrm{wt}, \mathrm{in}, \mathrm{out})$ is an NFA, then there are matrices $A_\sigma \in \mathbb{N}^{|Q| \times |Q|}$ for each symbol $\sigma \in \Sigma$ and vectors $a, b \in \mathbb{N}^{|Q|}$, s.t. M computes $a^T \cdot \left(\prod_{j=1}^{|w|} A_{w_j} \right) \cdot b$ for all $w \in \Sigma^\star$.*

Proof. We index A_σ, a and b using states $q, q' \in Q$. Choose $(A_\sigma)_{q,q'} = \mathrm{wt}(q, \sigma, q')$, $a_q = \mathrm{in}(q)$ and $b_q = \mathrm{out}(q)$. It is now easy to see that M computes exactly $a^T \cdot \left(\prod_{j=1}^{|w|} A_{w_j} \right) \cdot b$. ◀

When restricting to a unary alphabet $\Sigma = \{\sigma\}$, this degenerates the computed function to $a^T \cdot A_\sigma^{|w|} \cdot b$. In order to analyze the behaviour of these functions we first analyze the behaviour of the matrix power as a function in $|w|$ in the next two lemmas. Their proofs can be found in the appendix of the full version.

▶ **Lemma 3.20.** *Let $A \in \mathbb{N}^{k \times k}$. Then any diagonal entry of A^n is a function $f : \mathbb{N} \to \mathbb{N}$ with one of the following properties:*

1. *$f(n) = 0$ for all $n \in \mathbb{N} \setminus \{0\}$ and $f(0) = 1$.*
2. *There is a $p \in \mathbb{N} \setminus \{0\}$, such that for all $n \in \mathbb{N}$ we have $f(n) = \mathbb{1}_{n \equiv_p 0}$.*
3. *There is a $p \in \mathbb{N} \setminus \{0\}$ and a function $g \in 2^{\Theta(n)}$, such that for all $n \in \mathbb{N}$ we have $f(n) = \mathbb{1}_{n \equiv_p 0} \cdot g(n)$.*

These naturally correspond to vertices v in the multigraph defined by the adjacency matrix A with

1. no paths from v to v.
2. exactly one path from v to v of length p.
3. multiple walks from v to v where the lengths of all the walks from v to v have gcd p.

We can then lift this result to all entries of A^n.

▶ **Lemma 3.21.** *If $A \in \mathbb{N}^{k \times k}$, then each entry of A^n is an ultimately almost PORC function.*

▶ **Theorem 3.22** (Classification of univariate functional closure properties of $\#\mathsf{FA}$). *A function $\varphi : \mathbb{N} \to \mathbb{N}$ is a functional closure property of $\#\mathsf{FA}$ iff φ is an ultimately PORC function. This even holds when $\#\mathsf{FA}$ is restricted to unary languages.*

Proof. Lemma 3.18 already shows that every ultimately PORC function is a closure property of $\#\mathsf{FA}$. It remains to show that all functional closure properties of $\#\mathsf{FA}$ are ultimately PORC functions. For this let $\varphi : \mathbb{N} \to \mathbb{N}$ be a functional closure property of $\#\mathsf{FA}$. The function

$f : \{1\}^\star \to \mathbb{N}$ defined by $f(1^n) = n$ is computed by the left NFA in Figure 1 and thus in #FA. Consequently $\varphi \circ f \in$ #FA. Let $M = (Q, \{1\}, \mathrm{wt}, \mathrm{in}, \mathrm{out})$ be an NFA computing $\varphi \circ f$. By Lemma 3.19 this NFA induces a transition matrix $A \in \mathbb{N}^{|Q|\times|Q|}$ and vectors $a, b \in \mathbb{N}^{|Q|}$, s.t. $a^T A^n b = \varphi(f(1^n)) = \varphi(n)$ for all $n \in \mathbb{N}$. Every entry of A^n is an ultimately almost PORC function by Lemma 3.21 and thus $a^T A^n b = \varphi(n)$ is one as well. Let p be the quasiperiod of φ and let φ_i be one of the constituents of φ corresponding to some residue class $i \in \{0, \dots, p-1\}$. Assume for the sake of contradiction that φ_i grows in $2^{\Theta(n)}$, i.e. there is a constant $\gamma \in \mathbb{R}^+$ and $N \in \mathbb{N}$, s.t. for every $n \geq N$ we have $\varphi_i(n) \geq 2^{\gamma n}$. Consider the function $f_{p,i} : \{1\}^\star \to \mathbb{N}$ defined by $f_{p,i}(1^n) = p \cdot (2^n + N) + i$. We claim $f_{p,i}$ is in #FA. The function $1^n \mapsto 2^n$ is computed by the right NFA in Figure 1 and thus in #FA. The remainder of the claim follows by Lemma 3.1 and Lemma 3.2. Note that $f_{p,i}(1^n) \equiv_p i$, so $\varphi \circ f_{p,i} = \varphi_i \circ f_{p,i}$ has to be in #FA as well. Furthermore $\varphi(f_{p,i}(1^n)) = \varphi_i(f_{p,i}(1^n)) = \varphi_i(p \cdot (2^n + N) + i) \geq 2^{\gamma p \cdot (2^n+N)+\gamma i}$ for all $n \in \mathbb{N}$ which is larger than any NFA can compute, since NFAs can only compute functions that are at most linearly exponential in the length of the input. We conclude that none of the constituents φ_i of φ can be exponential, so they are instead all polynomials, making φ an ultimately PORC function. ◀

3.2 Multivariate functional closure properties

▶ **Theorem 3.23** (Classification of multivariate functional closure properties of #FA)**.** *A function $\varphi : \mathbb{N}^m \to \mathbb{N}$ is a functional closure property of* #FA *iff φ can be written as a finite sum of finite products of univariate ultimately PORC functions.*

Proof. By Theorem 3.22 any univariate ultimately PORC function is a closure property of #FA. As such any finite sum or finite product of them is also a closure property of #FA by Lemmas 3.1 and 3.2.

It remains to show that all functional closure properties of #FA are of this form. For this let $\varphi : \mathbb{N}^m \to \mathbb{N}$ be a functional closure property of #FA. Define the alphabet $\Sigma = \{\sigma_1, \dots, \sigma_m\}$ and the functions $f_i : \Sigma^\star \to \mathbb{N}$ where $f_i(w) := \#_i(w)$ is defined as the number of occurences $\#_i(w)$ of the symbol σ_i in w. Applying the closure property to $f_1, \dots, f_m$ gives that $\varphi \circ (f_1, \dots, f_m) \in$ #FA and thus is computed by an NFA $M = (Q, \Sigma, \mathrm{wt}, \mathrm{in}, \mathrm{out})$. This induces transition matrices $A_\sigma \in \mathbb{N}^{|Q|\times|Q|}$ for each symbol $\sigma \in \Sigma$ and vectors $a, b \in \mathbb{N}^{|Q|}$, s.t. $a^T \prod_{j=1}^{|w|} A_{w_j} b = \varphi(f_1(w), \dots, f_m(w))$ for all $w \in \Sigma^\star$. Restricting to words of the form $w = \sigma_1^{n_1}\sigma_2^{n_2}\cdots\sigma_m^{n_m}$ for $n_1, \dots, n_m \in \mathbb{N}$ gives $a^T \left(\prod_{i=1}^m A_{\sigma_i}^{n_i}\right) b = \varphi(n_1, \dots, n_m)$. Using Lemma 3.21 on each of the $A_{\sigma_i}^{n_i}$ we see that every entry of $A_{\sigma_i}^{n_i}$ is an ultimately almost PORC function in n_i. Consequently, every entry of $\prod_{i=1}^m A_{\sigma_i}^{n_i}$ is a finite sum of products of different ultimately almost PORC functions and the same holds for $a^T \left(\prod_{i=1}^m A_{\sigma_i}^{n_i}\right) b = \varphi(n_1, \dots, n_m)$.

We now look at the individual summands of φ and prove that we can rewrite each one as a product of ultimately PORC functions by one-by-one rewriting the exponential constituents. For this let $\varphi^{(1)}(n_1)\cdots\varphi^{(m)}(n_m)$ be one of the summands of φ where $\varphi^{(1)}, \dots, \varphi^{(m)}$ are all ultimately almost PORC functions, with periods $p_1, \dots, p_m$, offsets $N_1, \dots, N_m$ and constituents $\varphi_0^{(i)}, \dots, \varphi_{p_i-1}^{(i)}$ for each $i \in [m]$. If none of the constituents are exponential we are done. Otherwise let $\varphi_j^{(i)}$ be one of the exponential constituents, let $\gamma \in \mathbb{R}^+$ and let $N \in \mathbb{N}$, s.t. $\varphi_j^{(i)}(n_i) \geq 2^{\gamma n_i}$ for $n_i \geq N$. We claim we can set $\varphi_j^{(i)}(n_i) = 0$ without changing the product $\varphi^{(1)}(n_1)\cdots\varphi^{(m)}(n_m)$ for any $n_1, \dots, n_m \in \mathbb{N}$. Call the resulting functions $\psi^{(i)}$ and $\psi_j^{(i)}$. Assume for the sake of contradiction, that there are some $c_1, \dots, c_m \in \mathbb{N}$ where $\varphi^{(1)}(c_1)\cdots\varphi^{(m)}(c_m) \neq \varphi^{(1)}(c_1)\cdots\varphi^{(i-1)}(c_{i-1})\cdot\psi^{(i)}(c_i)\cdot\varphi^{(i+1)}(c_{i+1})\cdots\varphi^{(m)}(c_m)$. This implies that $\varphi^{(1)}(c_1)\cdots\varphi^{(i-1)}(c_{i-1})\cdot\varphi^{(i+1)}(c_{i+1})\cdots\varphi^{(m)}(c_m) \neq 0$ and $c_i \geq N_i$ as we didn't change any other functions except $\varphi^{(i)}$ for $n_i \geq N_i$.

Constructing constant functions $f'_k(w) = c_k$ for $k \neq i$ and the function $f'_i(w) = p_i \cdot (2^{|x|} + \max(N_i, N)) + j$ which are all in #FA. We see that $\varphi \circ (f'_1, \ldots, f'_m) \in$ #FA. Note that for any $w \in \Sigma^\star$ we have $f'_i(w) \geq \max(N_i, N)$ and $f'_i(w) \equiv_{p_i} j$ and thus $\varphi^{(i)} \circ f'_i = \psi^{(i)}_j \circ f'_i$. Combining all of this we again reach a contradiction to the fact that NFAs can only compute at most linearly exponential functions via

$$\begin{aligned}\varphi(f'_1(w), \ldots, f'_m(w)) &\geq \varphi^{(1)}(c_1) \cdots \varphi^{(i-1)}(c_{i-1}) \cdot \varphi^{(i)}(f'_i(w)) \cdot \varphi^{(i+1)}(c_{i+1}) \cdots \varphi^{(m)}(c_m) \\ &\geq \varphi^{(i)}(f'_i(w)) \;=\; \varphi^{(i)}_j(f'_i(w)) \;=\; \varphi^{(i)}_j(p_i \cdot (2^{|w|} + \max(N_i, N)) + j) \\ &\geq 2^{\gamma p_i \cdot (2^{|w|} + \max(N_i, N)) + \gamma j}\,.\end{aligned}$$

Note that the first inequality holds due to all summands of φ being non-negative. ◀

Deciding whether a function φ has such a representation may not always be directly visible, however if φ is a multivariate polynomial we can be more explicit. Every integer-valued multivariate polynomial has integer coefficients when represented in the binomial basis (see [12, Prop. 4.2.1] for a proof of this fact). We say a term $a \cdot \binom{x_1}{d_1} \cdots \binom{x_m}{d_m}$ dominates another term $a' \cdot \binom{x_1}{d'_1} \cdots \binom{x_m}{d'_m}$ if $d_i \geq d'_i$ for all $i \in [m]$. A term is a dominating term of φ if it has non-zero coefficient and it is not dominated by any other term with non-zero coefficient. We can use a similar approach to Lemma 3.16 to rewrite φ as a positive integer linear combination of products of shifted binomials (details can be found in the appendix of the full version).

▶ **Lemma 3.24.** *Let $\varphi(x_1, \ldots, x_m) = \sum_{i=1}^{r} a_i \cdot \prod_{j=1}^{m} \binom{x_j}{d_{i,j}}$ with $a_i \in \mathbb{Z}$ and the coefficients of the dominating terms being positive. Then there are $a'_1, \ldots, a'_{r'} \in \mathbb{N}$ and $c_1, \ldots, c_{r'} \in \mathbb{N}$ with $\varphi(x_1, \ldots, x_m) = \sum_{i=1}^{r'} a'_i \cdot \prod_{j=1}^{m} \binom{x_j - c_i}{d_{i,j}}$.*

We can then use a generalization of Lemma 3.17 and replace subtractions $x_i - c'$ by $\max(x_i - c', 0)$. However special care has to be taken for $x_i < c'$, in which case x_i has to be replaced by the corresponding constants first. This adds the additional condition on φ.

▶ **Lemma 3.25.** *Let $\varphi : \mathbb{N}^m \to \mathbb{N}$ be a multivariate polynomial with rational coefficients, such that whenever ψ is formed from φ by replacing any set of variables – including the empty subset – by constants from $\mathbb{N}$, then all dominating terms of ψ have positive coefficients. Then φ is a functional closure property of* #FA.

▶ **Lemma 3.26.** *A multivariate polynomial $\varphi : \mathbb{N}^m \to \mathbb{N}$ with rational coefficients is a functional closure property of* #FA *iff for every ψ that can be formed from φ by replacing any subset of variables – including the empty set – by constants from $\mathbb{N}$, then all dominating terms of ψ have positive coefficients.*

Proof sketch. Lemma 3.25 already proves that all multivariate polynomials of this form are functional closure properties of #FA. Now let $\varphi : \mathbb{N}^m \to \mathbb{N}$ be a multivariate polynomial with rational coefficient and a functional closure property of #FA. By Theorem 3.23 φ can be written as a finite sum of finite products of ultimately PORC functions. Note that the leading coefficient of each constituent of these ultimately PORC functions is positive. By multivariate polynomial interpolation we can now show that φ is already a finite sum of finite products of these constituents. The dominating terms of φ are then formed by products of the leading coefficients of the constituents and thus are positive. ◀

4 Promise closure properties

▶ **Definition 4.1.** *Let $S \subseteq \mathbb{N}^m$ and let $\varphi : \mathbb{N}^m \to \mathbb{N}$ be a function. We call φ a* functional promise closure property *of #FA with regard to S if for every $f_1, \ldots, f_m \in$ #FA defined on some shared alphabet Σ there is a function $g \in$ #FA with $g(w) = \varphi(f_1(w), \ldots, f_m(w))$ for every $w \in \Sigma^\star$ for which $(f_1(w), \ldots, f_m(w)) \in S$.*

We now want to show that if S fulfils some property, namely admitting polynomial cluster sequences (we postpone the definition to Definition 4.5), then for every functional promise closure property with regard to S there is a functional closure property of #FA that agrees with it on all tuples in S. In other words we can interpolate the functional promise closure property φ on all values of S to obtain a functional closure property φ' for all of #FA. The proof for this follows along the following ideas: First, similar to Theorem 3.23, use the functional closure property φ on the unary counting functions to find an equivalent function φ' that is almost a functional closure property. However after this step some of the constituents of the ultimately almost PORC functions might still be exponential. Theorem 3.23 then proceeded by showing that we can replace all these exponential constituents by the constant zero function, as otherwise we were able to reach a contradiction by constructing functions $f'_1, \ldots, f'_m$ and an infinite sequence of inputs $w^{(i)}$, such that $\varphi(f'_1(w^{(i)}), \ldots, f'_m(w^{(i)}))$ grows doubly exponential in the length of the inputs. However when dealing with functional promise closure properties we have to be more careful when choosing $f'_1, \ldots, f'_m$ and the $w^{(i)}$, because we need $(f'_1(w^{(i)}), \ldots, f'_m(w^{(i)})) \in S$ to reach a contradiction. Additionally we don't replace the exponential constituents by the constant zero-function but rather a polynomial that behaves the same for small inputs. For this we need a special variant of univariate polynomial interpolation that yields integer-valued polynomials that are non-negative for all inputs from $\mathbb{N}$:

▶ **Lemma 4.2.** *Let $c_0, \ldots, c_N \in \mathbb{N}$. Then there is an integer-valued polynomial $q : \mathbb{Q} \to \mathbb{Q}$ with $q(n) = c_n$ for all $n \in \{0, \ldots, N\}$ and $q(n') \geq 0$ for all $n' \in \mathbb{N}$.*

To be able to hit all of S consistently we use independent binary encodings, that allow us to hit all of $\mathbb{N}^m$.

▶ **Lemma 4.3** (Folklore)**.** *The function $f : \{0,1\}^\star \to \mathbb{N}$ defined by being the value of $w \in \{0,1\}^\star$ interpreted as a binary number is in #FA. Additionally it is possible to extend the domain of f to any alphabet $\Sigma \supseteq \{0,1\}$ where the value of f is determined while ignoring any symbols not in $\{0,1\}$.*

For any $n \in \mathbb{N}$ we denote by $\mathrm{bin}(n)$ the unique binary representation of n without leading zeros. We first show the methodology in detail by proving the univariate case.

▶ **Theorem 4.4.** *Let $S \subseteq \mathbb{N}$. Then any function $\varphi : \mathbb{N} \to \mathbb{N}$ is a functional promise closure property of #FA with regard to S iff there is a functional closure property $\psi : \mathbb{N} \to \mathbb{N}$ of #FA with $\varphi_{|S} = \psi_{|S}$.*

Proof. If such a ψ exists, we directly see that φ is a functional promise closure property of #FA with respect to S. Indeed for any $f \in$ #FA we construct $g = \psi \circ f \in$ #FA and see $g(w) = \varphi(f(w))$ for every $w \in \Sigma^\star$ with $f(w) \in S$.

Now on the other hand let φ be any functional promise closure property of #FA with regard to S. Again define the alphabet $\Sigma = \{1\}$ and the function $f : \Sigma^\star \to \mathbb{N}$ with $f(1^n) := n$. Applying the closure property to f gives that there is some $g \in$ #FA with $g(1^n) = \varphi(f(1^n)) = \varphi(n)$ for all $n \in S$. Let $M = (Q, \Sigma, \mathrm{wt}, \mathrm{in}, \mathrm{out})$ be an NFA computing g.

This induces a transition matrix $A \in \mathbb{N}^{|Q|\times|Q|}$ and vectors $a, b \in \mathbb{N}^{|Q|}$, s.t. $a^T A^n b = g(1^n)$ for all $n \in \mathbb{N}$ by Lemma 3.19. We now define $\chi(n) := a^T A^n b$ and see $\chi_{|S} = \varphi_{|S}$. Using Lemma 3.21 on A^n we see that every entry of A^n is an ultimately almost PORC function in n and as such ψ is also an ultimately almost PORC function. Let p be the quasiperiod of χ and let $\chi_0, \ldots, \chi_{p-1}$ be the constituents of χ and let $N \in \mathbb{N}$ be the offset after which χ is defined by the constituents.

We claim that we can now replace every exponential constituent by a polynomial one without changing the value of χ for any $n \in S$. For each $i \in \{0, \ldots, p-1\}$ we distinguish two cases, depending on whether $S_i := S \cap (p\mathbb{Z}+i)$ is finite or it is infinite. If S_i is a finite set, we replace χ_i' with a polynomial that interpolates the same values as χ_i on S_i. Lemma 4.2 ensures that this polynomial is integer-valued and non-negative for all of $\mathbb{N}$. If S_i is an infinite set, we replace χ_i by the constant zero function. Call the resulting ultimately PORC function ψ with constituents ψ_i. Assume for the sake of contradiction there is an $c \in S$, s.t. $\chi(c) \neq \psi(c)$. Clearly such a c would have to be at least N. Now let i be, s.t. $c \in S_i$. It must hold that $\chi_i(c) \neq \psi_i(c)$. Hence S_i cannot be a finite set, since χ_i and ψ_i agree on S_i. Therefore S_i must be infinite. Since χ_i is exponential there is a $\gamma \in \mathbb{R}^+$ and $N' \in \mathbb{N}$, s.t. $\chi_i(n) \geq 2^{\gamma n}$ for all $n \geq N'$. Let $f' \in \#\mathsf{FA}$ be the function of binary evaluation from Lemma 4.3 over the alphabet $\Sigma' = \{0,1\}$. Then there is a $g' \in \#\mathsf{FA}$ with $g'(\mathrm{bin}(n)) = \varphi(f'(\mathrm{bin}(n))) = \chi(f'(\mathrm{bin}(n)))$ for all $n \in S$. Since S_i is infinite, in particular S_i must contain infinitely many values bigger than $\max(N, N')$. For $n \in S_i$ with $n \geq \max(N, N')$ we now have

$$g'(\mathrm{bin}(n)) = \chi(f'(\mathrm{bin}(n))) = \chi(n) = \chi_i(n) \geq 2^{\gamma n} \geq 2^{\gamma 2^{|\mathrm{bin}(n)|-1}},$$

which is a contradiction to NFAs only being able to only compute functions that are at most linearly exponential in the input length. In conclusion S_i cannot be infinite either and thus c itself cannot exist. ◀

▶ **Definition 4.5.** *A set $S \subseteq \mathbb{N}^m$ admits* polynomial cluster sequences *if for every $N_1, \ldots, N_m \in \mathbb{N}$, and $p_1, \ldots, p_m \in \mathbb{N}$ the projection $\tau : S \to \{0, \ldots, N_1 + p_1 - 1\} \times \ldots \times \{0, \ldots, N_m + p_m - 1\}$ defined by $\tau(n_1, \ldots n_m) = (n_1 \operatorname{srem}_{N_1} p_1, \ldots, n_m \operatorname{srem}_{N_m} p_m)$ has the following property: Any preimage T of a singleton set under τ for every $i \in [m]$ has either bounded i-th coordinate or there is a polynomial $q : \mathbb{N} \to \mathbb{N}$ and an infinite subset $T' \subseteq T$ with unbounded i-th coordinate*[9] *and with $\sum_{j=1}^{m} n_j \leq q(n_i)$ for all $(n_1, \ldots, n_m) \in T'$. We call such an infinite subset a* polynomial cluster sequence *with regards to dimension i.*

We call τ the shifted grid projection *of S with respect to offsets $N_1, \ldots, N_m$ and quasiperiods $p_1, \ldots, p_m$.*

Intuitively this definition requires that each dimension is either bounded or can grow reasonably quickly together with the other dimensions, even when restricted to inputs from specific shifted residue classes. For example $\{(n^2, n^3) \mid n \in \mathbb{N}\}$ admits polynomial cluster sequences, while $\{(n, 2^n) \mid n \in \mathbb{N}\}$ does not.

▶ **Theorem 4.6.** *Let $S \subseteq \mathbb{N}^m$ admit polynomial cluster sequences. Then any function $\varphi : \mathbb{N}^m \to \mathbb{N}$ is a functional promise closure property of $\#\mathsf{FA}$ with regard to S iff there is a functional closure property $\psi : \mathbb{N}^m \to \mathbb{N}$ of $\#\mathsf{FA}$ with $\varphi_{|S} = \psi_{|S}$.*

The technical proof of the theorem can be found in the appendix of the full version. There are multiple natural families for the set S such that S admits polynomial cluster sequences which we describe in the following.

[9] note that this property also follows directly from the polynomial bound on the other coordinates in the i-th coordinate, but we have it as part of the definition for clarity.

▶ **Lemma 4.7.** *Any finite set $S \subseteq \mathbb{N}^m$ admits polynomial cluster sequences.*

Proof. Independent of the offsets and quasiperiods and $i \in [m]$ any subset of S always is finite and thus bounded in every dimension. ◀

An affine variety is defined as the zero set of a finite number of multivariate polynomials. A special case of affine varieties are *graph varieties* (also just called *graphs*, see [20, §2.4, Exe. 12]). An affine variety S is a graph variety if there exist a finite number of j-variate polynomials $\mu_1, \ldots, \mu_k$ such that $S = \{(s_1, \ldots, s_j, \mu_1(s_1, \ldots, s_j), \ldots, \mu_k(s_1, \ldots, s_j)) \mid (s_1, \ldots, s_j) \in \mathbb{Q}^j\} \subseteq \mathbb{Q}^{j+k}$. We call $s_1, \ldots, s_j$ the *free* variables, and the remaining variables the *dependent* variables. We call S a monotone graph variety if $\mu_1, \ldots, \mu_k$ are all monotone.

▶ **Lemma 4.8.** *Let $S \subseteq \mathbb{Q}^m = \mathbb{Q}^{j+k}$ be a monotone graph variety. Then the set $S \cap \mathbb{N}^m$ admits polynomial cluster sequences.*

The proof of this lemma can be found in the appendix of the full version. All in all this combines to the following theorem which characterizes the special case of multivariate polynomial functional promise closure properties, the technical details can be found in the appendix of the full version.

▶ **Theorem 4.9.** *Let $S \subseteq \mathbb{Q}^m = \mathbb{Q}^{j+k}$ be a monotone graph variety and let $I = I(S)$ be its vanishing ideal. A multivariate polynomial $\varphi : S \to \mathbb{N}$ is a functional promise closure property of #FA with regard to S if and only if there exists a $\psi \in I$ such that $\varphi + \psi$ is a multivariate functional closure property of #FA.*

5 Conclusion

We characterized the functional closure properties of #FA to be precisely the ultimately PORC functions in the univariate case and combinations of ultimately PORC functions in the multivariate case. Additionally we characterize promise functional closure properties of #FA with regard to some natural families of sets S. Natural further directions of research are now whether we can characterize the promise functional closure properties of #FA for more sets S and whether our methods can be applied to characterize functional closure properties for more powerful models of computation.

References

1 Rudolf Ahlswede and David E Daykin. An inequality for the weights of two families of sets, their unions and intersections. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 43:183–185, 1978.

2 Noga Alon and Joel H. Spencer. *The Probabilistic Method, Third Edition*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 2008.

3 Edwin F Beckenbach and Richard Bellman. *Inequalities*. Springer, Berlin, 1961.

4 Richard Beigel. Closure properties of GapP and #P. In *Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems*, pages 144–146. IEEE, 1997. doi:10.1109/ISTCS.1997.595166.

5 Swee Hong Chan and Igor Pak. Computational complexity of counting coincidences. *CoRR*, abs/2308.10214, 2023. doi:10.48550/arXiv.2308.10214.

6 Swee Hong Chan and Igor Pak. Equality cases of the alexandrov-fenchel inequality are not in the polynomial hierarchy. *CoRR*, abs/2309.05764, 2023. doi:10.48550/arXiv.2309.05764.

7 David A. Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms – An introduction to computational algebraic geometry and commutative algebra (2. ed.)*. Undergraduate texts in mathematics. Springer, 1997.

8 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of weighted automata*. Springer Science & Business Media, 2009.

9 Manfred Droste and Dietrich Kuske. Weighted automata. In Jean-Éric Pin, editor, *Handbook of Automata Theory*, pages 113–150. European Mathematical Society Publishing House, Zürich, Switzerland, 2021. doi:10.4171/Automata-1/4.

10 HG Hardy, JE Littlewood, and G. Pólya. *Inequalities*. Cambridge University Press, 1952.

11 Ulrich Hertrampf, Heribert Vollmer, and Klaus W Wagner. On the power of number-theoretic operations with respect to counting. In *Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference*, pages 299–314. IEEE, 1995. doi:10.1109/SCT.1995.514868.

12 Christian Ikenmeyer and Igor Pak. What is in #P and what is not? In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 860–871. IEEE, 2022. doi:10.1109/FOCS54457.2022.00087.

13 Christian Ikenmeyer, Igor Pak, and Greta Panova. Positivity of the symmetric group characters is as hard as the polynomial time hierarchy. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3573–3586. SIAM, 2023. doi:10.1137/1.9781611977554.ch136.

14 Ines Klimann, Sylvain Lombardy, Jean Mairesse, and Christophe Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theoretical Computer Science*, 327(3):349–373, 2004. doi:10.1016/j.tcs.2004.02.049.

15 Igor Pak. Complexity problems in enumerative combinatorics. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pages 3153–3180. World Scientific, 2018.

16 Igor Pak. Combinatorial inequalities. *Notices of the AMS*, 66(7), August 2019.

17 Igor Pak. What is a combinatorial interpretation? to appear: Proc. Open Problems in Algebraic Combinatorics. https://www.samuelfhopkins.com/OPAC/files/proceedings/pak.pdf, 2022.

18 J Peterson. Beviser for wilsons og fermats theoremer. *Tidsskrift for mathematik*, 2:64–65, 1872.

19 Marcel Paul Schützenberger. On the definition of a family of automata. *Inf. Control.*, 4(2-3):245–270, 1961. doi:10.1016/S0019-9958(61)80020-X.

20 Igor R Shafarevich. *Basic algebraic geometry 1: Varieties in projective space*. Springer Science & Business Media, 3 edition, 2013.

21 Michael Spivey. The Chu-Vandermonde identity via Leibniz's identity for derivatives. *The College Mathematics Journal*, 47(3):219–220, 2016.

22 Richard P Stanley. Positivity problems and conjectures in algebraic combinatorics. *Mathematics: frontiers and perspectives*, 295:319, 1999.

A Finite Presentation of Graphs of Treewidth at Most Three

Amina Doumane
Plume, LIP, CNRS, ENS de Lyon, France

Samuel Humeau
Plume, LIP, CNRS, ENS de Lyon, France

Damien Pous
Plume, LIP, CNRS, ENS de Lyon, France

Abstract

We provide a finite equational presentation of graphs of treewidth at most three, solving an instance of an open problem by Courcelle and Engelfriet.

We use a syntax generalising series-parallel expressions, denoting graphs with a small interface. We introduce appropriate notions of connectivity for such graphs (components, cutvertices, separation pairs). We use those concepts to analyse the structure of graphs of treewidth at most three, showing how they can be decomposed recursively, first canonically into connected parallel components, and then non-deterministically. The main difficulty consists in showing that all non-deterministic choices can be related using only finitely many equational axioms.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting; Mathematics of computing → Paths and connectivity problems

Keywords and phrases Graphs, treewidth, connectedness, axiomatisation, series-parallel expressions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.135

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://hal.science/hal-04560570` [10]

Supplementary Material *Software*: `https://perso.ens-lyon.fr/damien.pous/hypergraph/` [12] archived at `swh:1:dir:028863b2f75dde258591611a6c7c165e289db890`

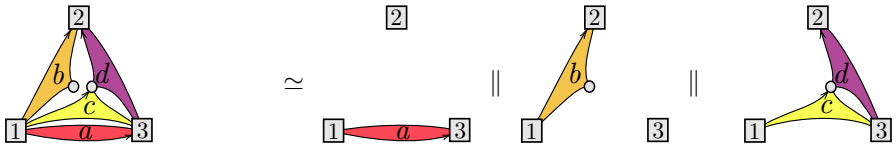
Funding This work was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX- 0007) operated by the French National Research Agency (ANR).

Acknowledgements The second author would like to thank Ugo Giocanti for several discussions about graph minors and connectivity.

1 Introduction

Treewidth is a graph parameter measuring how close a graph is from a forest. It is frequently encountered in parameterised complexity: many NP-complete problems become polynomial or even linear once parameterised using treewidth [4]. This parameter admits many equivalent definitions. It was discovered at least by Bertelè and Brioschi [3], Halin [11], and Robertson and Seymour [13] via tree decompositions for their celebrated graph minor theorem. It was subsequently characterised using k-trees [2], k-elimination graphs [17, 16, 15], and chordal graphs [13].

One may also consider syntaxes, most often generalising series-parallel expressions [1, 8, 6]. There, the idea is to use terms to denote graphs, and an important problem is to understand when two terms denote the same graph. To this end, Courcelle and Engelfriet gave an



$$\mathrm{l}\mathrm{f}b \parallel \mathrm{f}\big((24)\mathrm{l}(c \parallel (23)\mathrm{l}a) \parallel (14)\mathrm{l}d\big) \equiv (23)\mathrm{l}a \parallel \mathrm{l}\mathrm{f}b \parallel \mathrm{f}\big((24)\mathrm{l}c \parallel (14)\mathrm{l}d\big)$$

Figure 1 Two parsings of a given graph.

equational axiomatisation for arbitrary graphs [7, p. 117]: a structured set of equations on terms from which it is possible to equate all terms denoting a given graph. Unfortunately, this axiomatisation is infinite. They note how finite fragments of the syntax make it possible to capture precisely the graphs up to a given treewidth, while finite restrictions of their axiomatisation seem to be incomplete.

This brings them to the following question [7, p. 118]: for a given bound k, is there a finite equational presentation of graphs of treewidth at most k? The case of $k = 1$ concerns forests and is relatively easy. The case of $k = 2$ has been given a solution a few years ago [5, 9]. Here we give a positive answer for $k = 3$: we provide a syntax of terms with an interpretation map g from terms to graphs whose image is exactly the set of graphs of treewidth at most three, and we give a finite list of equations whose equational theory ($\equiv$) characterises graph isomorphism ($\simeq$). In symbols, we prove that for all terms t, u,

$$g(t) \simeq g(u) \quad \text{if and only if} \quad t \equiv u$$

Like Arnborg, Courcelle, Proskurowski and Seese [1], we work with hypergraphs with a list of designated vertices, the *sources*, used as an interface to perform the following operations:

- *parallel composition* ($\parallel$): glue the graphs together along their sources.
- *permutation* (pG): given a permutation p, reorder the sources of G according to p.
- *lift* ($\mathrm{l}G$): add an isolated vertex to G and append it as last source.
- *forget* ($\mathrm{f}G$): remove the last source of G (keeping it as a mere vertex of the graph).

Consider the four graphs depicted in Figure 1, each with three sources denoted with numbered squares. The neighbours of each edge are ordered, which we indicate by drawing an arrow from the first to the second neighbour. The first graph on the left is the parallel composition of the three other ones. The second one can be obtained from a binary edge a (with interface its endpoints) by applying a lift to add a third isolated source and then swapping the last two sources: $(23)\mathrm{l}a$. The third one can be obtained from a ternary edge b (again with interface its endpoints), by forgetting the third source and adding a fresh one via a lift: $\mathrm{l}\mathrm{f}b$. The last one can be constructed as $\mathrm{f}\big((24)\mathrm{l}c \parallel (14)\mathrm{l}d\big)$: reasoning top-down, we promote the inner vertex as a fourth source, and then we put in parallel two graphs each with a single ternary edge connecting three out of the four sources – both being obtained as appropriate permutations of lifted edges.

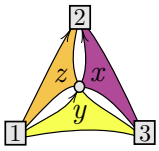
We call *parsings* the expressions we obtain when decomposing graphs as above.

The way we parsed the graph on the right generalises to all graphs: first promote all inner vertices as sources, and then build a large parallel composition of appropriately permuted and lifted edges. For instance, the first graph on the left also admits the following parsing:

$$\mathrm{f}\mathrm{f}\big((23)\mathrm{l}\mathrm{l}\mathrm{l}a \parallel (35)\mathrm{l}\mathrm{l}b \parallel (24)\mathrm{l}\mathrm{l}c \parallel (124)\mathrm{l}\mathrm{l}d\big)$$

While such a parsing exists for all graphs, it goes through an intermediate graph with a large interface. Instead, the graphs of treewidth at most k are exactly those for which we can find a parsing where all intermediate graphs have at most $k+1$ sources (Proposition 4.1, [1, Proposition 4.1]). This syntax for denoting precisely the graphs of treewidth at most k is the starting point for the present work.

A derived operation is of particular interest. Consider three graphs x, y, z each with three sources, and combine them as depicted on the right to obtain a new graph $\circ(x,y,z)$. This operation can be defined from the previous ones using the expression below.



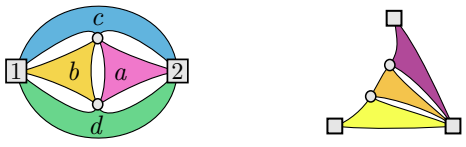
$$\circ(x,y,z) \triangleq \mathrm{f}\big((14)\mathrm{l}x \parallel (24)\mathrm{l}y \parallel (34)\mathrm{l}z\big)$$

It can be generalised into a n-ary operation on graphs with n sources, and when $n=2$ we recover the usual notion of *series composition* (with its arguments reversed).

Our goal is then to understand which laws are satisfied by the previous operations. Amongst the natural ones, we have that parallel composition is associative and commutative, that permutations commute over parallel compositions, and that applying two permutations in a row amounts to applying the composite permutation. There are more involved ones. For instance, we may also parse the first graph of Figure 1 by keeping edges a and c together as long as possible, resulting in the expression written below it. We thus have two rather different parsings for the same graph, which our axiomatisation should equate.

We proceed in two steps. First we use connectivity arguments in order to decompose any graph into a parallel composition of permuted lifts of *full prime graphs*: non-empty graphs which are connected through paths not using sources except possibly at their endpoints. Figure 1 actually provided an example of such a decomposition, which is always unique for a given graph. Using a few natural axioms, we show that every term can be rewritten under such a form (Proposition 5.1). This makes it possible to focus on full prime graphs in the second step, which is where the main difficulties arise.

A key property of full prime graphs of bounded treewidth is that either they are *atomic* (i.e., reduced to a permutation of an edge), or they have a *forget point*: at least one of their inner vertices can be promoted into a source without increasing the treewidth, thus making it possible to parse the graph as a forget operation. The difficulty is that forget points are not unique, resulting in several ways of parsing non-atomic full prime graphs. Consider for instance the following tetrahedron on the left, with only two vertices marked as sources.



Each of the two inner vertices is a forget point, so that modulo appropriate permutations of edges, we may parse this graph as $\mathrm{f}(\circ(a,b,c) \parallel d)$ or as $\mathrm{f}(\circ(a,b,d) \parallel c)$. In this case, the two forget points are relatively close, and one of our axioms makes it possible to jump directly from one parsing to the other. A similar situation arises with the graph given on the right.

The cornerstone of our completeness proof is the fact that two parsings of a non-atomic full prime graph can always be rewritten so as to agree on their forget point. This is Lemma 5.4, and most of the paper is devoted to proving it.

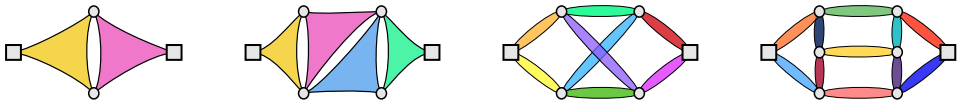


Figure 2 Separation pairs and graphs with several separation pairs.

To do so, we first delimit a class of vertices which we call *anchors*. Those can be thought of as a generalisation of cutvertices [8, Section 1.4]. When a full prime graph has an anchor, we show that we can use it as a universal agreement point (Lemma 7.1).

Unfortunately, there are graphs without anchors. We call them *hard*, they can be thought of as specific unseparable graphs [8, Section 1.4]. The rest of the proof consists in a structural analysis of hard graphs of treewidth at most three. We show that they admit *separation pairs*: every hard graph has the shape on the left of Figure 2, and every parsing can be rewritten as a double forget on some of these separation pairs (Lemma 8.5). However, as illustrated on the right, separation pairs are not unique. In order to finish the proof, we analyse how separation pairs relate to each other. There are easy cases like in the second graph of Figure 2 where the diagonal separation pair connects the two vertical ones, and previously encountered axioms make it possible to conclude. By analysing the shape of graphs excluding the triangle between their sources as a minor (Proposition 8.7 – note that cycles may still appear in such graphs), we show that all other situations reduce to one of the two graphs on the right of Figure 2. There, the two outer-vertical pairs of inner vertices are the only separation pairs, and they are disjoint. They correspond to distinct parsings of the same graph, and we must include the corresponding equations to complete our list of axioms.

For the sake of readability, some proofs and details are provided in the appendix of the full version [10].

2 Graphs, treewidth

A *ranked set* (or *signature*) is a set where every element has an associated natural number called its *arity*. Throughout the paper we fix a ranked set Σ of *letters*, which we call the *alphabet*. Given a set V, we denote by $\mathcal{L}(V)$ the ranked set of duplicate-free lists over V, where the arity of a list is its length.

We consider labelled and ordered hypergraphs with interfaces, defined as follows:

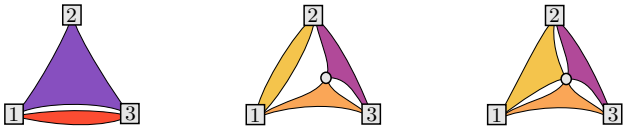
▶ **Definition 2.1.** *A* graph *is a tuple* $\langle V, E, n, l, i\rangle$ *where* V *is a finite set of* vertices*,* E *is a finite ranked set of* edges*,* $i \in \mathcal{L}(V)$ *is the* interface*, and* $n\colon E \to \mathcal{L}(V)$ *and* $l\colon E \to \Sigma$ *are arity-preserving functions respectively giving the* neighbours *and the* label *of each edge.*

The *elements* of a graph are its vertices and edges. The vertices appearing in the interface of a graph are its *sources*. Vertices (resp. elements) which are not sources are called *inner vertices (resp. elements)*. A vertex is *isolated* if it is not in the neighbourhood of any edge. The *size* of a graph G, written $|G|$, is its number of elements. The *arity* of a graph is that of its interface. Two graphs G, H are *isomorphic*, written $G \simeq H$, if their vertex and edge sets are related by structure-preserving bijections (cf. [10, Appendix A]). A graph is:

- *empty* if its only elements are its sources;
- *atomic* if its only inner element is an edge, whose neighbours comprise all the sources.

Our choice of considering ordered hypergraphs with interfaces comes from the need to be able to substitute graphs for edges. A *substitution* is an arity-preserving function from the alphabet to graphs. Given a graph G and a substitution σ, we write $G\sigma$ for the graph

obtained from G by replacing all its a-labelled edges by copies of the graph $\sigma(a)$, identifying pairwise the neighbours of the edges with the interfaces of the copies. We say that G *has shape* H when $G \simeq H\sigma$ for some substitution σ. For instance, the first graph of Figure 1 has the following shapes (amongst other ones):



(Note that the edge orientation is irrelevant in a graph used as a shape, as well as the labelling function – as long as it is injective, which will always be the case in the present paper.)

▶ **Proposition 2.2.** *For all graphs G, H and all substitutions σ, ρ, if $G \simeq H$ and $\sigma(a) \simeq \rho(a)$ for all letters a, then $G\sigma \simeq H\rho$.*

A graph in the sense of Definition 2.1 can be seen as a simple graph, its *skeleton*, by turning all its edges and its interface into cliques (removing duplicate edges if necessary). This operation makes it possible to use the standard notion of treewidth [8, Section 12.4]: the *treewidth* of a graph is that of its skeleton. Equivalently, we can define it as follows.

▶ **Definition 2.3.** *A* tree decomposition *of a graph is a tree whose nodes are labelled by sets of vertices, called* bags*, such that*

- *there is a bag containing all sources,*
- *for each edge there is a bag containing its neighbours,*
- *for each vertex, the bags containing it form a subtree.*

The width *of a tree decomposition is the size of its largest bag minus one. The* treewidth *of a graph is the minimal width of its possible tree decompositions.*

Observe that since the sources of a graph must be contained in a bag for all tree decompositions, the treewidth of a graph is at least its arity minus one. Therefore, at treewidth at most three, we only have graphs of arity up to four. The same constraint holds for the edge arities.

▶ **Proposition 2.4.** *If a graph G and the graphs in the image of a substitution σ all have treewidth at most k, then so does $G\sigma$.*

3 Graph operations

We can now define the operations discussed in the introduction.

▶ **Definition 3.1.** *Let G, H be graphs of arity k.*

- *The* parallel composition *of G and H, $G \parallel H$, is the graph of arity k obtained from the disjoint union of G and H by pairwise merging their sources.*
- *If $k \geq 1$, the* forget *of G, $\mathrm{f}G$, is the graph of arity $k-1$ obtained from G by removing the last vertex from its interface (keeping it in the vertex set).*
- *The* lift *of G, $\mathrm{l}G$, is the graph of arity $k+1$ obtained from G by adding a new isolated vertex, and appending it to its interface.*
- *For a permutation p of $[1, k]$, we denote by pG the graph obtained from G by permuting its interface according to p.*
- *We write $\emptyset_k$ for the empty graph of arity k, omitting k when it is clear from the context.*
- *For a letter $a \in \Sigma$, we also write a for the atomic graph whose edge is labelled a and has its neighbours appearing in the same order as in the interface.*

These operations all arise as substitutions of well chosen small graphs. By Proposition 2.4, this entails that they preserve treewidth (except of course for lifts, which may increase it when reaching the maximal arity). It remains to establish a few structural properties, in order to show that every graph of treewidth at most k can be constructed from the above operations up to arity $k+1$ (Proposition 4.1 below).

A *path* in a graph is a sequence of elements such that any two consecutive elements consist of an edge and one of its neighbours. An *inner path* is a path made only of inner elements, except possibly for the endpoints.

▶ **Definition 3.2.** *A graph is* prime *if it is not empty and all its inner elements are connected by inner paths; it is* full *if none of its sources are isolated.*

Observe that a graph is full prime iff all its elements are connected by inner paths. Amongst the graphs in Figure 1, the first one is full but not prime, the three other ones are prime, and only the last one is full prime.

▶ **Lemma 3.3.** *Every graph is isomorphic to a permutation of lifts of a full graph. This decomposition is unique up to permutation and isomorphism.*

▶ **Lemma 3.4.** *Every graph is isomorphic to a parallel composition of prime graphs. This decomposition is unique up to reindexing and isomorphism.*

We call *(prime) components of a graph* the prime graphs occurring in the latter decomposition. We call *reduced components of a graph* the full prime graphs obtained by removing isolated sources from its components.

We now give two key properties of full prime graphs of bounded treewidth. First, they are always atomic at maximal arity.

▶ **Proposition 3.5.** *Full prime graphs of treewidth at most k and arity $k+1$ are atomic.*

In the case of treewidth at most three, this means that when we decompose a graph of arity four into prime components, then except for a number of four-edges between the sources, we only get non-full components: drawing the graph as a tetrahedron between its sources, the non-atomic components are glued through the faces (or edges, or vertices, or nothing), and the interior of the tetrahedron remains empty. This corresponds to the characterisations of treewidth at most k graphs as partial k-trees [2] or as subgraphs of chordal graphs whose cliques are of size at most $k+1$ [13].

Second, non-atomic full prime graphs have *forget points*, which we define as follows.

▶ **Definition 3.6.** *When x is an inner vertex of a graph G, we write (G, x) for the graph obtained from G by appending x to its interface; if this graph has treewidth at most k then we say that x is a* k-forget point of G.

▶ **Proposition 3.7.** *Non-atomic full prime graphs of treewidth at most k have k-forget points.*

4 Terms and axioms

We finally provide a notion of *term* for denoting graphs. We use a multisorted syntax: the family $(\mathcal{T}_k)_{k\in\mathbb{N}}$ of sets of terms of a given arity is defined inductively by the following rules (where p ranges over permutations of $[1,k]$ and a over letters of arity k):

$$\frac{t,u\in\mathcal{T}_k}{t\parallel u\in\mathcal{T}_k}\qquad\frac{t\in\mathcal{T}_k}{\mathrm{l}t\in\mathcal{T}_{k+1}}\qquad\frac{t\in\mathcal{T}_{k+1}}{\mathrm{f}t\in\mathcal{T}_k}\qquad\frac{t\in\mathcal{T}_k}{pt\in\mathcal{T}_k}\qquad\frac{}{\emptyset\in\mathcal{T}_k}\qquad\frac{}{a\in\mathcal{T}_k}$$

This syntax matches the operations in Definition 3.1. Accordingly, we obtain a recursive arity-preserving function g from terms to graphs. We say that a term t *denotes* the graph G, or that t is a *parsing* of G, when $g(t) \simeq G$. For instance, the expressions below the graphs in Figure 1, seen as terms, are parsings of these graphs. Note that the number of inner vertices of a graph is the number of forgets appearing in any of its parsings.

We shall sometimes mention terms and refer implicitly to their graphs; for instance writing that a term is prime to mean that its graph is so.

The *width* of a term is the maximal arity of its subterms, minus one.

▶ **Proposition 4.1.** *A graph has treewidth at most k iff it has a parsing of width at most k.*

Proof. The backward implication is proven by induction on terms, using Proposition 2.4 and the observation that all operations arise as substitutions. For the direct implication, we proceed by lexicographic induction on the size of the graph followed by $k+1$ minus its arity (recall that the arity is always lower or equal to the treewidth plus one).

If the graph has isolated sources, we use Lemma 3.3 to write it as a permutation of lifts and proceed recursively. Otherwise it is full, and we decompose it into primes via Lemma 3.4:

- if there are no components then the graph is empty and has a trivial parsing;
- if there are at least two components then they are smaller, and we proceed recursively;
- otherwise the graph is (full) prime; either it is atomic and it admits a permutation of a letter as a parsing, or, by Proposition 3.7, it can be written as the forget of a graph of the same size but increased arity, which we may parse recursively. ◀

The previous proposition holds for all bounds k on the treewidth. Some of our results below also hold generically, and we will discuss these generalisations in Section 9. Still, from this point on we focus on treewidth at most three.

▶ **Convention 4.2.** *In the remainder, we only work with graphs of treewidth at most three. We simply call* terms *the terms of width at most three, and* forget points *the* 3*-forget points.*

A *(term) substitution* is an arity-preserving function from the alphabet to terms. Such a function σ extends uniquely to a homomorphism $\hat{\sigma}$ from terms to terms:

$$\hat{\sigma}(t \parallel u) \triangleq \hat{\sigma}(t) \parallel \hat{\sigma}(u) \qquad \hat{\sigma}(\mathrm{l}t) \triangleq \mathrm{l}\hat{\sigma}(t)$$
$$\hat{\sigma}(\emptyset) \triangleq \emptyset \qquad \hat{\sigma}(\mathrm{f}t) \triangleq \mathrm{f}\hat{\sigma}(t)$$
$$\hat{\sigma}(a) \triangleq \sigma(a) \qquad \hat{\sigma}(pt) \triangleq p\hat{\sigma}(t)$$

A *context of arity* $i \to o$ is a term of arity o with a single occurrence of a designated letter h of arity i, called the *hole*. Given such a context c and a term t of arity i, we write $c[t]$ for the term c where the hole is replaced by t. Note that $c[t] = \hat{\sigma}_t(c)$ for the substitution σ_t mapping h to t and fixing all other letters.

An *equational theory* is an equivalence relation R on terms, relating only terms of the same arity, and which is closed under contexts and substitutions:

- $(t,u) \in R$ entails $(c[t], c[u]) \in R$ for all contexts c of appropriate arity, and
- $(t,u) \in R$ entails $(\hat{\sigma}(t), \hat{\sigma}(u)) \in R$ for all substitutions σ.

Given two terms t, u, we write $t \cong u$ when $g(t) \simeq g(u)$. Thanks to Proposition 2.2, this relation ($\cong$) is an equational theory, and our goal is to provide a finite list of axioms that generates it.

We give such a list below; most axioms can be written explicitly, but three of them relate rather large terms, which are best presented by their graphs. This is why we rely on the following concept of forget axiom.

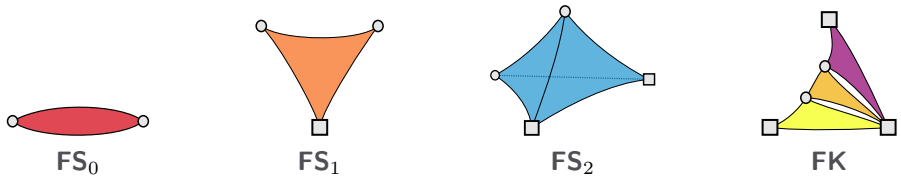


Figure 3 Forget axioms for anchors (swap and kite); x, y are the inner vertices.

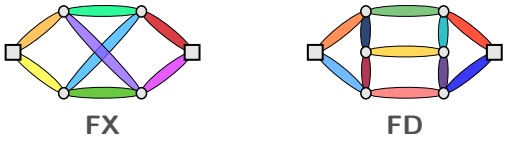


Figure 4 Forget axioms for hard graphs (cross and domino); x, y are the topmost inner vertices.

A *forget axiom* for a graph G with two forget points x, y is an equation of the shape $\mathrm{f}t \equiv \mathrm{f}u$ for some parsings t of (G, x) and u of (G, y).

▶ **Definition 4.3** (Finite axiomatisation). *We write $\equiv$ for the least equational theory containing the following axioms, for letters a, b, c of appropriate arities:*
A1. $a \parallel (b \parallel c) \equiv (a \parallel b) \parallel c$, $a \parallel b \equiv b \parallel a$, *and* $a \parallel \emptyset \equiv a$;
A2. $pqa \equiv (p \circ q)a$ *for all permutations* p, q, *and* $\mathrm{id}a \equiv a$ *where* id *is the identity permutation;*
A3. $p(a \parallel b) \equiv pa \parallel pb$ *and* $p\emptyset \equiv \emptyset$ *for all permutations* p;
A4. $\mathrm{l}(a \parallel b) \equiv \mathrm{l}a \parallel \mathrm{l}b$ *and* $\mathrm{l}\emptyset \equiv \emptyset$;
A5. $p\mathrm{f}a \equiv \mathrm{f}\dot{p}a$ *and* $\mathrm{l}pa \equiv \dot{p}\mathrm{l}a$ *where* $\dot{p}$ *is the extension of a permutation* p *of* $[1, k]$ *to* $[1, k+1]$;
A6. $\mathrm{l}\mathrm{f}a \equiv \mathrm{f}r\mathrm{l}a$ *and* $\mathrm{l}\mathrm{l}a \equiv r\mathrm{l}\mathrm{l}a$ *for the permutation* r *that swaps the last two elements;*
A7. $\mathrm{f}a \parallel b \equiv \mathrm{f}(a \parallel \mathrm{l}b)$;
FS. $\mathrm{f}\mathrm{f}a \equiv \mathrm{f}\mathrm{f}ra$ *for the permutation* r *that swaps the last two elements;*
as well as three forget axioms for the graphs with forget points **FK**, **FX** *and* **FD** *in Figures 3&4 (for some arbitrary choice of interface ordering, edge orientation, and injective edge labelling).*

The first eight items are universally quantified over all appropriate arities. For instance, associativity of parallel composition (in **A1**) is an axiom at each arity, and a, b and $\emptyset$ may have arity up to three in **A4**. Since we restrict globally to arities up to four, the list of axioms is nevertheless finite.

Also note that even though the axioms are expressed using letters, they yield laws which hold under all term substitutions since $\equiv$ is defined as an equational theory.

FS comprises three axioms depending on the arity of a (2, 3, or 4), which we shall sometimes refer to as $\mathbf{FS}_i$ (with $i = 0, 1$, or 2, respectively). These are forget axioms for the first three graphs in Figure 3, and we could have presented them as such. We cannot express $\mathbf{FS}_3$ as it would require an edge of arity five; **FK** intuitively is a weakened version of it, expressible at treewidth three.

We shall see at the end of Section 5 that the concrete choice of parsings for the forget axioms **FK**, **FX**, and **FD** is irrelevant thanks to **A1**-**7**. The interested reader may find concrete equations for them in [10, Appendix C].

▶ **Proposition 4.4** (Soundness). *If $t \equiv u$ then $t \cong u$.*

Proof. Since $\cong$ is an equational theory, it suffices to check that the two members of each axiom denote the same graph. This is by definition for the forget axioms, and a routine verification for the other ones. ◀

The axioms given textually in Definition 4.3 (**A1-7** and **FS**) intuitively correspond to those given for arbitrary graphs by Courcelle and Engelfriet [7, p.117], restricted to terms of width at most three. (Modulo some translation since our syntaxes and sets of operations differ.) This finite restriction is however incomplete for graphs of treewidth at most three: in their completeness proof, Courcelle and Engelfriet need axioms of the form $\mathbf{FS}_i$ for arbitrary large arity i, even when dealing with graphs of small treewidth. Somehow, we prove below that the forget axioms $\mathbf{FS}_0$, $\mathbf{FS}_1$, $\mathbf{FS}_2$, **FK**, **FX**, and **FD** suffice at treewidth at most three to fulfil the role played by this infinite family of axioms. Also observe that while **FK** easily generalises to deal with an arbitrary treewidth k (by replacing the source touching the three edges by $k-2$ sources), this is not the case for **FX** and **FD**.

5 Outline of the completeness proof

Using the first seven axioms, we easily obtain the following proposition. Together with Lemmas 3.3 and 3.4, it makes it possible to normalise terms and to focus on full prime ones.

▶ **Proposition 5.1.** *For all terms t, letters a, and graphs G, H, we have:*
1. *if $g(t) \simeq \emptyset$ then $t \equiv \emptyset$;*
2. *if $g(t) \simeq a$ then $t \equiv a$;*
3. *if $g(t) \simeq pG$ then there is a parsing u of G such that $t \equiv pu$;*
4. *if $g(t) \simeq \mathrm{l}G$ then there is a parsing u of G such that $t \equiv \mathrm{l}u$;*
5. *if $g(t) \simeq G \parallel H$ then there are parsings u of G and v of H such that $t \equiv u \parallel v$.*

Proof sketch. All items but the third are proven by induction on t. The first three items only need **A1-4**; the fourth one needs **A5-6**; the last one rests on the fourth one and **A7**. ◀

An item is patently missing from the previous proposition, for the forget operation. In fact, a statement similar to the third and fourth items does not hold for the forget operation: when t is a parsing of $\mathrm{f}G$, nothing guarantees that G has a parsing at all: its treewidth may be four; we need to restrict to those cases where the forgotten vertex is a forget point.

Given a term t denoting a graph G with an inner vertex x, we say that t *reaches* x if there is a parsing t' of (G, x) such that $t \equiv \mathrm{f}t'$. We can easily prove the following property.

▶ **Lemma 5.2.** *Every non-atomic full prime term reaches some forget point.*

Proof. By induction on t, using Proposition 5.1 until we find a forget operation. ◀

However, this property is not enough, because it gives no control on the forget point which is reached; instead, we would like to obtain:

▶ **Statement 5.3** (Reaching forget points)**.** *Full prime terms reach all their forget points.*

This statement holds, but we do not know how to prove it directly: we only get it *a posteriori*, from the completeness. We prove a variant of it in the next section (Lemma 7.1), for *anchors*.

Instead, the cornerstone of our proof is the following lemma: any two parsings of a non-atomic full prime graph may reach a common forget point.

▶ **Lemma 5.4** (Forget point agreement)**.** *For all parsings t, u of a non-atomic full prime graph, there is a forget point reached by both t and u.*

The rest of the paper is devoted to proving this lemma, with which we conclude as follows.

▶ **Theorem 5.5** (Completeness)**.** *For all parsings t, u of a given graph, we have $t \equiv u$.*

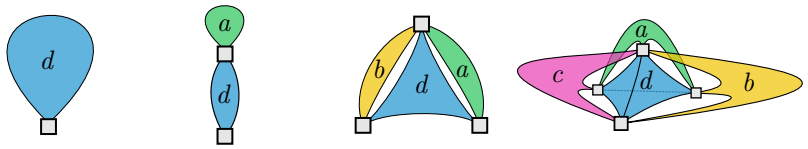


Figure 5 The series operations $\mathrm{s}(;d)$, $\mathrm{s}(a;d)$, $\mathrm{s}(a,b;d)$, and $\mathrm{s}(a,b,c;d)$.

Proof. We follow the same pattern as in the proof of the forward implication of Proposition 4.1: we proceed by lexicographic induction on the size of the graph followed by 4 minus its arity. If the graph has isolated sources, we use Lemma 3.3 and Proposition 5.1(3,4) to rewrite the parsings into permuted lifts and proceed recursively. Otherwise it is full, and we decompose it into primes via Lemma 3.4:

- if there are no components then both parsings are equivalent to $\emptyset$ by Proposition 5.1(1);
- if there are at least two components then we use Proposition 5.1(5) on both parsings and we proceed recursively.
- otherwise the graph must be (full) prime. Either it is atomic and we conclude by Proposition 5.1(2,3), or Lemma 5.4 gives us two terms t', u' such that $t \equiv \mathrm{f}(t')$, $t' \cong u'$, and $\mathrm{f}(u') \equiv u$, and we can conclude since $t' \equiv u'$ follows by induction hypothesis. ◀

We have only used Axioms **A1**-**7** up to this point. We prove below that those axioms are complete when there are few forget points. Say that a graph is *easy* if each of its non-atomic reduced components G has only one forget point x, and in turn (G, x) is easy. (This definition is well-founded by the same lexicographic ordering as we used in the above proof.)

▶ **Proposition 5.6.** *For all parsings t, u of an easy graph, we have $t \equiv u$.*

Proof. By adapting the previous completeness proof. Lemma 5.2 may replace Lemma 5.4 when we attain non-atomic full prime graphs: those are reduced components of the starting graph and since those have only one forget point, all their parsings reach that one. ◀

None of the graphs in Figures 3 and 4 are easy: they are all full prime, each with two forget points x, y (plus two symmetrical ones for **FX** and **FD**). Nevertheless, adding either x or y to their interface makes them easy (cf. [10, Lemma D.1]). This is why the precise choice of parsings does not matter when we use them as forget axioms in Definition 4.3.

6 Series decompositions

As explained in the introduction, there is a *series* operation on graphs which plays an important role. We slightly generalise it here, and we show how to use it to analyse graphs with a forget point.

Given k graphs $G_1, \dots, G_k$ of arity k and a graph H of arity $k+1$, we define the following graph of arity $k+1$, where p_i denotes the permutation which swaps i and $k+1$.

$$\mathrm{s}(G_1, \dots, G_k; H) \triangleq p_1 \mathrm{l} G_1 \parallel \cdots \parallel p_k \mathrm{l} G_k \parallel H$$

As is explicit from its definition, this operation on graphs is also a derived operation on terms. We illustrate its behaviour at each arity in Figure 5. We recover the operation from the introduction when the last argument is empty, and using a forget operation: we have $\circ(u, v, w) = \mathrm{fs}(u, v, w; \emptyset)$.

We use this operation in order to decompose full prime graphs along a given inner vertex.

▶ **Proposition 6.1** (Series decomposition). *For all full prime graphs G of arity k with an inner vertex x, there are graphs $G_1, \dots, G_k, H$ such that $(G, x) \simeq \mathrm{s}(G_1, \dots, G_k; H)$ and*
1. *all components of H are full, and*
2. *if the jth source of a component of G_i is isolated, then $i < j$.*

Such a decomposition is unique up to isomorphism.

We call G_i the *ith series argument*, and H the *series factor* of such a decomposition (at x).

Proof. We decompose (G, x) into prime components, which we classify according to their isolated sources. Since G is full prime, x is never isolated. Full components go into the series factor. Components where the first source is isolated go into the first series argument (under the permutation p_1, the first source gets swapped with x). Components where the second source is isolated but not the first one go into the second series argument. *Et caetera.* ◀

Also note that we can follow such decompositions at the term level, modulo **A1**-**7**:

▶ **Proposition 6.2.** *If a term t reaches the last source of a graph of the form $\mathrm{s}(G_1, \dots, G_k; H)$, then there are parsings $u_1, \dots, u_k, v$ of $G_1, \dots, G_k, H$ such that $t \equiv \mathrm{fs}(u_1, \dots, u_k; v)$.*

Proof. Consequence of Proposition 5.1. ◀

7 Anchors

In this section we define the concept of *anchors* – inner vertices with specific properties, and we prove the following variant of Statement 5.3:

▶ **Lemma 7.1** (Reaching anchors). *Full prime terms reach all their anchors.*

This lemma implies Lemma 5.4 when the considered graph has an anchor, by applying it to both parsings. Therefore, once Lemma 7.1 proved, it will only remain to prove Lemma 5.4 for anchor-free graphs (Section 8).

The definition of anchor is the following; we only consider them in full prime graphs.

▶ **Definition 7.2.** *An* anchor *in a (full prime) graph G is an inner vertex x such that either:*
1. *there are no full prime components in (G, x), or*
2. *there is an edge whose neighbours comprise at least x and all sources of G, or*
3. *there are two or more full prime components in (G, x).*

Using series decompositions, x is an anchor whenever the series factor at x is empty, or has only one edge, or has at least two components.

This last condition generalises the usual notion of cutvertex [8, Section 1.4]: such anchors at arity zero when there are only binary edges are exactly cutvertices as usual. Removing them disconnects the graph. The second branch in our definition may only arise with sources and/or hyperedges; it is convenient to have it for the present proof, but it would probably be more natural to disallow it in other contexts. Thanks to this alternative, all inner vertices in the graphs in Figure 3 are anchors. In contrast, the graphs in Figure 4 have no anchors.

We prove below that all forget points are anchors at arity three. The converse holds at all arities, but we only get it *a posteriori*, as a consequence of Lemma 7.1.

▶ **Proposition 7.3.** *At arity three, all forget points are anchors.*

Proof. If x is a forget point of a graph G of arity three, the full prime components of (G, x) must be atomic by Proposition 3.5. Thus x is an anchor, by either the first or the second condition in Definition 7.2. ◀

We also find anchors easily at arities zero and one.

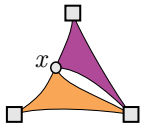
▶ **Proposition 7.4.** *Non-atomic full prime graphs of arity in* $\{0, 1, 3\}$ *have some anchor.*

Proof. Every inner vertex is an anchor at arity zero, either because it is isolated (anchor of the first kind), or because it belongs to some edge (anchor of the second kind). Every immediate neighbour of the only source is an anchor at arity one (of the second kind). At arity three we use Propositions 7.3 and 3.7. ◀

We first prove Lemma 7.1 for graphs of arity three, and then we show how to reduce the other cases to that one (Section 7.2).

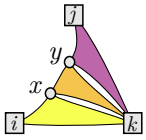
7.1 Parsing on anchors at arity three

A *checkpoint* between two vertices y, z of a graph G is an inner vertex x such that every inner path from y to z goes through x. When y, z are not mentioned explicitly, they are implicitly assumed to be sources. (This definition coincides with that from [5] at arity two.) Analysing the components of (G, x) by the sources they touch, we see that the checkpoints of a graph G of arity three are the inner vertices x for which G has the following shape:



This characterisation shows that checkpoints are anchors of the first kind in Definition 7.2. Another important property at arity three is that if there are two anchors, then they must be checkpoints for the same sources:

▶ **Proposition 7.5.** *At arity three, every full prime graph with two distinct anchors* x, y *has the following shape:*



Proof of Lemma 7.1 at arity three. We proceed by induction on the size of the graph.

Let t be a parsing of a full prime graph G of arity three, with an anchor x. By Lemma 5.2, t reaches some forget point y. If $x = y$ then we are done. Otherwise, y is a second anchor by Proposition 7.3, and G has the shape given by Proposition 7.5. Call A the graph between i,k,y, and B the one between j, k, y. Since t reaches y we have a parsing t' of (G, y) such that $t \equiv \mathrm{f}t'$. By analysing the full prime decomposition of (G, y), we can put t' under the form $(j4)\mathrm{l}u \parallel (i4)\mathrm{l}v$, where u is a parsing of A and v a parsing of B. Then we observe that x is a checkpoint in A, and thus an anchor, so that u reaches x by induction hypothesis. Therefore, $u \equiv \mathrm{f}u'$ for some parsing u' of (A, x). By analysing the full prime decomposition of (A, x) as before, we deduce that t reaches x using **FK**. ◀

7.2 Parsing on anchors at lower arities

We now finish the proof of Lemma 7.1 by showing how to reduce the other cases to that of arity three. We use for that the two following lemmas. The first one intuitively makes it possible to zoom on a specific inner vertex by going under a forget operation. The second one states that under some conditions, an anchor in $\mathrm{f}^i G$ is also an anchor in G, allowing to use the case of arity three.

▶ **Lemma 7.6.** *Let t be a term of arity up to two with an inner vertex x in a full prime component. There is a term u such that $t \equiv \mathrm{f}u$ and x is either the last source of u or an inner vertex of a full prime component of u.*

▶ **Lemma 7.7.** *For $i > 0$ let $\mathrm{f}^i G$ be a full prime graph of arity k and x, y two distinct inner vertices in a full prime component C of G. If x is an anchor of $\mathrm{f}^i G$ and is in one of the first k series arguments of the series decomposition of C at y, then x is a checkpoint in C.*

(Note that the only isolated sources of the components of the i last series arguments in the decomposition of C as above, are the i forgotten sources in $\mathrm{f}^i G$.)

Before we finish the proof of Lemma 7.1, also observe that **FS** implies $\mathrm{f}^j pu \equiv \mathrm{f}^j qu$ for all terms u of arity k and permutations p, q agreeing on the first $k - j$ elements.

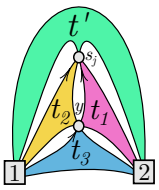
Proof of Lemma 7.1, at all arities. We prove the following generalisation, by induction on the lexicographic product of $|C|$ and $3 - k$:

> For all terms t of arity $k \leq 3$, for all $i \leq k$, if $\mathrm{f}^i t$ is full prime with an anchor x in a full prime component C of t, then $\mathrm{f}^i t$ reaches x.

If $k < 3$ then we use Lemma 7.6 to obtain u such that $\mathrm{f}^i t \equiv \mathrm{f}^{i+1} u$; if x is the last source of u then we are done, otherwise we conclude by induction, since x lies in a component of C.

We now assume $k = 3$, and we rewrite t as $t_C \parallel t'$, where t_C is the full prime component containing x. By Lemma 5.2, t_C reaches some forget point y. Take a series decomposition $t_C \equiv \mathrm{fs}(t_1, t_2, t_3; u)$ of t_C at y. The series factor u cannot contain x since all its components are atomic by Proposition 3.5. Thus x must be in a series argument.

If x is in one of the first $3 - i$ series arguments, then by Lemma 7.7 x must be a checkpoint, and thus an anchor of t_C. Using Lemma 7.1 at arity three (Section 7.1), we obtain that t_C reaches x. So does t by **A7**, and finally $\mathrm{f}^i t$ by **FS**. Otherwise, x is in one of the last i series arguments, say the jth one. The jth source s_j is the only source that is necessarily isolated in this argument. It is a source that is forgotten by the operations f^i (see the figure below for an illustration). Our goal is to "swap" this jth source for y in order to decrease the size of C. Indeed the new component containing x will then be contained in t_3, which is smaller in size than C.



Let S be the set of non-isolated sources in the component C' of $s(t_1, t_2, t_3; u)$ containing x. Using **FS** we get a term t'' such that $\mathrm{f}^i t \equiv \mathrm{f}^{|S|} t''$ and t'' denotes $g(t)$ with the inner vertices in S upgraded as sources. By definition of t'', C' is a full prime component of it. Furthermore, up to isolated sources, C' is the same component as the one of $s(t_1, t_2, t_3; u)$ containing x. This gives $|C'| < |C|$ and we conclude by induction on $\mathrm{f}^{|S|} t''$. ◀

At this point, we have used axioms **A1**-**7**, **FS**, and **FK**, but not **FX** and **FD**.

Figure 6 Separation pairs $(y \diamond x)$, and partial order on vertices $(y \prec x)$.

8 Separation pairs

We call *hard* the non-atomic anchor-free full prime graphs. Examples of such graphs were given in Figure 2. Hard graphs have arity two by Propositions 7.4 and 3.5, and it only remains to prove Lemma 5.4 for these graphs.

A *forget pair* in a graph G is a pair (x, y) of inner vertices such that (G, x, y) has treewidth at most three. A parsing t of a graph G *reaches a pair* (x, y) of inner vertices if there is some parsing t' of (G, x, y) such that $t \equiv \mathrm{fft}'$. By definition, a term may only reach forget pairs. By **FS**, a term reaches (x, y) iff it reaches (y, x); it follows that every term reaching (x, y) reaches both x and y.

Given two inner vertices x, y of a full prime graph of arity two, we write $y \diamond x$ when the graph has the shape on the left of Figure 6, and $y \prec x$ when the graph has the shape on the right (equivalently, when y is a checkpoint between x and some source). In the first case, we say that (x, y) is a *separation pair*. We prove a few properties about such inner vertices.

▶ **Lemma 8.1.** *If $y \prec x$ or $y \diamond x$ in a full prime graph G, then y is an anchor in (G, x), and every parsing of G reaching x also reaches (x, y).*

Proof. In both cases, there is no full prime component in (G, x, y), so that y is indeed an anchor in (G, x). Now if $t \equiv \mathrm{f}t'$ for some parsing t' of (G, x), then t' reaches y by Lemma 7.1: $t' \equiv \mathrm{f}t''$ for some parsing t'' of (G, x, y). Thus t reaches (x, y). ◀

It follows that a full prime term reaches a separation pair iff it reaches any of its constituents.

We write $S(x)$ for the series factor of a graph at some inner vertex x. When the graph is hard, x is not an anchor, so that $S(x)$ must be full prime and cannot contain just one edge.

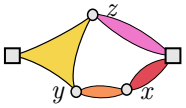
▶ **Proposition 8.2.** *For all hard graphs, $\prec$ is well-founded.*

Proof. Observe that if $y \prec x$ then $|S(y)| < |S(x)|$. ◀

▶ **Proposition 8.3.** *For all vertices x, y, z such that $y \prec x$ and $x \diamond z$, we have $y \diamond z$.*

Proof. Consider an inner path between the two sources; we have to show that it visits either y or z. It visits either x or z since (x, z) is a separation pair. If it visits x then it also visits y since y is a checkpoint between x and one of the two sources. ◀

When the graph is hard in the previous proposition, it must have the following shape:

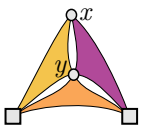


In such a case, we intuitively prefer the separation pair (y, z) to (x, z). This leads us to the notion of minimal separation pair. A vertex x is *minimal* if there is no vertex y such that $y \prec x$. A pair of vertices is minimal when its two elements are so.

The proposition below makes it possible to get separation pairs out of minimal vertices.

▶ **Proposition 8.4.** *Let x be a forget point in a hard graph. For all forget points y of $S(x)$, we have either $y \prec x$ or $y \diamond x$.*

Proof. Let us classify the components of (G, x, y) by their isolated sources. Full components cannot exist as by Proposition 3.5 they would be edges and both x and y would be anchors in G. So G must have the following shape:



If there is no inner path avoiding y between the sources in the bottom component, then $y \diamond x$. If there is no inner path avoiding y from x to one of the sources, then $y \prec x$. Otherwise there are at least two full prime components in (G, y), making y an anchor of G, which contradicts G being hard. ◀

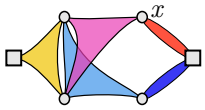
It follows easily that every hard graph has some separation pair – even a minimal one, which will be useful to reduce the number of cases to study. However, we need to be more precise and to keep track of terms.

▶ **Lemma 8.5.** *Every hard term reaches some minimal separation pair.*

Proof. Let t be a hard term. By Lemma 5.2 t reaches some forget point x, which we can choose minimal by Proposition 8.2 and Lemma 8.1. Accordingly, let t' be a parsing of (G, x) such that $t \equiv \mathrm{f}t'$. As $S(x)$ is full prime and non-atomic, it has a forget point y. Since x is minimal, (x, y) is a separation pair by Proposition 8.4. We can choose y to be minimal by Proposition 8.3, and t reaches (x, y) by Lemma 8.1. ◀

As before with forget points, separation pairs are not unique (even minimal ones), and we need to show how to move from one separation pair to another.

When studying the possible shapes of a hard graph H with distinct separation forget pairs, we end up with a few shapes S such as the following one:



In such situations, we know that (H, x) has treewidth at most three, but (S, x) has treewidth four. Therefore, for an appropriate notion of *minor*, (S, x) cannot be a minor of (H, x): the treewidth may not increase when taking minors. We use this information in order to refine the actual shape of H.

We use *sourced simple graphs* in order to define such a notion of minor. Those are triples (V, E, S) made of a set V of *vertices*, a set E of unordered binary *edges*, and a subset $S \subseteq V$ of *sources*. We write K_k for the clique over k sources. *Tree decompositions* and *treewidth* [8, Section 12.4] are adapted to sourced simple graphs by requiring the subset of sources to be contained in some bag. A *sourced minor of a (sourced simple) graph* is a (sourced simple) graph obtained from it by a sequence of the following operations: remove an edge, contract an edge, remove an isolated vertex. Those operations do not increase the treewidth.

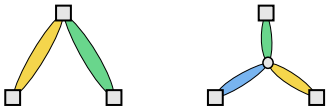
The *footprint* of a graph is the sourced simple graph obtained from it by replacing each edge by a clique over its neighbours, forgetting labels, and turning the interface into a mere set – note that we do not add a clique on the sources. A graph and its footprint have the same tree decompositions, and thus the same treewidth. A *sourced minor of a graph* is a sourced minor of its footprint.

Note that graphs excluding K_3 as a sourced minor need not be acyclic, as cycles may occur away from the sources. The point of using sourced minors is that they behave well with respect to substitutions, and thus shapes:

▶ **Proposition 8.6.** *Given a graph G and a substitution σ, if K_k is a sourced minor of $\sigma(a)$ for all letters a of arity k, then the footprint of G is a sourced minor of $G\sigma$.*

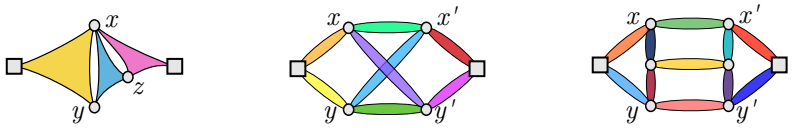
Now observe that full prime graphs of arity two admit the sourced edge K_2 as a sourced minor. At arity three, we have the following property instead.

▶ **Proposition 8.7.** *Every graph of arity three either has the sourced triangle K_3 as a sourced minor, or has one of the two following shapes:*



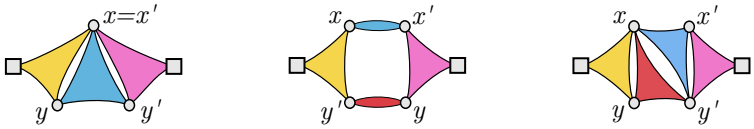
Let us continue our example and come back to the previous hard graph H. Assume that the binary edges in the given shape are instantiated by full prime graphs, and consider the three graphs instantiating the ternary edges. If those three graphs have the triangle as a sourced minor, then the footprint of (S, x) must be a sourced minor of (H, x), which is not possible. Therefore, at least one of them must have one of the shapes given by Proposition 8.7. By considering the different possibilities and continuing the same kind of reasoning, we end up proving that in this case, H must have shape **FX**. In the general case, we obtain the following classification.

▶ **Proposition 8.8.** *Let G be a hard graph with two minimal separation forget pairs (x, y) and (x', y'). Possibly up to swapping x and y, and/or x' and y', either (x, y') is a separation pair (**Case Z**), or G has one of the following shapes, where in the leftmost case, z is minimal and x', y' appear in the rightmost component:*



Case IH **Case X** **Case D**

The proof we provide in [10, Appendix F] actually establishes that in the case where (x, y') is a separation pair, then G has one of the following shapes:



Case Z1 **Case Z2** **Case Z3**

We finally deduce that hard terms reach all their minimal separation forget pairs.

▶ **Lemma 8.9.** *Let G be a hard graph with minimal separation forget pair (x', y'). All parsings t of G reaching a minimal separation pair (x, y) also reach (x', y').*

Proof. We fix G, x', y', we proceed by induction on the size of the component of (G, x, y) containing x', and we use Proposition 8.8. If (x, y') is a separation pair (**Case Z**), then we use Proposition 8.1 twice: t reaches x, thus (x, y'), thus y' in particular, thus (x', y'). In

Case IH we proceed by induction on (x, z), which is reached by t by similar reasoning, and for which the induction measure decreases. In **Case X** and **Case D**, we conclude directly by **FX** and **FD**, respectively. (In those latter two cases, we first need to rewrite t so that it agrees syntactically with the shape of the axiom we use. We use Lemma 7.1 for that, which requires showing that some of the inner vertices are anchors in appropriate subgraphs. To do so, we exploit the fact that G is hard, so that the subgraphs corresponding to the edges of the exhibited shape must be connected enough.) ◀

Together with Lemma 8.5, Lemma 5.4 follows, which concludes our completeness proof.

9 Conclusion and future work

We have provided a finite presentation of graphs of treewidth at most three. Our axiomatisation comprises axioms for dealing with full prime decompositions (**A1**-**7**), axioms for reaching anchors (**FS**, **FK**), and two axioms for hard graphs (**FX** and **FD**).

An obvious question for future work is whether the approach presented here generalises to the case of graphs of treewidth at most k.

It does so up to Section 7. The syntax we use readily characterises those graphs (Proposition 4.1), and the seven first axioms as well as the results in Section 5 are not specific to the case $k = 3$. Axioms for anchors (Figure 3) also generalise. Accordingly, our results about series decomposition and anchors (Lemma 7.1) extend easily. In particular, non-atomic full prime graphs of arity $0, 1$ and k would always have an anchor.

This means we also have finite presentations for bounds $k = 1, 2$. (For $k = 2$ this axiomatisation differs from the one previously proposed [5, 9], because the chosen syntax is different: there, graphs of arity three are not considered, and parallel composition may be applied to graphs of distinct arities.)

However, the results from Section 8 are specific to the case $k = 3$, and getting finite presentations for larger values of k seems difficult. Following the strategy presented here, we would need to prove the forget point agreement lemma (Lemma 5.4) for new hard graphs, which may have arities between 2 and $k - 1$.

Still, we would like to emphasise the utility of such a generalisation in the context of graph theory. Robertson and Seymour proved [14] that for all integers k there must be a finite list of excluded minors characterising the class of treewidth at most k graphs. Consider a minimal one amongst them, say H, along with a maximal clique over vertices $x_1, \ldots, x_i$. It is not difficult to prove that either H is K_{k+2}, or $(H, x_1, \ldots, x_i)$ is hard. Thus, studying hard graphs of treewidth $k + 1$ might lead to a better understanding of excluded minors for treewidth at most k.

References

1 Stefan Arnborg, Bruno Courcelle, Andrzej Proskurowski, and Detlef Seese. An algebraic theory of graph reduction. *J. ACM*, 40(5):1134–1164, 1993. doi:10.1145/174147.169807.

2 Stefan Arnborg and Andrzej Proskurowski. Characterization and recognition of partial 3-trees. *SIAM Journal on Algebraic Discrete Methods*, 7(2):305–314, 1986. doi:10.1137/0607033.

3 Umberto Bertelè and Francesco Brioschi. On non-serial dynamic programming. *J. Comb. Theory, Ser. A*, 14(2):137–148, 1973. doi:10.1016/0097-3165(73)90016-2.

4 Hans L. Bodlaender. Treewidth: Algorithmic techniques and results. In Igor Prívara and Peter Ruzicka, editors, *Proc. MFCS*, volume 1295 of *LNCS*, pages 19–36. Springer, 1997. doi:10.1007/BFB0029946.

5 Enric Cosme-Llópez and Damien Pous. K4-free graphs as a free algebra. In Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *Proc. MFCS*, volume 83 of *LIPIcs*, pages 76:1–76:14. Schloss Dagstuhl, 2017. doi:10.4230/LIPICS.MFCS.2017.76.

6 Bruno Courcelle. Graph grammars, monadic second-order logic and the theory of graph minors. In Neil Robertson and Paul D. Seymour, editors, *Proc. Graph Structure Theory*, volume 147 of *Contemporary Mathematics*, pages 565–590. American Mathematical Society, 1991.

7 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012. URL: http://www.cambridge.org/fr/knowledge/isbn/item5758776/?site_locale=fr_FR.

8 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

9 Christian Doczkal and Damien Pous. Treewidth-two graphs as a free algebra. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *Proc. MFCS*, volume 117 of *LIPIcs*, pages 60:1–60:15. Schloss Dagstuhl, 2018. doi:10.4230/LIPICS.MFCS.2018.60.

10 Amina Doumane, Samuel Humeau, and Damien Pous. A finite presentation of graphs of treewidth at most three, 2024. Version of this paper with the appendix. URL: https://hal.science/hal-04560570.

11 Rudolf Halin. S-functions for graphs. *Journal of geometry*, 8:171–186, 1976. doi:10.1007/BF01917434.

12 Damien Pous. Hypergraphs. Software, swhId: swh:1:dir:028863b2f75dde258591611a6c7c165e289db890 (visited on 2024-06-17). URL: https://perso.ens-lyon.fr/damien.pous/hypergraph/.

13 Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

14 Neil Robertson and Paul D. Seymour. Graph minors. XX. wagner's conjecture. *J. Comb. Theory, Ser. B*, 92(2):325–357, 2004. doi:10.1016/J.JCTB.2004.08.001.

15 Daniel P. Sanders. *Linear algorithms for graphs of tree-width at most four*. PhD thesis, Georgia Institute of Technology, 1993. URL: http://hdl.handle.net/1853/30061.

16 Petra Scheffler. Linear-time algorithms for NP-complete problems restricted to partial k-trees. Technical report, Academy of Sciences of the GDR, 1987.

17 Thomas V. Wimer. *Linear Algorithms on k-Terminal Graphs*. PhD thesis, Clemson University, 1993. URL: https://tigerprints.clemson.edu/arv_dissertations/28.

Improved Algorithm for Reachability in d-VASS

Yuxi Fu
BASICS, Shanghai Jiao Tong University, China

Qizhe Yang[1]
Shanghai Normal University, China

Yangluo Zheng
BASICS, Shanghai Jiao Tong University, China

Abstract

An F_d upper bound for the reachability problem in vector addition systems with states (VASS) in fixed dimension is given, where F_d is the d-th level of the Grzegorczyk hierarchy of complexity classes. The new algorithm combines the idea of the linear path scheme characterization of the reachability in the 2-dimension VASSes with the general decomposition algorithm by Mayr, Kosaraju and Lambert. The result improves the F_{d+4} upper bound due to Leroux and Schmitz (LICS 2019).

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Logic and verification

Keywords and phrases Petri net, vector addition system, reachability

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.136

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2404.14781`

Funding The support from the National Natural Science Foundation of China (62072299) is acknowledged.

Acknowledgements We thank Weijun Chen, Huan Long, Hao Wu and Qiang Yin for proofreading various versions of this paper.

1 Introduction

Petri nets, or equivalently vector addition system with states (VASS), are a well studied model of concurrency. A VASS consists of a finite state control where each state transition has as its effect an integer-valued vector, and its configurations are pairs of a state and a vector with *natural number* components. A transition may lead from one configuration to another by adding its effect component-wise, conditioned on that the components of the resulting vector remain non-negative. The *reachability problem*, which asks whether from one configuration there is a path reaching another configuration, lies in the center of the algorithmic theory of Petri nets and has found a wide range of applications due to its generic nature. Since the problem was shown to be decidable by Mayr [17] in 1981, its computational complexity had been a long-standing open problem in the field. In 2015, Leroux and Schmitz [14] presented the first complexity upper bound, stating that the reachability problem is cubic-Ackermannian. This was later improved to an Ackermannian upper bound in 2019, again by Leroux and Schmitz [15]. Regarding the hardness, in 2021 seminal works by Czerwiński and Orlikowski [6], and independently by Leroux [13], provided matching Ackermannian lower bounds, settling the exact complexity of the problem.

[1] corresponding author

Concerning the parameterization by dimension, i.e. the reachability problem in d-dimensional VASSes where d is fixed, there is still a gap in the known complexity bounds. Currently, we only have exact complexity bounds for dimension one and two [7, 1]. For dimension $d \geq 3$, the result of Leroux and Schmitz [15] shows that the problem is in F_{d+4}, the $(d{+}4)$-th level of the Grzegorczyk hierarchy of complexity classes. Note that a recent work by Yang and Fu [22] points out that the problem for dimension 3 is in $\mathsf{F}_3 = \mathsf{TOWER}$. On the other hand, the best known lower bound by Czerwiński, Jecker, Lasota, Leroux and Orlikowski [5] states that reachability in $(2d{+}3)$-dimensional VASSes is F_d-hard. Motivated by this gap, our paper focuses on the computational complexity of reachability problem in the fixed-dimensional VASSes.

Our contribution

In this paper we show that the reachability problem in the d-dimensional VASS is in F_d for $d \geq 3$, improving the previous F_{d+4} upper bound by Leroux and Schmitz [15], and generalizing the tower upper bound for the reachability problem in 3-VASS [22]. The new upper bound is obtained with the help of two novel technical lemmas.

1. Our main technical tool (Theorem 3.4) is a generalization of the linear path scheme characterization for the reachability relation in the 2-dimensional VASSes [1]. By borrowing the key idea from the work of Yang and Fu [22], we show that as long as the "geometric dimension" of a VASS (that is, the dimension of the vector space spanned by the effects of cyclic paths) is bounded by 2, its reachability relation can be characterized by short linear path schemes. We then apply the lemma to simplify the KLMST algorithm so that (i) a VASS is replaced by a short linear path scheme whenever its geometric dimension is no more than 2 and (ii) the linear path schemes will not be decomposed further. It is then routine [15], using the tools from [21], to show that the reachability problem in the d-dimensional VASS is in F_{d+1} for all $d \geq 3$.
2. Our second lemma (Lemma 6.3) allows us to improve further the bound from F_{d+1} to F_d. This is done by a careful analysis of the properties of the fast-growing functions [21].

Due to space limitation the proofs of the two lemmas are placed in the appendices.

Organization

Section 2 fixes notation, defines the VASS model and its reachability problem. Section 3 generalizes the linear path scheme characterization [1] to VASSes whose geometric dimension are bounded by 2. Section 4 recalls the characterization system of linear inequalities for linear path schemes. Section 5 makes use of the results of Section 3 to give an improved version of the classic KLMST decomposition algorithm. Section 6 analyzes the complexity of our modified algorithm, proving the main result. Section 7 concludes. Proofs omitted from the main text can be found in the appendices.

2 Preliminaries

We use $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ to denote respectively the set of non-negative integers, integers, and rational numbers. Let $n \in \mathbb{N}$ be a number, we write $[n]$ for the range $\{1, 2, \ldots, n\}$. Let $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{X}^d$ be d-dimensional vectors where $\mathbb{X}$ can be any one of $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$. We write $\boldsymbol{v}(i)$ for the i-th component of $\boldsymbol{v}$ where $i \in [d]$, so $\boldsymbol{v} = (\boldsymbol{v}(1), \ldots, \boldsymbol{v}(d))$. The *maximum norm* of $\boldsymbol{v}$ is defined to be $\|\boldsymbol{v}\| := \max_{i \in [d]} |\boldsymbol{v}(i)|$. We extend component-wise the order $\leq$ for vectors in $\mathbb{X}^d$, so $\boldsymbol{u} \leq \boldsymbol{v}$ if

and only if $\boldsymbol{u}(i) \leq \boldsymbol{v}(i)$ for all $i \in [d]$. Addition and subtraction of vectors are also component-wise, so $(\boldsymbol{u}+\boldsymbol{v})(i) = \boldsymbol{u}(i) + \boldsymbol{v}(i)$ for all $i \in [d]$. Define $\mathrm{supp}(\boldsymbol{v}) := \{i \in [d] : \boldsymbol{v}(i) \neq 0\}$ to be the set of indices of non-zero components of $\boldsymbol{v}$. This notation is extended to sets of vectors naturally, so $\mathrm{supp}(S) = \bigcup_{\boldsymbol{v}\in S} \mathrm{supp}(\boldsymbol{v})$ for any set $S \subseteq \mathbb{X}^d$.

For technical reasons we introduce the symbol ω that stands for the infinite element. Let $\mathbb{N}_\omega := \mathbb{N} \cup \{\omega\}$. We stipulate that $n < \omega$ for all $n \in \mathbb{N}$, and $x + \omega = \omega + x = \omega$ for all $x \in \mathbb{Z}$. Define the partial order $\sqsubseteq$ over $\mathbb{N}_\omega$ so that $x \sqsubseteq y$ if and only if $x = y$ or $y = \omega$ for all $x, y \in \mathbb{N}_\omega$. The relation $\sqsubseteq$ is also extended component-wise to vectors in $\mathbb{N}_\omega^d$.

Let Σ be a finite alphabet and $s, t \in \Sigma^*$ be two strings over Σ. We write st for the concatenation of s and t, and s^n for the concatenation of n copies of s where $n \in \mathbb{N}$. If $s = a_1 a_2 \dots a_\ell$ where $a_1, \dots, a_\ell \in \Sigma$, we write $|s| := \ell$ for the length of s, and $s[i \dots j] := a_i a_{i+1} \dots a_j$ for the substring of s between indices i and j where $1 \leq i \leq j \leq |s|$.

2.1 Vector Addition Systems with States

Let $d \geq 0$ be an integer. A *d-dimensional vector addition system with states (d-VASS)* is a pair $G = (Q, T)$ where Q is a finite set of *states* and $T \subseteq Q \times \mathbb{Z}^d \times Q$ is a finite set of *transitions*. Clearly a VASS can also be viewed as a directed graph with edges labelled by integer vectors. Given a word $\pi = (p_1, \boldsymbol{a}_1, q_1)(p_2, \boldsymbol{a}_2, q_2) \dots (p_n, \boldsymbol{a}_n, q_n) \in T^*$ over transitions, we say that π is a *path from p_1 to q_n* if $q_i = p_{i+1}$ for all $i = 1, \dots, n-1$. It is a *cycle* if we further have $p_1 = q_n$. The *effect* of π is defined to be $\Delta(\pi) := \sum_{i=1}^n \boldsymbol{a}_i$, and the *action word* of π is the word $[\![\pi]\!] := \boldsymbol{a}_1 \boldsymbol{a}_2 \dots \boldsymbol{a}_n$ over $\mathbb{Z}^d$. The *Parikh image* of π is the function $\phi \in \mathbb{N}^T$ mapping each transition to its number of occurrences in π. Given a function $\phi \in \mathbb{N}^T$ we also define $\Delta(\phi) := \sum_{t=(p,\boldsymbol{a},q)\in T} \phi(t) \cdot \boldsymbol{a}$. Note that $\Delta(\phi) = \Delta(\pi)$ if ϕ is the Parikh image of π. Let $L \subseteq T^*$ be a language (i.e. subset of words), we define its effect as $\Delta(L) := \{\Delta(\pi) : \pi \in L\}$.

The norm of a transition $t = (p, \boldsymbol{a}, q)$ is defined by $\|t\| := \|\boldsymbol{a}\|$. The norm of a path $\pi = t_1 t_2 \dots t_n$ is $\|\pi\| := \max_{i\in[n]} \|t_i\|$. For a d-VASS $G = (Q, T)$ we write $\|T\| := \max\{\|t\| : t \in T\}$. The *size* of G is defined by

$$|G| := |Q| + |T| + d \cdot |T| \cdot \|T\| \,. \tag{1}$$

Semantics of VASSes

Let $G = (Q, T)$ be a d-VASS. A *configuration* of G is a pair of a state $p \in Q$ and a vector $\boldsymbol{v} \in \mathbb{Z}^d$, written as $p(\boldsymbol{v})$. Let $\mathbb{D} \subseteq \mathbb{Z}^d$, we define the $\mathbb{D}$-semantics for G as follows. For each transition $t = (p, \boldsymbol{a}, q) \in T$, the one-step transition relation $\xrightarrow{t}_{\mathbb{D}}$ relates all pairs of configurations of the form $(p(\boldsymbol{u}), q(\boldsymbol{v}))$ where $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{D}$ and $\boldsymbol{v} = \boldsymbol{u} + \boldsymbol{a}$. Then for a word $\pi = t_1 t_2 \dots t_n \in T^*$, the relation $\xrightarrow{\pi}_{\mathbb{D}}$ is the composition $\xrightarrow{\pi}_{\mathbb{D}} := \xrightarrow{t_1}_{\mathbb{D}} \circ \cdots \circ \xrightarrow{t_n}_{\mathbb{D}}$. So $p(\boldsymbol{u}) \xrightarrow{\pi}_{\mathbb{D}} q(\boldsymbol{v})$ if and only if there are configurations $p_0(\boldsymbol{u}_0), \dots, p_n(\boldsymbol{u}_n) \in Q \times \mathbb{D}$ such that

$$p(\boldsymbol{u}) = p_0(\boldsymbol{u}_0) \xrightarrow{t_1}_{\mathbb{D}} p_1(\boldsymbol{u}_1) \xrightarrow{t_2}_{\mathbb{D}} \cdots \xrightarrow{t_n}_{\mathbb{D}} p_n(\boldsymbol{u}_n) = q(\boldsymbol{v}). \tag{2}$$

Also, when $\pi = \epsilon$ is the empty word, the relation $\xrightarrow{\epsilon}_{\mathbb{D}}$ is the identity relation over $Q \times \mathbb{D}$. Note that $\xrightarrow{\pi}_{\mathbb{D}}$ is non-empty only if π is a path. When $p(\boldsymbol{u}) \xrightarrow{\pi}_{\mathbb{D}} q(\boldsymbol{v})$ we also say that π induces (or is) a $\mathbb{D}$-*run* from $p(\boldsymbol{u})$ to $q(\boldsymbol{v})$. We emphasize that all configurations on this run lie in $\mathbb{D}$, and that they are uniquely determined by $p(\boldsymbol{u})$ and π. For a language $L \subseteq T^*$ we define $\xrightarrow{L}_{\mathbb{D}}$ as $\bigcup_{\pi\in L} \xrightarrow{\pi}_{\mathbb{D}}$. Finally, the $\mathbb{D}$-*reachability relation* of G is defined to be $\xrightarrow{*}_{\mathbb{D}} := \xrightarrow{T^*}_{\mathbb{D}}$.

For the above definitions, we shall often omit the subscript $\mathbb{D}$ when $\mathbb{D} = \mathbb{N}^d$.

We mention that in Section 5 we need to generalize the VASS semantics to configurations in $Q \times \mathbb{N}_\omega^d$, allowing ω components in vectors. The definitions of $\xrightarrow{t}_{\mathbb{N}_\omega^d}$, $\xrightarrow{\pi}_{\mathbb{N}_\omega^d}$, and $\xrightarrow{*}_{\mathbb{N}_\omega^d}$ are similar to the above.

Reachability problem

The reachability problem in vector addition systems with states is formulated as follows:

Reachability in d-Dimensional Vector Addition System with States
Input: A d-dimensional VASS $G = (Q, T)$, two configurations $p(\boldsymbol{u}), q(\boldsymbol{v}) \in Q \times \mathbb{N}^d$.
Question: Does $p(\boldsymbol{u}) \xrightarrow{*}_{\mathbb{N}^d} q(\boldsymbol{v})$ hold in G?

Note that we study the reachability problem for fixed-dimensional VASSes, where the dimension d is treated as a constant to allow more fine-grained analysis. So we shall use the big-O notation to hide constants that may depend on d. The general problem where the dimension can be part of the input was already shown to be Ackermann-complete [6, 13].

Cycle spaces and geometric dimensions

One of the key insights of [15] is a new termination argument for the KLMST decomposition algorithm based on the dimensions of vector spaces spanned by cycles in VASSes, which yielded the primitive recursive upper bound of VASS reachability problem in fixed dimensions. The vector spaces spanned by cycles also play an important role in our work.

▶ **Definition 2.1.** *Let G be a d-VASS. The* cycle space *of G is the vector space* $\mathrm{Cyc}(G) \subseteq \mathbb{Q}^d$ *spanned by the effects of all cycles in G, that is:*

$$\mathrm{Cyc}(G) := \mathrm{span}\{\Delta(\beta) : \beta \text{ is a cycle in } G\}. \tag{3}$$

The dimension of the cycle space of G is called the geometric dimension *of G. We say G is* geometrically k-dimensional *if* $\dim(\mathrm{Cyc}(G)) \leq k$.

3 Flattability of Geometrically 2-dimensional VASSes

A VASS is *flat* if each of its states lies on at most one cycle. Flat VASSes form an important subclass of VASSes due to its connection to Presburger arithmetic, and we refer the readers to [12] for a survey. In dimension 2, it was proved in [1] that 2-VASSes enjoy a stronger form of flat representation, known as the *linear path schemes*. A linear path scheme is a regular expression of the form $\alpha_0\beta_1^*\alpha_1 \dots \beta_k^*\alpha_k$ where $\alpha_0, \dots \alpha_k$ are paths and $\beta_1, \dots, \beta_k$ are cycles of the VASS, such that they form a path when joined together. The results of [1] show that the reachability relation of every 2-VASS can be captured by short linear path schemes.

Linear path schemes are extremely useful as they can be fully characterized by linear inequality systems so that standard tools for linear or integer programming can be applied. In this section, we generalize the results in [1] and show that the reachability relation of any d-VASS whose geometric dimension is bounded by 2 can also be captured by short linear path schemes.

Our proof follows closely to the lines of [1]. Given a geometrically 2-dimensional VASS G, we first cover $\mathbb{N}^d$ by the following two regions: one for the region far away from every axis:

$$\mathbb{O} := \{\boldsymbol{u} \in \mathbb{N}^d : \boldsymbol{u}(i) \geq D \text{ for all } i \in [d]\} \tag{4}$$

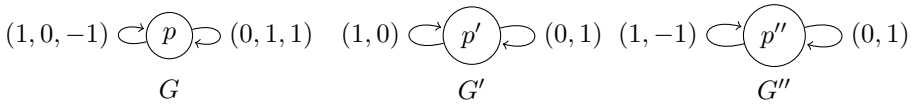


Figure 1 A geometrically 2-dimensional VASS G and two possible projections of it.

where D is some properly chosen threshold; the other one for the region close to some axis:

$$\mathbb{L} := \{\boldsymbol{u} \in \mathbb{N}^d : \boldsymbol{u}(i) \leq D' \text{ for some } i \in [d]\} \tag{5}$$

where D' is chosen slightly above D to create an overlap with $\mathbb{O}$. Let π be a run that we are going to capture by linear path schemes. We can extract its maximal prefix that lies completely in either $\mathbb{O}$ or $\mathbb{L}$, depending on where π starts. This prefix must end at a configuration that lies in $\mathbb{L} \cap \mathbb{O}$, if we haven't touched the end of π. From this configuration we then extract a maximal cycle that also ends in $\mathbb{L} \cap \mathbb{O}$. Continuing this fashion, we can break π into segments of runs that lie completely in $\mathbb{O}$ or $\mathbb{L}$, interleaved by cycles that start and end in $\mathbb{O}$ (actually in $\mathbb{L} \cap \mathbb{O}$). Note that the number of such cycles cannot exceed the number of states of G since they are maximal. Now we only need to capture the following three types of runs by short linear path schemes:

1. Runs that are cycles starting and ending in $\mathbb{O}$.
   This will be handled in Section A.4 in the appendix. Briefly speaking, since the geometric dimension of G is 2, the effect of such a cycle must belong to a plane in $\mathbb{Z}^d$. We will find a clever way to project this plane to a coordinate plane, and then project the d-VASS G onto this plane to get a 2-VASS. This is made possible by a novel technique called the "sign-reflecting projection" developed in Section A.3. Intuitively speaking, for any vector in a plane we are able to determine whether it belongs to a certain orthant by observing only 2 entries of this vector. The d-VASS can then be projected onto these 2 coordinates. (See Example 3.1 for a more concrete demonstration.) Now we apply the results of [1] to obtain a linear path scheme that captures the projected cyclic run. Combined with a lemma in [16], the projection guarantees that we can safely project it back to get a linear path scheme for the run in G.
2. Runs that lie completely in $\mathbb{O}$.
   This is just an easy corollary of the first type, and will also be handled in Section A.4. Just note that any run can be broken into a series of simple paths interleaved by cycles.
3. Runs that lie completely in $\mathbb{L}$.
   This will be handled in Section A.5, by a long and tedious case analysis. In principle, we are going to argue that any such run essentially corresponds to a run in some $(d-1)$-VASS, so that we can use induction.

▶ **Example 3.1.** Consider the geometrically 2-dimensional 3-VASS G as shown in Figure 1. It consists of a single state p and two transitions with effects $(1, 0, -1)$ and $(0, 1, 1)$. In order to apply the results of [1], one would like to derive a 2-VASS from G that reflects runs in G. A simple idea is to discard one coordinate of G. Two possibilities of this idea are shown in Figure 1 as G', which discards the third coordinate, and G'', which discards the second coordinate. However, not all of them are satisfactory. For example, the legal run $p(0,0) \xrightarrow{(1,0)} p(1,0)$ in G' reflects an illegal run $p(0,0,0) \xrightarrow{(1,0,-1)} p(1,0,-1)$ in G where the third coordinate goes negative. On the other hand, all runs in G'' can be safely projected back to a run in G. To see this, just observe that for any vector $\boldsymbol{v}$ in the linear span of $(1, 0, -1)$ and $(0, 1, 1)$, $\boldsymbol{v} \geq \mathbf{0}$ if and only if $\boldsymbol{v}(1) \geq 0$ and $\boldsymbol{v}(3) \geq 0$. Thus we can safely discard the second coordinate.

In general, the "sign-reflecting projection" developed in Section A.3 shows that any geometrically 2-dimensional VASS can be projected onto two coordinates so that the signs of these two coordinates reflects the signs of other coordinates.

In the rest of this section we just state formally our main technical results. The detailed proofs are placed in the appendix.

3.1 Linear Path Schemes

A *linear path scheme (LPS)* is a pair (G, Λ) where G is a VASS and Λ is a regular expression of the form $\Lambda = \alpha_0\beta_1^*\alpha_1 \dots \beta_k^*\alpha_k$ such that $\alpha_0, \dots, \alpha_k$ are paths in G and $\beta_1, \dots, \beta_k$ are cycles in G, and $\alpha_0\beta_1\alpha_1 \dots \beta_k\alpha_k$ is a path in G. We say an LPS (G, Λ) is *compatible to* a VASS G' if G' contains all states and transitions in Λ. Very often we shall omit the VASS G and say that Λ on its own is an LPS, with G understood as any VASS to which Λ is compatible. We write $|\Lambda| = |\alpha_0\beta_1\alpha_1 \dots \beta_k\alpha_k|$ for the length of Λ, $\|\Lambda\| = \|\alpha_0\beta_1\alpha_1 \dots \beta_k\alpha_k\|$ for its norm, and $|\Lambda|_* = k$ for the number of cycles in Λ.

We also treat Λ as the language defined by it, and thus for two configurations $p(\boldsymbol{u})$ and $q(\boldsymbol{v})$ we write $p(\boldsymbol{u}) \xrightarrow{\Lambda}_{\mathbb{D}} q(\boldsymbol{v})$ if and only if there exists $e_1, \dots, e_k \in \mathbb{N}$ such that

$$p(\boldsymbol{u}) \xrightarrow{\alpha_0\beta_1^{e_1}\alpha_1 \dots \beta_k^{e_k}\alpha_k}_{\mathbb{D}} q(\boldsymbol{v}). \tag{6}$$

Positive linear path schemes

A *positive LPS* is a regular expression of the form $\Lambda^+ = \alpha_0\beta_1^+\alpha_1 \dots \beta_k^+\alpha_k$ which is similar to a linear path scheme except that we require each cycle to be used at least once. We write $p(\boldsymbol{u}) \xrightarrow{\Lambda^+}_{\mathbb{D}} q(\boldsymbol{v})$ if and only if there are positive integers $e_1, \dots, e_k \in \mathbb{N}_{>0}$ such that

$$p(\boldsymbol{u}) \xrightarrow{\alpha_0\beta_1^{e_1}\alpha_1 \dots \beta_k^{e_k}\alpha_k}_{\mathbb{D}} q(\boldsymbol{v}). \tag{7}$$

A path π is said to be *admitted by* Λ^+ if $\pi = \alpha_0\beta_1^{e_1}\alpha_1 \dots \beta_k^{e_k}\alpha_k$ for some $e_1, \dots, e_k \in \mathbb{N}_{>0}$. We prefer positive LPSes as they can be easily characterized by linear inequality systems (see Section 4 for details). In fact, positive LPSes can be easily obtained from LPSes:

▶ **Lemma 3.2.** *For every linear path scheme Λ there exists a finite set S of positive linear path schemes compatible to the same VASSes with Λ, such that $\xrightarrow{\Lambda} \;= \bigcup_{\Lambda^+ \in S} \xrightarrow{\Lambda^+}$ and $|\Lambda^+| \le |\Lambda|$ for every $\Lambda^+ \in S$.*

Proof. Suppose $\Lambda = \alpha_0\beta_1^*\alpha_1 \dots \beta_k^*\alpha_k$. For each cycle component β_i^* in Λ we replace it by either β_i^+ or an empty word nondeterministically. Let S be the set of all resulting positive LPSes. It is obvious that S satisfies the desired properties. ◀

3.2 Main Results

The main results of this section is stated as follows.

▶ **Theorem 3.3.** *Let $G = (Q, T)$ be a geometrically 2-dimensional d-VASS. For every pair of configurations $p(\boldsymbol{u}), q(\boldsymbol{v}) \in Q \times \mathbb{N}^d$ with $p(\boldsymbol{u}) \xrightarrow{*} q(\boldsymbol{v})$ there exists a positive LPS Λ^+ compatible to G such that $p(\boldsymbol{u}) \xrightarrow{\Lambda^+} q(\boldsymbol{v})$ and $|\Lambda^+| \le |G|^{O(1)}$.*

We remark that the big-O term here and elsewhere in the paper hides constant that may depend on the dimension d, but does not depend on $G, \boldsymbol{u}, \boldsymbol{v}$ or anything else.

By Lemma 3.2 we know that positive LPSes can be obtained from LPSes. Thus theorem 3.3 follows easily from the following relaxed theorem, which will be proved in the appendix.

▶ **Theorem 3.4.** *Let $G = (Q, T)$ be a geometrically 2-dimensional d-VASS. For every pair of configurations $p(\boldsymbol{u}), q(\boldsymbol{v}) \in Q \times \mathbb{N}^d$ with $p(\boldsymbol{u}) \xrightarrow{*} q(\boldsymbol{v})$ there exists an LPS Λ compatible to G such that $p(\boldsymbol{u}) \xrightarrow{\Lambda} q(\boldsymbol{v})$ and $|\Lambda| \leq |G|^{O(1)}$.*

4 Characteristic Systems for Linear Path Schemes

The property that linear path schemes can be fully characterized by linear inequality systems is exploited in [2] to derive the PSPACE upper bound of the reachability problem in 2-VASSes. Here we recall this linear inequality system and its properties.

We mainly focus on positive linear path schemes. Fix $\Lambda = \alpha_0\beta_1^+\alpha_1 \dots \beta_k^+\alpha_k$ to be a positive LPS from state p to q that is compatible to some d-VASS $G = (Q, T)$, where $k = |\Lambda|_*$ is the number of cycles in Λ.

▶ **Definition 4.1** (cf. [2, Lem. 14])**.** *The* characteristic system $\mathcal{E}_{\mathrm{LPS}}(\Lambda)$ of the positive LPS Λ *is the system of linear inequalities such that a triple $\boldsymbol{h} = (\boldsymbol{u}, \boldsymbol{e}, \boldsymbol{v}) \in \mathbb{N}^d \times \mathbb{N}^k \times \mathbb{N}^d$ satisfies $\mathcal{E}_{\mathrm{LPS}}(\Lambda)$, written $\boldsymbol{h} \models \mathcal{E}_{\mathrm{LPS}}(\Lambda)$, if and only if the following conditions hold:*

1. *for every $i = 1, \dots, k$, $\boldsymbol{e}(i) \geq 1$;*
2. *for every $i = 0, \dots, k$ and every $j = 1, \dots, |\alpha_i|$,*

$$\boldsymbol{u} + \Delta(\alpha_0\beta_1^{\boldsymbol{e}(1)}\alpha_1 \dots \alpha_{i-1}\beta_i^{\boldsymbol{e}(i)}) + \Delta(\alpha_i[1 \dots j]) \geq \boldsymbol{0}; \tag{8}$$

3. *for every $i = 1, \dots, k$ and every $j = 1, \dots, |\beta_i|$,*

$$\boldsymbol{u} + \Delta(\alpha_0\beta_1^{\boldsymbol{e}(1)}\alpha_1 \dots \beta_{i-1}^{\boldsymbol{e}(i-1)}\alpha_{i-1}) \qquad + \Delta(\beta_i[1 \dots j]) \geq \boldsymbol{0}, \tag{9}$$

$$\boldsymbol{u} + \Delta(\alpha_0\beta_1^{\boldsymbol{e}(1)}\alpha_1 \dots \beta_{i-1}^{\boldsymbol{e}(i-1)}\alpha_{i-1}\beta_i^{\boldsymbol{e}(i)-1}) + \Delta(\beta_i[1 \dots j]) \geq \boldsymbol{0}; \tag{10}$$

4. *and finally, $\boldsymbol{u} + \Delta(\alpha_0\beta_1^{\boldsymbol{e}(1)}\alpha_1 \dots \beta_k^{\boldsymbol{e}(k)}\alpha_k) = \boldsymbol{v}$.*

The readers can easily verify that these constraints are indeed linear in terms of $\boldsymbol{u}, \boldsymbol{e}, \boldsymbol{v}$. The next lemma shows that $\mathcal{E}_{\mathrm{LPS}}(\Lambda)$ indeed captures all runs admitted by Λ.

▶ **Lemma 4.2.** *Let G be a d-VASS and $\Lambda = \alpha_0\beta_1^+\alpha_1 \dots \beta_k^+\alpha_k$ be a positive LPS from state p to q compatible to G. Then for every $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{N}^d$, $p(\boldsymbol{u}) \xrightarrow{\Lambda} q(\boldsymbol{v})$ if and only if there exists $\boldsymbol{e} \in \mathbb{N}^k$ such that $(\boldsymbol{u}, \boldsymbol{e}, \boldsymbol{v}) \models \mathcal{E}_{\mathrm{LPS}}(\Lambda)$. Moreover, for every $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{N}^d$ and every $\boldsymbol{e} \in \mathbb{N}^k$ such that $(\boldsymbol{u}, \boldsymbol{e}, \boldsymbol{v}) \models \mathcal{E}_{\mathrm{LPS}}(\Lambda)$, we have $p(\boldsymbol{u}) \xrightarrow{\alpha_0\beta_1^{\boldsymbol{e}(1)}\alpha_1 \dots \beta_k^{\boldsymbol{e}(k)}\alpha_k} q(\boldsymbol{v})$.*

We also need to introduce the homogeneous version of $\mathcal{E}_{\mathrm{LPS}}(\Lambda)$ for technical reasons.

▶ **Definition 4.3.** *The* homogeneous characteristic system $\mathcal{E}^0_{\mathrm{LPS}}(\Lambda)$ of Λ *is the system of linear inequalities such that a triple $\boldsymbol{h}_0 = (\boldsymbol{u}_0, \boldsymbol{e}_0, \boldsymbol{v}_0) \in \mathbb{N}^d \times \mathbb{N}^k \times \mathbb{N}^d$ satisfies $\mathcal{E}^0_{\mathrm{LPS}}(\Lambda)$, written $\boldsymbol{h}_0 \models \mathcal{E}^0_{\mathrm{LPS}}(\Lambda)$, if and only if the following conditions hold:*

1. *for every $i = 0, \dots, k$, $\boldsymbol{u}_0 + \Delta(\beta_1) \cdot \boldsymbol{e}_0(1) + \dots + \Delta(\beta_i) \cdot \boldsymbol{e}_0(i) \geq \boldsymbol{0}$;*
2. *$\boldsymbol{u}_0 + \Delta(\beta_1) \cdot \boldsymbol{e}_0(1) + \dots + \Delta(\beta_k) \cdot \boldsymbol{e}_0(k) = \boldsymbol{v}_0$.*

5 The Modified KLMST Decomposition Algorithm

In this section we apply our results of Section 3 to improve the notoriously hard KLMST decomposition algorithm for VASS reachability. Our narration will base on the work of Leroux and Schmitz [15]. For readers familiar with [15], the major modifications are listed below:

- The decomposition structure is now a sequence of generalized VASS reachability instances linked by (positive) linear path schemes rather than by single transitions.
- We introduce a new "cleaning" step that replaces all VASS instances which are geometrically 2-dimensional by polynomial-length linear path schemes compatible to them.
- We do not guarantee the exact preservation of action languages at each decomposition step. Instead, we only preserve a subset of action languages. This is a compromise since linear path schemes capture only the reachability relation but not every possible run. Nonetheless, it is enough for the reachability problem.

In this section we focus on the effectiveness and correctness of the modified KLMST decomposition algorithm. Its complexity will be analyzed in Section 6.

5.1 Linear KLM Sequences

The underlying decomposition structure in the KLMST algorithm was known as KLM sequences, named after Mayr[17], Kosaraju[10], and Lambert[11].

▶ **Definition 5.1.** *A* KLM tuple *of dimension d is a tuple $\langle p(\boldsymbol{x})Gq(\boldsymbol{y})\rangle$ where $G = (Q, T)$ is a d-VASS and $p(\boldsymbol{x}), q(\boldsymbol{y}) \in Q \times \mathbb{N}_\omega^d$ are two (generalized) configurations of G. A* KLM sequence *of dimension d is a sequence of KLM tuples interleaved by transitions of the form*

$$\xi = \langle p_0(\boldsymbol{x}_0)G_0q_0(\boldsymbol{y}_0)\rangle\, t_1\, \langle p_1(\boldsymbol{x}_1)G_1q_1(\boldsymbol{y}_1)\rangle\, t_2 \dots t_k\, \langle p_k(\boldsymbol{x}_k)G_kq_k(\boldsymbol{y}_k)\rangle\,, \tag{11}$$

where each tuple $\langle p_i(\boldsymbol{x}_i)G_iq_i(\boldsymbol{y}_i)\rangle$ is a KLM tuple of dimension d and each t_i is a transition of the form $(q_{i-1}, \boldsymbol{a}_i, p_i)$ from state q_{i-1} to p_i with effect $\boldsymbol{a}_i \in \mathbb{Z}^d$.

In this paper we generalize the definition of KLM sequences to allow (positive) linear path schemes to connect KLM tuples.

▶ **Definition 5.2.** *A* linear KLM sequence *of dimension d is a sequence*

$$\xi = \langle p_0(\boldsymbol{x}_0)G_0q_0(\boldsymbol{y}_0)\rangle\, \Lambda_1\, \langle p_1(\boldsymbol{x}_1)G_1q_1(\boldsymbol{y}_1)\rangle\, \Lambda_2 \dots \Lambda_k\, \langle p_k(\boldsymbol{x}_k)G_kq_k(\boldsymbol{y}_k)\rangle\,, \tag{12}$$

where each tuple $\langle p_i(\boldsymbol{x}_i)G_iq_i(\boldsymbol{y}_i)\rangle$ is a KLM tuple of dimension d and each Λ_i is a positive linear path scheme from state q_{i-1} to p_i.

One immediately sees that KLM sequences are just special cases of linear KLM sequences. Let ξ be a linear KLM sequence given as (12), we write $\xi_i := \langle p_i(\boldsymbol{x}_i)G_iq_i(\boldsymbol{y}_i)\rangle$ for the ith KLM tuple occurring in ξ.

Action languages

Let ξ be a linear KLM sequence given as (12). We say a path π from state p_0 to q_k is *admitted by* ξ, written $\xi \vdash \pi$, if π can be written as $\pi = \pi_0\rho_1\pi_1 \dots \rho_k\pi_k$ where π_i is a path from p_i to q_i in G_i for each $i = 0, \dots, k$, and ρ_i is a path admitted by Λ_i for each $i = 1, \dots, k$, such that there exist vectors $\boldsymbol{m}_0, \boldsymbol{n}_0, \dots, \boldsymbol{m}_k, \boldsymbol{n}_k \in \mathbb{N}^d$ such that

$$p_0(\boldsymbol{m}_0) \xrightarrow{\pi_0} q_0(\boldsymbol{n}_0) \xrightarrow{\rho_1} p_1(\boldsymbol{m}_1) \xrightarrow{\pi_1} q_1(\boldsymbol{n}_1) \xrightarrow{\rho_2} \cdots \xrightarrow{\rho_k} p_k(\boldsymbol{m}_k) \xrightarrow{\pi_k} q_k(\boldsymbol{n}_k) \tag{13}$$

and that $\boldsymbol{m}_i \sqsubseteq \boldsymbol{x}_i$, $\boldsymbol{n}_i \sqsubseteq \boldsymbol{y}_i$ for each $i = 0, \dots, k$.

The *action language* L_ξ of ξ is the language over $\mathbb{Z}^d$ defined by $L_\xi := \{[\![\pi]\!] : \xi \vdash \pi\}$, where we recall that $[\![\cdot]\!]$ is the word morphism mapping each transition to its effect.

We are more interested in the action languages because in some decomposition steps we have to modify the set of transitions, and only the action word of admitted runs can be preserved. Notice that action languages preserve not only the effects of admitted runs, but also their lengths.

Ranks and Sizes

Let t be a transition in a d-VASS G, we define $\mathrm{Cyc}(G/t)$ to be the vector space spanned by the effects of all cycles in G containing t. For the VASS G, let r_i be the number of transitions t in G such that $\dim(\mathrm{Cyc}(G/t)) = i$ for each $i = 0, \ldots, d$. Then the *rank* of G is defined as $\mathrm{rank}(G) = (r_d, \ldots, r_3) \in \mathbb{N}^{d-2}$. We also define the *full rank* of G as $\mathrm{rank}_{\mathrm{full}}(G) = (r_d, \ldots, r_0) \in \mathbb{N}^{d+1}$.

The following lemma was proved in [15], which shows that in a strongly connected VASS G, the space $\mathrm{Cyc}(G/t)$ corresponds to $\mathrm{Cyc}(G)$.

▶ **Lemma 5.3** ([15, Lem. 3.2]). *Let t be a transition of a strongly connected VASS G. Then* $\mathrm{Cyc}(G/t) = \mathrm{Cyc}(G)$.

The following corollary is immediate.

▶ **Corollary 5.4.** *Let G be a strongly connected d-VASS. Then* $\mathrm{rank}(G) = \mathbf{0}$ *if and only if G is geometrically 2-dimensional.*

Let ξ be a linear KLM sequence given as (12). We define the *rank* of ξ as $\mathrm{rank}(\xi) = \sum_{i=0}^{k} \mathrm{rank}(G_i)$, and the *full rank* of ξ as $\mathrm{rank}_{\mathrm{full}}(\xi) = \sum_{i=0}^{k} \mathrm{rank}_{\mathrm{full}}(G_i)$. We remark that the full rank corresponds to the rank defined in [15]. Ranks are ordered lexicographically: let $\boldsymbol{r} = (r_d, \ldots, r_0)$ and $\boldsymbol{r}' = (r'_d, \ldots, r'_0)$, we write $\boldsymbol{r} \leq_{\mathrm{lex}} \boldsymbol{r}'$ if $\boldsymbol{r} = \boldsymbol{r}'$ or the maximal i with $r_i \neq r'_i$ satisfies $r_i < r'_i$.

Recall that for a VASS G we write $|G|$ for its size as defined in (1). For a linear path scheme Λ, its length $|\Lambda|$ and norm $\|\Lambda\|$ are defined in Section 3.1. Let $\zeta = \langle p(\boldsymbol{x})Gq(\boldsymbol{y})\rangle$ be a KLM tuple of dimension d, its *size* is defined to be $|\zeta| = |G| + d \cdot (\|\boldsymbol{x}\| + \|\boldsymbol{y}\| + 1)$. Let ξ be a linear KLM sequence given as (12), we define its *size* as

$$|\xi| = \sum_{i=0}^{k} |\xi_i| + \sum_{i=1}^{k} d \cdot |\Lambda_i| \cdot (\|\Lambda_i\| + 1). \tag{14}$$

Note that the sizes defined in this paper reflect the sizes of unary encoding, thus have an exponential expansion in their binary encoding.

5.2 Characteristic Systems for Linear KLM Sequences

We define in this section the characteristic systems of linear KLM sequences, which are systems of linear inequalities that serve as an under-specification of admitted runs. Let $G = (Q, T)$ be a VASS, we first recall the *Kirchhoff system* $K_{G,p,q}$ of G with respect to states $p, q \in Q$, which is a system of linear equations such that a function $\phi \in \mathbb{N}^T$ is a model of $K_{G,p,q}$, written $\phi \models K_{G,p,q}$, if and only if

$$\mathbf{1}_q - \mathbf{1}_p = \sum_{t=(r,\boldsymbol{a},s)\in T} \phi(t) \cdot (\mathbf{1}_s - \mathbf{1}_r), \tag{15}$$

where $\mathbf{1}_p \in \{0,1\}^Q$ is the indicator function defined by $\mathbf{1}_p(q) = 1$ if $q = p$ and $\mathbf{1}_p(q) = 0$ otherwise. We also need the homogeneous version of $K_{G,p,q}$, denoted by $K^0_{G,p,q}$, where a function $\phi \in \mathbb{N}^T$ is a model of it, written $\phi \models K^0_{G,p,q}$, if and only if

$$\mathbf{0} = \sum_{t=(r,\boldsymbol{a},s)\in T} \phi(t) \cdot (\mathbf{1}_s - \mathbf{1}_r). \tag{16}$$

▶ **Definition 5.5.** *Let ξ be a linear KLM sequence given by*

$$\xi = \langle p_0(\boldsymbol{x}_0)G_0q_0(\boldsymbol{y}_0)\rangle\, \Lambda_1\, \langle p_1(\boldsymbol{x}_1)G_1q_1(\boldsymbol{y}_1)\rangle\, \Lambda_2 \ldots \Lambda_k\, \langle p_k(\boldsymbol{x}_k)G_kq_k(\boldsymbol{y}_k)\rangle \tag{17}$$

The characteristic system $\mathcal{E}(\xi)$ *is a set of linear (in)equalities such that a sequence*

$$\boldsymbol{h} = (\boldsymbol{m}_0, \phi_0, \boldsymbol{n}_0), \boldsymbol{e}_1, (\boldsymbol{m}_1, \phi_1, \boldsymbol{n}_1), \boldsymbol{e}_2, \ldots, \boldsymbol{e}_k, (\boldsymbol{m}_k, \phi_k, \boldsymbol{n}_k), \tag{18}$$

where each $(\boldsymbol{m}_i, \phi_i, \boldsymbol{n}_i) \in \mathbb{N}^d \times \mathbb{N}^{T_i} \times \mathbb{N}^d$ and each $\boldsymbol{e}_i \in \mathbb{N}^{|\Lambda_i|_*}$, is a model of $\mathcal{E}(\xi)$, written $\boldsymbol{h} \models \mathcal{E}(\xi)$, if and only if

1. *$\boldsymbol{m}_i \sqsubseteq \boldsymbol{x}_i$, $\phi_i \models K_{G,p,q}$, $\boldsymbol{n}_i \sqsubseteq \boldsymbol{y}_i$ and $\boldsymbol{n}_i = \boldsymbol{m}_i + \Delta(\phi_i)$ for every $i = 0, \ldots, k$;*
2. *$(\boldsymbol{n}_{i-1}, \boldsymbol{e}_i, \boldsymbol{m}_i) \models \mathcal{E}_{\mathrm{LPS}}(\Lambda_i)$ for every $i = 1, \ldots, k$.*

Similarly, the homogeneous characteristic system $\mathcal{E}^0(\xi)$ *is a set of linear (in)equalities such that a sequence $\boldsymbol{h}$ of the form (18) is a model of $\mathcal{E}^0(\xi)$, written $\boldsymbol{h} \models \mathcal{E}^0(\xi)$, if and only if*

1. *$\boldsymbol{m}_i(j) = 0$ whenever $\boldsymbol{x}_i(j) \neq \omega$, $\phi_i \models K^0_{G,p,q}$, $\boldsymbol{n}_i(j) = 0$ whenever $\boldsymbol{y}_i(j) \neq \omega$, and $\boldsymbol{n}_i = \boldsymbol{m}_i + \Delta(\phi_i)$ for every $i = 0, \ldots, k$;*
2. *$(\boldsymbol{n}_{i-1}, \boldsymbol{e}_i, \boldsymbol{m}_i) \models \mathcal{E}^0_{\mathrm{LPS}}(\Lambda_i)$ for every $i = 1, \ldots, k$.*

The sequence ξ is said to be satisfiable *if $\mathcal{E}(\xi)$ has a model, otherwise it's unsatisfiable.*

Let $\boldsymbol{h}$ be a model of $\mathcal{E}(\xi)$ (or $\mathcal{E}^0(\xi)$), we shall write $\boldsymbol{m}_i^{\boldsymbol{h}}, \phi_i^{\boldsymbol{h}}, \boldsymbol{n}_i^{\boldsymbol{h}}, \boldsymbol{e}_i^{\boldsymbol{h}}$ for the values of $\boldsymbol{m}_i, \phi_i, \boldsymbol{n}_i, \boldsymbol{e}_i$ assigned by $\boldsymbol{h}$, respectively. Recall that unsatisfiable linear KLM sequences have empty action languages.

▶ **Lemma 5.6** (cf. [15, Lem. 3.5])**.** *For any unsatisfiable linear KLM sequence ξ, $L_\xi = \emptyset$.*

5.2.1 Bounds on Bounded Values in $\mathcal{E}(\xi)$

We state here a lemma similar to [15, Lem. 3.7], which upper bounds the bounded values in the characteristic system $\mathcal{E}(\xi)$. Its proof can be found in the appendix, which is a straightforward application of tools in [3] and [18].

▶ **Lemma 5.7.** *Assume that $\xi = \langle p_0(\boldsymbol{x}_0)G_0q_0(\boldsymbol{y}_0)\rangle\, \Lambda_1 \cdots \Lambda_k\, \langle p_k(\boldsymbol{x}_k)G_kq_k(\boldsymbol{y}_k)\rangle$ is satisfiable. Then for every $0 \leq i \leq k$ we have:*

- *For every $1 \leq j \leq d$, the set of values $\boldsymbol{m}_i^{\boldsymbol{h}}(j)$ where $\boldsymbol{h} \models \mathcal{E}(\xi)$ is unbounded if, and only if, there exists a model $\boldsymbol{h}_0$ of $\mathcal{E}^0(\xi)$ such that $\boldsymbol{m}_i^{\boldsymbol{h}_0}(j) > 0$.*
- *For every $t \in T_i$, the set of values $\phi_i^{\boldsymbol{h}}(t)$ where $\boldsymbol{h} \models \mathcal{E}(\xi)$ is unbounded if, and only if, there exists a model $\boldsymbol{h}_0$ of $\mathcal{E}^0(\xi)$ such that $\phi_i^{\boldsymbol{h}}(t) > 0$.*
- *For every $1 \leq j \leq d$, the set of values $\boldsymbol{n}_i^{\boldsymbol{h}}(j)$ where $\boldsymbol{h} \models \mathcal{E}(\xi)$ is unbounded if, and only if, there exists a model $\boldsymbol{h}_0$ of $\mathcal{E}^0(\xi)$ such that $\boldsymbol{n}_i^{\boldsymbol{h}_0}(j) > 0$.*

Moreover, every bounded value of $\mathcal{E}(\xi)$ is bounded by $(10|\xi|)^{12|\xi|}$.

5.3 Cleaning of Linear KLM Sequences

In this section we define three conditions that require a linear KLM sequence to be *strongly connected*, *pure*, and *saturated*. Together with the satisfiability condition, they make up the so-called "clean" condition of linear KLM sequences. Note that the purity condition is new compared to [15], which requires every geometrically 2-dimensional VASSes occur in a linear KLM sequence to be replaced by linear path schemes.

Strongly Connected Sequences

A linear KLM sequence $\xi = \langle p_0(\boldsymbol{x}_0)G_0q_0(\boldsymbol{y}_0)\rangle\,\Lambda_1\cdots\Lambda_k\,\langle p_k(\boldsymbol{x}_k)G_kq_k(\boldsymbol{y}_k)\rangle$ is *strongly connected* if all the VASSes $G_0,\dots,G_k$ are strongly connected (as they are understood as directed graphs). One can easily obtain strongly connected sequences by expanding the strongly connected components of each VASS:

▶ **Lemma 5.8** ([15, Lem. 4.2]). *For any linear KLM sequence ξ which is not strongly connected, one can compute in time $\mathsf{exp}(|\xi|)$ a finite set Ξ of strongly connected linear KLM sequences such that $L_\xi = \bigcup_{\xi'\in\Xi} L_{\xi'}$ and that $\mathrm{rank}(\xi') \leq_{lex} \mathrm{rank}(\xi)$ and $|\xi'| \leq (2d+1)|\xi|$ for every $\xi' \in \Xi$.*

Pure Sequences

A KLM tuple $\langle p(\boldsymbol{x})Gq(\boldsymbol{y})\rangle$ is called *trivial* if $p(\boldsymbol{x}) = q(\boldsymbol{y})$ and G contains no transition and only a single state p. In this case we simply write $\langle p(\boldsymbol{x})\rangle$ for this tuple. Note that the action language of a trivial tuple contains exactly the empty word.

A linear KLM sequence $\xi = \langle p_0(\boldsymbol{x}_0)G_0q_0(\boldsymbol{y}_0)\rangle\,\Lambda_1\cdots\Lambda_k\,\langle p_k(\boldsymbol{x}_k)G_kq_k(\boldsymbol{y}_k)\rangle$ is said to be *pure* if ξ is strongly connected and for every $i = 0,\dots,k$, $\mathrm{rank}(G_i) = \mathbf{0}$ implies that the tuple $\langle p_i(\boldsymbol{x}_i)G_iq_i(\boldsymbol{y}_i)\rangle$ is trivial. By Corollary 5.4, a rank-$\mathbf{0}$ strongly connected VASS is geometrically 2-dimensional, and thus can be replaced by linear path schemes in case it is not trivial.

▶ **Lemma 5.9.** *Let ξ be a strongly connected linear KLM sequence. Whether ξ is pure is in* PSPACE. *If ξ is not pure, one can compute in space $\mathsf{poly}(|\xi|)$ a finite set Ξ of pure linear KLM sequences such that $\bigcup_{\xi'\in\Xi} L_{\xi'} \subseteq L_\xi$ and $\bigcup_{\xi'\in\Xi} L_{\xi'} \neq \emptyset$ whenever $L_\xi \neq \emptyset$, and such that $\mathrm{rank}(\xi') = \mathrm{rank}(\xi)$ and $|\xi'| \leq |\xi|^{O(1)}$ for all $\xi' \in \Xi$.*

Saturated Sequences

Let $\xi = \langle p_0(\boldsymbol{x}_0)G_0q_0(\boldsymbol{y}_0)\rangle\,\Lambda_1\cdots\Lambda_k\,\langle p_k(\boldsymbol{x}_k)G_kq_k(\boldsymbol{y}_k)\rangle$ be a linear KLM sequence. We say ξ is *saturated* if for every $0 \leq i \leq k$ and every $j \in [d]$, we have

- $\boldsymbol{x}_i(j) = \omega$ implies the set of values $\boldsymbol{m}_i^{\boldsymbol{h}}(j)$ where $\boldsymbol{h} \models \mathcal{E}(\xi)$ is unbounded; and
- $\boldsymbol{y}_i(j) = \omega$ implies the set of values $\boldsymbol{n}_i^{\boldsymbol{h}}(j)$ where $\boldsymbol{h} \models \mathcal{E}(\xi)$ is unbounded.

▶ **Lemma 5.10** ([15, Lem. 4.4]). *From any pure linear KLM sequence ξ, one can compute in time $\mathsf{exp}(|\xi|^{O(|\xi|)})$ a finite set Ξ of saturated pure linear KLM sequences such that $L_\xi = \bigcup_{\xi'\in\Xi} L_{\xi'}$, and such that $\mathrm{rank}(\xi') = \mathrm{rank}(\xi)$ and $|\xi'| \leq |\xi|^{O(|\xi|)}$ for every $\xi' \in \Xi$.*

Proof. By Lemma 5.7, if a variable $\boldsymbol{m}_i(j)$ or $\boldsymbol{n}_i(j)$ is bounded in $\mathcal{E}(\xi)$, we can replace the corresponding ω component in ξ by all possible values bounded by $(10|\xi|)^{12|\xi|} \leq |\xi|^{O(|\xi|)}$. ◀

The Cleaning Lemma

A linear KLM sequence ξ is called *clean* if it is satisfiable, strongly connected, pure and saturated. The lemmas 5.8 through 5.10 show how to make a linear KLM sequence clean.

▶ **Lemma 5.11.** *From any linear KLM sequence ξ, one can compute in time $\mathsf{exp}(g(|\xi|))$ a finite set $\mathrm{clean}(\xi)$ of clean linear KLM sequences such that $\bigcup_{\xi'\in\mathrm{clean}(\xi)} L_{\xi'} \subseteq L_\xi$ and $\bigcup_{\xi'\in\mathrm{clean}(\xi)} L_{\xi'} \neq \emptyset$ whenever $L_\xi \neq \emptyset$. Moreover, for every $\xi' \in \mathrm{clean}(\xi)$ we have $\mathrm{rank}(\xi') \leq_{lex} \mathrm{rank}(\xi)$ and $|\xi'| \leq g(|\xi|)$ where $g(x) = x^{x^{O(1)}}$.*

5.4 Decomposition of Linear KLM Sequences

In this section we recall three conditions that require a linear KLM sequence to be *unbounded*, *rigid*, and *pumpable*. If any one of them is violated, a decomposition into a set of linear KLM sequences with strictly smaller ranks can be performed. Essentially there is nothing new in this section compared to [15]. The decomposition operations in [15] can be directly applied here, since they operate on a single KLM tuple and produce KLM sequences that are just special cases of linear KLM sequences. The proofs in [15] can also be adapted easily, and we will omit the details here. Especially, the next lemma shows that the arguments of strict decrease in ranks are still valid even though we discard the lower three components of ranks.

▶ **Lemma 5.12.** *Let ξ' be a pure linear KLM sequence. For any linear KLM sequence ξ' with $\mathrm{rank}_{full}(\xi') <_{lex} \mathrm{rank}_{full}(\xi)$, we have $\mathrm{rank}(\xi') <_{lex} \mathrm{rank}(\xi)$.*

Unbounded Sequences

Let $\xi = \langle p_0(\boldsymbol{x}_0)G_0q_0(\boldsymbol{y}_0)\rangle\, \Lambda_1 \cdots \Lambda_k\, \langle p_k(\boldsymbol{x}_k)G_kq_k(\boldsymbol{y}_k)\rangle$ be a linear KLM sequence. We say ξ is *unbounded* if for all $i = 0, \ldots, k$ and every transition $t \in T_i$ where T_i is the set of transitions of G_i, the set of values $\phi_i^{\boldsymbol{h}}(t)$ where $\boldsymbol{h} \models \mathcal{E}(\xi)$ is unbounded. Bounded transitions can be expanded exhaustively according to the bounds given by Lemma 5.7.

▶ **Lemma 5.13** ([15, Lem. 4.6])**.** *Whether a linear KLM sequence ξ is unbounded is decidable in* NP*. Moreover, if ξ is pure and bounded, one can compute in time $\mathsf{exp}(|\xi|^{O(|\xi|)})$ a finite set Ξ of linear KLM sequences such that $L_\xi = \bigcup_{\xi' \in \Xi} L_{\xi'}$ and such that $\mathrm{rank}(\xi') <_{lex} \mathrm{rank}(\xi)$ and $|\xi'| < |\xi|^{O(|\xi|)}$ for every $\xi' \in \Xi$.*

Rigid Sequences

A coordinate $j \in [d]$ is said to be *fixed* by a VASS $G = (Q, T)$ if there exists a function $f_j : Q \to \mathbb{N}$ such that $f_j(q) = f_j(p) + \boldsymbol{a}(j)$ for every transition $(p, \boldsymbol{a}, q) \in T$. We also say that f_j *fixes* G at coordinate j in this case.

A KLM tuple $\langle p(\boldsymbol{x})Gq(\boldsymbol{y})\rangle$ is said to be *rigid* if for every coordinate j fixed by $G = (Q, T)$, there exists a function $g_j : Q \to \mathbb{N}$ that fixes G at coordinate j and such that $g_j(p) \sqsubseteq \boldsymbol{x}(j)$ and $g_j(q) \sqsubseteq \boldsymbol{y}(j)$.

A linear KLM sequence $\xi = \langle p_0(\boldsymbol{x}_0)G_0q_0(\boldsymbol{y}_0)\rangle\, \Lambda_1 \cdots \Lambda_k\, \langle p_k(\boldsymbol{x}_k)G_kq_k(\boldsymbol{y}_k)\rangle$ is said to be *rigid* if every tuple $\langle p_i(\boldsymbol{x}_i)G_iq_i(\boldsymbol{y}_i)\rangle$ in ξ is rigid.

▶ **Lemma 5.14** ([15, Lem. 4.9])**.** *From any pure linear KLM sequence ξ one can decide in polynomial time whether ξ is not rigid. Moreover, in that case one can compute in polynomial time a linear KLM sequence ξ' such that $L_\xi = L_{\xi'}$, $\mathrm{rank}(\xi') <_{lex} \mathrm{rank}(\xi)$, and $|\xi'| \leq |\xi|$.*

Pumpable Sequences

Given a KLM tuple $\langle p(\boldsymbol{x})Gq(\boldsymbol{y})\rangle$, recall the *forward and backward acceleration vectors* $\textsc{Facc}_{G,p}(\boldsymbol{x}), \textsc{Bacc}_{G,q}(\boldsymbol{y}) \in \mathbb{N}_\omega^d$ defined by

$$\textsc{Facc}_{G,p}(\boldsymbol{x})(j) = \begin{cases} \omega & \text{if } p(\boldsymbol{x}) \xrightarrow{*} p(\boldsymbol{x}') \text{ for some } \boldsymbol{x}' \text{ with } \boldsymbol{x}' \geq \boldsymbol{x}, \boldsymbol{x}'(j) > \boldsymbol{x}(j) \\ \boldsymbol{x}(j) & \text{otherwise} \end{cases} \tag{19}$$

$$\textsc{Bacc}_{G,q}(\boldsymbol{y})(j) = \begin{cases} \omega & \text{if } q(\boldsymbol{y}') \xrightarrow{*} q(\boldsymbol{y}) \text{ for some } \boldsymbol{y}' \text{ with } \boldsymbol{y}' \geq \boldsymbol{y}, \boldsymbol{y}'(j) > \boldsymbol{y}(j) \\ \boldsymbol{y}(j) & \text{otherwise} \end{cases} \tag{20}$$

A tuple $\langle p(\boldsymbol{x})Gq(\boldsymbol{y})\rangle$ is said to be *pumpable* if $\textsc{Facc}_{G,p}(\boldsymbol{x})(j) = \textsc{Bacc}_{G,q}(\boldsymbol{y})(j) = \omega$ for every coordinate j *not* fixed by G.

A linear KLM sequence given by $\xi = \langle p_0(\boldsymbol{x}_0)G_0q_0(\boldsymbol{y}_0)\rangle\, \Lambda_1 \cdots \Lambda_k\, \langle p_k(\boldsymbol{x}_k)G_kq_k(\boldsymbol{y}_k)\rangle$ is said to be *pumpable* if every tuple $\langle p_i(\boldsymbol{x}_i)G_iq_i(\boldsymbol{y}_i)\rangle$ in ξ is pumpable.

▶ **Lemma 5.15** ([15, Lem. 4.15]). *Whether a linear KLM sequence ξ is pumpable is decidable in* EXPSPACE. *Moreover, if ξ is pure and unpumpable, one can compute in time* $\mathsf{exp}(|\xi|^{O(1)})$ *a finite set Ξ of linear KLM sequences such that $L_\xi = \bigcup_{\xi'\in\Xi} L_{\xi'}$ and such that* $\mathrm{rank}(\xi') <_{lex} \mathrm{rank}(\xi)$ *and $|\xi'| < |\xi|^{O(1)}$ for every $\xi' \in \Xi$.*

Note that the $O(1)$ term here hides a constant depending on d, which essentially arises from a result on the coverability problem by Rackoff [19]. The $O(1)$ term also captures the difference between the sizes of linear KLM sequences defined here and that in [15].

The Decomposition Lemma

A linear KLM sequence is *normal* if it is clean, unbounded, rigid, and pumpable. The lemmas 5.13 through 5.15 show that when a clean linear KLM sequence is not normal, we are able to decompose it into a finite set of linear KLM sequences of strictly smaller ranks.

▶ **Lemma 5.16.** *From any clean linear KLM sequences ξ that is not normal, one can compute in time* $\mathsf{exp}(h(\xi))$ *a finite set* $\mathrm{dec}(\xi)$ *of clean linear KLM sequence such that $\bigcup_{\xi'\in\mathrm{dec}(\xi)} L_{\xi'} \subseteq L_\xi$ and $\bigcup_{\xi'\in\mathrm{dec}(\xi)} L_{\xi'} \neq \emptyset$ whenever $L_\xi \neq \emptyset$. Moreover, for every $\xi' \in \mathrm{dec}(\xi)$ we have* $\mathrm{rank}(\xi') <_{lex} \mathrm{rank}(\xi)$ *and $|\xi'| \leq h(\xi)$ where $h(x) = x^{x^{x^{O(1)}}}$.*

5.5 Normal Sequences

The following lemma shows that a normal linear KLM sequence is guaranteed to have non-empty action language, thus one can terminate the decomposition process once a normal sequence is produced.

▶ **Lemma 5.17.** *Let ξ be a normal linear KLM sequence, then there is a word $\sigma \in L_\xi$ whose length is bounded by $|\sigma| \leq \ell(|\xi|)$ where $\ell(x) \leq x^{O(x)}$.*

5.6 Putting All Together: The Modified KLMST Algorithm

Here we describe the modified KLMST decomposition algorithm for VASS reachability problem. Suppose we are given a d-VASS $G = (Q, T)$ and two configurations $p(\boldsymbol{m}), q(\boldsymbol{n}) \in Q \times \mathbb{N}^d$. To decide whether $p(\boldsymbol{m}) \xrightarrow{*} q(\boldsymbol{n})$ holds in G, it is enough to decide whether L_ξ is non-empty where $\xi = \langle p(\boldsymbol{m})Gq(\boldsymbol{n})\rangle$. To start with, we use Lemma 5.11 to clean the sequence ξ, and then choose $\xi^0 \in \mathrm{clean}(\xi)$ non-deterministically. If ξ^0 is normal then we are done by Lemma 5.17. Otherwise, we decompose ξ^0 using Lemma 5.16 and choose $\xi^1 \in \mathrm{dec}(\xi^0)$ non-deterministically. The procedure continues to produce a series of linear KLM sequences $\xi^0, \xi^1, \xi^2, \ldots$ where $\xi^{i+1} \in \mathrm{dec}(\xi^i)$, until either we finally obtain a normal sequence ξ^n, or at some point we have to abort because the decomposition of a linear KLM sequence is the empty set. The procedure terminates because $\mathrm{rank}(\xi^0) >_{\mathrm{lex}} \mathrm{rank}(\xi^1) >_{\mathrm{lex}} \mathrm{rank}(\xi^2) >_{\mathrm{lex}} \cdots$ form a decreasing chain of the well-order $(\mathbb{N}^{d-2}, <_{\mathrm{lex}})$, which must be finite. If $L_\xi = \emptyset$ then we cannot get a normal sequence since the action languages $L_\xi \supseteq L_{\xi^0} \supseteq L_{\xi^1} \supseteq \cdots$ are all empty. On the other hand, if $L_\xi \neq \emptyset$ then there are non-deterministic choices that always choose the linear KLM sequences with non-empty action languages, which finally lead to a normal sequence. This shows the correctness of the algorithm.

6 Complexity Upper Bound

The termination of the modified KLMST decomposition algorithm is guaranteed by a ranking function that decreases along a well-ordering. In order to analyze the length of this decreasing chain, we recall the so-called "length function theorems" by Schmitz [20] in Section 6.1. After that, we can locate the complexity upper bound of the algorithm in the fast-growing complexity hierarchy [21] which we recall in Section 6.2. Readers familiar with [15] may realize that the complexity upper bound for d-VASS can be improved to F_{d+1}, i.e. the $(d+1)$-th level in the fast-growing hierarchy, with our ranking function. In fact, by a careful analysis on a property of fast-growing functions, we further improve this bound to F_d.

In this section we assume some familiarity with ordinal numbers (see, e.g. [9]). We write ω here for the first infinite ordinal, not to be confused with the infinite element in previous sections.

6.1 Length of Sequences of Decreasing Ranks

Let ξ be a linear KLM sequence of dimension d with $\mathrm{rank}(\xi) = (r_d, \dots, r_3)$, we define the *ordinal rank* α_ξ of ξ as the ordinal number given by

$$\alpha_\xi := \omega^{d-3} \cdot r_d + \omega^{d-4} \cdot r_{d-1} + \cdots + \omega^0 \cdot r_3. \tag{21}$$

Note that $\mathrm{rank}(\xi) <_{\mathrm{lex}} \mathrm{rank}(\xi')$ if and only if $\alpha_\xi < \alpha_{\xi'}$. With this reformulation, we now focus on the decreasing chain of ordinal ranks.

Let $\alpha < \omega^\omega$ be an ordinal given in Cantor Normal Form as $\alpha = \omega^n \cdot c_n + \cdots + \omega^0 \cdot c_0$ where $n, c_0, \dots, c_n \in \mathbb{N}$, we define the *size* of α as $N\alpha := \max\{n, \max_{0 \le i \le n} c_i\}$. For the linear KLM sequence ξ with $\mathrm{rank}(\xi) = (r_d, \dots, r_3)$, we have $N\alpha_\xi = \max\{d-3, \max_{3 \le i \le d} r_i\} \le |\xi|$.

Given a number $n_0 \in \mathbb{N}$ and a function $h : \mathbb{N} \to \mathbb{N}$ that is monotone inflationary (that is, $x \le h(x)$ and $h(x) \le h(y)$ whenever $x \le y$), we say a sequence of ordinals $\alpha_0, \alpha_1, \dots$ is (n_0, h)-*controlled* if $N\alpha_i \le h^i(n_0)$ for all $i \in \mathbb{N}$, where $h^i(n_0)$ is the ith iteration of h on n_0.

Let $\xi^0, \xi^1, \dots$ be the linear KLM sequences produced in the modified KLMST algorithm, by Lemma 5.16 we know that the sequence of ordinal ranks

$$\alpha_{\xi^0} > \alpha_{\xi^1} > \cdots \tag{22}$$

is $(|\xi^0|, h)$-controlled where h is defined in Lemma 5.16. Recall that $\xi^0 \in \mathrm{clean}(\xi)$ where $\xi = \langle p(\boldsymbol{m}) G q(\boldsymbol{n}) \rangle$ corresponds to the input reachability instance. Then $|\xi^0| \le g(|\xi|)$ where g is defined in Lemma 5.11, and (22) is indeed $(g(n), h)$-controlled where $n := |\langle p(\boldsymbol{m}) G q(\boldsymbol{n}) \rangle|$.

Length function theorem

The length of the controlled sequence of ordinals (22) can be bounded in terms of the hierarchies of fast-growing functions of Hardy and Cichoń [4]. First recall that given a limit ordinal $\lambda \le \omega^\omega$, the standard *fundamental sequence* of λ is a sequence $(\lambda(x))_{x<\omega}$ defined inductively by

$$\omega^\omega(x) := \omega^{x+1}, \qquad (\beta + \omega^{k+1})(x) := \beta + \omega^k \cdot (x+1) \tag{23}$$

where $\beta + \omega^{k+1}$ is in Cantor Normal Form. Now given a function $h : \mathbb{N} \to \mathbb{N}$ that is monotone inflationary, we define the *Hardy hierarchy* $(h^\alpha)_{\alpha \le \omega^\omega}$ and the *Cichoń hierarchy* $(h_\alpha)_{\alpha \le \omega^\omega}$ as two families of functions $h^\alpha, h_\alpha : \mathbb{N} \to \mathbb{N}$ indexed by ordinals $\alpha \le \omega^\omega$ given inductively by

$$h^0(x) := x, \qquad h^{\alpha+1}(x) := h^\alpha(h(x)), \qquad h^\lambda(x) := h^{\lambda(x)}(x), \tag{24}$$

$$h_0(x) := 0, \qquad h_{\alpha+1}(x) := 1 + h_\alpha(h(x)), \qquad h_\lambda(x) := h_{\lambda(x)}(x). \tag{25}$$

Observe that Cichoń hierarchy counts the number of iterations of h in Hardy hierarchy, that is, $h^{h_\alpha(x)}(x) = h^\alpha(x)$. Also note that as h is monotone inflationary, by induction on α we have $h_\alpha(x) \leq h^\alpha(x)$. Now we give the length function theorem as follows.

▶ **Theorem 6.1** (Length function theorem, [20, Thm. 3.3]). *Let $n_0 \geq d-2$, then the maximal length of (n_0, h)-controlled decreasing sequences of ordinals in ω^{d-2} is $h_{\omega^{d-2}}(n_0)$.*

Small witness property

By Theorem 6.1 we can bound the length of (22), which then yields a bound on the minimal length of runs witnessing reachability.

▶ **Lemma 6.2** (Small witnesses). *Let $G = (Q, T)$ be a d-VASS where $d \geq 3$, let $p(\boldsymbol{m}), q(\boldsymbol{n}) \in Q \times \mathbb{N}^d$ be two configurations, and let $n := |\langle p(\boldsymbol{m}) G q(\boldsymbol{n})\rangle|$. If $p(\boldsymbol{m}) \xrightarrow{*} q(\boldsymbol{n})$ holds in G, then there is a path σ such that $p(\boldsymbol{m}) \xrightarrow{\sigma} q(\boldsymbol{n})$ and $|\sigma| \leq \ell(h^{\omega^{d-2}}(g(n)))$, where g, h, ℓ are defined in lemmas 5.11, 5.16, and 5.17.*

Proof. Suppose $p(\boldsymbol{m}) \xrightarrow{*} q(\boldsymbol{n})$, then there is a sequence of linear KLM sequence $\xi^0, \xi^1, \ldots, \xi^L$ produced in the modified KLMST algorithm, such that ξ^L is normal. We have discussed that the sequence of their ordinal ranks $\alpha_{\xi^0} > \alpha_{\xi^1} > \cdots > \alpha_{\xi^L}$ is $(g(n), h)$-controlled, so by Theorem 6.1 we have $L \leq h_{\omega^{d-2}}(g(n))$. From Lemma 5.16 and the fact that $h^{h_\alpha(x)} = h^\alpha(x)$, the size of ξ^L is bounded by

$$|\xi^L| \leq h^L(|\xi^0|) \leq h^{h_{\omega^{d-2}}(g(n))}(g(n)) = h^{\omega^{d-2}}(g(n)). \tag{26}$$

Now Lemma 5.17 bounds the length of the minimal witnesses by $\ell(h^{\omega^{d-2}}(g(n)))$. ◀

6.2 Fast-Growing Complexity Hierarchy

We recall the fast-growing hierarchy formally introduced by Schmitz [21] that captures the complexity class high above elementary. Define $H(x) := x + 1$, we shall use the Hardy hierarchy $(H^\alpha)_{\alpha<\omega^\omega}$, where for example $H^{\omega^2}(x) = 2^{x+1}(x+1)$ and $H^{\omega^3}(x)$ grows faster than the tower function. First we define the family $\mathcal{F}_\alpha := \bigcup_{\beta<\omega^\alpha} \mathsf{FDTIME}(H^\beta(n))$ which contains functions computable in deterministic time $O(H^\beta(n))$. Observe that, for example, $\mathcal{F}_3$ contains exactly the Kalmar elementary functions. Now we define

$$\mathsf{F}_\alpha := \bigcup_{p\in\mathcal{F}_\alpha} \mathsf{DTIME}(H^{\omega^\alpha}(p(n))) \tag{27}$$

which is the class of decision problems solvable in deterministic time $O(H^{\omega^\alpha}(p(n)))$. Note that non-deterministic time Turing machines can be made deterministic with an exponential overhead in $\mathcal{F}_3$, thus for $\alpha \geq 3$, we have equivalently that $\mathsf{F}_\alpha = \bigcup_{p\in\mathcal{F}_\alpha} \mathsf{NDTIME}(H^{\omega^\alpha}(p(n)))$. Observe that F_α is closed under reductions in $\mathcal{F}_\alpha$.

6.2.1 Relativized Fast-Growing Functions

In order to express the complexity of the modified KLMST algorithm in terms of the hierarchy $(\mathsf{F}_\alpha)_{\alpha<\omega}$, one needs to locate the function $h^{\omega^{d-2}}$ in the Hardy hierarchy $(H^\alpha)_{\alpha<\omega^\omega}$ where $h \in \mathcal{F}_3$ is the elementary function from Lemma 5.16. Previously we can upper bound $h^{\omega^{d-2}}$ by $H^{\omega^{d+1}}$ with the help of [21, Lem. 4.2]. Here we show a slightly better result, from which we can bound $h^{\omega^{d-2}}(x)$ by $H^{\omega^d}(O(x))$.

▶ **Lemma 6.3** (cf. [21, Lem. A.5]). *Let $h : \mathbb{N} \to \mathbb{N}$ be a monotone inflationary function, let $a, b, c \geq 1$ and $x_0 \geq 0$ be natural numbers. If for all $x \geq x_0$, $h(x) \leq H^{\omega^{b\cdot c}}(x)$, then $h^{\omega^a}(x) \leq H^{\omega^{b+a}}((c+1)x)$ for all $x \geq \max\{2c, x_0\}$.*

6.3 Upper Bounds for VASS Reachability

Now we analyze the time complexity of the modified KLMST algorithm. Given as input the d-VASS $G = (Q, T)$ and two configurations $p(\boldsymbol{m}), q(\boldsymbol{n})$, let $\xi := \langle p(\boldsymbol{m}) G q(\boldsymbol{n}) \rangle$ and $n := |\xi|$. The initial sequence $\xi^0 \in \mathrm{clean}(\xi)$ can be computed in (non-deterministic) time elementary in n by Lemma 5.11. Then the algorithm produces $\xi^0, \xi^1, \ldots, \xi^L$ with $L \leq h_{\omega^{d-2}}(g(n))$, where g, h are defined in lemmas 5.11, 5.16. Note that in each step, the sequence $\xi^{i+1} \in \mathrm{dec}(\xi^i)$ can be computed in time elementary in $|\xi^i|$ by Lemma 5.16, and the sizes $|\xi^i|$ are all bounded by $h^{\omega^{d-2}}(g(n))$ as we have discussed above in the proof of Lemma 6.2. To sum up, the entire algorithm finishes in non-deterministic time elementary in $h^{\omega^{d-2}}(g(n))$.

▶ **Lemma 6.4.** *On input a d-VASS $G = (Q, T)$ where $d \geq 3$ and $p(\boldsymbol{m}), q(\boldsymbol{n}) \in Q \times \mathbb{N}^d$, the modified KLMST algorithm finishes in non-deterministic time $e(h^{\omega^{d-2}}(g(n)))$ where $n = |\langle p(\boldsymbol{m}) G q(\boldsymbol{n}) \rangle|$, g, h are defined in lemmas 5.11, 5.16, and $e \in \mathcal{F}_3$ is some fixed function.*

Since h is an elementary function, there is a number $c \in \mathbb{N}$ such that h is eventually dominated by $H^{\omega^2 \cdot c}$. By Lemma 6.3 we can upper bound $h^{\omega^{d-2}}(x)$ by $H^{\omega^d}((c+1)x)$. Observe that the inner part $g(n)$ is elementary in the binary encoding size of the input $G, p(\boldsymbol{m}), q(\boldsymbol{n})$, thus can be captured by a function $p \in \mathcal{F}_3$. Finally, [21, Lem. 4.6] allows us to move the outermost function e to the innermost position. Hence we have the following upper bound.

▶ **Theorem 6.5.** *Reachability in d-dimensional VASS is in F_d for all $d \geq 3$.*

Also, by Lemma 6.2 there is a simple combinatorial algorithm for d-VASS reachability. We fist compute the bound $B := \ell(h^{\omega^{d-2}}(g(n)))$, which can be done in time elementary in B by [21, Thm. 5.1]. Then we can decide reachability by just enumerate all possible paths in G with length bounded by B.

7 Conclusion

We have shown that the reachability problem in d-dimensional vector addition system with states is in F_d, improving the previous F_{d+4} upper bound by Leroux and Schmitz [15]. By capturing reachability in geometrically 2-dimensional VASSes with linear path schemes, we are able to reduce significantly the number of decomposition steps in the KLMST decomposition algorithm. Combined with a careful analysis on fast-growing functions, we finally obtained the F_d upper bound. It should be noticed though, that our algorithm avoids computing the "full decomposition" [14] of KLM sequences, thus cannot improve the complexity of problems that essentially rely on the full decomposition, e.g., the VASS downward language inclusion problem [8, 23, 15].

It has been shown that the reachability problem in $(2d{+}3)$-VASS is F_d-hard [5]. In the case of 3-VASS, it is known that the reachability problem is PSPACE-hard. The gap between the lower bound and the upper bound $\mathsf{F}_3 = \mathsf{TOWER}$ [22] is huge. It is very unlikely that the problem is PSPACE-complete. Effort to uplift the lower bound is called for.

References

1 Michael Blondin, Matthias Englert, Alain Finkel, Stefan Göller, Christoph Haase, Ranko Lazic, Pierre McKenzie, and Patrick Totzke. The reachability problem for two-dimensional vector addition systems with states. *J. ACM*, 68(5):34:1–34:43, 2021. doi:10.1145/3464794.

2 Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in two-dimensional vector addition systems with states is pspace-complete. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 32–43. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.14.

3 Dmitry Chistikov and Christoph Haase. The taming of the semi-linear set. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 128:1–128:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.ICALP.2016.128.

4 E. A. Cichon and Elias Tahhan-Bittar. Ordinal recursive bounds for higman's theorem. *Theor. Comput. Sci.*, 201(1-2):63–84, 1998. doi:10.1016/S0304-3975(97)00009-1.

5 Wojciech Czerwinski, Ismaël Jecker, Slawomir Lasota, Jérôme Leroux, and Lukasz Orlikowski. New lower bounds for reachability in vector addition systems. In Patricia Bouyer and Srikanth Srinivasan, editors, *43rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2023, December 18-20, 2023, IIIT Hyderabad, Telangana, India*, volume 284 of *LIPIcs*, pages 35:1–35:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.FSTTCS.2023.35.

6 Wojciech Czerwinski and Lukasz Orlikowski. Reachability in vector addition systems is ackermann-complete. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1229–1240. IEEE, 2021. doi:10.1109/FOCS52979.2021.00120.

7 Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in succinct and parametric one-counter automata. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR 2009 - Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy, September 1-4, 2009. Proceedings*, volume 5710 of *Lecture Notes in Computer Science*, pages 369–383. Springer, 2009. doi:10.1007/978-3-642-04081-8_25.

8 Peter Habermehl, Roland Meyer, and Harro Wimmel. The downward-closure of petri net languages. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 466–477. Springer, 2010. doi:10.1007/978-3-642-14162-1_39.

9 Thomas Jech. *Set Theory*. Springer Monographs in Mathematics. Springer, 2003. doi:10.1007/3-540-44761-X.

10 S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 267–281, New York, NY, USA, 1982. Association for Computing Machinery. doi:10.1145/800070.802201.

11 Jean-Luc Lambert. A structure to decide reachability in petri nets. *Theor. Comput. Sci.*, 99(1):79–104, 1992. doi:10.1016/0304-3975(92)90173-D.

12 Jérôme Leroux. Flat petri nets (invited talk). In Didier Buchs and Josep Carmona, editors, *Application and Theory of Petri Nets and Concurrency - 42nd International Conference, PETRI NETS 2021, Virtual Event, June 23-25, 2021, Proceedings*, volume 12734 of *Lecture Notes in Computer Science*, pages 17–30. Springer, 2021. doi:10.1007/978-3-030-76983-3_2.

13 Jérôme Leroux. The reachability problem for petri nets is not primitive recursive. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1241–1252. IEEE, 2021. doi:10.1109/FOCS52979.2021.00121.

14 Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 56–67. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.16.

15 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785796.

16 Jérôme Leroux and Grégoire Sutre. On flatness for 2-dimensional vector addition systems with states. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings*, volume 3170 of *Lecture Notes in Computer Science*, pages 402–416. Springer, 2004. doi:10.1007/978-3-540-28644-8_26.

17 Ernst W. Mayr. An algorithm for the general petri net reachability problem. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 238–246, New York, NY, USA, 1981. Association for Computing Machinery. doi:10.1145/800076.802477.

18 Loic Pottier. Minimal solutions of linear diophantine systems: Bounds and algorithms. In Ronald V. Book, editor, *Rewriting Techniques and Applications, 4th International Conference, RTA-91, Como, Italy, April 10-12, 1991, Proceedings*, volume 488 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 1991. doi:10.1007/3-540-53904-2_94.

19 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6:223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

20 Sylvain Schmitz. Complexity bounds for ordinal-based termination - (invited talk). In Joël Ouaknine, Igor Potapov, and James Worrell, editors, *Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings*, volume 8762 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2014. doi:10.1007/978-3-319-11439-2_1.

21 Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Trans. Comput. Theory*, 8(1):3:1–3:36, 2016. doi:10.1145/2858784.

22 Qizhe Yang and Yuxi Fu. Reachability in 3-vass is in tower. *CoRR*, abs/2306.05710, 2023. doi:10.48550/arXiv.2306.05710.

23 Georg Zetzsche. The complexity of downward closure comparisons. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 123:1–123:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.ICALP.2016.123.

On Classes of Bounded Tree Rank, Their Interpretations, and Efficient Sparsification

Jakub Gajarský
University of Warsaw, Poland

Rose McCarty
School of Mathematics and School of Computer Science,
Georgia Institute of Technology, Altanta, GA, USA

Abstract

Graph classes of bounded tree rank were introduced recently in the context of the model checking problem for first-order logic of graphs. These graph classes are a common generalization of graph classes of bounded degree and bounded treedepth, and they are a special case of graph classes of bounded expansion. We introduce a notion of decomposition for these classes and show that these decompositions can be efficiently computed. Also, a natural extension of our decomposition leads to a new characterization and decomposition for graph classes of bounded expansion (and an efficient algorithm computing this decomposition).

We then focus on interpretations of graph classes of bounded tree rank. We give a characterization of graph classes interpretable in graph classes of tree rank 2. Importantly, our characterization leads to an efficient sparsification procedure: For any graph class $\mathcal{C}$ interpretable in a graph class of tree rank at most 2, there is a polynomial time algorithm that to any $G \in \mathcal{C}$ computes a (sparse) graph H from a fixed graph class of tree rank at most 2 such that $G = I(H)$ for a fixed interpretation I. To the best of our knowledge, this is the first efficient "interpretation reversal" result that generalizes the result of Gajarský et al. [LICS 2016], who showed an analogous result for graph classes interpretable in classes of graphs of bounded degree.

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Theory of computation → Fixed parameter tractability; Mathematics of computing → Graph theory

Keywords and phrases First-order model checking, structural graph theory, structural sparsity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.137

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2404.18904`

Funding *Jakub Gajarský*: Supported by the Polish National Science Centre SONATA-18 grant number 2022/47/D/ST6/03421. Parts of this work were developed while this author received funding from the European Research Council (ERC), grant agreement No 948057 – BOBR.
Rose McCarty: Supported by the National Science Foundation under Grant No. DMS-2202961.

1 Introduction

The graph classes and problems studied in this paper are motivated by considering the first-order (FO) model checking problem for graphs. This problem asks, given a (finite) graph G and sentence φ as input, whether G is a model of φ. This problem is known to be PSPACE-hard, and so we do not expect to obtain a polynomial algorithm solving it. This has motivated the study of the FO model checking problem from the perspective of parameterized complexity, which has led to the discovery of many beautiful connections between structural and algorithmic graph theory and (finite) model theory.

In the parameterized setting (where we consider the size of the formula φ as the parameter), we can easily obtain a brute-force algorithm with runtime $n^{O(|\varphi|)}$, the so-called XP algorithm. However, we are interested in the existence of algorithms with runtime $f(|\varphi|) \cdot n^c$, where c

is some fixed constant (FPT algorithms). We do not expect to obtain such an algorithm if the input graphs come from the class of all graphs, since this problem is AW[$*$]-complete. However, for various structurally restricted graph classes we know that such an algorithm exists. Identifying these graph classes is the topic of a long line of research, and recently a lot of progress has been made towards understanding the boundaries of efficient tractability [2, 4, 5].

For graph classes that admit a model checking algorithm with runtime $f(|\varphi|) \cdot n^c$ one can ask what is the dependence of the runtime on the size of the formula φ. Unfortunately, for most graph classes this dependence is *non-elementary*, that is, $f(k)$ grows like a tower of twos whose height grows with k. By a result of Frick and Grohe [9], we know that this cannot be avoided even when $\mathcal{C}$ is the class of all trees. While this result may seem very limiting, it turns out that the landscape of graph classes that admit an elementary model checking algorithm is surprisingly rich. Classical examples of such graph classes include classes of graphs of bounded degree [9], bounded treedepth, and bounded shrubdepth [11]. (The last two results were obtained in the more general context of MSO model checking.) The interest in this problem was renewed recently when Lampis [15] established elementary model checking for graph classes of bounded pathwidth. After this, Gajarský, Pilipczuk, Sokołowski, Stamoulis and Toruńczyk [13] introduced graph classes of bounded tree rank and proved that these classes also admit (under certain restrictions, see the fourth bullet point below) an elementary FO model checking algorithm.

Classes of bounded tree rank, introduced in [13], can be defined as follows (in the definition, the *depth* of a tree T is the number of edges on a shortest leaf-to-root path in T).

▶ **Definition 1.** *A graph class $\mathcal{C}$ has* tree rank *at most d if for every $r \in \mathbb{N}$ there exists a tree T of depth d such that no $G \in \mathcal{C}$ contains T as an r-shallow topological minor*[1].

For example, one can easily see that graph classes of bounded degree have tree rank at most 1, that the class of all trees of height d has tree rank d, and that the class of all trees does not have bounded tree rank.

We briefly summarize some of the basic properties of graph classes of bounded tree rank established in [13]:

- They generalize graph classes of bounded degree, bounded treedepth, and bounded pathwidth.
- They are strictly less general than classes of graphs of bounded expansion.
- The tree rank of a graph class $\mathcal{C}$ is at most d if and only if Splitter has a winning strategy in the so-called "d-round batched Splitter game" for every $G \in \mathcal{C}$. This game is a natural variant of the Splitter game, which characterizes nowhere dense graph classes and was introduced by Grohe, Kreutzer, and Siebertz in their landmark paper [14].
- If $\mathcal{C}$ is a class of bounded tree rank and there exists an elementary function $f : \mathbb{N} \to \mathbb{N}$ such that the size of T in Definition 1 is bounded by $f(r)$, then there is an elementary FO model checking algorithm for $\mathcal{C}$.
- If $\mathcal{C}$ is a monotone graph class of unbounded tree rank, then $\mathcal{C}$ has no FO model checking algorithm with elementary dependence on the size of the formula unless FPT=AW[$*$].

We note that the concepts considered above (bounded degree, bounded treedepth, bounded tree rank, bounded expansion, and nowhere denseness) are all examples of classes of *sparse* graphs. Given how naturally the concept of bounded tree rank fits into the general theory of

[1] We note that this definition of tree rank is seemingly off by 1 from the original definition in [13]. However, we measure the depth of trees differently than in [13], and so the two definitions ultimately coincide.

sparse graphs introduced by Nešetřil and Ossona de Mendez [17], we suspect that it will play an important role in structural and algorithmic graph theory, and that further investigation of its structural and combinatorial properties is desirable.

Since classes of sparse graphs are mostly well-understood (one exception being classes of bounded tree rank), in recent years there has been a trend to study more general classes that can be obtained from classes of sparse graphs by means of *interpretations* or *transductions*, which are graph transformations based in logic. This point of view is also very relevant for classes of bounded tree rank, but we first discuss it in the general setting of sparse graphs. We focus on the simpler setting of interpretations. Essentially, an interpretation I is given by a formula $\psi(x, y)$ (we will actually use a slightly more complicated setting; see Section 2). When applied to a graph H, the result is a new graph $I(H)$ with the same vertex set as H and with edge set $\{uv \ : \ H \models \psi(u, v)\}$. This notion generalises easily to graph classes by setting $I(\mathcal{C}) = \{I(H) \ : \ H \in \mathcal{C}\}$. Finally, we say that a graph class $\mathcal{D}$ is *interpretable* in a graph class $\mathcal{C}$ if there exists an interpretation I such that $\mathcal{D} \subseteq I(\mathcal{C})$.

As mentioned above, the recent trend is to study graph classes interpretable in sparse (or nowhere dense) classes of graphs. In particular, the model checking problem has been considered extensively, and was recently fully solved for such graph classes.

▶ **Theorem 2** ([5]). *Let $\mathcal{C}$ be a graph class interpretable in a nowhere dense graph class. Then there exists an FPT model checking algorithm for $\mathcal{C}$.*

In the context of graph classes interpretable in classes of sparse graphs, it is also natural consider the following problem, which we refer to as the "efficient sparsification problem" or "interpretation reversal problem". This problem was considered in [12] in the context of graph classes interpretable in classes of graphs of bounded degree.

▶ **Problem 1.** *Let $\mathcal{C}$ be a graph class interpretable in a nowhere dense graph class. Show that there exists a nowhere dense graph class $\mathcal{D}$, an interpretation I, and a polynomial time algorithm that given $G \in \mathcal{C}$ as input computes a graph $H \in \mathcal{D}$ such that $G = I(H)$.*

It is well-known (and easy to argue, see for instance [12]) that solving Problem 1 would lead to an alternative proof of Theorem 2. Indeed, before the result of [5] was established, this was considered the main line of attack on the model checking problem on graph class interpretable in a nowhere dense graph class. However, Problem 1 turned out to be very challenging, and despite considerable effort essentially no success has been achieved in solving it. Instead, Theorem 2 was proved by adapting the techniques for classes of sparse graphs from [14] to the dense setting. Despite this, Problem 1 remains of considerable interest, and solving it would likely lead to many new insights on the structure of graph classes interpretable in classes of sparse graphs.

Coming back to classes of bounded tree rank, it is natural to try to obtain an elementary analogue of Theorem 2.

▶ **Problem 2.** *Let $\mathcal{C}$ be a graph class interpretable in a graph class of bounded tree rank. Show that there exists an elementary FPT model checking algorithm for $\mathcal{C}$.*

In light of the results of [13], it is possible that one needs to put some extra restriction on $\mathcal{C}$ – perhaps requiring that the size of trees avoided as r-shallow topological minors is bounded by an elementary function of r. However, it is currently unknown whether this is necessary, even for the original theorem from [13].

In relation to interpretations of graph classes of bounded tree rank, the authors of [13] introduced the more general graph classes of bounded *rank*, and conjectured that these are precisely the interpretations of colored graph classes of bounded tree rank. In order to attack this conjecture, or Problem 2, it is natural to consider the following variant of Problem 1.

▶ **Problem 3.** *Let $\mathcal{C}$ be a graph class interpretable in a graph class of bounded tree rank. Show that there exists a graph class $\mathcal{D}$ of bounded tree rank, an interpretation I, and a polynomial time algorithm that given $G \in \mathcal{C}$ as input computes a graph $H \in \mathcal{D}$ such that $G = I(H)$.*

Our contribution

For sparse graphs we introduce the notion of the (r, m)-*rank* of a vertex. Roughly speaking, the (r, m)-rank of a vertex $v \in V(G)$ is a positive integer that measures how complicated the r-neighborhood of v is with respect to the parameter m. We also allow the (r, m)-rank to be ∞ if the r-neighborhood of v is too complicated. Then the (r, m)-*ranking* of a graph G is the function $f : V(G) \to \mathbb{N} \cup \{\infty\}$ that assigns to each vertex of G its (r, m)-rank. This definition has the advantage of being much more localized than Definition 1 of tree rank. Yet we prove that it can also be used to define tree rank, as follows.

▶ **Theorem 3.** *For any $d \in \mathbb{N}$ and any graph class $\mathcal{C}$, the following are equivalent:*
- *The tree-rank of $\mathcal{C}$ is at most d.*
- *For every $r \in \mathbb{N}$ there exists $m \in \mathbb{N}$ such that every vertex of every graph in $\mathcal{C}$ has (r, m)-rank at most d.*

Vertex rankings provide a natural notion of decomposition for graph classes of bounded tree rank. We will show that this decomposition can be computed in elementary FPT runtime with respect to the parameters r and m (see Theorem 14). Moreover, the notion of (r, m)-ranking (and the FPT algorithm computing it) very naturally extends to graph classes of bounded expansion.

▶ **Theorem 4.** *The following are equivalent for any graph glass $\mathcal{C}$:*
- *The class $\mathcal{C}$ is of bounded expansion.*
- *For every $r \in \mathbb{N}$ there exists $m \in \mathbb{N}$ such that every vertex of every graph in $\mathcal{C}$ has finite (r, m)-rank, that is, has (r, m)-rank not equal to ∞.*

We note that the (r, m)-rank of a vertex is preserved under graph automorphisms. This fact that rankings are "canonical" is one of the key advantages of Theorem 4.

One of the main motivations for proving alternate characterizations of sparse graph classes (like Theorems 3 and 4) is the efficient sparsification problems discussed earlier (Problems 1 and 3). In particular, we hope that this definition of (r, m)-rank can be generalized to accommodate for interpretations of classes of bounded expansion. While we cannot yet take care of this more general case, we are able to use the insights obtained from (r, m)-rank to find the following structural characterization of interpretations of classes of tree rank 2.

▶ **Theorem 5.** *The following are equivalent for any graph glass $\mathcal{C}$:*
- *The class $\mathcal{C}$ is interpretable in a graph class of tree rank 2.*
- *The class $\mathcal{C}$ is a perturbation of a locally almost near-covered graph class.*

We defer the definition of locally almost near-covered graph classes to Section 5. Our characterization leads to the following algorithmic sparsification result. For technical reasons we restrict ourselves to classes of graphs interpretable in classes of tree rank 2 for which the bounds in the definition of tree rank can be efficiently computed. We call such classes *efficiently bounded* (see Section 5.1) for precise definition.

▶ **Theorem 6.** *Let $\mathcal{C}$ be a graph class interpretable in an efficiently bounded graph class of tree rank 2. Then there exists a graph class $\mathcal{D}$ of tree rank 2, an interpretation I, and a polynomial time algorithm that given $G \in \mathcal{C}$ as input computes a graph $H \in \mathcal{D}$ such that $G = I(H)$.*

We remark that the degree of the polynomial in Theorem 6 depends on the class $\mathcal{C}$.

While at first sight Theorem 6 might seem like a modest improvement on the analogous result of [12] for graph classes of bounded degree (or equivalently, classes of tree rank at most 1), this is the first progress on the challenging Problem 1 since its introduction that applies to graph classes of unbounded treewidth. (For graph classes of bounded treewidth such a result is given in [18].) Moreover, our proof introduces new techniques and ideas that may be useful for solving the sparsification problem (Problem 1) in greater generality.

In particular, as part of the proof of Theorem 5 we prove a lemma (Lemma 23) about the behaviour of the k-near-twin relation on graphs that do not contain a half-graph as a semi-induced subgraph. This lemma may be of independent interest in the context of (monadically) stable graph classes, which play a prominent role in recent developments establishing connections between (finite) model theory and algorithmic graph theory [4, 6].

Related work

In [13] it was shown that classes of tree rank at most d can be characterized using the *batched Splitter game* with d rounds. Roughly speaking, in this game two players take turns, one of them trying to simplify the graph, and the other trying to keep the graph as complicated as possible. The result of [13] states that the player trying to simplify the graph (this player is called Splitter) wins in at most d rounds.

While this characterization is very nice and useful, it is a characterization in terms of a dynamic process, and as such it does not directly provide us with a useful notion of decomposition (in the sense of giving us a concrete compact object on which one can design algorithms). One could of course consider using as a decomposition the game tree arising from a winning play by Splitter, but this tree has size of order n^d. It is known that this can be circumvented by combining the Splitter game with *sparse neighborhood covers* as introduced in [14], but working with the resulting object is technically demanding. Compared to this, vertex rankings give us a static decomposition on which one can use bottom-up inductive arguments and can design algorithms.

For graph classes of bounded expansion, vertex rankings of a graph (which certify that it has only vertices of finite rank) are closely related to strong coloring orders [23] and admissibility orders [7]. These are total orders on the vertex set of a graph that satisfy certain properties, and which have proven to be very useful for showing properties of graphs of bounded expansion. The main idea used in the definition of rankings, when one checks whether a removal of a small set of vertices can separate a vertex v from the set of previously processed vertices, has been used before (see Appendix A1 in [22]). This was again in the context of defining a suitable total order on the vertex set of a graph. Compared to total orders, our definition of rankings essentially leads to a pre-order on the vertex set of a graph, and therefore it can capture situations when two vertices are equally complicated (have the same rank).

Regarding the sparsification problem, in [12] it was shown that there is a polynomial time algorithm for sparsifying graphs from graph classes that are interpretable in classes of bounded degree. Another result related to efficient sparsification is [8], where the authors showed how to efficiently interpret the class of map graphs in a nowhere dense class of graphs. Finally, the already mentioned result [18] establishes a sparsification result for graph classes interpretable in graph classes of bounded treewidth, extending earlier results in [19].

Outline of our approach

We now briefly outline the approach used to prove Theorems 5 and 6. This approach builds on the ideas used in [12] to sparsify classes of graphs interpretable in graph classes of bounded degree. The key notion behind this result was that of *k-near-twin* vertices. We say that

two vertices u, v of G are k-near-twins if $|N^G(v)\Delta N^G(u)| \leq k$, i.e. if they have the same neighborhoods with at most k exceptions. The idea behind the proof given in [12] is that graphs from graph classes interpretable in classes of bounded degree have a simple structure – for such graphs we can find a small k such that the k-near-twin relation is an equivalence on $V(G)$ with a bounded number of classes. This is then easily used to find a bounded number of *flips* (edge complementations between two subsets of $V(G)$) to produce a sparse graph H from which G can be recovered by an interpretation.

In the case of classes of graphs interpretable in graph classes of tree rank 2, the structure of the k-near-twin relation is much more complicated. In what follows we will actually focus on analysing this relation on graphs from graph classes interpreted in classes of tree rank 2 by an *interpretation of bounded range*. This is an interpretation in which the formula $\psi(x, y)$ has the property that there is a number b such that for every G and $u, v \in V(G)$ we have that if $\mathrm{dist}_G(u, v) > b$, then $G \not\models \psi(u, v)$. In other words, such interpretation will never create edges between vertices that are far apart in G. The case of interpretations of bounded range forms the techincal core of our approach, since the reduction from the general case to the case of bounded range can be achieved by using existing tools (see Section 5.3). For the rest of this overview, let us fix a graph class $\mathcal{C}$ interpretable in a class of graphs of tree rank at most 2 by an interpretation of bounded range.

We now proceed with analysing the k-near-twin relation on a graph G from $\mathcal{C}$. It is useful to think of this relation in terms of the *k-near-twin graph* of G, denoted by $NT_k(G)$. This graph has the same vertex set as G, and two vertices are adjacent in $NT_k(G)$ if they are k-near-twins in G. In the case of interpretations of bounded degree, this graph was a collection of a small number of cliques. In our case of interpretations of classes of tree rank 2, the connected components of the graph $NT_k(G)$ are not cliques, and there can be arbitrarily many of them. Moreover, the connected components of $NT_k(G)$ can have arbitrarily large diameter – this is important because if we could find a bound d such that the diameter of all connected components of $NT_k(G)$ was at most d, then all vertices in any component C would be kd-near-twins, and this could be exploited for sparsification. In our proofs of Theorems 5 and 6 we overcome all these difficulties. The key insight (Lemma 23) is that even though the connected components of $NT_k(G)$ can have arbitrarily large diameter, we can nevertheless guarantee that any two vertices in the same connected component are k'-near-twins, for k' depending only on k and the order of largest half-graph in G. Using this, we sparsify G as follows: For a suitably chosen k, we consider $NT_k(G)$ and create a partition $\mathcal{F}$ of $V(G)$ by putting two vertices in the same part if they are in the same connected component of $NT_k(G)$. The aforementioned Lemma 23 then guarantees that the vertices in the same part A of $\mathcal{F}$ are pairwise k'-near-twins. We then create a sparse graph $\mathcal{S}(G)$ from G as follows: If A and B are two large parts of $\mathcal{F}$ such that there are almost all edges between A and B (see the proof of Lemma 33 for precise meaning of "large" and "almost all"), we complement the adjacency between them, create new vertices v_A and v_B adjacent to all vertices of A and B, respectively, and create and edge between v_A and v_B. The introduction of new vertices v_A, v_B guarantees that we can recover G from $\mathcal{S}(G)$ by a simple interpretation. The technical part of the proof is establishing that $\mathcal{S}(G)$ comes from a fixed class $\mathcal{D}$ of graphs of tree rank 2 which depends only on $\mathcal{C}$.

Organisation of the paper

After preliminaries in the next section, we introduce vertex rankings in Section 3, and then we relate them to graph classes of bounded tree rank and show how to compute them in FPT runtime. In Section 4 we prove a lemma about the behaviour of the k-near-twin relation in

graphs excluding arbitrarily large half-graphs; a crucial tool for the last section. In Section 5 we prove our main results – characterization and a sparsification algorithm for graph classes interpretable in classes of tree rank 2.

Due to space restrictions, the proofs of some statements could not be included in the conference version of the paper. Such statements are marked with [∗] and their proofs can be found in the full version of the paper.

2 Preliminaries

Graph theory. We use $[k]$ to denote the set $\{1, \dots, k\}$. We use mostly standard graph theoretic notation. Let G be a graph. We write $N_r^G(v)$ for the *closed* r-neighborhood of a vertex v, that is, $N_r^G(v)$ is set of all vertices that are reachable from v by a path with at most r edges, including v. The *distance* between vertices u and v, denoted by $dist_G(u, v)$, is the minimum number of edges in a path between u and v. The *radius* of a graph is smallest integer r so that there exists a vertex that has distance at most r from every other vertex. Given a graph G and a set $X \subseteq V(G)$, we write $G[X]$ for the subgraph of G induced on X, and $G - X$ for the subgraph of G induced on $V(G) \setminus X$.

By a *tree* we mean a connected acyclic graph with a specified root vertex. The *depth* (respectively, *height*) of a tree T is the maximum number of edges (respectively, vertices) on any leaf-to-root path in T.

Let G be a graph, and let A and B be subsets of $V(G)$ with $A \cap B = \emptyset$ or $A = B$. By *flipping the edges between A and B*, we mean removing all edges uv in G with $u \in A$ and $v \in B$, and adding all new edges of the form uv where $u \in A$, $v \in B$, $u \neq v$, and $uv \notin E(G)$. For $k \in \mathbb{N}$ and a graph G, a *k-flip* of G is any graph that can be obtained from G by considering a partition $\mathcal{F}$ of $V(G)$ with $|\mathcal{F}| \leq k$, and flipping the edges between some pairs of parts of $\mathcal{F}$ (that is, for each pair of parts A and B of $\mathcal{F}$, we may choose whether or not to flip the edges between A and B). We say that a graph class $\mathcal{C}$ is a *perturbation* of a graph class $\mathcal{D}$ if there exists k such that every $G \in \mathcal{C}$ is a k-flip of some $H \in \mathcal{D}$.

Let S be a set of vertices of a graph G. Let $\mathcal{F}_S$ be the partition of $V(G)$ such that each $v \in S$ is in its own part and all vertices in $V(G) \setminus S$ are partitioned according to their adjacency to S (so vertices with the same neighbors in S are in the same part). An *S-flip* of G is any graph G' obtained by flipping the edges between some pairs of parts of $\mathcal{F}_S$.

Shallow topological minors and bounded tree rank. Let H be a graph. An *$\leq r$-subdivision* of H is any graph that can be obtained from H by replacing each edge uv of H by a path with endpoints u and v and with at most r internal vertices (so that all of the paths are internally disjoint). We call the original vertices of H the *principal* vertices of the $\leq r$-subdivision. We say that H is an *r-shallow topological minor*[2] of a graph G if G contains a subgraph that is isomorphic to an $\leq r$-subdivision of H.

▶ **Definition 7.** *We define $T_{d,m}$ to be the tree of depth d in which every non-leaf vertex has exactly m children.*

It is easily seen that the definition of graph classes of bounded tree rank given in Definition 1 is equivalent to the following:

[2] We note that this definition differs slightly from the standard definition, which says that H is a r-shallow minor of G if there is a subgraph of G isomorphic to a graph obtained from H by subdividing its edges at most $2r$ times.

▶ **Definition 8.** *A graph class $\mathcal{C}$ has tree rank at most d if for every $r \in \mathbb{N}$ there exists $m \in \mathbb{N}$ such that no $G \in \mathcal{C}$ contains $T_{d,m}$ as an r-shallow topological minor.*

Strong coloring numbers, admissibility, and bounded expansion. Let G be a graph, and let $\leq$ be an order on its vertex set. Fix a number $r \in \mathbb{N}$. For two vertices v and w of G, we say that w is *strongly r-reachable*[3] from v (with respect to $\leq$) if $w \geq v$ and there is a path from v to w of length at most r in G such that all vertices on this path apart from v and w are smaller than v in $\leq$. The *strong r-coloring number* of G, denoted by $\mathrm{scol}_r(G)$, is the minimum over all orderings $\leq$ of $V(G)$, of the maximum number of vertices that are strongly r-reachable from a single vertex v of G.

Similarly as above, let G be a graph, and let $\leq$ be an order on its vertex set. Fix a number $r \in \mathbb{N}$. The *r-backconnectivity* of a vertex v of G is the maximum number of paths of length at most r in G that start in v, end at vertices $w \geq v$, and are vertex-disjoint except for their common endpoint v. The *r-admissibility* of G, denoted by $\mathrm{adm}_r(G)$, is the minimum over all orderings $\leq$ of $V(G)$, of the maximum r-backconnectivity of a vertex v of G.

Classes of graphs of *bounded expansion* were introduced by Nešetřil and Ossona de Mendez as one of the key notions of their general theory of sparsity [17]. We do not provide the original definition of bounded expansion; instead we use the following characterizations.

▶ **Theorem 9** ([23])**.** *A class $\mathcal{C}$ of graphs has* bounded expansion *if and only if there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that for every $G \in \mathcal{C}$ and $r \in \mathbb{N}$, we have $\mathrm{scol}_r(G) \leq f(r)$.*

▶ **Theorem 10** ([7])**.** *A class $\mathcal{C}$ of graphs has* bounded expansion *if and only if there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that for every $G \in \mathcal{C}$ and $r \in \mathbb{N}$, we have $\mathrm{adm}_r(G) \leq f(r)$.*

Logic, interpretations and locality. We assume familiarity with first-order logic and basic notions related to it, such as signatures, quantifier rank, and so on. We model graphs as a structure with one binary irreflexive symmetric relation E. We work with colored graphs, and we model colors as unary predicates.

Interpretations are logic-based transformations that allow us to create new structures from old ones. A (simple) *interpretation* $I = (\psi, \delta)$ consists of two formulas $\psi(x, y)$ and $\delta(x)$. When applied to a graph G, an interpretation defines a new graph $I(G)$ with $V(I(G)) = \{v \in V(G) \ : \ G \models \delta(v)\}$ and $E(I(G)) = \{uv \ : \ u, v \in V(I(G)), u \neq v, \text{ and } G \models \psi(u, v)\}$. (Here we assume that $\psi(x, y)$ is symmetric and irreflexive, that is, for all G and $u, v \in V(G)$ we have $G \models \psi(u, v)$ if and only if $G \models \psi(v, u)$, and also for each $v \in V(G)$ we have that $G \not\models \psi(v, v)$ so that the resulting graph is undirected and does not contain loops.)

For a less general version of interpretation that uses only one formula $\psi(x, y)$, we sometimes use the notation $\psi(G)$ to denote the graph on the same vertex set as G and with $E(\psi(G)) := \{uv \ : \ u \neq v \text{ and } G \models \psi(u, v)\}$. For a graph class $\mathcal{C}$, we say that $\mathcal{C}$ is *interpretable* in a graph class $\mathcal{D}$ if there exists an interpretation I such that $\mathcal{C} \subseteq I(\mathcal{D})$.

A crucial role in this paper will be played by interpretations of bounded range. We say that a formula $\psi(x, y)$ is of *range* b if for all graphs G and all $u, v \in V(G)$ we have that $dist_G(u, v) > b$ implies $G \not\models \psi(u, v)$. This means that if the formula ψ is used in an interpretation, it will not create edges between vertices that were at distance more than b in the original graph. We say that an interpretation $I = (\psi, \delta)$ is of *bounded range* if there exists b such that ψ is of range b.

[3] We remark that many authors use the order $\leq$ in the opposite direction in the definition of strong reachability, that is, they require $w \leq v$ and that the path from v to w goes through vertices larger than v in $\leq$. The definition we chose will be convenient later on.

When working with graphs, we often mark vertices of a graph G with (new) unary predicates and call the resulting graph $\widehat{G}$. Formally, this means that we are extending the signature of G, and that the formulas that we later evaluate on $\widehat{G}$ are assumed to be over this new signature. To make the exposition more streamlined, we always do this implicitly.

Locality-based methods are commonly used in relation with first-order logic; probably the most commonly used such result is Gaifman's locality theorem [10]. We do not need the full statement of Gaifman's theorem, only the following lemma, which is its simple corollary.

▶ **Lemma 11.** *For every formula $\psi(x,y)$, there exist an integer r and a formula ψ' with the following property: Every graph G can be equipped with two unary predicates to obtain $\widehat{G}$ such that for any $u,v \in V(G)$ and any $S \subseteq V(G)$ that contains $N_r^G(u) \cup N_r^G(v)$, we have $G \models \psi(u,v) \iff \widehat{G}[S] \models \psi'(u,v)$.*

For completeness we explain how Lemma 11 follows from Gaifman's theorem. This theorem tells us that for a given formula $\psi(x,y)$ there exists another formula $\alpha(x,y)$ and sentences $\tau_1,\dots,\tau_k$ such that $\psi(x,y)$ can be written as a boolean combination of $\alpha(x,y),\tau_1,\dots,\tau_k$. Importantly, the formula $\alpha(x,y)$ has the property that there exists r such that for any G and any $u,v \in V(G)$ we have $G \models \alpha(u,v)$ if and only if $G[N_r^G(u) \cup N_r^G(v)] \models \alpha(u,v)$. For any G we can evaluate all sentences $\tau_1,\dots,\tau_k$ on G, and then the boolean combination of $\alpha(x,y),\tau_1,\dots,\tau_k$ reduces to one of four possibilities on G – we have that $\psi(x,y)$ is equivalent one of the following: $\alpha(x,y)$, $\neg\alpha(x,y)$, $\top$ or $\bot$. We can encode these four options with two bits of information, and so we mark all vertices of G with two unary predicates (all vertices in the same way) accordingly. Finally, we define formula $\psi'(x,y)$ to be the formula which first checks the unary predicates on vertex x to determine which of the four possibilities $\varphi(x,y)$ is equivalent to on G, and based on this "outputs" the value of $\alpha(x,y)$, $\neg\alpha(x,y)$, $\top$ or $\bot$. To finish the argument, we argue that evaluating $\alpha(x,y)$ on $\widehat{G}[S]$ gives the same answer as evaluating $\alpha(x,y)$ on G for any u,v,S such that $N_r^G(u) \cup N_r^G(v) \subseteq S$. This follows from the properties of α (which hold for any graph, including $\widehat{G}[S]$) which guarantee that $\widehat{G}[S] \models \alpha(u,v) \iff G[N_r^G(u) \cup N_r^G(v)] \models \alpha(u,v) \iff G \models \alpha(u,v)$.

3 Vertex rankings in sparse graphs

In this section we introduce a way to assign, for given parameters r and m, to every vertex of a graph its (r,m)-*rank*. Intuitively, the (r,m)-rank of a vertex measures how complicated the r-neighborhood of v is. Our definition is algorithmic and is given in Section 3.1 together with the proof that the (r,m)–rank can be computed in FPT runtime with respect to the parameters r and m. Then we prove Theorems 3 and 4 relating graph classes for which suitable rankings of vertices exist to graph classes of bounded tree rank (Section 3.2) and bounded expansion (Section 3.3).

3.1 The ranking algorithm

We now describe the ranking algorithm that (based on the parameters r and m) for every graph G assigns to every vertex of G either a positive integer or ∞.

Let G be a graph and r,m be positive integers. The (r,m)-ranking algorithm works as follows. Initially, each vertex is assigned rank ∞. Then the algorithm proceeds in rounds for $i = 1,2,3,\dots$ as follows. In the i-th round, the algorithm considers all vertices of rank ∞ (these are the vertices that have not received a finite rank in rounds $1,\dots,i-1$), and for each such vertex v it checks (in parallel) the following condition: Does there exist a set $S \subseteq V(G) \setminus \{v\}$ of size at most m such that $N_r^{G-S}(v) \setminus \{v\}$ contains only vertices of finite

rank? If yes, then v receives rank i, otherwise it keeps rank ∞. (Notice that in the first round the algorithm assigns rank 1 to vertices of degree at most m.) The algorithm stops when all vertices obtain a finite rank or when in some round no new vertex receives a rank. Thus, in any case, the algorithm stops after at most $|V(G)|$ rounds.

▶ **Definition 12.** *Let G be a graph. The* (r,m)-rank *of a vertex $v \in V(G)$ is the element of $\mathbb{N} \cup \{\infty\}$ assigned to v by the ranking algorithm. The* (r,m)-rank *of G is the maximum (r,m)-rank of any vertex of G.*

We remark that the definition of (r,m)-ranking of vertices in a graph G can be phrased without any explicit mention of the ranking algorithm: All vertices of degree at most m are assigned rank 1, and for every vertex v of degree more than m we define the (r,m)-rank of v to be the minimum number k such that there exists a set S of vertices so that $N_r^{G-S}(v) \setminus \{v\}$ contains only vertices of smaller rank; or ∞ if such k does not exist. This is easily seen to be equivalent to the algorithmic definition given above, but we prefer to stick with the algorithmic perspective in what follows.

3.1.1 Algorithmic considerations

Notice that the straightforward implementation of the ranking algorithm runs in time $\mathcal{O}(n^{m+4})$: we have at most n rounds, in each round we consider at most n vertices, and for each vertex we can in time $\mathcal{O}(n^{m+2})$ try removing all sets S of size m in time $\mathcal{O}(n^m)$ and check whether $G - S$ satisfies the desired condition in time $\mathcal{O}(n^2)$. Thus, we trivially get an XP algorithm. However, we can replace the subroutine which checks whether there exists a set S with $|S| \leq m$ with desired properties by a simple branch-and-bound procedure with FPT runtime.

▶ **Lemma 13.** *There is an algorithm with runtime $\mathcal{O}(r^m \cdot n^2)$ which correctly decides the following problem: Given G, $v \in V(G)$, $A \subseteq V(G) \setminus \{v\}$ and $r, m \in \mathbb{N}$ as input, decide whether there exist a set of set $S \subseteq V(G) \setminus \{v\}$ with $|S| \leq m$ such that $N_r^{G-S}(v) \cap A = \emptyset$.*

Proof. The algorithm checks whether there is a path of length at most r from v to any vertex in A. If no such path exists, the algorithm outputs YES. If such a path P exists and $m = 0$, the algorithm outputs NO. If $m > 0$ then at least one of the vertices on the path P has to be in the solution S, and so we can branch on all vertices $u \in V(P) \setminus \{v\}$ (there are at most r of them) and call the algorithm with input $G - \{u\}, v, A \setminus \{u\}$, r, $m - 1$. Then the algorithm returns YES if for at least one such u the recursive call returned YES, and returns NO otherwise.

We thus get an algorithm with branching bounded by r and depth bounded by m, and finding the path from v to A takes time $\mathcal{O}(n^2)$, so the claimed runtime follows. ◀

As an immediate consequence of the lemma we get the following.

▶ **Theorem 14.** *The (r,m)-ranking of any graph G can be computed in time $\mathcal{O}(r^m \cdot n^4)$.*

3.2 Vertex rankings and bounded tree rank

In this section we prove Theorem 3. The proof is split into Lemmas 15 and 16, which correspond to the two directions of the theorem.

▶ **Lemma 15.** *Let G be a graph that contains $T_{d,m+1}$ as an r-shallow topological minor. Then G has (r,m)-rank more than d.*

Proof. Let T be a subgraph of G that corresponds to an $\leq r$-subdivision of $T_{d,m+1}$. We prove that the principal vertices of T of height i (where the height is measured in $T_{d,m+1}$, and we think of leaves as having height 0) have (r,m)-rank at least $i+1$ in G. For $i=0$, the leaves of T clearly have rank at least 1. For $i>1$, let v be a vertex of T corresponding to a vertex of height i in $T_{d,m+1}$. Then there are $m+1$ internally disjoint paths connecting v to the corresponding $m+1$ children that have rank at least i. Therefore we cannot disconnect v from all of these children by deleting m vertices other than v. Thus v has (r,m)-rank at least $i+1$, which completes the proof. ◀

▶ **Lemma 16.** *Fix any $d,r\in\mathbb{N}$. For every m there exists $m'=m'(d,r,m)$ such that every graph G with (r,m')-rank more than d contains $T_{d,m}$ as an r-shallow topological minor.*

Proof. We will prove the lemma by induction on d. Actually, we will prove a slightly stronger statement – that for a suitably defined m' we have that every vertex of (r,m')-rank more than d is the root of an $\leq r$-subdivision of $T_{d,m}$ in G.

For $d=1$ we can set $m'=m-1$. Then if there exists a vertex of (r,m')-rank more than 1 in G, this vertex has at least m neighbors, and thus is the root of a $T_{1,m}$ subgraph in G. For $d>1$ we proceed as follows. For a given m, we wish to define a suitable m'. First, let $W_{d,m,r}$ denote the number of vertices of the graph obtained from $T_{d-1,m}$ by subdividing each edge r times. For convenience, set $M=m\cdot W_{d,m,r}+r\cdot m+m$. Let m'' be the number obtained by the inductive assumption applied to $d-1$, r, and M. Then we define $m':=\max\{m'',r\cdot m\}$. Let v be a vertex of (r,m')-rank more than d in G. Then for every set $S\subseteq V(G)\setminus\{v\}$ of size at most m', the set $N_r^{G-S}(v)\setminus\{v\}$ contains at least one vertex of (r,m')-rank at least d.

We claim that there exist paths $P_1,P_2,\ldots,P_m$ of length at most r that are disjoint other than at v, and so that each path P_i joins v to a vertex $u_i\neq v$ that has (r,m')-rank at least d. We find the paths greedily by adding one path at a time. Suppose that so far we have found the paths $P_1,\ldots,P_j$ for some $j<m$. Let S be the set of all vertices $u\neq v$ that are in any of the paths $P_1,\ldots,P_j$. This set S contains at most r vertices from each of the paths, and thus S has size at most $r(m-1)\leq m'$. Thus the set $N_r^{G-S}(v)\setminus\{v\}$ contains at least one vertex of (r,m')-rank at least d, and we can add another path P_{j+1} to our collection. This proves the claim. We note that this argument amounts to a Menger-type result about short paths, and this type of result has previously appeared in [16].

Since $m'\geq m''$, the (r,m'')-rank of each vertex of G is at least its (r,m')-rank. So by the inductive assumption applied to $d-1$, r, and M, each vertex u_i is the root of an $\leq r$-subdivision of $T_{d-1,M}$. Let us call this $\leq r$-subdivision rooted at u_i by T_i.

We now greedily build up the desired r-shallow topological minor of $T_{d,m}$ rooted at v as follows. Suppose that for some $j<m$, we have found that $T_1,\ldots,T_j$ contain, respectively, subgraphs $T_1',\ldots,T_j'$ so that

- for every $i\in\{1,\ldots,j\}$, the subgraph T_i' of T_i is an $\leq r$-subdivision of $T_{d-1,m}$ that is rooted at u_i,
- the subgraphs $T_1',\ldots,T_j'$ are pairwise vertex-disjoint, and
- for every $i\in\{1,\ldots,j\}$, the vertex u_i is the only vertex that is in both T_i' and in any of the paths $P_1,\ldots,P_m$.

We begin this greedy process with $j=0$. Now suppose that it holds for some $j<m$. We will prove it for $j+1$. This will complete the proof of Lemma 16.

So, we need to find a subgraph T_{j+1}' of T_{j+1} so that T_{j+1}' is an $\leq r$-subdivision of $T_{d-1,m}$ that is rooted at u_{j+1}, and the only vertex in common between T_{j+1}' and any of $T_1',\ldots,T_j'$ or $P_1,\ldots,P_m$ is u_{j+1}. Notice that each of the trees $T_1',\ldots,T_j'$ has at most $W_{d,m,r}$ vertices by the definition of $W_{d,m,r}$. Moreover, each of the paths $P_1,\ldots,P_m$ contributes at most r

additional vertices. So, since $j < m$, the number of vertices in any of $T'_1, \ldots, T'_j$ or $P_1, \ldots, P_m$ is at most $m \cdot W_{d,m,r} + r \cdot m$. Recall from the definition that M is this quantity plus m. So we can just delete the branches of T_{j+1} in which any of the vertices of $T'_1, \ldots, T'_j$ or $P_1, \ldots, P_m$, other than u_{j+1}, occur. In this manner we arrive at the desired subgraph T'_{j+1}. ◀

3.3 Vertex rankings and bounded expansion

In this section we prove Theorem 4, characterizing classes of bounded expansion using vertex rankings. The theorem follows from the two lemmas below combined with Theorems 9 and 10.

▶ **Lemma 17.** *Let G be a graph with $\mathrm{scol}_r(G) = m$. Then G has finite $(r, m-1)$-rank.*

We stress that in Lemma 17 we are not bounding the maximum value of $(r, m-1)$-rank of any vertex of G, but merely claiming that the $(r, m-1)$-ranking algorithm will assign to each vertex of G a finite value. This value, however, can be arbitrarily large (and in particular it is not bounded in terms of r and m). This can be seen already on the class of trees – the class of all trees has bounded strong coloring numbers (one can set $m := r + 1$), but on the complete m-ary tree of depth d the $(r, m-1)$-ranking algorithm will need d rounds to finish.

Proof of Lemma 17. Assume for contradiction that there exists a vertex of G of $(r, m-1)$-rank ∞. Fix an order $\leq$ on $V(G)$ that certifies that $\mathrm{scol}_r(G) = m$. Let v be the smallest vertex of G with respect to $\leq$ that has $(r, m-1)$-rank ∞. Let A be the set of all vertices below v in $\leq$, and let i be the maximum $(r, m-1)$-rank of any vertex in A. We claim that in round $(i+1)$ of the ranking algorithm vertex v received rank (meaning it has rank $i+1$), which is a contradiction with our assumption on v. To see this, let S be the set of all vertices that are strongly r-reachable from v in G. Note that every path of length at most r from v to a vertex $w \geq v$ must contain a vertex in S. It follows that $N_r^{G-S}(v) \setminus \{v\}$ contains only vertices of finite rank, a contradiction. ◀

Lemma 17 guarantees that the ranking algorithm succeeds on graphs with bounded strong coloring numbers. The next lemma tells us that the ranking algorithm computes a good admissibility ordering for an input graph G.

▶ **Lemma 18.** *Let $r, m \in \mathbb{N}$ with $r \geq 1$. Let G be a graph with finite (r, m)-rank. Then $\mathrm{adm}_r(G) \leq m$.*

Proof. Let $\leq$ be an ordering of $V(G)$ so that if $v \leq w$, then the (r, m)-rank of v is at most the (r, m)-rank of w. Thus we are ordering vertices of G by their rank, breaking ties arbitrarily. We claim that this ordering certifies that $\mathrm{adm}_r(G) \leq m$. Let v be any vertex of G, and let i be its (r, m)-rank. Let S be the set of vertices certifying that the (r, m)-rank of v is i. That is, S is a subset of $V(G) \setminus \{v\}$ of size at most m so that $N_r^{G-S}(v) \setminus \{v\}$ contains only vertices of (r, m)-rank strictly less than i.

Since any vertex $w \geq v$ has (r, m)-rank at least i, any path from v that ends in a vertex larger than v with respect to the ordering must contain a vertex of S. Therefore, if we consider a collection $P_1, \ldots, P_\ell$ of paths that maximizes the r-backconnectivity of v, then each P_i has to intersect S. Since all of these paths are disjoint except for their common end v, we get that $\mathrm{adm}_r(G) \leq m$, as desired. ◀

4 A lemma about edge-stable graphs

In this section we state and prove a lemma (Lemma 23) about the behaviour of the so-called k-near-twin relation on graphs that do not contain large half-graphs. This lemma will be crucial in the next sections and may be of independent interest in the theory of (monadically) stable graphs [20, 21] (see also [4] for the latest important developments in this area).

First we need a few definitions. A crucial role will be played by the notion of k-near-twins.

▶ **Definition 19.** *Let G be a graph, and let $k \in \mathbb{N}$. We say that two vertices are* k-near-twins *if $|N^G(v) \Delta N^G(u)| \leq k$.*

▶ **Definition 20.** *Let G be a graph, and let $k \in \mathbb{N}$. The* k-near-twin graph of G, *denoted by $NT_k(G)$, is the graph with vertex set $V(G)$ in which there is an edge between two vertices u and v if they are k-near-twins in G.*

▶ **Definition 21.** *A* half-graph of order t *is a graph with vertex set $\{u_1, \ldots, u_t, w_1, \ldots, w_t\}$ such that there is an edge between u_i and w_j if and only if $i \leq j$.*

▶ **Definition 22.** *We say that a graph G contains a half-graph of order t as a* semi-induced subgraph *if there are distinct vertices $u_1, \ldots, u_t, w_1, \ldots, w_t$ in G such that there is an edge between u_i and w_j if and only if $i \leq j$.*

Note that in the definition we do not say anything about edges between the vertices within the set $\{w_1, \ldots, w_t\}$ or edges between the vertices within the set $\{u_1, \ldots, u_t\}$; the edges within these sets can be arbitrary.

Our main lemma in this section is the following.

▶ **Lemma 23** (∗). *There is a function $h : \mathbb{N}^2 \to \mathbb{N}$ so that for any $k, t \in \mathbb{N}$, if G is a graph with no half-graph of order t as a semi-induced subgraph, and u and v are vertices in the same component of the k-near-twin graph of G, then u and v are $h(k, t)$-near-twins in G.*

The technical proof of the lemma can be found in the full version of the paper.

5 Interpretations of graph classes of tree rank 2

In this section we prove Theorems 5 and 6. First we will establish some properties of classes of tree rank at most 2 (Section 5.1). After this, in Section 5.2, we give a characterization of graph classes interpretable in graph classes of tree rank at most 2 by interpretations of *bounded range*. This section contains the technical core of our results – the key result is Theorem 31, proof of which is split into Lemmas 32 and 33. The case of general interpretations (Theorem 5) is presented in Section 5.3; it will easily follow from results in Section 5.2. Finally, we prove our main algorithmic result, Theorem 6, in Section 5.4.

5.1 Classes of tree rank at most 2

We start with a characterization of graph classes of tree rank 2.

▶ **Definition 24.** *A class of graphs $\mathcal{C}$ has* locally almost[4] bounded degree *if there exist functions $f, d : \mathbb{N} \to \mathbb{N}$ such that for every $r \in \mathbb{N}$, every $G \in \mathcal{C}$, and every $v \in V(G)$, the set $N_r^G(v)$ contains at most $f(r)$ vertices with degree larger than $d(r)$ in G.*

[4] We remark that the adjective "almost" is sometimes used differently in context of structural graph parameters. For example, for a graph class $\mathcal{C}$, having almost bounded flipwidth (see [22]) means that for every ϵ there exists c such that every $G \in \mathcal{C}$ has flipwidth at most cn^ϵ. Our usage of the adjective "almost" is different – it refers to having a bounded number of exceptions.

▶ **Lemma 25.** *A class $\mathcal{C}$ of graphs has tree rank at most 2 if and only if has locally almost bounded degree.*

In the proof of the lemma we will use the following simple result from [13, Lemma 13].

▶ **Lemma 26.** *Let $r, t \in \mathbb{N}$, let H be a graph of radius at most r, and let $S \subseteq V(H)$. If $|S| \geq t^r + 1$, then there exists $u \in V(H)$ such that there are t vertices in S that are distinct from u and can be reached from u by internally vertex-disjoint paths of length at most r.*

Proof of Lemma 25. By Theorem 3, it suffices to prove that a class $\mathcal{C}$ is of locally almost bounded degree if and only if for every $r \in \mathbb{N}$ there exists $m \in \mathbb{N}$ such that every graph in $\mathcal{C}$ has (r, m)-rank at most 2.

First, assume that $\mathcal{C}$ is of locally almost bounded degree with respect to functions f and d. Let $r \in \mathbb{N}$, and set $m := \max\{f(r), d(r)\}$. We claim that the (r, m)-ranking algorithm will assign the number 1 or 2 to each $v \in V(G)$. In the first round, all vertices of degree at most $d(r) \leq m$ will get rank 1. In the second round, we know that each $N_r^G(v)$ contains at most $f(r)$ vertices of degree larger than $d(r)$. Thus we can delete a set $S \subseteq V(G) \setminus \{v\}$ of size at most $f(r)$ so that $N_r^{G-S}(v) \setminus \{v\}$ contains only vertices of rank 1. The result follows.

For the other direction, suppose that for every $r \in \mathbb{N}$ there exists $m \in \mathbb{N}$ such that every graph in $\mathcal{C}$ has (r, m)-rank at most 2. Let $r \in \mathbb{N}$, and set $f(r) := (m+1)^r$ and $d(r) := m$. Let $v \in V(G)$ be arbitrary, and let S be the set of vertices with (r, m)-rank exactly 2 in $N_r^G(v)$ (these are the vertices of degree more than m). Going for a contradiction, we may assume that $|S| > f(r)$, since otherwise we are done. Thus, by Lemma 26 applied to r, $t := m + 1$ and S in the subgraph H of G induced on $N_r^G(v)$, there exists a vertex $u \in N_r^G(v)$ such that there are $m+1$ vertices in S that are distinct from u and can be reached from u by internally vertex-disjoint paths of length at most r. As all of the vertices in S have (r, m)-rank 2, this shows that the (r, m)-rank of u is more than 2, which is a contradiction. Thus $\mathcal{C}$ is of locally almost bounded degree with functions f and d, as desired. ◀

Algorithmic considerations. For our algorithms we will need to assume that the functions which are used in the definitions of our graph classes are efficiently computable. In particular, we will need to be able to efficiently test whether a given graph G comes from a fixed graph class of locally almost bounded degree given by functions d and f. This can be done under fairly relaxed conditions on functions d and f, which we now describe. The key observation is that for any graph G we need to check the conditions imposed by functions f and d from Definition 24 only for values of r with $r \leq n$ (where $n = |V(G)|$), and that checking the conditions is trivial whenever $f(r) > n$ or $d(r) > n$.

With this in mind, we say that a function $h : \mathbb{N} \to \mathbb{N}$ is *nice* if there exists a an algorithm which inputs two numbers r and n represented in binary with $r \leq n$ and in time $poly(n)$ either correctly answers that $h(r) > n$ or outputs $h(r)$ (which is upper bounded by n).

Note that a function which is very fast growing and complicated to compute (with respect to its input which has length $\lceil \log(r) \rceil$) can still be nice. This is because it may be easy to check that $h(r) > n$, in which case no further computation is required, and if $h(r) \leq n$, we know that n is much larger than $\lceil \log(r) \rceil$, and then we have $poly(n)$ time at our disposal to compute $h(r)$. It is easy to verify that functions such as 2^r, $\mathrm{tow}_\ell(r)$ (tower of twos of height ℓ with r on top), $\mathrm{tow}_r(2)$ (tower of twos of height r) and also r^r are nice functions.

Coming back to graph classes of tree rank 2, we will use the following definition to specify graph classes for which we can prove Theorem 6.

▶ **Definition 27.** *We say that a class $\mathcal{C}$ of graphs of tree rank 2 is* efficiently bounded *if there exists a nice function $g : \mathbb{N} \to \mathbb{N}$ such that for every $r \in \mathbb{N}$ there exists $m \in \mathbb{N}$ such that no $G \in \mathcal{C}$ contains $T_{2,m}$ as an r-shallow topological minor.*

By inspecting the proof of Lemma 25 one easily checks that any efficiently bounded class $\mathcal{C}$ of tree rank 2 is a class of locally almost bounded degree for which the there exist functions f and d which ceritfy this. If we moreover assume that $\mathcal{C}$ is *maximum* class of locally bounded degree with respect to f and g (meaning it contains every G that satisfies conditions given by f and d for all $r \leq |V(G)|$), then we can efficiently test for any G whether $G \in \mathcal{C}$. This is easily achieved by testing the conditions from Definition 24 for all values r up to $|V(G)|$. Since the functions f and d are nice, these values are either too large (if $f(r) > n$ or $d(r) > n$ which can be efficiently checked), or efficiently computable. This yields the following lemma.

▶ **Lemma 28.** *Let $\mathcal{C}$ be a class of graphs with locally almost bounded degree given by nice functions $f, d : \mathbb{N} \to \mathbb{N}$. Assume that $\mathcal{C}$ is maximum such class with respect to f and d. Then there is a polynomial time algorithm that determines whether $G \in \mathcal{C}$.*

In subsequent sections, it would be desirable to show at various places that some function is nice (or at least can be upper bounded by some nice function). We will not spell out these arguments explicitly; we just note here that one can often combine two nice functions to obtain another nice function. In particular, one easily checks that if f and g are functions such that for all r we have $f(r) \geq r$ and $g(r) \geq r$, then also $h := g \circ f$ is nice with $h(r) \geq r$. This because for given r, n with $r \leq n$ we can check whether $f(r) > n$, and if yes then we know that $h(r) = g(f(n)) > n$. On the other hand if $f(r) \leq n$, then the input condition for function g is satisfied, and we can just check whether $g(f(r)) > n$ or compute the value of $g(f(r))$ efficiently.

5.2 The case of interpretations of bounded range

We now proceed by giving a characterization of graph classes that can be obtained from graph classes of tree rank at most 2 by an interpretation of bounded range (Theorem 31).

In what follows we will use the notions of k-near-twins and near-twin graphs introduced in Section 4.

▶ **Definition 29.** *We say that a graph G is* (k, m)-near-covered *if for every set $S \subseteq V(G)$ such that the elements of S are mutually not k-near-twins we have $|S| \leq m$.*

We remark that the definition of (k, m)-near-covered graphs introduced in [12] was slightly different – there it was required that there exists a set of at most m vertices such that every vertex v of G is a k-near-twin of some $w \in S$. One can easily check that the two definitions are functionally equivalent (i.e. up to changing k and m in one definition to k' and m' in the other definition). We use Definition 29 because it will be more convenient in our arguments.

▶ **Definition 30.** *A graph class $\mathcal{C}$ is* locally almost near-covered *if there exist functions $k, m : \mathbb{N} \to \mathbb{N}$ such that for every $r \in \mathbb{N}$, every $G \in C$, and every $v \in V(G)$, the subgraph of G induced on $N_r^G(v)$ is $(k(r), m(r))$-near-covered.*

▶ **Theorem 31.** *A class $\mathcal{C}$ is a interpretable from a class of locally almost bounded degree by a bounded range interpretation if and only if $\mathcal{C}$ is locally almost near-covered.*

The forward direction of Theorem 31 is handled by the following lemma, which can be proven by combining the locality-based Lemma 11 with the characterization of graph classes interpretable in graph classes of bounded degree in terms of near-covered graphs given in [12].

▶ **Lemma 32** (∗). *Let I be a bounded range interpretation, and let $\mathcal{C}$ be a class of locally almost bounded degree. Then $I(\mathcal{C})$ is locally almost near-covered.*

The rest of this section is dedicated to proving the backward direction of Theorem 31

▶ **Lemma 33.** *Let $\mathcal{C}$ be a locally almost near-covered graph class. Then $\mathcal{C}$ is interpretable in a class $\mathcal{D}$ of locally almost bounded degree by an interpretation I of bounded range.*

Moreover, there is a polynomial algorithm that to every graph G computes a graph $\mathcal{S}(G)$ so that if $G \in \mathcal{C}$, then $\mathcal{S}(G) \in \mathcal{D}$ and $G = I(\mathcal{S}(G))$.

We will need the following simple lemma showing that locally almost near-covered graph classes do not contain arbitrarily large half-graphs.

▶ **Lemma 34.** *Let $\mathcal{C}$ be a locally almost near-covered graph class with respect to functions k and m, and set $t := m(2) \cdot k(2) + m(2) + 1$. Then no graph in $\mathcal{C}$ contains a half-graph of order t as a semi-induced subgraph.*

Proof. Assume for contradiction that such a graph $G \in \mathcal{C}$ exists. Let $u_1, \ldots, u_t, w_1, \ldots, w_t$ be the vertices of a half-graph of order t that G contains as a semi-induced subgraph. Then all of these vertices $u_1, \ldots, u_t, w_1, \ldots, w_t$ are in $N_2^G(u_1)$. Also, for any $i, j \in \{1, 2, \ldots, t\}$ with $i - j > k(2)$, we know that u_i and u_j are not $k(2)$-near twins, since their adjacency differs in vertices $w_j, w_{j+1}, \ldots, w_{i-1}$. Therefore, the vertices $u_1, u_{k(2)+2}, u_{2k(2)+3}, \ldots u_{m(2)\cdot k(2)+m(2)+1}$ form a set of $m(2) + 1$ pairwise not $k(2)$-near-twins, a contradiction with $\mathcal{C}$ being locally almost near-covered. ◀

In the proof of Lemma 33 we will also use the following lemma taken from [12, Corollary 5.3]. We note that in that paper, an extra assumption was made that amounts to saying that no vertex in A is a k-near-twin of any vertex in B. However, that assumption was not used in the proof given in [12], and so same proof works to prove the following lemma.

▶ **Lemma 35.** *Let G be a graph and let A and B be two subsets of $V(G)$ that are either disjoint or have $A = B$ and such that $|A| \geq 5k + 1$, $|B| \geq 5k + 1$, all vertices in A are pairwise k-near-twins, and all vertices in B are pairwise k-near-twins. Then either*

1. *every vertex of A is adjacent to at most $2k$ vertices of B and every vertex of B is adjacent to at most $2k$ vertices of A, or*
2. *every vertex of A is adjacent to all but at most $2k$ vertices of B and every vertex of B is adjacent to all but at most $2k$ vertices of A*

We will also use the following immediate corollary of Lemma 35.

▶ **Corollary 36.** *Let G be a graph, and let A and B be two subsets of $V(G)$ that satisfy the assumptions of Lemma 35. If there exists a vertex $v \in A$ with $|N^G(v) \cap B| > 2k$, then in the graph G' obtained from G by flipping the edges between A and B, every vertex $u \in A$ has $|N^{G'}(u) \cap B| \leq 2k$.*

Finally, we will use the following Ramsey-type result, which is routine to prove.

▶ **Lemma 37.** *There exists functions $R, D : \mathbb{N}^3 \to \mathbb{N}$ so that for all $k, m, s \in \mathbb{N}$, the following holds for every graph G with no subgraph isomorphic to the complete bipartite graph $K_{s,s}$. If $X \subseteq V(G)$ is such that $|X| \geq R(k, m, s)$ and every vertex in X has degree at least $D(k, m, s)$, then there exists $Y \subseteq X$ such that $|Y| \geq m$ and every vertex $y \in Y$ has at least k neighbors in $V(G) \setminus Y$ that are not adjacent to any other $y' \in Y \setminus \{y\}$.*

Proof of Lemma 33. Let $\mathcal{C}$ be a locally almost near-covered graph class, and let k and m be the functions certifying this. We will describe a procedure $\mathcal{S}$ that to every $G \in \mathcal{C}$ assigns a graph $\mathcal{S}(G)$ such that $G = I(\mathcal{S}(G))$ for a fixed interpretation I of bounded range. This procedure will also return some graph $\mathcal{S}(G)$ even when G is not in $\mathcal{C}$, however in this case we make no guarantees about the graph $\mathcal{S}(G)$. We will then show that the graph class $\mathcal{D} := \{\mathcal{S}(G) \ : \ G \in \mathcal{C}\}$ is of locally almost bounded degree. Since by construction we will have $\mathcal{C} \subseteq I(\mathcal{D})$, this will finish the proof.

Let $G \in \mathcal{C}$ be arbitrary. Let $h := h(k(3), t)$, where h is the function from Lemma 23 and $t := m(2) \cdot k(2) + m(2) + 1$ is the integer from Lemma 34. Thus every pair of vertices in the same component of the $k(3)$-near-twin graph of G are h-near-twins in G. We define a partition $\mathcal{F}$ of $V(G)$ by putting two vertices into the same part if they are in the same connected component of the $k(3)$-near-twin graph of G (this creates a partition since being in the same connected component is an equivalence relation). We call parts $A, B \in \mathcal{F}$ (possibly with $A = B$) *mutually heavy* if $|A| \geq 5h + 1$, $|B| \geq 5h + 1$, and there exists a vertex $v \in B$ with $|N^G(v) \cap A| > 2h$. (Note that in this case, by Lemma 35, there also exists a vertex $u \in A$ with $|N^G(u) \cap B| > 2h$.) Similarly, we call a part $A \in \mathcal{F}$ *heavy* if it is mutually heavy with any part $B \in \mathcal{F}$ (possibly with $B = A$).

Creating the sparse graph $\mathcal{S}(G)$ from G. We can now describe the sparse graph $\mathcal{S}(G)$ associated to G. We start from G and proceed as follows:

1. For any heavy part $A \in \mathcal{F}$, we introduce a new vertex v_A, make it adjacent to all vertices in A, and mark it with a unary predicate R.
2. For any mutually heavy parts $A, B \in \mathcal{F}$, we flip between A and B. If $A = B$, then we mark v_A with a unary predicate F. If $A \neq B$, then we put an edge between v_A and v_B.

All other adjacencies that were not part of any flip remain as in G. Note that we can define this graph $\mathcal{S}(G)$ even when G is not in $\mathcal{C}$. Also note that by Corollary 36, for all mutually heavy parts $A, B \in \mathcal{F}$ and every vertex $v \in A$, we have that $|N^{\mathcal{S}(G)}(v) \cap B| \leq 2h$. We will use this important property later on.

Algorithmic considerations. It is easily seen that the construction of $\mathcal{S}(G)$ can be done in polynomial time from G.

Recovering G from $S(G)$ by an interpretation. We now show that there is an interpretation I such that $G = I(\mathcal{S}(G))$. Let u and v be two vertices from $V(G)$. In the construction of $\mathcal{S}(G)$, the adjacency between two vertices only changed in the second part of the construction. Thus, the formula $\psi(x, y)$ will complement the adjacency between distinct vertices x and y if

- x and y have the same (unique) neighbor w marked with the predicate R and this w is also marked with predicate F, or
- x and y each have a different neighbor marked with the predicate R and these neighbors are adjacent.

In all other cases we have $\psi(x, y) = E(x, y)$. The conditions above are easily expressed by a first-order formula. Note that $\psi(x, y)$ only keeps the existing edges or creates edges between vertices at distance at most 3 and therefore ψ is of range 3. To define the vertex set of G from the graph $\mathcal{S}(G)$, the formula $\delta(x)$ just keeps the vertices that are not marked with R (the original vertices of G).

Showing that the class $\mathcal{D} := \{\mathcal{S}(G) : G \in \mathcal{C}\}$ is of locally almost bounded degree. It remains to argue that $\mathcal{D}$ is of locally almost bounded degree. Due to space restrictions this argument is omitted from the conference version and can be found in the full version of the paper. ◀

5.3 The case of full interpretations

In this section we finish the proof of our main result, Theorem 5, which states that a class $\mathcal{C}$ of graphs is interpretable in a graph class of tree rank 2 if and only if $\mathcal{C}$ is a perturbation of a locally almost near-covered graph class.

For the forward direction, it is known (see for example [1, 3]) that every interpretation can decomposed into two steps in the following sense.

▶ **Lemma 38.** *For every interpretation I there exists an interpretation I' of bounded range and $k \in \mathbb{N}$ such that for every G we have that $I(G)$ is a k-flip of $I'(G)$.*

The forward direction of Theorem 5 then follows immediately from Lemmas 32 and 38. The backward direction follows from Lemma 33: If $\mathcal{C}$ is a perturbation of a locally almost near-covered graph class, then $\mathcal{C}$ is a perturbation of $I(\mathcal{D})$ for some graph class $\mathcal{D}$ of locally almost bounded degree. Since perturbations (k-flips) can be modelled by interpretations, add unary predicates to graphs from $\mathcal{D}$ and adjust I to I' such that $\mathcal{C} \subseteq I'(\mathcal{D})$.

5.4 Proof of Theorem 6

In this section we prove Theorem 6 by showing that for every graph class $\mathcal{C}$ interpretable in a graph class of tree rank 2, there is a polynomial time algorithm that to every $G \in \mathcal{C}$ computes $H \in \mathcal{D}$ such that $G = I(H)$, where I is a fixed interpretation and $\mathcal{D}$ is a fixed class of graphs of locally almost bounded degree (which is the same as tree rank 2 by Lemma 25).

We will rely on the following lemma which was proven in [1].

▶ **Lemma 39.** *Let $\mathcal{D}$ be an NIP class of graphs, and let I be an interpretation. Then there exist s, b, and a formula $\psi(x, y)$ of range b such that for every $G \in I(\mathcal{D})$ there exist $S \subseteq V(G)$ of size at most s, an S-flip G' of G, and a graph $H \in \mathcal{D}$ such that $G' = \psi(H)$.*

We do not define the notion of a graph class being NIP (see for example the relevant sections in [1]), but just note that this notion is very general and it is easily established that classes of tree rank 2 are NIP, so the lemma can be applied in our setting.

We now proceed with the proof Theorem 6. Since $\mathcal{C}$ is interpretable in a graph class of tree rank 2, there exist an interpretation I and a class $\mathcal{D}$ of tree rank at most 2 such that $\mathcal{C} \subseteq I(\mathcal{D})$. By applying Lemma 39 to $\mathcal{D}$ and I, we obtain numbers s and b and a formula ψ of range b. Since ψ is of bounded range, we know by Lemma 32 that the class $\psi(\mathcal{D})$ is locally almost near-covered. By Lemma 33 applied to $\psi(\mathcal{D})$, there exists a graph class $\mathcal{D}'$ of locally almost bounded degree, an interpretation I', and a polynomial time algorithm $\mathcal{A}$ that to any $G' \in \psi(\mathcal{D})$ computes a graph $\mathcal{S}(G') \in \mathcal{D}'$ such that $G' = I'(\mathcal{S}(G'))$. Moreover, by inspecting the proofs of Lemma 32 and Lemma 33 one can check that the functions f and d which certify that $\mathcal{D}'$ is a class of locally almost bounded degree are nice functions, and so we can use Lemma 28 to check membership in $\mathcal{D}'$.

We now consider the algorithm that for a graph $G \in \mathcal{C}$ does the following:

1. Go through all subsets S of $V(G)$ of size at most s.
2. For each such set S, go through all possible S-flips G' of G.
3. Apply the algorithm $\mathcal{A}$ from Lemma 33 to G' to obtain a graph $\mathcal{S}(G')$.
4. Check whether $G' = I'(\mathcal{S}(G'))$ and use Lemma 28 to check whether $\mathcal{S}(G') \in \mathcal{D}'$, if yes, then output $\mathcal{S}(G')$.

We now argue the correctness of the algorithm. By Lemma 39, at least one G' will be such that $G' = \psi(H)$ for some $H \in \mathcal{D}$. Then, since $G' \in \psi(\mathcal{D})$, we have (by Lemma 33) that the algorithm $\mathcal{A}$ returns a graph $\mathcal{S}(G')$ that is in $\mathcal{D}'$. Since for $\mathcal{S}(G')$ we have $G' = I'(\mathcal{S}(G'))$

and since from G' one can easily recover G by performing a bounded number of flips, we can adjust the interpretation I' to an interpretation I'' such that $G = I''(\mathcal{S}(G'))$, as desired. (We add unary predicates to $\mathcal{S}(G')$ in order to "mark" the sets that we flip between.)

The runtime of the algorithm is easily seen to be polynomial in $|V(G)|$: We have $|V(G)|^s$ possible sets of size s and consequently we have $|V(G)|^s$ iterations, where the number s depends only on the graph class $\mathcal{C}$. In each iteration we invoke the polynomial time algorithm $\mathcal{A}$ from Lemma 33.

6 Conclusions

We conclude with two open ended questions that may deserve further attention.

1. Is there a way of defining vertex rankings for dense graphs analogous to our rankings? Ideally, such rankings should be computable in FPT time.
2. While the proof of Lemma 33 is technical, our construction of the graph $\mathcal{S}(G)$ from the graph G is simple: First we determine which components of the k-near-twin graph of G to flip between, and then we use this information to construct $\mathcal{S}(G)$. It may be interesting to see under which conditions on G we can claim that the graph $\mathcal{S}(G)$ comes from a class of sparse graphs. Also, using our construction recursively may lead to interesting results: If $\mathcal{S}(G)$ is not a sparse graph, one may consider $\mathcal{S}(\mathcal{S}(G))$, and so on.

References

1 Édouard Bonnet, Jan Dreier, Jakub Gajarský, Stephan Kreutzer, Nikolas Mählmann, Pierre Simon, and Szymon Toruńczyk. Model checking on interpretations of classes of bounded local cliquewidth. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '22, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3531130.3533367.

2 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Toruńczyk. Twin-width iv: Ordered graphs and matrices. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 924–937, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3520037.

3 Samuel Braunfeld, Jaroslav Nesetril, Patrice Ossona de Mendez, and Sebastian Siebertz. Decomposition horizons: from graph sparsity to model-theoretic dividing lines. *CoRR*, abs/2209.11229, 2022. doi:10.48550/arXiv.2209.11229.

4 Jan Dreier, Ioannis Eleftheriadis, Nikolas Mählmann, Rose McCarty, Michał Pilipczuk, and Szymon Toruńczyk. First-order model checking on monadically stable graph classes, 2023. arXiv:2311.18740.

5 Jan Dreier, Nikolas Mählmann, and Sebastian Siebertz. First-order model checking on structurally sparse graph classes. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 567–580, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3564246.3585186.

6 Jan Dreier, Nikolas Mählmann, Sebastian Siebertz, and Szymon Toruńczyk. Indiscernibles and Flatness in Monadically Stable and Monadically NIP Classes. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 125:1–125:18, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2023.125.

7 Zdeněk Dvořák. Constant-factor approximation of the domination number in sparse graphs. *European Journal of Combinatorics*, 34(5):833–840, 2013.

8 Kord Eickmeyer and Ken-ichi Kawarabayashi. FO model checking on map graphs. In Ralf Klasing and Marc Zeitoun, editors, *Fundamentals of Computation Theory – 21st International Symposium, FCT 2017, Bordeaux, France, September 11-13, 2017, Proceedings*, volume 10472 of *Lecture Notes in Computer Science*, pages 204–216. Springer, 2017. doi:10.1007/978-3-662-55751-8_17.

9 Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Annals of Pure and Applied Logic*, 130(1-3):3–31, 2004.

10 Haim Gaifman. On local and non-local properties. *Studies in Logic and the Foundations of Mathematics*, 107:105–135, 1982.

11 Jakub Gajarský and Petr Hlinený. Kernelizing MSO properties of trees of fixed height, and some consequences. *Log. Methods Comput. Sci.*, 11(1), 2015. doi:10.2168/LMCS-11(1:19)2015.

12 Jakub Gajarský, Petr Hlinený, Jan Obdrzálek, Daniel Lokshtanov, and M. S. Ramanujan. A new perspective on FO model checking of dense graph classes. *ACM Trans. Comput. Log.*, 21(4):28:1–28:23, 2020. doi:10.1145/3383206.

13 Jakub Gajarský, Michał Pilipczuk, Marek Sokołowski, Giannos Stamoulis, and Szymon Toruńczyk. Elementary first-order model checking for sparse graphs, 2024. arXiv:2401.16230.

14 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

15 Michael Lampis. First Order Logic on Pathwidth Revisited Again. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 132:1–132:17, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2023.132.

16 L. Lovász, V. Neumann-Lara, and M. Plummer. Mengerian theorems for paths of bounded length. *Periodica Mathematica Hungarica*, 9(4):269–276, 1978.

17 Jaroslav Nešetřil and Patrice Ossona De Mendez. *Sparsity: graphs, structures, and algorithms*, volume 28. Springer Science & Business Media, 2012.

18 Jaroslav Nesetril, Patrice Ossona de Mendez, Michal Pilipczuk, Roman Rabinovich, and Sebastian Siebertz. Rankwidth meets stability. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 2014–2033. SIAM, 2021. doi:10.1137/1.9781611976465.120.

19 Jaroslav Nešetřil, Roman Rabinovich, Patrice Ossona de Mendez, and Sebastian Siebertz. Linear rankwidth meets stability. In *31st ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 1180–1199. SIAM, 2020. doi:10.1137/1.9781611975994.72.

20 Saharon Shelah. Stability, the f.c.p., and superstability; model theoretic properties of formulas in first order theory. *Annals of Mathematical Logic*, 3(3):271–362, 1971.

21 Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1):247–261, 1972.

22 Szymon Toruńczyk. Flip-width: Cops and robber on dense graphs. *FOCS 2023, accepted*, arXiv abs/2302.00352, 2023.

23 Xuding Zhu. Colouring graphs with bounded generalized colouring number. *Discrete Mathematics*, 309(18):5562–5568, 2009.

Deciding Linear Height and Linear Size-To-Height Increase of Macro Tree Transducers

Paul Gallot
Universität Bremen, Germany

Sebastian Maneth
Universität Bremen, Germany

Keisuke Nakano
Tohoku University, Sendai, Japan

Charles Peyrat
ENS Paris-Saclay, France

Abstract

We present a novel normal form for (total deterministic) macro tree transducers (mtts), called "depth proper normal form". If an mtt is in this normal form, then it is guaranteed that each parameter of each state appears at arbitrary depths in the output trees of that state. Intuitively, if some parameter only appears at certain bounded depths in the output trees of a state, then this parameter can be eliminated by in-lining the corresponding output paths at each call site of that state. We use regular look-ahead in order to determine which of the paths should be in-lined. As a consequence of changing the look-ahead, a parameter that was previously appearing at unbounded depths, may be appearing at bounded depths for some new look-ahead; for this reason, our construction has to be iterated to obtain an mtt in depth-normal form. Using the normal form, we can decide whether the translation of an mtt has linear height increase or has linear size-to-height increase.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases automata, formal language theory, macro tree transducer, normal form

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.138

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2307.16500` [18]

Funding This work was partially supported by JSPS KAKENHI Grant Numbers 21K11744 and 22H00520.

Acknowledgements We thank the anonymous reviewers of a previous version of this paper for their critical comments.

1 Introduction

Tree transducers are fundamental devices in theoretical computer science. They generalize the finite state transductions from strings to (finite, ranked) trees and were invented in the 1970s in the context of compiler theory and mathematical linguistics. The most basic such transducers are the top-down tree transducer [24, 23] and the bottom-up tree transducer [25], see also [6]. These transducers traverse their input tree once, but may process subtrees in several copies. It is well known that these transducers have *linear height increase* ("LHI"), see e.g. [17].

In this paper we deal with a more powerful kind of tree transducer: the macro tree transducer [13] ("mtt"). Mtts can be seen as particularly simple functional programs on trees restricted to primitive recursion via (input) tree pattern matching. Alternatively, mtts can be seen as context-free tree grammars (introduced in [23] as "context-free dendrogrammars";

see also [15, 11, 12] and [19, Section 15]), the nonterminals of which are controlled by a top-down tree storage (in the spirit of [7]). It is an open problem, if it is decidable for a given mtt whether or not its translation can be realized by a top-down tree transducer (with "origin" semantics, this is decidable [14]). As mentioned above, it is a necessary condition for the mtt to have linear height increase ("LHI"). This raises the question, can we decide for a given mtt, whether or not its translation has LHI? Here we give an affirmative answer to this question. It is also an open problem, if it is decidable for a given mtt whether or not its translation can be realized by an attributed tree transducer [20, 21, 16] ("att"). It is well-known that atts have *linear size-to-height increase* ("LSHI"), see, e.g., [17]. This raises the question, can we decide for a given mtt, whether or not its translation is of LSHI? We give an affirmative answer. Note that it was conjectured in [10] that the methods of that paper could be adapted to give such an affirmative answer.

Let us now discuss our results in more detail. To decide both the LHI and LSHI properties, we introduce a new normal form called "depth proper". An mtt is depth proper if each parameter of every (reachable) state appears at infinitely many different depths (for different input trees). The idea of our construction is to eliminate parameters that only appear at bounded depths; we use regular look-ahead to determine which bounded paths to output at a given moment. Since in this way we may generate output "earlier" than the original transducer, new "helper states" need to be introduced which continue the translation at the correct input nodes. Both issues, the change of look-ahead and the introduction of new states may cause the newly constructed transducer *not* to be depth proper. For this reason our construction has to be iterated.

To understand the idea of the construction, let us examine a small example. We consider input and output trees over a binary symbol f and the nullary symbol a and an mtt with the following rules.

$$\begin{array}{llll} q_0(f(x_1,x_2)) & \to & f(q_1(x_2), q_2(x_2, q_{\mathrm{id}}(x_1))) & q_2(f(x_1,x_2),y_1) \to f(y_1, q_0(x_2)) \\ q_0(a) & \to & a & q_2(a,y_1) \to f(y_1,a) \\ q_1(f(x_1,x_2)) & \to & q_{\mathrm{id}}(x_1) & q_{\mathrm{id}}(f(x_1,x_2)) \to f(q_{\mathrm{id}}(x_1), q_{\mathrm{id}}(x_2)) \\ q_1(a) & \to & a & q_{\mathrm{id}}(a) \to a \end{array}$$

The transducer realizes the following translation:

$$f(t_1, f(t_2, \underbrace{f(t_3, f(t_4, \dots))}_{t})) \Rightarrow f(t_2, f(t_1, f(t_4, f(t_3, \dots))))$$

The following shows in detail how the rules of the mtt are applied to produce as output first the tree t_2 and then the tree t_1:

$$\begin{array}{rl} & q_0(f(t_1, f(t_2,t))) \\ \Rightarrow_{\text{first rule}} & f(q_1(f(t_2,t)), q_2(f(t_2,t), q_{\mathrm{id}}(t_1))) \\ \Rightarrow_{\text{second rule}} & f(q_{\mathrm{id}}(t_2), q_2(f(t_2,t), q_{\mathrm{id}}(t_1))) \\ \Rightarrow^*_{\text{last two rules}} & f(t_2, q_2(f(t_2,t), q_{\mathrm{id}}(t_1))) \\ \Rightarrow_{\text{third rule}} & f(t_2, f(q_{\mathrm{id}}(t_1), q_0(t))) \\ \Rightarrow^*_{\text{last two rules}} & f(t_2, f(t_1, q_0(t))) \end{array}$$

An mtt uses two different types of variables. The first argument of each state in the left-hand side of every rule of the mtt is always of type input tree and performs pattern matching on the current node of the input tree. For this pattern matching, *input variables* of the form x_1 and x_2 are used to denote the first and second subtree of the current input node, respectively. The (possible) next arguments of a state in the left-hand side of a rule are the (accumulating) *parameters* $y_1, y_2, \dots$ of that state. In our example, only the state q_2

has exactly one parameter y_1. Parameters are used to built up output trees in a bottom-up fashion. In the example, consider the application of the first rule, i.e., going from line one to line two in the previous display: here the first (and only) parameter y_1 of state q_2 is instantiated by the output tree that is produced by the call $q_{\mathrm{id}}(t_1)$.

Observe that state q_2 is *not* depth proper: each tree that it outputs is of the form $f(y_1, t)$ where t does not contain the parameter y_1. The idea of our construction is to replace each occurrence of state q_2 in the right-hand side of any rule by this tree "fragment", where at the position of t there will be the new "helper state" $[q_2, 2]$. The path "2" indicates that this state should produce the tree at the second child position of the output tree produced by q_2. We obtain the following (the rules of q_0, q_1 and input a are as before):

$$\begin{array}{lcl}
q_0(f(x_1,x_2)) & \to & f(q_1(x_2), f(q_{\mathrm{id}}(x_1), [q_2,2](x_2))) \\
q_1(f(x_1,x_2)) & \to & q_{\mathrm{id}}(x_1) \\
[q_2,2](f(x_1,x_2)) & \to & q_0(x_2) \\
[q_2,2](a) & \to & a \\
q_{\mathrm{id}}(f(x_1,x_2)) & \to & f(q_{\mathrm{id}}(x_1), q_{\mathrm{id}}(x_2)) \\
q_{\mathrm{id}}(a) & \to & a
\end{array}$$

It should be clear that the new transducer is equivalent to the original one. Moreover, the new transducer uses no parameters whatsoever, therefore it is depth proper. Given a depth proper mtt, we can decide the LSHI property as follows. We consider input trees which contain exactly one special marked input leaf (it will be marked by a state p of the look-ahead automaton, to act as a place-holder for any input tree for which the look-ahead automaton arrives in state p). For such input trees, the mtt produces output trees which only contain nested state calls to the special input leaf. The original transducer has LSHI if and only if the range of this transducer is finite (which is known to be decidable [5]). In a similar way we can decide LHI: here we consider input trees with multiple marked input leaves. To show that if such ranges are not finite, then the translation does not have LSHI (or LHI), is done via pumping arguments (which use depth properness); these pumping arguments are technically rather involved, but are (somewhat) similar to the ones used in [10] to show that it is decidable whether or not an mtt has *linear size increase* (LSI).

If we restrict the translations of mtts to LSI, then we obtain exactly the MSO definable tree translations [10]. Note that this class of translation has recently been characterized by new models of tree transducers, the *streaming tree transducer* [2] and even more recently the *register tree transducer* [3]. The LSI property is decidable for mtts (it can even be decided for compositions of mtts, and if so, then the translation is effectively MSO definable [8]). To decide LSI, the given mtt is first transformed into "proper" normal form. Properness guarantees that (1) each state (except possibly the initial state) produces infinitely many output trees and that (2) each parameter of a state is instantiated with infinitely many distinct argument trees. Note that input properness is a generalization of the proper form of [1]. Once in proper normal form, it suffices to check if the transducer is "finite copying". This means that (a) each node of each input tree is processed only a bounded number of times and that (b) each parameter of every state is copied only a bounded number of times.

2 Preliminaries

The set $\{0, 1, \dots\}$ of natural numbers is denoted by $\mathbb{N}$. For $k \in \mathbb{N}$ we denote by $[k]$ the set $\{1, \dots, k\}$; thus $[0] = \emptyset$. A ranked alphabet (set) consists of an alphabet (set) Σ together with a mapping $\mathrm{rank}_\Sigma : \Sigma \to \mathbb{N}$ that assigns to each symbol $\sigma \in \Sigma$ a natural number called its "rank". We write $\sigma^{(k)} \in \Sigma$ to denote that $\sigma \in \Sigma$ and $\mathrm{rank}_\Sigma(\sigma) = k$. By $\Sigma^{(k)}$ we denote the symbols of Σ that have rank k.

The set T_Σ of (finite, ranked, ordered) trees over Σ is the smallest set of strings S such that if $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $s_1, \ldots, s_k \in S$, then also $\sigma(s_1, \ldots, s_k) \in S$. We write σ instead of $\sigma()$. For a tree $s = \sigma(s_1, \ldots, s_k)$ with $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $s_1, \ldots, s_k \in T_\Sigma$, we define the set $V(s) \subseteq \mathbb{N}^*$ of nodes of s as $\{\varepsilon\} \cup \{iu \mid i \in [k], u \in V(s_i)\}$; thus, nodes are strings over positive integers, where ε denotes the root node of s, and for a node u, ui denotes the i-th child of u. For $u \in V(s)$ we denote by $s[u]$ the label of u in s and by s/u the subtree rooted at u. Formally, let $s = \sigma(s_1, \ldots, s_k)$ and define $s[\epsilon] = \sigma$, $s[iu] = s_i[u]$, $s/\varepsilon = s$, and $s/iu = s_i/u$ for $\sigma \in \Sigma^{(k)}$, $k \geq 0$, $s_1, \ldots, s_k \in T_\Sigma$, $i \geq 1$ and $u \in V(s_i)$ such that $iu \in V(s)$.

We fix two special sets of symbols: the set $X = \{x_1, x_2, \ldots\}$ of variables and the set $Y = \{y_1, y_2, \ldots\}$ of parameters. For $k \geq 1$ let $X_k = \{x_1, \ldots, x_k\}$ and $Y_k = \{y_1, \ldots, y_k\}$. Let A be a set that is disjoint from Σ. Then the set $T_\Sigma(A)$ of trees over Σ indexed by A is defined as $T_{\Sigma'}$ where $\Sigma' = \Sigma \cup A$ and $\mathrm{rank}_{\Sigma'}(a) = 0$ for $a \in A$ and $\mathrm{rank}_{\Sigma'}(\sigma) = \mathrm{rank}_\Sigma$ for $\sigma \in \Sigma$.

For a ranked alphabet Σ and a set A the ranked set $\langle \Sigma, A \rangle$ consists of all symbols $\langle \sigma, a \rangle$ with $\sigma \in \Sigma$ and $a \in A$; the rank of $\langle \sigma, a \rangle$ is defined as $\mathrm{rank}_\Sigma(\sigma)$.

2.1 Tree Substitution

Let Σ be a ranked alphabet and let $s, t \in T_\Sigma$. For $u \in V(s)$ we define the tree $s[u \leftarrow t]$ that is obtained from s by replacing the subtree rooted at node u by the tree t. Let $\sigma_1, \ldots, \sigma_n \in \Sigma^{(0)}$, $n \geq 1$ be pairwise distinct symbols and let $t_1, \ldots, t_n \in T_\Sigma$. Then $t[\sigma_i \leftarrow t_i \mid i \in [n]]$ is the tree obtained from t by replacing each occurrence of σ_i by the tree t_i. We have defined trees as particular strings, and this is just ordinary string substitution (because we only replace symbols of rank zero). We refer to this as "first-order tree substitution".

In "second-order tree substitution" it is possible to replace internal nodes u (of a tree s) by new trees. These new trees use parameters to indicate where the "dangling" subtrees s/ui of the node u are to be placed. Let $\sigma_1 \in \Sigma^{(k_1)}, \ldots, \sigma_n \in \Sigma^{(k_n)}$ be pairwise distinct symbols with $n \geq 1$, $k_1, \ldots, k_n \in \mathbb{N}$, and $t_i \in T_\Sigma(Y_{k_i})$ for $i \in [n]$. Let $s \in T_\Sigma$. Then $s[\![\sigma_i \leftarrow t_i \mid i \in [n]]\!]$ denotes the tree that is inductively defined as (abbreviating $[\![\sigma_i \leftarrow t_i \mid i \in [n]]\!]$ by $[\![\ldots]\!]$) follows: for $s = \sigma(s_1, \ldots, s_k)$, if $\sigma \notin \{\sigma_1, \ldots, \sigma_n\}$ then $s[\![\ldots]\!] = \sigma(s_1[\![\ldots]\!], \ldots, s_k[\![\ldots]\!])$ and if $\sigma = \sigma_j$ for some $j \in [n]$ then $s[\![\ldots]\!] = t_j[y_i \leftarrow s_i[\![\ldots]\!] \mid i \in [k]]$.

2.2 Macro Tree Transducers

A (deterministic bottom-up) *tree automaton* A is given by a tuple (P, Σ, h) where P is a finite set of states, Σ is a ranked alphabet, and h is a collection of mappings $h_\sigma : P^k \to P$ with $\sigma \in \Sigma^{(k)}$ and $k \geq 0$. The extension of h to a mapping $\hat{h} : T_\Sigma \to P$ is defined recursively as $\hat{h}(\sigma(s_1, \ldots, s_k)) = h_\sigma(\hat{h}(s_1), \ldots, \hat{h}(s_k))$ for every $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $s_1, \ldots, s_k$. For every $p \in P$ we define the subset L_p of trees in T_Σ as $\{s \in T_\Sigma \mid \hat{h}(s) = p\}$. We assume that $L_p \neq \emptyset$ for every $p \in P$.

A (total deterministic) *macro tree transducer with (regular) look-ahead* ("mttr") M is given by a tuple $(Q, P, \Sigma, \Delta, q_0, R, h)$, where

- Q is a ranked alphabet of *states*,
- Σ and Δ are ranked alphabet of *input* and *output symbols*,
- (P, Σ, h) is a tree automaton (called the *look-ahead automaton* of M),
- $q_0 \in Q^{(0)}$ is the *initial state*, and
- R is the *set of rules*, where for each $q \in Q^{(m)}$, $m \geq 0$, $\sigma \in \Sigma^{(k)}, k \geq 0$, and $p_1, \ldots, p_k \in P$ there is exactly one rule of the form

  $$\langle q, \sigma(x_1 : p_1, \ldots, x_k : p_k)\rangle(y_1, \ldots, y_m) \to t$$

  with $t \in T_{\Delta \cup \langle Q, X_k\rangle}(Y_m)$.
  The right-hand side t of such a rule is denoted by $\mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)$

We use a notation that is slightly different from the one used in the Introduction: instead of, e.g., $q_2(x_2, q_{\mathrm{id}}(x_1))$ we write $\langle q_2, x_2\rangle(\langle q_{\mathrm{id}}, x_1\rangle)$. Thus, we use angular brackets $\langle\ldots\rangle$ to indicate a state call on an input subtree, and use round brackets (after the angular brackets), to indicate the parameter arguments of the particular state call.

The semantics of an mttr M (as above) is defined as follows. We define the derivation relation $\Rightarrow_M$ as follows. For two trees $\xi_1, \xi_2 \in T_{\Delta\cup\langle Q,T_\Sigma\rangle}(Y)$, $\xi_1 \Rightarrow_M \xi_2$ if there exists a node u in ξ_1 with $\xi_1/u = \langle q, s\rangle(t_1, \ldots, t_m)$, $q \in Q^{(m)}$, $m \geq 0$, $s = \sigma(s_1, \ldots, s_k)$, $\sigma \in \Sigma^{(k)}$, $k \geq 0$, $s_1, \ldots, s_k \in T_\Sigma$, $t_1, \ldots, t_m \in T_{\Delta\cup\langle Q,T_\Sigma\rangle}(Y)$, and $\xi_2 = \xi_1[u \leftarrow \xi]$ where ξ equals

$$\zeta[\![\langle q', x_i\rangle \leftarrow \langle q', s_i\rangle \mid q' \in Q, i \in [k]]\!][y_j \leftarrow t_j \mid j \in [m]]$$

and $\zeta = \mathrm{rhs}_M(q, \sigma, \langle \hat{h}(s_1), \ldots, \hat{h}(s_k)\rangle)$. Since M is total deterministic (i.e., for every state q, input symbol $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and look-ahead states $p_1, \ldots, p_k$, M contains exactly one corresponding rule) there is for every ξ_1 a unique tree $\xi' \in T_\Delta(Y)$ such that $\xi_1 \Rightarrow_M^* \xi'$. For every $q \in Q^{(m)}$, $m \geq 0$ and $s \in T_\Sigma$ we define the *q-translation of s*, denoted by $M_q(s)$, as the unique tree t in $T_\Delta(Y_m)$ such that $\langle q, s\rangle(y_1, \ldots, y_m) \Rightarrow_M^* t$. We denote the *translation realized by M* also by M, i.e., $M = M_{q_0}$ and for every $s \in T_\Sigma$, $M(s) = M_{q_0}(s)$ is the unique tree $t \in T_\Delta$ such that $\langle q_0, s\rangle \Rightarrow_M^* t$.

Let M be an mttr as before. We define *the extension $\widehat{M}$ of M* which can also process look-ahead states at leaves of input trees. Let $\widehat{M} = (Q, P, \hat{\Sigma}, \hat{\Delta}, q_0, R \cup \hat{R}, h \cup h')$ where $\hat{\Sigma} = \Sigma \cup \{p^{(0)} \mid p \in P\}$ and $\hat{\Delta} = \Delta \cup \{\langle q, p\rangle^{(m)} \mid q \in Q^{(m)}, p \in P, m \geq 0\}$. For every $q \in Q^{(m)}$, $m \geq 0$, and $p \in P$ we let $h(p) = p$ and we let the rule $\langle q, p\rangle(y_1, \ldots, y_m) \to \langle q, p\rangle(y_1, \ldots, y_m)$ be in $\hat{R}$; note that the $\langle q, p\rangle$ on the right-hand side of this rule is an output symbol. For the original transducer M we say that the pair (q, p) is *reachable (in M)* if there is an input tree $s \in T_{\hat{\Sigma}}$ such that $\langle q, p\rangle$ occurs in $\widehat{M}(s)$. Clearly it is decidable for a given pair (q, p), whether or not it is reachable; this is because (1) inverse translations of mttrs effectively preserve regularity [13, 22], (2) the set of all trees in $T_{\hat{\Delta}}$ that contain at least one occurrence of $\langle q, p\rangle$ is (effectively) regular, and (3) emptiness of regular tree languages is decidable [4].

We say that M is *nondeleting*, if for every state $q \in Q^{(m)}$, $\sigma \in \Sigma^{(k)}, k \geq 0$, $p_1, \ldots, p_k \in P$, and $j \in [m]$, there is at least one occurrence of y_j in $\mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)$. The next proposition is proved in [9, Lemma 6.7] (for mttrs that do not copy parameters, but the proof works analogously for arbitrary mttrs).

▶ **Proposition 1.** *For every mttr, an equivalent nondeleting mttr M can be constructed. For every state q of a nondeleting mttr M of rank m and for every $j \in [m]$: $M_q(s)$ contains at least one occurrence of y_j, for every $s \in T_\Sigma$.*

It is well known that the finiteness of ranges of compositions of mttrs is decidable [5]. A (partial nondeterministic) top-down tree transducer with look-ahead ("topr" for short) is an mttr as before, where $Q = Q^{(0)}$ and R may contain none or several rules for each given q and σ.

▶ **Proposition 2.** *([5, Theorem 4.5]) For a given composition of mttrs and (partial nondeterministic) toprs it is decidable whether or not the range of the composition is finite. In the case of finiteness, the range can be constructed.*

3 Depth Proper Normal Form

The depth proper normal form requires that each parameter of each state q occurs at unbounded depth in the output trees of that state (for each given look-ahead state p such that (q, p) is reachable). Formally, let q be a state of rank $m \geq 1$, $j \in [m]$, and $p \in P$. If

(q,p) is reachable, then for every natural number n there must exist an input tree $s_n \in L_p$ such that y_j occurs at depth $> n$ in the tree $M_q(s_n)$. Conversely, we say that parameter y_j is *depth-bounded* for q and p if there exists an n for which no such input tree $s_n \in L_p$ exists; more generally, we say that $Z \subseteq Y_m$ is *depth-bounded* for q and p, if each $y \in Z$ is depth-bounded for q and p.

If Z is depth-bounded for q and p, then there are only finitely many output paths in the trees in $M_q(L_p)$ under which the parameters from Z occur. The *Z-skeleton* of an arbitrary tree t is obtained from t by replacing each top-most node u such that t/u does not contain any occurrence of a parameter from Z by some symbol. Clearly, Z is depth-bounded for q and p if and only if the Z-skeleta of all trees in $M_q(L_p)$ form a finite set.

Let Δ be an arbitrary ranked alphabet, $m \geq 1$, $t \in T_\Delta(Y_m)$, and $Z \subseteq Y_m$. Let us write $\mathsf{ps}(t) \subseteq Y_m$ for the set of parameters occurring in t. Let us now be more specific as to which symbols replace the top-most nodes u of t such that $\mathsf{ps}(t/u) \cap Z = \emptyset$. Since in our construction later we will want to obtain a transducer that is nondeleting, it will be helpful to know which parameters appear in a given deleted tree. Therefore we replace such nodes u by the set $\mathsf{ps}(t/u)$. We denote by $\lfloor t \rfloor_Z$ the Z-skeleton of t and define it inductively as follows (where $\delta \in \Delta$):

$$\lfloor t \rfloor_Z = \begin{cases} t & \text{if } t \in Z \\ \delta(\lfloor t_1 \rfloor_Z, \ldots, \lfloor t_n \rfloor_Z) & \text{if } \mathsf{ps}(t) \cap Z \neq \emptyset \text{ and } t = \delta(t_1, \ldots, t_n) \\ \mathsf{ps}(t) & \text{if } \mathsf{ps}(t) \cap Z = \emptyset. \end{cases}$$

The definition of $\lfloor t \rfloor_Z$ is extended to sets L of trees as $\lfloor L \rfloor_Z = \{\lfloor t \rfloor_Z \mid t \in L\}$. We call *$Y$-nodes* the nodes u in $V(\lfloor t \rfloor_Z)$ such that $\lfloor t \rfloor_Z / u = Z' \subseteq Y_m$. We denote by $\mathcal{U}(\lfloor t \rfloor_Z)$ the set of Y-nodes on $\lfloor t \rfloor_Z$. The notion of parameters in a tree naturally extends to Z-skeleta with, for a Y-node labeled Z': $\mathsf{ps}(Z') = Z'$. The proof of the next lemma is straightforward by induction on t (see full version of this paper: Lemma 3 in [18]).

▶ **Lemma 3.** *Let Δ be a ranked alphabet, $m \geq 1$, $Z \subseteq Y_m$, and $t \in T_\Delta(Y_m)$. (1) $t = \lfloor t \rfloor_Z [u \leftarrow t/u \mid u \in \mathcal{U}(\lfloor t \rfloor_Z)]$. (2) $\mathsf{ps}(\lfloor t \rfloor_Z) = \mathsf{ps}(t)$.*

Finally, we define depth properness for mtts with look-ahead.

▶ **Definition 4.** *The mttr $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ is in* depth proper normal form *(or, synonymously, M is* depth proper*) if for every $q \in Q^{(m)}$, $m \geq 1$, and $p \in P$ it holds that if (q,p) is reachable, then $\lfloor M_q(L_p) \rfloor_{\{y_j\}}$ is infinite for all $j \in [m]$.*

From now on we will want to make use of the following definitions:

$$\begin{aligned} F_p &= \{q \in Q^{(m)} \mid \exists j \in [m], \lfloor M_q(L_p) \rfloor_{\{y_j\}} \text{ is finite}\} \\ Y(q,p) &= \{y_j \mid j \in [\mathrm{rank}_Q(q)] \text{ such that } \lfloor M_q(L_p) \rfloor_{\{y_j\}} \text{ is finite}\} \end{aligned}$$

It should be clear that $\lfloor M_q(L_p) \rfloor_{Y(q,p)}$ is finite for every q and p, as stated in the next lemma (the proof is in the full version: Lemma 5 in [18]).

▶ **Lemma 5.** *Let M be an mtt, q a state of M, and p a look-ahead state of M. Then $\lfloor M_q(L_p) \rfloor_{Y(q,p)}$ is finite.*

3.1 Construction of the Normal Form and Examples

Let M be an mttr as before. We assume that M is nondeleting (which is justified by Proposition 1). The idea of the construction is as follows. First, we determine all reachable pairs (q,p) such that $Y(q,p) \neq \emptyset$. Let (q,p) be such a pair and let $Z = Y(q,p)$. An occurrence of $\langle q, x_i\rangle$ in a right-hand side $\mathrm{rhs}_M(q', \sigma, \langle p_1, \ldots, p_k\rangle)$ such that $p_i = p$ is called a (q,p)-*call*. Our aim is to replace each (q,p)-call by an appropriate tree from $\lfloor M_q(L_p)\rfloor_Z$. Just which tree is the appropriate one will be determined by regular look-ahead. Moreover, such trees should be modified not to contain leaf nodes labeled by subsets of Y: such nodes will be replaced by calls of new "helper states".

▶ **Definition 6.** *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be a nondeleting mttr that is not depth proper. We construct the new mttr $\pi(M) = (Q \cup H, P', \Sigma, \Delta, q_0, R', h')$. For every $q \in Q^{(m)}$ and $p \in P$ such that $Y(q,p) \neq \emptyset$, H contains the following set of helper states:*

$$\{[q,p,t,u]^{(|U|)} \mid t \in \lfloor M_q(L_p)\rfloor_{Y(q,p)}, u \in V(t), t/u = U \subseteq Y_m\}$$

and P' contains (p,φ) for any function φ that assigns to each $q \in F_p$ a tree in $\lfloor M_q(L_p)\rfloor_{Y(q,p)}$. Observe that H and P' are well defined, because $\lfloor M_q(L_p)\rfloor_{Y(q,p)}$ is finite by Lemma 5. Note that $|U| \leq |Y_m \setminus Y(q,p)|$; since $Y(q,p)$ is non-empty this implies that the rank of each helper state is at most $(r-1)$, where r is the maximal rank of the states in Q.

For every $q \in Q^{(m)}$, $m \geq 0$, $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $(p_1,\varphi_1), \ldots, (p_k,\varphi_k) \in P'$ we let the rule

$$\langle q, \sigma(x_1 : (p_1,\varphi_1), \ldots, x_k : (p_i,\varphi_i)\rangle(y_1, \ldots, y_m) \to rhs_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)[\![.]\!]$$

be in R', where the second-order tree substitution $[\![.]\!]$ is defined as follows.

$$[\![.]\!] = [\![\langle q', x_i\rangle \leftarrow \varphi_i(q')\big[u \leftarrow [q', p_i, \varphi_i(q'), u](y_{j_1}, \ldots, y_{j_n}) \mid \varphi_i(q')/u = \{y_{j_1}, \ldots, y_{j_n}\}, j_1 < \cdots < j_n\big] \mid q' \in F_{p_i}, i \in [k]]\!].$$

We define $h'_\sigma((p_1,\varphi_1), \ldots, (p_k,\varphi_k)) = (p,\varphi)$ where $p = h_\sigma(p_1, \ldots, p_k)$ and, using the special second-order substitution $[\![\ldots]\!]^{\$}$ from Definition 8, for every $q \in F_p$,

$$\varphi(q) = \Big\lfloor rhs_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)[\![\langle q', x_i\rangle \leftarrow \varphi_i(q') \mid q' \in F_{p_i}, i \in [k]]\!]^{\$}\Big\rfloor_{Y(q,p)}.$$

The special second-order substitution $[\![\ldots]\!]^{\$}$ is the same as the normal one except that the special first-order substitution is applied for each involved first-order substitution. The special first-order substitution is the same as the normal one except that it gives special treatment to Y-nodes which are replaced by Y-nodes containing all parameters occurring in trees to be substituted for the parameters in the original Y-nodes.

Note that, when $L_p \neq \emptyset$ for all $p \in P$, then $L_{(p,\phi)} \neq \emptyset$ for all $(p,\phi) \in P'$. For every helper state $[q,p,t,u] \in H^{(n)}$, $n \geq 0$, $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $(p_1,\varphi_1), \ldots, (p_k,\varphi_k) \in P'$ such that $h_\sigma(p_1, \ldots, p_k) = p$ we let the rule

$$\langle [q,p,t,u](\sigma(x_1 : (p_1,\varphi_1), \ldots, x_k : (p_k,\varphi_k))\rangle(y_1, \ldots, y_n) \to \xi/u[y_{j_\nu} \leftarrow y_\nu \mid \nu \in [n]]$$

be in R' where $t/u = \{y_{j_1}, \ldots, y_{j_n}\}$, $j_1 < \cdots < j_n$, $\xi = rhs_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)[\![.]\!]$, and $[\![.]\!]$ is the substitution from above.

We now show how the depth proper normal form is achieved using an example. An additional example (which makes more interesting use of helper states) can be found in the full version of this paper: at page 21 in [18]. Let $M = (Q, \{p\}, \Sigma, \Delta, q_0, R, h_0)$ with $Q = \{q_0^{(0)}, q_1^{(1)}, q_2^{(2)}\}$, $\Sigma = \{a^{(1)}, b^{(1)}, e^{(0)}\}$, and $\Delta = \{f^{(2)}, g^{(1)}, e^{(0)}\}$ be an mttr where $(\Sigma, \{p\}, h_0)$ with $L_p = T_\Sigma$ and R consists of these rules:

$$\begin{aligned}
\langle q_1, a(x)\rangle(y_1) &\to \langle q_2, x\rangle(y_1, \langle q_1, x\rangle(y_1)) & \langle q_2, a(x)\rangle(y_1, y_2) &\to f(y_1, \langle q_1, x\rangle(g(y_2)))\\
\langle q_1, b(x)\rangle(y_1) &\to y_1 & \langle q_2, b(x)\rangle(y_1, y_2) &\to f(y_2, y_1)\\
\langle q_1, e\rangle(y_1) &\to g(y_1) & \langle q_2, e\rangle(y_1, y_2) &\to f(y_2, y_1)
\end{aligned}$$

We suppose that the q_0-rules are defined so that all states are reachable. Now we have $F_p = \{q_2\}$, $Y(q_2, p) = \{y_1\}$, and $\lfloor M_{q_2}(L_p)\rfloor_{\{y_1\}} = \{t_1, t_2\}$ with $t_1 = f(y_1, \{y_2\})$ and $t_2 = f(\{y_2\}, y_1)$. As before, we can rewrite q_2-calls with the skeleta, but since there are two possibilities t_1 and t_2, we need to separate the rules according to the input using the look-ahead. In general, F_p contains several states each of which may have multiple skeleta, so each look-ahead contains a finite map from F_p to skeleta. Let $\varphi_1 = \{q_2 \mapsto t_1\}$ and $\varphi_2 = \{q_2 \mapsto t_2\}$ such that $L_{p,\varphi_1} = \{a(s) \mid s \in \mathcal{T}_\Sigma\}$ and $L_{p,\varphi_2} = \{b(s) \mid s \in \mathcal{T}_\Sigma\} \cup \{e\}$. The (q_1, a)-rule containing a q_2-call is separated as

$$\begin{aligned}
\langle q_1, a(x : (p, \varphi_1))\rangle(y_1) &\to f(y_1, \langle [q_2, p, t_1, 2], x\rangle(\langle q_1, x\rangle(y_1)))\\
\langle q_1, a(x : (p, \varphi_2))\rangle(y_1) &\to f(\langle [q_2, p, t_2, 1], x\rangle(\langle q_1, x\rangle(y_1)), y_1)
\end{aligned}$$

where $[q_2, p, t_1, 2]$ and $[q_2, p, t_2, 1]$ are helper states. Each helper state has rank 1 because the corresponding node in the skeleton is a Y-node of length 1. The arguments of the call are inherited from the arguments of the original q_2-call that occur in the sequence. For example, $\langle [q_2, p, t_1, 2], x\rangle$ is called with $\langle q_1, x\rangle(y_1)$ since t_1 has a Y-node $\{y_2\}$ and the original q_2-call has $\langle q_1, x\rangle(y_1)$ as the second argument. The rules of these helper states are constructed from the original q_2-rule with substitution (which causes nothing since no states in F_p are called) and extracting a subtree at the Y-node, that is,

$$\begin{aligned}
\langle [q_2, p, t_1, 2], a(x : (p, \varphi))\rangle(y_1) &\to \langle q_1, x\rangle(g(y_1))\\
\langle [q_2, p, t_2, 1], b(x : (p, \varphi))\rangle(y_1) &\to y_1\\
\langle [q_2, p, t_2, 1], e\rangle(y_1) &\to y_1
\end{aligned}$$

where $\varphi \in \{\varphi_1, \varphi_2\}$ and we had to rename the parameter y_2 into y_1 (because the helper states only refer to y_2). Note that rules for $([q_2, p, t_1, 2], b)$, $([q_2, p, t_1, 2], e)$ and $([q_2, p, t_2, 1], a)$ do not have to be considered. These rules are not referred because the states are never called with the input symbols due to their look-ahead. For example, the $[q_2, p, t_1, 2]$-call occurs only in the (q_1, a)-rule with $x \in L_{p,\varphi_1}$ in which the root symbol cannot be b.

Thereby we have been able to remove every call of states in F_p. However, new improper states may be generated by the separation of rules because of the look-ahead introduction. In fact, we have $F_{p,\varphi_2} = \{q_1, q_2, [q_2, p, t_2, 1]\}$ in the example above. Since every q_2-call has already been removed in the previous step, we have to apply the same technique again for the calls of q_1 and $[q_2, p, t_2, 1]$. We have $Y(q_1, (p, \varphi_2)) = \{y_1\}$ and $Y([q_2, p, t_2, 1], (p, \varphi_2)) = \{y_1\}$. Moreover $\left\lfloor M'_{q_1}(L_{p,\varphi_2})\right\rfloor_{\{y_1\}} = \{y_1, g(y_1)\}$ and $\left\lfloor M'_{[q_2,p,t_2,1]}(L_{p,\varphi_2})\right\rfloor_{\{y_1\}} = \{y_1\}$.

Look-ahead has to be introduced to determine which skeleton to output. Two maps over F_{p,φ_2}, except for q_2 whose call has already been removed, are defined: $\varphi_3 = \{q_1 \mapsto y_1, [q_2, p, t_2, 1] \mapsto y_1\}$ and $\varphi_4 = \{q_1 \mapsto g(y_1), [q_2, p, t_2, 1] \mapsto y_1\}$ such that $L_{p,\varphi_2,\varphi_3} = \{b(s) \in L_{p,\varphi_2} \mid s \in \mathcal{T}_\Sigma\}$ and $L_{p,\varphi_2,\varphi_3} = \{e\}$. The (q_1, a)- and $([q_2, p, t_1, 2], a)$-rules with look-ahead φ_2 which contains a q_2-call are separated as follows:

$$\begin{array}{lcl}
\langle q_1, a(x:(p,\varphi_2,\varphi_3))\rangle(y_1) & \to & f(y_1,y_1)\\
\langle [q_2,p,t_1,2], a(x:(p,\varphi_2,\varphi_3))\rangle(y_1) & \to & g(y_1)\\
\langle q_1, a(x:(p,\varphi_2,\varphi_4))\rangle(y_1) & \to & f(g(y_1),y_1)\\
\langle [q_2,p,t_1,2], a(x:(p,\varphi_2,\varphi_4))\rangle(y_1) & \to & g(g(y_1))
\end{array}$$

The resulting mttr is depth proper.

3.2 Correctness Proof and Termination of Iteration

Here we prove the correctness of transducer $\pi(M)$ that was defined in Definition 4. Lemma 7 establishes the correctness of the look-ahead, relates the states of $\pi(M)$ to those of M, and shows that the transducer $\pi(M)$ is nondeleting. The latter is needed, so that the construction of π can be carried out iteratively (recall from Definition 4 that M is required to be nondeleting in order to construct $\pi(M)$). To prove Point (2) we use a "special" kind of second-order tree substitution which replaces Y-nodes by new Y-nodes consisting of parameters in the output trees that would have been substituted for the parameters in the original Y-node (Definition 8).

▶ **Lemma 7.** *Let M be a nondeleting mttr and $N = \pi(M)$ be the mttr of Definition 6, both with the tuples as in that definition. Let $s \in T_\Sigma$ with $\hat{h}'(s) = (p,\varphi)$.*
(1) *$p = \hat{h}(s)$,*
(2) *$\forall q \in F_p$: $\varphi(q) = \lfloor M_q(s) \rfloor_{Y(q,p)}$,*
(3) *$\forall q \in Q$: $N_q(s) = M_q(s)$,*
(4) *$\forall q \in F_p$ and $u \in V(t)$ with $t = \varphi(q)$ and $t/u = \{y_{j_1}, \dots, y_{j_n}\}$ with $j_1 < \cdots < j_n$: $N_{[q,p,t,u]}(s) = M_q(s)/u[y_{j_\nu} \leftarrow y_\nu \mid \nu \in [n]]$, and*
(5) *the mttr N is nondeleting.*

We first need a small lemma showing that the skeleton of the output of an mttr M can be directly computed from given a input tree by modifying the rules of M. For this lemma we first need to define how to compute second-order substitutions of skeleta, which will be used for the modification of the right-hand sides of rules. We do so on a *nondeleting* mttr M, i.e. such that states always use all their parameters.

▶ **Definition 8.** *Let Γ be a ranked alphabet and let $t_1, \dots, t_n \in T_\Gamma(Y)$. Let $s \in T_\Gamma(Y_n \cup \mathcal{P}(Y_n))$. The* special first-order substitution $[y_i \leftarrow t_i \mid i \in [n]]^{\$}$ *(for short $[.]^{\$}$) applied to s is inductively defined as:*

$$s[.]^{\$} = \begin{cases} t_i & \text{if } s = y_i \text{ for } i \in [n] \\ \gamma(s_1[.]^{\$}, \dots, s_k[.]^{\$}) & \text{if } s = \gamma(s_1, \dots, s_k) \\ \bigcup_{i \in U} \mathsf{ps}(t_i) & \text{if } s = \{y_i \mid i \in U\} \subseteq Y_n \text{ for some } U \subseteq [n]. \end{cases}$$

Let $\gamma_1^{(k_1)}, \dots, \gamma_n^{(k_n)} \in \Gamma$, $n \geq 1$ be pairwise different symbols and assume now that $t_i \in T_\Gamma(Y_{k_i} \cup \mathcal{P}(Y_{k_i}))$ for $i \in [n]$ and that $s \in T_\Gamma(Y_n)$. The special second-order substitution $[\![\gamma_i \leftarrow t_i \mid i \in [n]]\!]$ *(for short $[\![.]\!]^{\$}$) applied to s is inductively defined as:*

$$s[\![.]\!]^{\$} = \begin{cases} t_i[y_j \leftarrow s_j[\![.]\!]^{\$} \mid j \in [k_i]]^{\$} & \text{if } s = \gamma_i(s_1, \dots, s_{k_i}) \text{ for } i \in [n] \\ \gamma(s_1[\![.]\!]^{\$}, \dots, s_k[\![.]\!]^{\$}) & \text{if } s = \gamma(s_1, \dots, s_k) \text{ with } \gamma \notin \{\gamma_1, \dots, \gamma_n\} \\ s & \text{if } s = y_j \text{ for } j \in [n]. \end{cases}$$

For all sets $Z \subseteq Y_m$ such that no Y-node in $t[\![.]\!]^{\$}$ intersects Z, we define the Z-skeleton $\lfloor t[\![.]\!]^{\$} \rfloor_Z$ of $t[\![.]\!]^{\$}$ inductively as before, with a special case for Y-nodes: for all Y-nodes S we have $\lfloor S \rfloor_Z = S \subseteq Y_m \setminus Z$.

▶ **Lemma 9.** *Let M be a nondeleting mttr as before. Let $q \in Q$, $\sigma \in \Sigma^{(k)}$, and $p_1, \ldots, p_k \in P$. Let $p = h(\sigma(p_1, \ldots, p_k))$ and $t = rhs_M(q, \sigma, \langle p_1, \ldots, p_k \rangle)$. Let $s_1 \in L_{p_1}, \ldots, s_k \in L_{p_k}$. By $[\![.]\!]^{\$}$ we denote the substitution $[\![\langle q', x_i \rangle \leftarrow \lfloor M_{q'}(s_i) \rfloor_{Y(q',p_i)} \mid q' \in Q, i \in [k]]\!]^{\$}$ and by $[\![M]\!]$ we denote $[\![\langle q', x_i \rangle \leftarrow M_{q'}(s_i) \mid q' \in Q, i \in [k]]\!]$.*

(1) If $y \in Y(q,p)$ and y occurs in t in the j-th argument of a node $\langle q', x_i \rangle$ for $q' \in Q$ and $i \in [k]$, then $y_j \in Y(q', p_i)$.

(2) No Y-node in $t[\![.]\!]^{\$}$ intersects $Y(q,p)$.

(3) $\lfloor t[\![.]\!]^{\$} \rfloor_{Y(q,p)} = \lfloor t[\![M]\!] \rfloor_{Y(q,p)}$

Proof. If some $y_j \notin Y(q', p_i)$ then $\lfloor M_{q'}(L_{p_i}) \rfloor_{y_j}$ is infinite and, if y occurs in t_j ($j \in [m]$), then $\lfloor M_q(L_p) \rfloor_y$ is also infinite and $y \notin Y(q,p)$. So (1) holds.

If $y \in Y(q,p)$ occurs in a Y-node of $t[\![.]\!]^{\$}$, then it occurs in t in the j-th argument of a node $\langle q', x_i \rangle$ with $y_j \notin Y(q', p_i)$, which contradicts (1). So (2) holds.

The statement (3) is proved by induction on t. The cases of $t = y_j$ and $t = \gamma(t_1, \ldots, t_n)$ are easy. In the case of $t = \langle q', x_i \rangle(t_1, \ldots, t_m)$, we have

$$\begin{aligned}
\left\lfloor t[\![.]\!]^{\$} \right\rfloor_{Y(q,p)} &= \left\lfloor \lfloor M_{q'}(s_i) \rfloor_{Y(q',p_i)} [y_j \leftarrow t_j[\![.]\!]^{\$} \mid j \in [m]]^{\$} \right\rfloor_{Y(q,p)} \\
&= \left\lfloor \lfloor M_{q'}(s_i) \rfloor_{Y(q',p_i)} [y_j \leftarrow t_j[\![M]\!] \mid j \in [m]]^{\$} \right\rfloor_{Y(q,p)} \\
&= \lfloor M_{q'}(s_i)[y_j \leftarrow t_j[\![M]\!] \mid j \in [m]] \rfloor_{Y(q,p)} \\
&= \lfloor t[\![M]\!] \rfloor_{Y(q,p)} .
\end{aligned}$$

◀

We can now prove Lemma 7:

Proof. All the statements are proven by induction on the structure of s. Let $s = \sigma(s_1, \ldots, s_k)$ with $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $s_1, \ldots, s_k \in T_\Sigma$. For $i \in [k]$ let $\hat{h}'(s_i) = (p_i, \varphi_i)$. By the definition of h', $p = h_\sigma(p_1, \ldots, p_k)$, which is equal to $\hat{h}(s)$. Thus, Statement (1) holds. For Statement (2) let $q \in F_p$: Then $\varphi(q)$ is defined as $\lfloor \zeta[\![\varphi_i]\!]^{\$} \rfloor_{Y(q,p)}$ where $\zeta = \mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k \rangle)$ and $[\![\varphi_i]\!]^{\$}$ denotes the special substitution $[\![\langle q', x_i \rangle \leftarrow \varphi_i(q') \mid q' \in F_{p_i}, i \in [k]]\!]^{\$}$. By induction, $\lfloor \zeta[\![\varphi_i]\!]^{\$} \rfloor_{Y(q,p)}$ equals $\left\lfloor \zeta[\![\langle q', x_i \rangle \leftarrow \lfloor M_{q'}(s_i) \rfloor_{Y(q',p_i)} \mid q' \in F_{p_i}, i \in [k]]\!]^{\$} \right\rfloor_{Y(q,p)}$. By Lemma 9(3) the latter equals $\lfloor \zeta[\![\langle q', x_i \rangle \leftarrow M_{q'}(s_i) \mid q' \in F_{p_i}, i \in [k]]\!] \rfloor_{Y(q,p)} = \lfloor M(s) \rfloor_{Y(q,p)}$.

We now prove Statement (3). Let $q \in Q$. Then $N_q(s) = \zeta[\![.]\!][\![N]\!]$, where $\zeta = \mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k \rangle)$, $[\![.]\!]$ is the substitution as in the construction, and $[\![N]\!] = [\![\langle r, x_i \rangle \leftarrow N_r(s_i) \mid r \in Q', i \in [k]]\!]$. By the induction hypothesis of Statement (2), we can replace $\varphi_i(q')$ by $\lfloor M_{q'}(s_i) \rfloor_{Y(q',p_i)}$ in the substitution $[\![.]\!]$. This gives

$$\begin{aligned}
\zeta[\![\langle q', x_i \rangle \leftarrow \lfloor M_{q'}(s_i) \rfloor_{Y(q',p_i)} [u' \leftarrow [q', p_i, \varphi_i(q'), u'](y_{j_1}, \ldots, y_{j_n}) \mid \\
\varphi_i(q')/u' = \{y_{j_1}, \ldots, y_{j_n}\}, j_1 < \cdots < j_n] \mid q' \in F_{p_i}, i \in [k]]\!][\![N]\!].
\end{aligned}$$

This can be written as $\zeta[\![.]\!][\![H]\!][\![Q]\!]$, where $[\![H]\!] = [\![\langle q', x_i \rangle \leftarrow N_{q'}(s_i) \mid q' \in H, i \in [k]]\!]$ and $[\![Q]\!] = [\![\langle q', x_i \rangle \leftarrow N_{q'}(s_i) \mid q' \in (Q \setminus F_{p_i}), i \in [k]]\!]$. By induction of Statement (4) the substitution $[\![H]\!]$ replaces the subtree $[q', p_i, \varphi_i(q'), u'](y_{j_1}, \ldots, y_{j_n})$ by the tree $M_{q'}(s_i)/u[y_{j_\nu} \leftarrow y_\nu \mid \nu \in [n]][y_\nu \leftarrow y_{j_\nu} \mid \nu \in [n]] = M_{q'}(s_i)/u$. Thus we obtain:

$$\begin{aligned}
\zeta[\![\langle q', x_i \rangle \leftarrow \lfloor M_{q'}(s_i) \rfloor_{Y(q',p_i)} [u' \leftarrow M_{q'}(s_i)/u' \mid u' \in \mathcal{U}(\lfloor M_{q'}(s_i) \rfloor_{Y(q',p_i)})] \\
\mid q' \in F_{p_i}, i \in [k]]\!][\![Q]\!]
\end{aligned}$$

By Lemma 3 (for $Z = Y(q', p_i)$ and $t = M_{q'}(s_i)$) the tree on the right of the arrow in the leftmost second-order substitution equals $M_{q'}(s_i)$. We have:

$$\zeta[\![\langle q', x_i\rangle \leftarrow M_{q'}(s_i) \mid q' \in F_{p_i}, i \in [k]]\!][\![\langle q', x_i\rangle \leftarrow N_{q'}(s_i) \mid q' \in Q \setminus F_{p_i}, i \in [k]]\!].$$

By induction of Statement (3), $N_{q'}(s_i) = M_{q'}(s_i)$ for $q \in Q \setminus F_{p_i}$. This gives us exactly $M_q(s)$, by the definition of the semantics of mttrs. Thus,

$$N_q(s) = \zeta[\![.]\!][\![N]\!] = M_q(s). \tag{1}$$

This concludes the proof of Statement (3).

We now prove Statement (4). Let $q \in F_p$ and $u \in V(t)$ with $t = \varphi(q)$ and $t/u \subseteq Y$. By the definition of the rules for the helper states, $N_{[q,p,t,u]}(s) = (\zeta[\![.]\!])/u[\![N]\!][y]$ where $t/u = \{y_{j_1}, \ldots, y_{j_n}\}$, $j_1 < \cdots < j_n$, and $[y] = [y_{j_\nu} \leftarrow y_\nu \mid \nu \in [n]]$. It follows from Lemma 9(1) that if $\langle q', x_i\rangle$ occurs in $\zeta = \mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)$ and $q \in F_p$, then $q' \notin Q \setminus F_{p_i}$. Hence, every proper ancestor v of u is labeled by a symbol in Δ, i.e., $(\zeta[\![.]\!][y])[v] \in \Delta$. This implies that we can move the "$/u$" operation of taking the subtree at node u to the right (after the application of the substitution $[\![N]\!]$) in the above displayed formula. We obtain $\zeta[\![.]\!][\![N]\!]/u[y]$. By the right equation in Formula 1, this equals $M_q(s)/u[y]$.

To prove Statement (5), let $q \in Q^{(m)}$, $m \geq 0$. Then

$$\zeta' = \mathrm{rhs}_N(q, \sigma, \langle (p_1, \varphi_1), \ldots, (p_k, \varphi_k)\rangle) = \zeta[\![.]\!],$$

where $\zeta = \mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)$ and $[\![.]\!]$ is as before. By Statement (2), $[\![.]\!]$ substitutes occurrences of $\langle q', x_i\rangle$ with $i \in [k]$ and $q' \in F_{p_i}$ by the tree $\lfloor M_{q'}(s_i)\rfloor_{Y(q',p_i)}$ in which leaves labeled by $Z \subseteq Y_m$ are replaced by $\langle q_H, x_i\rangle(y_{j_1}, \ldots, y_{j_n})$ with $Z = \{y_{j_1}, \ldots, y_{j_n}\}$. By Lemma 3(2) this implies that y_j occurs in ζ' for each $j \in [m]$. ◀

We show that the iteration of the construction $\pi(M)$ will terminate with a transducer that is depth proper. First, let us discuss what property a single iteration of π ensures. Let $p \in P$. Note that the set F_p is defined independently of reachability, i.e., F_p may contain states q such that (q, p) is *not* reachable. Let φ such that $(p, \varphi) \in P'$. Then $F_p \subseteq F_{(p,\varphi)}$. This inclusion follows from Lemma 7 as follows: let $s \in L_{(p,\varphi)}$ and let $q \in F_p$ be of rank m. The latter means that there exists a $j \in [m]$ and a number n such that every occurrence of y_j in $M_q(s')$ is at depth $\leq n$ for every $s' \in L_p$. By Lemma 7 (1), $s \in L_p$ and by Lemma 7 (3), $N_q(s) = M_q(s)$. Thus, every occurrence of y_j in $N_q(s)$ also occurs at depth $\leq n$. So $q \in F_{(p,\varphi)}$.

We now consider reachability. We say that a state q is *depth proper*, if for all $p \in P$ such that (q, p) is reachable, $q \notin F_p$. If $q \in F_p$, then for all φ such that $(p, \varphi) \in P'$ it holds that $(q, (p, \varphi))$ is not reachable. This property follows immediately from the definition of look-ahead and the rules of $\pi(M)$: the substitution $[\![.]\!]$ replaces each state call $\langle q', x_i\rangle$ with $q \in F_{p_i}$ by a tree that does not contain states of Q. So, if $(q, (p, \varphi))$ is reachable, then $q \notin F_p$; however, it may be that $q \in F_{(p,\varphi)}$, which means that q is not depth proper. It means that if $F_{(p,\varphi)} = F_p$ for all $(p, \varphi) \in P'$, then all states $q \in Q$ are depth proper. Let $Q_0 = Q$ and consider now the iterated application of π. Clearly, after some iterations of π, it will hold that $F_{(p,\varphi)} = F_p$ for all $(p, \varphi) \in P'$. To see this, consider the chain of inclusions

$$F_p \cap Q_0 \subseteq F_{(p,\varphi_1)} \cap Q_0 \subseteq \cdots \subseteq F_{(p,\varphi_1,\ldots,\varphi_k)} \cap Q_0 \subseteq \ldots$$

for any maps φ_i introduced in the look-ahead of $\pi^i(M)$. Since Q_0 is finite, the chain contains only finitely many strict inclusions. Hence there is a minimal n such that $F_{(p,\varphi_1,\ldots,\varphi_n)} \cap Q_0 = F_{(p,\varphi_1,\ldots,\varphi_{n'})} \cap Q_0$ for all $n' > n$.

Consider a tree with an artificial root node which contains all such chains, i.e., for each $p \in P$ there is exactly one child of the root node labeled F_p, and a node labeled F_p has children labeled $F_{(p,\varphi)}$ for each $(p,\varphi) \in P'$, etc. Moreover, a node labeled $F_{(p,\varphi_1,\ldots,\varphi_n)}$ as in the chain above is a leaf of this tree. Since each node of this tree is finitely branching (because P' is finite) and each path has finite length, we know by König's lemma that the tree is finite. Thus, if d is the depth of this tree, then for the mttr $M' = \pi^d(M)$, all states in Q_0 are depth proper.

Let m be the maximal rank of the states in Q_0. Since all helper states are of rank $< m$, we know that M' contains no improper states of rank $\geq m$. We now proceed in the same fashion and construct a transducer $M'' = \pi^{n'}(M')$ which contains no improper states of rank $\geq (m-1)$. In a similar way we eventually obtain an mttr for which *all states* are depth proper (and which is equivalent to M). Thus, even though we do not constructively derive a precise bound, we know that after *some* number of applications of π we are sure to obtain a depth proper mttr.

Before we state the main theorem of this section, we need the following lemma (the proof is a straightforward reduction to Proposition 2 and can be found in the full version of this paper: Lemma 8 in [18]).

▶ **Lemma 10.** *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be an mttr and let $q \in Q^{(m)}$, $m \in \mathbb{N}$, $j \in [m]$, and $p \in P$. It is decidable whether or not $\lfloor M_q(L_p) \rfloor_{\{y_j\}}$ is finite. In case of finiteness, $\lfloor M_q(L_p) \rfloor_{\{y_j\}}$ can be constructed.*

Since for a pair (q,p) it is decidable whether or not it is reachable (see Section 2.2), Lemma 10 implies that it is decidable whether or not a given mttr is depth proper.

▶ **Theorem 11.** *For every mttr M, we can construct an equivalent mttr M' such that M' is depth proper.*

Proof. There is a nondeleting mttr M_0 equivalent to M ([9] or Proposition 1). We repeatedly construct equivalent transducers $\pi(M)$, $\pi(\pi(M))$, etc. until a proper mttr is obtained (which is decidable by Lemma 10). The repetition terminates (first eliminating all reachable calls of improper states of the highest rank m, then those or rank $m-1$, etc.) as explained above. ◀

4 Linear Height and Linear Size-to-Height Increase

In this section we define the Linear Height and Linear Size-to-Height Increase properties. We then characterize and give decision algorithms for those properties by using the depth proper form.

Let Γ be a ranked alphabet and t be a tree over Γ. We define the size $|t|$ of a tree t as its number of nodes $|V(t)|$. The height $\mathsf{ht}(t)$ of t is defined as $\mathsf{ht}(t) = 0$ if $t \in \Gamma^{(0)}$ and $\mathsf{ht}(t) = 1 + \max\{\mathsf{ht}(t_i) \mid i \in [k]\}$ if $t = \gamma(t_1, \ldots, t_k)$ for $\gamma \in \Gamma^{(k)}$, $k \geq 1$, and $t_1, \ldots, t_k \in T_\Gamma$.

Let M be an mttr (with input ranked alphabet Σ). Then M has *linear size-to-height increase* (for short LSHI) if there exists a number c such that for every input tree $s \in T_\Sigma$: $\mathsf{ht}(M(s)) \leq c \cdot |s|$. The mttr M has *linear height increase* (for short LHI) if there exists a number c such that for every input tree $s \in T_\Sigma$: $\mathsf{ht}(M(s)) \leq c \cdot \mathsf{ht}(s)$.

We now introduce two additional properties for mttrs which will allow us to decide whether a given mttr has LSHI or LHI. Recall that $\widehat{M}$ denotes the extension of M: $\widehat{M}$ can translate input trees which may contain leaves that are labeled by elements from P (the set of look-ahead states of M). Whenever the state q of M, of rank m, encounters an input node u labeled by an element p of P, the transducer $\widehat{M}$ outputs $\langle q,p \rangle(y_1, \ldots, y_m)$. We call a tree in $s \in T_\Sigma(P)$ a Σ-*context* if it contains exactly one occurrence u of an element of P.

We say that the mttr M is *finite nesting* (for short fnest), if there exists a number c such that for every Σ-context s there are at most c-many occurrences of symbols $\langle q, p\rangle$ with $q \in Q$ on any path of the tree $\widehat{M}(s)$; in this case, we say that c is a *nesting bound* of M. We say that M is *finite yield nesting* (for short fynest), if there exists a number c such that for every input tree $s \in T_\Sigma(P)$ there are at most c-many occurrences of symbols from $\langle q, p\rangle$ with $q \in Q$ on any path of the tree $\widehat{M}(s)$; in this case, we say that c is a *yield nesting bound* of M. The proof of the next lemma is straightforward (by reduction to Proposition 2).

▶ **Lemma 12.** *Let M be an mttr. Then (1) it is decidable whether or not M is finite nesting and (2) it is decidable whether or not M is finite yield nesting.*

Proof. Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$. We use the extension $\widehat{M} = (\hat{Q}, P, \hat{\Sigma}, \hat{\Delta}, q_0, \hat{R}, h)$ of M with input trees in $s \in T_\Sigma(P)$ which contain (1) exactly one or (2) arbitrarily many occurrences of elements of P. We then use a nondeterministic top-down tree transducer N which chooses any path in the tree $\widehat{M}(s)$ and outputs only the elements from $\langle Q, P\rangle$ on that path, now seen as unary symbols. The resulting output language $N(\widehat{M}(T_\Sigma))$ is finite if and only if M is (1) fnest or (2) fynest.

Formally, $N = (\{q_1^{(0)}\}, \hat{\Delta}, \Gamma, q_1, R')$ where $\Gamma = \langle Q, P\rangle \cup \{e^{(0)}\}$. For every $\delta \in \Delta^{(k)}$, $k \geq 1$, and $i \in k$ we let the rule $\langle q_1, \delta(x_1, \ldots, x_k)\rangle \to \langle q_1, x_i\rangle$ be in R'. For every $\delta \in \Delta^{(0)}$ we let the rule $\langle q_1, \delta\rangle \to e$ be in R'. For every $\langle q, p\rangle \in \langle Q, P\rangle^{(m)}$, $m \geq 1$, and $i \in [m]$ we let the rule $\langle q_1, \langle q, p\rangle(x_1, \ldots, x_m)\rangle \to \langle q, p\rangle(\langle q_1, x_i\rangle)$ be in R'. For every $\langle q, p\rangle \in \langle Q, P\rangle^{(0)}$ we let the rule $\langle q_1, \langle q, p\rangle\rangle \to \langle q, p\rangle$ be in R'. It is straightforward to show (by induction on the structure of s), that $N(\widehat{M}(T_\Sigma(P)))$ is finite if and only if M is fynest. Let L be the set of trees in $T_\Sigma(P)$ which contain exactly one occurrence of an element of P. It is straightforward to show (by induction on the structure of s), that $N(\widehat{M}(L))$ is finite if and only if M is fnest. ◀

Informally the next lemma is easy to understand, e.g., for Statement (1), if M is finite nesting with bound c, then a single node of an input tree can only "contribute" at most $c \cdot \text{mhr}$ to the height of the output tree, where mhr denotes the maximum height of the right-hand side of any rule of the mttr.

▶ **Lemma 13.** *Let M be an mttr. (1) If M is finite nesting, then it is of linear size-to-height increase. (2) If M is finite yield nesting, then it is of linear height increase.*

Proof. Informally, we can understand this lemma by looking at a given path O in an output tree and, using origin semantics, at how many nodes along this path have their origin in different parts of the input tree.

For (1), the finite nesting property gives a bound c on the number of state calls to a single input node, nested along path O. Intuitively, noting mhr the maximum height of the right-hand side of a rule, $c \cdot \text{mhr}$ is a bound on the number of output nodes along path O with their origin in a single input node. This bound clearly implies that the height of the output (maximum number of nodes on a path) is linearly bounded by the size of the input.

For (2), instead of looking at a single input node, we look at all the input nodes at a given depth d in the input. The finite yield nesting property implies a bound c on the nesting (along a path O) of state calls to input nodes of depth d. Each such call may produce at most mhr nodes along path O with their origin in a node of depth d. So c.mhr is a bound for the number of nodes along path O with their origin in a node of depth d.

Formally, we apply $\widehat{M}$ to a tree $t \in T_\Sigma(P)$. We modify t by substituting nodes in P, and we bound the growth of the height of $\widehat{M}(t)$ for each substitution. We will conclude by stating that any input tree $s \in T_\Sigma$ can be built by successive substitutions, and so the height of the output is linearly bounded by the number of substitutions (which will be the size of s for (1), and the height of s for (2)). Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ and let mhr be the maximum height of the right-hand side of any rule in R. Let s be a fixed tree in T_Σ.

To prove (1), consider an arbitrary set U of pairwise independent (i.e. not being descendants of each other) nodes of a fixed input tree $s \in T_\Sigma$. Let $s' = s[u \leftarrow h(s/u) \mid u \in U]$, let $u \in U$, and $\sigma = s[u] \in \Sigma^{(k)}$ with $k \geq 0$. Let c be a nesting bound for M, then, along any output path in $\widehat{M}(s')$, there are at most c state calls $\langle q, s'/u\rangle$ with origin u in s'. Then $\widehat{M}(s'[u \leftarrow \sigma(h(s/u1), \ldots, h(s/uk))])$ is obtained by replacing such state calls with the corresponding right-hand side of rules, which implies that: $\mathsf{ht}(\widehat{M}(s'[u \leftarrow \sigma(h(s/u1), \ldots, h(s/uk))])) \leq c \cdot \mathrm{mhr} + \mathsf{ht}(\widehat{M}(s'))$. The tree $s \in T_\Sigma$ can be obtained from the tree $h(s) \in T_\Sigma(P)$ by $|s|$ such substitutions. The height of $\widehat{M}(h(s)) = \langle q_0, h(s)\rangle$ is 0. So $\mathsf{ht}(M(s)) \leq c \cdot \mathrm{mhr} \cdot |s|$, so M of linear size-to-height increase.

To prove (2), for $i \in [\mathsf{ht}(s)]$, let U_i be the set of nodes at depth i in s, and let s_i be the tree obtained from s by replacing all nodes $u \in U_i$ by $h(s/u)$. Let c be a yield nesting bound for M, then, along any output path O in $\widehat{M}(s_i)$, there are at most c state calls $\langle q, s_i/u\rangle$ with $u \in U_i$. Then $\widehat{M}(s_{i+1}) = \widehat{M}(s_i[u \leftarrow \sigma_u(h(s/u1), \ldots, h(s/uk)) \mid u \in U_i])$ is obtained by replacing these state calls in $\widehat{M}(s_i)$ with the corresponding right-hand side of rules, so $\mathsf{ht}(\widehat{M}(s_{i+1})) \leq c \cdot \mathrm{mhr} + \mathsf{ht}(\widehat{M}(s_i))$. By applying this $\mathsf{ht}(s) + 1$ times, starting from s_0, we obtain that the height of $M(s)$ is $\leq c \cdot \mathrm{mhr} \cdot (\mathsf{ht}(s) + 1)$. So M is of linear-height-increase. ◀

The next lemma is a central piece of the paper. This is where we use the *depth proper property*.

▶ **Lemma 14.** *Let M be an mttr that is depth proper. (1) If M is not finite nesting, then M does not have linear size-to-height increase. (2) If M is not finite yield nesting, then M does not have linear height increase.*

Proof. Let M be given by a tuple as usual. To prove (1), assume that M is not fnest. We will use this and the *depth proper property* to show that M does not have LSHI. Since M is not fnest (and has only finitely many states) there must be some state $q \in Q^{(m)}$ with $m \geq 1$ that occurs arbitrarily often on paths of output trees of $\widehat{M}$. More precisely, there are infinite sequences of contexts $c_0, c_1, \ldots$ and numbers $n_0 < n_1 < \cdots$ such that q occurs $\geq n_0$ times on a path in $\widehat{M}(c_0)$ and q occurs $\geq n_1$ times on a path in $\widehat{M}(c_0[u_0 \leftarrow c_1])$ where u_0 is the path in c_0 to a node $p \in P$, etc. From this we can deduce (by considering sufficiently many numbers n_i), similarly to the proof of Lemma 6.5 of [10], that M is "(nested) input pumpable", i.e., there exist $q_1, q_2, j, s_0, s_1, u_0, u_1, p$ such that

1. $\langle q_1, p\rangle$ occurs in $\widehat{M}(s_0[u_0 \leftarrow p])$,
2. $\widehat{M}_{q_1}(s_1[u_1 \leftarrow p])$ has either: a subtree $\langle q_1, p\rangle(t_1, \ldots, t_m)$ such that some $t_{j'}$ contains a subtree $\langle q_2, p\rangle(\xi_1, \ldots, \xi_l)$ where ξ_j contains $y_{j'}$ for some $j' \in [m]$, or a subtree $\langle q_2, p\rangle(t_1, \ldots, t_l)$ such that t_j contains a subtree $\langle q_1, p\rangle(\xi_1, \ldots, \xi_m)$,
3. $\widehat{M}_{q_2}(s_1[u_1 \leftarrow p])$ has a subtree $\langle q_2, p\rangle(t_1, \ldots, t_l)$ such that t_j contains y_j, and
4. $p = h(s_1/u_1) = h(s_1[u_1 \leftarrow p])$.

By "pumping", i.e., considering $s_n = s_0[u_0 \leftarrow s_1[u_1 \leftarrow s_1[u_1 \leftarrow \ldots]]]$ with n replacements of the node u_1, we obtain that $\widehat{M}_{q_1}(s_n)$ contains a path with at least n nested occurrences of $\langle q_2, p\rangle$. Note that this proof is simpler than that of Lemma 6.5 of [10] because we only look here at the height of outputs instead of the size of outputs. This is simpler because, in a mttr, a state call can copy a parameter containing large outputs of other state calls, causing a size growth of the output that is difficult to track, but these copies cannot be copied vertically on top of each other, so the output height is easier to track.

This is where we use the *depth proper property*: we first assume by contradiction that M has LSHI, i.e., there exists a c such that for every input tree $s \in T_\Sigma$: $\mathsf{ht}(M(s)) \leq c \cdot |s|$. Since M is *depth proper*, we may choose $s \in L_p$ such that $M_{q_2}(s)$ contains an occurrence

of y_j at depth $\geq c \cdot c_1 + 1$, where $c_1 = |s_1[u_1 \leftarrow p]| - 1$. We know that $\widehat{M}_{q_1}(s_n)$ contains at least n nested occurrences of q_2 (where the j-th subtree of q_2 always contains further nested occurrences of q_2). Now let $t_n = s_0[u_0 \leftarrow s_n[u_1^n \leftarrow s]]$ and take $n > c(c_0 + c_2)$, where $c_0 = |s_0[u_0 \leftarrow p]| - 1$ and $c_2 = |s|$. Since $|t_n| = c_0 + nc_1 + c_2$, we obtain that $\mathsf{ht}(M(t_n)) > c \cdot |t_n|$ because $\mathsf{ht}(M(t_n)) \geq n(cc_1 + 1) > ncc_1 + c(c_0 + c_2) = c \cdot |t_n|$ by the choice of n. So *nested input pumpability* implies that M is not of LSHI.

We now prove that if M is not finite nesting, then it must be *nested input pumpable*. In order to do so, we first introduce a few notations and characterize *nested input pumpability* and the *finite nesting* property using these notations.

Let c be a Σ-context and $q \in Q$ be a state of M. To talk about the nesting of states in $\widehat{M}_q(c)$, we first give a notation for paths:

1. For any node u at depth n in $\widehat{M}_q(c)$, we note the path to node u as the sequence of pairs:

   $$(\ell_1, i_1)\,(\ell_2, i_2)\ldots(\ell_n, i_n)\,(\ell_{n+1}, \bot)$$

   where $i_1, \ldots, i_n$ are indexes such that $u = i_1\, i_2 \ldots i_n$ and, for all $j \leq n+1$, ℓ_j is the label of node $i_1 \ldots i_{j-1}$ or, if node $i_1 \ldots i_{j-1}$ is labeled by a state call $\langle q', p\rangle$, then $\ell_j = q'$,
2. Since we are only interested in the nesting of states, we remove from such paths all pairs (ℓ_j, i_j) where $\ell \in \Delta$. We obtain nesting sequences of the form:

   $$(q_1, k_1)\,(q_2, k_2)\ldots(q_n, k_n)\,(\ell_{n+1}, k_{n+1})$$

   where ℓ_{n+1} is either a state in Q or a parameter, $k_{n+1} \in \{\bot\} \cup \mathbb{N}$, and for all $j \leq n$, $k_j \in [m_j]$ where m_j is the arity of state q_j.
3. For each such sequence, if $\ell_{n+1} = q_{n+1} \in Q$ then we write:

   $$(q, \bot) \to_c (q_1, k_1)\,(q_2, k_2)\ldots(q_n, k_n)\,(\ell_{n+1}, k_{n+1})$$

   Otherwise $\ell_{n+1} = y_k$ is a parameter of q, $k_{n+1} = \bot$ and we write:

   $$(q, k) \to_c (q_1, k_1)\,(q_2, k_2)\ldots(q_n, k_n)$$

This defines a relation $\to_c \subseteq \Theta \times \Theta^*$ where $\Theta = \{(q, k) \mid q \in Q^{(m)}, k \in [m] \cup \{\bot\}\}$ and Θ^* denotes the set of (possibly empty) sequences of elements of Θ. Note that if $(q, \bot) \to_c w$, then in the nesting sequence $w \in \Theta^*$ only the last pair may contain $\bot$. A *nesting loop* is given by a Σ-context c with a leaf labeled p such that $h(c) = p$, and two pairs $(q_1, k_1), (q_2, k_2) \in \Theta$ such that:

- $(q_1, k_1) \to_c w_1\,(q_1, k_1)\,w_2\,(q_2, k_2)\,w_3$ or $(q_1, k_1) \to_c w_1\,(q_2, k_2)\,w_2\,(q_1, k_1)\,w_3$,
- $(q_2, k_2) \to_c w_4\,(q_2, k_2)\,w_5$,
- (q_1, p) is reachable, i.e., there exists Σ-context c_0 with a leaf labeled p such that $\langle q_1, p\rangle$ appears in $\widehat{M}(c_0)$,

for some nesting sequences $w_1, w_2, w_3, w_4, w_5 \in \Theta^*$. This allows us to rephrase the *nested input pumpability* property as the existence of a *nesting loop*. We want to prove that if M is not finite nesting then it has a nesting loop.

We extend the relation $\to_c$ to sequences of pairs on the left so that, for pairs (q_1, k_1), $(q_2, k_2) \in \Theta$ and sequences $w_1, w_2 \in \Theta^*$, if $(q_1, k_1) \to_c w_1$ and $(q_2, k_2) \to_c w_2$ then $(q_1, k_1)\,(q_2, k_2) \to_c w_1 w_2$. More generally, for all sequences $w_1, w_1', w_2, w_2' \in \Theta^*$, if $w_1 \to_c w_1'$ and $w_2 \to_c w_2'$ then $w_1\, w_2 \to_c w_1'\, w_2'$. We can now show the following claim:

▷ Claim 15. For all Σ-contexts c and c' with leafs labeled resp. p and p' such that $p = h(c')$, we can define the Σ-context $c \cdot c' = c[p \leftarrow c']$ and, for all sequences $w, w'' \in \Theta^*$, if $w \to_{c\cdot c'} w''$ then there exists a sequence $w' \in \Theta^*$ such that $w \to_c w' \to_{c'} w''$.

Proof. We only need to show this for $w = (q_0, k_0) \in \Theta$ because of the definition of $\to_c$ on sequences of pairs. Because $(q_0, k_0) \to_{c \cdot c'} w''$, there must be a path Π in $\widehat{M_{q_0}}(c \cdot c')$ reducing to w'' (by removing pairs in $\Delta \times \mathbb{N}$ and removing $(y_{k_0}, \bot)$ if $k_0 \neq \bot$). Because $\widehat{M_{q_0}}(c \cdot c') = \widehat{M_{q_0}}(c)[\langle q, p\rangle \leftarrow \widehat{M_q}(c')]$, path Π can be similarly obtained from a path Π' in $\widehat{M_{q_0}}(c)$ by substituting each (q, k) with a path in $\widehat{M_q}(c')$. More specifically, noting $(q_1, k_1), \ldots, (q_n, k_n)$ the pairs in path Π' that are in Θ (in order of apparition in Π'), we substitute in Π':

- each occurrence of a pair $(q_i, k_i) \in \Theta$ by a path Π'_i such that $\Pi'(y_{k_0,\bot})$ is a path in $\widehat{M_{q_i}}(c')$ (for $i \leq n$),
- each occurrence of a pair $(q_n, \bot)$ by a path Π'_n in $\widehat{M_{q_n}}(c')$.

We get: $\Pi = \Pi'[(q_i, k_i) \leftarrow \Pi'_i]$ and, by removing pairs in $(\Delta \times \mathbb{N}) \cup (Y^m \times \{\bot\})$:

$$w'' = w'_1 \, w'_2 \, \ldots \, w'_n$$

where for all $i \leq n$, w'_i is obtained from Π'_i by removing pairs in $(\Delta \times \mathbb{N}) \cup (Y^m \times \{\bot\})$. Then, for all $i \leq n$ and by definition of Π'_i, we have $(q_i, k_i) \to_{c'} w'_i$. So $(q_1, k_1) \ldots (q_n, k_n) \to_{c'} w'_1 \, w'_2 \, \ldots \, w'_n = w''$.

We note w' the sequence obtained from Π' by removing pairs in $(\Delta \times \mathbb{N}) \cup (Y^m \times \{\bot\})$. Then $w' = (q_1, k_1) \ldots (q_n, k_n)$ and so $(q_0, k_0) \to_c w' \to_{c'} w''$. ◁

We could also prove that $\to_{c \cdot c'} = \to_{c'} \circ \to_c$, but it is not necessary for this proof.

To prove that M has a nesting loop (i.e. M is nested input pumpable), we assume that M is not finite nesting. Then, for all $n \in \mathbb{N}$, there exists a Σ-context c_n such that: $(q_0, \bot) \to_{c_n} w$ for some $w \in \Theta^*$ with $|w| \geq n$. We can decompose any such c_n into a concatenation $c_{n,1} \cdot c_{n,2} \cdot \ldots c_{n,r}$ and use the claim to obtain:

$$(q_0, \bot) \to_{c_{n,1}} w_1 \to_{c_{n,2}} w_2 \cdots \to_{c_{n,r}} w_r$$

where $w_1, w_2, \ldots w_r \in \Theta^*$ and $|w_r| = |w| \geq n$. By choosing a large enough n, we will show how to find a *nesting loop*. To do that, we decompose c_n into several contexts and use the claim.

A Σ-context c is *atomic* if its leaf labeled in P is a child node of its root. Let $c = \sigma(t_1, \ldots, t_{i-1}, p_i, t_{i+1}, \ldots, t_k)$ be an atomic Σ-context and $q \in Q$ a state of M. Noting $p_j = h(t_j)$ for $j \neq i$, there is in M a rule $\langle q, \sigma(x_1 : p_1, \ldots, x_k : p_k)\rangle(y_1, \ldots, y_m) \to t$. Then $\widehat{M_q}(c) = t[\langle q', x_j\rangle \leftarrow M_{q'}(c/j) \mid j \neq i]$. Because $M_{q'}(c/j) \in T_\Delta$ for all $q' \in Q$ and $j \neq i$, the nesting of state calls in $\widehat{M_q}(c)$ is the nesting of state calls of the form $\langle q', x_i\rangle$ in t. So, for $(q, k) \in \Theta$, the length of nesting sequences w such that $(q, k) \to_c w$ is bounded by the height of t. There is a finite number of rules for M, so there is a finite number of such t and the length of sequences $w \in \Theta$ such that $(q, k) \to_c w$ has an upper bound B that does not depend on q, k or c. In other words, for all $(q, k) \in \Theta$, $w \in \Theta^*$ and atomic Σ-context c we have:

$$(q, k) \to_c w \quad \Rightarrow \quad |w| \leq B$$

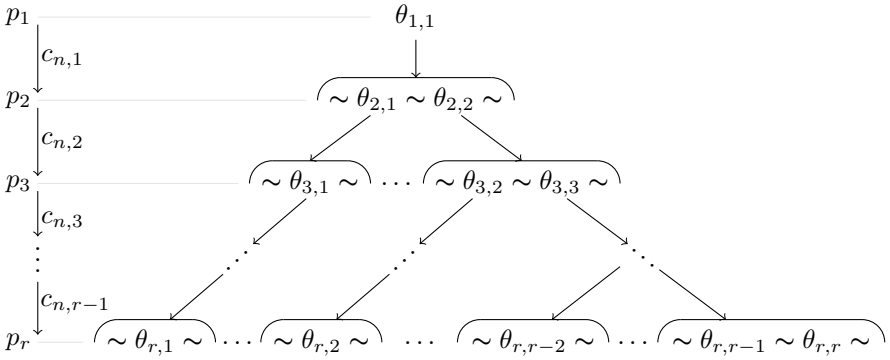
Moreover, for all $w_1, w_2 \in \Theta^*$: $w_1 \to_c w_2 \quad \Rightarrow \quad |w_2| \leq B\,|w_1|$.

We decompose the Σ-context c_n into atomic Σ-contexts $c_{n,1}, c_{n,2}, \ldots, c_{n,r}$. Since $(q_0, \bot) \to_{c_{n,1}} w_1 \to_{c_{n,2}} w_2 \cdots \to_{c_{n,r}} w_r$, we have $|w_r| \leq B^r$ and, since $|w_r| \geq n$: $n \leq B^r$. So, by choosing n large enough, we can also make r as big as we want. In order to find a nesting loop, we require more structure on the nesting sequences $c_{n,1}, \ldots, c_{n,r}$. The precise structure we need is described in the following claim:

▷ Claim 16. For all for all $r \in \mathbb{N}$, if there exists a Σ-context c, a pair $\theta \in \Theta$ and a sequence $w \in \Theta^*$ with $\theta \to_c w$ and $|w| \geq B^r$, then there exists Σ-contexts $c_1, \ldots, c_{r-1}$, look-ahead states $p_1, \ldots, p_r$ and pairs $\theta_{i,j} \in \Theta$ for all $i, j \in [r]$ with $j \leq i$ such that:

- for all $i \in [r-1]$, $h(c_i) = p_i$ and c_i has a leaf labeled p_{i+1},
- $\theta_{1,1} = \theta$,
- for all $i, j \in [r-1]$ with $j < i$, there exists $w_{i,j}, w'_{i,j} \in \Theta^*$ such that: $\theta_{i,j} \to_{c_i} w_{i,j}\, \theta_{i+1,j}\, w'_{i,j}$,
- for all $i \in [r-1]$, there exists $w_i, w'_i, w''_i \in \Theta^*$ such that either $\theta_{i,i} \to_{c_i} w_i\, \theta_{i+1,i}\, w'_i\, \theta_{i+1,i+1}\, w''_i$ or $\theta_{i,i} \to_{c_i} w_i\, \theta_{i+1,i+1}\, w'_i\, \theta_{i+1,i}\, w''_i$.

These conditions can be summed up graphically. To simplify the picture, we replace all sequences $w_{i,j}, w'_{i,j}, w_i, w'_i, w''_i$ for $i, j \in [r]$ with the symbol $\sim$.



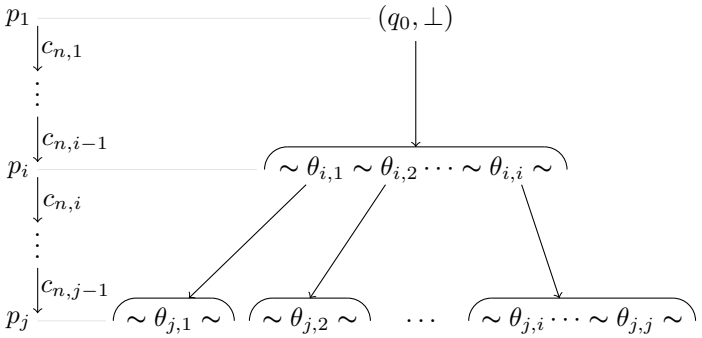
Note that, in this representation, we chose to represent $\theta_{i,i} \to_{c_i} w_i\, \theta_{i+1,i}\, w'_i\, \theta_{i+1,i+1}\, w''_i$ instead of $\theta_{i,i} \to_{c_i} w_i\, \theta_{i+1,i+1}\, w'_i\, \theta_{i+1,i}\, w''_i$ for all $i \in [r-1]$. But this distinction does not matter to the proof of the claim. From now on we use $\sim$ to denote arbitrary sequences in Θ^* which we will not use to find a nesting loop.

Proof. We prove this by induction on r. Let c be a Σ-context, θ a pair in Θ and w a sequence in Θ^* with $\theta \to_c w$ and $|w| \geq B^{r+1}$. We split c into atomic Σ-contexts $c'_1, \ldots, c'_n$, then we have sequences $w_1, \ldots, w_{n-1} \in \Theta^*$ such that $\theta \to_{c'_1} w_1 \cdots \to_{c'_{n-1}} w_{n-1} \to_{c'_n} w$. Let i be the largest i such that there is a pair θ' in the sequence w_i with $\theta' \to_{c'_{i+1} \cdots c'_n} w'$ and $|w'| \geq B^r$. If we had $|w'| \geq B^{r+1}$ then, because c'_{i+1} is atomic, we would have a θ'' in the sequence c'_{i+1} with $\theta'' \to_{c'_{i+2} \cdots c'_n} w''$ and $|w''| \geq B^r$. So $B^r \leq |w'| < B^{r+1} \leq |w|$. Therefore there is another pair $\theta_{2,1}$ in w_i (other than θ') with $\theta_{2,1} \to_{c'_{i+1} \cdots c'_n} w''$ and $|w''| \geq 1$.

Since $\theta' \to_{c'_{i+1} \cdots c'_n} w'$ and $|w'| \geq B^r$, we use the induction hypothesis on θ' and $c'_{i+1} \cdots c'_n$. In order to prove the induction for $r+1$, we rename the Σ-contexts $c_1, \ldots c_{r-1}$, look-ahead states $p_1, \ldots, p_r$ and pairs $\theta_{i,j}$ (for $j \leq i \leq r$) into Σ-contexts $c_2, \ldots c_r$, look-ahead states $p_2, \ldots, p_{r+1}$ and pairs $\theta_{i+1,j+1}$ (for $j \leq i \leq r$). Then $c'_{i+1} \cdots c'_n = c_2 \cdots c_r$.

Since $\theta_{2,1} \to_{c_2 \cdots c_r} w''$ with $|w''| \geq 1$, there are pairs $\theta_{3,1}, \ldots, \theta_{n,1}$ such that $\theta_{n,1}$ appears in the sequence w'' and, for all $i \in [r]$ with $i \geq 2$, $\theta_{i,1} \to_{c_i} w'_i\, \theta_{i+1,1}\, w''_i$ with $w'_i, w''_i \in \Theta^*$. To conclude, we choose $c_1 = c'_1 \ldots c'_i$, $p_1 = h(c_1)$ and $\theta_{1,1} = \theta$. ◁

In order to find a nesting loop, we need two indexes i and j with:



Formally, we require two indexes i, j with $i < j < r$ which share

- the same look-ahead state $h(c_{n,i}) = h(c_{n,j})$,
- the same pair $\theta_{i,i} = \theta_{j,j} \in \Theta$ and
- the same set of pairs $\{\theta_{i,\ell}\}_{0 \leq \ell \leq i} = \{\theta_{j,\ell}\}_{0 \leq \ell \leq j}$.

We ensure the existence of such i, j by choosing $r \geq |P|\,|Q|\,(m+1)\,2^{|Q|(m+1)} + 1$ where m is the maximum arity of states. We now show how to build the nesting loop from indexes i, j. We note $p = h(c_{n,i}) = h(c_{n,j})$, $(q_1, k_1) = \theta_{i,i} = \theta_{j,j}$ and $S = \{\theta_{i,\ell}\}_{0 \leq \ell \leq i} = \{\theta_{j,\ell}\}_{0 \leq \ell \leq j}$. We note $c' = c_{n,i}.c_{n,i+1}.\ldots.c_{n,j-1}$. Note that c' has a leaf labeled p and $h(c') = p$.

We need the sets $\{\theta_{i,\ell}\}_{0 \leq \ell \leq i}$ and $\{\theta_{j,\ell}\}_{0 \leq \ell \leq j}$ to be equal so that the pairs $\theta_{i,k}$ for $k \leq i$ loop on each other through the loop c'. Formally, noting $\theta'_0 = \theta_{j,i}$, for all $\theta'_k \in S$ for $k \in \mathbb{N}$, there exists $\theta'_{k+1} \in S$ such that $\theta'_k \to'_c \sim \theta'_{k+1} \sim$. Since $S \subseteq \Theta$ is finite, there must be $n, m \in \mathbb{N}$ such that $\theta'_n = \theta'_{n+m}$ (with $m \geq 1$), so $\theta'_n \to_{c'^m} \sim \theta'_n \sim$. Also $(q_1, k_1) \to_{c'} \sim \theta'_0 \sim (q_1, k_1) \sim$ and $\theta'_0 \to_{c'^n} x_n$, so $(q_1, k_1) \to_{c'^{n+1}} \sim \theta'_n \sim (q_1, k_1) \sim$ and, for all $m' \in \mathbb{N}$: $(q_1, k_1) \to_{c'^{m'}} \sim (q_1, k_1) \sim \to_{c'^{n+1}} \sim \theta'_n \sim (q_1, k_1) \sim$. Finally, for $c = c'^{m(n+1)}$, we have $(q_1, k_1) \to_c \sim \theta'_n \sim (q_1, k_1) \sim$ and $\theta'_n \to_c \theta'_n$. So we have a *nesting loop*.

In conclusion, if M is not finite nesting, then it is *nested input pumpable*, and so it does not have linear size-to-height increase.

The proof of Statement (2) can be given in a very similar way as for (1), here we only outline the changes to the notations which allow to adapt the proof of (1) to (2). We replace Σ-contexts with elements of the set $T_\Sigma(P)$ containing possibly several leafs labeled in P. The rest of the notational changes are consequences of this change. Given a $s \in T_\Sigma(P)$, we now consider the nesting of state calls called on distinct subtrees of s with potentially distinct look-ahead states. We augment pairs in Θ so as to include the look-ahead, so $\Theta = \{(q, k, p) \mid q \in Q^{(m)}, k \in [m] \cup \{\bot\}, p \in P\}$. The notation $(q, k, p) \to_s (q_1, k_1, p_1) \ldots (q_n, k_n, p_n)$ means that $h(s) = p$ and calls to states $q_1, \ldots, q_n$ on leafs of s labeled $p_1, \ldots, p_n$ resp. are nested on parameters $y_{k_1}, \ldots, y_{k_n}$ along a path in $\widehat{M}_q(s)$. This means that, when concatenating contexts, we write $s(s_1, \ldots, s_m)$ instead of $s \cdot s'$.

For (2), similarly to nesting loops for (1), we define a *yield nesting loop* as given by contexts $s_1, s_2 \in T_\Sigma(P)$, look-ahead states $p_1, p_2 \in P$ and triplets $(q_1, k_1, p_1), (q_2, k_2, p_2) \in \Theta$ such that:

- $h(s_1) = p_1$, $h(s_2) = p_2$, s_1 has two leafs labeled p_1 and p_2, s_2 has one leaf labeled p_2,
- $\langle q_1, p_1 \rangle$ is reachable,
- either $(q_1, k_1, p_1) \to_{s_1} \sim (q_2, k_2, p_2) \sim (q_1, k_1, p_1) \sim$
  or $(q_1, k_1, p_1) \to_{s_1} \sim (q_1, k_1, p_1) \sim (q_2, k_2, p_2) \sim$,
- $(q_2, k_2, p_2) \to_{s_2} \sim (q_2, k_2, p_2) \sim$.

We say that M is *yield nested input pumpable* when it has either a *yield nesting loop* or a *nesting loop*. Note that the existence of either of these loops contradicts the linear height increase property. To prove (2) we prove that infinite yield nesting implies the existence

of either a yield nesting loop or a nesting loop. That proof works similarly to (1): M is not fynest so we can find large enough nesting sequences (but with the new definition of $\rightarrow_s$), find a repeating triplet (q_1, k_1, p_1), pump the loop enough times that a triplet (q_2, k_2, p_2) loops onto itself. Note that if the nested calls to (q_1, k_1, p_1) and (q_2, k_2, p_2) in $(q_1, k_1, p_1) \rightarrow_{s_1} \sim (q_2, k_2, p_2) \sim (q_1, k_1, p_1) \sim$ are on the same leaf in s_1 (with $p_1 = p_2$), then we get a nesting loop (otherwise we get a yield nesting loop). ◀

From Theorem 11 and Lemmas 12, 13, and 14 we obtain our following main theorem.

▶ **Theorem 17.** *Let M be an mttr. Then (1) it is decidable whether or not M has linear size-to-height increase (2) it is decidable whether or not M is linear height-increase.*

5 Conclusions

We have proven that for a given macro tree transducer (with look-ahead) it is decidable whether or not it has linear height increase (LHI) and, whether or not it has linear size-to-height increase (LSHI). Both decision procedures rely on a novel normal form that is called "depth-proper" normal form. Roughly speaking the normal form requires that each parameter of every state of the transducer appears at arbitrary large depths in output trees generated by that state (and for a given look-ahead state).

One major open problem in the field is to prove a Conjecture of Joost Engelfriet (from around the year 2000), that the translation of an mttr can be defined by an attributed tree transducer (atts) if and only if the translation has "linear size to number of distinct output subtrees" increase. Note that deciding such property is out of reach (it is at least as difficult as deciding equivalence of atts). To prove this characterization, different loops need to be considered which produce unbounded numbers of states in (partial) output trees. We believe that the depth normal form will be instrumental in reducing the number of different such loops that must be considered and therefore will be of great help in proving the conjecture.

References

1 Alfred V. Aho and Jeffrey D. Ullman. Translations on a context-free grammar. *Inf. Control.*, 19(5):439–475, 1971. doi:10.1016/S0019-9958(71)90706-6.

2 Rajeev Alur and Loris D'Antoni. Streaming tree transducers. *J. ACM*, 64(5):31:1–31:55, 2017. doi:10.1145/3092842.

3 Mikolaj Bojanczyk and Amina Doumane. First-order tree-to-tree functions. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 252–265. ACM, 2020. doi:10.1145/3373718.3394785.

4 Hubert Comon-Lundh, Max Dauchet, Rémi Gilleron, Cristof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications*. URL: https://jacquema.gitlabpages.inria.fr/files/tata.pdf, November 2007.

5 Frank Drewes and Joost Engelfriet. Decidability of the finiteness of ranges of tree transductions. *Inf. Comput.*, 145(1):1–50, 1998. doi:10.1006/inco.1998.2715.

6 Joost Engelfriet. Bottom-up and top-down tree transformations – A comparison. *Math. Syst. Theory*, 9(3):198–231, 1975. doi:10.1007/BF01704020.

7 Joost Engelfriet. Context-free grammars with storage. *CoRR*, abs/1408.0683, 2014. arXiv:1408.0683.

8 Joost Engelfriet, Kazuhiro Inaba, and Sebastian Maneth. Linear-bounded composition of tree-walking tree transducers: linear size increase and complexity. *Acta Informatica*, 58(1-2):95–152, 2021. doi:10.1007/s00236-019-00360-8.

9 Joost Engelfriet and Sebastian Maneth. Macro tree transducers, attribute grammars, and MSO definable tree translations. *Inf. Comput.*, 154(1):34–91, 1999. doi:10.1006/inco.1999.2807.

10 Joost Engelfriet and Sebastian Maneth. Macro tree translations of linear size increase are MSO definable. *SIAM J. Comput.*, 32(4):950–1006, 2003. doi:10.1137/S0097539701394511.

11 Joost Engelfriet and Erik Meineche Schmidt. IO and OI. I. *J. Comput. Syst. Sci.*, 15(3):328–353, 1977. doi:10.1016/S0022-0000(77)80034-2.

12 Joost Engelfriet and Erik Meineche Schmidt. IO and OI. II. *J. Comput. Syst. Sci.*, 16(1):67–99, 1978. doi:10.1016/0022-0000(78)90051-X.

13 Joost Engelfriet and Heiko Vogler. Macro tree transducers. *J. Comput. Syst. Sci.*, 31(1):71–146, 1985. doi:10.1016/0022-0000(85)90066-2.

14 Emmanuel Filiot, Sebastian Maneth, Pierre-Alain Reynier, and Jean-Marc Talbot. Decision problems of tree transducers with origin. *Inf. Comput.*, 261:311–335, 2018. doi:10.1016/j.ic.2018.02.011.

15 M. J. Fischer. *Grammars with Macro like Productions.* PhD thesis, Harvard University, 1968. See also *Proc. 9th Sympos. on SWAT*, pp. 131-142, 1968.

16 Zoltán Fülöp. On attributed tree transducers. *Acta Cybern.*, 5(3):261–279, 1981. URL: https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/3218.

17 Zoltán Fülöp and Heiko Vogler. *Syntax-Directed Semantics – Formal Models Based on Tree Transducers.* Monographs in Theoretical Computer Science. An EATCS Series. Springer, 1998. doi:10.1007/978-3-642-72248-6.

18 Paul Gallot, Sebastian Maneth, Keisuke Nakano, and Charles Peyrat. Deciding linear height and linear size-to-height increase for macro tree transducers, 2024. arXiv:2307.16500.

19 Ferenc Gécseg and Magnus Steinby. Tree languages. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, pages 1–68. Springer, 1997. doi:10.1007/978-3-642-59126-6_1.

20 Donald E. Knuth. Semantics of context-free languages. *Math. Syst. Theory*, 2(2):127–145, 1968. doi:10.1007/BF01692511.

21 Donald E. Knuth. Correction: Semantics of context-free languages. *Math. Syst. Theory*, 5(1):95–96, 1971. doi:10.1007/BF01702865.

22 Thomas Perst and Helmut Seidl. Macro forest transducers. *Inf. Process. Lett.*, 89(3):141–149, 2004. doi:10.1016/j.ipl.2003.05.001.

23 William C. Rounds. Mappings and grammars on trees. *Math. Syst. Theory*, 4(3):257–287, 1970. doi:10.1007/BF01695769.

24 James W. Thatcher. Generalized sequential machine maps. *J. Comput. Syst. Sci.*, 4(4):339–367, 1970. doi:10.1016/S0022-0000(70)80017-4.

25 James W. Thatcher. There's a lot more to finite automata theory than you would have thought. In *Proc. 4th Ann. Princeton Conf. on Informations Sciences and Systems*, pages 263–276, 1970. Also published in revised form under the title "Tree automata: an informal survey" in Currents in the Theory of Computing (ed. A. V. Aho), Prentice-Hall, 1973, 143–172.

T-Rex: Termination of Recursive Functions Using Lexicographic Linear Combinations

Raphael Douglas Giles
The University of Melbourne, Australia

Vincent Jackson
The University of Melbourne, Australia

Christine Rizkallah
The University of Melbourne, Australia

Abstract

We introduce a powerful termination algorithm for structurally recursive functions that improves on the core ideas behind lexicographic termination algorithms for functional programs. The algorithm generates linear-lexicographic combinations of primitive measure functions measuring the recursive structure of terms. We introduce a measure language that enables the simplification and comparison of measures and we prove meta-theoretic properties of our measure language. Moreover, we demonstrate our algorithm, on an untyped first-order functional language and prove its soundness and that it runs in polynomial time. We also provide a Haskell implementation. As part of this work, we also show how to solve the maximisation of negative vector-components as a linear program.

2012 ACM Subject Classification Theory of computation → Program analysis

Keywords and phrases Termination, Recursive functions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.139

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Supplementary Material

Software (Source Code): `https://github.com/vjackson725/term_check` [9]
archived at `swh:1:dir:c7dd8ada9bdd696f86cd11bc6751817ed37ad120`

1 Introduction

To guarantee the *total correctness* of a program, it is essential to prove that the program terminates [10]. While the halting problem is undecidable for general recursive functions [8, 21], there has been a long line of work on creating termination algorithms that automatically determine whether certain classes of functions terminate [14, 22, 11, 7, 2, 1].

In the context of functional programming, a termination algorithm typically takes a recursive function f as input and attempts to find a function from terms in the language to some well-founded order; this function is called a measure function. When the measure is applied to the arguments of f, it strictly decreases at each recursive call site [22, 11]. Such a measure can be as simple as a single argument to the function that always decreases at each recursive call. Such a simple measure is sufficient for demonstrating the termination of primitive recursive functions [18]. For example, consider the following function defining the binomial coefficient:

$$\begin{aligned} \mathit{choose}\ \langle 0, k\rangle &= 1 \\ \mathit{choose}\ \langle n, 0\rangle &= 1 \\ \mathit{choose}\ \langle \mathbf{S}\, n, \mathbf{S}\, k\rangle &= \mathit{choose}\, \langle n, \mathbf{S}\, k\rangle + \mathit{choose}\, \langle n, k\rangle \end{aligned}$$

To prove that *choose* terminates, we observe that the first projection of the argument, labelled n, strictly decreases at each recursive call site. Since n is a natural number and the natural numbers are well-ordered, we can conclude that *choose* must eventually terminate. Thus

a decreasing measure of *choose* is $\mathbf{size} \circ \pi_1$. The measure of the argument to the *choose* function is the function $\mathbf{size} \circ \pi_1$, where the function π_1 projects the first argument out of the tuple and **size** returns the interpretation of object language number n in the underlying language or logic.

Of course, termination arguments are not always as simple as finding a single argument that decreases at every recursive call. Consider the following example:

$$\begin{array}{lcl} ex_1\,\langle 0, 0\rangle & = & 0 \\ ex_1\,\langle \mathbf{S}\,x, y\rangle & = & ex_1\ \langle x, \mathbf{S}\,y\rangle \\ ex_1\,\langle x, \mathbf{S}\,y\rangle & = & ex_1\ \langle x, y\rangle. \end{array}$$

In this example, no single part of the argument decreases at every recursive call. Hence, we cannot simply find a measure by considering each projection separately. However, this function terminates, and the argument for its termination can be generalised to determine the termination of a larger class of functions.

Note that the first projection of the argument never increases in any recursive call, and in the first recursive call, it strictly decreases. Hence the first recursive call can only be applied a finite number of times, bounded by the size of the first projection. With this fact, we may set the first recursive call aside. Observe that, considered alone, the second recursive call can also only be called a finite number of times, as the second projection of the argument always decreases. Thus, there can only be a finite number of recursive calls to this function overall. This idea is captured by the fact that the lexicographic order of the size of the first followed by that of the second argument decreases at each recursive call.

The Isabelle/HOL theorem prover [17] has a state-of-the-art termination algorithm that uses this idea for automatically detecting termination arguments [7, 2]. This more involved lexicographic order algorithm is quite powerful, covering termination arguments for various functions with non-trivial termination proofs, including the merge function on sorted lists and the Ackermann function [3]. However, it cannot handle the following function:

$$\begin{array}{lcl} ex_2\,\langle 0, 0\rangle & = & 0 \\ ex_2\,\langle \mathbf{S}\,(\mathbf{S}\,x), y\rangle & = & ex_2\ \langle x, \mathbf{S}\,y\rangle \\ ex_2\,\langle x, \mathbf{S}\,(\mathbf{S}\,y)\rangle & = & ex_2\ \langle \mathbf{S}\,x, y\rangle \end{array}$$

Though there is no lexicographic combination of generated measures that decreases at every recursive call, we know that this function must terminate because the sum of the arguments is always decreasing. More precisely, the measure $(\lambda t.\ (\mathbf{size} \circ \pi_1)\ t + (\mathbf{size} \circ \pi_2)\ t)$ decreases at every recursive call.

A termination algorithm that considers sums of the same generated measures as those of Isabelle/HOL would be successful in proving termination for ex_2. If we were to also consider linear combinations with coefficients in $\mathbb{N}$, it would also prove termination for ex_1, since we could take $(\lambda t.\ (2 \cdot ((\mathbf{size} \circ \pi_1)\ t)) + (\mathbf{size} \circ \pi_2)\ t)$.

The Isabelle/HOL algorithm can also not handle the following example, which converts from sparse lists to lists. Sparse lists provide a space-efficient representation of lists that contain many repeated elements. For example the list $[\mathtt{a}, \mathtt{a}, \mathtt{a}, \mathtt{h}, \mathtt{h}]$ would be represented as $[(\mathtt{a}, 3), (\mathtt{h}, 2)]$. We can more formally define the data type $\mathbf{SparseList}\ \alpha$ as inductively generated by elements $\mathbf{SNil}$ and $\mathbf{SCons}\ (x : \alpha)\ (n : \mathbb{N})\ (xs : \mathbf{SparseList}\ \alpha)$. The following function converts sparse lists to regular lists; defined through the usual **Nil** and **Cons** constructors:

$$\begin{array}{lcl} \mathit{toList}\ \mathbf{SNil} & = & \mathbf{Nil} \\ \mathit{toList}\ (\mathbf{SCons}\ x\ 0\ xs) & = & \mathit{toList}\ xs \\ \mathit{toList}\ (\mathbf{SCons}\ x\ (\mathbf{S}\,n)\ xs) & = & \mathbf{Cons}\ x\ (\mathit{toList}\ (\mathbf{SCons}\ x\ n\ xs)). \end{array}$$

$$
\begin{array}{lcl}
ack\,\langle 0, n\rangle & = & \mathbf{S}\,n \\
ack\,\langle \mathbf{S}\,m, 0\rangle & = & ack\,\langle m, \mathbf{S}\,0\rangle \\
ack\,\langle \mathbf{S}\,m, \mathbf{S}\,n\rangle & = & ack\,\langle m, ack\,\langle \mathbf{S}\,m, n\rangle\rangle
\end{array}
\qquad
\begin{array}{l}
\\
\text{call 1} \\
\text{call 2} \\
\text{call 3}
\end{array}
\begin{array}{c}
\mathbf{size} \circ \pi_1 \quad \mathbf{size} \circ \pi_2 \\
\left[\begin{array}{cc}
< & ? \\
< & ? \\
\leq & <
\end{array}\right]
\end{array}
$$

Figure 1 The Ackermann function is presented on the left and its difference matrix on the right.

This function terminates because either the number of **SCons** constructors decreases or it stays the same and the number n decreases. As this function has a lexicographic termination argument, we might expect the Isabelle/HOL termination algorithm to be able to handle it. The reason it cannot is that Isabelle/HOL only generates measures that consider the difference in the number of **SCons** constructors for each recursive function call – missing the fact that the number *inside* the **SCons** decreases.

We provide a novel algorithm, called T-Rex, that proves termination for a large class of functions including all the examples above. It first generates a set of measures based on structural size that is sufficiently detailed to handle examples such as *toList* where the decrease in structural size occurs within another data structure. It then determines whether there exists a lexicographic combination of these measures that decreases at every recursive call, or such an $\mathbb{N}$-linear combination, as well as complex measures combining the two (see ex_4). We demonstrate how to analyse the termination of functional programs by operating directly on and measuring the recursive structure of terms, without relying on the type structure or using higher-order logic and theories of term rewriting. This means that T-Rex can be used to analyse the termination of functional programs without relying on an underlying theorem prover, making it accessible to a wider community. We provide an implementation of our language and our T-Rex algorithm in Haskell.

2 Isabelle/HOL's Termination Algorithm

Isabelle/HOL's lexicographic termination algorithm [7] compares the measures of each argument to the function to that of each recursive call. For example, consider the Ackermann function (Figure 1). The Ackermann function has three recursive calls for which we need to show a decrease between the structural size of the initial and recursive arguments.

The size-change information for these calls is represented in a matrix, where each row corresponds to a recursive call and each column corresponds to a measure of part of the argument. The size-change comparison matrix for *ack* example (Figure 1) consists of the two columns that represent the size-change relation on the first and the second components of the argument tuple, respectively.

The entries record the change between the original and recursive arguments, recorded with the symbols $<$, $\leq$, and ?. The symbol $<$ means that there is a strict decrease in the size of the recursive call argument compared to the initial one, the symbol $\leq$ means that the size decreases or remains the same, and the symbol ? means that the size either increased or the relationship between the sizes is unknown.

From this point forward we omit the labels of the rows and columns of matrices, since these labels clutter the notation and as we wish to orient the reader slowly to thinking about these as matrices in the sense of linear algebra. This will ease introducing the ideas for our extension to the termination algorithm. Isabelle/HOL's lexicographic algorithm works by mimicking the informal argument given in the introduction. It attempts to find

arguments that always either decrease or stay the same size (i.e., that have $<$ and $\leq$ entries) and removes these recursive calls from consideration. Phrased in terms of the matrix, the algorithm repeatedly finds a column of the matrix with only $<$ and $\leq$ entries, and which must contain a $<$ entry, and removes every row of the matrix that has a $<$ entry in that column. If it eventually removes all the rows of the matrix, then the function must terminate.

To find the corresponding measure, it just keeps track of the measure associated with each column. Let m_1 be the measure associated with the first encountered column that only contains $<$ and $\leq$ entries, m_2 the measure associated with the second, and so on, up to the measure associated with the last such column m_n. The *lexicographic* measure, $[m_1, m_2, \ldots, m_n]_{\text{lex}}$, takes the measures m_1 to m_n, and combines their results into a lexicographically ordered tuple. Isabelle/HOL's lexicographic procedure produces a sequence of measures that, when combined lexicographically, decreases at every recursive call.

Returning to the Ackermann example, the sequence of row eliminations computed by the above algorithm is

$$\begin{bmatrix} < & ? \\ < & ? \\ \leq & < \end{bmatrix} \rightsquigarrow \begin{bmatrix} \leq & < \end{bmatrix} \rightsquigarrow \varnothing.$$

where $\varnothing$ stands for the empty matrix. The corresponding measure computed by this procedure, that decreases at every recursive call is $[\mathbf{size} \circ \pi_1, \mathbf{size} \circ \pi_2]_{\text{lex}}$.

Limitations of the Isabelle/HOL algorithm

As noted earlier the Isabelle/HOL algorithm, while very effective for many problems, fails to prove termination for examples such as *ex₂*. We can see this directly by generating the corresponding size-change matrix for *ex₂*:

$$\begin{bmatrix} < & ? \\ ? & < \end{bmatrix},$$

which cannot be reduced using the above procedure, since there is no column with only $<$ and $\leq$ entries. In order to extend this algorithm to prove the termination of such functions, we need more information in these matrices. The information we will use is the numeric increase or decrease of the measured size, when such a value is computable.

In this example, we can compute exactly how much each measure changes at each recursive call and we want to maintain a matrix with entries corresponding to those numeric size-changes, namely:

$$\begin{bmatrix} -2 & 1 \\ 1 & -2 \end{bmatrix}.$$

With this information, we can add the columns of this matrix, to produce a single column corresponding to the sum of these size changes, which in this case is negative in every entry, i.e. decreases at every recursive call, hence, proving termination.

This idea requires we solve three problems: how to generate such matrices, how to find such $\mathbb{N}$-linear combinations, and how to integrate this approach with the lexicographic algorithm. We first develop a language as well as a measure language in order to demonstrate how measure generation works. From there, the remainder of the paper is dedicated to solving these three problems.

Function names $f : N$ Variables $x, y, z : V$

$$\begin{array}{lll} \text{Terms } t \in T & ::= & \mathbf{var}\, x \mid \langle\rangle \mid \langle t, t\rangle \mid \mathbf{inl}\ t \mid \mathbf{inr}\ t \mid \mathbf{roll}\,\{t\} \mid \mathbf{app}\, f\, t \\ \text{Patterns } p \in P & ::= & \mathbf{var}\, x \mid \langle\rangle \mid \langle p, p\rangle \mid \mathbf{inl}\ p \mid \mathbf{inr}\ p \mid \mathbf{roll}\,\{p\} \end{array}$$

Function body $B = [(P \times T)]$ Program $\Gamma : [(N \times B)] ::= \overline{(f, b)}$

$0 = \mathbf{roll}\,\{\mathbf{inl}\,\langle\rangle\}$ $\mathbf{S}\, n = \mathbf{roll}\,\{\mathbf{inr}\ n\}$

Figure 2 Language Syntax.

3 Language

As our termination algorithm targets functional languages, we introduce a core calculus that supports common features of these languages, such as recursion, pattern matching, and standard algebraic data types. Standard ADTs can be encoded in our core language through the constructors that the language provides: recursive structures (roll), products, sums and unit. Our language is untyped, but would permit the addition of the standard type system for these constructs. Our language does not support lambda terms. The concrete syntax of our language is similar to Haskell function definitions. We provide examples in Sections 1, 2.

The syntax of our language is presented in Figure 2. A program is a sequence of function declarations, which are themselves a pair of a function name and function body. Each function body is a sequence of defining equations which are pairs of a pattern p and a term t. We will write the elements of a function body in the form $f\ p = t$, where f is the function name.

Each pattern has a corresponding term that it destructs on. Patterns contain binding variables $\mathbf{var}\, x, \mathbf{var}\, y, \mathbf{var}\, z$, written x, y, z for short, patterns for destructing sums $\mathbf{inl}\ p$ and $\mathbf{inr}\ p$, a pattern for destructing tuples $\langle p, p\rangle$, the pattern for destructing units $\langle\rangle$, and the pattern for destructing rolls $\mathbf{roll}\,\{p\}$. Terms include constructors for each of these as well as function application $\mathbf{app}\, f\, t$, written, $f\ t$ for short. When defining a function, terms describe how to build up the structure of the output using the variables bound by the input pattern.

4 Measure Language

We can prove that a function terminates by finding a measure of the size of the function's argument and showing that the argument's size decreases on every recursive call. This is called a *measure function*, and it is of type $T \to \mathcal{O}$ from the set of terms T to some well-ordered set $\mathcal{O}$. In particular, we use the set of natural numbers ($\mathbb{N}$) for primitive measures and linear combinations of primitive measures, and the set of lists of natural numbers ($[\mathbb{N}]$) under the lexicographic order for our overall lexicographic-linear measures.

There are five basic measures, **unroll**, **uninl**, **uninr**, π_1, and π_2. These deconstruct the basic datatypes, **roll**, **inl** , **inr** , and the left and right subterms of $\langle -, -\rangle$ respectively. The **unroll** destructor adds 1 to the return value, and **uninl** and **uninr** immediately terminate when passed the opposite constructors **inr** and **inl** respectively. In addition, there are two constant measures **case01** and **case10**, that attempts to match the input term as a sum, with **case01** returning 0 for **inl** and 1 for **inr** , and **case10** doing the opposite. There are two measure operators: $\triangleleft$ and **fix**. The measure operator $\triangleleft$ functionally composes two measures, and the **fix** measure operator repeatedly evaluates the given measure.

$$
\begin{array}{lrl}
\text{Measure destructor } m_{\mathrm{d}} & ::= & \mathbf{unroll} \mid \mathbf{uninl} \mid \mathbf{uninr} \mid \pi_1 \mid \pi_2 \\
\text{Recursive measure } m_{\mathrm{r}} & ::= & m_{\mathrm{d}} \mid m_{\mathrm{r}} \triangleleft m_{\mathrm{d}} \\
\text{Primitive measure } m & : & M \\
m & ::= & (\mathbf{fix}\, m_{\mathrm{r}}) \mid \mathbf{case01} \mid \mathbf{case10} \mid m \triangleleft m_{\mathrm{d}} \\
\text{Measure State } M_{st} & = & (M \times T \times \mathbb{N}) \uplus \mathbb{N}
\end{array}
$$

Measure Evaluation Semantics $(-)\Downarrow : (M \times T \times \mathbb{N}) \to M_{st}$

$$
\begin{array}{lcll}
((m \triangleleft \mathbf{unroll}), (\mathbf{roll}\, \{t\}), i)\Downarrow & = & (m, t, 1+i)\Downarrow & \\
((m \triangleleft \mathbf{uninl}), (\mathbf{inl}\ t), i)\Downarrow & = & (m, t, i)\Downarrow & \\
((m \triangleleft \mathbf{uninr}), (\mathbf{inr}\ t), i)\Downarrow & = & (m, t, i)\Downarrow & \\
((m \triangleleft \mathbf{uninr}), (\mathbf{inl}\ t), i)\Downarrow & = & i & \\
((m \triangleleft \mathbf{uninl}), (\mathbf{inr}\ t), i)\Downarrow & = & i & \\
((m \triangleleft \pi_1), \langle s, t\rangle, i)\Downarrow & = & (m, s, i)\Downarrow & \\
((m \triangleleft \pi_2), \langle s, t\rangle, i)\Downarrow & = & (m, t, i)\Downarrow & \\
(\mathbf{case01}, (\mathbf{inl}\ t), i)\Downarrow & = & i & \\
(\mathbf{case01}, (\mathbf{inr}\ t), i)\Downarrow & = & 1+i & \\
(\mathbf{case10}, (\mathbf{inl}\ t), i)\Downarrow & = & 1+i & \\
(\mathbf{case10}, (\mathbf{inr}\ t), i)\Downarrow & = & i & \\
((\mathbf{fix}\, m), t, i)\Downarrow & = & (((\mathbf{fix}\, m) \triangleleft m), t, i)\Downarrow & \\
(m, t, i)\Downarrow & = & (m, t, i) & \mathbf{otherwise}
\end{array}
$$

Measure Functional Semantics $[\![-]\!] : M \to (T \rightharpoonup \mathbb{N})$

$$[\![m]\!]\, t \quad = \quad (m, t, 0)\Downarrow$$

Note that this function is partial as the result of $\Downarrow$ is not always a natural number.

Figure 3 Measure language semantics.

Semantics

The big-step evaluation semantics of the measure language $(-)\Downarrow$, defined in Figure 3, takes as input a triple consisting of a term t in T, a measure m and a natural number. If the measure exhaustively destructures the term through the evaluation, the semantics returns a natural number, and if the evaluation is stuck, the semantics returns a triple of the same type as the input triple.

Note that the evaluation semantics is stuck when it encounters a term that is a variable, a function application, or the term and the measure have mismatched term constructors and measure destructors. The functional semantics abstracts from this case, only being defined for closed terms where the measure and term constructors match.

We will prove that our algorithm correctly demonstrates termination (when it succeeds in finding a measure) by showing that, for every initial argument a_{i} and every argument a_{r} to the recursive calls, $m\ a_{\mathrm{r}} < m\ a_{\mathrm{i}}$ [11]. Note that this method requires that a_{r} and a_{i} are *closed*, that is contain no variables or function calls. We will ensure this in our proofs by substituting variables for closed terms and functions for functions from closed terms to closed terms.

Bounded Difference

In general, we will not be able to determine the exact measure of a term statically, as measure execution can get stuck. Thus, we will need to approximate the difference between the partial evaluations of a measure. This approximation must be an upper bound on the difference between the measures (when applied to any substitution which results in a natural number).

Bounded difference $(\dot{-}) : M_{st} \to M_{st} \to \mathbb{Z} \uplus \{\omega\}$

$$
\begin{array}{rcll}
(m, s, i) & \dot{-} & (m, s, j) & = & i - j \\
i & \dot{-} & (m', t, j) & = & i - j \\
(m, s, i) & \dot{-} & j & = & \omega \\
(m, s, i) & \dot{-} & (m', t, j) & = & \omega \quad \textbf{otherwise}.
\end{array}
$$

Figure 4 Bounded difference.

The bounded difference (Figure 4), written $\dot{-}$, is a conservative estimate of the true difference between the size of a measure applied to two arguments. It has four cases, which depend on whether the measure fully executes and returns an integer, or whether the execution gets stuck, resulting in a measure-term-integer triple. In the first case, the value is exactly what the true subtraction would yield, as the s terms cancel. In the second case, we take the pessimistic assumption that t evaluates to a natural number which maximises the overall size of the difference; in this case, this occurs when t would evaluate to 0, so we may safely ignore the term. The third and fourth cases, we again assume the term s maximises the overall size. However, here s could evaluate to any natural number, and there is no finite bound. Thus in these cases, we return the error value ω, to denote that the subtraction is unboundedly large. We extend the order on integers to $\mathbb{Z} \uplus \{\omega\}$ by asserting that for all $n \in \mathbb{Z}$, $n \leq \omega$.

▶ **Lemma 1** (Difference is Bounded). *For all measures $m : M$, terms $t_1, t_2 : T$, and substitutions θ, where $[\![m]\!]\,\theta(t_1)$ and $[\![m]\!]\,\theta(t_2)$ are defined,*

$$[\![m]\!]\,\theta(t_1) - [\![m]\!]\,\theta(t_2) \;\leq\; (m, t_1, 0)\Downarrow \dot{-}\, (m, t_2, 0)\Downarrow.$$

5 Combining Measures

The termination of a function can be proven by finding a well-founded measure of the size of the arguments, such that the size of the arguments always decreases between the initial and recursive calls of the function.

Our termination algorithm builds measures from three components: *primitive measures*, *linear measures*, and *lexicographic measures*. Primitive measures are functions that transform a term into a natural number, representing a certain measure of the size of that term. We use "primitive measure" to refer both to the syntactic construct (M) and the corresponding mathematical functions ($T \to \mathbb{N}$).

A *linear combination of measures* is a weighted sum of measures. Given a vector of measures $\mathbf{m} : (\alpha \to \mathbb{N})^n$ and a vector of weights $\mathbf{w} : \mathbb{N}^n$ of the same length, we define

$$
\begin{array}{lcl}
(\cdot) & : & \mathbb{N}^n \to (\alpha \to \mathbb{N})^n \to (\alpha \to \mathbb{N}) \\
\mathbf{w} \cdot \mathbf{m} & = & (\lambda x.\ \sum_{i=1}^{m} \mathbf{w}_i \cdot \mathbf{m}_i(x)).
\end{array}
$$

In our termination algorithm, we will only take linear combinations of primitive measures, specialising the above to $\mathbb{N}^n \to (T \to \mathbb{N})^n \to (T \to \mathbb{N})$.

Since linear programming methods need to work in an ordered field, the tools we use will produce linear combinations of measures with positive *rational* coefficients. Finding such a linear combination is equivalent to finding a linear combination with natural coefficients, as the rational weights can be transformed to integers by multiplication of the least common multiple of the divisors of the weights. Note that the Haskell implementation of our algorithm uses fixed-point numbers, rather than exact arithmetic, which can lead to precision loss in this step.

6 Termination Algorithm

We want to show that our functions terminate, by showing that there is a measure of the arguments that decreases between every recursive call. We do this by combining primitive measures into linear combinations, and then making lexicographic combinations of these. Our termination algorithm is composed of three main phases:

1. Generating primitive measures of structural size for the sub-terms of the argument.
2. Generating the *primitive measure difference matrix*, that captures the size change between the initial and recursive arguments, as measured by the primitive measure functions.
3. Using the primitive measure difference matrix to find a lexicographic-linear combination of the primitive measure functions that decreases over every recursive call.

Finding lexicographic-linear combinations can be further broken down into two steps

i. Using the integer-only columns of the primitive measure difference matrix to find a linear combination of the primitive measure functions that decreases over the recursive calls that correspond to these columns.
ii. Using the linear combination procedure iteratively to determine a lexicographic combination of the linear measure functions that decreases over all recursive calls including those that do not have a simple integer difference.

To see how this algorithm works in the case of lexicographic termination, consider ex_3.

$$\begin{array}{lcl} ex_3\ \langle x, y, \mathbf{S}\, z\rangle & = & ex_3\ \langle x, y, z\rangle \\ ex_3\ \langle x, \mathbf{S}\, y, 0\rangle & = & ex_3\ \langle x, y, h\, y\rangle \\ ex_3\ \langle \mathbf{S}\, x, 0, 0\rangle & = & ex_3\ \langle x, h\, x, h\, x\rangle \end{array}$$

Note that, in this example h is some arbitrary function, and that we assume that any function called in a function definition (apart from the function being defined) terminates. That is, we show termination conditional on all called functions also terminating. However, when proving termination, we do not use any knowledge about the behaviour of called functions; so we have no bounds on the value that h will return. The primitive measures for ex_3 and the resulting difference matrix are

$$\begin{array}{l} m_1 = (\mathbf{fix}\,(\mathbf{uninr} \triangleleft \mathbf{unroll})) \triangleleft \pi_1 \\ m_2 = (\mathbf{fix}\,(\mathbf{uninr} \triangleleft \mathbf{unroll})) \triangleleft \pi_2 \\ m_3 = (\mathbf{fix}\,(\mathbf{uninr} \triangleleft \mathbf{unroll})) \triangleleft \pi_3 \end{array} \qquad \begin{bmatrix} 0 & 0 & -1 \\ 0 & -1 & \omega \\ -1 & \omega & \omega \end{bmatrix}.$$

The value ω marks the worst-case value difference. The function h is assumed to terminate but may return a term of arbitrary size; thus the maximum bound of the change in value cannot be any integer. Note that ω prevents any column containing it from being used in a linear programming problem.

The ex_3 function has a lexicographic termination argument, as m_1 decreases or stays the same everywhere, m_2 decreases or stays the same when m_1 stays the same, and m_3 decreases when m_1 and m_2 stay the same. By the lexicographic elimination algorithm,

$$\begin{bmatrix} 0 & 0 & -1 \\ 0 & -1 & \omega \\ -1 & \omega & \omega \end{bmatrix} \rightsquigarrow \begin{bmatrix} 0 & -1 & \omega \\ -1 & \omega & \omega \end{bmatrix} \rightsquigarrow \begin{bmatrix} 0 & -1 & \omega \end{bmatrix} \rightsquigarrow \varnothing.$$

Linear and lexicographic termination arguments can be combined together to find a linear-lexicographic termination measure. To illustrate this, consider the function ex_4:

$$\begin{array}{lcl} ex_4\ \langle \mathbf{S}\, x, y, z, v, w\rangle & = & ex_4\ \langle x, h\, y, z, v, \mathbf{S}\, w\rangle \\ ex_4\ \langle x, \mathbf{S}\, y, \mathbf{S}\, z, v, w\rangle & = & ex_4\ \langle h\, x, y, z, \mathbf{S}\, v, \mathbf{S}\, w\rangle \\ ex_4\ \langle x, y, z, \mathbf{S}\,\mathbf{S}\,\mathbf{S}\, v, w\rangle & = & ex_4\ \langle x, h\, y, \mathbf{S}\, z, v, \mathbf{S}\, w\rangle \end{array}$$

where h represents an arbitrary function. The following is the difference matrix for ex_4:

$$\begin{bmatrix} -1 & \omega & 0 & 0 & 1 \\ \omega & -1 & -1 & 1 & 1 \\ 0 & \omega & 1 & -3 & 1 \end{bmatrix}.$$

Solving it requires both the linear and lexicographic aspects of our approach. This is again a situation where the pure lexicographic algorithm is not applicable. Moreover, in this case, one cannot find a positive rational linear combination of all the columns that results in a vector that is strictly less than 0 in all its entries. Using both ideas, we want to find some linear combination of the last three columns in the matrix, so as to eliminate some of the ω entries. Fortunately, it is not difficult to see that, labelling the columns of the matrix $c_1, c_2, \ldots, c_5$, one can take $2c_3 + c_4 + 0c_5 = (0 \ -1 \ -1)^\mathsf{T}$. (Where $(-)^\mathsf{T}$ is vector/matrix transposition.) This yields the following matrix

$$\begin{bmatrix} -1 & \omega & 0 \\ \omega & -1 & -1 \\ 0 & \omega & -1 \end{bmatrix}.$$

We now use the lexicographic algorithm to remove the last column and bottom-most two rows, resulting in the matrix $\begin{bmatrix} -1 & \omega \end{bmatrix}$, and then to reduce this matrix to $\varnothing$. Thus we have obtained a termination proof. The corresponding measure that decreases at every recursive call is $[(2 \cdot (\mathbf{size} \triangleleft \pi_3) + (\mathbf{size} \triangleleft \pi_4)), (\mathbf{size} \triangleleft \pi_1)]_{\text{lex}}$.

6.1 Primitive Measure Generation

Each term has several primitive measures, which are generated by observing how that term is destructured in the definition of the recursive function. These primitive measures are made of a composition of two parts: a function which descends through the non-recursive part of a term to extract the recursively constructed term, and a measure of that recursive term.

For example, take the two terms $\langle \mathbf{roll}\,\{\mathbf{inr}\ x\}, \langle\rangle\rangle$ and $\langle x, \langle\rangle\rangle$, and assume the former is the input argument to a function, and the second is the recursive argument. We can clearly see that the structural size of the left part of the tuple decreases from input to recursive call, due to the removal of the **roll**. We capture this intuition with the primitive measure function $(\mathbf{fix}\,(\mathbf{uninr} \triangleleft \mathbf{unroll})) \triangleleft \pi_1$. The purpose of this measure is to extract the recursive part of the argument, i.e. to remove the $\langle -, \langle\rangle\rangle$ structure, and then to count the number of nestings of the structure $\mathbf{roll}\,\{\mathbf{inr}\ -\}$.

Primitive measure generation is composed of two parts: generating the recursive part of the measure and generating the function that extracts the recursive subterm. The function $primM_{\text{R}}$ generates the recursive parts of the primitive measures. It takes a variable name x, a partially constructed recursive measure m_r, and a term t. The measure is initialised with id for notational convenience where $m \triangleright \text{id} = m$; in our implementation we use lists of measure destructors.

The recursive measures are generated by the function

$$
\begin{array}{lcl}
primM_{\mathrm{R}}(x, m_r, \mathbf{var}\, y) & = & \begin{cases} \{\mathbf{fix}\, m_r\} & \mathbf{if}\ x = y \wedge m_r \neq \mathrm{id} \\ \emptyset & \mathbf{if}\ x \neq y \vee m_r = \mathrm{id} \end{cases} \\
primM_{\mathrm{R}}(x, m_r, \langle\rangle) & = & \emptyset \\
primM_{\mathrm{R}}(x, m_r, \langle t_1, t_2\rangle) & = & primM_{\mathrm{R}}(x, (\pi_1 \triangleleft m_r), t_1) \\
 & & \cup\, primM_{\mathrm{R}}(x, (\pi_2 \triangleleft m_r), t_2) \\
primM_{\mathrm{R}}(x, m_r, \mathbf{inl}\ t) & = & primM_{\mathrm{R}}(x, \mathbf{uninl} \triangleleft m_r, t) \\
primM_{\mathrm{R}}(x, m_r, \mathbf{inr}\ t) & = & primM_{\mathrm{R}}(x, \mathbf{uninr} \triangleleft m_r, t) \\
primM_{\mathrm{R}}(x, m_r, \mathbf{roll}\,\{t\}) & = & primM_{\mathrm{R}}(x, \mathbf{unroll} \triangleleft m_r, t) \\
primM_{\mathrm{R}}(x, m_r, \mathbf{app}\, f\, t) & = & \emptyset.
\end{array}
$$

Consider the example above comparing x and $\mathbf{roll}\,\{\mathbf{inl}\ x\}$. As expected, the generator produces the set $\{\mathbf{fix}\,(\mathbf{uninl} \triangleleft \mathbf{unroll})\}$. A more complex example is comparing x and $\mathbf{roll}\,\{\langle x, x\rangle\}$; the measure set generated is $\{\mathbf{fix}\,(\pi_1 \triangleleft \mathbf{unroll}), \mathbf{fix}\,(\pi_2 \triangleleft \mathbf{unroll})\}$ with a generated measure for each occurrence of x.

We can now generate a set of measures for recursive terms. However, we still need to generate the part of the measure that extracts the subterm where the recursion occurs. This part of the measure is appended to the recursive measures generated by calling $primM_{\mathrm{R}}$. It is constructed by recursion on both the initial and recursive arguments.

$$
\begin{array}{lcl}
primM(m, \mathbf{var}\, x, u) & = & \{r \triangleleft m \mid r \in primM_{\mathrm{R}}(x, \mathrm{id}, u)\} \\
primM(m, t, \mathbf{var}\, y) & = & \{r \triangleleft m \mid r \in primM_{\mathrm{R}}(y, \mathrm{id}, t)\} \\
primM(m, \langle\rangle, \langle\rangle) & = & \emptyset \\
primM(m, \langle t_1, t_2\rangle, \langle u_1, u_2\rangle) & = & primM(\pi_1 \triangleleft m, t_1, u_1) \\
 & & \cup\, primM(\pi_2 \triangleleft m, t_2, u_2) \\
primM(m, \mathbf{inl}\ t, \mathbf{inl}\ u) & = & primM(\mathbf{uninl} \triangleleft m, t, u) \\
 & & \cup\, \{\mathbf{case01} \triangleleft m, \mathbf{case10} \triangleleft m\} \\
primM(m, \mathbf{inr}\ t, \mathbf{inr}\ u) & = & primM(\mathbf{uninr} \triangleleft m, t, u) \\
 & & \cup\, \{\mathbf{case01} \triangleleft m, \mathbf{case10} \triangleleft m\} \\
primM(m, \mathbf{roll}\,\{t\}, \mathbf{roll}\,\{u\}) & = & primM(\mathbf{unroll} \triangleleft m, t, u) \\
primM(m, t, u) & = & \emptyset \qquad\qquad \mathbf{otherwise}
\end{array}
$$

Note that sums, in addition to the measures of recursive size-change of the components, also have measures to detect when a sum switches from left to right (or vice-versa). Due to these measures and the fact we are finding linear combinations of primitive measures, we do not need to generate quadratically many measures for the combinations of the left and right submeasures, as done by others [7].

To generate the primitive measures for a function f, we first apply *primM* to every initial and recursive argument pair in the equations defining f, initialising m with id. For each measure m generated through this process, if m is of the form $(\mathbf{fix}\ \ (m' \triangleleft \cdots \triangleleft m')) \triangleleft m''$, with the m' repeated, we simplify it to $(\mathbf{fix}\ \ m') \triangleleft m''$. This step is useful in cases where, for example, a function removes the same pattern multiple times; like ex_2, which removes multiple **S** constructors. This simplification step ensures the simplified measure function only removes one of these patterns at a time. The simplified measure returns the same size as $[\![m]\!]$ when $[\![m]\!]$ is defined, and it is defined for more terms; this property follows from the definition of the evaluation function.

With this method for primitive measure generation in hand, we return to the *toList* example from Section 1. To demonstrate how this measure generation works, we first need to unfold the SparseList and List constructors in *toList* according to Figure 5,

$$\begin{array}{llll} \mathbf{Nil} & = \mathbf{roll}\,\{\mathbf{inl}\,\langle\rangle\} & \mathbf{SNil} & = \mathbf{roll}\,\{\mathbf{inl}\,\langle\rangle\} \\ \mathbf{Cons}\ x\ xs & = \mathbf{roll}\,\{\mathbf{inr}\,\langle x, xs\rangle\} & \mathbf{SCons}\ x\ n\ xs & = \mathbf{roll}\,\{\mathbf{inr}\,\langle x, \langle n, xs\rangle\rangle\} \end{array}$$

Figure 5 Encodings of lists and sparse lists.

$$\begin{array}{lcl} \mathit{toList}\ (\mathbf{roll}\,\{\mathbf{inl}\,\langle\rangle\}) & = & \mathbf{roll}\,\{\mathbf{inl}\,\langle\rangle\} \\ \mathit{toList}\ (\mathbf{roll}\,\{\mathbf{inr}\,\langle x, \langle \mathbf{roll}\,\{\mathbf{inl}\,\langle\rangle\}, xs\rangle\rangle\}) & = & \mathit{toList}\ xs \\ \mathit{toList}\ (\mathbf{roll}\,\{\mathbf{inr}\,\langle x, \langle \mathbf{roll}\,\{\mathbf{inr}\ n\}, xs\rangle\rangle\}) & = & \mathbf{roll}\,\{\mathbf{inr}\,\langle x, \langle \mathit{toList}\ (\mathbf{roll}\,\{\mathbf{inr}\,\langle x, \langle n, xs\rangle\rangle\})\rangle\rangle\}. \end{array}$$

The primitive measure generation algorithm produces measures corresponding to the recursive uses of both n and xs. The two measures, along with their difference matrix, are

$$\begin{array}{ll} m_0 = \mathbf{fix}\,(\pi_2 \triangleleft \pi_2 \triangleleft \mathbf{uninr} \triangleleft \mathbf{unroll}) & \\ m_1 = \mathbf{fix}\,(\mathbf{uninr} \triangleleft \mathbf{unroll}\,) \triangleleft (\pi_1 \triangleleft \pi_2 \triangleleft \mathbf{uninr} \triangleleft \mathbf{unroll}) & \begin{bmatrix} -1 & \omega \\ 0 & -1. \end{bmatrix} \end{array}$$

These can then be combined into an overall decreasing measure as $[m_0, m_1]_{\text{lex}}$.

6.2 Primitive Measure Difference Matrix

The fundamental data structure used in our termination-checking algorithm, T-Rex, is the *primitive measure difference matrix*, $D : (\mathbb{Z} \uplus \{\omega\})^{n \times k}$, or *difference matrix* for short. Rows in D represent argument-pairs and columns represent primitive measures. The elements are the conservative size change between the initial argument and the recursive argument. Formally, for any function definition (with n recursive calls), let the recursive call matrix of that function be $R : (T)^{n \times 2}$ where R_{i1} is the pattern of the i-th recursive call (lifted to a term) and R_{i2} is the recursive argument term of the i-th recursive call. We define the difference matrix D of a vector of k primitive measures, $\mathbf{m} : M^k$, as $D_{ij} = (\mathbf{m}_j, R_{i2}, 0)\Downarrow \mathbin{\dot{-}^\star} (\mathbf{m}_j, R_{i1}, 0)\Downarrow$ where $1 \le i \le n$ and $1 \le j \le k$. This matrix captures the size-change information of the measures $\mathbf{m}$ on every recursive call in the function. Note that it uses the upper bound difference, $(\mathbin{\dot{-}^\star})$, and so the estimated difference can be unbounded, i.e. ω.

Returning to the function ex_2, the entries of the difference matrix evaluate to

$$\begin{array}{llll} m_{\mathbf{S}}\ x & \mathbin{\dot{-}^\star} & (m_{\mathbf{S}}\ x + 2) & = -2 \\ (m_{\mathbf{S}}\ x + 1) & \mathbin{\dot{-}^\star} & m_{\mathbf{S}}\ x & = 1 \end{array} \quad \text{and} \quad \begin{array}{llll} m_{\mathbf{S}}\ y & \mathbin{\dot{-}^\star} & (m_{\mathbf{S}}\ y + 2) & = -2 \\ (m_{\mathbf{S}}\ y + 1) & \mathbin{\dot{-}^\star} & m_{\mathbf{S}}\ y & = 1 \end{array}$$

where $m_{\mathbf{S}} = \mathbf{fix}\,(\mathbf{uninr} \triangleleft \mathbf{unroll})$,

This results in the following difference matrix:

$$\begin{bmatrix} -2 & 1 \\ 1 & -2 \end{bmatrix}.$$

In example ex_3, the arbitrary functions result in the measures becoming incomparable, resulting in ω. Consider the reduced differences that do not result in 0,

$$\begin{array}{llll} m_{\mathbf{S}}\ y & \mathbin{\dot{-}^\star} & (m_{\mathbf{S}}\ y + 1) & = -1 \\ m_{\mathbf{S}}\ (h\ y) & \mathbin{\dot{-}^\star} & 1 & = \omega \\ m_{\mathbf{S}}\ (h\ x) & \mathbin{\dot{-}^\star} & 1 & = \omega \end{array}$$

where $m_{\mathbf{S}} = \mathbf{fix}\,(\mathbf{uninr} \triangleleft \mathbf{unroll})$.

When the measure of the recursive argument, on the left-hand side, is not a concrete integer (e.g. $m_{\mathbf{S}}\ (h\ y)$), but the right hand side is, we can only pessimistically conclude that the value could be any large natural number. Thus we return ω, to mark that we cannot handle this case using linear arithmetic.

6.3 Finding Linear Measures

We wish to use our primitive measures to create better measures; that is, measures that decrease on more recursive calls. We will first consider *linear* combinations of measures. As we discussed in the last section, the difference matrix captures the size change information of the arguments as they change between recursive calls. This information helps determine weights that can be used to construct a linear combination of the primitive measures. Note that this phase only considers D matrices that contain no ω values. This enables applying standard linear solvers. The ω values are handled at a later phase.

To find suitable weights, first we will show that *any* vector of positive weights $\mathbf{w} : (\mathbb{Z}^+)^n$ that satisfies the equation $D\mathbf{w} \leq 0$ produces a non-increasing measure. Secondly, we will determine how to pick the *best* such vector: that is, the one that produces a measure that *strictly* decreases between the arguments of as many recursive calls as possible.

Decreasing linear combinations

The following lemma shows that any non-positive solution to $D\mathbf{w}$ can be used to construct a non-increasing measure.

▶ **Lemma 2** (Soundness of linear measure construction). *Given a vector of positive weights $\mathbf{w} : (\mathbb{Z}^+)^n$, a vector of measures $\mathbf{m} : M^n$, if there are no ω values in D and $(D\mathbf{w})_i \leq 0$ for row i, then for each initial-recursive argument pair R_i, and all substitutions θ where, for every j, $[\![\mathbf{m}_j]\!]\ \theta(R_{i1})$ and $[\![\mathbf{m}_j]\!]\ \theta(R_{i2})$ are defined, then $(\mathbf{w} \cdot [\![\mathbf{m}]\!])\ \theta(R_{i2}) - (\mathbf{w} \cdot [\![\mathbf{m}]\!])\ \theta(R_{i1}) \leq 0$. Similarly, if $(D\mathbf{w})_i < 0$, then $(\mathbf{w} \cdot [\![\mathbf{m}]\!])\ \theta(R_{k2}) - (\mathbf{w} \cdot [\![\mathbf{m}]\!])\ \theta(R_{k1}) < 0$. (Where $[\![-]\!]$ is lifted pointwise.)*

To give an example, we return to the function ex_2, which has the difference matrix

$$\begin{bmatrix} -2 & 1 \\ 1 & -2 \end{bmatrix}.$$

The weight vector $\mathbf{w} = (1\ 1)^\mathsf{T}$ produces the all-negative vector $D\mathbf{w} = (-1\ \ -1)^\mathsf{T}$. As such, we can conclude that the measure $(1\ 1)^\mathsf{T} \cdot (m_1\ m_2)^\mathsf{T}$ guarantees that ex_2 terminates.

The maximal negative entries problem

This problem is concerned with finding the best weights $\mathbf{w}$, that ensure that the maximum number of recursive calls decrease. To see why not just any weight will do, consider again the example ex_4. We showed that its matrix,

$$\begin{bmatrix} -1 & \omega & 0 & 0 & 1 \\ \omega & -1 & -1 & 1 & 1 \\ 0 & \omega & 1 & -3 & 1 \end{bmatrix}$$

was solvable using our proposed algorithm. However, had we chosen instead the linear combination $(1\ 1\ 0)^\mathsf{T}$, this would have resulted in the output vector $(0\ 0\ \ -2)^\mathsf{T}$, which only removes the final recursive call / row, resulting in the unsolvable subproblem:

$$\begin{bmatrix} -1 & \omega \\ \omega & -1 \end{bmatrix}.$$

Thus, we want to pick weights such that not only $D\mathbf{w} \leq 0$, but also that $D\mathbf{w}$ has as many negative entries as possible. We call this the *maximal negative entries* (MNE) problem.

▶ **Definition 3** (The Maximal Negative Entries Problem). *For a matrix A, find a vector $\mathbf{x}$ such that the vector $A\mathbf{x}$ has a maximal number of negative entries.*

Reducing the MNE problem to a linear program

We want to find a programmatic method to solve this problem. This problem can be reduced to solving a linear program. This is fortunate as linear programs are well-studied and they are solvable in polynomial time [5].

The MNE problem tells us to maximise the number of strictly negative entries of $A\mathbf{x}$, such that $\mathbf{x} \geq 0$ and $A\mathbf{x} \leq 0$. We can reframe this as finding some slack vector $\mathbf{c} \geq 0$, such that $A\mathbf{x} + \mathbf{c} = 0$ and $\mathbf{c}$ has a maximal number of *positive* entries (as $A\mathbf{x} = -\mathbf{c}$).

Secondly, note that solutions to this problem are linear, in the sense that if $(\mathbf{x}, \mathbf{c})$ is a solution to $A\mathbf{x} + \mathbf{c} = 0$ and $\mathbf{c}$ has maximal positive entries, then, for a positive scalar k, $A(k\mathbf{x}) + k\mathbf{c} = k(A\mathbf{x} + \mathbf{c}) = 0$, and so $(k\mathbf{x}, k\mathbf{c})$ is also a solution. The crucial thing to note here is that we can scale up or down the size of any such solution vector by any positive scaling factor. This means that a solution exists exactly when a solution exists that satisfies the additional condition that $\mathbf{c}$ has entries that are either exactly 0 or greater than or equal to 1. This allows us to take the final step in our transformation of this problem.

Break up $\mathbf{c}$ (with entries now guaranteed to be 0 or $\geq$1) into $\mathbf{b} + \mathbf{z}$, where b is bounded between 0 and 1 and $0 \leq z$, The optimisation goal "maximise the sum of $\mathbf{b}$" will produce a solution $(\mathbf{x}, \mathbf{b}, \mathbf{z})$ where $\mathbf{b}$ has entries that are either 0 or 1 (as $\mathbf{b} + \mathbf{z}$ is 0 or ≥ 1) and $\mathbf{z}$ is the slack necessary to make the sum cancel. Note that $\mathbf{b}_i$ is 1 exactly when $(A\mathbf{x})_i < 0$, thus the number of negative entries in $A\mathbf{x}$ is equal to the sum of $\mathbf{b}$. As the sum of b is maximised, the number of negative entries in $A\mathbf{x}$ is maximised.

Lastly, as $\mathbf{z}$ is only constrained to be ≥ 0, $\mathbf{z}$ is a slack vector of this new program. We may simplify the program by removing $\mathbf{z}$ and turning the equality constraints to inequality constraints. Thus, the maximal negative entries problem is equivalent to solving the linear programming problem specified in Algorithm 1. Note that the program *fails* to find a satisfactory solution for our purposes when it returns the all-zero vector.

Algorithm 1 LinMNE: A linear program for the MNE problem.

$$
\begin{array}{rcl}
\textbf{maximise} & \sum_{i=0}^{n-1} \mathbf{b}_i & \\
\textbf{subject to} & & \\
\mathbf{x}_i, \mathbf{b}_i & \in & \mathbb{Q} \\
A\mathbf{x} + \mathbf{b} & \leq & 0 \\
0 & \leq & \mathbf{x}_i \\
0 & \leq & \mathbf{b}_i \leq 1
\end{array}
$$

Uniqueness of the maximal negative entry set

Even though the above program will find a set with a maximal number of negative entries, we have not yet established that such a set is unique. If there are two solutions that identify a different set of rows that decrease, this may lead to a different order of row elimination in the lexicographic stage of our algorithm. Such a possibility threatens the complexity and completeness of our algorithm. However, there is, in fact, a *unique* maximal set of negative entries that solves the linear program. Note that this uniqueness is in the set of entries, not in the solution; there can be many solutions, but all of them will show the *same* set of recursive calls decrease.

▶ **Theorem 4** (Unique Maximal Negative Entries). *For every matrix $A \in \mathbb{Q}^{n \times m}$, if $\langle \mathbf{x}, \mathbf{b} \rangle$ and $\langle \mathbf{x}', \mathbf{b}' \rangle$ are solutions of the MNE problem for A, then $\mathbf{b} = \mathbf{b}'$.*

6.4 Finding Lexicographic Measures of Linear Combinations

The lexicographic phase of the algorithm solves the termination problem for a matrix of weights that may include ω; the value that marks that a finite bound on the size change could not be found. The algorithm finds a lexicographic combination of linear measures, such that whenever a linear measure was approximated to ω, there is a lexicographically larger linear measure that returns a natural number. This lexicographic algorithm follows the same structure as Isabelle/HOL's [7]. Our overall **T-Rex** algorithm proceeds as follows:

Algorithm 2 The T-Rex Termination Algorithm.

function T-Rex($m : [M], D :$ Matrix $(\mathbb{Z} \uplus \omega)) : [[\mathbb{N} \times M]] \uplus$ **fail**
  (Precondition: length$(m) =$ columnN(D).)
  $out \coloneqq []$
  repeat
    $N \coloneqq$ Extract the purely numeric ($\mathbb{Z}$) columns of D
    if $N = \varnothing$ **then return fail end if** ▷ (if there are no purely numeric columns, fail)
    $(x, b) \coloneqq$ LinMNE(N)
    $x \coloneqq$ select non-zero weights in x
    if $x = \varnothing$ **then return fail end if** ▷ (if no measures were selected, fail)
    $x \coloneqq x \cdot$ lcm(denoms(w)) ▷ (normalise the weights to natural numbers)
    $D \coloneqq D$ without rows where b is 1
    (Note: recall the linear program establishes that $(D\mathbf{x})_i$ is negative when $\mathbf{b}_i = 1$.)
    $wm \coloneqq$ weights in x paired with their corresponding measures in m
    $out \coloneqq out ::_{\mathrm{r}} wm$ ▷ ($::_{\mathrm{r}}$ is concatenate to end)
  until $M = \varnothing$
  return out
end function
(Note: we elide the tracking of column indices for clarity.)

Since linear program solving is polynomial time [5] and the number of loops performed is bounded by the number of rows in D, we can deduce that our algorithm also runs in polynomial time. But of course, we would like to know slightly more than just the correctness and complexity of our algorithm. Our algorithm is complete in the following sense. If a lexicographic-linear combination of primitive measures exists, our algorithm decides termination.

▶ **Lemma 5.** *Let f be a function and let A be a matrix whose columns correspond to some set of measures P whose elements we allow to be any $\mathbb{N}$-linear combination of the columns of the difference matrix of f. Then, there exists a lexicographic combination of the elements of P which decreases at every recursive call of f if and only if we can reduce A to $\emptyset$ by removing rows per the lexicographic algorithm.*

▶ **Theorem 6** (Decision Power of T-Rex)**.** *Given a function f with an associated set of primitive measures P, if there exists some lexicographic order of $\mathbb{N}$-linear combinations of elements of P which decreases at every recursive call of f, then **T-Rex** will succeed.*

7 Related Work

There is a relevant body of work in the context of analysing the termination of imperative programs called synthesis of ranking functions. This was initially introduced by Floyd [10], and later incorporated into Hoare logic to allow for proving the total correctness of imperative programs. Ranking functions are directly analogous to measures. In the context of imperative programs, ranking functions map variables that are updated in a loop body to a well-ordered set, and to prove termination they must decrease at each iteration. Typically in this context, the termination of a single loop with a single branch is studied, which broadly corresponds in our context to studying functions with a single recursive call.

A line of work on termination analysis is concerned with finding classes of programs (e.g. programs that operate on reals) with *linear loops* for which termination is decidable, by relating this problem to stability in control theory [20]. Part of the linear loops literature is concerned with synthesising ranking functions for linear-constraint loops [6, 19]. Here, we have a set of variables, a loop guard describing a linear constraint on these variables, and at each iteration, we have updated our vector of variables $\mathbf{x}$ by an affine transformation $A\mathbf{x} + \mathbf{b}$ [13]. (These affine transformations can always be converted to a linear transformation $A'\mathbf{x}$ by adding more variables.) The kind of ranking functions synthesised for these problems are similar to the lexicographic combinations of linear transformations that we see in our work [6]. One main distinction is that in our work, there is almost always more than one recursive call, and in their work, there is almost always some non-trivial condition on the loop guard, providing different challenges and resulting in different algorithms for handling termination within different programming language paradigms.

Linear and lexicographic termination techniques have also been explored in the context of probabilistic imperative programming. Similar algorithms to the measure combination step of the algorithm, utilising linear programming to combine size-change information, have been independently developed in this context [4]. However, a direct comparison between them presents a non-trivial challenge as this work operates on generalised transition systems generated from analysis of imperative code, while our work operates on primitive measures. Even given these similarities, our improved method of primitive measure generation handles cases that the Isabelle/HOL algorithm does not, even with the purely lexicographic solver.

Our work is directly inspired by and most closely related to Isabelle/HOL's lexicographic termination algorithm [2, 7]. It constructs termination matrices based on the comparison of arguments between the initial and recursive calls, as we do, but they do not compute a numeric difference. Since this approach ignores the numerical difference and conflates increase with uncertainty, it cannot be used to find linear combinations of measures. While our matrices carry more information, our lexicographic matrix elimination algorithm is structurally similar to Bulwahn et. al.'s lexicographic matrix elimination algorithm [7]. Note that we do not drop columns, as this can cause unsoundness if the final column is not wholly decreasing. Our algorithm also works without access to types or higher-order logic simplification theories. Hence, it is extensible to various functional languages and can be used without theorem-proving experience.

Another approach is the size-change termination method [16, 12], which is orthogonal to our method, as it tracks the size change of *data* (i.e. the constructors around variables) in function arguments, whereas we track the size change of the function arguments, disregarding the flow of variables. Types can be used to restrict functions to use terminating recursion [1], but this method requires user annotation, and cannot find complex linear termination measures.

Other approaches for functional termination checking involve using an external term-rewriting checker to show the termination of function programs [15]. This has the advantage of bringing well-developed termination checkers for term rewriting systems to functional termination checking, but requires that the semantics of the functional language be reflected in a term rewriting system. Further, any witness of termination must be translated backwards through this semantics. Our approach gives simple measure functions as witnesses.

8 Conclusion

We have developed T-Rex: a novel termination algorithm for recursive functions. It can prove the termination of functions by lexicographically ordering linear combinations of primitive measures that decrease at every recursive call. The primitive measures are computed directly

by examining the structure of the recursive program. We define a language for simplifying and comparing primitive measures that are used as part of our termination algorithm and prove meta-theoretic properties of our measure language. We prove that T-Rex is sound, that it runs in polynomial time and that it covers a large class of programs and demonstrate the algorithm on an untyped first-order functional language. We provide an implementation of the untyped language, measure language and of T-Rex in Haskell.

References

1 Andreas Abel. Termination checking with types. *RAIRO - Theoretical Informatics and Applications*, 38(4):277–319, 2004. doi:10.1051/ita:2004015.

2 Andreas Abel and Thorsten Altenkirch. A predicative analysis of structural recursion. *Journal of Functional Programming*, 12(1):1–41, 2002. doi:10.1017/S0956796801004191.

3 Wilhelm Ackermann. Zum hilbertschen aufbau der reellen zahlen. *Mathematische Annalen*, 99:118–133, 1928. URL: https://api.semanticscholar.org/CorpusID:123431274.

4 Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotnỳ. Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. *Proceedings of the ACM on Programming Languages*, 2(POPL):1–32, 2017.

5 Bengt Aspvall and Richard E Stone. Khachiyan's linear programming algorithm. *Journal of Algorithms*, 1(1):1–13, 1980. doi:10.1016/0196-6774(80)90002-4.

6 Amir M Ben-Amram and Samir Genaim. Ranking functions for linear-constraint loops. *Journal of the ACM (JACM)*, 61(4):1–55, 2014.

7 Lukas Bulwahn, Alexander Krauss, and Tobias Nipkow. Finding Lexicographic Orders for Termination Proofs in Isabelle/HOL. In Klaus Schneider and Jens Brandt, editors, *Theorem Proving in Higher Order Logics*, volume 4732, pages 38–53. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISSN: 0302-9743, 1611-3349 Series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-540-74591-4_5.

8 Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:354–363, 1936.

9 Raphael Douglas Giles and Vincent Jackson. T-Rex. Software, version 1.0.0., swhId: swh:1:dir:c7dd8ada9bdd696f86cd11bc6751817ed37ad120 (visited on 2024-06-18). URL: https://github.com/vjackson725/term_check.

10 R. W. Floyd. Assigning meaning to programs. In J. T. Schwartz, editor, *Mathematical aspects of computer science: Proc. American Mathematics Soc. symposia*, volume 19, pages 19–31, Providence RI, 1967. American Mathematical Society.

11 Jürgen Giesl. Automated termination proofs with measure functions. In Ipke Wachsmuth, Claus-Rainer Rollinger, and Wilfried Brauer, editors, *KI-95: Advances in Artificial Intelligence*, pages 149–160, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg. doi:10.1007/3-540-60343-3_33.

12 Pierre Hyvernat. The size-change termination principle for constructor based languages. *Log. Methods Comput. Sci.*, 10(1), 2014. doi:10.2168/LMCS-10(1:11)2014.

13 Toghrul Karimov, Engel Lefaucheux, Joël Ouaknine, David Purser, Anton Varonka, Markus A Whiteland, and James Worrell. What's decidable about linear loops? *Proceedings of the ACM on Programming Languages*, 6(POPL):1–25, 2022.

14 Shmuel M. Katz and Zohar Manna. A closer look at termination. *Acta Informatica*, 5(4):333–352, December 1975.

15 Alexander Krauss, Christian Sternagel, René Thiemann, Carsten Fuhs, and Jürgen Giesl. Termination of Isabelle functions via termination of rewriting. In Marko van Eekelen, Herman Geuvers, Julien Schmaltz, and Freek Wiedijk, editors, *Interactive Theorem Proving*, pages 152–167, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-22863-6_13.

16 Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The size-change principle for program termination. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '01, pages 81–92, New York, NY, USA, 2001. Association for Computing Machinery. doi:10.1145/360204.360210.

17 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*. Number 2283 in Lecture Notes in Computer Science. Springer, 2002.

18 Rozsa Peter. Über die verallgemeinerung der theorie der rekursiven funktionen für abstrakte mengen geeigneter struktur als definitionsbereiche. *Acta Mathematica Academiae Scientiarum Hungaricae*, 12:271–314, 1962. URL: https://api.semanticscholar.org/CorpusID:121988998.

19 Andreas Podelski and Andrey Rybalchenko. A complete method for the synthesis of linear ranking functions. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 239–251. Springer, 2004.

20 Ashish Tiwari. Termination of linear programs. In Rajeev Alur and Doron A. Peled, editors, *Computer Aided Verification*, pages 70–82, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

21 Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936.

22 Christoph Walther. Argument-bounded algorithms as a basis for automated termination proofs. In Ewing Lusk and Ross Overbeek, editors, *9th International Conference on Automated Deduction*, pages 602–621, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.

A Proofs

A.1 Proofs from Section 4

To prove Lemma 1, we need the following lemmas.

▶ **Lemma 7** (Measure Evaluation respects Substitution). *For all substitutions θ,*

1. *if $(m, t, i)\Downarrow = j$ then $(m, \theta(t), i)\Downarrow = j$; and*
2. *if $(m, t, i)\Downarrow = (m', t', j)$ then $(m, \theta(t), i)\Downarrow = (m', \theta(t'), j)\Downarrow$.*

Proof Sketch. By induction on $(m, t, i)\Downarrow$. We will only prove illustrative cases.

Stuck $(m, t, i)\Downarrow = (m, t, i)$: Note that t is a variable, function, or doesn't match m. If t is a variable or function, $\theta(t)$ could be a new term, but in this case $(m, \theta(t), i)\Downarrow = (m, \theta(t), i)\Downarrow$. If t is a non-matching term, then θ cannot change it, and $(m, \theta(t), i)\Downarrow = (m, \theta(t), i)$.

Uninr/Inl $(m \triangleleft \mathbf{uninr}, \mathbf{inl}\ t, i)\Downarrow = i$: as $\theta(\mathbf{inl}\ t) = \mathbf{inl}\ \theta(t)$, $(m \triangleleft \mathbf{uninr}, \theta(\mathbf{inl}\ t), i)\Downarrow = i$. The other base cases proceed similarly.

Unroll/Roll $((m \triangleleft \mathbf{unroll}), (\mathbf{roll}\ \{t\}), i)\Downarrow = (m, t, 1 + i)\Downarrow$:
As $\theta(\mathbf{roll}\ \{t\}) = \mathbf{roll}\ \{\theta(t)\}$,

$$(m \triangleleft \mathbf{unroll}, \theta(\mathbf{roll}\ \{t\}), i)\Downarrow = (m \triangleleft \mathbf{unroll}, \mathbf{roll}\ \{\theta(t)\}, i)\Downarrow = (m, \theta(t), i + 1)\Downarrow,$$

as required.

Fix $((\mathbf{fix}\, m), t, i)\Downarrow = (((\mathbf{fix}\, m) \triangleleft m), t, i)\Downarrow$:

$$((\mathbf{fix}\, m), \theta(t), i)\Downarrow = (((\mathbf{fix}\, m) \triangleleft m), \theta(t), i)\Downarrow,$$

as required.

The other inductive cases proceed similarly. ◀

▶ **Lemma 8** (Measure Evaluation Counter Export).
If $(m, \theta(t), i + j)\Downarrow = k$ iff $(m, \theta(t), i)\Downarrow = j + k$.

Proof Sketch. A simple proof by splitting the iff, then induction on $(-)\Downarrow$. ◀

With these lemmas in hand, we can prove Lemma 1.

Proof of Lemma 1. Recall that we are guaranteed that $(m, \theta(t_1), 0)\Downarrow$ and $(m, \theta(t_2), 0)\Downarrow$ are integers. We proceed by cases on $(m, t_1, 0)\Downarrow \mathbin{\dot{-}} (m, t_2, 0)\Downarrow$.

Case 1: $(m, t_1, 0)\Downarrow = (m', s, i)$ and $(m, t_2, 0)\Downarrow = (m', s, j)$. Then

$$\begin{aligned}
&(m, \theta(t_1), 0)\Downarrow - (m, \theta(t_2), 0)\Downarrow \\
&= (m', \theta(s), i)\Downarrow - (m', \theta(s), j)\Downarrow \quad \text{(Case assumptions \& Lemma 7)} \\
&= ((m', \theta(s), 0)\Downarrow + i) - ((m', \theta(s), 0)\Downarrow + j) \quad \text{(Lemma 8)} \\
&= i - j \\
&= (m', s, i) \mathbin{\dot{-}} (m', s, j) \\
&= (m, t_1, 0)\Downarrow \mathbin{\dot{-}} (m, t_2, 0)\Downarrow.
\end{aligned}$$

Case 2: $(m, t_1, 0)\Downarrow = i$ and $(m, t_2, 0)\Downarrow = (m', s, j)$. Then

$$\begin{aligned}
&(m, \theta(t_1), 0)\Downarrow - (m, \theta(t_2), 0)\Downarrow \\
&= i - (m', \theta(s), j)\Downarrow \quad \text{(Case assumptions \& Lemma 7)} \\
&= i - (m', \theta(s), j)\Downarrow \quad \text{(Lemma 8)} \\
&\leq i - j \\
&= i \mathbin{\dot{-}} (m', s, j) \\
&= (m, t_1, 0)\Downarrow \mathbin{\dot{-}} (m, t_2, 0)\Downarrow.
\end{aligned}$$

Case 3+4: $(m, t_1, 0)\Downarrow \mathbin{\dot{-}} (m, t_2, 0)\Downarrow$ evaluates to ω.

True as $(m, \theta(t_1), 0)\Downarrow - (m, \theta(t_2), 0)\Downarrow$ is an integer, and for all $k \in \mathbb{Z}$, $k \leq \omega$. ◀

A.2 Proofs from Section 6

For the following proof, we need to enrich ω with the properties that $\omega + k = k + \omega = \omega$ for all $k \in \mathbb{Z}$ and $n \cdot \omega = \omega$ for all $n \in \mathbb{Z}^+$.

Proof of Lemma 2. Recall the initial and recursive arguments are stored in R. Thus, for any recursive call in D, indexed by i, we have

$$\begin{aligned}
(D\mathbf{w})_i &= \sum_{j=0}^{k} \mathbf{w}_j \cdot R_{ij} \\
&= \sum_{j=0}^{k} \mathbf{w}_j \cdot ((\mathbf{m}_j, R_{i2}, 0)\Downarrow \mathbin{\dot{-}} (\mathbf{m}_j, R_{i1}, 0)\Downarrow) \\
&\geq \sum_{j=0}^{k} \mathbf{w}_j \cdot ((\mathbf{m}_j, \theta(R_{i2}), 0)\Downarrow - (\mathbf{m}_j, \theta(R_{i1}), 0)\Downarrow) \qquad \text{(Lemma 1)} \\
&= \sum_{j=0}^{k} \mathbf{w}_j \cdot ([\![\mathbf{m}_j]\!]\ \theta(R_{i2})) \ - \ \sum_{j=0}^{k} \mathbf{w}_j \cdot ([\![\mathbf{m}_j]\!]\ \theta(R_{i1})) \\
&= (\lambda x.\ \sum_{j=0}^{k} \mathbf{w}_j \cdot ([\![\mathbf{m}_j]\!]\ x))\ \theta(R_{i2}) \ - \ (\lambda x.\ \sum_{j=0}^{k} \mathbf{w}_j \cdot ([\![\mathbf{m}_j]\!]\ x))\ \theta(R_{i1}) \\
&= (\mathbf{w} \cdot [\![\mathbf{m}]\!])\ \theta(R_{i2}) \ - \ (\mathbf{w} \cdot [\![\mathbf{m}]\!])\ \theta(R_{i1}).
\end{aligned}$$

Thus $(D\mathbf{w})_i \leq 0$ implies $(\mathbf{w} \cdot [\![\mathbf{m}]\!])\ \theta(R_{i2}) - (\mathbf{w} \cdot [\![\mathbf{m}]\!])\ \theta(R_{i1}) \leq 0$, and $(D\mathbf{w})_i < 0$ implies $(\mathbf{w} \cdot [\![\mathbf{m}]\!])\ \theta(R_{i2}) - (\mathbf{w} \cdot [\![\mathbf{m}]\!])\ \theta(R_{i1}) < 0$. ◀

Proof of Theorem 4. Assume there are two solutions with a different set of negative entries $\langle \mathbf{x}, \mathbf{b} \rangle$ and $\langle \mathbf{x}', \mathbf{b}' \rangle$. Let $\mathbf{x}'' = \mathbf{x} + \mathbf{x}'$ and $\mathbf{b}''_j = \max(\mathbf{b}_j, \mathbf{b}'_j)$ for each $0 \leq j < n$. Then, the tuple $\langle \mathbf{x}'', \mathbf{b}'' \rangle$ is a more optimal solution.

Firstly, $\langle \mathbf{x}'', \mathbf{b}'' \rangle$ has a greater objective. As, by assumption, there is some component different between $\mathbf{b}$ and $\mathbf{b}'$, thus we have $\sum_{i=0}^{n-1} \mathbf{b}_i < \sum_{i=0}^{n-1} \mathbf{b}''_i$ and $\sum_{i=0}^{n-1} \mathbf{b}'_i < \sum_{i=0}^{n-1} \mathbf{b}''_i$. Thus the objective is larger.

Secondly, $\langle \mathbf{x}'', \mathbf{b}'' \rangle$ satisfies all constraints, as (i) by the fact we are taking a sum of non-negative values, $0 \leq \mathbf{x}''$, (ii) by the fact we are taking a maximum, $0 \leq \mathbf{b}'' \leq 1$, and (iii) $A\mathbf{x}'' + \mathbf{b}'' \leq 0$ as

$$\begin{aligned} A\mathbf{x}'' + \mathbf{b}'' \leq 0 &\iff A(\mathbf{x} + \mathbf{x}') + \max(\mathbf{b}, \mathbf{b}') \leq 0 \\ &\iff A\mathbf{x} + A\mathbf{x}' + \max(\mathbf{b}, \mathbf{b}') \leq 0 \\ &\impliedby A\mathbf{x} + A\mathbf{x}' + \mathbf{b} + \mathbf{b}' \leq 0 \\ &\impliedby A\mathbf{x} + \mathbf{b} \leq 0 \wedge A\mathbf{x}' + \mathbf{b}' \leq 0 \end{aligned}$$

Thus, our assumption was incorrect, and, by classical contradiction, the two solutions cannot have a different set of negative entries. ◀

To prove Lemma 5, we require the following auxiliary lemma establishing the soundness of lexicographic-linear combination:

▶ **Lemma 9** (Soundness of lexicographic-linear combination)**.** *Given a list of weighted measures* $\mathbf{wm} : [[\mathbb{N} \times M]]$ *returned by the algorithm (Algorithm 2), then for each initial-recursive argument pair* R_k*, and all substitutions* θ *where* $\mathbf{m}_{ij}\ \theta(R_{k1})$ *and* $\mathbf{m}_{ij}\ \theta(R_{k2})$ *are defined (for every* i *and* j*), then*

$$[\ldots, \mathbf{w}_i \cdot [\![\mathbf{m}_i]\!], \ldots]_{\text{lex}}\ \theta(R_{k2}) < [\ldots, \mathbf{w}_i \cdot [\![\mathbf{m}_i]\!], \ldots]_{\text{lex}}\ \theta(R_{k1}).$$

(Where $\mathbf{w}_i$ *and* $\mathbf{m}_i$ *are the unzipping of* $\mathbf{wm}_i$*, and* $(\cdot)$ *and* $[\![-]\!]$ *are lifted pointwise over lists.)*

Proof. By the construction of the output list and Lemma 2, there is a first $\mathbf{w}_i$, $\mathbf{m}_i$ such that $\mathbf{w}_i \cdot [\![\mathbf{m}_i]\!]\ \theta(R_{k2}) < \mathbf{w}_i \cdot [\![\mathbf{m}_i]\!]\ \theta(R_{k1})$ and for every $i' < i$, $\mathbf{w}_{i'} \cdot [\![\mathbf{m}_{i'}]\!]\ \theta(R_{k2}) = \mathbf{w}_{i'} \cdot [\![\mathbf{m}_{i'}]\!]\ \theta(R_{k1})$. This is the definition of a lexicographic decrease ◀

Proof of Lemma 5. Suppose there was some lexicographic combination of elements of P, which we will call $[m_1, m_2, \ldots m_n]_{\text{lex}}$, which decreases at every recursive call of f but which could not be found by removing rows from the associated matrix. At some point, by assumption, this sequence of measures $[m_1, m_2, \ldots m_n]_{\text{lex}}$ must not be reachable by removing rows from the matrix A, per the rule in the lexicographic algorithm. Call the first such measure m_i. Hence, by assumption once we remove all the rows corresponding to measures $m_1, \ldots, m_{i-1}$, we must not be able to proceed using the lexicographic algorithm. Hence, there can be no column in the corresponding matrix with all entries less than or equal to 0 with at least one negative entry, namely the column associated with the measure m_i cannot have this property. But this means that there exists a recursive call on which the measure $[m_1, m_2, \ldots m_i]_{\text{lex}}$ does not decrease, which is a contradiction. ◀

Proof of Theorem 6. Suppose this were not the case and there is a lexicographic order of linear combinations of the primitive measures that could not be found by our algorithm. By definition **T-Rex** can only fail if, at some stage of evaluation, it reaches a stage where there is no non-trivial solution to the MNE problem using the numeric columns of the matrix. But using Theorem 4, we know that each time we take a linear combination of numeric columns, we get a vector whose number of negative entries is maximal and where the location of these negative entries is unique. Hence, for any given numeric part of the matrix, there is a unique set of rows R that is removed by the lexicographic phase of the algorithm such that any other set of rows that could be removed by the process of taking linear combinations is a subset of R. ◀

The 2-Dimensional Constraint Loop Problem Is Decidable

Quentin Guilmant
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Engel Lefaucheux
Loria, Nancy, France

Joël Ouaknine
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

James Worrell
Department of Computer Science, University of Oxford, UK

Abstract

A linear constraint loop is specified by a system of linear inequalities that define the relation between the values of the program variables before and after a single execution of the loop body. In this paper we consider the problem of determining whether such a loop terminates, i.e., whether all maximal executions are finite, regardless of how the loop is initialised and how the non-determinism in the loop body is resolved. We focus on the variant of the termination problem in which the loop variables range over $\mathbb{R}$. Our main result is that the termination problem is decidable over the reals in dimension 2. A more abstract formulation of our main result is that it is decidable whether a binary relation on $\mathbb{R}^2$ that is given as a conjunction of linear constraints is well-founded.

2012 ACM Subject Classification Theory of computation → Program verification

Keywords and phrases Linear Constraints Loops, Minkowski-Weyl, Convex Sets, Asymptotic Expansions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.140

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2405.12992` [6]

Funding *Engel Lefaucheux*: ANR BisoUS (ANR-22-CE48-0012).
Joël Ouaknine: Also affiliated with Keble College, Oxford as emmy.network Fellow, and supported by DFG grant 389792660 as part of TRR 248 (see `https://perspicuous-computing.science`).
James Worrell: Work supported by UKRI Frontier Research Grant EP/X033813/1.

1 Introduction

The problem of deciding loop termination is of fundamental importance in software verification. Deciding termination is already challenging for very simple classes of programs. One such class consists of *linear constraint loops*. These are single-path loops in which both the loop guard and the loop update are given by conjunctions of linear inequalities over the program variables. Such a loop can be written as follows, where A, B are matrices of rational numbers, $\boldsymbol{a}$, $\boldsymbol{b}$ are vectors of rational numbers, and $\boldsymbol{x}, \boldsymbol{x}'$ represent the respective values of the program variables before and after the loop update:

$$\mathsf{P} : \mathsf{while}\ (B\,\boldsymbol{x} \geq \boldsymbol{b})\ \mathsf{do}\ A\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{x}'\end{pmatrix} \geq \boldsymbol{a},$$

Such loops are inherently non-deterministic, since the effect of the loop body is described by a collection of constraints. Note in passing that the loop guard can be folded into the constraints that describe the loop body and so, without loss of generality, the guard can

be assumed to be trivial. Linear constraint loops naturally arise as abstractions of other programs. For example, linear constraints can be used to model size changes in program variables, data structures, or terms in a logic program (see, e.g. [9]).

A linear constraint loop is said to *terminate* if there is no initial value of the loop variables from which the loop has an infinite execution. The Termination Problem asks to decide whether a given loop terminates. As such, the Termination Problem depends on the numerical domain that the program variables range over: typically one considers either $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$.

One approach to proving termination of linear constraint loops involves synthesizing linear ranking functions [2]. However, it is well-known that there are terminating loops that admit no linear ranking function. In the special case of deterministic linear constraint loops (i.e., where the loop body applies an affine function to the program variables) decidability of termination over $\mathbb{R}$ was shown by Tiwari [10], decidability of termination over $\mathbb{Q}$ was shown by Braverman [5], and decidability of termination over $\mathbb{Z}$ was established in [7].[1] All three papers build on an analysis of the spectrum of the matrix that determines the update function in the loop body. To the best of our knowledge, decidability of termination of linear constraint loops over $\mathbb{R}$, $\mathbb{Q}$, and $\mathbb{Z}$ remains open. It is known however that termination for multi-path constraint loops is undecidable (i.e., where disjunctions are allowed in the linear constraints that define the update map). It is moreover known that termination of single-path constraint loops is undecidable if irrational constants are allowed in the constraints [3]. One of the few known positive results is the restricted case that all the constraints are octagonal, in which case termination is decidable over integers [4]. (Recall that a constraint is said to be *octagonal* if it is a conjunction of propositions of the form $\pm x_i \pm x_j \leq a$, for variables x_i, x_j and constant $a \in \mathbb{Z}$.)

In this paper we study the termination of linear constraint loops over the reals in dimension at most 2. We give a sufficient and necessary condition that such a loop be non-terminating in the form of a *witness of non-termination*. This is given in Definition 1. Here one should think of K as the transition relation of a linear constraint loop, while $\mathrm{rec}(K)$ is the *recession cone* of K, i.e., the set of vectors v such that $w + \lambda v \in K$ for all $w \in K$ and $\lambda \geq 0$. The witness of non-termination is essentially a finite representation of an infinite execution of the loop in the spirit of the geometric non-termination arguments of [8] and the recurrent sets of [1].

▶ **Definition 1.** *Let E be a Euclidean space. Let $K \subseteq E^2$ be a convex set. A witness $\mathcal{W}(K)$ consists of a linear map $M : E \to E$, a closed cone $C \subseteq E$, and $v, w \in E$, such that*

($\exists u1$) $MC \subseteq C$

($\exists u2$) $\forall x \in C \qquad (x, Mx) \in \mathrm{rec}(K)$

($\exists u3$) $(v, w) \in K$

($\exists u4$) $w - v \in C$.

If E has dimension at most 2 and K is a polyhedron, then the existence of such a witness can be expressed in the theory of real closed fields. (The restriction to dimension 2 entails that every cone is generated by a most 3 vectors, whereas there is no such upper bound in dimension 3.) Thus we obtain a polynomial-time reduction of the Termination Problem for constraint loops to the decision problem for the theory of real closed fields with a bounded number of quantifier, which is decidable in polynomial space.

[1] These works in fact consider loop guards that feature a mix of strict and non-strict inequalities, whereas in the present paper we consider only non-strict inequalities.

The following is our main result, which characterises non-termination in terms of the above notion of witness. We refer to Section 2.3 for the notion of MW-convex set, suffice to say here that this class includes all polyhedra and that the main property of MW-convex sets used in the proof is that for every linear projection π and MW-convex set K we have $\pi(\mathrm{rec}(K)) = \mathrm{rec}(\pi(K))$. Further background about convex sets is contained in Section 2.2.

▶ **Theorem 2.** *Let E be a Euclidean space of dimension at most 2. Let $K \subseteq E^2$ be MW-convex. There is a sequence $(u_n)_{n\in\mathbb{N}} \in E^{\mathbb{N}}$ such that $(u_n, u_{n+1}) \in K$ for all $n \in \mathbb{N}$ if and only if there exists a witness $\mathcal{W}(K)$.*

2 Preliminaries

2.1 Notation

A superscript $*$ removes 0 from a set. Namely, $\mathbb{R}^* = \mathbb{R} \setminus \{0\}$, $\mathbb{N}^* = \mathbb{N} \setminus \{0\}$ and so on. $\mathbb{R}_+$ stands for all non-negative real numbers and $\mathbb{R}_+^*$ for all the positive real numbers. Also, for $n, m \in \mathbb{N}$ such that $n \leq m$, we let $[\![n \,;\, m]\!]$ be the set of integers between n and m inclusive, namely $[\![n \,;\, m]\!] = \{n, n+1, \ldots, m\}$.

Landau Notations. We use the Landau notations. Let $d \in \mathbb{N}^*$. Let $\|\cdot\|$ be any norm over $\mathbb{R}^d$ (recall that all norms on $\mathbb{R}^d$ are equivalent). Let $u : \mathbb{N} \to \mathbb{R}^d$, $w : \mathbb{N} \to \mathbb{R}^d$ and $v : \mathbb{N} \to \mathbb{R}$ be sequences. We then have the following notations:

- $u_n = \underset{n\to+\infty}{\mathrm{o}}(v_n)$ when for all $\varepsilon \in \mathbb{R}_+^*$ there is some $N \in \mathbb{N}$ such that for all $n \geq N$, we have $\|u_n\| \leq \varepsilon |v_n|$.
- $u_n = \underset{n\to+\infty}{\mathrm{O}}(v_n)$ when there is some $M \in \mathbb{R}_+^*$ and some some $N \in \mathbb{N}$ such that for all $n \geq N$, we have $\|u_n\| \leq M|v_n|$.
- $u_n = \underset{n\to+\infty}{\Omega}(v_n)$ when there is some $M \in \mathbb{R}_+^*$ and some some $N \in \mathbb{N}$ such that for all $n \geq N$, we have $\|u_n\| \geq M|v_n|$.
- $u_n \underset{n\to+\infty}{\sim} w_n$ if $u_n - w_n = \underset{n\to+\infty}{\mathrm{o}}(\|w_n\|)$.
- $u_n = w_n + \underset{n\to+\infty}{\mathrm{o}}(v_n)$ if $u_n - w_n = \underset{n\to+\infty}{\mathrm{o}}(v_n)$.
- $u_n = w_n + \underset{n\to+\infty}{\mathrm{O}}(v_n)$ if $u_n - w_n = \underset{n\to+\infty}{\mathrm{O}}(v_n)$.

We keep the same notations if the sequences are undefined at a finite number of points in $\mathbb{N}$.

2.2 Convex Sets

Throughout this section E is an arbitrary Euclidean space.

These results are already known but for the sake of completeness, some proofs are written here anyway.

▶ **Definition 3.** *Let $S \subseteq E$. The **affine hull** of S, denoted* $\mathrm{AffHull}(S)$*, the **convex hull** of S, denoted* $\mathrm{ConvHull}(S)$*, and the **vector space spanned by** S, denoted* $\mathrm{Vect}(S)$*, are defined by*

$$\begin{aligned}
\mathrm{AffHull}(S) &= \left\{ \sum_{i=1}^{k} \alpha_i x_i \mid \alpha_i \in \mathbb{R}, x_i \in S, \sum_{i=1}^{k} \alpha_i = 1 \right\} \\
\mathrm{ConvHull}(S) &= \left\{ \sum_{i=1}^{k} \alpha_i x_i \mid \alpha_i \in [0;1], x_i \in S, \sum_{i=1}^{k} \alpha_i = 1 \right\} \\
\mathrm{Vect}(S) &= \left\{ \sum_{i=1}^{k} \alpha_i x_i \mid \alpha_i \in \mathbb{R}, x_i \in S \right\}
\end{aligned}$$

▶ **Definition 4.** *Let $K \subseteq E$ be a convex set. The **relative interior** of K, denoted* $\mathrm{ri}(K)$*, is defined by:*

$$\mathrm{ri}(K) = \{\, x \in K \mid \exists U \in \mathcal{O}(E), (x \in U) \wedge (U \cap \mathrm{AffHull}(K) \subseteq K) \}$$

where $\mathcal{O}(E)$ stands for the set of open subsets of E.

In other words, the relative interior of a convex set C is its interior with respect to the induced topology on the affine subspace spanned by C.

We have the following properties for the relative interior:

▶ **Proposition 5.** *Let $K \subseteq E$ be a non-empty convex set. Denoting as usual by $\overline{K}$ the smallest closed subset of E containing K, we have:*

(i) $\mathrm{ri}(K)$ *is a non-empty convex set*
(ii) $\mathrm{ri}(K) \subseteq K \subseteq \overline{K}$
(iii) $\mathrm{AffHull}(\mathrm{ri}(K)) = \mathrm{AffHull}(K)$
(iv) $\mathrm{ri}(K) = \mathrm{ri}(\overline{K})$
(v) $\overline{\mathrm{ri}(K)} = \overline{K}$

▶ **Proposition 6.** *Let K be a non-empty convex set and x, y such that $x \in \mathrm{ri}(K)$ and $y \in K \setminus \mathrm{ri}(K)$. Then for all $\lambda \in (\,0\,;1\,]$ we have $\lambda x + (1-\lambda)y \in \mathrm{ri}(K)$.*

▶ **Definition 7.** *Let $K \subseteq E$ be a non-empty convex set. The **recession cone** of K, denoted* $\mathrm{rec}(K)$*, is the set* $\mathrm{rec}(K) = \{\, z \in E \mid K + \mathbb{R}_+ z \subseteq K \}$.

Note that we always have $0 \in \mathrm{rec}(K)$. Also, the recession cone is indeed a cone, as it is stable under positive scalar multiplication by definition.

▶ **Lemma 8.** *Let $K \subseteq E$ be a convex set. Let $\pi : E \to E$ be a linear projection. Then $\pi(\mathrm{rec}(K)) \subseteq \mathrm{rec}(\pi(K))$.*

Proof. Let $x \in \pi\,(\mathrm{rec}(K))$. There is $y \in \mathrm{Ker}\,\pi$ such that $x + y \in \mathrm{rec}(K)$. Let $a \in \pi(K)$ and $b \in \mathrm{Ker}\,\pi$ such that $a + b \in K$. Then,

$$\begin{aligned}
& \forall \lambda \in \mathbb{R}_+ \qquad (a+b) + \lambda(x+y) \in K \\
\text{Hence,} \quad & \forall \lambda \in \mathbb{R}_+ \qquad a + \lambda x \in \pi(K) \\
\text{and} \quad & x \in \mathrm{rec}(\pi(K)).
\end{aligned}$$

◀

If K is closed, we even have an alternative characterization of the recession cone which requires a seemingly weaker property but that turns out to be equivalent.

▶ **Proposition 9.** *Let $K \subseteq E$ be a non-empty closed convex set. Then*

$$\mathrm{rec}(K) = \{\, z \in E \mid \exists x \in K \quad x + \mathbb{R}_+ z \subseteq K \}$$

Proof. We proceed by double inclusion.

(⊆) This direction is easy : if for all $x \in K$, $x + \mathbb{R}_+ z \subseteq K$, since $K \neq \varnothing$, there is at least one x such that $x + \mathbb{R}_+ z \subseteq K$.

(⊇) Let $z \in \mathbb{R}^d$ such that there is some $x \in K$ such that $x + \mathbb{R}_+ z \subseteq K$. Let $y \in K$. We have to show that for any $t_0 \in \mathbb{R}_+$, $y + t_0 z \in K$. Note that, by convexity, for all $\lambda \in [\,0\,;1\,]$, for all $t \in \mathbb{R}_+$ we have

$$(1-\lambda)y + \lambda(x + tz) \in K$$

$$\text{We then define the function} \qquad \lambda : \begin{cases} [\,t_0\,;+\infty\,) & \to & [\,0\,;1\,] \\ t & \mapsto & \dfrac{t_0}{t} \end{cases}$$

$$\text{hence} \qquad \forall t \geq t_0 \qquad (1-\lambda(t))\, y + \lambda(t) x + t_0 z \in K$$

$$\text{We also have} \qquad (1-\lambda(t))\, y + \lambda(t) x + t_0 z \underset{t \to +\infty}{\longrightarrow} y + t_0 z$$

Since K is closed, we then deduce that for all $y + t_0 z \in K$. Since this holds for any $y \in K$ and any $t_0 \in \mathbb{R}_+$ we end up with $z \in \mathrm{rec}(K)$. ◀

When considering a closed convex set, we can look at its relative interior to get the same recession cone.

▶ **Proposition 10.** *Let $K \subseteq E$ be a non-empty closed convex set. Then* $\mathrm{rec}(K) = \mathrm{rec}(\mathrm{ri}(K))$.

Proof. We proceed by double inclusion.

(⊆) Let $v \in \mathrm{rec}(K)$. Let $x \in \mathrm{ri}(K)$. In particular, $x \in K$. By definition, for any $\lambda \in \mathbb{R}_+$, $x + \lambda v \in K$. Let $S = \{\, \lambda \in \mathbb{R}_+ \mid x + \lambda v \in K \setminus \mathrm{ri}(K) \}$. We just have to show that $S = \varnothing$. Assume $S \neq \varnothing$ and consider $\mu \in S$. Let $\lambda > \mu$. Note that

$$x + \mu v = \left(1 - \frac{\mu}{\lambda}\right) x + \frac{\mu}{\lambda}\,(x + \lambda v)$$

We have two cases:

- $\underline{\lambda \in S}$, in this case, using Proposition 6, since $x \in \mathrm{ri}(K)$ and $x + \lambda v \in K \setminus \mathrm{ri}(K)$, we have $x + \mu v \in \mathrm{ri}\, K$, which is a contradiction.
- $\underline{\lambda \notin S}$, since, by Proposition 5, $\mathrm{ri}(K)$ is convex, $x \in \mathrm{ri}(K)$ and $x + \lambda v \in \mathrm{ri}(K)$, we again reach $x + \mu v \in \mathrm{ri}(K)$, a contradiction.

Both cases are impossible. Therefore, $S = \varnothing$.

(⊇) Let $v \in \mathrm{rec}(\mathrm{ri}(K))$. By Proposition 5, there is some $x \in \mathrm{ri}(K)$. Therefore, for all $\lambda \in \mathbb{R}_+$, $x + \lambda v \in \mathrm{ri}(K) \subseteq K$. By Proposition 9, we get that $v \in \mathrm{rec}(K)$. ◀

▶ Remark 11. Note that if K is not closed we have, thanks to Proposition 5, $\mathrm{rec}(\overline{K}) = \mathrm{rec}(\mathrm{ri}(K))$ but we may have $\mathrm{rec}(K) \neq \mathrm{rec}(\mathrm{ri}(K))$.

▶ **Lemma 12.** *Let C be a closed convex cone in E. Let $u : E \to E$ be linear. Then $u(C)$ is a closed convex cone.*

Proof. By definition of a cone,

$$C = \{0\} \cup \mathbb{R}_+ \{\, x \in C \mid \|x\| = 1 \}\,.$$

Since C is closed, $\{ x \in C \mid \|x\| = 1\}$ is bounded and closed in a vector space of finite dimension, hence it is compact. By linearity of u,

$$u(C) = \{0\} \cup \mathbb{R}_+ u(\{ x \in C \mid \|x\| = 1\}).$$

Since u is linear over a vector space of finite dimension, it is continuous. Thus, the set $u(\{ x \in C \mid \|x\| = 1\})$ is also compact, hence closed. The continuity of the norm ensures that $u(C)$ is closed. By linearity of u, we also get that $u(C)$ is a convex cone. ◀

▶ **Lemma 13.** *Let C be a non-trivial convex cone in E. Let $x \in \mathrm{ri}(C) \setminus \{0\}$ and $u \in \mathrm{Vect}(C)$. Then there is $\lambda \geq 0$ such that $u + \lambda x \in C$.*

Proof. If $x = u$ then $\lambda = 0$ works. We then assume $x \neq u$. Since $u \in \mathrm{Vect}(C)$, there is $\mu \in (\,0\,;1\,)$ $\mu u + (1-\mu)x \in \mathrm{ri}(C)$. Therefore, for any $\lambda \in \mathbb{R}_+$, $\lambda\,(\mu u + (1-\mu)x) \in \mathrm{ri}(C)$. In particular, for $\lambda = \dfrac{1}{\mu}$ (which exists since $\mu \neq 0$),

$$u + \frac{1-\mu}{\mu} x \in \mathrm{ri}(C)$$

and we indeed have $\dfrac{1-\mu}{\mu} \geq 0$. ◀

2.3 Minkowski-Weyl Convex Sets

▶ **Definition 14.** *A closed convex set K is said to be **MW-convex** if there is a compact convex set K' such that $K = K' + \mathrm{rec}(K)$.*

This property comes from the Minkowski-Weyl Theorem for polyhedra :

▶ **Theorem 15** (Minkowski-Weyl)**.** *Let $K \subseteq \mathbb{R}^d$. The following statements are equivalent:*

(i) *$K = \{x \in \mathbb{R}^d \mid Ax \leq b\}$ for some matrix $A \in \mathbb{R}^{n \times d}$ and $b \in \mathbb{R}^n$.*

(ii) *There are finitely many points $x_1, \dots x_k, \in P$ and finitely many directions $v_1, \dots, v_p$ such that*

$$K = \mathrm{ConvHull}(\{x_1, \dots, x_k\}) + \sum_{i=1}^{p} \mathbb{R}_+ v_i.$$

Needing this property, we will assume that the sets K we consider are MW-convex. Note that, among others, polyhedrons are MW-convex, and thus our results apply to a more general class of sets.

One of the main benefits of MW-convex sets is that they behave very nicely with linear projections. Unlike other convex sets, the projections "commute" with the operator rec, giving a reciprocal to Lemma 8.

▶ **Lemma 16.** *Let $K \subseteq \mathbb{R}^d$ be MW-convex. Let π be a linear projection over $\mathbb{R}^d$. We have $\mathrm{rec}(\pi(K)) \subseteq \pi(\mathrm{rec}(K))$.*

Proof. Let $x \in \mathrm{rec}(\pi(K))$. If $x = 0$ then we immediately have $x \in \pi(\mathrm{rec}(K))$. Therefore, we may assume $x \neq 0$. For $a \in \pi(K)$, we have $a + \mathbb{R}_+ x \subseteq \pi(K)$. Thus,

$$\forall \lambda \in \mathbb{R}_+ \quad \exists b(\lambda) \in \mathrm{Ker}\,\pi \qquad a + \lambda x + b(\lambda) \in K.$$

Let K' convex compact such that $K = K' + \mathrm{rec}(K)$. Therefore, for all $\lambda \in \mathbb{R}_+$ there are $a'(\lambda) \in \pi(K')$ and $x'(\lambda) \in \pi(\mathrm{rec}(K))$ such that

$$a + \lambda x = a'(\lambda) + x'(\lambda).$$

Since $a'(\lambda) \in \pi(K')$ and that $\pi(K')$ is compact (as the continuous image of a compact), there is $a' \in \pi(K')$ and an increasing sequence $(\lambda_n)_{n\in\mathbb{N}}$ that tends to infinity such that

$$a'(\lambda_n) \underset{n\to+\infty}{\longrightarrow} a'.$$

Thus

$$\lambda_n x - x'(\lambda_n) \underset{n\to+\infty}{\longrightarrow} a' - a.$$

We then get that

$$\frac{x'(\lambda_n)}{\lambda_n} = x + \frac{a - a'}{\lambda_n} + \underset{n\to+\infty}{\mathrm{o}}\left(\frac{1}{\lambda_n}\right) \qquad \text{and} \qquad \frac{x'(\lambda_n)}{\lambda_n} \underset{n\to+\infty}{\longrightarrow} x.$$

Also $\dfrac{x'(\lambda_n)}{\lambda_n} \in \pi(\mathrm{rec}(K))$. Moreover, using Lemma 12, $\pi(\mathrm{rec}(K))$ is closed. Hence, we have $x \in \pi(\mathrm{rec}(K))$ what concludes the proof. ◀

The converse inclusion is true for general convex sets (Lemma 8). Combining this to Lemma 16, we have:

▶ **Corollary 17.** *Let $K \subseteq \mathbb{R}^d$ be MW-convex. Let π be a linear projection over $\mathbb{R}^d$. We have* $\mathrm{rec}(\pi(K)) = \pi(\mathrm{rec}(K))$.

▶ **Corollary 18.** *Let $K \subseteq \mathbb{R}^d$ be MW-convex. Let $\pi : \mathbb{R}^d \to \mathbb{R}^d$ be a linear projection. Then $\pi(K)$ is MW-convex.*

Proof. We write $K = K' + \mathrm{rec}(K)$ where K' is a convex compact set. Hence, since π is continuous (linear in a finite dimensional space), $\pi(K')$ is also compact. Moreover, by linearity of π, we get that

$$\pi(K) = \pi(K') + \pi(\mathrm{rec}(K)).$$

By Lemma 12, $\pi(\mathrm{rec}(K))$ is a closed convex cone. Hence, $\pi(K)$ is closed convex as a sum of closed convex sets. By Corollary 17, we get

$$\pi(K) = \pi(K') + \mathrm{rec}(\pi(K)).$$ ◀

2.4 Accumulation Expansions

We consider an arbitrary Euclidean space E of dimension $d \in \mathbb{N}$. We denote $\langle \cdot, \cdot \rangle$ its scalar product and $\|\cdot\|$ the associated norm.

To study the sequences of the constraint loop problem, we need to identify the asymptotic directions these sequences are going towards, building a form of asymptotic expansion of those sequences. We thus introduce the concept of accumulation expansion. As sequences may point in several directions, we consider the expansion of a subsequence that has a single main direction.

▶ **Definition 19.** *Let $(u_n)_{n\in\mathbb{N}}$ be a sequence of E. An **accumulation expansion** of $(u_n)_{n\in\mathbb{N}}$ consists in an increasing function $\psi : \mathbb{N} \to \mathbb{N}$, an integer $p \in [\![0 \ ; \ d]\!]$, some vectors $z_1, \dots, z_{p+1} \in E$ and sequences $(\alpha_{k,n})_{n\in\mathbb{N}}$ for $k \in [\![1 \ ; \ p]\!]$ such that*

(AE1) $\forall k \in [\![1 \ ; \ p]\!] \qquad \|z_k\| = 1$

(AE2) $\forall k, k' \in [\![1 \ ; \ p]\!] \qquad \langle z_k, z_{k'} \rangle = \begin{cases} 1 & \text{if } k = k' \\ 0 & \text{if } k \neq k' \end{cases}$

(AE3) $\forall k \in [\![1 \ ; \ p]\!] \qquad \langle z_k, z_{p+1} \rangle = 0$

(AE4) $\forall k \in [\![1 \ ; \ p]\!] \qquad \forall n \in \mathbb{N} \qquad \alpha_{k,n} > 0$

(AE5) $\forall k \in [\![1 \ ; \ p]\!] \qquad \alpha_{k,n} \underset{n\to+\infty}{\longrightarrow} +\infty$

(AE6) $\forall m \in [\![1 \ ; \ p]\!] \qquad \alpha_{m,n} \underset{n\to+\infty}{\sim} \left\| u_{\psi(n)} - \sum_{k=1}^{m-1} \alpha_{k,n} z_k \right\|$

(AE7) $\forall k \in [\![1 \ ; \ p-1]\!] \qquad \alpha_{k+1,n} = \underset{n\to+\infty}{o}(\alpha_{k,n})$

(AE8) $\forall n \in \mathbb{N} \qquad \forall \ell \leq m \in [\![1 \ ; \ p]\!] \qquad \left\langle z_\ell, u_{\psi(n)} - \sum_{k=1}^{m} \alpha_{k,n} z_k \right\rangle = 0$

(AE9) $u_{\psi(n)} = \sum_{k=1}^{p} \alpha_{k,n} z_k + z_{p+1} + \underset{n\to+\infty}{o}(1)$.

Abusing notations, we will say that $u_{\psi(n)} = \sum_{k=1}^{p} \alpha_{k,n} z_k + z_{p+1} + \underset{n\to+\infty}{o}(1)$ is an accumulation expansion of $(u_n)_{n\in\mathbb{N}}$.

▶ **Definition 20.** *Let $u = (u_n)_{n\in\mathbb{N}}$ be a sequence of E. The set $\mathcal{D}_u$ of **principal directions** of u is defined by*

$$\mathcal{D}_u = \left\{ z \in E \;\middle|\; \begin{array}{c} u_{\psi(n)} = \sum_{k=1}^{p} \alpha_{k,n} z_k + z_{p+1} + \underset{n\to+\infty}{o}(1) \text{ is an accumulation expansion} \\ p \geq 1 \qquad \text{and} \qquad z = z_1 \end{array} \right\}.$$

In other words, $\mathcal{D}_u$ is the set of directions that are in the dominant position of some accumulation expansion of u such that $p \geq 1$. It also corresponds to the dominant directions of an unbounded sequence.

For $x \in E \setminus \{0\}$ we denote $\widetilde{x} = \dfrac{x}{\|x\|}$ the associated normalized vector.

▶ **Lemma 21.** *Let $(u_n)_{n\in\mathbb{N}}$ be an unbounded sequence of E. There exist $z \in E$ a unit vector, an increasing function $\varphi : \mathbb{N} \to \mathbb{N}$ and a sequence $(\alpha_n)_{n\in\mathbb{N}}$ such that*

- $\forall n \in \mathbb{N} \qquad \alpha_n > 0$
- $\alpha_n \underset{n\to+\infty}{\longrightarrow} +\infty$
- $\alpha_n \underset{n\to+\infty}{\sim} \left\| u_{\varphi(n)} \right\|$
- $u_{\varphi(n)} = \alpha_n z + \underset{n\to+\infty}{o}(\alpha_n)$
- $\forall n \in \mathbb{N} \qquad u_{\varphi(n)} - \alpha_n z \in z^\perp$ *where $z^\perp$ means the vector subspace of E orthogonal to* $\mathrm{Vect}(\{z\})$

Proof. Since $(u_n)_{n\in\mathbb{N}}$ is unbounded, we can assume that we have an increasing function $\varphi : \mathbb{N} \to \mathbb{N}$ such that for all $n \in \mathbb{N}$, $u_{\varphi(n)} \neq 0$ and $\left\| u_{\varphi(n)} \right\| \underset{n\to+\infty}{\longrightarrow} +\infty$. Therefore the sequence $\left(\widetilde{u_{\varphi(n)}} \right)_{n\in\mathbb{N}}$ is well defined. Moreover, as it is bounded by definition, up to refining φ, we can assume that it converges to some $z \in E$. Let π be the orthogonal projection onto $\mathbb{R}z$. We define α_n to be the unique real number such that $\pi(u_{\varphi(n)}) = \alpha_n z$. As $\widetilde{u_{\varphi(n)}} \underset{n\to+\infty}{\longrightarrow} z$, we have that $\alpha_n \underset{n\to+\infty}{\sim} \left\| u_{\varphi(n)} \right\|$. Therefore, up to refining φ, we can assume that $\alpha_n \underset{n\to+\infty}{\longrightarrow} +\infty$ and $\alpha_n > 0$. Moreover, we have $u_{\varphi(n)} = \alpha_n z + \underset{n\to+\infty}{o}(\alpha_n)$. Finally, by definition of π, for all $n \in \mathbb{N}$, $u_{\varphi(n)} - \alpha_n z \in z^\perp$. ◀

▶ **Proposition 22.** *Any sequence $u = (u_n)_{n\in\mathbb{N}}$ of E admits accumulation expansions. Moreover, if u is unbounded, then $\mathcal{D}_u$ is not empty.*

Proof. If $(u_n)_{n\in\mathbb{N}}$ is bounded, then it has an accumulation point z_1. Hence, taking $p = 0$, all the points are trivially true except Point (AE9). Taking any ψ given by the definition of accumulation point lead to $u_{\psi(n)} = z_1 + \underset{n\to+\infty}{o}(1)$.

Assume now that $(u_n)_{n\in\mathbb{N}}$ is unbounded. We proceed by induction on $d = \dim E$.

- If $d = 1$, consider z_1 and $(\alpha_{1,n})_{n\in\mathbb{N}}$ and ψ given by Lemma 21. By definition, $\|z_1\| = 1$ and $u_{\psi(n)} - \alpha_{1,n}z_1 \in z_1^\perp = \{0\}$. Taking $p = 1$ and $z_2 = 0$ satisfies all the required properties. Moreover, $z_1 \in \mathcal{D}_u$.
- Assume the proposition holds for any Euclidean space of dimension $d-1$. Consider z_1, $\left(\alpha'_{1,n}\right)_{n\in\mathbb{N}}$ and φ given by Lemma 21. By definition $\|z_1\| = 1$ and $u_{\varphi(n)} - \alpha'_{1,n}z_1 \in z_1^\perp$. Since $z_1 \neq 0$, $\dim z_1^\perp = d-1$. We can thus apply the induction hypothesis on the sequence $\left(u_{\varphi(n)} - \alpha'_{1,n}z_1\right)_{n\in\mathbb{N}}$ in $z_1^\perp$. Let φ' be the function given by the induction hypothesis. Let $\psi = \varphi \circ \varphi'$ and $\alpha_{1,n} = \alpha'_{1,\varphi'(n)}$.
  Every point is immediately satisfied either by the induction hypothesis or the fact that z_1 is orthogonal to any point in $z_1^\perp$, except for Point (AE7): It remains to prove that if $p \geq 2$, then $\alpha_{2,n} = \underset{n\to+\infty}{o}(\alpha_{1,n})$. By induction hypothesis we know that

  $$\alpha_{2,n} \underset{n\to+\infty}{\sim} \left\|u_{\psi(n)} - \alpha_{1,n}z_1\right\|.$$

  Moreover, by Lemma 21

  $$\left\|u_{\varphi(n)} - \alpha'_{1,n}z_1\right\| = \underset{n\to+\infty}{o}\left(\alpha'_{1,n}\right).$$

  Since $(\alpha_{1,n})_{n\in\mathbb{N}}$ is a subsequence of $\left(\alpha'_{1,n}\right)_{n\in\mathbb{N}}$, we have

  $$\alpha_{2,n} \underset{n\to+\infty}{\sim} \left\|u_{\psi(n)} - \alpha_{1,n}z_1\right\| = \underset{n\to+\infty}{o}(\alpha_{1,n})$$

  as required. Moreover, $z_1 \in \mathcal{D}_u$. ◀

We now state a relation between the directions within the accumulation expansion and the set $\mathrm{rec}(K)$.

▶ **Proposition 23.** *Let E be an Euclidean space. Let $K \subseteq E$ be MW-convex. Let $u = (u_n)_{n\in\mathbb{N}}$ be an unbounded sequence in K. Let $u_{\varphi(n)} = \sum_{k=1}^{p} \alpha_{k,n}z_k + z_{p+1} + \underset{n\to+\infty}{o}(1)$ be an accumulation expansion of $(u_n)_{n\in\mathbb{N}}$. Then, there are some positive real numbers $(\beta_{k,\ell})_{1\leq\ell<k\leq p+1}$ such that*

$$\forall k \in [\![1\ ;\ p]\!] \qquad z_k + \sum_{\ell=1}^{k-1} \beta_{k,\ell}z_\ell \in \mathrm{rec}(K)$$

and $$z_{p+1} + \sum_{\ell=1}^{p} \beta_{p+1,\ell}z_\ell \in K.$$

Proof. For $k \in [\![1\ ;\ p]\!]$, we consider $\pi_k : E \to E$ the orthogonal projection onto the vector space $\mathrm{Vect}((z_1, \ldots, z_{k-1})^\perp)$. Let us first show that $z_k \in \pi_k(\mathrm{rec}(K))$. Let $\lambda \in \mathbb{R}_+$ and define

$$\lambda_{k,n} = \frac{\lambda}{\alpha_{k,n}}.$$

Note that for large enough n, $\lambda_{k,n} \in [\,0\,;1\,]$. Without loss of generality, we assume $\lambda_{k,n} \in [\,0\,;1\,]$. Then, by convexity,

$$\lambda_{k,n} u_{\varphi(n)} + (1-\lambda_{k,n}) u_0 \in K.$$

Moreover,

$$\begin{aligned}\pi_k\left(\lambda_{k,n} u_{\varphi(n)} + (1-\lambda_{k,n})u_0\right) &= \lambda z_k + \sum_{\ell=k+1}^{p} \lambda_{k,n}\alpha_{\ell,n} z_\ell \\ &\quad + \lambda_{k,n} z_{p+1} + (1-\lambda_{k,n})\pi_k(u_0) + \underset{n\to+\infty}{\mathrm{o}}(\lambda_{k,n}) \\ &\underset{n\to+\infty}{\longrightarrow} \lambda z_k + \pi_k(u_0).\end{aligned}$$

Also, thanks to Corollary 18, we have $\overline{\pi_k(K)} = \pi_k(K)$. Using now Proposition 9, we then conclude that $z_k \in \mathrm{rec}(\pi_k(K))$. Finally, using Corollary 17,

$$z_k \in \pi_k(\mathrm{rec}(K)).$$

We now prove the proposition by induction on k. For $k = 1$, our preliminary result gives in particular that $z_1 \in \mathrm{rec}(K)$.

Assume now that $(\beta_{q,\ell})_{1\le\ell<q<k}$ have been defined for some $k \in [\![\,1\;;\;p\,]\!]$. Since $z_k \in \pi_k(\mathrm{rec}(K))$ as proven earlier, there are some real numbers $(\gamma_{k,\ell})_{\ell\in[\![\,1\;;\;k-1\,]\!]}$ such that

$$z_k + \sum_{\ell=1}^{k-1} \gamma_{k,\ell} z_\ell \in \mathrm{rec}(K).$$

If all the $\gamma_{k,\ell}$ are positive then fixing $\beta_{k,\ell} = \gamma_{k,\ell}$ satisfies the proposition. Let $\ell \in [\![\,1\;;\;k-1\,]\!]$ maximum such that $\gamma_{k,\ell} \le 0$. Then, as by hypothesis we have that $z_\ell + \sum_{j=1}^{\ell-1} \beta_{\ell,j} z_j \in \mathrm{rec}(K)$, we can deduce that

$$z_k + \sum_{j=1}^{k-1} \gamma_{k,j} z_j + (1+|\gamma_{k,\ell}|)\left(z_\ell + \sum_{j=1}^{\ell-1} \beta_{\ell,j} z_j\right) \in \mathrm{rec}(K).$$

Considering $\gamma'_{k,j} = \begin{cases} \gamma_{k,j} & j > \ell \\ 1 & j = \ell, \\ \gamma_{k,j} + (1+|\gamma_{k,\ell}|)\beta_{\ell,j} & j < \ell \end{cases}$

we end up with $z_k + \sum_{\ell=1}^{k-1} \gamma'_{k,\ell} z_\ell \in \mathrm{rec}(K)$ with one less non-positive coefficient. Repeating this procedure until every coefficient is positive lead to a sum of the desired shape, thus establishing the induction hypothesis holds on k and therefore concluding the induction.

Let $\pi_{p+1} : E \to E$ the orthogonal projection on $\mathrm{Vect}((z_1,\dots,z_p)^\perp)$. We have

$$\pi_{p+1}\left(u_{\varphi(n)}\right) \underset{n\to+\infty}{\longrightarrow} z_{p+1}.$$

By Corollary 18,

$$z_{p+1} \in \overline{\pi_{p+1}(K)} = \pi_{p+1}(K).$$

Thus, there are some real numbers $(\gamma_{p+1,\ell})_{\ell\in[\![\,1\;;\;k\,]\!]}$ such that

$$z_{p+1} + \sum_{\ell=1}^{p} \gamma_{p+1,\ell} z_\ell \in K.$$

Doing the same work as above, we can add some elements of $\mathrm{rec}(K)$ so that we end up with some positive $(\beta_{p+1,\ell})_{\ell\in[\![1\ ;\ k]\!]}$ such that

$$z_{p+1}+\sum_{\ell=1}^{p}\gamma_{p+1,\ell}z_\ell\in K.$$ ◀

The two following corollaries specialise this result for some form of sequences.

▶ **Corollary 24.** *Let E an Euclidean space. Let $\pi : E \to E$ be a linear projection. Let $K \subseteq E$ be MW-convex. Let $u = (u_n)_{n\in\mathbb{N}}$ be an unbounded sequence in K and $x \in \mathcal{D}_{\pi(u)}$. Let*

$$f(\mathrm{id}-\pi)(u_{\varphi(n)})\left\|\pi(u_{\varphi(n)})\right\| = \sum_{k=1}^{p}\alpha_{k,n}z_k + z_{p+1} + \underset{n\to+\infty}{o}(1)$$

be an accumulation expansion of $\left(\dfrac{(\mathrm{id}-\pi)(u_n)}{\|\pi(u_n)\|}\right)_{n\in\mathbb{N}}$ *such that*

$$\widetilde{\pi(u_{\varphi(n)})} \underset{n\to+\infty}{\longrightarrow} x.$$

Then, there are some positive real numbers $(\beta_{k,\ell})_{1\le\ell<k\le p+1}$ such that

$$\forall k\in[\![1\ ;\ p+1]\!] \qquad z_k+\sum_{\ell=1}^{k-1}\beta_{k,\ell}z_\ell\in\mathrm{rec}(K) \qquad \text{and} \qquad x+z_{p+1}+\sum_{\ell=1}^{p}\beta_{p+1,\ell}z_\ell\in\mathrm{rec}(K)$$

Proof. We have

$$(\mathrm{id}-\pi)\left(u_{\varphi(n)}\right) = \sum_{k=1}^{p}\left\|\pi(u_{\varphi(n)})\right\|\alpha_{k,n}z_k + \left\|\pi(u_{\varphi(n)})\right\|z_{p+1} + \underset{n\to+\infty}{o}\left(\left\|\pi(u_{\varphi(n)})\right\|\right)$$

Also, provided $\widetilde{u_{\varphi(n)}} \underset{n\to+\infty}{\longrightarrow} x$, we have

$$\pi\left(u_{\varphi(n)}\right) = \left\|\pi(u_{\varphi(n)})\right\|x + \underset{n\to+\infty}{o}\left(\left\|\pi(u_{\varphi(n)})\right\|\right).$$

Therefore

$$u_{\varphi(n)} = \sum_{k=1}^{p}\left\|\pi(u_{\varphi(n)})\right\|\alpha_{k,n}z_k + \left\|\pi(u_{\varphi(n)})\right\|(x+z_{p+1}) + \underset{n\to+\infty}{o}\left(\left\|\pi(u_{\varphi(n)})\right\|\right).$$

The result is obtained by applying Proposition 23 to this accumulation expansion of $(u_n)_{n\in\mathbb{N}}$. Note that in this case we in fact have a truncated accumulation expansion so the case $p+1$ is not the last element of an actual accumulation expansion. That is why we get $\mathrm{rec}(K)$ instead of K even for $p+1$. ◀

▶ **Corollary 25.** *Let E an Euclidean space. Let $K \subseteq E^2$ be MW-convex. Let $\pi : E \to E$ be a linear projection. Let $u = (u_n)_{n\in\mathbb{N}}$ be an unbounded sequence in E such that*

$$\forall n\in\mathbb{N} \qquad (u_n,u_{n+1})\in K$$

and $x \in \mathcal{D}_u$. Let

$$\frac{u_{\varphi(n)+1}}{\left\|u_{\varphi(n)}\right\|} = \sum_{k=1}^{p}\alpha_{k,n}z_k + z_{p+1} + \underset{n\to+\infty}{o}(1)$$

be an accumulation expansion of $\left(\frac{u_{n+1}}{\|u_n\|}\right)_{n\in\mathbb{N}}$ *such that*

$\widetilde{u_{\varphi(n)}} \underset{n\to+\infty}{\longrightarrow} x$.

Then, there are some positive real numbers $(\beta_{k,\ell})_{1\leq\ell<k\leq p+1}$ *such that*

$$\forall k \in [\![1 \ ; \ p]\!] \qquad \left(0, z_k + \sum_{\ell=1}^{k-1} \beta_{k,\ell} z_\ell\right) \in \mathrm{rec}(K) \quad \textit{and} \quad \left(x, z_{p+1} + \sum_{\ell=1}^{p} \beta_{p+1,\ell} z_\ell\right) \in \mathrm{rec}(K)$$

and such that for sufficiently large n,

$$\left\langle \pi\left(z_{p+1} + \sum_{\ell=1}^{p} \beta_{p+1,\ell} z_\ell\right), \pi\left(z_{p+1} + \sum_{k=1}^{p} \alpha_{k,n} z_k\right)\right\rangle \geq 0.$$

Moreover, there is some $i \in [\![1 \ ; \ p+1]\!]$ *such that* $\pi(z_i) \notin \mathrm{Ker}(\pi)$*, this inequality can be taken to be strict.*

Proof. We first apply Corollary 24 to the sequence $((u_n, u_{n+1}))_{n\in\mathbb{N}}$ and the projection on the first component to get some positive real numbers $(\beta_{k,\ell})_{1\leq\ell<k\leq p+1}$ such that

$$\forall k \in [\![1 \ ; \ p]\!] \quad \left(0, z_k + \sum_{\ell=1}^{k-1} \beta_{k,\ell} z_\ell\right) \in \mathrm{rec}(K) \quad \text{and} \quad \left(x, z_{p+1} + \sum_{\ell=1}^{p} \beta_{p+1,\ell} z_\ell\right) \in \mathrm{rec}(K).$$

Let $k_0 \in [\![1 \ ; \ p+1]\!]$ minimum such that $z_k \notin \mathrm{Ker}\,\pi$.

- If there is no such k_0, then
$$\left\langle \pi\left(z_{p+1} + \sum_{\ell=1}^{p} \beta_{p+1,\ell} z_\ell\right), \pi\left(z_{p+1} + \sum_{k=1}^{p} \alpha_{k,n} z_k\right)\right\rangle = 0$$
and the proof is complete.
- If $k_0 = p+1$, then
$$\left\langle \pi\left(z_{p+1} + \sum_{\ell=1}^{p} \beta_{p+1,\ell} z_\ell\right), \pi\left(z_{p+1} + \sum_{k=1}^{p} \alpha_{k,n} z_k\right)\right\rangle = \langle \pi(z_{p+1}), \pi(z_{p+1})\rangle > 0.$$
- Otherwise, $k_0 \in [\![1 \ ; \ p]\!]$ and $\|\pi(z_{k_0})\| \neq 0$. Let
$$S_n(\lambda) = \left\langle \pi\left(z_{p+1} + \sum_{\ell=1}^{p} \beta_{p+1,\ell} z_\ell\right) + \lambda\pi\left(z_{k_0} + \sum_{\ell=1}^{k_0-1} \beta_{k_0,\ell} z_\ell\right), \pi\left(\sum_{k=1}^{p} \alpha_{k,n} z_k + z_{p+1}\right)\right\rangle.$$
We have
$$\begin{aligned}
S_n(\lambda) &= \left\langle \pi(z_{p+1}) + \sum_{\ell=k_0}^{p} \beta_{p+1,\ell}\pi(z_\ell) + \lambda\pi\left(z_{k_0}\right), \sum_{k=k_0}^{p} \alpha_{k,n}\pi(z_k) + \pi(z_{p+1})\right\rangle \\
&= \left\langle \pi(z_{p+1}) + \sum_{\ell=k_0}^{p} \beta_{p+1,\ell}\pi(z_\ell) + \lambda\pi\left(z_{k_0}\right), \pi(z_{p+1})\right\rangle \\
&\quad + \sum_{k=k_0}^{p} \alpha_{k,n} \left\langle \pi(z_{p+1}) + \sum_{\ell=k_0}^{p} \beta_{p+1,\ell}\pi(z_\ell) + \lambda\pi\left(z_{k_0}\right), \pi(z_k)\right\rangle \\
&= \alpha_{k_0,n} \left\langle \pi(z_{p+1}) + \sum_{\ell=k_0}^{p} \beta_{p+1,\ell}\pi(z_\ell) + \lambda\pi\left(z_{k_0}\right), \pi(z_{k_0})\right\rangle + \underset{n\to+\infty}{\mathrm{o}}(\alpha_{k_0,n}) \\
&= \alpha_{k_0,n} \left(\lambda\|\pi(z_{k_0})\|^2 + \left\langle \pi(z_{p+1}) + \sum_{\ell=k_0}^{p} \beta_{p+1,\ell}\pi(z_\ell), \pi(z_{k_0})\right\rangle\right) + \underset{n\to+\infty}{\mathrm{o}}(\alpha_{k_0,n}).
\end{aligned}$$

Therefore, taking any $\lambda > 0$ such that

$$\lambda > -\frac{\left\langle \pi(z_{p+1}) + \sum_{\ell=k_0}^{p} \beta_{p+1,\ell}\pi(z_\ell), \pi(z_{k_0}) \right\rangle}{\|\pi(z_{k_0})\|^2}$$

we get $S_n(\lambda) \underset{n\to+\infty}{\longrightarrow} +\infty$. Thus, for sufficiently large n,

$$\left\langle \pi\left(z_{p+1} + \sum_{\ell=1}^{p} \beta_{p+1,\ell} z_\ell\right) + \lambda\pi\left(z_{k_0} + \sum_{\ell=1}^{k_0-1} \beta_{k_0,\ell} z_\ell\right), \pi\left(\sum_{k=1}^{p} \alpha_{k,n} z_k + z_{p+1}\right) \right\rangle > 0.$$

Also,

$$\left(x, z_{p+1} + \sum_{\ell=1}^{p} \beta_{p+1,\ell} z_\ell + \lambda z_{k_0} + \lambda \sum_{\ell=1}^{k_0-1} \beta_{k_0,\ell} z_\ell\right) = \underbrace{\left(x, z_{p+1} + \sum_{\ell=1}^{p} \beta_{p+1,\ell} z_\ell\right)}_{\in \mathrm{rec}(K)} + \underbrace{\lambda}_{\geq 0} \underbrace{\left(0, z_{k_0} + \sum_{\ell=1}^{k_0-1} \beta_{k_0,\ell} z_\ell\right)}_{\in \mathrm{rec}(K)}$$

$$\left(x, z_{p+1} + \sum_{\ell=1}^{p} \beta_{p+1,\ell} z_\ell + \lambda z_{k_0} + \lambda \sum_{\ell=1}^{k_0-1} \beta_{k_0,\ell} z_\ell\right) \in \mathrm{rec}(K).$$

Thus, considering

$$\beta'_{p+1,\ell} = \begin{cases} \beta_{p+1,\ell} & \ell > k_0 \\ \beta_{p+1,\ell} + \lambda & \ell = k_0 \\ \beta_{p+1,\ell} + \lambda\beta_{k_0,\ell} & \ell < k_0 \end{cases}$$

instead of the $\beta_{k+1,\ell}$s, we get the desired result. ◀

3 Deciding the Constraint Loop Problem

The goal of this section is to establish Theorem 2. This will be done by showing equivalence between the existence of a witness of the form given by Definition 1 and the existence of an infinite run of a constraint loop. The easy direction in this argument – constructing an infinite execution from a witness – is the purpose of Proposition 27 in Subsection 3.2. In fact, there is an even easier case, namely certifying the existence of bounded infinite run, is dealt with in Section 3.1. It states that a bounded infinite run exists if an only if there is a fixed point. This proof holds in any dimension and relies on a simpler certificate. We will also reuse this result in the specific cases of dimension 1 and 2.

The main objective in this section is to construct a witness from an infinite execution. We provide the proof of sufficient condition in Subsection 3.2. This will help motivate the definition of witness. Subsection 3.3 deals with the simple 1-dimensional case, and Subsection 3.4 handles the dimension-2 case, which is more challenging. Because of the difficulty of this proof we only provide high level explanation here. For a complete proof, we refer to the full-version of this article [6].

3.1 Deciding the Existence of a Bounded Sequence

▶ **Proposition 26.** *Let E be a vector space of dimension $d \in \mathbb{N}$. Let $K \subseteq E^2$ be closed convex. Denoting $\Delta_E = \{(x,x) \mid x \in E\} \subseteq E^2$, we have that $K \cap \Delta_E \neq \varnothing$ if and only if there is a bounded sequence $u = (u_n)_{n\in\mathbb{N}}$ of E such that for all $n \in \mathbb{N}$, $(u_n, u_{n+1}) \in K$.*

Proof.
(⇒) Let $(x,x) \in K \cap \Delta_E$. The sequence constantly equal to x satisfy the proposition.
(⇐) Assume now that there exists a bounded sequence $(u_n)_{n\in\mathbb{N}}$ such that for all $n \in \mathbb{N}$, $(u_n, u_{n+1}) \in K$. Let $n \in \mathbb{N}^*$ and define $x_n = \frac{1}{n}\sum_{p=0}^{n}(u_p, u_p)$ and $y_n = \frac{1}{n}\sum_{p=0}^{n-1}(u_p, u_{p+1})$. We have

$$\|x_n - y_n\| = \frac{1}{n}\|(u_n, u_0)\|.$$

Since the sequence $(u_n)_{n\in\mathbb{N}}$ is bounded, there is a positive real number M such that

$$\forall n \in \mathbb{N}^* \qquad \|x_n - y_n\| \leq \frac{M}{n}.$$

In particular, both sequences $(x_n)_{n\in\mathbb{N}^*}$ and $(y_n)_{n\in\mathbb{N}^*}$ must have the same accumulation points. As these sequences are bounded (and since they are in a vector space of finite dimension), such a point exists. Let us denote it x. Notice that since K is closed and convex, for all positive integer n, $y_n \in K$ and thus $x \in K$. Moreover, by definition, for all positive integer n, $x_n \in \Delta_E$. This set is again closed, thus $x \in \Delta_E$. This proves that

$$x \in K \cap \Delta_E \neq \varnothing.$$ ◀

3.2 A Sufficient Condition for the Existence of a Sequence

▶ **Proposition 27.** *Let E be an Euclidean space of dimension d. Let $K \subseteq E^2$ be MW-convex. If there exists a witness $\mathcal{W}(K)$, then, there is a sequence $(u_n)_{n\in\mathbb{N}} \in E^{\mathbb{N}}$ such that*

$$\forall n \in \mathbb{N} \qquad (u_n, u_{n+1}) \in K.$$

Proof. Assume we have a witness $\mathcal{W}(K)$. We then take M, v, w, C as given by the witness and define the following sequence:

$$u_0 = v \qquad \text{and} \qquad u_1 = w$$
$$\forall n \in \mathbb{N} \qquad u_{n+2} - u_{n+1} = M\,(u_{n+1} - u_n)$$

Remark first that for all $n \in \mathbb{N}$, $u_{n+1} - u_n \in C$. This can be proven by induction, noting that the initialisation is given by Point ($\exists u4$) and the induction step comes from Point ($\exists u1$).

We now prove by induction that $\forall n \in \mathbb{N} \qquad (u_n, u_{n+1}) \in K$.

- By Point ($\exists u3$), $(u_0, u_1) \in K$.
- Assume that for some $n \in \mathbb{N}$, $(u_n, u_{n+1}) \in K$. As $u_{n+1} - u_n \in C$ as shown before, by Point ($\exists u2$)

$$(u_{n+1} - u_n, u_{n+2} - u_{n+1}) \in \mathrm{rec}(K).$$

Thus $(u_{n+1}, u_{n+2}) = (u_n, u_{n+1}) + (u_{n+1} - u_n, u_{n+2} - u_{n+1}) \in K + \mathrm{rec}(K) = K$.

By the induction principle we conclude that for all $n \in \mathbb{N}$, $(u_n, u_{n+1}) \in K$. ◀

3.3 Necessary Condition for the Existence of a 1-Dimensional Sequence

We establish the main result in the one dimensional case. Note that we prove a slightly stronger certificate here, which is not necessary in itself, but which we need for the 2 dimensional case.

▶ **Proposition 28.** *Let E be an Euclidean space of dimension 1. Let $K \subseteq E^2$ be MW-convex. Let a sequence $(u_n)_{n\in\mathbb{N}} \in E^{\mathbb{N}}$ such that $(u_n, u_{n+1}) \in K$ for all $n \in \mathbb{N}$. Let $\gamma \in \operatorname{cone}(\mathcal{D}_u)$ such that $(0,\gamma) \in \operatorname{rec}(K)$ (note that at least $\gamma = 0$ works). Then, there are $a \in \mathbb{R}^*$, a closed convex cone $C \subseteq E$ and $x, y \in E$ such that*

(i) $aC \subseteq C$
(ii) $\forall x \in C \qquad (x, ax) \in \operatorname{rec}(K)$
(iii) $(x, y) \in K$
(iv) $y - x \in C$
(v) $\gamma \in C$

Proof. Without loss of generality, as E is an Euclidean space of dimension 1, we assume $E = \mathbb{R}$. If $(u_n)_{n\in\mathbb{N}}$ is bounded, then, by Proposition 26 there exists $z \in \mathbb{R}$ such that $(z, z) \in K$. Then $\gamma = 0$ and we can select $y = x = z$, $C = \{0\}$ and $a \in \mathbb{R}^*$ arbitrary (e.g. 1) to produce the requested witness.

We now assume that $(u_n)_{n\in\mathbb{N}}$ is unbounded. By Proposition 22, it admits accumulation expansions and $\mathcal{D}_u \neq \varnothing$. The only two possible accumulation directions are 1 and -1. We consider three cases:

- If $\mathcal{D}_u = \{-1, 1\}$. Take φ_1 and φ_{-1} such that $\widetilde{u_{\varphi_1(n)}} \underset{n\to+\infty}{\longrightarrow} 1$ and $\widetilde{u_{\varphi_{-1}(n)}} \underset{n\to+\infty}{\longrightarrow} -1$. Up to extracting a subsequence, we have the accumulation expansions

$$\frac{u_{\varphi_1(n)+1}}{\left\|u_{\varphi_1(n)}\right\|} = \sum_{k=1}^{p} \alpha_{k,n} z_k + z_{p+1} + \underset{n\to+\infty}{\mathrm{o}}(1)$$

and $$\frac{u_{\varphi_{-1}(n)+1}}{\left\|u_{\varphi_{-1}(n)}\right\|} = \sum_{k=1}^{p'} \alpha'_{k,n} z'_k + z'_{p'+1} + \underset{n\to+\infty}{\mathrm{o}}(1)\,.$$

  Then, by Corollary 25, there are $\alpha, \beta \in \mathbb{R}$ such that

$$(1, \alpha) \in \operatorname{rec}(K) \qquad \text{and} \qquad (-1, \beta) \in \operatorname{rec}(K).$$

  Let $\delta = \begin{cases} \gamma & \text{if } \gamma \neq 0 \\ \alpha + \beta & \text{if } \gamma = 0. \end{cases}$

  Therefore, either $(0, \delta) = (0, \gamma) \in \operatorname{rec}(K)$, or $(0, \delta) = (1, \alpha) + (-1, \beta)$ and $(0, \delta) \in \operatorname{rec}(K)$ by conic combinations.

  - If $\delta = 0$ then $\gamma = 0$ and $\alpha = -\beta$.
    - If $\alpha = 0$, then $\beta = 0$, $(u_1, u_1) = \underbrace{(u_0, u_1)}_{\in K} + \underbrace{(u_1 - u_0, 0)}_{\in \operatorname{rec}(K)} \in K$.

      We then choose for instance $a \in \mathbb{R}^*$, $C = \{0\}$ and $x = y = u_1$.
    - If $\alpha \neq 0$, then we just have to take $a = \alpha$, $C = \mathbb{R}$, $x = u_0$, $y = u_1$.

    Note that in both these cases we trivially have $\gamma \in C$.
  - If $\delta > 0$ then, for large enough n, $n\delta + \alpha > 0$. Moreover, as $\operatorname{rec}(K)$ is a cone,

$$(1, n\delta + \alpha) = \underbrace{n(0, \delta)}_{\in \operatorname{rec}(K)} + \underbrace{(1, \alpha)}_{\in \operatorname{rec}(K)} \in \operatorname{rec}(K).$$

We then take $a = n\delta + \alpha > 0$, $C = \mathbb{R}_+$, $x = u_k$, $y = u_{k+1}$, for some k such that $u_{k+1} - u_k + > 0$. This exists since $1 \in \mathcal{D}_u$ and hence $(u_n)_{n\in\mathbb{N}}$ is not bounded from above. Note also that since $\delta > 0$ then $\gamma \geq 0$. Thus $\gamma \in C$.

- If $\delta < 0$ then, for large enough n, $n\delta + \beta < 0$. Moreover, as $\mathrm{rec}(K)$ is a cone,

$$(-1, n\delta + \beta) = \underbrace{n(0,\delta)}_{\in \mathrm{rec}(K)} + \underbrace{(-1,\beta)}_{\in \mathrm{rec}(K)} \in \mathrm{rec}(K).$$

We then take $a = -n\delta - \beta > 0$, $C = \mathbb{R}_-$, $x = u_k$, $y = u_{k+1}$ for some k such that $u_{k+1} - u_k < 0$. This exists since $-1 \in \mathcal{D}_u$ and hence $(u_n)_{n\in\mathbb{N}}$ is not bounded from below. Note also that since $\delta < 0$ then $\gamma \leq 0$. Thus $\gamma \in C$.

- If $\mathcal{D}_u = \{1\}$, then, similarly to the first case, using Corollary 25, there is some $\alpha \in \mathbb{R}_+$ such that $(1, \alpha) \in \mathrm{rec}(K)$. Note also that $\gamma \geq 0$ and that $(1, \alpha + \gamma) \in \mathrm{rec}(K)$. Let k such that $u_{k+1} - u_k > 0$. This exists since $1 \in \mathcal{D}_u$ and hence $(u_n)_{n\in\mathbb{N}}$ is not bounded from above.
  - If $\alpha + \gamma = 0$, then, $\alpha = \gamma = 0$ and

$$(u_{k+1}, u_{k+1}) = \underbrace{(u_k, u_{k+1})}_{\in K} + \underbrace{(u_{k+1} - u_k, 0)}_{\in \mathrm{rec}(K)} \in K.$$

We then choose for instance $a \in \mathbb{R}^*$, $C = \{0\}$ and $x = y = u_{k+1}$.

  - If $\alpha + \gamma > 0$, then we just have to take $a = \alpha + \gamma$, $C = \mathbb{R}_+$, $x = u_k$ and $y = u_{k+1}$.

  Note that in both cases, $\gamma \in \mathbb{R}_+ = C$.
- The case $\mathcal{D}_u = \{-1\}$ can be made similarly to the previous point. ◀

We are now ready to prove the special case of Theorem 2 in which E has dimension 1 (see Section 1). Without loss of generality we just consider $E = \mathbb{R}$. The necessary condition is given by the application of Proposition 28 with $\gamma = 0$. The sufficient condition is given by Proposition 27.

3.4 Necessary Condition for the Existence of a 2-Dimensional Sequence

We now move to 2-dimensional Euclidean spaces and prove that the existence of a witness as given by Definition 1 is implied by the existence of an infinite sequence. This, combined with Proposition 27 will imply Theorem 2.

For the entire section, we thus fix E to be an Euclidean space of dimension 2, $K \subseteq E^2$ to be MW-convex and thus satisfying $K = K' + \mathrm{rec}(K)$ where K' is a compact convex set. We assume that there exists a sequence $(u_n)_{n\in\mathbb{N}} \in E^{\mathbb{N}}$ such that for all $n \in \mathbb{N}$, $(u_n, u_{n+1}) \in K$.

We start by two technical lemmas to lighten the proof of the proposition.

▶ **Lemma 29.** *Assume that $\mathcal{D}_u$ is not empty and for all $x \in \mathrm{cone}\,\mathcal{D}_u$, if $(0, x) \in \mathrm{rec}(K)$, then $x = 0$. Denoting $\mathcal{C}_u = \mathrm{cone}\,\mathcal{D}_u$, we have that for all $x \in \mathcal{C}_u$, there is $s(x) \in \mathcal{C}_u$ such that $(x, s(x)) \in \mathrm{rec}(K)$.*

Proof. Let $x \in \mathcal{C}_u$. By definition, we can consider $x_1, \ldots, x_m \in \mathcal{D}_u$ and $\lambda_1, \ldots, \lambda_m \in \mathbb{R}_+$ such that $x = \sum_{i=1}^{m} \lambda_i x_i$. By definition of $\mathcal{D}_u$, for $i \in [\![1 \ ; \ m]\!]$ there is an increasing function $\varphi_i : \mathbb{N} \to \mathbb{N}$ such that $\widetilde{u_{\varphi_i(n)}} \underset{n\to+\infty}{\longrightarrow} x_i$. Using Proposition 22, the sequence $\left(\left(u_{\varphi_i(n)}, u_{\varphi_i(n)+1}\right)\right)_{n\in\mathbb{N}}$ admits an accumulation expansion

$$\left(u_{\varphi_i\circ\psi_i(n)}, u_{\varphi_i\circ\psi_i(n)+1}\right) = \sum_{k=1}^{p_i} \alpha_{i,k,n}(z_{i,k,1}, z_{i,k,2}) + (z_{i,p+1,1}, z_{i,p+1,2}) + \underset{n\to+\infty}{\mathrm{o}}(1).$$

In particular, for $k \in [\![1 \,;\, p_i]\!]$ minimum such that $z_{i,k,1} \neq 0$, we have $z_{i,k,1} \in \mathbb{R}_+^* x_i$. Since the first component is not bounded, such a k exists. Let $\mu_i > 0$ such that $z_{i,k,1} = \mu_i x_i$. Now, applying Proposition 23, $(z_{i,1,1}, z_{i,1,2}) \in \mathrm{rec}(K)$ and $\|(z_{i,1,1}, z_{i,1,2})\| = 1$. Therefore, if $k > 1$, then $z_{i,1,1} = 0$ and $\|z_{i,1,2}\| = 1$. Hence $z_{i,1,2} \in \mathcal{D}_u$. This contradicts the hypothesis that for all $x \in \mathcal{C}_u$, if $(0, x) \in \mathrm{rec}(K)$, then $x = 0$. Thus, $k = 1$. Considering $s(x_i) = \dfrac{1}{\mu_i} z_{i,1,2}$ satisfies the claim for x_i. Thus, defining $s(x) = \sum_{i=1}^{m} \lambda_i s(x_i)$ establishes the lemma. ◀

▶ **Lemma 30.** *Assume that $\mathcal{D}_u$ is not empty, that for all $x \in \mathrm{cone}\, \mathcal{D}_u$, if $(0, x) \in \mathrm{rec}(K)$, then $x = 0$ and for all $x \in E$, $(x, x) \notin K$. Denoting $\mathcal{C}_u = \mathrm{cone}\, \mathcal{D}_u$, for all $x \in \mathcal{D}_u$, there are $\delta(x) \in E$ and $\lambda \in \mathbb{R}_+^*$ such that $(\delta(x), \lambda x + \delta(x)) \in K \cup \mathrm{rec}(K)$.*

Proof. Let $x \in \mathcal{D}_u$ and the accumulation expansion

$$u_{\varphi(n)} = \sum_{k=1}^{p} \alpha_{k,n} z_k + z_{p+1} + \underset{n \to +\infty}{\mathrm{o}}(1)$$

with $p > 0$ and $z_1 = x$. By convexity, we have

$$\forall n \in \mathbb{N} \qquad \frac{1}{\varphi(n)} \sum_{k=0}^{\varphi(n)-1} (u_k, u_{k+1}) \in K.$$

Up to refining φ, we can assume that we also have the accumulation expansion

$$\frac{1}{\varphi(n)} \sum_{k=0}^{\varphi(n)-1} (u_k, u_{k+1}) = \sum_{k=1}^{q} \beta_{k,n} (w_{k,1}, w_{k,2}) + (w_{q+1,1}, w_{q+1,2}) + \underset{n \to +\infty}{\mathrm{o}}(1)\,.$$

Therefore

$$\begin{aligned}
\sum_{k=1}^{q} \beta_{k,n} (w_{k,2} - w_{k,1}) + w_{q+1,2} - w_{q+1,1} &= \frac{1}{\varphi(n)} \sum_{k=0}^{\varphi(n)-1} (u_{k+1} - u_k) + \underset{n \to +\infty}{\mathrm{o}}(1) \\
&= \frac{u_{\varphi(n)} - u_0}{\varphi(n)} + \underset{n \to +\infty}{\mathrm{o}}(1) \\
&= \sum_{k=1}^{p} \frac{\alpha_{k,n}}{\varphi(n)} z_k + \underset{n \to +\infty}{\mathrm{o}}(1)\,.
\end{aligned}$$

If $\left(\dfrac{\alpha_{1,n}}{\varphi(n)} \right)_{n \in \mathbb{N}}$ has an accumulation point, say λ, up to refining φ, we assume that it converges to it. By definition of an accumulation expansion, we then have for all $k \in [\![1 \,;\, q]\!]$, $w_{k,1} = w_{k,2}$ Therefore, $w_{q+1,2} - w_{q+1,1} = \lambda x$.

By Proposition 23, there are some positive real numbers $\gamma_1, \ldots, \gamma_q$ such that

$$\sum_{k=1}^{q} \gamma_k (w_{k,1}, w_{k,2}) + (w_{q+1,1}, w_{q+1,2}) \in K.$$

The difference between the two coordinates of this vector is λx. Since λ is the limit of a positive sequence, $\lambda \geq 0$. Also, provided that there is no $a \in E$ such that $(a, a) \in K$ by hypothesis, we have $\lambda \neq 0$. Therefore, considering $\delta(x) = \sum_{k=1}^{q} \gamma_k w_{k,1} + w_{q+1,1}$ we get $(\delta(x), \lambda x + \delta(x)) \in K$.

Now suppose that $\left(\frac{\alpha_{1,n}}{\varphi(n)}\right)_{n\in\mathbb{N}}$ has no accumulation point. Since it is positive, we have

$$\frac{\alpha_{1,n}}{\varphi(n)} \underset{n\to+\infty}{\longrightarrow} +\infty.$$

Thus, there is $k \in [\![1 \ ; \ q]\!]$ minimum such that $w_{k,1} \neq w_{k,2}$ and for this k, we have

$$\beta_{k,n}(w_{k,2} - w_{k,1}) \underset{n\to+\infty}{\sim} \frac{\alpha_{1,n}}{\varphi(n)} x.$$

Therefore, there is $\lambda > 0$ such that $w_{k,2} - w_{k,1} = \lambda x$. By Proposition 23, there are some positive real numbers $\gamma_1, \ldots, \gamma_{k-1}$ such that

$$\sum_{\ell=1}^{k-1} \gamma_\ell (w_{\ell,1}, w_{\ell,2}) + (w_{k,1}, w_{k,2}) \in \mathrm{rec}(K).$$

The difference between the two coordinates of this vector is λx. Therefore, considering $\delta(x) = \sum_{\ell=1}^{k-1} \gamma_\ell w_{\ell,1} + w_{k,1}$ we have $(\delta(x), \lambda x + \delta(x)) \in \mathrm{rec}(K)$. ◀

▶ **Proposition 31.** *There exists a witness $\mathcal{W}(K)$.*

For the detailed proof we refer to the full version [6]. Here we just give an overview of the proof.

Proof sketch. The proof is divided into several cases under the structure described in Figure 1. Among all these cases, Case 6 is by far the most difficult, followed by Cases 2 and 5, then Case 4 (quite easy) and finally the almost trivial Cases 1 and 3. In this proof we denote $\mathcal{C}_u = \mathrm{cone}(\mathcal{D}_u)$.

- **Case 1:** There is a fixed point (x, x) in K. In this case we just need to take $v = w = x$, M arbitrary and $C = \{0\}$ to get a Witness. This just leads to a constant sequence.
- **Case 2:** No fixed point but there is $x \in \mathcal{C}_u \setminus \{0\}$ such that $(0, x) \in \mathrm{rec}(K)$. In this case we are going to try to make use of Proposition 28. Let $\pi : E \to E$ be the orthogonal projection onto $x^\perp$ and let $\widehat{\pi} : E^2 \to E^2$ be such that

  $$\forall e, f \in E, \widehat{\pi}(e, f) = (\pi(e), \pi(f)) .$$

  Assume that we have found some x' such that $(x, x') \in \mathrm{rec}(K)$. We then can write $x' = \gamma x + y$ for some y orthogonal to x and some $\gamma \in \mathbb{R}$. Then we have $\widehat{\pi}(x, x') = (0, y)$. Also $(0, y) \in \widehat{\pi}(\mathrm{rec}(K)) = \mathrm{rec}(\widehat{\pi}(K))$. Thus if we can build x' such that $y \in \mathrm{cone}(\mathcal{D}_{\pi(u)})$, we would be allowed to apply Proposition 28. This requires some work. The idea is to write $x = \sum_{i=1}^{n} a_i x_i$ with $x_i \in \mathcal{D}_u$ and $a_i \geq 0$ then apply Corollary 25 for all $i \in [\![1 \ ; \ n]\!]$ (see the full version [6] for details). Assume this is done. There are $a \in \mathbb{R}^*$, a closed convex cone $C \subseteq x^\perp$ and $v, w \in x^\perp$ such that
  - $aC \subseteq C$
  - $\forall c \in C \qquad (c, ac) \in \mathrm{rec}(\widehat{\pi}(K))$
  - $(v, w) \in \widehat{\pi}(K)$
  - $w - v \in C$
  - $y \in C$

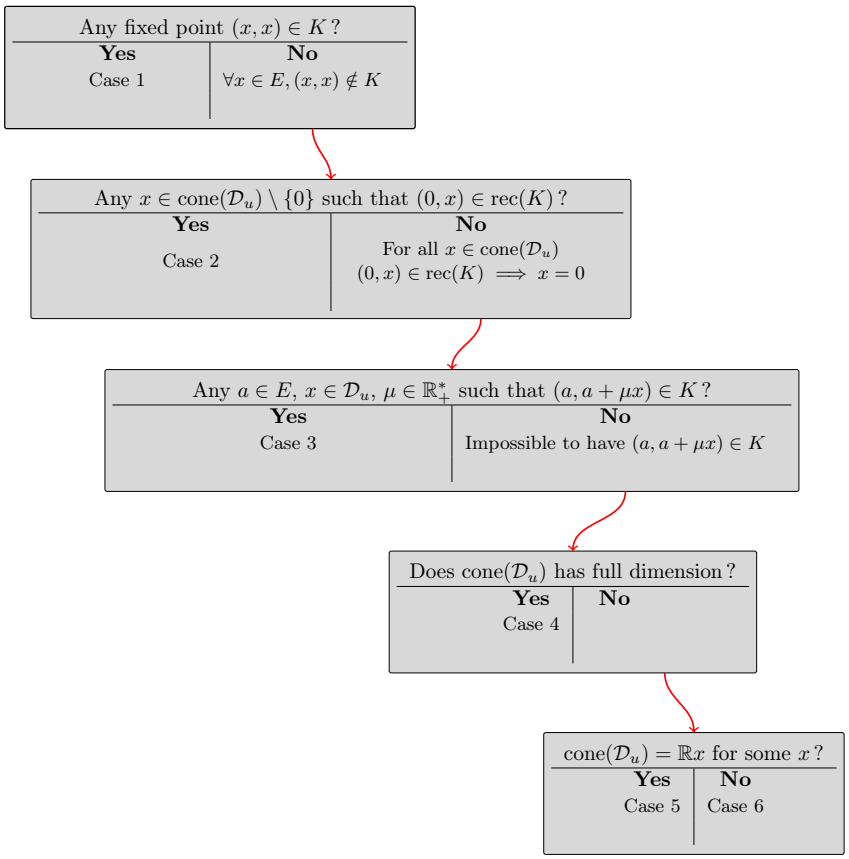


Figure 1 The case disjunction structure.

Again, since $\text{rec}(\widehat{\pi}K) = \widehat{\pi}(\text{rec}(K))$, for all $\xi \in C$, there are $b_\xi, c_\xi \in \mathbb{R}$ such that $(b_\xi x + \xi, c_\xi x + a\xi) \in \text{rec}(K)$. For all $\xi \in C$, we fix b_ξ and c_ξ such that $(|b_\xi|, |c_\xi|)$ is minimal for the lexicographic order and among all these possibilities, such that (b_ξ, c_ξ) is maximal for the lexicographic order. We denote

$$\gamma_0(\xi) = \max(1, \gamma, b_\xi) \qquad \text{and} \qquad \gamma_1(\xi) = \max(|a|, b_{a\xi}, c_\xi)$$

and for $n \geq 1$,

$$\gamma_{2n}(\xi) = \max\left(a^{2n}, a^{2n} b_\xi, a^{2(n-1)} c_{a\xi}\right)$$

$$\gamma_{2n+1}(\xi) = \max(|a|^{2n+1}, a^{2n} b_{a\xi}, a^{2n} c_\xi).$$

For $n \in \mathbb{N}$, let

$$\chi_n(\xi) = \gamma_n(\xi) x + a^n \xi \qquad \text{and} \qquad b'_{n,\xi} = \begin{cases} a^{2n} b_\xi & n \in 2\mathbb{N} \\ a^{2n} b_{a\xi} & n \in 2\mathbb{N} + 1. \end{cases}$$

We some algebraic manipulations and intensively using that $(0, x) \in \text{rec}(K)$ to add missing weight on x in the second component, we get

$$\forall n \in \mathbb{N} \qquad \left(\chi_n(\xi), \chi_{n+1}(\xi) + \left(\gamma_n(\xi) - b'_{n,\xi}\right) \chi_0(y)\right) \in \text{rec}(K).$$

Recalling that $w - v \in C$ we define $C' = \mathbb{R}_+ x + \sum_{n\in\mathbb{N}} (\mathbb{R}_+\chi_n(w-v) + \mathbb{R}_+\chi_n(y))$ This is the cone we want to use. It is finitely generated. We can also see that it cannot contain line. Since all such two-dimensional cones are generated by at most two vectors we can find such generating vectors. M will just be a matrix defined thanks to its behavior on these vectors and C' is defined to get stability. Finally, up to add some component on x again we can get our starting conditions thanks to v and w (See details in the full version [6]).

- **Case 3:** No fixed point or $x \in \mathcal{C}_u \setminus \{0\}$ such that $(0, x) \in \mathrm{rec}(K)$. However there are $a \in E$, $x \in \mathcal{D}_u$ and $\mu \in \mathbb{R}_+^*$ such that $(a, a+\mu x) \in K$. This means that their is a principle direction of u along which it is possible to take a first step. In this case, we select $C = \mathcal{C}_u$. C is a non empty closed convex cone of $\mathbb{R}^2$, thus, there are two vectors $x_1, x_2 \in C \setminus \{0\}$ such that either $C = \mathbb{R}x_1 + \mathbb{R}_+x_2$ or $C = \mathbb{R}_+x_1 + \mathbb{R}_+x_2$ or $C = \mathbb{R}x_1 + \mathbb{R}x_2$. Let $I \subseteq \{1, 2\}$, $I \neq \varnothing$ the largest set such that $(x_i)_{i\in I}$ is a free family. Using the function s defined by Lemma 29, we define M such that $Mx_i = s(x_i)$ for all $i \in I$. Noting that since, for $i \in I$, $-x_i \in C$, $(0, s(x_i) + s(-x_i)) \in \mathrm{rec}(K)$, we have that $s(-x_i) = -s(x_i)$, this choice of M satisfies Points ($\exists u1$) and ($\exists u2$). We now choose $v = a$ and $w = a + \mu x$. By assumption, $(v, w) \in K$. Also, $w - v = \mu x \in \mathcal{C}_u = C$. C, v and w thus satisfy Points ($\exists u3$) and ($\exists u4$).
- **Case 4:** No fixed point, $x \in \mathcal{C}_u \setminus \{0\}$ such that $(0, x) \in \mathrm{rec}(K)$ or $a \in E$, $x \in \mathcal{D}_u$, $\mu \in \mathbb{R}_+^*$ such that $(a, a + \mu x) \in K$. However $\mathcal{D}_u$ spans the entire space E. Given that, take $(a, b) \in K$. Using Lemma 13 there is $\lambda \geq 0$ such that $y := b - a + \lambda x \in \mathcal{C}_u$. Let $v = a + \lambda\delta(x)$ and $w = b + \lambda(x + \delta(x))$ with δ given by Lemma 30. We then have $(u, v) \in K$. Let $C = \overline{\mathrm{cone}\{\, s^k(y) \mid k \in \mathbb{N}\}}$ with s being the function defined in Lemma 29. C is a closed convex cone in a 2-dimensional vector space, therefore there are vectors $\zeta_1, \zeta_2 \in C \setminus \{0\}$ such that

  $$C \in \{\mathbb{R}_+\zeta_1 + \mathbb{R}_+\zeta_2, \mathbb{R}\zeta_1 + \mathbb{R}_+\zeta_2, \mathbb{R}_+\zeta_1 + \mathbb{R}\zeta_2, \mathbb{R}\zeta_1 + \mathbb{R}\zeta_2\}.$$

  Let $(\zeta_{i,n})_{n\in\mathbb{N}}$ be a sequence in $\mathrm{cone}\{\, s^k(y) \mid k \in \mathbb{N}\}$ such that

  $$\zeta_{i,n} \underset{n\to+\infty}{\longrightarrow} \zeta_i.$$

  If $(s(\zeta_{i,n}))_{n\in\mathbb{N}}$ is unbounded then Proposition 23 ensures that there is some $\zeta_i' \in \mathcal{D}_{(s(\zeta_{i,n}))_{n\in\mathbb{N}}}$ such that $(0, \zeta_i') \in \mathrm{rec}(K)$ and $\zeta_i' \in \mathcal{C}_u$. This is impossible by assumption on $\mathcal{C}_u$. Therefore, it is bounded and we have an accumulation point $\zeta_i' \in C$. Since $\mathrm{rec}(K)$ is closed, we also have $(\zeta_i, \zeta_i') \in \mathrm{rec}(K)$. Let $I \subseteq \{1, 2\}$ maximal such that $(\zeta_i)_{i\in I}$ is a free family. Let M be a matrix such that

  $$\forall i \in I \qquad M\zeta_i = \zeta_i'.$$

- **Case 5:** No fixed point, $x \in \mathcal{C}_u \setminus \{0\}$ such that $(0, x) \in \mathrm{rec}(K)$ or $a \in E$, $x \in \mathcal{D}_u$, $\mu \in \mathbb{R}_+^*$ such that $(a, a + \mu x) \in K$ and $\mathcal{C}_u$ is a line $\mathcal{C}_u = \mathbb{R}x$. This case uses the induction hypothesis (Proposition 28) and similar techniques as in Case 2. The main change here is that we use the function s defined by Lemma 29. Here $s(x)$ will have to be collinear with x. In stead of adding multiples of $(0, x)$, we have access to some $(x, \gamma x) \in \mathrm{rec}\, K$ and are allowed negative coefficients which makes the case relatively easy. See details in the full version [6].
- **Case 6:** No fixed point, $x \in \mathcal{C}_u \setminus \{0\}$ such that $(0, x) \in \mathrm{rec}(K)$ or $a \in E$, $x \in \mathcal{D}_u$, $\mu \in \mathbb{R}_+^*$ such that $(a, a + \mu x) \in K$ and $\mathcal{C}_u = \mathbb{R}_+x$ for some x. Let $y \in x^\perp$ such that $\|y\| = 1$. The main goal of this case is to find $a, b \geq 0$ and $c, d \in \mathbb{R}$ such that

  $$(x, ax) \in \mathrm{rec}(K) \qquad \text{and} \qquad (dx + y, cx + by) \in \mathrm{rec}(K) \qquad \text{and} \qquad c \geq db.$$

This can be achieved by a very careful look at the asymptotic behavior of the $(u_n)_{n\in\mathbb{N}}$ and more precisely its components along x and y. Namely, the component along x must blow up significantly faster than the one along y. This is where the difficulty of this case lies. We refer to the full version [6] for the details. This naturally leads to choose C and M such that:

$$C = \mathbb{R}_+x + \mathbb{R}_+(dx+y) \qquad \text{and} \qquad Mx = ax \qquad \text{and} \qquad M(dx+y) = cx + by$$

immediately satisfying $(\exists u1)$ and $(\exists u2)$. With the same technics we can show that there is some $n \in \mathbb{N}$ such that

$$\langle u_{n+1} - u_n, x\rangle \geq d\,\langle u_{n+1} - u_n, y\rangle\,.$$

Then considering $v = u_n$ and $w = u_{n+1}$.

$$\begin{aligned} w - v = u_{n+1} - u_n &= \langle u_{n+1} - u_n, x\rangle\, x + \langle u_{n+1} - u_n, y\rangle\, y \\ &= \left(\langle u_{n+1} - u_n, x\rangle - d\,\langle u_{n+1} - u_n, y\rangle\right) x + \langle u_{n+1} - u_n, y\rangle\,(dx+y) \in C. \end{aligned}$$

Hence, Points $(\exists u3)$ and $(\exists u4)$ are satisfied by C, v, w. ◀

References

1. Amir M. Ben-Amram, Jesús J. Doménech, and Samir Genaim. Multiphase-linear ranking functions and their relation to recurrent sets. In *Static Analysis - 26th International Symposium, SAS 2019, Proceedings*, volume 11822 of *Lecture Notes in Computer Science*, pages 459–480. Springer, 2019.
2. Amir M. Ben-Amram and Samir Genaim. Ranking functions for linear-constraint loops. *Journal of the ACM*, 61(4):1–55, 2014. doi:10.1145/2629488.
3. Amir M. Ben-Amram, Samir Genaim, and Abu Naser Masud. On the termination of integer loops. *ACM Transactions on Programming Languages and Systems*, 34(4):1–24, 2012. doi:10.1145/2400676.2400679.
4. Marius Bozga, Radu Iosif, and Filip Konecný. Deciding conditional termination. *Log. Methods Comput. Sci.*, 10(3), 2014. doi:10.2168/LMCS-10(3:8)2014.
5. Mark Braverman. Termination of integer linear programs. In *Computer Aided Verification 2006*, volume 4144 of *LNCS*, pages 372–385. Springer Berlin Heidelberg, 2006. doi:10.1007/11817963_34.
6. Quentin Guilmant, Engel Lefaucheux, Joël Ouaknine, and James Worrell. The 2-dimensional constraint loop problem is decidable, 2024. arXiv:2405.12992.
7. Mehran Hosseini, Joël Ouaknine, and James Worrell. Termination of linear loops over the integers. In *ICALP 2019*, volume 132 of *LIPIcs*, pages 118:1–118:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern/Saarbruecken, Germany, 2019. doi:10.4230/LIPICS.ICALP.2019.118.
8. Jan Leike and Matthias Heizmann. Geometric nontermination arguments. In *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018*, volume 10806 of *Lecture Notes in Computer Science*, pages 266–283. Springer, 2018.
9. Naomi Lindenstrauss and Yehoshua Sagiv. Automatic termination analysis of logic programs. In Lee Naish, editor, *Logic Programming, Proceedings of the Fourteenth International Conference on Logic Programming, 1997*, pages 63–77. MIT Press, 1997.
10. Ashish Tiwari. Termination of linear programs. In *Computer Aided Verification 2004*, volume 3114 of *LNCS*, pages 70–82. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-27813-9_6.

Flattability of Priority Vector Addition Systems

Roland Guttenberg
Technical University of Munich, Germany

Abstract

Vector addition systems (VAS), also known as Petri nets, are a popular model of concurrent systems. Many problems from many areas reduce to the *reachability problem* for VAS, which consists of deciding whether a target configuration of a VAS is reachable from a given initial configuration. One of the main approaches to solve the problem on practical instances is called *flattening*, intuitively removing nested loops. This technique is known to terminate for semilinear VAS due to [22]. In this paper, we prove that also for VAS with nested zero tests, called Priority VAS, flattening does in fact terminate for all semilinear reachability relations. Furthermore, we prove that Priority VAS admit semilinear inductive invariants. Both of these results are obtained by defining a well-quasi-order on runs of Priority VAS which has good pumping properties.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Priority Vector Addition Systems, Semilinear, Inductive Invariants, Geometry, Flattability, Almost Semilinear, Transformer Relation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.141

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2402.09185` [9]

Acknowledgements I thank my PhD advisor Javier Esparza for reading a first draft and providing feedback. I thank the anonymous reviewers for their helpful feedback.

1 Introduction

Vector addition systems (VAS), also known as Petri nets, are a popular model of concurrent systems. VAS have a very rich theory and have been intensely studied. In particular, the *reachability problem* for VAS, which consists of deciding whether a target configuration of a VAS is reachable from a given initial configuration, has been studied for over 50 years. It was proved decidable in the 1980s [28, 16, 17], but its complexity (Ackermann-complete) could only be determined recently [6, 7, 23].

In [19] and [22], Leroux proved two fundamental results about the reachability sets of VAS. In [19], he showed that every configuration outside the reachability set $\mathbf{R}$ of a VAS is separated from $\mathbf{R}$ by a semilinear inductive invariant (for basic facts on semilinear sets see e.g. [11]). This immediately led to a very simple algorithm for the reachability problem consisting of two semi-algorithms, one enumerating all possible paths to certify reachability, and one enumerating all semilinear sets and checking if they are separating inductive invariants.

In [22], he proved that if the reachability set of a VAS is semilinear, then it is *flattable*. Flattability states the existence of a finite sequence $\rho_1, \ldots, \rho_r$ of transition sequences such that every reachable vector can be reached via a sequence in $\rho_1^* \ldots \rho_r^*$, i.e., by means of a "flat" expression without nested loops. Flattability leads to an algorithm for deciding whether a semilinear set is included in or equal to the reachability set of a given VAS, i.e. whether a VAS has the set of desired behaviours. If it is not included, guess the violating configuration and check it is unreachable, otherwise guess a linear path scheme and verify it.


One major branch of ongoing research in the theory of VAS studies whether results like the above extend to more general systems [30, 3, 4, 31, 1, 25, 27, 13, 18, 8, 26, 2]. In particular, the reachability problem has been proved decidable for Priority VAS, an extension of VAS in which counters can be tested for zero, albeit in restricted manner: there is a total order on the counters such that whenever a counter is tested for 0, all smaller counters are simultaneously tested as well. In a famous but very technical paper, Reinhardt proved that the reachability problem remains decidable for Priority VAS [30]. In [3] and later in his thesis [4], Bonnet presented a more accessible proof which was obtained by extending the result of [19], separability by inductive semilinear sets.

In this paper we extend the result of [22] to arbitrary Priority VAS, and on the way obtain another proof that [19] extends. That is, we show that 1. Priority VAS admit semilinear inductive invariants, and 2. semilinear Priority VAS are flattable. Notice that 2. was not known even for the special case of one testable counter. Furthermore, as remarked in [26], while two-dimensional vector addition systems with a zero test and a reset are effectively semilinear [8], they are not flattable in general. Hence, our second result establishes a theoretical limit of flattability.

These results are obtained via two technical contributions of independent interest.

Regular expressions for Priority VAS. We give a new characterization of the reachability relations of Priority VAS. More precisely, we show that a relation is the reachability relation of a Priority VAS if and only if it can be represented as a regular expression over the reachability relations of standard VAS, with the restriction that the Kleene star operation can only be applied to monotone relations. For example in case of the Priority VASS in Figure 1 as $\mathcal{V}$, we would consider the VASS without the zero test transition as $\mathcal{V}_0$, and if we are interested in the reachability relation starting at q_s, ending at q_s and requiring counter x to start and end at 0, formally $\rightarrow^*_{\mathcal{V},q_s\rightarrow q_s} \cap\{x_{in} = x_{out} = 0\}$, then we would rewrite $\rightarrow^*_{\mathcal{V},q_s\rightarrow q_s} \cap\{x_{in} = x_{out} = 0\} = (\rightarrow^*_{\mathcal{V}_0,q_s\rightarrow q_t} \cap\{x_{in} = x_{out} = 0\})^*$. I.e., instead we consider the inner normal VASS starting at q_s, ending at q_t and fixing x to 0 at start and end, then taking the reflexive transitive closure of this relation. In general zero testing a coordinate will generate an expression of the form $\mathbf{E}^*$, where $\mathbf{E}$ fixes some coordinates to 0 at start and end. One important aspect of this characterization is that all intersections with linear relations (for example here with $x_{in} = x_{out} = 0$) can be pushed purely to the inner VASS level, where they were dealt with in [22]. Hence in our arguments we only have to consider how to deal with $\circ$ and $*$, not with intersections or projections.

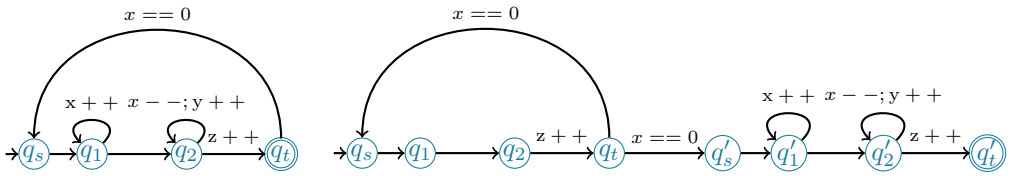


Figure 1 Example of a PVASS and an equivalent (for $x_{in} = 0$) flattened version.

Characterizations of complicated relations using RegEx over simpler relations have already proven useful in other contexts [29, 12].

A well-quasi-order (wqo) on the set of runs of Priority VAS. A wqo on a set is a partial order such that every subset has finitely many minimal elements. There exist wqos on many kinds of objects: vectors, sequences, trees, or graphs. A wqo on the set of runs of a PVAS

provides the following decomposition: Let $\mathrm{src}(\rho)$ denote the source of a run ρ, $\mathrm{tgt}(\rho)$ denote its target and $\mathrm{ends}(\rho) = (\mathrm{src}(\rho), \mathrm{tgt}(\rho))$ its *pair of ends*, i.e. source/target pair. Let Ω_{min} be the finite set of minimal runs. Then the reachability relation $\to^*_{\mathcal{V}}$ can be written as $\to^* = \bigcup_{\rho\in\Omega_{min}} \{\mathrm{ends}(\rho') \mid \rho' \geq \rho\}$, i.e. we observe that every pair of configurations $\mathbf{c}, \mathbf{c}'$ such that $\mathbf{c} \to^* \mathbf{c}'$ is witnessed by some run, and then we split runs into one group for every minimal run. Intuitively, this reduces the problem of proving flattability of a VAS to proving flattability in each of the groups determined by the minimal runs. The proof that semilinear VAS are flattable follows this scheme. More precisely, the proof, given in [22], takes the wqo on the runs of a VAS introduced by Jancar in [15], proves that it satisfies certain pumping properties, properties shown in [15, 20, 21] and new ones, and derives the result. We proceed in the same way, starting from a wqo on the set of runs of a Priority VAS[1].

Let us now cconsider the concrete case of pumping and flattening in the example in Figure 1. Consider the run $\rho : q_s \to q_1 \to q_2 \to q_t$ which does not use any of the self-loops. Intuitively, if we did not have the $x_{in} = x_{out} = 0$ restriction, both the x and y coordinates can be pumped arbitrarily along this run. In order to pump the x coordinate simply add one use of the self-loop on q_1, and to increase the y coordinate add both self-loops once. Observe that despite using a loop which is not non-negative in the second case, and hence could not be arbitrarily often repeated by itself, this loop only decreases a coordinate which was already pumped prior in the run, i.e. the sequence of loops can be pumped.

Using this basic image of pumping, we can now describe the idea of our proof. In the example, define $\mathbf{E} := \to^*_{\mathcal{V}_0, q_s\to q_t} \cap \{x_{in} = x_{out} = 0\}$, then the target is $\mathbf{E}^*$ as explained above. One first determines by induction hypothesis a decomposition of $\mathbf{E}$ into groups of runs. In this case there is one group, as the run above is the unique minimal run.

Now we proceed to describe runs of $\mathbf{E}^*$ as sequences of *which group of* $\mathbf{E}$ *was used*. I.e. we performed one important mental step here: A run of $\mathbf{E}^*$ is now viewed as a sequence $\mathbf{C}_0 \to_{\mathbf{E}} \mathbf{C}_1 \to_{\mathbf{E}} \cdots \to_{\mathbf{E}} \mathbf{C}_r$ of steps in $\mathbf{E}$, i.e. every step is now one application of the outer loop, and we *abstract away* the information of how precisely these steps look, in particular how often the inner self-loops were taken. This leads to pumping a vector into a run having two cases: Either add more outer loops/transitions, the same as for VASSs, or *increase one of the already existing loops*. For example for the run $q_s(0,0,0) \to_{\mathbf{E}} q_t(0,0,1)$ described above, with respect to $\mathbf{E}^*$, we can pump both the y and z coordinates arbitrarily. Pumping y is an instance of *increasing an existing loop*: Change the existing loop by taking the inner self-loops. On the other hand, pumping z is the other type: We simply add more instances of the outer loop. The resulting PVASS without any nested loops is depicted in the right of Figure 1. Observe in particular that for pumping those vectors it was not necessary to add arbitrarily many repetitions of the outer loop which take the inner loops arbitrarily often.

Structure of the paper. In Section 2 we define a few preliminaries. Section 3 introduces VAS and Priority VAS. Our first result, the characterization of the reachability relations of Priority VAS in terms of regular expressions, is proved in Section 4. In Section 5 we define well-quasi-orders, and in particular our novel wqo on runs of Priority VAS. Section 5.4 introduces geometric preliminaries and previous results about VAS needed to state and prove our results. Section 6 defines flattability, and proves our main result.

[1] The wqo for VAS does not respect the zero tests and hence does not work, a new ordering is necessary.

2 Preliminaries

We let $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{Q}_{\geq 0}$ denote the sets of natural numbers containing 0, integers, and (non-negative) rational numbers. We use uppercase letters for sets/relations and boldface for vectors and sets/relations of vectors. For the i-th entry of a vector $\mathbf{x} \in \mathbb{Q}^d$ we write $\mathbf{x}(i)$.

Given sets $\mathbf{X}, \mathbf{Y} \subseteq \mathbb{Q}^d, Z \subseteq \mathbb{Q}$, we write $\mathbf{X} + \mathbf{Y} := \{\mathbf{x} + \mathbf{y} \mid \mathbf{x} \in \mathbf{X}, \mathbf{y} \in \mathbf{Y}\}$ for the Minkowski sum and $Z \cdot \mathbf{X} := \{\lambda \cdot \mathbf{x} \mid \lambda \in Z, \mathbf{x} \in \mathbf{X}\}$. By identifying elements $\mathbf{x} \in \mathbb{Q}^d$ with $\{\mathbf{x}\}$, we define $\mathbf{x} + \mathbf{X} := \{\mathbf{x}\} + \mathbf{X}$, and similarly $\lambda \cdot \mathbf{X} := \{\lambda\} \cdot \mathbf{X}$ for $\lambda \in \mathbb{Q}$.

A set $\mathbf{L} \subseteq \mathbb{N}^d$ is *linear* if $\mathbf{L} = \mathbf{b} + \mathbb{N}\mathbf{p}_1 + \cdots + \mathbb{N}\mathbf{p}_r$ with $\mathbf{b}, \mathbf{p}_1, \ldots, \mathbf{p}_r \in \mathbb{N}^d$. A relation $\mathbf{L} \subseteq \mathbb{N}^{d'} \times \mathbb{N}^{d''}$ is linear if it is linear when viewed as a set. A set/relation $\mathbf{S}$ is *semilinear* if it is a finite union of linear sets/relations. The semilinear sets/relations coincide with the sets/relations definable via formulas $\varphi \in \mathrm{FO}(\mathbb{N}, +)$, also called Presburger Arithmetic.

Given relations $\mathbf{R}_1 \subseteq \mathbb{N}^{d'} \times \mathbb{N}^{d_{mid}}$ and $\mathbf{R}_2 \subseteq \mathbb{N}^{d_{mid}} \times \mathbb{N}^{d''}$, we write $\mathbf{R}_1 \circ \mathbf{R}_2 = \{(\mathbf{v}, \mathbf{w}) \in \mathbb{N}^{d'} \times \mathbb{N}^{d''} \mid \exists \mathbf{x} \in \mathbb{N}^{d_{mid}} : (\mathbf{v}, \mathbf{x}) \in \mathbf{R}_1, (\mathbf{x}, \mathbf{w}) \in \mathbf{R}_2\}$ for composition. Given $\mathbf{R} \subseteq \mathbb{N}^{d'} \times \mathbb{N}^{d'}$, we write $\mathbf{R}^*$ for the reflexive and transitive closure (w.r.t. $\circ$).

Let $j, d', d'' \in \mathbb{N}$ with $j \leq d', d''$. A relation $\mathbf{R} \subseteq \mathbb{N}^{d'} \times \mathbb{N}^{d''}$ is *monotone in the j-th last coordinate* if for every $(\mathbf{x}, \mathbf{y}) \in \mathbf{R}$ we also have $(\mathbf{x} + \mathbf{e}_{d'+1-j}, \mathbf{y} + \mathbf{e}_{d''+1-j}) \in \mathbf{R}$, where $\mathbf{e}_k$ is the k-th unit vector[2]. A relation $\mathbf{R} \subseteq \mathbb{N}^{d'} \times \mathbb{N}^{d''}$ is *monotone* if $d' = d''$ and $\mathbf{R}$ is monotone in every coordinate.

3 Vector Addition Systems and Priority Vector Addition Systems

A *priority vector addition system with states* (PVASS) $\mathcal{V}$ of dimension $d \in \mathbb{N}$ is a finite directed multigraph (Q, E), whose edges e are labelled with a pair of a vector $f(e) \in \mathbb{Z}^d$ and a number $g(e) \in \{0, \ldots, d\}$. The set of configurations of $\mathcal{V}$ is $Q \times \mathbb{N}^d$. An edge $e = (p, p')$ with label $(f(e), g(e))$ induces a relation $\to_e$ on configurations via $\mathbf{c} = (q, \mathbf{x}) \to_e \mathbf{c}' = (q', \mathbf{x}')$ if and only if $q = p, q' = p'$, $\mathbf{x}(j) = 0$ for all $1 \leq j \leq g(e)$ and $\mathbf{x}' = \mathbf{x} + f(e)$. Intuitively, the edge can only be used in state p to move to state p' and adds the vector $f(e)$ to the current configuration. However, two conditions have to be fulfilled: We have to again arrive at a configuration $\mathbf{c}'$ (i.e. $\mathbf{x}'$ has to stay non-negative), and $\mathbf{x}$ must be 0 on the first $g(e)$ coordinates. We say that these coordinates are *tested for 0*. Observe that contrary to Minsky machines, if a counter i is tested for 0, also all smaller counters $j \leq i$ are tested for 0.

We write $\to_{\mathcal{V}} = \bigcup_{e \in E} \to_e$ and let $\to_{\mathcal{V}}^*$ denote its reflexive and transitive closure. A run of $\mathcal{V}$ is a finite sequence $\rho = (\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_k)$ of configurations such that $\mathbf{c}_i \to_{\mathcal{V}} \mathbf{c}_{i+1}$ for all $0 \leq i \leq k-1$. The *source* of the run ρ is the configuration $\mathrm{src}(\rho) := \mathbf{c}_0$, and the *target* is $\mathrm{tgt}(\rho) := \mathbf{c}_k$. The *pair of ends* of ρ is $\mathrm{ends}(\rho) = (\mathrm{src}(\rho), \mathrm{tgt}(\rho))$. A configuration tgt is reachable from src in $\mathcal{V}$ if $\mathrm{src} \to_{\mathcal{V}}^* \mathrm{tgt}$, or equivalently if there exists a run ρ with $\mathrm{ends}(\rho) = (\mathrm{src}, \mathrm{tgt})$.

A *priority vector addition system* (PVAS) $\mathcal{V}$ is a PVASS with only one state, a *vector addition system with states* (VASS) is a PVASS where $g(e) = 0$ for every edge e, i.e. no counter is ever tested for 0. A VAS is a PVAS which is also a VASS.

Since the class of reachability relations of PVASS is lacking some important closure properties, and we do not want to distinguish between PVASS and PVAS all the time, we instead consider a larger class of sets (which coincides for the two models). Intuitively, not every run is accepting anymore, instead a run has to start in a given initial state p and end in a given final state q, and certain counters have to start and/or end with fixed values. The idea is to then view the relation as subset of $\mathbb{N}^{d'} \times \mathbb{N}^{d''}$ where d', d'' are the number of input and respectively output counters which are not fixed.

[2] The reason for starting to count coordinates from the end will be explained in the next section.

▶ **Definition 1** ([5]). *A relation* $\mathbf{X} \subseteq \mathbb{N}^{d'} \times \mathbb{N}^{d''}$ *is a* (P)VASS section *if there exists a (P)VASS* $\mathcal{V}$ *of dimension* $d \geq d', d''$*, states* p, q *and vectors* $\mathbf{b}_s \in \mathbb{N}^{d-d'}, \mathbf{b}_t \in \mathbb{N}^{d-d''}$ *such that* $\mathbf{X} = \{(\mathbf{x}, \mathbf{y}) \in \mathbb{N}^{d'} \times \mathbb{N}^{d''} \mid (p, (\mathbf{b}_s, \mathbf{x})) \to_{\mathcal{V}}^* (q, (\mathbf{b}_t, \mathbf{y}))\}$.

This is the reason for defining monotonicity counting from the end: The same counter has different indices as unit vector because of fixing a different number of coordinates.

We write the section defined by the PVASS $\mathcal{V}$, the states p, q and the vectors $\mathbf{b}_s, \mathbf{b}_t$ as $\binom{\mathcal{V},p,q}{\mathbf{b}_s,\mathbf{b}_t}$. If $\mathcal{V}$ is a PVAS, then we leave away the unique states and only write $\binom{\mathcal{V}}{\mathbf{b}_s,\mathbf{b}_t}$. The reason for this notation is that PVASS sections should be viewed as an intersection of two relations: The reachability relation with fixed source state and target state, and the linear relation defined by the fixed coordinates. In the notation we like to split these parts.

We have three remarks on the definition of PVASS sections.

▶ Remark 2. We fix coordinates starting from the first, i.e. the most often zero tested coordinates are fixed first. This does not restrict the class of PVAS sections.

▶ Remark 3. At the cost of increasing the dimension by 3, states are a special case of fixed never-zero-tested coordinates [14], hence (P)VASS sections can equivalently be defined by (P)VAS. Furthermore, similar to how zero tests in Minsky machines can be assumed to only change the state, we will always require that $f(e)(j) = 0$ for all $j \leq g(e)$, i.e. any counter which is being zero tested is not updated. To obtain this assumption, simply move to an intermediate state from which you perform the additions afterwards.

▶ Remark 4. When using states, we can w.l.o.g. require $\mathbf{b}_s = 0^{d-d'}$ and $\mathbf{b}_t = 0^{d-d''}$, i.e. fixed coordinates are fixed to 0. However, since it is sometimes preferable to not use states, we allow general vectors $\mathbf{b}_s, \mathbf{b}_t$ for the fixed coordinates.

4 Equivalence of PVASS and Regular Expressions over VASS

Next we define the grammar which we will then prove to be equivalent to PVASS sections. Intuitively, one considers regular expressions where leaves/letters are VASS sections $\mathbf{Y}$. Intermediate relations might have different input and output dimensions, hence non-terminals depend on the dimensions, and composition requires matching dimensions.

▶ **Definition 5.** *Consider the following grammar with non-terminals* $\mathbf{E}_{d',d''}$ *for* $d', d'' \in \mathbb{N}$*:*

$$\mathbf{E}_{d',d''} = \mathbf{Y}_{d',d''} \mid \mathbf{E}_{d',d_{mid}} \circ \mathbf{E}_{d_{mid},d''} \mid \mathbf{E}_{d',d''} \cup \mathbf{E}_{d',d''}$$
$$\mathbf{E}_{d',d'} = \mathbf{Y}_{d',d'} \mid \mathbf{E}_{d',d_{mid}} \circ \mathbf{E}_{d_{mid},d'} \mid \mathbf{E}_{d',d'} \cup \mathbf{E}_{d',d'} \mid \mathbf{E}^*_{d',d'},$$

where the $\mathbf{Y}_{d',d''}$ *are VASS sections* $\subseteq \mathbb{N}^{d'} \times \mathbb{N}^{d''}$*. An expression* $\mathbf{E}_{d',d''}$ *defines in a natural way a relation* $\mathrm{Rel}(\mathbf{E}_{d',d''}) \subseteq \mathbb{N}^{d'} \times \mathbb{N}^{d''}$*, by interpreting* $\circ$ *as composition of relations,* $\cup$ *as union and* $*$ *as reflexive transitive closure. We usually write* $\mathbf{E}$ *instead of* $\mathbf{E}_{d',d''}$ *when the dimensions* $in(\mathbf{E}) = d'$ *and* $out(\mathbf{E}) = d''$ *are clear.*

Before we state the main theorem of this section, there are two things to note about this definition of the semantics. 1) $*$ by definition adds reflexivity, however it only does so in the non-fixed counters. This was one goal of the definition allowing different dimensions of intermediate objects. 2) The semantics for composition however are not as intuitive as it might seem, see the following example.

▶ **Example 6.** Let $\mathcal{V}$ be the 1-dimensional VAS with two transitions, incrementing x and decrementing x. Then its reachability relation is $\to_{\mathcal{V}}^* = \mathbb{N} \times \mathbb{N}$. Consider $\mathrm{Rel}(\binom{\mathcal{V}}{\epsilon,0} \circ \binom{\mathcal{V}}{1,\epsilon}) \subseteq \mathbb{N} \times \mathbb{N}$. We have $\mathrm{Rel}(\binom{\mathcal{V}}{\epsilon,0}) = \mathbb{N} \times \{\epsilon\}$ and $\mathrm{Rel}(\binom{\mathcal{V}}{1,\epsilon}) = \{\epsilon\} \times \mathbb{N}$, where we write $\epsilon \in \mathbb{N}^0$ for the

unique empty product. Despite the fixed coordinates $0, 1 \in \mathbb{N}$ not matching up, we obtain $\mathrm{Rel}(\binom{\mathcal{V}}{\epsilon,0} \circ \binom{\mathcal{V}}{1,\epsilon}) = \mathbb{N} \times \mathbb{N}$ by definition. This is due to the composition being only defined on the remaining, i.e. non-fixed coordinates.

We can now state the main theorem of this section.

▶ **Theorem 7.** *A relation $\mathbf{X} \subseteq \mathbb{N}^{d'} \times \mathbb{N}^{d''}$ is a PVASS section iff $\mathbf{X} = \mathrm{Rel}(\mathbf{E})$ for some $\mathbf{E}$.*

Proof. "⇒": Let $\mathbf{X} = \binom{\mathcal{V}}{\mathbf{b}_s,\mathbf{b}_t} \subseteq \mathbb{N}^{d'} \times \mathbb{N}^{d''}$ for a d-dimensional PVAS $\mathcal{V}$ *without states*, since they produce the same class of sections as in Remark 3. We will prove by induction on $k := \max_{e \in E} g(e)$, i.e. the maximal zero-tested counter, that every PVAS section $\mathbf{X}_k \subseteq \mathbb{N}^{d'} \times \mathbb{N}^{d''}$ has an equivalent expression $\mathbf{E}_k$ in the grammar. In the base case $k = 0$, $\mathcal{V}$ is actually a VAS, and hence $\mathbf{Y} := \mathbf{X}_k$ is an equivalent expression in the grammar.

Induction step: $k-1 \to k$: For $j \in \{k-1, k\}$ let $E_j := \{e \in E \mid g(e) \leq j\}$, in particular $Z_k := E_k \setminus E_{k-1}$ are the edges with $g(e) = k$, i.e. testing counters $1, \dots, k$. Let $\mathcal{V}_j$ be the PVASS with edges E_j and the same labels $f(e)$ and $g(e)$ as $\mathcal{V}$. By induction, for $\mathcal{V}_{k-1}$ and any vectors $\mathbf{b}_s, \mathbf{b}_t$ there exists $\mathbf{E}_{k-1,\mathbf{b}_s,\mathbf{b}_t}$ with $\mathrm{Rel}(\mathbf{E}_{k-1,\mathbf{b}_s,\mathbf{b}_t}) = \binom{\mathcal{V}_{k-1}}{\mathbf{b}_s,\mathbf{b}_t}$.

Importantly, the semantics $\to_e$ of a single action $e \in E$, even if e performs a zero test, can be defined by a VASS. In particular for Z_k we define the following d-dimensional VASS $\mathcal{V}_{Z_k}$ with $|Z_k| + 2$ states $Q = \{q_{in}, q_{fin}\} \cup Z_k$ and $2|Z_k|$ actions: For every $e \in Z_k$, let $e' = (q_{in}, e)$ with label $f(e') = f(e)$ and $e'' = (e, q_{fin})$ with label $f(e'') = \mathbf{0}$. Intuitively, we non-deterministically choose an $e \in Z_k$ and execute its action, afterwards moving to q_{fin}. We do not perform the zero test, instead this will be done using the VASS section. Namely we define $\mathbf{E}_{Z_k} := \binom{\mathcal{V}_{Z_k},q_{in},q_{fin}}{0^k,0^k}$, i.e. we require the first k counters to be 0 at the start and end. That their values will still be 0^k also at the end follows by the assumption in Remark 3, that zero tests do not change the counters they are testing. Then we define

$\mathbf{E}_{k,\mathbf{b}_s,\mathbf{b}_t} := \mathbf{E}_{k-1,\mathbf{b}_s,\mathbf{b}_t} \cup \mathbf{E}_{k-1,\mathbf{b}_s,0^k} \circ \left(\mathbf{E}_{k-1,0^k,0^k} \cup \mathbf{E}_{Z_k}\right)^* \circ \mathbf{E}_{k-1,0^k,\mathbf{b}_t}$.

Intuitively, the expression says the following: Either the zero testing actions in Z_k are never used, or we move from $\mathbf{b}_s$ to a configuration with the first k counters fixed to 0, then repeatedly either move to another configuration with those counters 0 without using Z_k, or we can use Z_k. The computation ends using $\mathcal{V}_{k-1}$ and reaching the given target $\mathbf{b}_t$.

Well-definedness: We have to prove that in this expression $\mathbf{E}_k$ the operations $\circ, \cup, *$ are only used on matching dimensions. This follows since our specified targets and sources coincide. For example in the union $\mathbf{E}_{k-1,0^k,0^k} \cup \mathbf{E}_{Z_k}$, both parts fix $0^k, 0^k$ as required.

Correctness: It is clear that $\mathrm{Rel}(\mathbf{E}_{k,\mathbf{b}_s,\mathbf{b}_t}) \subseteq \binom{\mathcal{V}_k}{\mathbf{b}_s,\mathbf{b}_t}$, since the expression describes a special form of runs from $\mathbf{b}_s$ to $\mathbf{b}_t$. For the other direction, let $(\mathbf{c}_0, \dots, \mathbf{c}_r)$ be a run of $\mathcal{V}_k$ such that $\mathbf{c}_0 \in \{\mathbf{b}_s\} \times \mathbb{N}^{d-d'}$ and $\mathbf{c}_r \in \{\mathbf{b}_t\} \times \mathbb{N}^{d-d''}$. We have to show that $(\mathbf{c}_0, \mathbf{c}_r) \in \mathrm{Rel}(\mathbf{E}_{k,\mathbf{b}_s,\mathbf{b}_t})$. Case 1: The run does not use actions in Z_k. Then the run shows membership in $\binom{\mathcal{V}_{k-1}}{\mathbf{b}_s,\mathbf{b}_t} \subseteq \mathrm{Rel}(\mathbf{E}_{k-1,\mathbf{b}_s,\mathbf{b}_t}) \subseteq \mathrm{Rel}(\mathbf{E}_{k,\mathbf{b}_s,\mathbf{b}_t})$.

Case 2: The run does use Z_k. Let $\pi_k \colon \mathbb{N}^d \to \mathbb{N}^{d-k}$ be the projection to the last $d-k$ coordinates, i.e. it removes the anyways fixed coordinates. Let $i_1, \dots, i_s$ be the indices such that $\mathbf{c}_{i_j} \to \mathbf{c}_{i_j+1}$ uses an action $\mathbf{a}_j \in Z_k$, i.e. $(\pi_k(\mathbf{c}_{i_j}), \pi_k(\mathbf{c}_{i_j+1})) \in \mathrm{Rel}(\mathbf{E}_{Z_k})$. Then the part of the run $(\mathbf{c}_{i_j+1}, \dots, \mathbf{c}_{i_{j+1}})$ does not use any Z_k transitions. Hence $(\pi_k(\mathbf{c}_{i_j+1}), \pi_k(\mathbf{c}_{i_{j+1}})) \in \binom{\mathcal{V}_{k-1}}{0^k,0^k} \subseteq \mathrm{Rel}(\mathbf{E}_{k-1,0^k,0^k})$ for all $j \in \{1, \dots, s\}$. Hence we already obtain $(\pi_k(\mathbf{c}_{i_1}), \pi_k(\mathbf{c}_{i_s+1})) \in \mathrm{Rel}((\mathbf{E}_{k-1,0^k,0^k} \cup \mathbf{E}_{Z_k})^*)$. Now similar to π_k, let $\pi_{d-d'}, \pi_{d-d''}$ be the projections removing the first $d-d', d-d''$ coordinates. Since $(\mathbf{c}_0, \dots, \mathbf{c}_{i_1})$ does not use Z_k and $\mathbf{c}_0 \in \{\mathbf{b}_s\} \times \mathbb{N}^{d-d'}$, we obtain $(\pi_{d-d'}(\mathbf{c}_0), \pi_k(\mathbf{c}_{i_1})) \in \binom{\mathcal{V}_{k-1}}{\mathbf{b}_s,0^k} \subseteq \mathrm{Rel}(\mathbf{E}_{k-1,\mathbf{b}_s,0^k})$ and similarly $(\pi_k(\mathbf{c}_{i_s+1}), \pi_{d-d''}(\mathbf{c}_r)) \in \binom{\mathcal{V}_{k-1}}{0^k,\mathbf{b}_t} \subseteq \mathrm{Rel}(\mathbf{E}_{k-1,0^k,\mathbf{b}_t})$. Altogether we obtain $(\pi_{d-d'}(\mathbf{c}_0), \pi_{d-d''}(\mathbf{c}_r)) \in \mathrm{Rel}(\mathbf{E}_{k,\mathbf{b}_s,\mathbf{b}_t})$.

"⇐": This follows by structural induction. The construction follows the standard conversion RegEx to ε-NFA, while adding some obvious zero tests.

As our definition of Rel($\mathbf{Y}$) is representation independent, we choose to represent every Rel($\mathbf{Y}$) using VASS of the same dimension d, and with $\mathbf{b}_s = 0^k$ and $\mathbf{b}_t = 0^l$ for some $k, l \in \mathbb{N}$. That every VASS section has a representation with $\mathbf{b}_s = 0^k$ and $\mathbf{b}_t = 0^l$ follows since one can simply add a new initial and final state q_{in}, q_{fin}, and add $\mathbf{b}_s$ when leaving q_{in} respectively subtract $\mathbf{b}_t$ when entering q_{fin}. To guarantee that all VASS have the same dimension d, add unused counters. We will prove by induction that every subexpression $\mathbf{E}'$ of the given starting expression $\mathbf{E}$ has an equivalent PVASS section, where the PVASS has dimension d and $\mathbf{b}_s = 0^{d-in(\mathbf{E})}, \mathbf{b}_t = 0^{d-out(\mathbf{E})}$. In the base case $\mathbf{E} = \mathbf{Y}$ we have required this above.

$\mathbf{E} \cup \mathbf{E}'$: By induction hypothesis, we can write Rel($\mathbf{E}$) = $\binom{\mathcal{V},p,q}{0^{d'},0^{d''}}$ and Rel($\mathbf{E}'$) = $\binom{\mathcal{V}',p',q'}{0^{d'},0^{d''}}$ using d dimensional PVASS $\mathcal{V}, \mathcal{V}'$. That both sections use the same d' and d'' follows by the restriction on expressions. Our new PVASS simply has a new initial and final state, and performs a non-deterministic choice whether to move to p and simulate $\mathcal{V}$ or move to state p' and simulate $\mathcal{V}'$.

Formally, write $\mathcal{V} = (Q, E)$ and $\mathcal{V} = (Q', E')$. We define the new PVASS $\mathcal{V}''$ as $(Q \cup Q' \cup \{q_{in}, q_{out}\}, E \cup E' \cup \{(q_{in}, p), (q_{in}, p'), (q, q_{out}), (q', q_{out})\}$. All copied edges keep their labels, the four new edges get the label $g(e) = 0$, i.e. they do not zero test, and $f(e) = 0^d$, i.e. they do not change the counters. It is easy to see that Rel($\mathbf{E} \cup \mathbf{E}'$) = $\binom{\mathcal{V}'',q_{in},q_{out}}{0^{d'},0^{d''}}$.

$\mathbf{E} \circ \mathbf{E}'$: By induction hypothesis, we can write $\mathbf{E} = \binom{\mathcal{V},p,q}{0^{d-d'},0^{d-d_{mid}}}$ and $\mathbf{E}' = \binom{\mathcal{V}',p',q'}{0^{d-d_{mid}},0^{d-d''}}$ using d dimensional PVASS $\mathcal{V}, \mathcal{V}'$. Our new PVASS first simulates $\mathcal{V}$, then checks that the first d_{mid} counters are 0, before simulating $\mathcal{V}'$. Checking the intermediate configuration will be done using a zero test.

Formally, write $\mathcal{V} = (Q, E)$ and $\mathcal{V}' = (Q', E')$. We define the PVASS $\mathcal{V}''$ as $(Q \cup Q', E \cup E' \cup \{(q, p')\}$, where prior edges keep their labels, and $f(q, p') = 0^d$ and $g(q, p') = d - d_{mid}$, i.e. we zero test the first $d - d_{mid}$ counters. It is easy to see that Rel($\mathbf{E} \circ \mathbf{E}'$) = $\binom{\mathcal{V}'',p,q'}{0^{d-d'},0^{d-d''}}$.

$\mathbf{E}^*$: By induction hypothesis, we can write $\mathbf{E} = \binom{\mathcal{V},p,q}{0^{d-d'},0^{d-d'}}$, where both vectors have the same number of fixed coordinates by the restriction on the grammar. We now simply add a new initial state and an edge from q to this new initial state which performs zero tests on the first $d - d'$ coordinates.

Formally, write $\mathcal{V} = (Q, E)$ and define $\mathcal{V}' = (Q \cup \{q_{in}\}, E \cup \{(q_{in}, p), (q, q_{in})\})$, where edges $e \in E$ keep their labels, and $f(q_{in}, p) = f(q, q_{in}) = 0^d$, i.e. counters are not changed, and $g(q_{in}, p) = g(q, q_{in}) = d - d'$, i.e. the first $d - d'$ counters are zero tested. It is easy to see that Rel($\mathbf{E}^*$) = $\binom{\mathcal{V}',q_{in},p}{0^{d-d'},0^{d-d'}}$. ◀

The "⇒" direction breaks down for a Minsky machine. Namely for $k = 2$, one subexpression is $(\binom{\mathcal{V}_1}{0^2,0^2} \cup \mathbf{E}_{Z_2})^*$. The parts we take the union of do not have the same dimension, as the first coordinate should be free in $\mathbf{E}_{Z_2}$ for a Minsky machine.

In future sections we will require expressions where $*$ is only used on relations $\mathbf{X}$ which are monotone. Surprisingly, we can without loss of generality require this. Before we prove this, let us first provide an example of a valid expression in the grammar where this fails.

▶ **Example 8.** Let $\mathcal{V}$ be the PVAS of dimension 2 without any transitions. Consider the expression $(\binom{\mathcal{V}}{\epsilon,0^1} \circ \binom{\mathcal{V}}{0^1,\epsilon})^*$. The expression below the $*$ says that you start with any configuration, fix the first counter to 0 and end with any configuration. Since the PVASS $\mathcal{V}$ does not have any transitions, in order for the composition to be possible, you have to have already started with the first counter equal to 0, and also end with such a configuration.

Hence the better expression would be $(\binom{\mathcal{V}}{0^1,0^1} \circ \binom{\mathcal{V}}{0^1,0^1})^*$. This expression fulfills the property that if $\mathbf{E}'$ is a subexpression of $\mathbf{E}^*$, then $in(\mathbf{E}') \geq in(\mathbf{E})$ and $out(\mathbf{E}') \geq out(\mathbf{E})$, i.e. interior nodes have fewer fixed coordinates. This will suffice for monotonicity.

▶ **Lemma 9.** *Let $d' \in \mathbb{N}$ and $\mathbf{E}$ expression such that every subexpression $\mathbf{E}'$ fulfills $in(\mathbf{E}') \geq d'$ and $out(\mathbf{E}') \geq d'$. Then $\mathrm{Rel}(\mathbf{E})$ is monotone in the j-th last coordinate for all $j \leq d'$.*

Proof. By structural induction. In the base case, $\mathbf{Y} = \binom{\mathcal{V},p,q}{\mathbf{b}_s,\mathbf{b}_t}$ for a VASS $\mathcal{V}$. Since VAS reachability relations are monotone in every coordinate, when the last d' coordinates are neither fixed in the input nor output then $\mathbf{Y}$ is monotone in these coordinates.

$\mathbf{E} \cup \mathbf{E}'$: Relations monotone in the j-last coordinate are clearly stable under union.

$\mathbf{E} \circ \mathbf{E}'$: Let $j \leq d'$, $d_1 = in(\mathbf{E}), d_2 = out(\mathbf{E})$, $d_3 = out(\mathbf{E}')$. Since the subexpression $\mathbf{E}$ fulfills $in(\mathbf{E}) \geq d'$ and $out(\mathbf{E}) \geq d'$ by assumption, $\mathrm{Rel}(\mathbf{E})$ is by induction monotone in the j-th last coordinate. Same for $\mathrm{Rel}(\mathbf{E}')$. Let $(\mathbf{x}_1, \mathbf{x}_2) \in \mathrm{Rel}(\mathbf{E})$ and $(\mathbf{x}_2, \mathbf{x}_3) \in \mathrm{Rel}(\mathbf{E}')$. By monotonicity in the j-th last coordinate we obtain $(\mathbf{x}_1 + \mathbf{e}_{d_1+1-j}, \mathbf{x}_2 + \mathbf{e}_{d_2+1-j}) \in \mathrm{Rel}(\mathbf{E})$ and $(\mathbf{x}_2 + \mathbf{e}_{d_2+1-j}, \mathbf{x}_3 + \mathbf{e}_{d_3+1-j}) \in \mathrm{Rel}(\mathbf{E}')$. Hence $(\mathbf{x}_1 + \mathbf{e}_{d_1+1-j}, \mathbf{x}_3 + \mathbf{e}_{d_3+1-j}) \in \mathrm{Rel}(\mathbf{E} \circ \mathbf{E}')$.

$\mathbf{E}^*$: Let $j \leq d'$. By repeatedly applying the case of $\circ$, we obtain for all $n \in \mathbb{N}$ that $\mathbf{E}^n$ is monotone in the j-th last coordinate. Again by stability of such relations under union, the relation $\mathbf{E}^* = \bigcup_{n\in\mathbb{N}} \mathbf{E}^n$ is monotone in the j-th last coordinate. ◀

▶ **Corollary 10.** *For every expression $\mathbf{E}$, there is another expression $\mathbf{E}'$ with $\mathrm{Rel}(\mathbf{E}) = \mathrm{Rel}(\mathbf{E}')$, where $*$ is only applied on monotone relations.*

Proof. Let $\mathbf{E}$ be an expression. Apply first $\Leftarrow$ and then $\Rightarrow$ of Theorem 7. The constructed expression fulfills for every subexpression $\mathbf{E}^*$ and subsubexpression $\mathbf{E}'$ of $\mathbf{E}^*$ that $in(\mathbf{E}') \geq in(\mathbf{E})$ and $out(\mathbf{E}') \geq out(\mathbf{E})$. Then by Lemma 9, $\mathrm{Rel}(\mathbf{E})$ is monotone in every one of the last $in(\mathbf{E}) = out(\mathbf{E})$ coordinates, i.e. in every coordinate. Hence $\mathrm{Rel}(\mathbf{E})$ is monotone. ◀

5 A Well-Quasi-Order on Runs of Priority VAS

Starting from this section, we also use $*$ for repeated concatenation, not only transitive closure $*$ of repeated composition. To distinguish them, we write $*_{concat}$ for concatenation $*$.

A partial order $(\mathbf{X}, \leq)$ is a reflexive, transitive and antisymmetric relation $\leq \subseteq \mathbf{X} \times \mathbf{X}$. A set $\mathbf{U} \subseteq \mathbf{X}$ is upward-closed if for all $\mathbf{x} \in \mathbf{U}$ and all $\mathbf{x}' \geq \mathbf{x}$ we have $\mathbf{x}' \in \mathbf{U}$. Every subset $\mathbf{X}' \subseteq \mathbf{X}$ is contained in a unique minimal upward-closed set $\lceil \mathbf{X}' \rceil := \{\mathbf{x}' \in \mathbf{X} \mid \exists \mathbf{x} \in \mathbf{X}' \colon \mathbf{x} \leq \mathbf{x}'\}$. A basis of an upward-closed set $\mathbf{U}$ is a subset $\mathbf{F} \subseteq \mathbf{U}$ such that $\lceil \mathbf{F} \rceil = \mathbf{U}$.

A partial order is a well-quasi-order if every upward closed set $\mathbf{U} \subseteq \mathbf{X}$ has a finite basis $\mathbf{F}$. Or equivalently, for every infinite sequence $\mathbf{x}_1, \mathbf{x}_2, \cdots \subseteq \mathbf{X}$ there are indices $i < j$ with $\mathbf{x}_i \leq \mathbf{x}_j$, or equivalently there are indices $(i_m)_{m\in\mathbb{N}}$ such that $\mathbf{x}_{i_m} \leq \mathbf{x}_{i_k}$ for all $m \leq k$.

Most well-quasi-orders, in particular the ones we will need, are constructed from the following basic ordering by applying standard closure properties stated afterwards:

▶ **Example 11.** Let $\mathbf{F}$ be finite. Then the equality relation $=$ is a well-quasi-order on $\mathbf{F}$.

▶ **Lemma 12.** *Let $(\mathbf{X}_1, \leq_1), (\mathbf{X}_2, \leq_2)$ be wqo's. Then $((\mathbf{X}_1 \cup \mathbf{X}_2, \leq_1 \cup \leq_2)$ is a wqo.*

▶ **Lemma 13** (Dickson's Lemma)**.** *$(\mathbf{X}_1, \leq_1), (\mathbf{X}_2, \leq_2)$ wqo's $\Rightarrow$ $(\mathbf{X}_1 \times \mathbf{X}_2, \leq_1 \times \leq_2)$ wqo.*

▶ **Lemma 14** (Higman's Lemma)**.** *Let $(\mathbf{\Sigma}, \leq)$ be a well-quasi-order. Then $(\Sigma^{*concat}, \leq^{*concat})$ is a well-quasi-order, where $\leq^{*concat}$ is the scattered subword ordering defined via $w = (\mathbf{x}_1, \ldots, \mathbf{x}_r) \leq w' = (\mathbf{y}_1, \ldots, \mathbf{y}_s)$ $\iff$ there exists an injective order preserving function $f \colon \{1, \ldots, r\} \to \{1, \ldots, s\}$ such that $\mathbf{x}_i \leq \mathbf{y}_{f(i)}$ for all $i \in \{1, \ldots, r\}$.*

5.1 Well-Quasi-Order for VAS

In order to explain the wqo for expressions, we start by defining Jancar's wqo ordering for runs of a VAS $\mathcal{V}$. A run is no longer viewed as a sequence of configurations, but instead as element of $\Omega(\mathcal{V}) = \mathbb{N}^d \times (\mathbb{N}^d \times E)^{*_{concat}} \times \mathbb{N}^d$, where E is the set of edges of the VAS. The first part is $\mathrm{src}(\rho)$, the last part is $\mathrm{tgt}(\rho)$ and the middle parts are the steps of the run. One can extend this to states by replacing $\mathbb{N}^d$ by $Q \times \mathbb{N}^d$ everywhere and requiring states to coincide. $\Omega(\mathcal{V})$ is well-quasi-ordered by Lemma 13 and Lemma 14.

This wqo is carefully engineered to ensure that the relation $\{\mathrm{ends}(\rho') \mid \rho' \geq \rho\} =: \mathrm{ends}(\rho) + \mathbf{P}_\rho$ has good properties. $\mathbf{P}_\rho$ is called the *transformer relation* of the run. A vector $(\mathbf{v}, \mathbf{w})$ is *pumpable into* ρ if $(\mathbf{v}, \mathbf{w}) \in \mathbf{P}_\rho$. The minimal property we want $\mathbf{P}_\rho$ to fulfill is closure under addition, i.e. if a vector can be pumped once, then it can be pumped arbitrarily often. To understand why exactly the above expression ensures this, consider Figure 2.

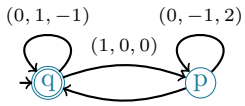


Figure 2 Typical example of a non-semilinear VAS [14]. Edges e are only labelled with their update $f(e)$, since this is a VASS, i.e. every edge fulfills $g(e) = 0$.

If we were to replace the middle part of the Jancar ordering with $(\mathbb{N}^d)^*$ instead, then the run $q(0,1,1) \to p(1,1,1)$ would be smaller than the run $q(0,1,1) \to q(0,2,0) \to p(1,2,0) \to p(1,1,2)$. Hence pumping properties would tell us that $p(1,1,n)$ should be reachable for every $n \geq 1$, which is obviously wrong. One might also consider the ordering without explicitly remembering the start and end configurations, but then some vectors with negative components might be claimed to be pumpable, since Higman might insert the smaller run in the middle. Pumping negative vectors is however trivially impossible.

Above we provided one way to define the transformer relation $\mathbf{P}_\rho$. As shown in [20, Lemma 7.5], there is an important equivalent characterization. Given a configuration $\mathbf{c}$, one defines the transformer relation $\mathbf{P}_\mathbf{c}$ for this configuration via $(\mathbf{x}, \mathbf{y}) \in \mathbf{P}_\mathbf{c} \iff \mathbf{c} + \mathbf{x} \to_\mathcal{V}^* \mathbf{c} + \mathbf{y}$. Intuitively, you utilize the existence of configuration $\mathbf{c}$ to *transform* $\mathbf{x}$ into $\mathbf{y}$. Another intuition is that $\mathbf{c}$ is a *capacity* which allows us to slightly enter the negative (up to $\mathbf{c}$). This relation is a generalization of "pumping possible via self-loop on a state". The equivalent characterization of $\mathbf{P}_\rho$ is: Write $\rho = (\mathbf{c}_0, \dots, \mathbf{c}_r)$, then $\mathbf{P}_\rho = \mathbf{P}_{\mathbf{c}_0} \circ \dots \circ \mathbf{P}_{\mathbf{c}_r}$.

Consider the example in Figure 1 in the introduction. At any configuration $\mathbf{c}_1$ in state q_1, we have $\mathbf{P}_{\mathbf{c}_1} = \{(x,y,z),(x',y,z) \mid x' \geq x\}$, i.e. one can "transform x into a larger number" and at a configuration $\mathbf{c}_2$ in state q_2, we have $\mathbf{P}_{\mathbf{c}_2} = \{(x,y,z),(x',y',z) \mid x+y = x'+y', x \geq x'\}$, i.e. one can "transform any number of x into the same number of y". This leads any run ρ through states $q_s \to q_1 \to q_2 \to q_t$ to be able to arbitrarily increase x and y, as mentioned in the introduction. Beware though that if a VASS has a complicated nested loop structure, then $\mathbf{P}_\mathbf{c}$ can be non-semilinear.

5.2 Well-Quasi-Order for Expressions

We will now define for every expression $\mathbf{E}$ a well-quasi-ordered set of runs $\Omega(\mathbf{E})$. We want the different segments of the runs to be labelled with which subexpression of $\mathbf{E}$ they belong to. Hence let Tag be a set containing unique start and end labels $\lambda_s(\mathbf{E}')$ and $\lambda_t(\mathbf{E}')$ for every node $\mathbf{E}'$ in the syntax tree of $\mathbf{E}$. The set $\Omega(\mathbf{E})$ will be modelled after the Jancar ordering, though configurations might now have different dimensions. It is important to

understand why $(\mathbb{N}^d \times E)^{*concat}$ has to be used for the Jancar ordering to lead to nice properties. The answer: It is equivalent to using $(\mathbb{N}^d \times E \times \mathbb{N}^d)^{*concat}$: When thinking of $\to_{\mathcal{V}}^*$ as $\{\to_{e_1} \cup \cdots \cup \to_{e_m}\}^*$, this means every letter is supposed to be a run of the expression $\mathbf{E}$ inside the $*$. For a VASS, this means $(\mathrm{src}, \mathrm{tgt}) \in \to_e$ tagged with the choice of e.

▶ **Definition 15.** *If $\mathbf{Y} \subseteq \mathbb{N}^{d'} \times \mathbb{N}^{d''}$ is a VAS section represented by VAS $\mathcal{V}$, $\mathbf{b}_s, \mathbf{b}_t$, let π_{in}, π_{out} be the projections projecting away fixed coordinates of in- and output. Let $\Omega_{\mathbf{b}_s,\mathbf{b}_t}$ be the set of runs $\rho \in \Omega(\mathcal{V})$, whose source and target have the correct values on fixed coordinates. We define $\Omega(\mathbf{Y}) = \{(\pi_{in}(\mathrm{src}(\rho)), \lambda_s(\mathbf{Y})w\lambda_t(\mathbf{Y}), \pi_{out}(\mathrm{tgt}(\rho))) \mid \rho = (\mathrm{src}(\rho), w, \mathrm{tgt}(\rho)) \in \Omega_{\mathbf{b}_s,\mathbf{b}_t}(\mathcal{V})\}$. I.e. we consider runs of $\mathcal{V}$ with source and target adhering to the fixed coordinates, add markers in the word w, and instead of storing the full source and target, we only store the non-fixed coordinates. For $\rho \in \Omega(\mathbf{Y})$, we refer to the projected configurations as $\mathrm{src}(\rho)$ and $\mathrm{tgt}(\rho)$.*

We define $\Omega(\mathbf{E}_1 \cup \mathbf{E}_2) = \Omega(\mathbf{E}_1) \cup \Omega(\mathbf{E}_2)$, i.e. we simply take unions of the sets of runs.

We define $\Omega(\mathbf{E}_1 \circ \mathbf{E}_2) = \{\lambda_s(\mathbf{E}_1)\rho_1\lambda_t(\mathbf{E}_1)\lambda_s(\mathbf{E}_2)\rho_2\lambda_t(\mathbf{E}_2) \mid \mathrm{tgt}(\rho_1) = \mathrm{src}(\rho_2)\} \subseteq Tag \times \Omega(\mathbf{E}_1) \times Tag^2 \times \Omega(\mathbf{E}_2) \times Tag$. I.e. we concatenate the runs if possible and use markers.

We define $\Omega(\mathbf{E}^*) = \{\lambda_s(\mathbf{E})\rho_1\lambda_t(\mathbf{E}) \dots \lambda_s(\mathbf{E})\rho_n\lambda_t(\mathbf{E}) \mid n \in \mathbb{N}, \rho_1, \dots, \rho_n \in \Omega(\mathbf{E}), \mathrm{tgt}(\rho_i) = \mathrm{src}(\rho_{i+1})\} \subseteq \mathbb{N}^{in(\mathbf{E}^*)} \times (\Omega(\mathbf{E}) \cup Tag)^{*concat} \times \mathbb{N}^{out(\mathbf{E}^*)}$. I.e. we consider all concatenations of any length $n \in \mathbb{N}$ and add tags splitting the different parts ρ_i.

▶ **Definition 16.** *We define a wqo $\leq_{\Omega(\mathbf{E})}$ on $\Omega(\mathbf{E})$ recursively. For VAS sections $\mathbf{Y}$ we use the Jancar ordering, observing that fixed coordinates coincide for every run, and can hence be ignored. For the recursive definition we use Lemmas 12, 13 and 14.*

Whenever we concatenate runs, we do not write the tags, because they can be inferred. Their existence is however important, as we can see for example for $\mathbf{E}_1 \circ \mathbf{E}_2$: We have $\rho \leq_{\Omega(\mathbf{E}_1 \circ \mathbf{E}_2)} \rho'$ if and only if $\rho = \rho_1\rho_2, \rho' = \rho_1'\rho_2'$ with $\rho_1, \rho_1' \in \Omega(\mathbf{E}_1), \rho_2, \rho_2' \in \Omega(\mathbf{E}_2)$ such that $\rho_1 \leq_{\Omega(\mathbf{E}_1)} \rho_1'$ and $\rho_2 \leq_{\Omega(\mathbf{E}_2)} \rho_2'$. This is where the tags will become important: From the tags, we can infer how ρ is supposed to be split into ρ_1 and ρ_2, and similarly for ρ'. Let us give a different example for why we need the tags for every subexpression. Imagine we consider the expression $\mathbf{Y} \cup \mathbf{Y}'$, where $\mathrm{Rel}(\mathbf{Y}) = \mathbb{N} \cdot (2, 1)$ and $\mathrm{Rel}(\mathbf{Y}') = \mathbb{N} \cdot (1, 2)$. If we wrote the empty run ρ as $(0, \epsilon, 0)$, i.e. did not label it, it would not be clear whether it can pump $\mathbb{N} \cdot (2, 1)$ or $\mathbb{N} \cdot (1, 2)$, this depends on which subexpression it belongs to.

5.3 Comparison with other Well-Quasi-Orders

In prior literature, some wqos for Priority VAS [3, 4] and even for the more general model of Grammar VAS [24] were introduced. In this subsection we compare our wqo to theirs.

In a grammar VAS paths are restricted to a given context-free grammar G, and it is known that Priority VAS correspond to the subclass of Grammar VAS, where the grammar is thin/finite index. In a thin grammar, every non-terminal has a rank (called index) which can only decrease and for every production $X \to YZ$, either $\mathrm{rank}(Y) < \mathrm{rank}(X)$ or $\mathrm{rank}(Z) < \mathrm{rank}(X)$, i.e. only one of the produced non-terminals can have the same rank.

The equivalence of PVASS with these grammars can in fact be seen via our RegEx characterization: One can implement $\mathbf{E}^*$ via $S \to XS$, where X implements $\mathbf{E}$. Similarly composition $\mathbf{E} \circ \mathbf{E}'$ can be implemented via $S \to XX'$ where X implements $\mathbf{E}$, and X' implements $\mathbf{E}'$. In [24] a wqo on runs based on Kruskal's tree ordering on syntax trees is defined. It can be shown that their ordering for the grammar obtained from a RegEx coincides with our ordering. In fact this is another motivation for the markers we use for splitting runs: Syntax trees naturally distinguish between being in the left or right branch of the RegEx. In [24] however they did not manage to show some of the pumping properties of the well-quasi-order which we require, and will be able to prove using our RegEx.

Another example are Bonnet's works [3, 4]. His well-quasi-order is based on a repeated application of Higman's Lemma, similar to our ordering. The only difference is that in [4] the finest split of any run is chosen. In our terminology, if the expression is $\mathbf{E}^*$, where $\mathbf{E}$ requires the first coordinate to be 0, then a run $\rho = \rho_1\rho_2$ is always split as $\rho_1, \rho_2 \in \Omega(\mathbf{E})$ in [4], i.e. $\rho \in \Omega(\mathbf{E}^2)$. While in our case also $\rho \in \Omega(\mathbf{E})$ is possible, this is determined by the markers. When limitting ourselves to runs with the finest split the orderings coincide.

5.4 Geometric Preliminaries

In this section we repeat some definitions from the VASS literature, pertaining to the pumping properties the relations $\mathbf{P}_\rho$ fulfill. Readers familiar with the notions can skip this section. For a visual representation of the geometric definitions see e.g. [10]. We state definitions for sets, they apply to relations $\mathbf{R} \subseteq \mathbb{Q}^{d'} \times \mathbb{Q}^{d''}$ by viewing them as set $\mathbf{R} \subseteq \mathbb{Q}^{d'+d''}$.

Cones and periodic sets. A set $\mathbf{C} \subseteq \mathbb{Q}^{\mathrm{d}}$ is a *cone* if $\mathbf{0} \in \mathbf{C}$, $\mathbf{C} + \mathbf{C} \subseteq \mathbf{C}$ and $\mathbb{Q}_{>0}\mathbf{C} \subseteq \mathbf{C}$. Given a set $\mathbf{F} \subseteq \mathbb{Q}^{\mathrm{d}}$, the cone generated by $\mathbf{F}$ is the smallest cone containing $\mathbf{F}$.

A cone $\mathbf{C}$ is *definable* if it is definable in $\mathrm{FO}(\mathbb{Q}, +, \geq)$.

A set $\mathbf{P} \subseteq \mathbb{N}^{\mathrm{d}}$ is a *periodic set* if $\mathbf{P} + \mathbf{P} \subseteq \mathbf{P}$ and $\mathbf{0} \in \mathbf{P}$. For any set $\mathbf{F} \subseteq \mathbb{N}^{\mathrm{d}}$, the periodic set $\mathbf{F}^*$ generated by $\mathbf{F}$ is the smallest periodic set containing $\mathbf{F}$. A periodic set $\mathbf{P}$ is *finitely generated* if $\mathbf{P} = \mathbf{F}^*$ for some finite set $\mathbf{F}$.

Finitely generated periodic sets provide an equivalent way to define linear sets as sets of the form $\mathbf{b} + \mathbf{P}$, where $\mathbf{b} \in \mathbb{N}^{\mathrm{d}}$ and $\mathbf{P} \subseteq \mathbb{N}^d$ is a finitely generated periodic set.

Smooth Periodic Sets. The periodic relation $\mathbf{P}_\rho$ for a run ρ of a VASS is rarely finitely generated, but it is smooth, a class introduced by Leroux in [22]. In order to define smooth, we first reintroduce the set of directions of a periodic set.

▶ **Definition 17.** *[22, 10] Let $\mathbf{P}$ be a periodic set. A vector $\mathbf{v} \in \mathbb{Q}^{\mathrm{d}}$ is a* direction *of $\mathbf{P}$ if there exists $m \in \mathbb{N}_{>0}$ and a point $\mathbf{x}$ such that $\mathbf{x} + \mathbb{N} \cdot m\mathbf{v} \subseteq \mathbf{P}$, i.e. some line in direction $\mathbf{v}$ is fully contained in $\mathbf{P}$. The set of directions of $\mathbf{P}$ is denoted* $\mathrm{dir}(\mathbf{P})$.

We can now define smooth periodic sets.

▶ **Definition 18.** *[22, 10] Let $\mathbf{P}$ be a periodic set.*

- *$\mathbf{P}$ is* asymptotically definable *if $\mathrm{dir}(\mathbf{P})$ is a definable cone.*
- *$\mathbf{P}$ is* well-directed *if every sequence $(\mathbf{p}_m)_{m\in\mathbb{N}}$ of vectors $\mathbf{p}_m \in \mathbf{P}$ has an infinite subsequence $(\mathbf{p}_{m_k})_{k\in\mathbb{N}}$ such that $\mathbf{p}_{m_j} + \mathbb{N}(\mathbf{p}_{m_k} - \mathbf{p}_{m_j}) \subseteq \mathbf{P}$ for all $k \geq j$.*
- *$\mathbf{P}$ is* smooth *if it is asymptotically definable and well-directed.*

▶ **Example 19.** Examples of smooth periodic sets are $\mathbf{P}_1 = \{(0,0)\} \cup (1,1) + \mathbb{N}^2$ and $\mathbf{P}_2 = \{(x,y) \in \mathbb{N}^2 \mid y \leq x^2\}$. We have $\mathrm{dir}(\mathbf{P}_2) \setminus \{(0,0)\} = \{(x,y) \in \mathbb{Q}_{\geq 0} \mid x > 0\}$. I.e. except pure north, every vector in $\mathbb{Q}^2_{\geq 0}$ is a direction of $\mathbf{P}_2$. On the other hand $\mathrm{dir}(\mathbf{P}_1) = \mathbb{Q}^2_{\geq 0}$.

$\mathbf{P}_2$ is a very typical example: One idea with $\mathrm{dir}(\mathbf{P})$ is to store the asymptotic steepness of the upper function, and ignore whether it is exponential or quadratic if it is superlinear.

▶ **Example 20.** Examples of non-smooth sets are $\mathbf{P}'_1 = \{(x,y) \mid x \geq \sqrt{2}y\}$ and $\mathbf{P}'_2 = (\{(0,1)\} \cup \{(2^m, 1) \mid m \in \mathbb{N}\})^* = \{(x,n) \in \mathbb{N}^2 \mid x \text{ has at most } n \text{ bits set to 1 in binary.}\}$. $\mathbf{P}'_1$ is not asymptotically definable, because defining $\mathrm{dir}(\mathbf{P}'_1)$ requires irrationals, while $\mathbf{P}'_2$ is not well-directed (see observation 2 below).

We make a few observations:

1. The set $\mathrm{dir}(\mathbf{P})$ is a cone. It is by definition closed under non-negative scalar multiplication (due to the m in the definition). Furthermore, if two lines in different directions $\mathbf{v}$ and $\mathbf{v}'$ are contained in $\mathbf{P}$, then by periodicity $\mathbf{P}$ also contains a $\mathbf{v}, \mathbf{v}'$ plane, and so $\mathbf{P}$ contains a line in every direction between $\mathbf{v}$ and $\mathbf{v}'$. For more details see [22, Lemma V.7]. $\mathrm{dir}(\mathbf{P})$ should be viewed as a kind of "limit cone" containing $\mathbf{P}$, it is however only one possible definition for a "limit cone" of $\mathbf{P}$, other cones were considered in prior papers [20, 21].
2. The definition of well-directed is stated this way to relate to wqo's, but the most important case of definition 18 is when the $\mathbf{p}_m$ are all on the same infinite line $\mathbf{x} + \mathbf{v} \cdot \mathbb{N}$. Then the definition equivalently states that $\mathbf{v} \in \mathrm{dir}(\mathbf{P})$, i.e. some infinite line in direction $\mathbf{v}$ is contained in $\mathbf{P}$. This makes sets where points are "too scarce" non-smooth. For instance, the set $\mathbf{P}_2$ of Example 19 contains infinitely many points on a horizontal line, but no full horizontal line, which would correspond to an arithmetic progression.

Almost semilinear relations. We reintroduce almost-semilinear sets, introduced by Leroux in [20, 21, 22]. Intuitively, they generalize semilinear sets by replacing finitely generated periodic sets with smooth periodic sets.

▶ **Definition 21** ([21, 22]). *A set* $\mathbf{X}$ *is* almost linear *if* $\mathbf{X} = \mathbf{b} + \mathbf{P}$*, where* $\mathbf{b} \in \mathbb{N}^d$ *and* $\mathbf{P}$ *is a smooth periodic set, and* almost semilinear *if it is a finite union of almost linear sets.*

It was shown in [21, 22] that VAS reachability sets/relations are almost semilinear. However, it is easy to find almost semilinear sets that are not reachability sets of any VAS. One reason is that the definition of a smooth periodic set only restricts the "asymptotic behavior" of the set, which can be "simple" even if the set itself is very "complex".

▶ **Example 22.** Let $\mathbf{X} \subseteq \mathbb{N}_{>0}$ be any set, for example $\mathbf{X} := \{m \in \mathbb{N} \mid m \text{ is Gödel-number of non-halting TM}\}$. Then $\mathbf{P} := \{(0,0)\} \cup (\{1\} \times \mathbf{X}) \cup \mathbb{N}^2_{>1}$ is a smooth periodic set. Indeed, it contains a line in every direction, and is thus well-directed and asymptotically definable.

To eliminate these types of sets Leroux required that every intersection of the set with a semilinear set is still almost semilinear. For instance, the intersection of the set $\mathbf{X}$ in Example 22 and the linear set $(1,0) + (0,1) \cdot \mathbb{N}$ is not almost semilinear. This leads to the following main theorems of [22], which we want to extend to Priority VAS:

▶ **Theorem 23** ([22, Theorem IX.1]). *For every semilinear relation* $\mathbf{S}$ *and reachability relation* $\mathbf{R}$ *of a VAS,* $\mathbf{R} \cap \mathbf{S}$ *is a finite union of relations* $\mathbf{b} + \mathbf{P}$*, where* $\mathbf{P}$ *is smooth periodic and for every linear relation* $\mathbf{L} \subseteq \mathbf{b} + \mathbf{P}$ *there exists a* $\mathbf{p} \in \mathbf{P}$ *such that* $\mathbf{p} + \mathbf{L}$ *is flattable.*

Since projecting away fixed coordinates preserves almost semilinearity, namely the periodic sets are anyways 0 on fixed coordinates, this theorem also holds for VAS sections.

As mentioned in the introduction, the same paper proceeded to prove the following:

▶ **Theorem 24** ([22, Theorem XI.2]). *The reachability relation of a VAS is flattable if and only if it is semilinear.*

The hard part is of course to prove that semilinear implies flattable. Let us quickly recap how to obtain Theorem 24 from Theorem 23 as our main proof will not repeat this step, we will stop at obtaining Theorem 23 for Priority VAS.

Leroux in [22] defines a dimension $\dim(\mathbf{S}) \in \mathbb{N}$ of semilinear sets $\mathbf{S}$. The important aspect of the dimension is that for every linear set $\mathbf{L}$, we have $\dim(\mathbf{L} \setminus (\mathbf{p} + \mathbf{L})) < \dim(\mathbf{L})$. For example $\mathbb{N}^2 \setminus [(x, y) + \mathbb{N}^2]$ for any $x, y \in \mathbb{N}$ is a finite union of lines, and hence 1-dimensional. The proof then proceeds by induction on the dimension of $\mathbf{X} := \mathbf{R} \cap \mathbf{S}$.

Base $k = 0$: Since 0-dimensional implies finite, such sets are flattable.

Step $k \to k + 1$: Let $\mathbf{X} = \mathbf{R} \cap \mathbf{S}$ semilinear with $\dim(\mathbf{X}) = k + 1$. We have to show that $\mathbf{X}$ is flattable. Since flattable relations are closed under union, we can assume that $\mathbf{X}$ is not only semilinear but even linear. Using $\mathbf{X}$ as $\mathbf{L}$ in Theorem 23 (this requires combining almost linear components correctly, and hence some fiddling) there exists a vector $\mathbf{p}$ such that $\mathbf{p} + \mathbf{X}$ is flattable. Since $\dim(\mathbf{X} \setminus (\mathbf{p} + \mathbf{X})) < \dim(\mathbf{X})$, $\mathbf{X} \setminus (\mathbf{p} + \mathbf{X})$ is flattable by induction. Since flattable relations are closed under union, $\mathbf{X} = [\mathbf{X} \setminus (\mathbf{p} + \mathbf{X})] \cup [\mathbf{p} + \mathbf{X}]$ is flattable.

With a similar induction on the dimension, Leroux obtained the following:

▶ **Theorem 25** ([20, Theorem 9.2]). *Let $\mathbf{R}$ be reflexive, transitive and such that for every semilinear $\mathbf{S}$, $\mathbf{R} \cap \mathbf{S}$ is almost semilinear. Then $\mathbf{R}$ admits semilinear inductive invariants.*

As a corollary of Theorem 25 and the PVAS version of Theorem 23, namely Theorem 31 of the next section, we obtain the following.

▶ **Corollary 26.** *Let $\mathcal{V}$ be a PVASS, and $\mathbf{C}_s, \mathbf{C}_t$ two configurations such that $\mathbf{C}_s \not\to^* \mathbf{C}_t$. Then there exists a semilinear inductive invariant $\mathbf{S}$ such that $\mathbf{C}_s \in \mathbf{S}, \mathbf{C}_t \notin \mathbf{S}$.*

Proof. Let $\mathbf{R}$ be the reachability relation of $\mathcal{V}$. Clearly $\mathbf{R}$ is reflexive and transitive. By Theorem 31, every intersection $\mathbf{R} \cap \mathbf{S}$ with a semilinear $\mathbf{S}$ is a finite union of $\mathbf{b} + \mathbf{P}$, where $\mathbf{P}$ is smooth periodic, i.e. $\mathbf{R} \cap \mathbf{S}$ is almost semilinear. Hence $\mathbf{R}$ fulfills all assumptions of Theorem 25, whose conclusion is the existence of a separating inductive invariant $\mathbf{S}$ as claimed. ◀

6 Semilinear Priority VAS are Flattable

In this section we define flattability and prove that Theorem 23 holds also for PVAS. We work on expressions $\mathbf{E}$, and hence define flattability for expressions $\mathbf{E}$ by structural induction, instead of defining flattability on PVAS directly.

▶ **Definition 27.** *Base case: $\mathbf{R} \subseteq \mathrm{Rel}(\mathbf{Y})$ is flattable w.r.t. $\mathbf{Y}$ if and only if $\mathbf{R}$ is semilinear.*

For $\mathbf{E}_1 \cup \mathbf{E}_2$, a relation $\mathbf{R} \subseteq \mathrm{Rel}(\mathbf{E}_1 \cup \mathbf{E}_2)$ is flattable w.r.t. $\mathbf{E}_1 \cup \mathbf{E}_2$ if and only if there exist relations $\mathbf{R}_i \subseteq \mathrm{Rel}(\mathbf{E}_i)$ flattable w.r.t. $\mathbf{E}_i$ such that $\mathbf{R} \subseteq \mathbf{R}_1 \cup \mathbf{R}_2$.

For $\mathbf{E}_1 \circ \mathbf{E}_2$, a relation $\mathbf{R} \subseteq \mathrm{Rel}(\mathbf{E}_1 \circ \mathbf{E}_2)$ is flattable w.r.t. $\mathbf{E}_1 \circ \mathbf{E}_2$ if and only if there exist relations $\mathbf{R}_i \subseteq \mathrm{Rel}(\mathbf{E}_i)$ flattable w.r.t. $\mathbf{E}_i$ such that $\mathbf{R} \subseteq \mathbf{R}_1 \circ \mathbf{R}_2$.

For $\mathbf{E}^*$, remember that by Corollary 10 we can assume that $\mathrm{Rel}(\mathbf{E})$ is monotone. Let $in(\mathbf{E}) = d'$. We first make a preliminary definition: Given a vector $\mathbf{v} = (\mathrm{src}, \mathrm{tgt}) \in \mathrm{Rel}(\mathbf{E})$, its closure under monotonicity is $m(\mathbf{v}) = \{\mathbf{v}\} + \mathbb{N}(\mathbf{e}_1, \mathbf{e}_1) + \cdots + \mathbb{N}(\mathbf{e}_{d'}, \mathbf{e}_{d'})$, where $\mathbf{e}_i \in \mathbb{N}^{d'}$ is the i-th unit vector. We define the monotone transitive closure of $\mathbf{v}$ as $\mathrm{mtc}(\mathbf{v}) = m(\mathbf{v})^*$, i.e. it is the relation of source and target configurations, such that the target can be reached by repeatedly applying only $\mathbf{v}$, potentially at a larger configuration. A linear path scheme $\mathbf{S}$ is a relation which can be written as $\mathrm{mtc}(\mathrm{src}_1, \mathrm{tgt}_1) \circ \cdots \circ \mathrm{mtc}(\mathrm{src}_r, \mathrm{tgt}_r)$ with $(\mathrm{src}_i, \mathrm{tgt}_i) \in \mathrm{Rel}(\mathbf{E})$. This has to be defined using mtc *since for expressions we do not have a finite set E of "edges" anymore. But we want to express that the same edge sequences are taken.*

A relation $\mathbf{R} \subseteq \mathrm{Rel}(\mathbf{E}^*)$ is flattable w.r.t. $\mathbf{E}^*$ if and only if there exist finitely many relations $\mathbf{R}_1, \ldots, \mathbf{R}_k \subseteq \mathrm{Rel}(\mathbf{E})$ flattable w.r.t. $\mathbf{E}$ and linear path schemes $\mathbf{S}_0, \ldots, \mathbf{S}_k$ such that $\mathbf{R} \subseteq \mathbf{S}_0 \circ \mathbf{R}_1 \circ \mathbf{S}_1 \circ \mathbf{R}_2 \circ \cdots \circ \mathbf{R}_k \circ \mathbf{S}_k$. I.e., in terms of the ideas in the introduction, we have k loops where we are allowed to adapt the "inner" transitions, those are reflected by $\mathbf{R}_i$. In between those loops, we are allowed to use linear path schemes of the outer loop.

Let us mention some basic properties of this definition, proved in the full version [9]. In particular, we compare this definition with the transition word definition of flattable.

▶ **Definition 28.** *Let $\mathcal{V} = (Q, E)$ be a PVAS. For $w = (e_1, \dots, e_k) \in E^*$ define $\to_w := \to_{e_1} \circ \cdots \circ \to_{e_k}$. Let $\to_w^*$ denote the reflexive and transitive closure of $\to_w$.*

A relation $\mathbf{R}$ is transition word flattable w.r.t. $\mathcal{V}$ if there exist transition sequences $w_1, \dots, w_r \in E^*$ such that $\mathbf{R} \subseteq \to_{w_1}^* \circ \cdots \circ \to_{w_r}^*$.

▶ **Lemma 29.** *Let $\mathbf{E}$ be an expression.*

1. *If $\rho' \geq_{\Omega(\mathbf{E})} \rho$ are runs, then $\mathrm{dir}(\rho) + \mathbb{N}(\mathrm{dir}(\rho') - \mathrm{dir}(\rho))$ is flattable w.r.t. $\mathbf{E}$, and all the corresponding runs are $\geq_{\Omega(\mathbf{E})} \rho$.*
2. *If $\mathbf{R}, \mathbf{R}' \subseteq \mathrm{Rel}(\mathbf{E})$ are flattable w.r.t. $\mathbf{E}$, then also $\mathbf{R} \cup \mathbf{R}'$ is.*
3. *Let $\mathcal{V}$ be a PVAS such that $\mathbf{E}$ is its expression in Theorem 7. If a relation $\mathbf{R} \subseteq \mathrm{Rel}(\mathbf{E})$ is flattable in our sense, then $\mathbf{R} \subseteq \to_{\mathcal{V}}^*$ is transition word flattable.*

In our proofs we will need to distribute + over ∘.

▶ **Lemma 30.** *Let $(\mathbf{R}_i)_{i=1}^r \subseteq \mathbb{N}^{d'} \times \mathbb{N}^{d_{mid}}$ and $(\mathbf{R}'_i)_{i=1}^r \subseteq \mathbb{N}^{d_{mid}} \times \mathbb{N}^{d''}$ be relations. Then $\sum_{i=1}^r (\mathbf{R}_i \circ \mathbf{R}'_i) \subseteq (\sum_{i=1}^r \mathbf{R}_i) \circ (\sum_{i=1}^r \mathbf{R}'_i)$.*

6.1 Proof Outline for Theorem 31

In this subsection we provide a proof outline for Theorem 31. As remarked in Section 5.4, Theorem 31 suffices to obtain semilinear inductive invariants and flattability.

▶ **Theorem 31.** *For every semilinear relation $\mathbf{S}$ and reachability relation $\mathbf{R}$ of a PVAS, $\mathbf{R} \cap \mathbf{S}$ is a finite union of relations $\mathbf{b} + \mathbf{P}$, where $\mathbf{P}$ is smooth periodic and for every linear relation $\mathbf{L} \subseteq \mathbf{b} + \mathbf{P}$ there exists a $\mathbf{p} \in \mathbf{P}$ such that $\mathbf{p} + \mathbf{L}$ is flattable.*

The starting point is to extend the definition of the transformer relation to expressions. Let $\mathbf{E}$ be an expression, and $\rho \in \Omega(\mathbf{E})$ a run. Then we define the *transformer relation of ρ w.r.t. $\mathbf{E}$* via $\mathbf{P}_{\mathbf{E},\rho} := \{\mathrm{ends}(\rho') - \mathrm{ends}(\rho) \mid \rho' \geq \rho\}$, exactly as in the VAS case. The corresponding almost semilinear component is $\mathrm{Comp}(\mathbf{E}, \rho) = \mathrm{ends}(\rho) + \mathbf{P}_{\mathbf{E},\rho}$.

The outline now consists of two parts: First we reduce Theorem 31 to Theorem 32 and Lemma 33. The closure property of Lemma 33 is easy to see, we give a proof in the full version [9]. Hence afterwards the outline will focus on proving Theorem 32.

▶ **Theorem 32.** *Let $\mathbf{E}$ be an expression, $\rho \in \Omega(\mathbf{E})$. Then $\mathbf{P}_{\mathbf{E},\rho}$ is smooth, periodic and:*

1. *Let $\mathbf{R}_1, \mathbf{R}_2 \subseteq \mathrm{Comp}(\mathbf{E}, \rho)$ flattable. Then $\mathbf{R}_1 + \mathbf{R}_2 - \mathrm{ends}(\rho)$ is flattable.*
2. *Every direction of $\mathrm{Comp}(\mathbf{E}, \rho)$ is flattable, i.e. for every $(\mathbf{e}, \mathbf{f}) \in \mathrm{dir}(\mathbf{P}_{\mathbf{E},\rho})$ there exists $(\mathbf{a}, \mathbf{b}) \in \mathbf{P}_{\mathbf{E},\rho}$, $n \in \mathbb{N}$ such that $\mathrm{ends}(\rho) + (\mathbf{a}, \mathbf{b}) + \mathbb{N}n(\mathbf{e}, \mathbf{f})$ is flattable w.r.t. $\mathbf{E}$.*

Property 2. is rather self-explanatory, important is property 1. The statement of property 1. is that if two relations are flattable using the same minimal run ρ, then not just the sum is flattable, but even the sum *ignoring the base point* $\mathrm{ends}(\rho)$ is flattable.

▶ **Lemma 33.** *Let $\mathbf{S} \subseteq \mathbb{N}^{d'} \times \mathbb{N}^{d''}$ be semilinear, and $\mathbf{X} \subseteq \mathbb{N}^{d'} \times \mathbb{N}^{d''}$ a PVAS section. Then $\mathbf{X} \cap \mathbf{S}$ is a PVAS section, and hence has an equivalent expression $\mathrm{Rel}(\mathbf{E})$ by Theorem 7.*

Proof of Theorem 31. Use Lemma 33 to obtain an expression $\mathbf{E}$ with $\mathrm{Rel}(\mathbf{E}) = \mathbf{R} \cap \mathbf{S}$. We write $\mathrm{Rel}(\mathbf{E}) = \bigcup_{\rho \in \Omega(\mathbf{E})_{min}} \mathrm{ends}(\rho) + \mathbf{P}_{\mathbf{E},\rho}$ as mentioned in the introduction. By Theorem 32, these periodic relations are smooth, hence only the flattability claim is left. Let $\mathbf{L} \subseteq \mathrm{ends}(\rho) + \mathbf{P}_{\mathbf{E},\rho}$ linear, i.e. $\mathbf{L} = \mathbf{b} + \mathbb{N}\mathbf{p}_1 + \cdots + \mathbb{N}\mathbf{p}_r$. By property 2. in Theorem 32, there exist

$n_i \in \mathbb{N}$ and $(\mathbf{a}_i, \mathbf{b}_i) \in \mathbf{P}_{\mathbf{E},\rho}$ such that $\mathbf{R}_i := \mathrm{ends}(\rho) + (\mathbf{a}_i, \mathbf{b}_i) + \mathbb{N} n_i \mathbf{p}_i$ is flattable for every i. We define the finite set $\mathbf{F} := \mathbf{b} + \{0, \dots, n_1\} \cdot \mathbf{p}_1 + \dots + \{0, \dots, n_r\} \cdot \mathbf{p}_r \subseteq \mathbf{L} \subseteq \mathrm{Comp}(\mathbf{E}, \rho)$. Since this set is finite, it is flattable. By property 1. in Theorem 32, the relation

$\sum_{i=1}^{r} \mathbf{R}_i + \mathbf{F} - r \cdot \mathrm{ends}(\rho) = \mathrm{ends}(\rho) + \sum_{i=1}^{r} [(\mathbf{a}_i, \mathbf{b}_i) + \mathbb{N} n_i \mathbf{p}_i] + (\mathbf{F} - \mathrm{ends}(\rho))$ is flattable. Defining $\mathbf{p} := \sum_{i=1}^{r} (\mathbf{a}_i, \mathbf{b}_i) + (\mathbf{b} - \mathrm{ends}(\rho)) \in \mathbf{P}_{\mathbf{E},\rho}$ by periodicity, the theorem is proven. ◀

Next we outline how to prove Theorem 32. The first step is an equivalent characterization of the transformer relation via "self-loop on $\mathbf{c}$" relations similar to $\mathcal{V}$. Let us start by defining these relations, which is similar to VAS.

Consider an expression $\mathbf{E}^*$ and a configuration $\mathbf{c} \in \mathbb{N}^{d'}$. We define the relation $\mathbf{P}_{\mathbf{E}^*,\mathbf{c}}$ via $(\mathbf{x}, \mathbf{y}) \in \mathbf{P}_{\mathbf{E}^*,\mathbf{c}} \iff \exists \rho \in \Omega(\mathbf{E}^*) : \mathrm{ends}(\rho) = (\mathbf{x} + \mathbf{c}, \mathbf{y} + \mathbf{c})$.

The equivalent characterization of $\mathbf{P}_{\mathbf{E},\rho}$ via relations $\mathbf{P}_{\mathbf{E}^*,\mathbf{c}}$ is as follows:

▶ **Lemma 34.** *Let $\mathbf{E}$ be an expression, and $\rho \in \Omega(\mathbf{E})$. Then there exist subexpressions $\mathbf{E}_i^*$ of $\mathbf{E}$ and configurations $\mathbf{c}_i$ occurring along ρ such that $\mathbf{P}_{\mathbf{E},\rho} = \mathbf{P}_{\mathbf{E}_1^*,\mathbf{c}_1} \circ \dots \circ \mathbf{P}_{\mathbf{E}_r^*,\mathbf{c}_r}$.*

This leaves us with three things to prove: Firstly, Lemma 34 itself. Secondly, that $\mathbf{P}_{\mathbf{E}^*,\mathbf{c}}$ is smooth and properties 1. and 2. of Theorem 32 hold for $\mathbf{P}_{\mathbf{E}^*,\mathbf{c}}$ (actually a slightly stronger version 2.' of property 2.). Thirdly, that composition preserves the properties of Theorem 32.

We dedicate one subsection to every step, with the second coming last. This is because steps 1 and 3 are new, while step 2 is based on [22]. Also, in order to understand why PVAS are flattable, Lemma 34 and step 3 contain the essence: Similar to VAS, pumping is a *sequence* of special self-loops, a very linear object. The fact that the different parts now use different expressions $\mathbf{E}_i^*$ is irrelevant for our composition proof.

6.2 Proving Lemma 34, Equivalent Definition of Transformer Relation

Proof by structural induction. For simplicity, we call a relation $\mathbf{P}_{\mathbf{E},\rho}$ decomposable if it has an equivalent description as in the lemma. The base case of VAS sections is clear.

$\mathbf{E}_1 \cup \mathbf{E}_2$: Let $\rho \in \Omega(\mathbf{E}_1 \cup \mathbf{E}_2)$. W.l.o.g. $\rho \in \Omega(\mathbf{E}_1)$. By definition of the wqo on runs in this case, we have $\mathbf{P}_{\mathbf{E}_1 \cup \mathbf{E}_2,\rho} = \mathbf{P}_{\mathbf{E}_1,\rho}$, which is decomposable by induction.

$\mathbf{E}_1 \circ \mathbf{E}_2$: Let $\rho \in \Omega(\mathbf{E}_1 \circ \mathbf{E}_2)$. Write $\rho = \rho_1 \rho_2$ with $\rho_1 \in \Omega(\mathbf{E}_1)$, $\rho_2 \in \Omega(\mathbf{E}_2)$, which is a unique split because of the tags on steps of the run. We claim that $\mathbf{P}_{\mathbf{E}_1 \circ \mathbf{E}_2,\rho} = \mathbf{P}_{\mathbf{E}_1,\rho_1} \circ \mathbf{P}_{\mathbf{E}_2,\rho_2}$.

Proof of claim: "$\subseteq$": Let $\rho' \geq \rho$. Then $\rho' = \rho_1' \rho_2'$ such that $\rho_1' \in \Omega(\mathbf{E}_1), \rho_2' \in \Omega(\mathbf{E}_2), \rho_1' \geq_{\mathbf{E}_1} \rho_1, \rho_2' \geq_{\mathbf{E}_2} \rho_2$. Then $\mathrm{ends}(\rho') - \mathrm{ends}(\rho) = (\mathrm{ends}(\rho_1') - \mathrm{ends}(\rho_1)) \circ (\mathrm{ends}(\rho_2') - \mathrm{ends}(\rho_2)) \in \mathbf{P}_{\mathbf{E}_1,\rho_1} \circ \mathbf{P}_{\mathbf{E}_2,\rho_2}$, where composition is possible since $\mathrm{tgt}(\rho_1') = \mathrm{src}(\rho_2'), \mathrm{tgt}(\rho_1) = \mathrm{src}(\rho_2)$.

The other direction "$\supseteq$" is clear, namely concatenate ρ_1' and ρ_2'.

Now simply use that both $\mathbf{P}_{\mathbf{E}_1,\rho_1}$ and $\mathbf{P}_{\mathbf{E}_2,\rho_2}$ are decomposable by induction.

This leaves the hardest case $\mathbf{E}^*$: For configurations $\mathbf{c}, \mathbf{c}' \in \mathbb{N}^{d'}$, we write $\mathbf{c} \to_{\mathbf{E}} \mathbf{c}'$ if there exists a run $\eta \in \Omega(\mathbf{E})$ such that $\mathrm{ends}(\eta) = (\mathbf{c}, \mathbf{c}')$, and call η a *generalized transition*. We want to emphasize the important fact that generalized transitions contain path information in $\Omega(\mathbf{E})$, and are not only an element of $\mathrm{Rel}(\mathbf{E})$. We let $\to_{\mathbf{E}}^*$ denote its reflexive and transitive closure. A first decomposition in this case contains relations $\mathbf{P}_{\mathbf{E},\eta_i}$, which correspond to "increasing existing transitions" as mentioned in the introduction. This leads to writing $\mathbf{P}_{\mathbf{E}^*,\rho}$ as a composition of alternating $\mathbf{P}_{\mathbf{E}^*,\mathbf{c}_i}$ and $\mathbf{P}_{\mathbf{E},\eta_i}$.

▶ **Lemma 35.** *Let $\rho = (\mathrm{src}(\rho), \eta_1 \dots \eta_r, \mathrm{tgt}(\rho)) \in \Omega(\mathbf{E}^*)$. The following equality holds: $\mathbf{P}_{\mathbf{E}^*,\rho} = \mathbf{P}_{\mathbf{E}^*,\mathrm{src}(\eta_1)} \circ \mathbf{P}_{\mathbf{E},\eta_1} \circ \mathbf{P}_{\mathbf{E}^*,\mathrm{src}(\eta_2)} \circ \dots \circ \mathbf{P}_{\mathbf{E},\eta_r} \circ \mathbf{P}_{\mathbf{E}^*,\mathrm{tgt}(\rho)}$.*

Proof. "$\Rightarrow$": Let $\rho' \geq_{\Omega(\mathbf{E}^*)} \rho$. We have to prove $\mathrm{ends}(\rho') - \mathrm{ends}(\rho) \in$ the claimed composition. Write $\rho' = \eta_1' \dots \eta_s'$ according to the tags. By definition of the wqo, there exists an order-preserving injective function $f\colon \{1, \dots, r\} \to \{1, \dots, s\}$ such that $\eta_i \leq_{\Omega(\mathbf{E})} \eta_{f(i)}'$. In particular,

$\text{ends}(\eta'_{f(i)}) \geq \text{ends}(\eta_i)$. We define the sequence of vectors $\mathbf{v}_i := \text{src}(\eta'_{f(i)}) - \text{src}(\eta_i)$ for $i \in \{1, \dots, r\}$, and $\mathbf{w}_i := \text{tgt}(\eta'_{f(i)}) - \text{tgt}(\eta_i)$ for $i \in \{1, \dots, r\}$. We also define $\mathbf{w}_0 := \text{src}(\rho') - \text{src}(\rho)$ and $\mathbf{v}_{r+1} := \text{tgt}(\rho') - \text{tgt}(\rho)$. For every $i \in \{1, \dots, r\}$, the run $\eta'_{f(i)}$ shows that $(\mathbf{v}_i, \mathbf{w}_i) \in \mathbf{P}_{\mathbf{E},\eta_i}$. The runs $\rho_i := \eta_{f(i)+1} \dots \eta_{f(i+1)-1}$ for $i \in \{1, \dots, r-1\}$ (these are possibly empty runs) prove that $\text{tgt}(\eta_i) + \mathbf{w}_i \to^*_{\mathbf{E}} \text{src}(\eta_{i+1}) + \mathbf{v}_{i+1}$. Since $\text{tgt}(\eta_i) = \text{src}(\eta_{i+1})$, we obtain $(\mathbf{w}_i, \mathbf{v}_{i+1}) \in \mathbf{P}_{\mathbf{E}^*,\text{tgt}(\eta_i)}$. A similar argument proves $(\mathbf{w}_0, \mathbf{v}_1) \in \mathbf{P}_{\mathbf{E}^*,\text{src}(\rho)}$ and $(\mathbf{w}_r, \mathbf{v}_{r+1}) \in \mathbf{P}_{\mathbf{E}^*,\text{tgt}(\rho)}$. Hence $\text{ends}(\rho') - \text{ends}(\rho) = (\mathbf{w}_0, \mathbf{v}_{r+1})$ is in the claimed composition.

"$\Leftarrow$": Let $(\mathbf{w}_i)_{i=0}^{r}$ and $(\mathbf{v}_i)_{i=1}^{r+1}$ such that $(\mathbf{w}_i, \mathbf{v}_{i+1}) \in \mathbf{P}_{\mathbf{E},\eta_i}$ for $i \in \{1, \dots, r\}$, $(\mathbf{v}_i, \mathbf{w}_i) \in \mathbf{P}_{\mathbf{E}^*,\text{tgt}(\eta_i)}$ for $i \in \{1, \dots, r-1\}$, $(\mathbf{w}_0, \mathbf{v}_1) \in \mathbf{P}_{\mathbf{E}^*,\text{src}(\rho)}$ and $(\mathbf{w}_r, \mathbf{v}_{r+1}) \in \mathbf{P}_{\mathbf{E}^*,\text{tgt}(\rho)}$. Let η'_i be generalized transitions witnessing $(\mathbf{w}_i, \mathbf{v}_{i+1}) \in \mathbf{P}_{\mathbf{E},\eta_i}$, let ρ_i for $i \in \{1, \dots, r-1\}$ be runs witnessing $(\mathbf{v}_i, \mathbf{w}_i) \in \mathbf{P}_{\mathbf{E}^*,\text{tgt}(\eta_i)}$, let ρ_0 witness $(\mathbf{w}_0, \mathbf{v}_1) \in \mathbf{P}_{\mathbf{E}^*,\text{src}(\rho)}$ and ρ_{r+1} witness $(\mathbf{w}_r, \mathbf{v}_{r+1}) \in \mathbf{P}_{\mathbf{E}^*,\text{tgt}(\rho)}$. Then $\rho' := \rho_0\eta'_1\rho_1 \dots \rho_{r-1}\eta'_r\rho_r$ fulfills $\rho' \in \Omega(\mathbf{E}^*)$ because sources and targets of the different parts coincide. In fact $\rho' \geq_{\Omega(\mathbf{E}^*)} \rho$, by choosing $f(i)$ to point at the index at which η'_i occurs in ρ'. ◀

This finishes proving Lemma 34 by observing that $\mathbf{P}_{\mathbf{E},\eta_i}$ are decomposable by induction.

We remark that the proof does obtain an explicit description of which $\mathbf{P}_{\mathbf{E}_i^*,\mathbf{c}_i}$ to use, but we stated Lemma 34 this way to stress that different $\mathbf{E}_i^*$ intertwine in the composition.

6.3 Preserving Smoothness and Flattability Under Composition

In this subsection we prove that if $\mathbf{P}_{\mathbf{E},\mathbf{c}}$ are smooth periodic relations fulfilling properties 1. and 2. of Theorem 32, then also their composition fulfills these conditions. While periodic relations are closed under composition (use e.g. Lemma 30), smooth periodic relations are not. We already slightly changed the definition of well-directed to accomplish this goal (compare with [22, 10]), but we still need to carefully choose the inductive statement. We choose to replace condition 2. by 2.' formulated as follows, which is similar to [22]:

2.': For every well-directed periodic $P \subseteq \mathbf{P}$ there exists a definable cone R such that $\text{dir}(P) \subseteq R$ and for every $(\mathbf{e}, \mathbf{f}) \in R$ there exist $(\mathbf{a}, \mathbf{b}) \in P, n \in \mathbb{N}$ such that $(\mathbf{a}, \mathbf{b}) + \mathbb{N}n(\mathbf{e}, \mathbf{f}) \subseteq \mathbf{P}$. In case of $\mathbf{P}_{\mathbf{E},\rho}$ we require $\text{ends}(\rho) + (\mathbf{a}, \mathbf{b}) + \mathbb{N}n(\mathbf{e}, \mathbf{f}) \subseteq \text{Comp}(\mathbf{E}, \rho)$ to be flattable.

The idea of property 2.' is to remove one basic difficulty of composition: Suddenly not all runs of $\mathbf{P}_1$ are useful anymore, only those which can be continued into $\mathbf{P}_2$. We will see in the proof how 2' takes care of this problem. Formally, property 2' says that even if $P \subseteq \mathbf{P}$ is non-smooth, we can find a definable R with $\text{dir}(P) \subseteq R \subseteq \text{dir}(\mathbf{P})$, and it only contains flattable directions. With the choice $P = \mathbf{P}$ this implies property 2. Important to notice is that the choice $R = \text{dir}(\mathbf{P})$ would always be best if not for the very crucial $\in P$ part, which we will use in the proof. Again, $\in P$ is easy to motivate. Imagine one is interested in the reachability set from a fixed point, i.e. in only pumping the target. Then choose $P := \{\mathbf{0}\} \times \mathbb{N}^d$. Property 2 would state existence of a line $(\mathbf{a}, \mathbf{b}) + \mathbb{N}(0, \mathbf{w}) \subseteq \mathbf{P}_{\mathbf{E},\rho}$. We actually want a line $(\mathbf{0}, \mathbf{b}) + \mathbb{N}(\mathbf{0}, \mathbf{w})$, i.e. with $(\mathbf{a}, \mathbf{b}) \in P$ as in property 2'.

▶ **Lemma 36.** *Let $\mathbf{P}_1, \mathbf{P}_2$ be smooth periodic relations fulfilling property 2.'. Then $\mathbf{P}_1 \circ \mathbf{P}_2$ is smooth periodic fulfilling property 2'. If $\mathbf{P}_1 = \mathbf{P}_{\mathbf{E}_1,\rho_1}$ and $\mathbf{P}_2 = \mathbf{P}_{\mathbf{E}_2,\rho_2}$ for $\rho = \rho_1\rho_2$, then in addition the flattability claims of property 1. and 2.' hold for $\mathbf{P}_{\mathbf{E}_1 \circ \mathbf{E}_2,\rho} = \mathbf{P}_{\mathbf{E}_1,\rho_1} \circ \mathbf{P}_{\mathbf{E}_2,\rho_2}$.*

Proof. Periodic: A composition of periodic relations is again periodic by Lemma 30.

Well-directed: Let $(\mathbf{v}_n, \mathbf{w}_n)_n \subseteq \mathbf{P}_1 \circ \mathbf{P}_2$ be a sequence. Then there exist intermediate values $\mathbf{x}_n$ such that $(\mathbf{v}_n, \mathbf{x}_n) \in \mathbf{P}_1, (\mathbf{x}_n, \mathbf{w}_n) \in \mathbf{P}_2$ for all n. Since $\mathbf{P}_1$ is well-directed, there exists a subsequence such that $(\mathbf{v}_{n_j}, \mathbf{x}_{n_j}) + \mathbb{N}(\mathbf{v}_{n_k} - \mathbf{v}_{n_j}, \mathbf{x}_{n_k} - \mathbf{x}_{n_j}) \subseteq \mathbf{P}_1$. Since $\mathbf{P}_2$ is well-directed we obtain additionally $(\mathbf{x}_{n_j}, \mathbf{w}_{n_j}) + \mathbb{N}(\mathbf{x}_{n_k} - \mathbf{x}_{n_j}, \mathbf{w}_{n_k} - \mathbf{w}_{n_j}) \subseteq \mathbf{P}_2$ for some subsubsequence. Together we have $(\mathbf{v}_{n_j}, \mathbf{w}_{n_j}) + \mathbb{N}(\mathbf{v}_{n_k} - \mathbf{v}_{n_j}, \mathbf{w}_{n_k} - \mathbf{w}_{n_j}) \subseteq \mathbf{P}_1 \circ \mathbf{P}_2$.

Property 2.': Let $P \subseteq \mathbf{P}_1 \circ \mathbf{P}_2$ be well-directed periodic. Define $P' := \{(\mathbf{v}, \mathbf{x}, \mathbf{w}) \mid (\mathbf{v}, \mathbf{w}) \in P, (\mathbf{v}, \mathbf{x}) \in \mathbf{P}_1, (\mathbf{x}, \mathbf{w}) \in \mathbf{P}_2\}$. P' is well-directed by an argument as above, but this time we even have to choose a subsubsubsequence. Consider the projections π_{12} and π_{23} to $(\mathbf{v}, \mathbf{x})$ and $(\mathbf{x}, \mathbf{w})$ respectively. $P_1 := \pi_{12}(P') \subseteq \mathbf{P}_1$ and $P_2 := \pi_{23}(P') \subseteq \mathbf{P}_2$ are projections of a well-directed periodic relation and therefore themselves well-directed periodic. Hence we can apply property 2.' for them to obtain definable cones R_1 and R_2 with $\mathrm{dir}(P_i) \subseteq R_i \subseteq \mathrm{dir}(\mathbf{P}_i)$. We claim property 2' holds with $R = R_1 \circ R_2$.

First we have to show that $\mathrm{dir}(P) \subseteq R_1 \circ R_2$. Let $(\mathbf{v}, \mathbf{w}) \in \mathrm{dir}(P)$. By definition of the set of directions, by potentially scaling with a positive integer, there exists $(\mathbf{v}_0, \mathbf{w}_0)$ such that $(\mathbf{v}_n, \mathbf{w}_n) := (\mathbf{v}_0, \mathbf{w}_0) + n(\mathbf{v}, \mathbf{w}) \in \mathbf{P}_1 \circ \mathbf{P}_2$ for every n. Therefore there exist intermediate values $\mathbf{x}_n$ such that $(\mathbf{v}_n, \mathbf{x}_n, \mathbf{w}_n) \in P'$ for all n. Since P' is well-directed, there exists a subsequence such that $(\mathbf{v}_{n_j}, \mathbf{x}_{n_j}, \mathbf{w}_{n_j}) + \mathbb{N}(\mathbf{v}_{n_k} - \mathbf{v}_{n_j}, \mathbf{x}_{n_k} - \mathbf{x}_{n_j}, \mathbf{w}_{n_k} - \mathbf{w}_{n_j}) \subseteq P'$. Hence $(\mathbf{v}_{n_k} - \mathbf{v}_{n_j}, \mathbf{x}_{n_k} - \mathbf{x}_{n_j}) \in \mathrm{dir}(P_1) \subseteq R_1$ and $(\mathbf{x}_{n_k} - \mathbf{x}_{n_j}, \mathbf{w}_{n_k} - \mathbf{w}_{n_j}) \in \mathrm{dir}(P_2) \subseteq R_2$. Therefore $(\mathbf{v}_{n_k} - \mathbf{v}_{n_j}, \mathbf{w}_{n_k} - \mathbf{w}_{n_j}) = (n_k - n_j)(\mathbf{v}, \mathbf{w}) \in R_1 \circ R_2$. This implies $(\mathbf{v}, \mathbf{w}) \in R_1 \circ R_2$.

Now let $(\mathbf{e}, \mathbf{g}) \in R$. Then there exists $\mathbf{f}$ such that $(\mathbf{e}, \mathbf{f}) \in R_1$ and $(\mathbf{f}, \mathbf{g}) \in R_2$. By definition of R_i, by potentially scaling, there exist $(\mathbf{a}_1, \mathbf{b}_1) \in P_1$ and $(\mathbf{b}_2, \mathbf{c}_2) \in P_2$ such that $(\mathbf{a}_1, \mathbf{b}_1) + \mathbb{N}(\mathbf{e}, \mathbf{f}) \subseteq \mathbf{P}_1$, and $(\mathbf{b}_2, \mathbf{c}_2) + \mathbb{N}(\mathbf{f}, \mathbf{g}) \subseteq \mathbf{P}_2$. Since P_1 and P_2 are projections of P', there exist $\mathbf{c}_1$ and $\mathbf{a}_2$ such that $(\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i) \in P'$ for $i \in \{1, 2\}$. Hence $(\mathbf{a}_2, \mathbf{b}_2) \in P_1$ and $(\mathbf{b}_1, \mathbf{c}_1) \in P_2$. By periodicity of $\mathbf{P}_1$, we have $(\mathbf{a}_1 + \mathbf{a}_2, \mathbf{b}_1 + \mathbf{b}_2) + \mathbb{N}(\mathbf{e}, \mathbf{f}) \subseteq \mathbf{P}_1$ and similarly by periodicity of $\mathbf{P}_2$ we have $(\mathbf{b}_1 + \mathbf{b}_2, \mathbf{c}_1 + \mathbf{c}_2) + \mathbb{N}(\mathbf{f}, \mathbf{g}) \subseteq \mathbf{P}_2$. Altogether we have $(\mathbf{a}_1 + \mathbf{a}_2, \mathbf{c}_1 + \mathbf{c}_2) \in P$ and $(\mathbf{a}_1 + \mathbf{a}_2, \mathbf{c}_1 + \mathbf{c}_2) + \mathbb{N}(\mathbf{e}, \mathbf{g}) \subseteq \mathbf{P}_1 \circ \mathbf{P}_2$ as required. $(\mathbf{a}_1, \mathbf{b}_1) \in P_1$ was crucial here such that we could obtain a fitting $\mathbf{c}_1$, and accordingly for $(\mathbf{b}_2, \mathbf{c}_2) \in P_2$.

Asymptotically definable: Define $P := \mathbf{P}_1 \circ \mathbf{P}_2$. By property 2', we have $\mathrm{dir}(\mathbf{P}_1 \circ \mathbf{P}_2) = R_1 \circ R_2$, which as composition of definable cones is itself a definable cone.

Flattability claim in 2': We proved 2' and constructed the direction using well-directedness. By Lemma 29(1.), relations obtained this way from the wqo. on runs are flattable.

Property 1.: Let $\mathbf{R}, \mathbf{R}' \subseteq \mathrm{ends}(\rho) + \mathbf{P}_{\mathbf{E}_1 \circ \mathbf{E}_2, \rho}$ be flattable w.r.t. $\mathbf{E}_1 \circ \mathbf{E}_2$. We have to prove that $\mathbf{R} + \mathbf{R}' - \mathrm{ends}(\rho)$ is flattable w.r.t. $\mathbf{E}_1 \circ \mathbf{E}_2$. By definition of flattable, there exist relations $\mathbf{R}_1, \mathbf{R}_1'$ flattable w.r.t. $\mathbf{E}_1$ and $\mathbf{R}_2, \mathbf{R}_2'$ flattable w.r.t. $\mathbf{E}_2$ such that $\mathbf{R} \subseteq \mathbf{R}_1 \circ \mathbf{R}_2$ and $\mathbf{R}' \subseteq \mathbf{R}_1' \circ \mathbf{R}_2'$. Hence $\mathbf{R} + \mathbf{R}' - \mathrm{ends}(\rho) \subseteq (\mathbf{R}_1 \circ \mathbf{R}_2) + (\mathbf{R}_1' \circ \mathbf{R}_2') - (\mathrm{ends}(\rho_1) \circ \mathrm{ends}(\rho_2))$. By Lemma 30 we obtain $\mathbf{R} + \mathbf{R}' - \mathrm{ends}(\rho) \subseteq (\mathbf{R}_1 + \mathbf{R}_1' - \mathrm{ends}(\rho_1)) \circ (\mathbf{R}_2 + \mathbf{R}_2' - \mathrm{ends}(\rho_2))$. Applying property 1. for the subexpressions $\mathbf{E}_1$ and $\mathbf{E}_2$, we obtain the claim. ◀

6.4 Transformer Relations are Smooth with Flattable Directions

In this subsection we prove that the transformer relation $\mathbf{P}_{\mathbf{E}^*, \mathbf{c}}$ is a smooth periodic relation fulfilling properties 1 and 2' (see Section 6.3). We start with a reminder of the notation.

We write $\mathbf{x} \to_{\mathbf{E}} \mathbf{y}$ for $(\mathbf{x}, \mathbf{y}) \in \mathrm{Rel}(\mathbf{E})$ and denote its reflexive transitive closure as $\to_{\mathbf{E}}^*$. Remember that $(\mathbf{x}, \mathbf{y}) \in \mathbf{P}_{\mathbf{E}^*, \mathbf{c}} \iff \mathbf{c} + \mathbf{x} \to_{\mathbf{E}}^* \mathbf{c} + \mathbf{y}$. We first prove that $\mathbf{P}_{\mathbf{E}^*, \mathbf{c}}$ is periodic.

▶ **Lemma 37.** *Let* $\mathbf{c} + \mathbf{x}_1 \to_{\mathbf{E}}^* \mathbf{c} + \mathbf{y}_1$ *and* $\mathbf{c} + \mathbf{x}_2 \to_{\mathbf{E}}^* \mathbf{c} + \mathbf{y}_2$. *Then* $\mathbf{c} + \mathbf{x}_1 + \mathbf{x}_2 \to_{\mathbf{E}}^* \mathbf{c} + \mathbf{y}_1 + \mathbf{y}_2$.

Proof. Since $*$ in expressions is only used on monotone relations $\mathrm{Rel}(\mathbf{E})$, $\to_{\mathbf{E}}$ is monotone. Hence also $\to_{\mathbf{E}}^*$ is monotone. By monotonicity, $\mathbf{c} + \mathbf{x}_1 + \mathbf{x}_2 \to_{\mathbf{E}}^* \mathbf{c} + \mathbf{y}_1 + \mathbf{x}_2 \to_{\mathbf{E}}^* \mathbf{c} + \mathbf{y}_1 + \mathbf{y}_2$. ◀

That $\mathbf{P}_{\mathbf{E}^*, \mathbf{c}}$ is well-directed follows from Lemma 29 (1.).

The proof of Property 1. is similar to the above proof of Lemma 37. Indeed, the lemma did not duplicate $\mathbf{c}$, so in different notation $\mathbf{r}_1 \to_{\mathbf{E}}^* \mathbf{s}_1$ and $\mathbf{r}_2 \to_{\mathbf{E}}^* \mathbf{s}_2$ imply $\mathbf{r}_1 + \mathbf{r}_2 - \mathbf{c} \to_{\mathbf{E}}^* \mathbf{s}_1 + \mathbf{s}_2 - \mathbf{c}$. This leaves proving that $\mathbf{P}_{\mathbf{E}^*, \mathbf{c}}$ fulfills property 2.' and hence is asymptotically definable.

Since $(\mathbf{0},\mathbf{0}) \in \mathbf{P}_{\mathbf{E}^*,\mathbf{c}}$, monotonicity implies that $\mathbf{P}_{\mathbf{E}^*,\mathbf{c}}$ is reflexive. In [22, Section VIII] many lemmas were proven for reflexive periodic relations, we reuse many of them. This leads to a long sequence of restating lemmas, we prefer to end this part of the main text by sketching the idea, for the formal proof see the full version [9].

Let $P \subseteq \mathbf{P}_{\mathbf{E}^*,\mathbf{c}}$ periodic. In order to not confuse $\Omega(\mathbf{E})$ and $\Omega(\mathbf{E}^*)$, we write $\eta \in \Omega(\mathbf{E})$ and $\rho \in \Omega(\mathbf{E}^*)$. We call η a *generalized transition*. Write $\gamma = (\mathbf{E}^*, \mathbf{c}, P)$ and denote by Ω_γ the set of runs $\rho \in \Omega(\mathbf{E}^*)$ such that $\mathrm{ends}(\rho) \in (\mathbf{c},\mathbf{c}) + P$. The idea is to split counters into bounded and unbounded counters for Ω_γ. The bounded counters will all be stored in the states of a graph, and this leads to pumping corresponding to cycles in the graph. Namely any transition sequence will correspond to a path in the graph, and since bounded counters cannot be pumped, any pumping sequence has to restore all the bounded counters, i.e. be a cycle in the graph. The unbounded counters on the other hand will all be unbounded *simultaneously*, at which point the condition that counters have to stay non-negative will intuitively not influence possible behaviours anymore.

We hence define the graph of bounded counters. Let $Q_\gamma \subseteq \mathbb{N}^{d'}$ be the set of configurations occurring on some run $\rho \in \Omega_\gamma$. We denote by I_γ the set of indices such that $\{q(i) \mid q \in Q_\gamma\}$ is finite, i.e. the set of bounded counters. We consider the projection $\pi_\gamma \colon \mathbb{N}^{d'} \to \mathbb{N}^{I_\gamma}$ to the bounded counters. We now define a finite directed multigraph G_γ with vertices $S_\gamma := \pi_\gamma(Q_\gamma)$. For edges (s,t), first consider the set $\Omega_{s,t}$ of generalized transitions η with $\pi_\gamma(\mathrm{src}(\eta)) = s$ and $\pi_\gamma(\mathrm{tgt}(\eta)) = t$, which occur in some run $\rho \in \Omega_\gamma$. We add an edge (s,t) for every minimal (w.r.t. $\leq_{\Omega(\mathbf{E})}$) element of $\Omega_{s,t}$. We let $s_\gamma = \pi_\gamma(\mathbf{c})$ denote the "initial state" for this graph.

Clearly runs correspond to paths in G_γ, since the graph has projections of configurations as states, and generalized transitions as edges. Regarding unbounded counters, the proof that all of them are unbounded *simultaneously* is in the full version [9]. Finally, we then obtain a formula for a definable cone R overapproximating $\mathrm{dir}(P)$ by first considering the finitely many minimal cycles in G_γ. Every cycle $\eta_1 \dots \eta_m$ provides us with a smooth periodic relation $\mathbf{P}_{\mathbf{E},\eta_1} \circ \dots \circ \mathbf{P}_{\mathbf{E},\eta_m}$ of pumping possible along this cycle. These relations are smooth periodic by induction and Lemmas 34 and 36. We conclude using [22, Theorem VII.1], whose statement is essentially the following: If $\mathbf{P}_1, \dots, \mathbf{P}_m$ are *reflexive* asymptotically definable periodic relations, then $(\bigcup_{i=1}^m \mathbf{P}_i)^*$ is also asymptotically definable.

7 Conclusion

We have given a new characterization of PVAS sections as RegEx over VAS sections, and extended the abstract properties of almost semilinear sets to PVAS sections. We have concluded that therefore if the reachability relation of a PVAS is semilinear, then it is flattable, and moreover if a configuration is not reachable, then it is separated by a semilinear inductive invariant. This leaves two main unknowns for PVAS which are known for VAS: 1) The decidability of the semilinearity problem, that is, given a PVAS, decide if its reachability relation/set is semilinear. 2) The complexity of the reachability problem.

We leave these as future work. We believe that combining our characterization of PVAS sections and wqo on runs with ideas from [10] might allow for progress on these open problems.

Furthermore, this research can also be viewed as progress towards the open question whether the reachability problem for Pushdown/Grammar VASS is decidable, where Pushdown VASS have a stack in addition to the counters. Namely it is known [1] that Priority VASS are equivalent to a subclass of Pushdown VASS.

References

1 Mohamed Faouzi Atig and Pierre Ganty. Approximating petri net reachability along context-free traces. In *FSTTCS*, volume 13 of *LIPIcs*, pages 152–163. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011.

2 Michael Blondin and François Ladouceur. Population protocols with unordered data. In *ICALP*, volume 261 of *LIPIcs*, pages 115:1–115:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

3 Rémi Bonnet. The reachability problem for vector addition system with one zero-test. In *MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 145–157. Springer, 2011.

4 Rémi Bonnet. *Theory of Well-Structured Transition Systems and Extended Vector-Addition Systems*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, 2013. URL: `http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/bonnet-phd13.pdf`.

5 Lorenzo Clemente, Wojciech Czerwinski, Slawomir Lasota, and Charles Paperman. Separability of reachability sets of vector addition systems. In *STACS*, volume 66 of *LIPIcs*, pages 24:1–24:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

6 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for petri nets is not elementary. In *STOC*, pages 24–33. ACM, 2019.

7 Wojciech Czerwinski and Lukasz Orlikowski. Reachability in vector addition systems is ackermann-complete. In *FOCS*, pages 1229–1240. IEEE, 2021.

8 Alain Finkel, Jérôme Leroux, and Grégoire Sutre. Reachability for two-counter machines with one test and one reset. In *FSTTCS*, volume 122 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

9 Roland Guttenberg. Flattability of priority vector addition systems, 2024. `arXiv:2402.09185`.

10 Roland Guttenberg, Mikhail A. Raskin, and Javier Esparza. Geometry of reachability sets of vector addition systems. In *CONCUR*, volume 279 of *LIPIcs*, pages 6:1–6:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

11 Christoph Haase. A survival guide to presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018.

12 Christoph Haase and Georg Zetzsche. Presburger arithmetic with stars, rational subsets of graph groups, and nested zero tests. In *LICS*, pages 1–14. IEEE, 2019.

13 Piotr Hofman, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, Sylvain Schmitz, and Patrick Totzke. Coverability trees for petri nets with unordered data. In *FoSSaCS*, volume 9634 of *Lecture Notes in Computer Science*, pages 445–461. Springer, 2016.

14 John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comput. Sci.*, 8:135–159, 1979.

15 Petr Jancar. Decidability of a temporal logic problem for petri nets. *Theor. Comput. Sci.*, 74(1):71–93, 1990.

16 S. Rao Kosaraju. Decidability of reachability in vector addition systems. In *STOC*, pages 267–281. ACM, 1982.

17 Jean-Luc Lambert. A structure to decide reachability in petri nets. *Theor. Comput. Sci.*, 99(1):79–104, 1992.

18 Ranko Lazic and Sylvain Schmitz. The complexity of coverability in ν-petri nets. In *LICS*, pages 467–476. ACM, 2016.

19 Jérôme Leroux. The general vector addition system reachability problem by presburger inductive invariants. In *LICS*, pages 4–13. IEEE Computer Society, 2009.

20 Jérôme Leroux. Vector addition system reachability problem: A short self-contained proof. In *LATA*, volume 6638 of *Lecture Notes in Computer Science*, pages 41–64. Springer, 2011.

21 Jérôme Leroux. Vector addition systems reachability problem (A simpler solution). In *Turing-100*, volume 10 of *EPiC Series in Computing*, pages 214–228. EasyChair, 2012.

22 Jérôme Leroux. Presburger vector addition systems. In *LICS*, pages 23–32. IEEE Computer Society, 2013. URL: `https://hal.science/hal-00780462v2`.

23 Jérôme Leroux. The reachability problem for petri nets is not primitive recursive. In *FOCS*, pages 1241–1252. IEEE, 2021.

24 Jérôme Leroux, M. Praveen, Philippe Schnoebelen, and Grégoire Sutre. On functions weakly computable by pushdown petri nets and related systems. *Log. Methods Comput. Sci.*, 15(4), 2019.

25 Jérôme Leroux, M. Praveen, and Grégoire Sutre. Hyper-ackermannian bounds for pushdown vector addition systems. In *CSL-LICS*, pages 63:1–63:10. ACM, 2014.

26 Jérôme Leroux and Grégoire Sutre. Reachability in two-dimensional vector addition systems with states: One test is for free. In *CONCUR*, volume 171 of *LIPIcs*, pages 37:1–37:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

27 Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. On the coverability problem for pushdown vector addition systems in one dimension. In *ICALP (2)*, volume 9135 of *Lecture Notes in Computer Science*, pages 324–336. Springer, 2015.

28 Ernst W. Mayr. An algorithm for the general petri net reachability problem. In *STOC*, pages 238–246. ACM, 1981.

29 Ruzica Piskac and Viktor Kuncak. Linear arithmetic with stars. In *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 268–280. Springer, 2008.

30 Klaus Reinhardt. Reachability in petri nets with inhibitor arcs. In *RP*, volume 223 of *Electronic Notes in Theoretical Computer Science*, pages 239–264. Elsevier, 2008.

31 Fernando Rosa-Velardo and David de Frutos-Escrig. Decidability and complexity of petri nets with unordered data. *Theor. Comput. Sci.*, 412(34):4439–4451, 2011.

An Efficient Quantifier Elimination Procedure for Presburger Arithmetic

Christoph Haase
Department of Computer Science, University of Oxford, UK

Shankara Narayanan Krishna
Department of Computer Science & Engineering, IIT Bombay, India

Khushraj Madnani
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Om Swostik Mishra
Department of Mathematics, IIT Bombay, India

Georg Zetzsche
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Abstract

All known quantifier elimination procedures for Presburger arithmetic require doubly exponential time for eliminating a single block of existentially quantified variables. It has even been claimed in the literature that this upper bound is tight. We observe that this claim is incorrect and develop, as the main result of this paper, a quantifier elimination procedure eliminating a block of existentially quantified variables in singly exponential time. As corollaries, we can establish the precise complexity of numerous problems. Examples include deciding (i) monadic decomposability for existential formulas, (ii) whether an existential formula defines a well-quasi ordering or, more generally, (iii) certain formulas of Presburger arithmetic with Ramsey quantifiers. Moreover, despite the exponential blowup, our procedure shows that under mild assumptions, even NP upper bounds for decision problems about quantifier-free formulas can be transferred to existential formulas. The technical basis of our results is a kind of small model property for parametric integer programming that generalizes the seminal results by von zur Gathen and Sieveking on small integer points in convex polytopes.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases Presburger arithmetic, quantifier elimination, parametric integer programming, convex geometry

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.142

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding Funded by the European Union (ERC, FINABIS, 101077902). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.
Christoph Haase: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 852769, ARiAT).
Khushraj Madnani: Supported in part by the Deutsche Forschungsgemeinschaft (DFG) project 389792660 TRR 248–CPEC.

Acknowledgements We are grateful to (i) Pascal Bergsträßer, Moses Ganardi, and Anthony W. Lin for discussions about Weispfenning's lower bound, (ii) Pascal Baumann, Eren Keskin, Roland Meyer for discussions on polyhedra, (iii) Anthony W. Lin and Matthew Hague for explaining some aspects of their results on monadic decomposability, and (iv) Gaëtan Regaud for proofreading.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 142; pp. 142:1–142:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Presburger arithmetic is the first-order theory of the integers with addition and order. This theory was shown decidable by Mojżesz Presburger in 1929 [25] by establishing a quantifier elimination procedure in the extended structure additionally consisting of infinitely many predicates $m \mid \cdot$ for all integers $m > 0$, asserting divisibility by a constant. Recall that a logical theory T admits quantifier elimination whenever for any formula $\Phi(y_1, \ldots, y_k) \equiv \exists x\, \varphi(x, y_1, \ldots, y_k)$ with φ being quantifier free there is a computable quantifier-free formula $\Psi(y_1, \ldots, y_k)$ such that $\Phi \leftrightarrow \Psi$ is a tautology in T. Presburger's quantifier elimination procedure has non-elementary running time. In the early 1970s, Cooper [6] developed an improved version of Presburger's procedure, which was later shown to run in triply exponential time [23]. Ever since, various other quantifier elimination procedures have been established and analyzed, especially for fragments of Presburger arithmetic with a fixed number of quantifier alternations, see e.g. [26, 32]. Weispfenning [33] analyzed lower bounds for quantifier-elimination procedures and showed that, assuming unary encoding of numbers, *any* quantifier elimination procedure requires triply exponential time. In the same paper, Weispfenning also claims that any algorithm eliminating a single block of existential quantifiers inherently requires *doubly* exponential time [33, p. 50].

The main contribution of this paper is to develop a quantifier elimination procedure for Presburger arithmetic that eliminates a block of existentially quantified variables in *singly* exponential time. This, of course, contradicts Weispfenning's claim, which actually turns out to be incorrect as we point out in detail in Appendix C. The key technical insight underlying our procedure is a kind of small model property for parametric integer programming. Given an integer matrix $A \in \mathbb{Z}^{\ell \times n}$ and $\boldsymbol{b} \in \mathbb{Z}^{\ell}$, recall that integer programming is to decide whether there is some $\boldsymbol{x} \in \mathbb{Z}^n$ such that $A\boldsymbol{x} \leq \boldsymbol{b}$. It is well-known by the work of von zur Gathen and Sieveking [31], and Borosh and Treybig [4], that if such an $\boldsymbol{x}$ exists then there is one whose bit length is polynomially bounded in the bit lengths of A and $\boldsymbol{b}$. In this paper, we refer to the situation in which $\boldsymbol{b}$ is not fixed and provided as a parameter as *parametric integer programming*. Our main technical result states that, in this setting, if $A\boldsymbol{x} \leq \boldsymbol{b}$ has a solution for a given $\boldsymbol{b} \in \mathbb{Z}^{\ell}$ then there are $D \in \mathbb{Q}^{n \times \ell}$ and $\boldsymbol{d} \in \mathbb{Q}^n$, both of bit length polynomial in the bit length of A, such that $\boldsymbol{x} = D\boldsymbol{b} + \boldsymbol{d}$ is integral and also a solution. Observe that there is only an exponential number (in the bit length of A) of possible choices for D and $\boldsymbol{d}$. Eliminating a block of variables $\boldsymbol{x}$ from a system of linear inequalities thus becomes easy: we have that $A\boldsymbol{x} \leq B\boldsymbol{y} + \boldsymbol{c}$ is equivalent to the disjunction of systems of the form $A(D(B\boldsymbol{y} + \boldsymbol{c}) + \boldsymbol{d}) \leq B\boldsymbol{y} + \boldsymbol{c}$ for all D and $\boldsymbol{d}$ of bit length polynomial in A. Using standard arguments, this approach can then be turned into a quantifier elimination procedure that eliminates a block of existentially quantified variables in exponential time.

2 Preliminaries

Throughout this paper, all vectors $\boldsymbol{z}$ are treated as column vectors unless mentioned otherwise. For a vector $\boldsymbol{x} \in \mathbb{Q}^n$, let $\|\boldsymbol{x}\|_\infty$ be the maximal absolute value of all components of $\boldsymbol{x}$. Moreover, let $\|\boldsymbol{x}\|_{\mathsf{frac}}$ be the maximal absolute value of all *numerators* and *denominators* of components in $\boldsymbol{x}$. The latter is important for representations: Note that a vector $\boldsymbol{x} \in \mathbb{Q}^n$ with $\|\boldsymbol{x}\|_{\mathsf{frac}} \leq m$ can be represented using $O(n \log m)$ bits. We use analogous notations $\|A\|_\infty$ and $\|A\|_{\mathsf{frac}}$ for matrices A. We will sometimes refer to the *Hadamard inequality* [19], which implies that for a square matrix $A \in \mathbb{Z}^{n \times n}$, we have $|\det(A)| \leq n^{n/2} \cdot \|A\|_\infty^n$. In particular, the determinant of A is at most exponential in the maximal absolute value of entries of A.

Presburger arithmetic. *Presburger arithmetic* (PA) is the first-order theory of the structure $\langle \mathbb{Z}; +, <, 0, 1\rangle$. In order to enable quantifier elimination, we have to permit modulo constraints. Thus technically, we are working with the structure $\langle \mathbb{Z}; +, <, (\equiv_m)_{m\in\mathbb{Z}}, 0, 1\rangle$, where $a \equiv_m b$ stands for $a \equiv b \bmod m$. In our syntax, we allow atomic formulas of the forms $a_1x_1 + \cdots + a_nx_n \leq b$ (called *linear inequalities*) or $a_1x_1 + \cdots + a_nx_n \equiv b \bmod m$ (called *modulo* or *divisibility constraints*), where $x_1, \ldots, x_n$ are variables and $a_1, \ldots, a_n, b, m \in \mathbb{Z}$ are constants encoded in binary. A formula is *quantifier-free* if it contains no quantifiers or, equivalently, is a Boolean combination of atomic formulas. Notice that conjunctions of linear inequalities can be written as systems of linear inequalities $A\boldsymbol{x} \leq \boldsymbol{b}$.

The *size* of a PA Formula φ, denoted $|\varphi|$, is the number of letters used to write it down, where we assume all constants to be encoded in binary. (Sometimes, we say that a formula obeys a size bound even if constants are encoded in unary; but this will be stated explicitly).

Fixed quantifier alternation fragments. The Σ_k fragment of PA consists of formulas of the form $\exists \boldsymbol{u_1} \forall \boldsymbol{u_2} \ldots Q_k \boldsymbol{u_k} \colon \varphi(\boldsymbol{u_1}, \boldsymbol{u_2}, \ldots, \boldsymbol{u_k}, \boldsymbol{z})$ where $\boldsymbol{u_i}$ is a vector of quantified variables, $\boldsymbol{z}$ is a vector of free variables, $\varphi(\boldsymbol{u_1}, \boldsymbol{u_2}, \ldots, \boldsymbol{u_k}, \boldsymbol{z})$ is a quantifier free PA formula, and Q_k denotes $\forall$ or $\exists$ depending on whether k is even or odd respectively. Similarly, the Π_k fragment of PA consists of formulas of the form $\forall \boldsymbol{u_1} \exists \boldsymbol{u_2} \ldots Q_k \boldsymbol{u_k} \colon \varphi(\boldsymbol{u_1}, \boldsymbol{u_2}, \ldots, \boldsymbol{u_k}, \boldsymbol{z})$ where Q_k denotes $\forall$ or $\exists$ depending on whether k is odd or even respectively.

Bounded existential Presburger arithmetic. In addition to our quantifier elimination result, we shall prove a somewhat stronger version, which states that one can compute a compact representation of a quantifier-free formula in polynomial time. As compact representations, we introduce a syntactic variant of existential Presburger arithmetic, which we call *bounded existential Presburger arithmetic*, short $\exists^{\leq}$PA. Essentially, $\exists^{\leq}$PA requires all quantifiers to be restricted to bounded intervals, but also permits polynomials over the quantified variables. Using standard methods, one can translate every formula in $\exists^{\leq}$PA in polynomial time into an $\exists$PA formula. However, the converse is not obvious, and our main results states that this is possible. Syntactically, an $\exists^{\leq}$PA formula over free variables $y_1, \ldots, y_m$ is of the form

$$\exists^{\leq k_1} x_1 \cdots \exists^{\leq k_n} x_n \colon \varphi,$$

where $x_1, \ldots, x_n$ are variables, each $k_i \in \mathbb{N}$ is a number given in binary, and φ is a quantifier-free formula where every atom is of one of the forms:

$$\sum_{i=1}^{m} p_i y_i \leq q \quad \text{or} \quad \sum_{i=1}^{m} p_i y_i \equiv r \bmod q, \tag{1}$$

where $p_1, \ldots, p_m, q, r \in \mathbb{Z}[x_1, \ldots, x_n]$ are polynomials over the variables $x_1, \ldots, x_n$. Thus, where $\exists$PA allows constant integral coefficients, $\exists^{\leq}$PA allows polynomials from $\mathbb{Z}[x_1, \ldots, x_n]$. The quantifiers $\exists^{\leq k_i} x_i$ are interpreted as "there exists $x_i \in \mathbb{Z}$ with $|x_i| \leq k_i$".

▶ Remark 2.1. Now indeed, a $\exists^{\leq}$PA formula can be converted in polynomial time into an $\exists$PA formula: The bounded quantification is clearly expressible in $\exists$PA. The terms p_iy_i and q in (1) (recall p_i and q are polynomials are from $\mathbb{Z}[x_1, \ldots, x_n]$) are also expressible, because multiplication with exponentially bounded variables can be expressed using polynomial-size $\exists$PA formulas. This is because given a polynomial $p \in \mathbb{Z}[x_1, \ldots, x_n]$ and a variable y, we can construct a polynomial-size existential formula $\pi(x_1, \ldots, x_n, y, z)$ expressing $z = p(x_1, \ldots, x_n) \cdot y \wedge |x_1| \leq k_1 \wedge \cdots \wedge |x_n| \leq k_n$. This, in turn, follows from the fact that given ℓ in unary, we can construct an existential formula $\mu_\ell(x, y, z)$, of size linear in ℓ, expressing $z = x \cdot y \wedge |x| \leq 2^\ell$ (see [17, Sec. 3.1] or [18, p. 7]). Thus, we can construct π in $\exists$PA by introducing a variable for each subterm of p (which can clearly all be bounded exponentially).

Making $\exists^{\leq}$PA formulas quantifier-free. Moreover, a $\exists^{\leq}$PA formula can easily be converted (in exponential time) into an exponential-size quantifier-free formula: Just take an exponential disjunction over all assignments of the existentially bounded variables $x_1, \ldots, x_n$ and replace the variables by their values in all the atoms. Thus, $\exists^{\leq}$PA formulas can be regarded as compact representations of quantifier-free formulas.

3 Main results

Here, we state and discuss implications of the main result of this paper:

▶ **Theorem 3.1.** *Given a formula of* $\exists$PA*, we can construct in polynomial time an equivalent formula in* $\exists^{\leq}$PA*.*

From Theorem 3.1, we can deduce the following, since by the remark Remark 2.1 in Section 2, one can easily convert a $\exists^{\leq}$PA formula into an exponential-sized quantifier-free formula.

▶ **Corollary 3.2.** *Given a formula φ in existential Presburger arithmetic, we can compute in exponential time an equivalent quantifier-free formula ψ of size exponential in φ. Moreover, all constants in ψ are encoded in unary.*

In Section 6, we will see that an exponential blowup cannot be avoided when eliminating a block of existential quantifiers, even if we allow constants to be encoded in binary in the quantifier-free formula.

There are several applications of Theorem 3.1 and Corollary 3.2. The most obvious type of applications are those, where, for every problem[1] that is in NP (resp. coNP) for quantifier-free formulas, the same problem belongs to NEXP (resp. coNEXP) for existential formulas. Oftentimes, this yields optimal complexity. We mention some examples.

A direct consequence of Corollary 3.2 (and the NP membership of the quantifier free fragment of PA) is the following.

▶ **Corollary 3.3.** *The Σ_2-fragment of Presburger arithmetic belongs to* NEXP*.*

The NEXP upper bound is known and was shown by Haase [17, Thm. 1]. In fact, the Σ_2-fragment is known to be NEXP-complete: An NEXP lower bound was shown much earlier by Grädel [14], already for the $\exists\forall^*$-fragment.

Ramsey quantifiers. In fact, combining Corollary 3.2 with the results from [2], we can strengthen Corollary 3.3. The *Ramsey quantifier* $\exists^{\mathsf{ram}}$ states the existence of infinite (directed) cliques. More precisely, if $\varphi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ is a Presburger formula where $\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors of n variables each, then $\exists^{\mathsf{ram}}(\boldsymbol{x}, \boldsymbol{y})\colon \varphi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ is satisfied for $\boldsymbol{z}$ if and only if there exists an infinite sequence $\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots \in \mathbb{Z}^n$ of pairwise distinct vectors with $\varphi(\boldsymbol{a}_i, \boldsymbol{a}_j, \boldsymbol{z})$ for every $i < j$. As mentioned in [2], Ramsey quantifiers can be applied to deciding liveness properties, deciding monadic decomposability (see below), and deciding whether a formula defines a well-quasi-ordering (see below).

In [2, Thm. 5.1], it is shown that if $\varphi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ is an $\exists$PA formula, then one can compute in polynomial time an $\exists$PA formula $\varphi'(\boldsymbol{z})$ equivalent to $\exists^{\mathsf{ram}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})\colon \varphi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$.

[1] To be precise: Every *semantic* problem, meaning one that only depends on the set defined by the input formula.

▶ **Corollary 3.4.** *Given a Σ_2-formula $\varphi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$, we can construct an exponential-size* ∃PA *formula equivalent to $\exists^{\mathsf{ram}}(\boldsymbol{x}, \boldsymbol{y})\colon \varphi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$. In particular, deciding the truth of $\exists^{\mathsf{ram}}(\boldsymbol{x}, \boldsymbol{y})\colon \psi$ for Σ_2-formulas $\psi(\boldsymbol{x}, \boldsymbol{y})$ is* NEXP*-complete.*

Indeed, Corollary 3.2 lets us convert $\varphi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ into an exponential-size existential formula φ', so that we can apply the above result of [2] to the formula $\exists^{\mathsf{ram}}(\boldsymbol{x}, \boldsymbol{y})\colon \varphi'(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$, which results in an equivalent exponential ∃PA formula. The NEXP lower bound in the second statement follows from NEXP-hardness of the Σ_2-fragment and the fact that for a given Σ_2-formula χ without free variables, the statement $\exists^{\mathsf{ram}}(\boldsymbol{x}, \boldsymbol{y})\colon \chi \wedge \boldsymbol{x} < \boldsymbol{y}$ is equivalent to χ.

Detecting WQOs. A *well-quasi-ordering* (WQO) is a reflexive and transitive ordering $(X, \leq)$ such that for every sequence $x_1, x_2, \ldots \in X$, there are $i < j$ with $x_i \leq x_j$. Well-quasi-orderings are of paramount importance in the widely applied theory of well-structured transition systems [7, 1, 10]. The problem of deciding whether a given Presburger formula $\varphi(\boldsymbol{x}, \boldsymbol{y})$ defines a WQO was recently raised by Finkel and Gupta [8], with the hope of establishing automatically that certain systems are well-structured. As observed in [9, Prop. 12], this problem reduces to evaluating Ramsey quantifiers, which is decidable by [28]. Based on an NP algorithm for Ramsey quantifiers, it is shown in [2, Sec. 8.3] that given a quantifier-free formula $\varphi(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors of n variables each, it is coNP-complete whether the relation $R \subseteq \mathbb{Z}^n \times \mathbb{Z}^n$ defined by φ is a WQO. Our results allow us to settle the complexity for existential formulas:

▶ **Corollary 3.5.** *Given an* ∃PA *formula φ, it is* coNEXP*-complete to decide whether φ defines a WQO.*

The upper bound follows directly from Corollary 3.2 and the fact that it is coNP-complete to decide whether a given quantifier-free formula defines a well-quasi-ordering [2, Sec. 8.3]. This yields a coNEXP procedure overall. It should be noted that Corollary 3.5 can also be deduced from Corollary 3.4 (using the same idea as in [2, Sec. 8.3]). However, we find it instructive to demonstrate how quantifier elimination permits a direct transfer of the coNP algorithm as a black box. We show the coNEXP lower bound in Section 5.

Monadic decomposability. A Presburger formula is *monadic* if each of its atoms contains at most one variable. Moreover, we say that a Presburger formula φ is *monadically decomposable* if φ is equivalent to a monadic Presburger formula. Motivated by the role monadic formulas play in constraint databases [15, 21], Veanes, Bjørner, and Nachmanson, and Bereg recently raised the question of how to decide whether a given formula is monadically decomposable [30]. For Presburger arithmetic, decidability follows from [13, p. 1048] and for quantifier-free formulas, monadic decomposability was shown coNP-complete in [20, Thm. 1] (in [2, Cor. 8.1], the coNP upper bound is shown via Ramsey quantifiers). Corollary 3.2 allows us to settle the case of ∃PA formulas.

▶ **Corollary 3.6.** *Monadic decomposability of* ∃PA *formulas is* coNEXP*-complete.*

This is because given an ∃PA formula, we can compute an exponential-sized quantifier-free formula and apply the existing coNP procedure, yielding a coNEXP upper bound overall. Again, the coNEXP upper bound could also be deduced from Corollary 3.4 (but this proof shows again how to transfer algorithms using quantifier elimination). The coNEXP lower bound follows the same idea as the coNP lower bound in [2], see Section 5.

NP upper bounds. In addition to new NEXP and coNEXP upper bounds, Theorem 3.1 can also be used to obtain NP upper bounds. Suppose we have a predicate $\mathfrak{p}$ on sets of integral vectors. That is, for each $S \subseteq \mathbb{Z}^m$ for some $m \in \mathbb{N}$, either $\mathfrak{p}(S)$ is true or not. We call this predicate *admissible* if for any $m \in \mathbb{N}$, $S_1, S_2 \subseteq \mathbb{Z}^m$, we have that $\mathfrak{p}(S_1 \cup S_2)$ implies $\mathfrak{p}(S_1)$ or $\mathfrak{p}(S_2)$. Let us see some examples:

- **(i)** The predicate $\mathfrak{p}$ with $\mathfrak{p}(S)$ if and only if $S \neq \emptyset$.
- **(ii)** The predicate $\mathfrak{p}$ with $\mathfrak{p}(S)$ if and only if S is infinite.
- **(iii)** The predicate $\mathfrak{p}$ with $\mathfrak{p}(S)$ if and only if $S \subseteq \mathbb{Z}$ and S contains a power of 2.
- **(iv)** The predicate $\mathfrak{p}$ with $\mathfrak{p}(S)$ if and only if $S \subseteq \mathbb{Z}^{2k}$ and viewed as a relation $S \subseteq \mathbb{Z}^k \times \mathbb{Z}^k$, S has an infinite clique.
- **(v)** The predicate $\mathfrak{p}$ with $\mathfrak{p}(S)$ if and only if $S \subseteq \mathbb{Z}$ and S contains infinitely many primes.
- **(vi)** The predicate $\mathfrak{p}$ with $\mathfrak{p}(S)$ if and only if $S \subseteq \mathbb{Z}^2$ and S contains a pair $(x, 2^x)$.

For each such predicate, we consider the problem $\mathfrak{p}(\exists\text{PA})$:

Input An $\exists$PA formula φ with m free variables for some $m \in \mathbb{N}$.

Question Does $\mathfrak{p}(S)$ hold, where $S \subseteq \mathbb{Z}^m$ is the set defined by φ?

Moreover, $\mathfrak{p}(\text{QF})$ is the restriction of the problem where the input formula φ is quantifier-free.

For several of the examples above, it is known that $\mathfrak{p}(\exists\text{PA})$ is in NP: For (i) and (ii), these are standard facts, and for (iii), this follows from NP-completeness of existential Büchi arithmetic [16, Thm. 1]. For (iv), this follows from the fact that Ramsey quantifiers can be evaluated in NP [2, Thm 5.1]. Our results imply that for proving NP upper bounds, we may always assume a quantifier-free input formula. This is perhaps surprising, because one might expect that for non-linear predicates, it is difficult to bound the quantified variables.

▶ **Corollary 3.7.** *For every admissible predicate $\mathfrak{p}$, the problem $\mathfrak{p}(\exists\text{PA})$ is in NP if and only if $\mathfrak{p}(\text{QF})$ is in NP.*

Here, the "only if" direction is trivial, and the "if" direction follows from Theorem 3.1. This is because Theorem 3.1 allows us to assume that φ is given as a $\exists^{\leq}$PA formula $\exists^{\leq k_1} x_1 \cdots \exists^{\leq k_n} x_n \colon \psi(x_1, \ldots, x_n, y_1, \ldots, y_m)$. Moreover, admissibility of $\mathfrak{p}$ implies that $\mathfrak{p}$ is satisfied for φ if and only if there exists an assignment $(a_1, \ldots, a_n)$ for the bounded variables such that the quantifier-free formula $\psi(a_1, \ldots, a_n, y_1, \ldots, y_m)$ satisfies $\mathfrak{p}$. Thus, we can guess the assignment (which occupies polynomially many bits) and run the NP algorithm for quantifier-free formulas.

4 Quantifier elimination

In this section, we prove Theorem 3.1. The following is our main geometric ingredient.

▶ **Proposition 4.1.** *Let $A \in \mathbb{Z}^{\ell \times n}$ and $\boldsymbol{b} \in \mathbb{Z}^\ell$, and let Δ be an upper bound on all absolute values of the subdeterminants of A. If the system $A\boldsymbol{x} \leq \boldsymbol{b}$ has an integral solution, then it has an integral solution of the form $D\boldsymbol{b} + \boldsymbol{d}$, where $D \in \mathbb{Q}^{n \times \ell}$ and $\boldsymbol{d} \in \mathbb{Q}^n$ with $\|D\|_{\mathsf{frac}} \leq \Delta$ and $\|\boldsymbol{d}\|_{\mathsf{frac}} \leq n\Delta^2$.*

Before we prove Proposition 4.1, let us see how it implies Theorem 3.1 and Corollary 3.2.

Proof of Corollary 3.2. While Corollary 3.2 follows from Theorem 3.1, it follows very directly from Proposition 4.1 and the proof is a good warm-up for the proof of Theorem 3.1. Therefore, we first derive Corollary 3.2 from Proposition 4.1. Suppose we are given a Presburger formula $\exists \boldsymbol{x} \colon \varphi(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, \ldots, y_m)$ are variables and φ is quantifier-free.

It is well-known that divisibility constraints can be eliminated in favor of existentially quantified variables, since $a \equiv b \bmod m$ if and only if $\exists x\colon a - b = mx$. Thus, we may assume that φ contains no divisibility constraints. Then, by moving all negations inwards and using the standard equivalence $\neg(r \leq t) \iff t + 1 \leq r$, we may assume that φ is a positive Boolean combination of atoms $\boldsymbol{a}^\top \boldsymbol{x} \leq \boldsymbol{b}^\top \boldsymbol{y} + c$, where $\boldsymbol{a} \in \mathbb{Z}^n$, $\boldsymbol{b} \in \mathbb{Z}^m$, and $c \in \mathbb{Z}$.

By bringing φ into DNF, we can write it as a disjunction of exponentially many systems of inequalities of the form $A\boldsymbol{x} \leq B\boldsymbol{y} + \boldsymbol{c}$, where $A \in \mathbb{Z}^{\ell \times n}$, $B \in \mathbb{Z}^{\ell \times m}$, and $\boldsymbol{c} \in \mathbb{Z}^\ell$. Thus, it suffices to construct a quantifier-free formula for $\exists \boldsymbol{x}\colon A\boldsymbol{x} \leq B\boldsymbol{y} + \boldsymbol{c}$. Let Δ be an upper bound for all absolute values of subdeterminants of A. Since the transformation into DNF does not change the appearing constants, we have that $\Delta \leq n^{n/2}\|A\|_\infty^n$ is at most exponential in the size of the input formula.

According to Proposition 4.1, a vector $\boldsymbol{x}$ with $\varphi(\boldsymbol{x}, \boldsymbol{y})$ exists if and only if there exists a matrix $D \in \mathbb{Q}^{n \times \ell}$ and $\boldsymbol{d} \in \mathbb{Q}^n$ with $\|D\|_{\mathsf{frac}} \leq \Delta$ and $\|\boldsymbol{d}\|_{\mathsf{frac}} \leq n\Delta^2$ such that (i) substituting $D(B\boldsymbol{y} + \boldsymbol{c}) + \boldsymbol{d}$ for $\boldsymbol{x}$ satisfies $A\boldsymbol{x} \leq B\boldsymbol{y} + \boldsymbol{c}$ and also (ii) the vector $D(B\boldsymbol{y} + \boldsymbol{c}) + \boldsymbol{d}$ is integral. Therefore, the formula $\exists \boldsymbol{x}\colon A\boldsymbol{x} \leq B\boldsymbol{y} + \boldsymbol{c}$ is equivalent to

$$\bigvee_{(D,\boldsymbol{d}) \in P} A(D(B\boldsymbol{y} + \boldsymbol{c}) + \boldsymbol{d}) \leq B\boldsymbol{y} + \boldsymbol{c} \ \wedge \ D(B\boldsymbol{y} + \boldsymbol{c}) + \boldsymbol{d} \in \mathbb{Z}^n$$

where P is the set of all pairs $(D, \boldsymbol{d})$ with $D \in \mathbb{Q}^{n \times \ell}$, $\boldsymbol{d} \in \mathbb{Q}^n$, $\|D\|_{\mathsf{frac}} \leq \Delta$, and $\|\boldsymbol{d}\|_{\mathsf{frac}} \leq n\Delta^2$. Clearly, P contains at most exponentially many elements. Moreover, note that the condition $D(B\boldsymbol{y} + \boldsymbol{c}) + \boldsymbol{d} \in \mathbb{Z}^n$ is a set of n modulo constraints. ◀

Proof of Theorem 3.1. The proof of Theorem 3.1 is similar to the above construction – we just need to circumvent the exponential conversion into DNF. We proceed as follows.

As above, we are given a Presburger formula $\exists \boldsymbol{x}\colon \varphi(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, \ldots, y_m)$ are variables and φ is quantifier-free. Moreover, we may assume that φ contains no divisibility constraints and is a positive Boolean combination of atoms $\boldsymbol{a}^\top \boldsymbol{x} \leq \boldsymbol{b}^\top \boldsymbol{y} + c$, where $\boldsymbol{a} \in \mathbb{Z}^n$, $\boldsymbol{b} \in \mathbb{Z}^m$, and $c \in \mathbb{Z}$.

Let $\boldsymbol{a}_i^\top \boldsymbol{x} \leq \boldsymbol{b}_i^\top \boldsymbol{y} + c_i$ for $i = 1, \ldots, \ell$ be the set of all atoms occurring in φ and let $A \in \mathbb{Z}^{\ell \times n}$ be the matrix with rows $\boldsymbol{a}_i^\top$ and $B \in \mathbb{Z}^{\ell \times m}$ be the matrix of rows $\boldsymbol{b}_i^\top$, and let $\boldsymbol{c} \in \mathbb{Z}^\ell$ be the (column) vector with entries $c_1, \ldots, c_\ell$. Thus, our formula φ consists of ℓ atoms, each of which is a row in the system of linear inequalities $A\boldsymbol{x} \leq B\boldsymbol{y} + \boldsymbol{c}$. Let φ' be the formula obtained from φ by replacing the atom $\boldsymbol{a}_i^\top \boldsymbol{x} \leq \boldsymbol{b}_i^\top \boldsymbol{y} + c_i$ by $z_i = 1$, where z_i, $i \in \{1, \ldots, \ell\}$, is a fresh variable for each of the ℓ atoms. Now let Δ be an upper bound on all absolute values of the subdeterminants of A. Then $\Delta \leq n^{n/2}\|A\|_\infty^n$ is at most exponential in the size of the input formula. Consider the formula

$$\begin{aligned} &\exists z_1, \ldots, z_\ell \in \{0,1\}\colon \quad \exists D \in \mathbb{Q}^{n \times \ell}, \|D\|_{\mathsf{frac}} \leq \Delta\colon \\ &\qquad \exists \boldsymbol{d} \in \mathbb{Q}^n,\ \|\boldsymbol{d}\|_{\mathsf{frac}} \leq n\Delta^2\colon \quad \varphi' \ \wedge \ D(B\boldsymbol{y} + \boldsymbol{c}) + \boldsymbol{d} \in \mathbb{Z}^n \ \wedge \ \bigwedge_{i=1}^{\ell} (z_i = 1 \to \psi_i)\,, \end{aligned} \tag{2}$$

where ψ_i is the formula $\boldsymbol{a}_i^\top (D(B\boldsymbol{y} + \boldsymbol{c}) + \boldsymbol{d}) \leq \boldsymbol{b}_i^\top \boldsymbol{y} + c_i$. Note that (2) is expressible in $\exists^{\leq}$PA: We introduce (i) one variable for each z_i, (ii) two variables for each entry of D (one for the numerator, and one for the denominator), and (iii) two variables for each entry of $\boldsymbol{d}$.

Each of the n divisibility constraints of $D(B\boldsymbol{y} + \boldsymbol{c}) + \boldsymbol{d} \in \mathbb{Z}^n$ and each of the atoms ψ_i can be written in the forms (1). To see this, let $u_1, \ldots, u_k$ be the bounded variables used for the numerators or denominators in D and $\boldsymbol{d}$. Observe that the vector $B\boldsymbol{y}$ is a linear combination of $\boldsymbol{y}$ with integer coefficients. The matrix D and the vector $\boldsymbol{d}$ consist

of quotients of bounded variables, hence rational functions in $\mathbb{Z}[u_1, \ldots, u_k]$. Thus, the vector $D(B\boldsymbol{y} + \boldsymbol{c}) + \boldsymbol{d}$ has in each entry an expression $s + \sum_{i=1}^{m} r_i y_i$, where $r_1, \ldots, r_m, s \in \mathbb{Z}[u_1, \ldots, u_k]$. Hence, by multiplying with the product of all denominators, we can write each inequality $\boldsymbol{a}_i^\top (D(B\boldsymbol{y} + \boldsymbol{c}) + \boldsymbol{d}) \leq \boldsymbol{b}_i^\top \boldsymbol{y} + c_i$ in the form of (1). Moreover, for the requirement $D(B\boldsymbol{y} + \boldsymbol{c}) + \boldsymbol{d} \in \mathbb{Z}^n$, we can write each row of $D(B\boldsymbol{y} + \boldsymbol{c}) + \boldsymbol{d}$ as a quotient $\frac{1}{q}(r + \sum_{i=1}^{m} p_i y_i)$, where $q, r, p_1, \ldots, p_m \in \mathbb{Z}[u_1, \ldots, u_k]$, so that membership in $\mathbb{Z}$ is equivalent to $\sum_{i=1}^{m} p_i y_i \equiv -r \pmod{q}$.

Let us argue why (2) is equivalent to $\exists \boldsymbol{x} \colon \varphi(\boldsymbol{x}, \boldsymbol{y})$. Clearly, if (2) is satisfied, then $\boldsymbol{z} = (z_1, \ldots, z_\ell)$ yields a set of atoms that, if satisfied, makes φ true. Moreover, the vector $D(B\boldsymbol{y} + \boldsymbol{c}) + \boldsymbol{d}$ is an integer vector that satisfies all the atoms specified by $\boldsymbol{z}$.

Conversely, suppose $\varphi(\boldsymbol{x}, \boldsymbol{y})$ holds for some $\boldsymbol{x} \in \mathbb{Z}^n$ and $\boldsymbol{y} \in \mathbb{Z}^m$. First, we set exactly those z_i to 1 for which the i-th atom in φ is satisfied by $\boldsymbol{x}, \boldsymbol{y}$. Recall that each row of A (and each row of B, and of $\boldsymbol{c}$) corresponds to an atom in φ. Let A' be the matrix obtained from A by selecting those rows that correspond to atoms that are satisfied by our $\boldsymbol{x}$ and $\boldsymbol{y}$. Define B' similarly from B, and $\boldsymbol{c}'$ from $\boldsymbol{c}$. Then we have $A'\boldsymbol{x} \leq B'\boldsymbol{y} + \boldsymbol{c}'$. Now Proposition 4.1 yields a matrix D' and a vector $\boldsymbol{d}$ (each with n rows) with $A'(D'(B'\boldsymbol{y} + \boldsymbol{c}') + \boldsymbol{d}) \leq B'\boldsymbol{y} + \boldsymbol{c}'$. Now the set of rows of $B'\boldsymbol{y} + \boldsymbol{c}'$ is a subset of the rows of $B\boldsymbol{y} + \boldsymbol{c}$, so by inserting zero-columns into D', we can construct a matrix D with $D(B\boldsymbol{y} + \boldsymbol{c}) = D'(B'\boldsymbol{y} + \boldsymbol{c}')$. Hence, we have $A'(D(B\boldsymbol{y} + \boldsymbol{c}) + \boldsymbol{d}) \leq B'\boldsymbol{y} + \boldsymbol{c}'$. The latter means exactly that ψ_i is satisfied for every i with $z_i = 1$. Thus, this choice of $z_1, \ldots, z_\ell$, D, and $\boldsymbol{d}$ satisfies (2). ◀

4.1 Constructing solutions as affine transformations

4.1.1 Convex geometry

Before we start with the proof of Proposition 4.1, we recall some standard definitions from convex geometry from Schrijver's book [29]. Below, we let $\mathbb{R}_+ = \{r \in \mathbb{R} \mid r \geq 0\}$. A *polyhedron* is a set $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$, where A is an $\ell \times n$ integer matrix and $\boldsymbol{b} \in \mathbb{Z}^\ell$. Let $C \subseteq \mathbb{R}^\ell$, then C is a *convex cone* if $\lambda\boldsymbol{x} + \mu\boldsymbol{y} \in C$ for all $\boldsymbol{x}, \boldsymbol{y} \in C$ and $\lambda, \mu \in \mathbb{R}_+$. Given a set $X \subseteq \mathbb{R}^\ell$,

$$\operatorname{cone}(X) = \{\lambda_1 \boldsymbol{x}_1 + \cdots + \lambda_t \boldsymbol{x}_t \mid t \geq 0,\ \boldsymbol{x}_1, \ldots, \boldsymbol{x}_t \in X,\ \lambda_1, \ldots, \lambda_t \in \mathbb{R}_+\}\,.$$

The *convex hull* of a set $X \subseteq \mathbb{R}^\ell$ is the smallest convex set containing that set, i.e.,

$$\text{conv. hull}(X) = \{\lambda_1 \boldsymbol{x}_1 + \cdots + \lambda_t \boldsymbol{x}_t \mid t \geq 1,\ \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_t \in X, \\ \lambda_1, \ldots, \lambda_t \in \mathbb{R}_+,\ \lambda_1 + \cdots + \lambda_t = 1\}\,.$$

Next, we recall some terminology concerning the structure of polyhedra. The *characteristic cone* of a polyhedron $P = \{\boldsymbol{x} \mid A\boldsymbol{x} \leq \boldsymbol{b}\} \subseteq \mathbb{R}^n$ is the set $\text{char. cone}(P) \coloneqq \{\boldsymbol{y} \in \mathbb{R}^n \mid A\boldsymbol{y} \leq 0\}$. The *lineality space* of polyhedron P is the set $\text{lin. space}(P) \coloneqq \{\boldsymbol{y} \in \mathbb{R}^n \mid A\boldsymbol{y} = 0\}$.

▶ **Definition 4.2** (Faces). *Given a polyhedron $P \subseteq \mathbb{R}^n$, $F \subseteq P$ is a* face *of P if and only if F is non-empty and*

$$F = \{\boldsymbol{x} \in P \mid A'\boldsymbol{x} = \boldsymbol{b}'\}$$

for some subsystem $A'\boldsymbol{x} \leq \boldsymbol{b}'$ of $A\boldsymbol{x} \leq \boldsymbol{b}$. We call $F \subseteq P$ a proper face *of P if $F \neq \emptyset$ and $F \neq P$.*

It follows that P has only finitely many faces. A *minimal face* of P is a face not containing any other face. We have the following characterization of minimal faces [29, Thm 8.4],

▶ **Proposition 4.3.** *A set F is a minimal face of a polyhedron $P \subseteq \mathbb{R}^n$ if and only if $\emptyset \neq F \subseteq P$ and*

$$F = \{\boldsymbol{x} \in \mathbb{R}^n \mid A'\boldsymbol{x} = \boldsymbol{b}'\}$$

for some subsystem $A'\boldsymbol{x} \leq \boldsymbol{b}'$ of $A\boldsymbol{x} \leq \boldsymbol{b}$, such that the matrix A' has the same rank as A.

The following is shown in [29, Sec. 8.8]:

▶ **Proposition 4.4.** *Let C be the cone $\{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{0}\}$. There is a finite collection $G_1, G_2 \ldots, G_s$ of subsets, which are of the form $G_i = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a}_i^\top \boldsymbol{x} \leq 0,\ A'\boldsymbol{x} = \boldsymbol{0}\}$, where $\begin{bmatrix} A' \\ \boldsymbol{a}_i^\top \end{bmatrix}$ is a subset of the rows of A, such that the following holds. If we choose for each $i = 1, \ldots, s$ a vector $\boldsymbol{y}_i$ from $G_i \setminus \text{lin. space}(C)$ and choose $\boldsymbol{z}_0, \ldots, \boldsymbol{z}_t$ in $\text{lin. space}(C)$ such that $\text{lin. space}(C) = \text{cone}(\boldsymbol{z}_0, \ldots, \boldsymbol{z}_t)$, then*

$$C = \text{cone}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_s, \boldsymbol{z}_0, \ldots, \boldsymbol{z}_t).$$

Here, the sets G_i are also called minimal proper faces (but Proposition 4.4 is not a characterization of those).

4.1.2 Proof of Proposition 4.1

We now prove Proposition 4.1. For the remainder of the section, let $A \in \mathbb{Z}^{\ell \times n}$ and $\boldsymbol{b} \in \mathbb{Z}^\ell$. Moreover, let Δ be an upper bound on all absolute values of the sub-determinants of A. Our first step is a simple application of standard facts about polyhedra.

▶ **Lemma 4.5.** *If the system $A\boldsymbol{x} \leq \boldsymbol{b}$ has a solution in $\mathbb{Q}^n$, then it has one of the form $\frac{1}{a}E\boldsymbol{b}$, where $E \in \mathbb{Z}^{n \times \ell}$, $a \in \mathbb{Z} \setminus \{0\}$, $|a| \leq \Delta$, and $\|E\|_\infty \leq \Delta$.*

Proof. It is well-known that if $A\boldsymbol{x} \leq \boldsymbol{b}$ has a rational solution, then there is a solution inside a minimal face of the polyhedron $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$ defined by the system of linear inequalities $A\boldsymbol{x} \leq \boldsymbol{b}$ [29, Thm. 8.5]. Recall that a *minimal face* is a non-empty subset $F \subseteq P$ of the form

$$F = \{\boldsymbol{x} \in \mathbb{R}^n \mid A'\boldsymbol{x} = \boldsymbol{b}'\}, \tag{3}$$

where $A'\boldsymbol{x} \leq \boldsymbol{b}'$ is a subset of the inequalities in $A\boldsymbol{x} \leq \boldsymbol{b}$ such that the matrix A' has the same rank as A (see Proposition 4.3 or [29, Thm. 8.4]). Suppose F is a non-empty minimal face and satisfies (3). Here, we may assume that the rows of A' are linearly independent (otherwise, we can remove redundant rows without changing F). This means, A' can be written as $A' = (B \quad C)$ such that B is invertible. Then the vector $\boldsymbol{x}^* := (B^{-1}\boldsymbol{b}' \quad \boldsymbol{0})^\top$ belongs to F. Since $F \subseteq P$, we know that $A\boldsymbol{x}^* \leq \boldsymbol{b}$. By Cramer's rule, the entry (j, i) of B^{-1} is $\frac{(-1)^{i+j}\det(B_{ij})}{\det(B)}$, where B_{ij} is the matrix obtained from B by removing the i-th row and j-th column. Note that $|\det(B_{ij})| \leq \Delta$ and $|\det(B)| \leq \Delta$. In particular, $\boldsymbol{x}^*$ can be written as $\frac{1}{a}E\boldsymbol{b}$, where $a = \det(B)$ and $\|E\|_\infty \leq \Delta$. ◀

We also employ the following well-known fact, which again uses standard arguments.

▶ **Lemma 4.6.** *There are integral vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_s$ with each component being at most Δ in absolute value, such that $\{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{0}\} = \text{cone}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_s)$.*

Proof. Let $C = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{0}\}$. The lemma follows from Proposition 4.4. First, it is a consequence of Cramer's rule that we can choose $\boldsymbol{z}_0, \ldots, \boldsymbol{z}_t$ as a basis of $\text{lin. space}(C) = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} = \boldsymbol{0}\}$ so that all $\boldsymbol{z}_0, \ldots, \boldsymbol{z}_t$ are integral and have absolute values at most Δ in all components. For example, see [29, Cor. 3.1c]. It remains to pick from each set

$$G_i \setminus \text{lin. space}(C) = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a}_i^\top \boldsymbol{x} < 0,\ A'\boldsymbol{x} = \boldsymbol{0}\}$$

an integral vector with all components bounded by Δ. For this, we can proceed similarly to Lemma 4.5. As a subset of rows of A, the matrix $B = \begin{bmatrix} A' \\ \boldsymbol{a}_i^\top \end{bmatrix}$ has rank at most n, and we may assume $\boldsymbol{a}_i \neq \boldsymbol{0}$ (otherwise $G_i \setminus \text{lin. space}(C)$ would be empty). Moreover, we may assume that the rows of B are linearly independent, as otherwise we can remove rows from A' without changing G_i. We can thus write $B = (E \quad F)$, where E is invertible. By Cramer's rule (see, e.g. [29, Sec. 3.2]), the j-th component of the vector $\boldsymbol{y} = E^{-1}(0, \ldots, 0, -1)$ can be written as $\frac{1}{\det(E)} \det(\tilde{E})$, where $\tilde{E}$ is obtained from E by replacing the j-th column by $(0, \ldots, 0, -1)$. This means, the vector $|\det(E)| \cdot \boldsymbol{y}$ has only integer components and all of them have absolute value at most Δ. Now let $\boldsymbol{y}^*$ be the vector obtained from $|\det(E)| \cdot \boldsymbol{y}$ adding as many 0's as F has columns. Then we have $B\boldsymbol{y}^* = E(|\det(E)| \cdot \boldsymbol{y}) = (0, \ldots, 0, -|\det(E)|)$ and thus $\boldsymbol{y}^* \in G_i \setminus \text{lin. space}(C)$. ◀

We also rely on the well-known theorem of Carathéodory [29, Cor. 7.1(i)].

▶ **Theorem 4.7** (Carathéodory's theorem). *If $X \subseteq \mathbb{R}^n$ is some subset and $\boldsymbol{x} \in \text{cone}(X)$, then there are linearly independent $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m \in X$ with $\boldsymbol{x} \in \text{cone}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)$.*

The following lemma is the key ingredient for proving Proposition 4.1. Its proof closely follows the ideas of [29, Thm. 17.2], which Schrijver attributes to Cook, Gerards, Schrijver, and Tardos [5]. The latter shows that for every rational $\boldsymbol{x}$ that maximizes an expression $\boldsymbol{c}^\top \boldsymbol{x}$ among the solutions of $A\boldsymbol{x} \leq \boldsymbol{b}$, there is a close-by integral vector that maximizes this expression among all integral vectors.

▶ **Lemma 4.8.** *Suppose the system $A\boldsymbol{x} \leq \boldsymbol{b}$ has an integral solution, and let $\boldsymbol{r} \in \mathbb{Q}^n$ be a rational solution. Then there is an integral solution $\boldsymbol{z}^* \in \mathbb{Z}^n$ with $\|\boldsymbol{z}^* - \boldsymbol{r}\|_\infty \leq n\Delta$.*

Proof. An illustration of the proof is given in Figure 1. Let $\boldsymbol{z}$ be an integral solution to $A\boldsymbol{x} \leq \boldsymbol{b}$. Split the equations $A\boldsymbol{x} \leq \boldsymbol{b}$ into $A_1\boldsymbol{x} \leq \boldsymbol{b}_1$ and $A_2\boldsymbol{x} \leq \boldsymbol{b}_2$ such that $A_1\boldsymbol{r} \leq A_1\boldsymbol{z}$ and $A_2\boldsymbol{r} \geq A_2\boldsymbol{z}$. In other words, we split A, $\boldsymbol{b}$ into two sets of rows, depending on in which coordinates $\boldsymbol{r}$ resp. $\boldsymbol{z}$ is larger. Now consider the cone $C = \{\boldsymbol{x} \in \mathbb{R}^n \mid A_1\boldsymbol{x} \geq \boldsymbol{0},\ A_2\boldsymbol{x} \leq \boldsymbol{0}\}$. Then, by the choice of A_1 and A_2, we have $\boldsymbol{z} - \boldsymbol{r} \in C$ and therefore

$$\boldsymbol{z} - \boldsymbol{r} = \lambda_1\boldsymbol{y}_1 + \cdots + \lambda_t\boldsymbol{y}_t,$$

where $\lambda_1, \ldots, \lambda_t \geq 0$ are real numbers and $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_t$ are some linearly independent vectors chosen from the set of integer vectors $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_s\}$ provided by Lemma 4.6 satisfying $C = \text{cone}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_s\}$. The choice of linearly independent vectors is possible due to Carathéodory's theorem. In particular, each $\boldsymbol{y}_i$ has maximal absolute value at most Δ and we have $t \leq n$.

Observe that for any $\mu_1, \ldots, \mu_t$ with $0 \leq \mu_i \leq \lambda_i$ for $i \in [1, t]$, the vector

$$\boldsymbol{r} + \mu_1\boldsymbol{y}_1 + \cdots + \mu_t\boldsymbol{y}_t$$

is still a solution to $A\boldsymbol{x} \leq \boldsymbol{b}$. Indeed, $A_1\boldsymbol{y}_i \geq \boldsymbol{0}$ and $A_2\boldsymbol{y}_i \leq \boldsymbol{0}$ implies

$$A_1(\boldsymbol{r} + \mu_1\boldsymbol{y}_1 + \cdots + \mu_t\boldsymbol{y}_t) \leq A_1\boldsymbol{z} \leq \boldsymbol{b}_1, \text{ and}$$
$$A_2(\boldsymbol{r} + \mu_1\boldsymbol{y}_1 + \cdots + \mu_t\boldsymbol{y}_t) \leq A_2\boldsymbol{r} \leq \boldsymbol{b}_2,$$

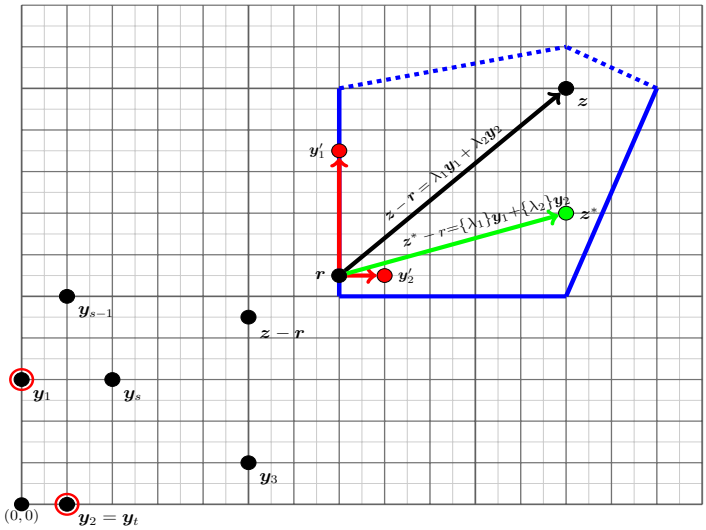


Figure 1 The main idea behind Lemma 4.8. The region enclosed by blue lines depicts the solution space of the given system of linear inequalities. As mentioned in the lemma, $\boldsymbol{r}$ and $\boldsymbol{z}$ are respectively the given rational and integral solutions. Due to Lemma 4.6, we know that C (containing $\boldsymbol{z} - \boldsymbol{r}$) can be obtained as a cone of integer vectors $\boldsymbol{y}_1, \dots \boldsymbol{y}_s$. Moreover, by Carathéodory's theorem, we know that there are t linearly independent ($t = 2$ in this case) vectors whose cone contains $\boldsymbol{z} - \boldsymbol{r}$. Intuitively, these vectors $(\boldsymbol{y}_1, \boldsymbol{y}_2)$ form a coordinate system for searching the required z^*. For $i \in \{1, 2\}$, $\boldsymbol{y}_i' = \boldsymbol{y}_i + \boldsymbol{r}$, $\{\lambda_i\} = \lambda_i - \lfloor \lambda_i \rfloor$.

and thus $A(\boldsymbol{r} + \mu_1 \boldsymbol{y}_1 + \cdots + \mu_t \boldsymbol{y}_t) \leq \boldsymbol{b}$. In particular, the vector

$$\boldsymbol{z}^* = \boldsymbol{r} + (\lambda_1 - \lfloor \lambda_1 \rfloor)\boldsymbol{y}_1 + \cdots + (\lambda_t - \lfloor \lambda_t \rfloor)\boldsymbol{y}_t$$

is a solution to $A\boldsymbol{x} \leq \boldsymbol{b}$. Moreover, $\boldsymbol{z}^*$ is obtained from $\boldsymbol{z}$ by subtracting integer multiples of the integer vectors $\boldsymbol{y}_1, \dots, \boldsymbol{y}_t$, and thus $\boldsymbol{z}^*$ is integral as well. Finally, we have

$$\|\boldsymbol{z}^* - \boldsymbol{r}\|_\infty = \|(\lambda_1 - \lfloor \lambda_1 \rfloor)\boldsymbol{y}_1 + \cdots + (\lambda_t - \lfloor \lambda_t \rfloor)\boldsymbol{y}_t\|_\infty \leq \sum_{i=1}^{t} \|\boldsymbol{y}_i\|_\infty \leq n\Delta. \qquad \blacktriangleleft$$

Proof of Proposition 4.1. According to Lemma 4.5, there is a rational solution $\frac{1}{a}E\boldsymbol{b}$ to $A\boldsymbol{x} \leq \boldsymbol{b}$, where $E \in \mathbb{Z}^{n \times \ell}$, $a \in \mathbb{Z} \setminus \{0\}$, $|a| \leq \Delta$, and $\|E\|_\infty \leq \Delta$. We set $D := \frac{1}{a}E$. Now since $A\boldsymbol{x} \leq \boldsymbol{b}$ has an integral solution, Lemma 4.8 yields an integral solution $\boldsymbol{z}^*$ close to $D\boldsymbol{b}$, meaning $\|\boldsymbol{z}^* - D\boldsymbol{b}\|_\infty \leq n\Delta$. We set $\boldsymbol{d} := \boldsymbol{z}^* - D\boldsymbol{b}$. Then of course $D\boldsymbol{b} + \boldsymbol{d} = \boldsymbol{z}^*$ is an integral solution to $A\boldsymbol{x} \leq \boldsymbol{b}$. Moreover, we clearly have $\|\boldsymbol{d}\|_\infty \leq n\Delta$. It remains to show that even $\|\boldsymbol{d}\|_{\mathsf{frac}} \leq n\Delta^2$. Indeed, since $\boldsymbol{z}^*$ is integral, $\boldsymbol{b}$ is integral, and $D = \frac{1}{a}E$ with integral E, we know that in $\boldsymbol{d} = \boldsymbol{z}^* - D\boldsymbol{b}$, every entry can be written with a as its denominator. As this fraction has absolute value at most $n\Delta$ and $|a| \leq \Delta$, both numerator and denominator have absolute value at most $n\Delta^2$. $\blacktriangleleft$

5 Matching complexity lower bounds

In this section we prove the lower bounds for Corollaries 3.5 and 3.6.

Detecting WQOs. We begin with the lower bound for Corollary 3.5. That is, we show that deciding whether an existential Presburger formula defines a WQO is coNEXP hard. The idea is essentially the same as the coNP lower bound for detecting WQOs for quantifier-free

formulas in [3, Sec. 8]. The proof follows from reducing the satisfiability problem for Π_2 sentences to WQO-definability of existential Presburger formulas. Given an instance γ of a Π_2 sentence, we synthesize an existential Presburger formula φ and show that φ defines a WQO iff γ is satisfiable. The coNEXP-completeness of Π_2 sentences follows from the NEXP-completeness of Σ_2 sentences [17].

Consider an instance of a Π_2 sentence

$$\gamma \coloneqq \forall \boldsymbol{y} \colon \exists \boldsymbol{x} \colon \psi(\boldsymbol{x}, \boldsymbol{y})$$

where ψ is quantifier-free, $\boldsymbol{x}$ ranges over $\mathbb{Z}^n$, and $\boldsymbol{y}$ ranges over $\mathbb{Z}^m$. The goal is to construct an existential PA formula φ such that φ defines a WQO iff γ is true. First we define the formula

$$\Gamma(\boldsymbol{y}) \coloneqq \exists \boldsymbol{x} \psi(\boldsymbol{x}, \boldsymbol{y})$$

Now, define the existential Presburger formula φ as follows.

$$\begin{aligned}\varphi((x, \boldsymbol{x}), (y, \boldsymbol{y})) \coloneqq &(x < 0 \wedge y < 0) \vee (x > 0 \wedge y > 0) \vee (x < 0 \wedge y = 0) \\ &\vee (x = 0 \wedge y > 0) \vee (x = 0 \wedge y = 0) \\ &\vee (x < 0 \wedge y > 0 \wedge \Gamma(\boldsymbol{y})).\end{aligned}$$

Here, both $\boldsymbol{x}$ and $\boldsymbol{y}$ range over $\mathbb{Z}^m$, hence φ defines a relation in $\mathbb{Z}^{1+m} \times \mathbb{Z}^{1+m}$. Since the existential quantifiers of Γ can be moved in front of φ, φ is an existential Presburger instance.

▶ **Lemma 5.1.** *φ defines a WQO if and only if γ is true i.e. $\Gamma(w)$ is true $\forall\, \boldsymbol{w} \in \mathbb{Z}^m$.*

Proof. ($\Rightarrow$) Let φ define a WQO. Assume for contradiction there exists $\boldsymbol{w} \in \mathbb{Z}^m$ such that $\Gamma(\boldsymbol{w})$ is false. Notice that, by definition, $\varphi((-1, \boldsymbol{w}), (0, \boldsymbol{w}))$ and $\varphi((0, \boldsymbol{w}), (1, \boldsymbol{w}))$ are true. By transitivity, we must have that $\varphi((-1, \boldsymbol{w}), (1, \boldsymbol{w}))$ is true. Therefore, $\Gamma(\boldsymbol{w})$ must be true. This is a contradiction.
($\Leftarrow$) Let $\Gamma(\boldsymbol{w})$ be true for all $\boldsymbol{w} \in \mathbb{Z}^m$. Let A, B and C be sets of all vectors over $\mathbb{Z}^{1+m}$ with negative, zero and positive first component, respectively. It is easy to see that φ relates all vectors within each of A, B and C. Further, $\varphi(\boldsymbol{u}, \boldsymbol{v})$ is true if

- $\boldsymbol{u} \in A$ and $\boldsymbol{v} \in B$, or
- $\boldsymbol{u} \in B$ and $\boldsymbol{v} \in C$, or
- $\boldsymbol{u} \in A$ and $\boldsymbol{v} \in C$.

This means that φ must be a transitive, reflexive relation. Hence, φ trivially defines a WQO: in any infinite sequence $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots$ of vectors over $\mathbb{Z}^{1+m}$, we can always find $\boldsymbol{u}_i, \boldsymbol{u}_j$ with $i < j$ such that both $\boldsymbol{u}_i, \boldsymbol{u}_j$ belong to either A or B or C. Since φ relates all vectors within each of these, the lemma follows. ◀

Monadic decomposability. Let us now show the lower bound for Corollary 3.6, i.e., that monadic decomposability for ∃PA formulas is coNEXP-hard. The idea is the same as the coNP-hardness for quantifier-free formulas in [2][2]. We reduce from the Π_2-fragment of Presburger arithmetic, which is known to be coNEXP-complete (see the discussion around Corollary 3.3). Suppose we are given a Π_2-formula $\varphi = \forall \boldsymbol{x} \exists \boldsymbol{y} \colon \psi(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x}$ contains n variables, and $\boldsymbol{y}$ contains m variables. We claim that the existential formula $\kappa = \exists \boldsymbol{y} \colon \psi(\boldsymbol{x}, \boldsymbol{y}) \vee z_1 = z_2$ (which has free variables $\boldsymbol{x}$, z_1, z_2) is monadically decomposable if and only if φ holds (see Section 5), which would clearly complete the reduction.

[2] As Anthony W. Lin and Matthew Hague explained to us, it would also not be difficult to adapt the idea of the coNP lower bound in [20, Lem. 2].

Indeed, if φ holds, then κ is satisfied for every vector in $\mathbb{Z}^{n+2}$ and is thus clearly monadically decomposable. Conversely, if φ does not hold, then there is some $\boldsymbol{a} \in \mathbb{Z}^n$ so that $\exists \boldsymbol{y} \colon \psi(\boldsymbol{a}, \boldsymbol{y})$ fails to hold. If κ were monadically decomposable, then so would the formula $\kappa \wedge \boldsymbol{x} = \boldsymbol{a}$, but this is equivalent to $z_1 = z_2$, which is clearly not monadically decomposable. This establishes the claim and hence coNEXP-hardness.

6 An exponential lower bound for quantifier elimination

Our main results show that one can eliminate a block of existential quantifiers with only an exponential blow-up. Using an example from [17, Thm. 2], we will now prove an exponential lower bound, even if constants are encoded in binary.

In the presence of binary encoded constants, we cannot use Weispfenning's lower bound argument [33, Thm. 3.1] (even for a singly exponential lower bound), which compares norms of vectors in finite sets defined by ∃PA vs. quantifier-free formulas. Indeed, it is a simple consequence of Pottier's bounds on Hilbert bases [24] that finite sets defined by ∃PA formulas consist of at most exponentially large vectors. With binary encoded constants, one easily constructs quantifier-free formulas defining finite sets of exponentially large vectors.

Instead, we measure the periodicity of infinite sets. Recall that every Presburger formula with one free variable defines an *ultimately periodic* set $S \subseteq \mathbb{Z}$, meaning that there are $n_0, p \in \mathbb{N}$, $p \geq 1$, such that for every $n \in \mathbb{Z}$, $|n| \geq n_0$, we have $n + p \in S$ if and only if $n \in S$. Such a p is called a *period* of S. For a formula φ with one free variable, we denote by $|\varphi|_{\mathsf{p}}$ the *smallest* period of the set defined by φ. In [17, Thm. 2], Haase constructs[3] a sequence $(\Phi_n(x))_{n \geq 0}$ of ∃PA formulas of size $O(n^2)$ such that $|\Phi_n|_{\mathsf{p}}$ is at least $2^{2^{\Omega(n)}}$. The following will imply that the formulas Φ_n require exponential-sized quantifier-free equivalents:

▶ **Lemma 6.1.** *Let φ be quantifier-free with one free variable. Then $|\varphi|_{\mathsf{p}} \leq 2^{|\varphi|}$.*

Proof. We prove this by structural induction. If φ is an atom $ax \leq b$, then $|\varphi|_{\mathsf{p}} = 1$. If φ is an atom $ax \equiv b \bmod c$ with constants a, b, c written in binary, then $|\varphi|_{\mathsf{p}} \leq |c| \leq 2^{|\varphi|}$. Moreover, $|\neg\varphi|_{\mathsf{p}} = |\varphi|_{\mathsf{p}}$. Now observe that if $S_1, S_2 \subseteq \mathbb{Z}$ are ultimately periodic sets, then we have $|S_1 \cup S_2|_{\mathsf{p}} \leq |S_1|_{\mathsf{p}} \cdot |S_2|_{\mathsf{p}}$ and $|S_1 \cap S_2|_{\mathsf{p}} \leq |S_1|_{\mathsf{p}} \cdot |S_2|_{\mathsf{p}}$. This implies $|\varphi_1 \vee \varphi_2|_{\mathsf{p}} \leq |\varphi_1|_{\mathsf{p}} \cdot |\varphi_2|_{\mathsf{p}} \leq 2^{|\varphi_1|+|\varphi_2|} \leq 2^{|\varphi|}$ and similarly $|\varphi_1 \wedge \varphi_2|_{\mathsf{p}} \leq |\varphi_1|_{\mathsf{p}} \cdot |\varphi_2|_{\mathsf{p}} \leq 2^{|\varphi_1|+|\varphi_2|} \leq 2^{|\varphi|}$. ◀

Now indeed, if $(\varphi_n)_{n \geq 0}$ is a sequence of quantifier-free equivalents of $(\Phi_n)_{n \geq 0}$, then for some constant $c > 0$ and large n, we have $2^{|\varphi_n|} \geq |\varphi_n|_{\mathsf{p}} = |\Phi_n|_{\mathsf{p}} \geq 2^{2^{cn}}$ and hence $|\varphi_n| \geq 2^{cn}$.

References

1 Parosh A. Abdulla, Karlis Čerāns, Bengt Jonsson, and Yih-Kuen Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Inform. and Comput.*, 160(1–2):109–127, 2000. doi:10.1006/inco.1999.2843.

2 Pascal Bergsträßer, Moses Ganardi, Anthony W. Lin, and Georg Zetzsche. Ramsey quantifiers in linear arithmetics. In *Proc. POPL 2024*, pages 1–32, 2024. doi:10.1145/3632843.

3 Pascal Bergsträßer, Moses Ganardi, Anthony W. Lin, and Georg Zetzsche. Ramsey quantifiers in linear arithmetics, 2023. doi:10.48550/arXiv.2311.04031.

4 Itshak Borosh and Leon B. Treybig. Bounds on positive integral solutions of linear Diophantine equations. *P. Am. Math. Soc.*, 55:299–304, 1976. doi:10.1090/S0002-9939-1976-0396605-3.

[3] See also Appendix B.

5 W. Cook, A. M. H. Gerards, A. Schrijver, and É. Tardos. Sensitivity theorems in integer linear programming. *Math. Program.*, 34:251–264, 1986. doi:10.1007/BF01582230.

6 D. C. Cooper. Theorem proving in arithmetic without multiplication. In Bernard Meltzer and Donald Michie, editors, *Proceedings of the Seventh Annual Machine Intelligence Workshop, Edinburgh, 1971*, volume 7, pages 91–99. Edinburgh University Press, 1972.

7 Alain Finkel. A generalization of the procedure of Karp and Miller to well structured transition systems. In *Proc. ICALP 1987*, volume 267 of *Lecture Notes in Computer Science*, pages 499–508. Springer, 1987. doi:10.1007/3-540-18088-5_43.

8 Alain Finkel and Ekanshdeep Gupta. The well structured problem for Presburger counter machines. In *Proc. FSTTCS 2019*, volume 150 of *LIPIcs*, pages 41:1–41:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.FSTTCS.2019.41.

9 Alain Finkel and Ekanshdeep Gupta. The well structured problem for Presburger counter machines. *CoRR*, abs/1910.02736, 2019. doi:10.48550/arXiv.1910.02736.

10 Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1–2):63–92, 2001. doi:10.1016/S0304-3975(00)00102-X.

11 Michael J. Fischer and Michael O. Rabin. Super-exponential complexity of Presburger arithmetic. In Bob F. Caviness and Jeremy R. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 122–135, Vienna, 1998. Springer Vienna. doi:10.1007/978-3-7091-9459-1_5.

12 Martin Fürer. The complexity of Presburger arithmetic with bounded quantifier alternation depth. *Theor. Comput. Sci.*, 18:105–111, 1982. doi:10.1016/0304-3975(82)90115-3.

13 Seymour Ginsburg and Edwin H Spanier. Bounded regular sets. *P. Am. Math. Soc.*, 17(5):1043–1049, 1966. doi:10.1090/S0002-9939-1966-0201310-3.

14 Erich Grädel. Dominoes and the complexity of subclasses of logical theories. *Ann. Pure Appl. Log.*, 43(1):1–30, 1989. doi:10.1016/0168-0072(89)90023-7.

15 Stéphane Grumbach, Philippe Rigaux, and Luc Segoufin. Spatio-temporal data handling with constraints. *GeoInformatica*, 5(1):95–115, 2001. doi:10.1023/A:1011464022461.

16 Florent Guépin, Christoph Haase, and James Worrell. On the existential theories of Büchi arithmetic and linear p-adic fields. In *Proc. LICS 2019*, pages 1–10. IEEE, 2019. doi:10.1109/LICS.2019.8785681.

17 Christoph Haase. Subclasses of Presburger arithmetic and the weak EXP hierarchy. In *Proc. CSL-LICS 2014*, pages 47:1–47:10. ACM, 2014. doi:10.1145/2603088.2603092.

18 Christoph Haase and Georg Zetzsche. Presburger arithmetic with stars, rational subsets of graph groups, and nested zero tests. In *Proc. LICS 2019*, pages 1–14. IEEE, 2019. doi:10.1109/LICS.2019.8785850.

19 Jacques Hadamard. Rèsolution d'une question relative aux dèterminants. *B. Sci. Math.*, 2(17):240–246, 1893.

20 Matthew Hague, Anthony W. Lin, Philipp Rümmer, and Zhilin Wu. Monadic decomposition in integer linear arithmetic. In *Proc. IJCAR 2020*, volume 12166 of *Lecture Notes in Computer Science*, pages 122–140. Springer, 2020. doi:10.1007/978-3-030-51074-9_8.

21 Gabriel Kuper, Leonid Libkin, and Jan Paredaens. *Constraint Databases*. Springer, 2000.

22 Mohan Nair. On Chebyshev-type inequalities for primes. *Am. Math. Mon.*, 89(2):126–129, 1982. doi:10.2307/2320934.

23 Derek C. Oppen. A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic. *J. Comput. Syst. Sci.*, 16(3):323–332, 1978. doi:10.1016/0022-0000(78)90021-1.

24 Loïc Pottier. Minimal solutions of linear diophantine systems: Bounds and algorithms. In *Proc. RTA 1991*, volume 488 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 1991. doi:10.1007/3-540-53904-2_94.

25 Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du I congres de Mathematiciens des Pays Slaves*, pages 92–101. Ksiaznica Atlas, 1929.

26 C. R. Reddy and Donald W. Loveland. Presburger arithmetic with bounded quantifier alternation. In *Proc. STOC 1978*, pages 320–325, New York, NY, USA, 1978. ACM. doi:10.1145/800133.804361.

27 J. Barkley Rosser and Lowell Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois J. Math.*, 6(1):64–94, 1962. doi:10.1215/ijm/1255631807.

28 Sasha Rubin. Automata presenting structures: A survey of the finite string case. *Bull. Symb. Log.*, 14(2):169–209, 2008. doi:10.2178/BSL/1208442827.

29 Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1986.

30 Margus Veanes, Nikolaj S. Bjørner, Lev Nachmanson, and Sergey Bereg. Monadic decomposition. *J. ACM*, 64(2):14:1–14:28, 2017. doi:10.1145/3040488.

31 Joachim von zur Gathen and Malte Sieveking. A bound on solutions of linear integer equalities and inequalities. *Proceedings of the American Mathematical Society*, 72(1):155–158, 1978. doi:10.1090/S0002-9939-1978-0500555-0.

32 Volker Weispfenning. The complexity of almost linear Diophantine problems. *J. Symb. Comput.*, 10(5):395–403, 1990. doi:10.1016/S0747-7171(08)80051-X.

33 Volker Weispfenning. Complexity and uniformity of elimination in Presburger arithmetic. In *Proc. ISSAC 1997*, pages 48–53. ACM, 1997. doi:10.1145/258726.258746.

A More Details for Lemma 4.6

We recall Cramer's rule which has been used in the proof.

▶ **Proposition A.1** (Cramer's rule). *Let a system of n linear equations for n unknowns be represented as*

$$Ax = b\,,$$

where A is an invertible $(n \times n)$ matrix. This system has as unique solution given by $x = (x_1, x_2, \ldots, x_n)$ where,

$$x_i = \frac{\det(A_i)}{\det(A)}$$

A_i is the matrix formed by replacing the ith column of A by b.

B Sets with large periods

The formula Φ_n constructed by Haase in [17, Thm. 2] defines the set

$$S_n = \{a \in \mathbb{N} \mid \exists b\colon 1 < b < 2^n,\ b \text{ divides } a\}.$$

and Haase argues that the smallest period of S_n is $2^{2^{\Omega(n)}}$. While the latter is true, the argument in [17] does not quite show this. The proof of [17, Thm. 2] argues that the smallest period of S_n is the least common multiple of the numbers $\{1, \ldots, 2^n - 1\}$, which is lower bounded by $2^{2^{\Omega(n)}}$ according to Nair [22]. However, as we will see, the smallest period of S_n is in fact a slightly smaller number. It is still lower bounded $2^{2^{\Omega(n)}}$, but this requires a different argument. We present a correction.

An easy fix for the result would be to instead define the set

$$\begin{aligned} S'_n &= \{a \in \mathbb{N} \mid \exists b\colon 1 < b < 2^n,\ b \text{ does not divide } a\} \\ &= \{a \in \mathbb{N} \mid \exists b, c\colon 1 < b < 2^n,\ 1 \le c < 2^n,\ b \text{ divides } a + c\}, \end{aligned}$$

for which a simple modification of the formulas Φ_n in [17] yields a polynomial-sized $\exists$PA formula Φ'_n. Moreover, the smallest period of S'_n is indeed the least common multiple of $\{1, \ldots, 2^n - 1\}$, and so Nair's bound would apply.

However, one can show that the smallest period of S_n is indeed lower bounded by $2^{2^{\Omega(n)}}$, just not by the least common multiple of $\{1, \ldots, 2^n - 1\}$. For any natural $n \in \mathbb{N}$, define the *primorial* of n, in symbols $n\#$, as the product of all primes $\leq n$. Thus, if $p_1, p_2, \ldots$ is the sequence of all primes in ascending order and $\pi(n)$ is the number of all prime numbers $\leq n$, then

$$n\# = \prod_{i=1}^{\pi(n)} p_i.$$

▷ Claim B.1. The smallest period of S_n is $2^n\#$.

Proof. Clearly, $2^n\#$ is a period of S_n: S_n is the set of all numbers that have a prime divisor among $\{2, \ldots, 2^n - 1\}$, and adding or subtracting the product of all these primes does not change that.

It remains to show that $2^n\#$ is the smallest period of S_n. Suppose k is a period of S_n. We will show that every prime p with $1 < p < 2^n$ is a divisor of k, which will clearly establish the claim. Let $\{p_1, \ldots, p_\ell\}$ be the primes in $\{2, \ldots, 2^n - 1\}$. Towards a contradiction, suppose there is a prime p_j, $1 \leq j \leq \ell$, that does not divide k. By the Chinese Remainder Theorem, the system of congruences

$$\begin{aligned} x &\equiv 1 \pmod{p_i} && \text{for each } i \in \{1, \ldots, \ell\},\ i \neq j, \\ x &\equiv -k \pmod{p_j} \end{aligned}$$

has infinitely many solutions $a \in \mathbb{N}$. For each such a, we have $a \notin S_n$, because a is not divisible by any p_i, $1 \leq i \leq \ell$. However, $a + k$ is divisible by p_j, and thus $a + k \in S_n$. Therefore, k cannot be a period of S_n. ◁

Using Claim B.1, we can now obtain the $2^{2^{\Omega(n)}}$ lower bound for the smallest period of S_n. This is because equation (3.14) of [27] implies that for every $m \geq 563$, we have $m\# \geq 2^{m-1}$. In particular, for $n \geq 10$, we have $2^n\# \geq 2^{2^n - 1}$. This proves that $|\Phi_n|_{\mathsf{p}}$ is lower bounded by $2^{2^{\Omega(n)}}$.

C Incorrect lower bounds on eliminating a block of existential quantifiers

We elaborate on a flaw in Weispfenning's paper [33] which is a consequence of misinterpreting results from the literature, from which he incorrectly concludes that the elimination of a block of existential quantifiers from a formula of Presburger arithmetic results in an inherent doubly exponential blow-up.

The main result of Section 3 of [33] is Theorem 3.1, which states that performing quantifier elimination on *arbitrary* formulas of Presburger arithmetic results in an inherent triply exponential blow-up, assuming unary encoding of numbers. To this end, Weispfenning invokes a result by Fischer and Rabin [11] who showed that there exists a function $g\colon \mathbb{N} \to \mathbb{N}$ such that for almost all n,

$$g(n) \geq 2^{2^{2^{n+1}}},$$

and who gave a family of formulas $\Phi_n(x, y, z)$ of Presburger arithmetic of size linear in n such that $\Phi_n(x, y, z)$ holds if and only if $0 \leq x, y, z < g(n)$ and $x \cdot y = z$. He then goes on concluding that the smallest quantifier-free formula defining the set $\{z \in \mathbb{Z} \mid \Phi_n(1, z, z)\}$ requires a formula of size at least $g(n)$, assuming unary encoding of numbers.

Weispfenning then continues sketching how to adapt this approach in the presence of a bounded number of quantifier alternations. To this end, he appeals to a result by Fürer [12], which states that for some constant $r > 0$, one can define multiplication up to

$$2^{2^{(n/a)^{ra}}} \tag{4}$$

using a formula of length n and a quantifier alternations. Adapting his line of reasoning from the general case, Weispfenning applies this to $a = 1$ and concludes that eliminating a block of existential quantifiers yields an inherent doubly exponential blow up. Fürer does indeed claim the existence of such a family in the third paragraph in [12, p. 108]. However, a close inspection of Fürer's proof reveals that these formulas are not constructed *for every a and n*, but only *for infinitely many a and n*. More specifically, Fürer supposes some given $k, m \in \mathbb{N}$ and constructs a formula of length $c(mk \log k + 1)$ and $2m + d$ quantifier alternations (see the seventh paragraph in [12, p. 108]). Here, c and d appear to be unspecified constants. By choosing $a = 2m + d$ and $n = c(mk \log k + 1)$, Fürer's claims then yield multiplication up to (4) for a suitable $r > 0$. In particular, Fürer's construction does not yield the existence of such formulas for *every* $a \in \mathbb{N}$.

Of course, from the fact that existential Presburger arithmetic allows for defining ultimately periodic sets with a doubly exponential period, cf. Appendix B, it is not unreasonable to believe that this could somehow be turned into a lower bound similar to the one claimed by Weispfenning. However, such large periods can already be produced by an exponential intersection of divisibility constraints and thus do not imply a doubly exponential lower bound on the formula size after eliminating a block of existentially quantified variables.

Forcing, Transition Algebras, and Calculi

Go Hashimoto
Kyushu University, Fukuoka, Japan

Daniel Găină
Kyushu University, Fukuoka, Japan

Ionuţ Ţuţu
Simion Stoilow Institute of Mathematics of the Romanian Academy, Bucharest, Romania

Abstract

We bring forward a logical system of *transition algebras* that enhances many-sorted first-order logic using features from dynamic logics. The sentences we consider include compositions, unions, and transitive closures of transition relations, which are treated similarly to the actions used in dynamic logics in order to define necessity and possibility operators. This leads to a higher degree of expressivity than that of many-sorted first-order logic. For example, one can finitely axiomatize both the finiteness and the reachability of models, neither of which are ordinarily possible in many-sorted first-order logic. We introduce syntactic entailment and study basic properties such as compactness and completeness, showing that the latter does not hold when standard finitary proof rules are used. Consequently, we define proof rules having both finite and countably infinite premises, and we provide conditions under which completeness can be proved. To that end, we generalize the forcing method introduced in model theory by Robinson from a single signature to a category of signatures, and we apply it to obtain a completeness result for signatures that are at most countable.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Proof theory; Theory of computation → Equational logic and rewriting

Keywords and phrases Forcing, institution theory, calculi, algebraic specification, transition systems

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.143

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2404.16111` [23]

Funding The work presented in this paper has been partially supported by Japan Society for the Promotion of Science, grant number 23K11048.

1 Introduction

Algebraic specification is one of the main approaches to formal methods that supports both the formal specification of software and hardware systems and the formal verification of their requirements. The underlying logic of an algebraic specification language is often presented as an institution [16], a category-theoretic formalization of the intuitive notion of logic that includes its syntax, semantics and the satisfaction relation between them. A lot of theoretical computer science has been developed within institution theory [9, 31, 10] based on the principle that formal specification should be based rigorously upon a concrete institution. Two notable specification languages have been designed by following this principle: CafeOBJ in Japan [11] and CASL in Europe [1]. However, there also is an exception given by the Maude system [3], which was originally developed at SRI International in the United States. The underlying logic of Maude, called rewriting logic [25], is not given as an institution, which has led to a series of developments that diverged from mainstream institution-theoretic approaches to topics such as modularization and heterogeneity [4].


Motivation. The main goal of the present study is to apply a body of methods and principles developed within institutional model theory for defining a denotational semantics of algebraic specification languages that are executable by term rewriting such that:

1. it enjoys the modular properties of the logic underlying CafeOBJ such as the *satisfaction condition* for signature morphisms, and therefore it can be formalized as an *institution*;
2. it has the rich expressivity of rewriting logic, in the sense that it can provide a semantics for the Maude language, in general, and for its strategy language [12], in particular.

In algebraic specification languages executable by rewriting such as Maude and CafeOBJ, systems are specified using two kinds of atomic statements: (*a*) *equations*, which define an algebraic structure on system states, with constructors and derived operations, for example; and (*b*) *transition rules*, which capture the behaviour of a system by telling us how the states may change as a result of certain actions. In the present contribution, we propose a logic of *transition algebras* where the models consist of many-sorted algebras equipped with binary relations that give semantics to the transition rules. From transition rules one can construct *actions* by applying composition, union, and the Kleene star (i.e., the reflexive and transitive closure of a relation). For the sake of simplicity, we omit the subsorting relation [14].

Many-sorted logical systems. Many-sorted logics are widely acknowledged as being suitable for applications in computer science. However, in pure mathematical logic, they tend to be classified as "inessential variations" [28] of their unsorted forms. This might be true w.r.t. some classical aspects such as compactness or axiomatizability. However, in general, moving from the unsorted to the many-sorted case is a far from trivial task. Allowing for multiple sorts, and thus for multiple carriers in models, some of which may be empty, alters the properties of the logics and significantly increases the complexity of proofs.

An important example of logical property that does not have a straightforward many-sorted generalization is Craig interpolation [7]. This property generally holds in unsorted first-order logic, but fails to hold in the many-sorted variant of the logic; a counterexample can be found, for example, in [2]. Finding the most general criteria for Craig interpolation property was an open problem originally stated in [33]. A solution based on techniques advanced in institutional model theory was provided in [22] after nearly two decades.

Moreover, as noticed in [13], if we admit models with potentially empty carrier sets, then proof rules for unsorted (or single-sorted) first-order logic may be unsound for its many-sorted counterpart. This already suggests that generalizations to other variants of many-sorted first-order logic may pose difficulties. The completeness results proved in an institutional setting, such as [29, 21, 18] are applicable to logical systems where models interpret sorts as non-empty sets. In fact, we are not aware of any completeness result for many-sorted first-order logic in which models interpret sorts as possibly empty sets.

Forcing. In the present contribution, we prove the completeness of the many-sorted logic of transition algebras by applying *forcing*. This technique was originally introduced by Paul Cohen [5, 6] in set theory to show the independence of the continuum hypothesis from the other axioms of Zermelo-Fraenkel set theory. Robinson [30] developed an analogous theory of forcing in model theory. In our setting, forcing is a technique used for constructing expanded models of consistent sets of sentences. More specifically, it allows one to expand a set of sentences while preserving satisfiability even if compactness does not hold in the underlying logical system. Transition algebra is not compact for the same reason classical dynamic propositional logic is not compact. Therefore, the classical Henkin method for proving completeness, which relies on compactness, is not applicable to transition algebra. Another

issue arises when proving completeness from the fact that we work with models with empty carriers, because the addition of constants does not necessarily preserve the consistency of theories. Therefore, we generalize the forcing method from a single signature to a category of signatures using ideas from institutional model theory such that the so-called Henkin constants can be added when needed in a way that preserves consistency.

An extended version of this work, which includes all proofs, is available on arXiv [23].

2 Transition algebra

In this section, we define *the logic of many-sorted transition algebras*, or *transition algebra* (TA), for short. We present, in order: signatures, models, sentences, and the TA satisfaction relation.

Signatures. The signatures we consider are ordinary algebraic signatures endowed with polymorphic transition labels and monotonic function symbols. We denote them by tuples of the form $(S, F \supseteq M, L)$, where:

- (S, F) is a many-sorted algebraic signature consisting of a set S of *sorts* and a family $F = \{F_{w,s} \mid w \in S^*, s \in S\}$ of sets of *function symbols*;
- M is a family of subsets $M_{w,s} \subseteq F_{w,s}$ of *monotonic* function symbols; and
- L is a set whose elements we call *transition labels*.

We often write $\sigma\colon w \to s \in F$ to indicate that $\sigma \in F_{w,s}$, and we refer to $w \in S^*$ and $s \in S$ as the *arity* and *sort*, respectively, of the symbol σ. Under this notation, F can also be regarded as an ordinary set consisting of *function declarations* of the form $\sigma\colon w \to s$. When w is the empty arity, we may speak of $\sigma\colon \to s$ as a *constant* (symbol) of sort s.

Throughout the paper, we let Σ, Σ', and Σ_i range over arbitrary signatures of the form $(S, F \supseteq M, L)$, $(S', F' \supseteq M', L')$, and $(S_i, F_i \supseteq M_i, L_i)$, respectively.

As usual in institution theory [9, 31], important constructions such as signature extensions with constants as well as open formulae and quantifiers are realized in a multi-signature setting, so moving between signatures is common. A *signature morphism* $\chi\colon \Sigma \to \Sigma'$ consists of an ordinary algebraic signature morphism $\chi\colon (S, F) \to (S', F')$ such that $\chi(M) \subseteq M'$ together with a function $L \to L'$, which we typically denote using the same symbol, χ.

▶ Remark 1. Signature morphisms compose componentwise. Their composition has identities and is associative, thus leading to a category Sig of signatures.

Models. Given a signature Σ, a Σ-*model* $\mathfrak{A}$ is an (S, F)-algebra $\mathfrak{A}$ that interprets every label $\lambda \in L$ as a *many-sorted transition relation* $\lambda^{\mathfrak{A}} \subseteq \mathfrak{A} \times \mathfrak{A}$ (that is, $\lambda^{\mathfrak{A}} = \{\lambda^{\mathfrak{A}}_s \mid s \in S\}$ and $\lambda^{\mathfrak{A}}_s \subseteq \mathfrak{A}_s \times \mathfrak{A}_s$ for all sorts $s \in S$) that respects monotonic function symbols (that is, for all function symbols $\sigma\colon s_1 \cdots s_n \to s$ in M, all tuples $(a_1, \ldots, a_n) \in \mathfrak{A}_{s_1} \times \cdots \times \mathfrak{A}_{s_n}$, all indices $k \in \{1, \ldots, n\}$, and all elements $b \in \mathfrak{A}_{s_k}$, if $\langle a_k, b\rangle \in \lambda^{\mathfrak{A}}_{s_k}$ then $\langle \sigma^{\mathfrak{A}}(a_1 \ldots, a_k, \ldots, a_n), \sigma^{\mathfrak{A}}(a_1 \ldots, b, \ldots, a_n)\rangle \in \lambda^{\mathfrak{A}}_s$).

A *homomorphism* $h : \mathfrak{A} \to \mathfrak{B}$ over a signature Σ is an algebraic (S, F)-homomorphism that preserves transitions: $h(\lambda^{\mathfrak{A}}) \subseteq \lambda^{\mathfrak{B}}$ for all $\lambda \in L$. It is easy to see that Σ-homomorphisms form a category, which we denote by $\mathsf{Mod}(\Sigma)$, under their obvious componentwise composition.

▶ Remark 2. Every signature morphism $\chi\colon \Sigma \to \Sigma$ determines a *model-reduct functor* $\_{\restriction_\chi}\colon \mathsf{Mod}(\Sigma') \to \mathsf{Mod}(\Sigma)$ such that:

- for every Σ'-model $\mathfrak{A}'$, $(\mathfrak{A}'{\restriction_\chi})_s = \mathfrak{A}'_{\chi(s)}$ for each sort $s \in S$, $\sigma^{\mathfrak{A}'{\restriction_\chi}} = \chi(\sigma)^{\mathfrak{A}'}$ for each symbol $\sigma \in F$, and $\lambda^{\mathfrak{A}'{\restriction_\chi}} = \chi(\lambda)^{\mathfrak{A}'}$ for each label $\lambda \in L$; and
- for every Σ'-homomorphism $h'\colon \mathfrak{A}' \to \mathfrak{B}'$, $(h'{\restriction_\chi})_s = h'_{\chi(s)}$ for each $s \in S$.

Moreover, the mapping $\chi \mapsto \_{\restriction_\chi}$ is functorial.

For any signature morphism $\chi : \Sigma \to \Sigma'$, any Σ-model $\mathfrak{A}$ and any Σ'-model $\mathfrak{A}'$ if $\mathfrak{A} = \mathfrak{A}'|_\chi$, we say that $\mathfrak{A}$ is the *χ-reduct* of $\mathfrak{A}'$, and that $\mathfrak{A}'$ is a *χ-expansion* of $\mathfrak{A}$. For example, for any many-sorted set X (say, of variables) that is disjoint from the set of constant-function symbols in Σ, consider the inclusion morphism $\iota_X : \Sigma \hookrightarrow \Sigma[X]$, where $\Sigma[X] = (S, F[X] \supseteq M, L)$ is the signature obtained from $\Sigma = (S, F \supseteq M, L)$ by adding the elements of X to F as new constant-operation symbols of appropriate sort. Then an expansion of a Σ-model $\mathfrak{A}$ along ι_X can be seen as a pair $\langle \mathfrak{A}, g : X \to \mathfrak{A} \rangle$, where g is a valuation of X in $\mathfrak{A}$.

As in many-sorted algebra, there is a special, initial model in $\mathsf{Mod}(\Sigma)$, which we denote by T_Σ, whose elements are ground terms built from function symbols, and whose transition relations are all empty. The Σ-model $T_\Sigma(X)$ of terms with variables from X is defined as the ι_X-reduct of $T_{\Sigma[X]}$; i.e., $T_\Sigma(X) = T_{\Sigma[X]}|_{\iota_X}$. The following property is an immediate consequence of the initiality of T_Σ.

▶ Remark 3. Any signature morphism $\chi\colon \Sigma \to \Sigma'$ determines uniquely a Σ-homomorphism $T_\Sigma \to T_{\Sigma'}|_\chi$. In order to simplify notations later on, we denote that homomorphism by $\chi\colon T_\Sigma \to T_{\Sigma'}|_\chi$; therefore, for any Σ-term $\sigma(t_1, t_2, \ldots, t_n)$, we have $\chi(\sigma(t_1, t_2, \ldots, t_n)) = \chi(\sigma)(\chi(t_1), \chi(t_2), \ldots, \chi(t_n))$.

Sentences. The *actions* over a signature Σ are defined by the following grammar:

$$\mathfrak{a} ::= \lambda \mid \mathfrak{a} \fatsemi \mathfrak{a} \mid \mathfrak{a} \cup \mathfrak{a} \mid \mathfrak{a}^*$$

where λ is a transition label of Σ. We let A denote the set of all actions obtained from transition labels declared in a signature Σ, and we extend the notational convention that we use for the components of signatures to their corresponding sets of actions; that is, we usually denote by A' the set of actions over a signature Σ', by A_i the set of actions over a signature Σ_i, and so on. Moreover, through a slight abuse of notation, we also denote by $\chi\colon A \to A'$ the canonical map determined by a signature morphism $\chi\colon \Sigma \to \Sigma'$.

To define sentences, we assume a countably infinite set of *variable names* $\{v_i \mid i < \omega\}$. A *variable* for a signature Σ is a triple $\langle v_i, s, \Sigma \rangle$, where v_i is a variable name and s is a sort in Σ – the third component is used only to ensure that variables are distinct from the constant-operation symbols declared in Σ, which is essential when dealing with quantifiers. The set $\mathsf{Sen}(\Sigma)$ of *sentences* over Σ is given by the following grammar:

$$\phi ::= t_1 = t_2 \mid t_1 \overset{\mathfrak{a}}{\Rightarrow} t_2 \mid \neg\phi \mid \bigvee \Phi \mid \exists X \cdot \phi'$$

where (a) t_1 and t_2 are (S, F)-terms of the same sort; (b) $\mathfrak{a} \in A$ is an action; (c) Φ is a finite set of Σ-sentences; and (d) X is a finite set of variables for Σ and ϕ' is a $\Sigma[X]$-sentence.

When $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$, we may write $\phi_1 \vee \phi_2 \vee \cdots \vee \phi_n$ instead of $\bigvee \Phi$. Besides the above core connectives, we also make use of the following convenient (and standard) abbreviations: $\bigwedge \Phi := \neg \bigvee \{\neg\phi \mid \phi \in \Phi\}$ for finite conjunctions; $\bot := \bigvee \emptyset$ for falsity; $\top := \bigwedge \emptyset = \neg\bot$ for truth; $\phi_1 \to \phi_2 := \neg\phi_1 \vee \phi_2$ for implications; and $\forall X \cdot \phi' := \neg\exists X \cdot \neg\phi'$ for universally quantified sentences.

▶ Remark 4. Any signature morphism $\chi\colon \Sigma \to \Sigma'$ can be canonically extended to a *sentence-translation function* $\chi\colon \mathsf{Sen}(\Sigma) \to \mathsf{Sen}(\Sigma')$ given by:

- $\chi(t_1 = t_2) = (\chi(t_1) = \chi(t_2))$;
- $\chi(t_1 \overset{\mathfrak{a}}{\Rightarrow} t_2) = \chi(t_1) \overset{\chi(\mathfrak{a})}{\Longrightarrow} \chi(t_2)$;
- $\chi(\neg\phi) = \neg\chi(\phi)$;
- $\chi(\bigvee \Phi) = \bigvee \chi(\Phi)$; and
- $\chi(\exists X \cdot \phi') = \exists X' \cdot \chi'(\phi')$, where $X' = \{\langle x, \chi(s), \Sigma' \rangle \mid \langle x, s, \Sigma \rangle \in X\}$ and $\chi' : \Sigma[X] \to \Sigma'[X']$ is the extension of χ mapping each variable $\langle x, s, \Sigma \rangle \in X$ to $\langle x, \chi(s), \Sigma' \rangle \in X'$.

Moreover, this sentence-translation mapping is functorial in χ.

For the sake of simplicity, we identify variables only by their name and sort, provided that there is no danger of confusion. Using this convention, each inclusion morphism $\iota\colon \Sigma \hookrightarrow \Sigma'$ determines an inclusion function $\iota\colon \mathsf{Sen}(\Sigma) \hookrightarrow \mathsf{Sen}(\Sigma')$, which corresponds to the approach of classical model theory. This simplifies the presentation greatly. A situation when we cannot apply this convention arises when translating a Σ-sentence $\exists X \cdot \phi$ along the inclusion $\iota_X : \Sigma \hookrightarrow \Sigma[X]$.

Satisfaction relation. Actions are interpreted as binary transition relations in models. Given a model $\mathfrak{A}$ over a signature Σ, and actions $\mathfrak{a}, \mathfrak{a}_1, \mathfrak{a}_2 \in A$, we have:

- $(\mathfrak{a}_1 \fatsemi \mathfrak{a}_2)^{\mathfrak{A}} = \mathfrak{a}_1^{\mathfrak{A}} \fatsemi \mathfrak{a}_2^{\mathfrak{A}}$ (i.e., diagrammatic composition of binary relations);
- $(\mathfrak{a}_1 \cup \mathfrak{a}_2)^{\mathfrak{A}} = \mathfrak{a}_1^{\mathfrak{A}} \cup \mathfrak{a}_2^{\mathfrak{A}}$ (the union of binary relations); and
- $(\mathfrak{a}^*)^{\mathfrak{A}} = (\mathfrak{a}^{\mathfrak{A}})^*$ (the reflexive and transitive closure of binary relations).

We define the *satisfaction relation* between models and sentences as follows:

- $\mathfrak{A} \models t_1 = t_2$ iff $t_1^{\mathfrak{A}} = t_2^{\mathfrak{A}}$;
- $\mathfrak{A} \models t_1 \overset{\mathfrak{a}}{\Rightarrow} t_2$ iff $(t_1^{\mathfrak{A}}, t_2^{\mathfrak{A}}) \in \mathfrak{a}^{\mathfrak{A}}$;
- $\mathfrak{A} \models \neg\phi$ iff $\mathfrak{A} \not\models \phi$,
- $\mathfrak{A} \models \bigvee \Phi$ iff $\mathfrak{A} \models \phi$ for some sentence $\phi \in \Phi$, and
- $\mathfrak{A} \models \exists X \cdot \phi'$ iff $\mathfrak{A}' \models \phi'$ for some expansion $\mathfrak{A}'$ of $\mathfrak{A}$ to the signature $\Sigma[X]$.

For the sake of simplicity, we write $d_1 \overset{\mathfrak{a}}{\Longrightarrow} d_2$ if $\langle d_1, d_2 \rangle \in \mathfrak{a}^{\mathfrak{A}}$.

Let ϕ, ϕ' be sets of Σ-sentences, $\mathfrak{A}$ a Σ-model. We also use the following notations:

- $\mathfrak{A} \models \Phi$ iff $\mathfrak{A} \models \phi$ for all sentences $\phi \in \Phi$;
- $\Gamma \models \Phi$ iff $\mathfrak{A} \models \Gamma$ implies $\mathfrak{A} \models \Phi$ for all Σ-models $\mathfrak{A}$.

In particular, we write $\Gamma \models \phi$ instead of $\Gamma \models \{\phi\}$ for any set of sentences Γ and any single sentence ϕ.

▶ **Proposition 5.** *For all signature morphisms $\chi : \Sigma \to \Sigma'$, all Σ'-models $\mathfrak{A}$ and all sentences $\phi \in \mathsf{Sen}(\Sigma)$ we have: $\mathfrak{A}{\restriction_\chi} \models \phi$ iff $\mathfrak{A} \models \chi(\phi)$.*

▶ **Example 6** (CCS)**.** To illustrate the expressivity of transition algebra, we refer to Robin Milner's *calculus of communicating systems* (CCS) [26, 27], which is emblematic of a broad family of formal languages used for modelling and reasoning about concurrency. In a nutshell, CCS is a process calculus that enables syntactic descriptions of concurrent systems to be written, and subsequently manipulated and analysed, based on two kinds of atomic entities – process identifiers and channel names – and a handful of composition operators.

To start, we assume two sets: *PI* of *process identifiers*, and *CN* of so-called *channel names*, which capture the interaction capabilities of processes. Take, for instance, the following famous quote by Alfréd Rényi: "A mathematician is a machine for turning coffee into theorems" [32]. This can be modelled in CCS as an interaction between two processes, a mathematician and a coffee vending machine, that trade coffee (in exchange, perhaps, of coins or some other form of payment) in order to jointly produce theorems. Hence, we can consider theorems, coffee, and coins as types of interactions between the two processes.

For each channel name $c \in CN$, we let $\overline{c}$ be a new symbol, distinct from all channel names, called the *co-name* of c. We also let $\overline{CN} = \{\overline{c} \mid c \in CN\}$ be the set of all co-names, and $L = CN \cup \overline{CN}$ be the set of *labels*. Intuitively, we may regard the symbols in CN as inputs of some process, and the symbols in $\overline{CN}$ as outputs. Besides labels, we also consider an additional *silent-action* symbol, denoted τ, that indicates an internal, unobservable behaviour of the system under consideration. Altogether, we refer to the symbols in $A = L \cup \{\tau\}$ as CCS *actions*.[1] *Processes* over CN and PI are defined according to the following grammar:

[1] Although they share the name "action", CCS actions are conceptually very different from the actions we have defined for transition algebra. To distinguish the two, we always prefix the former by CCS.

$$P \ ::= \ 0 \mid \pi \mid a \,.\, P \mid P + P \mid P \text{ `|' } P \mid P \setminus k$$

where (a) 0 denotes a special terminal, inactive process; (b) $\pi \in PI$ is a process identifier; (c) $a \in A$ is a CCS action, which can be used to prefix a process P in order to form a new process, $a \,.\, P$, that intuitively performs a then continues as P; (d) '+' denotes the non-deterministic choice between two processes; (e) '|' denotes the parallel composition of two processes; and (f) $k \in CN$ is a channel name, which can be used in expressions like $P \setminus k$ to form a new, restricted process with the same interaction capabilities as P except for the labels k and $\overline{k}$. To simplify the presentation, we omit the relabelling operator because it plays no role in the examples we consider in this paper and it could be added with ease if needed. For a comprehensive account of CCS, see for example [27].

A CCS *context*, or *program*, is a set of *declarations* of the form $\pi ::= P$, where $\pi \in PI$ is a process identifier and P is a process conforming to the grammar introduced above, such that any two distinct declarations have distinct left-hand sides; in other words, we do not admit multiple declarations of the same process identifier within a given context.

Using this syntax, the interaction between mathematicians and coffee vending machines announced in Alfréd Rényi's quote can be formalized as a parallel composition of processes over $PI = \{\texttt{Mathematician}, \texttt{CoffeeVM}\}$ and $CN = \{\texttt{coin}, \texttt{coffee}, \texttt{theorem}\}$, which we write as $\texttt{Mathematician} \mid \texttt{CoffeeVM}$, where `Mathematician` and `CoffeeVM` are process identifiers "defined" recursively according to the following context:

- $\texttt{Mathematician} ::= \overline{\texttt{coin}} \,.\, \texttt{coffee} \,.\, \overline{\texttt{theorem}} \,.\, \texttt{Mathematician}$,
- $\texttt{CoffeeVM} ::= \texttt{coin} \,.\, \overline{\texttt{coffee}} \,.\, \texttt{CoffeeVM}$.

Thanks to the expressivity of transition algebra, we can easily capture both the syntax and the operational semantics of CCS. For the syntax of processes, it suffices to consider a many-sorted TA signature with $S = \{\texttt{Channel}, \texttt{Action}, \texttt{Process}\}$ and with F given by the following function symbols (which employ OBJ's [15] and Maude's [3] mixfix notation):

- $0\colon \to \texttt{Process}$,
- $\pi\colon \to \texttt{Process}$ for each process identifier $\pi \in PI$,
- $a\colon \to \texttt{Action}$ for each CCS action $a \in A$,
- $\_\,.\,\_\colon \texttt{Action Process} \to \texttt{Process}$,
- $\_ + \_\,,\ \_ \mid \_\colon \texttt{Process Process} \to \texttt{Process}$,
- $k\colon \to \texttt{Channel}$ for each channel name $k \in CN$,[2]
- $\_ \setminus \_\colon \texttt{Process Channel} \to \texttt{Process}$.

The parallel-composition operator is the only monotonic function symbol of this example. For convenience, we also declare the parallel-composition operator and the non-deterministic choice as associative, commutative, and with identity 0. These properties can be presented – as usual in algebraic specification – using plain equations. To capture and reason about the behaviour of processes, we regard each CCS action as an atomic action (i.e., a transition label) in transition algebra – and those are the only TA labels that we consider here.

The transitional semantics of a CCS program *Pgm* is given by the following collection of axioms. The transition-algebra sentences below are all universally quantified over variables P, P', Q, Q' of sort `Process` and k of sort `Channel`; however, we drop the quantifiers in order to simplify the notation. We also use names for the axioms (at the beginning of each line) that are indicative of the transition rules defined in [27].

[2] To avoid subsorting, we overload channel names, which can be seen either as constants of sort `Channel` or as constants of sort `Action` depending on the context in which they are used.

- (Act) $a \,.\, P \overset{a}{\Rightarrow} P$ for all $a \in A$,
- (Sum) $P \overset{a}{\Rightarrow} P' \to P + Q \overset{a}{\Rightarrow} P'$ for all $a \in A$,
- (Com) $P \overset{c}{\Rightarrow} P' \wedge Q \overset{\overline{c}}{\Rightarrow} Q' \to P \mid Q \overset{\tau}{\Rightarrow} P' \mid Q'$ for all $c \in CN$,
- (Res) $P \overset{a}{\Rightarrow} P' \wedge a \neq k \wedge \overline{a} \neq k \to P \setminus k \overset{a}{\Rightarrow} P' \setminus k$ for all $a \in A$,[3]
- (Con) $P \overset{a}{\Rightarrow} P' \to \pi \overset{a}{\Rightarrow} P'$ for all $\pi ::= P \in Pgm$.

To simplify some of the notations used later on in the paper, for any process P and any non-empty and finite sequence $K = (k_i \mid 1 \leq i \leq n)$ of channel names, we also write $P \setminus K$ in place of $P \setminus k_1 \setminus \cdots \setminus k_n$ and we consider the following derived form of the axiom (Res):

- (Res^*) $P \overset{a}{\Rightarrow} P' \wedge \bigwedge\{a \neq k_i \wedge \overline{a} \neq k_i \mid 1 \leq i \leq n\} \to P \setminus K \overset{a}{\Rightarrow} P' \setminus K$ for all $a \in A$.

Similar encodings of CCS in languages that support transitions can be found in the rewriting-logic literature, notably in [24, 34, 8]. But the encodings presented therein rely on a notion of *derivative* of a process instead of reasoning about plain processes, which is usually because labelled transitions cannot be used in the conditions of Horn clauses such as *Sum* and *Com*. That is, the axiomatization is done in terms of pairs $\langle \alpha, P \rangle$, where P is a process and α is a CCS action or a sequence of CCS actions leading to P. In TA, this additional step can be avoided because the use of labelled transitions is unrestricted, which allows our axioms to be nearly to-the-letter transcriptions of Robin Milner's rules for CCS.

3 Entailment relations

In this section, we define the proof-theoretic properties necessary for proving our results such as entailment relation, soundness and completeness. Before we proceed, let us recall an example from [13], which shows that classical rules of first-order deduction are not sound.

▶ **Example 7.** Let $\Sigma = (S, F)$ be an algebraic signature consisting of:

- two sorts, that is, $S = \{Elt, Bool\}$, and
- five function symbols $F = \{true :\to Bool, false :\to Bool, \sim\_ : Bool \to Bool, \_\&\_ : Bool\ Bool \to Bool, \_ + \_ : Bool\ Bool \to Bool, foo : Elt \to Bool\}$.

Let Γ be a set of sentences over Σ which consists of the following sentences:

- $\sim true = false$ and $\sim false = true$,
- $\forall y \cdot y \ \& \sim y = false$ and $\forall y \cdot y \ \& \ y = y$,
- $\forall y \cdot y + \sim y = true$ and $\forall y \cdot y \ + \ y = y$, and
- $\forall x \cdot \ \sim foo(x) = foo(x)$.

Using the ordinary rules of first-order deduction, we can show that

$$\begin{aligned} true &= foo(x) + \sim foo(x) \\ &= foo(x) + foo(x) \\ &= foo(x) \\ &= foo(x) \ \& \ foo(x) \\ &= foo(x) \ \& \sim foo(x) \\ &= false \end{aligned} \tag{1}$$

As a result, one would expect $true = false$ to hold in all algebras satisfying Γ. But that is not the case. To see why, suppose $\mathfrak{A}$ is the algebra obtained from T_Σ through a factorization under the congruence relation $\equiv_\Gamma$ generated by Γ, that is:

[3] As usual in CCS, we extend the over-line notation employed for channel co-names to a bijection $\overline{\cdot} : A \to A$ given by $\overline{\overline{c}} = c$ for all channel names c and by $\overline{\tau} = \tau$ for the silent action.

- $\mathfrak{A}_{Bool} = \{true/_{\equiv_\Gamma}, false/_{\equiv_\Gamma}\}$ and $\mathfrak{A}_{Elt} = \emptyset$,
- $\sim$ is interpreted as the negation, & as the conjunction and + as the disjunction, and
- $foo^{\mathfrak{A}}$ is the empty function.

Clearly, the algebra $\mathfrak{A}$ satisfies the sentences in Γ referring to the negation, conjunction, and disjunction of booleans. Moreover, since there is no function from $\{x\}$ to $\mathfrak{A}_{Elt} = \emptyset$, we have $\mathfrak{A} \models_\Sigma \forall x \cdot \sim foo(x) = foo(x)$. It follows that $\mathfrak{A} \models_\Sigma \Gamma$ but $\mathfrak{A} \not\models_\Sigma true = false$.

This shows that moving from the unsorted to the many-sorted case is not as straightforward as one might expect. Since a model may have some empty domains, one needs to design proof rules that take into account changes of signatures.

▶ **Definition 8** (Entailment relation). *An entailment relation* $\vdash = \{\vdash_\Sigma\}_{\Sigma \in |\mathsf{Sig}^{\mathsf{TA}}|}$ *is a family of binary relations between sets of sentences indexed by signatures, that is,* $\vdash_\Sigma \subseteq \mathcal{P}(\mathsf{Sen}(\Sigma)) \times \mathcal{P}(\mathsf{Sen}(\Sigma))$ *for all first-order signatures* Σ*, such that the following properties are satisfied:*

$$(Monotonicity)\ \frac{\Gamma \supseteq \Phi}{\Gamma \vdash_\Sigma \Phi} \qquad (Transitivity)\ \frac{\Gamma \vdash_\Sigma \Phi \qquad \Phi \vdash_\Sigma \Psi}{\Gamma \vdash_\Sigma \Psi}$$

$$(Union)\ \frac{\Gamma \vdash_\Sigma \phi \text{ for all } \phi \in \Phi}{\Gamma \vdash_\Sigma \Phi} \qquad (Translation)\ \frac{\Gamma \vdash_\Sigma \Phi}{\chi(\Gamma) \vdash_{\Sigma'} \chi(\Phi)} \text{ where } \chi : \Sigma \to \Sigma'$$

For the sake of simplicity, we write $\Gamma \vdash_\Sigma \phi$ rather than $\Gamma \vdash_\Sigma \{\phi\}$. Also, we omit the subscript Σ from the notation $\vdash_\Sigma$ when it is clear from the context. An example of entailment relation is $\models$. It is straightforward to prove that $\models$ satisfies $(Monotonicity)$, $(Transitivity)$, $(Union)$ and $(Translation)$.

▶ **Definition 9** (Entailment properties). *An entailment relation* $\vdash$ *is* sound (complete) *if* $\vdash \subseteq \models$ ($\models \subseteq \vdash$)*. An entailment relation* $\vdash$ *is* α-compact*, where* α *is an infinite cardinal, if*

$$\Gamma \vdash_\Sigma \phi \text{ implies } \Gamma_\alpha \vdash_\Sigma \phi \text{ for some subset } \Gamma_\alpha \subseteq \Gamma \text{ of cardinality } \mathsf{card}(\Gamma_\alpha) < \alpha,$$

for all signatures Σ*, all sets of* Σ*-sentences* Γ *and all* Σ*-sentences* ϕ*. If* $\alpha = \omega$*, we say, simply, that* $\vdash$ *is compact.*

The dynamic entailment relation is defined in two steps. Firstly, we define an entailment relation to reason about the logical consequences of atomic sentences, given as equations or relations. Secondly, we define the dynamic entailment relation by adding proof rules to deal with actions, Boolean connectives and quantifiers.

3.1 Basic entailment relation

The fragment obtained from TA by restricting the sentences to atoms is studied in this section.

▶ **Definition 10** (Basic entailment relation). The basic entailment relation $\vdash^b$ *is the least entailment relation closed under the following basic proof rules:*

$$(R)\ \frac{}{\Gamma \vdash_\Sigma t = t} \qquad (S)\ \frac{\Gamma \vdash_\Sigma t_1 = t_2}{\Gamma \vdash_\Sigma t_2 = t_1} \qquad (T)\ \frac{\Gamma \vdash_\Sigma t_1 = t_2 \qquad \Gamma \vdash_\Sigma t_2 = t_3}{\Gamma \vdash_\Sigma t_1 = t_3}$$

$$(F)\ \frac{\Gamma \vdash_\Sigma t_i = t_i' \quad \text{for } 1 \leq i \leq n}{\Gamma \vdash_\Sigma \sigma(t_1, \ldots, t_n) = \sigma(t_1', \ldots, t_n')} \qquad (P)\ \frac{\Gamma \vdash_\Sigma t_1 = t_1' \quad \Gamma \vdash_\Sigma t_2 = t_2' \quad \Gamma \vdash_\Sigma t_1 \overset{\lambda}{\Longrightarrow} t_2}{\Gamma \vdash_\Sigma t_1' \overset{\lambda}{\Longrightarrow} t_2'}$$

$$(M)\ \frac{\Gamma \vdash_\Sigma t_j \overset{\lambda}{\Longrightarrow} u_j}{\Gamma \vdash_\Sigma f(t_1, \ldots, t_j, \ldots, t_n) \overset{\lambda}{\Longrightarrow} f(t_1, \ldots, u_j, \ldots, t_n)} \quad \text{where } f \in M$$

▶ **Lemma 11** (Basic compactness). *The basic entailment relation is compact.*

Any set of atomic sentences E defined over a signature Σ determines a congruence $\equiv^E := \{t_1 \equiv^E t_2 \mid E \vdash_\Sigma t_1 = t_2\}$ on T_Σ. One can construct a model $\mathfrak{A}^E$ from the initial model of terms T_Σ factorized by the congruence $\equiv^E$ interpreting each transition label λ in Σ as the set $\{(t_1, t_2) \mid t_1, t_2 \in T_{\Sigma,s} \text{ and } E \vdash_\Sigma t_1 \overset{\lambda}{\Longrightarrow} t_2\}$.

▶ **Lemma 12.** *Let E be a set of atomic sentences defined over a signature Σ. For all Σ-models $\mathfrak{A}$, we have $\mathfrak{A} \models E$ iff there exists a unique homomorphism $\mathfrak{A}^E \to \mathfrak{A}$.*

Lemma 12 says that the satisfaction of E by a model $\mathfrak{A}$ is equivalent with the existence of a unique homomorphism from $\mathfrak{A}^E$ to $\mathfrak{A}$. In particular, $\mathfrak{A}^E$ is the initial model of E. See [9] for a proof of Lemma 12.

▶ **Proposition 13** (Basic completeness). *For any set of atomic sentences E and any atomic sentence φ defined over a signature Σ, the following are equivalent:*

$$(a)\ E \models \varphi, \qquad (b)\ \mathfrak{A}^E \models \varphi, \qquad \text{and} \qquad (c)\ E \vdash^b \varphi.$$

3.2 Dynamic entailment relation

The dynamic entailment relation is built on top of basic entailment relation by adding the proof rules to reason about actions, Boolean connectives, and first-order quantifiers.

▶ **Definition 14** (Dynamic entailment relation). The dynamic entailment relation $\vdash$ *is the least entailment relation closed under the basic proof rules presented in Definition 10 and the following proof rules:*

Proof rules for actions

$$(Comp_I)\ \frac{\Gamma \vdash_\Sigma t_1 \overset{\mathfrak{a}_1}{\Longrightarrow} t \qquad \Gamma \vdash_\Sigma t \overset{\mathfrak{a}_2}{\Longrightarrow} t_2}{\Gamma \vdash_\Sigma t_1 \overset{\mathfrak{a}_1 ; \mathfrak{a}_2}{\Longrightarrow} t_2}$$

$$(Comp_E)\ \frac{\Gamma \vdash_\Sigma t_1 \overset{\mathfrak{a}_1 ; \mathfrak{a}_2}{\Longrightarrow} t_2 \quad \Gamma \cup \{t_1 \overset{\mathfrak{a}_1}{\Longrightarrow} x, x \overset{\mathfrak{a}_2}{\Longrightarrow} t_2\} \vdash_{\Sigma[x]} \phi}{\Gamma \vdash_\Sigma \phi}$$

$$(Union_I)\ \frac{\Gamma \vdash_\Sigma t_1 \overset{\mathfrak{a}_i}{\Longrightarrow} t_2}{\Gamma \vdash_\Sigma t_1 \overset{\mathfrak{a}_1 \cup \mathfrak{a}_2}{\Longrightarrow} t_2} \qquad (Union_E)\ \frac{\Gamma \vdash_\Sigma t_1 \overset{\mathfrak{a}_1 \cup \mathfrak{a}_2}{\Longrightarrow} t_2 \quad \Gamma \cup \{t_1 \overset{\mathfrak{a}_i}{\Longrightarrow} t_2\} \vdash_\Sigma \phi \text{ for all } i \in \{1,2\}}{\Gamma \vdash_\Sigma \phi}$$

$$(Star_I)\ \frac{\Gamma \vdash_\Sigma t_1 \overset{\mathfrak{a}^n}{\Longrightarrow} t_2}{\Gamma \vdash_\Sigma t_1 \overset{\mathfrak{a}^*}{\Longrightarrow} t_2} \qquad (Star_E)\ \frac{\Gamma \vdash_\Sigma t_1 \overset{\mathfrak{a}^*}{\Longrightarrow} t_2 \quad \Gamma \cup \{t_1 \overset{\mathfrak{a}^n}{\Longrightarrow} t_2\} \vdash_\Sigma \phi \text{ for all } n \in \omega}{\Gamma \vdash_\Sigma \phi}$$

Proof rules for Boolean connectives

$$(Neg_D)\ \frac{\Gamma \vdash_\Sigma \neg\neg\phi}{\Gamma \vdash_\Sigma \phi} \qquad (False)\ \frac{\Gamma \vdash_\Sigma \bot}{\Gamma \vdash_\Sigma \phi}$$

$$(Neg_I)\ \frac{\Gamma \cup \{\phi\} \vdash_\Sigma \bot}{\Gamma \vdash_\Sigma \neg\phi} \qquad (Neg_E)\ \frac{\Gamma \vdash_\Sigma \neg\phi}{\Gamma \cup \{\phi\} \vdash_\Sigma \bot}$$

$$(Disj_I)\ \frac{\Gamma \vdash_\Sigma \phi}{\Gamma \vdash_\Sigma \vee\Phi} \text{ where } \phi \in \Phi \qquad (Disj_E)\ \frac{\Gamma \vdash_\Sigma \vee\Phi \quad \Gamma \cup \{\phi\} \vdash_\Sigma \gamma \text{ for all } \phi \in \Phi}{\Gamma \vdash_\Sigma \gamma}$$

Proof rules for first-order quantifiers

$$(Quant_I)\ \frac{\Gamma \cup \{\phi\} \vdash_{\Sigma[X]} \gamma}{\Gamma \cup \{\exists X \cdot \phi\} \vdash_\Sigma \gamma} \qquad (Quant_E)\ \frac{\Gamma \cup \{\exists X \cdot \phi\} \vdash_\Sigma \gamma}{\Gamma \cup \{\phi\} \vdash_{\Sigma[X]} \gamma}$$

$$(Subst)\ \frac{\Gamma \vdash_\Sigma \theta(\phi)}{\Gamma \vdash_\Sigma \exists X \cdot \phi} \qquad \text{where } \theta : X \to T_\Sigma \text{ is a substitution}$$

▶ **Proposition 15** (ω_1-compactness). *We have that*

1. *the dynamic entailment relation $\vdash$ is ω_1-compact, and*
2. *the satisfaction relation $\models$ is not ω_1-compact.*

The first statement holds because the dynamic entailment relation is generated by proof rules with an at most countable number of premises. For uncountable signatures Σ, the satisfaction relation $\models_\Sigma$ is not ω_1-compact. It follows that the dynamic logic proposed in this contribution, TA, is not complete. However, the restriction of TA to countable signatures, TA^c, is complete. Since TA^c is not compact, the Henkin method for proving completeness is not applicable.

▶ **Example 16** (Analysis of CCS programs). Recall the CCS description of the interaction between mathematicians and coffee vending machines discussed in Section 2, and let `Institute` be an abbreviation for the following process:

$(\texttt{Mathematician} \mid \texttt{CoffeeVM}) \setminus (\texttt{coin}, \texttt{coffee})$.

In this context, can we check, as an example, that the process `Institute` is able to continuously output theorems? The property can be formalized in TA as a transition

$$\texttt{Institute} \xRightarrow{\tau^* \,;\, \overline{\texttt{theorem}} \,;\, \tau^*} \texttt{Institute}$$

whose τ-components correspond to internal communications between sub-processes of the institute – i.e., mathematicians and vending machines. Therefore, we need to check an entailment of the form $\Gamma \vdash_\Sigma \phi$, where (a) Σ is the TA-signature that consists of the process identifiers `Mathematician` and `CoffeeVM` together with the CCS process-building operators for action prefixing, non-deterministic choice, parallel composition, etc., discussed in Section 2; (b) Γ is the set of Σ-sentences given by the axiom schemas *Act*, *Sum*, *Com*, *Res**, and *Con* listed on page 7 together with equations pertaining to the axiomatization of CCS actions (e.g., $\overline{\texttt{theorem}} \neq \texttt{coffee}$), as well as equations that capture elementary properties of processes such as the associativity, commutativity, and identity element of the non-deterministic-choice and parallel-composition operators; and (c) ϕ is the transition written above.

The proof mimics the following chain of CCS transitions:

$$\begin{aligned}\texttt{Institute} &\xRightarrow{\tau} (\texttt{coffee} \,.\, \overline{\texttt{theorem}} \,.\, \texttt{Mathematician} \mid \overline{\texttt{coffee}} \,.\, \texttt{CoffeeVM}) \setminus (\texttt{coin}, \texttt{coffee}) \\ &\xRightarrow{\tau} (\overline{\texttt{theorem}} \,.\, \texttt{Mathematician} \mid \texttt{CoffeeVM}) \setminus (\texttt{coin}, \texttt{coffee}) \\ &\xRightarrow{\overline{\texttt{theorem}}} (\texttt{Mathematician} \mid \texttt{CofeeVM}) \setminus (\texttt{coin}, \texttt{coffee}) = \texttt{Institute}\end{aligned}$$

To shorten the presentation of the proof, we use the following derived proof rule:

$$(GMP)\ \frac{\Gamma \vdash_\Sigma \forall X \cdot \bigwedge \Phi \to \gamma \quad \Gamma \vdash_\Sigma \theta(\Phi)}{\Gamma \vdash_\Sigma \theta(\gamma)} \qquad \text{where } \theta : X \to T_\Sigma \text{ is a substitution.}$$

In addition, we simplify the notations by writing down only the conclusions of entailments and by abbreviating `Mathematician` as `M`, `CoffeeVM` as `CVM`, and the sequence of channel names $(\texttt{coin}, \texttt{coffee})$ as K. This leads us to the (sketch of) proof tree depicted in Figure 1.

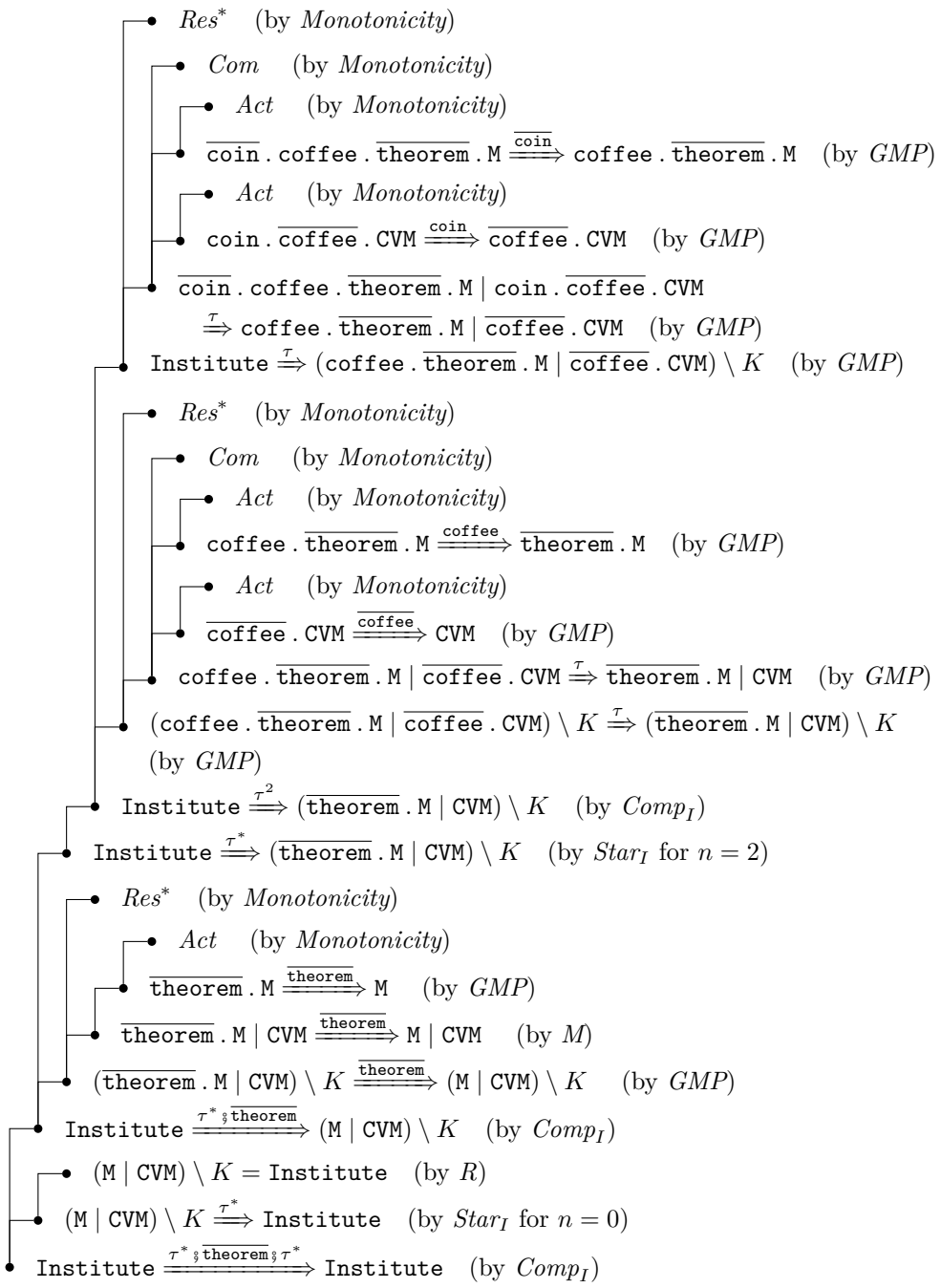
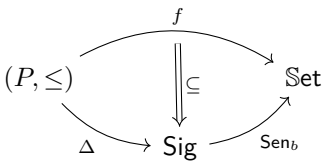


Figure 1 Proof tree for the continuous output of theorems by the process `Institute`.

4 Forcing

In this section, we develop a forcing technique for proving completeness which extends in a non-straightforward way the classical forcing from one signature to a category of signatures.

▶ **Definition 17** (Forcing property). *A forcing property is a tuple $\mathbb{P} = (P, \leq, \Delta, f)$, where:*



1. *$(P, \leq)$ is a partially ordered set with a least element 0.*
   The elements of P are traditionally called conditions.
2. *$\Delta : (P, \leq) \to \mathsf{Sig}$ is a functor, which maps each arrow $(p \leq q) \in (P, \leq)$ to an inclusion $\Delta_p \subseteq \Delta_q$.*
3. *$f : (P, \leq) \to \mathbb{S}\mathsf{et}$ is a functor from the small category $(P, \leq)$ to the category of sets $\mathbb{S}\mathsf{et}$ such that $f \subseteq \Delta; \mathsf{Sen}_b$ is a natural transformation, that is: (a) $f(p) \subseteq \mathsf{Sen}_b(\Delta_p)$ for all conditions $p \in P$, and (b) $f(p) \subseteq f(q)$ for all arrows $(p \leq q) \in (P, \leq)$.*
4. *If $f(p) \models \phi$ then $\phi \in f(q)$ for some $q \geq p$, for all atoms $\phi \in \mathsf{Sen}_b(\Delta_p)$.*

A classical forcing property is a particular case of forcing property such that $\Delta_p = \Delta_q$ for all conditions $p, q \in P$.

▶ **Example 18** (Syntactic forcing). Let Σ be a base signature and C an S-sorted set of new constants such that $\mathsf{card}(C_s) = \omega$ for all $s \in S$. Let $\mathbb{P} = (P, \leq, \Delta, f)$ be a forcing property defined as follows:

- P is the set of presentations of the form $p = (\Delta_p, \Gamma_p)$, where (a) Δ_p is obtained from Σ by adding a finite set C_p of constants from C, and (b) $\Gamma_p \subseteq \mathsf{Sen}(\Delta_p)$ is consistent, that is, $\Gamma_p \nvdash_{\Delta_p} \bot$.
- $p \leq q$ iff $\Delta_p \subseteq \Delta_q$ and $\Gamma_p \subseteq \Gamma_q$, for all conditions $p, q \in P$.
- Δ is the forgetful functor which maps each condition $p \in P$ to Δ_p.
- $f(p) = \Gamma_p \cap \mathsf{Sen}_b(\Delta_p)$, for all conditions $p \in P$.

The syntactic forcing described in the example above is used to prove completeness. The constants from C are traditionally called Henkin constants and are used as witnesses for existentially quantified sentences obtained by extending an initial theory to a maximally consistent set of sentences.

As usual, forcing properties determine suitable relations between conditions and sentences.

▶ **Definition 19** (Forcing relation). *Let $\mathbb{P} = \langle P, \leq, \Delta, f \rangle$ be a forcing property.* The forcing relation $\Vdash$ *between conditions $p \in P$ and sentences from $\mathsf{Sen}(\Delta_p)$ is defined by induction on the structure of sentences, as follows:*

- *$p \Vdash \varphi$ if $\varphi \in f(p)$, for all atomic sentences $\varphi \in \mathsf{Sen}_b(\Delta_p)$.*
- *$p \Vdash t_1 \overset{\mathfrak{a}_1 ; \mathfrak{a}_2}{\Longrightarrow} t_2$ if $p \Vdash t_1 \overset{\mathfrak{a}_1}{\Longrightarrow} t$ and $p \Vdash t \overset{\mathfrak{a}_2}{\Longrightarrow} t_2$ for some $t \in T_{\Delta_p}$.*
- *$p \Vdash t_1 \overset{\mathfrak{a}_1 \cup \mathfrak{a}_2}{\Longrightarrow} t_2$ if $p \Vdash t_1 \overset{\mathfrak{a}_1}{\Longrightarrow} t_2$ or $p \Vdash t_1 \overset{\mathfrak{a}_2}{\Longrightarrow} t_2$.*
- *$p \Vdash t_1 \overset{\mathfrak{a}^*}{\Longrightarrow} t_2$ if $p \Vdash t_1 \overset{\mathfrak{a}^n}{\Longrightarrow} t_2$ for some natural number $n \in \omega$.*
- *$p \Vdash \neg\phi$ if there is no $q \geq p$ such that $q \Vdash \phi$.*
- *$p \Vdash \vee\Phi$ if $p \Vdash \phi$ for some $\phi \in \Phi$.*
- *$p \Vdash \exists X \cdot \phi$ if $p \Vdash \theta(\phi)$ for some substitution $\theta : X \to T_{\Delta_p}$.*

The relation $p \Vdash \phi$ in $\mathbb{P}$, is read as p forces ϕ. We say that p weakly forces ϕ, in symbols, $p \Vdash^w \phi$, if $p \Vdash \neg\neg\phi$.

A few basic properties of forcing are presented below.

▶ **Lemma 20** (Forcing properties). *Let $\mathbb{P} = (P, \leq, \Delta, f)$ be a forcing property. For all conditions $p \in P$ and all sentences $\phi \in \mathsf{Sen}(\Delta_p)$ we have:*

1. *$p \Vdash \neg\neg\phi$ iff for each $q \geq p$ there is a condition $r \geq q$ such that $r \Vdash \phi$.*
2. *If $p \leq q$ and $p \Vdash \phi$ then $q \Vdash \phi$.*
3. *If $p \Vdash \phi$ then $p \Vdash \neg\neg\phi$.*
4. *We can not have both $p \Vdash \phi$ and $p \Vdash \neg\phi$.*

The second property stated in the above lemma shows that the forcing relation is preserved along inclusions of conditions. The fourth property shows that the forcing relation is consistent, that is, a condition cannot force all sentences. The remaining conditions are about negation.

▶ **Definition 21** (Generic set). *Let $\mathbb{P} = (P, \leq, \Delta, f)$ be a forcing property. A subset of conditions $G \subseteq P$ is generic if*

1. *G is an ideal, that is: (a) for all $p \in G$ and all $q \leq p$ we have $q \in G$, and (b) for all $p, q \in G$ there exists $r \in G$ such that $p \leq r$ and $q \leq r$; and*
2. *for all conditions $p \in G$ and all sentences $\phi \in \mathsf{Sen}(\Delta_p)$ there exists a condition $q \in G$ such that $q \geq p$ and either $q \Vdash \phi$ or $q \Vdash \neg\phi$ holds.*

We write $G \Vdash \phi$ if $p \Vdash \phi$ for some $p \in G$.

A generic set G describes a reachable model which satisfies all sentences forced by the conditions in G.

▶ **Lemma 22** (Existence). *Let $\mathbb{P} = (P, \leq, \Delta, f)$ be a forcing property. If any signature in $\{\Delta_p\}_{p \in P}$ is countable then every $p \in P$ belongs to a generic set.*

Proof sketch. Let $\text{pair} : \omega \times \omega \to \omega$ be a bijective function defined by $\text{pair}(i, j) := \big((i + j)(i + j + 1) + 2j\big)/2$ for all $i, j \in \omega$. For all conditions $p \in P$, let $\psi_p : \omega \to \mathsf{Sen}(\Delta_p)$ be a bijective mapping, which gives an enumeration of $\mathsf{Sen}(\Delta_p)$. We define an increasing chain of conditions $p_0 \leq p_1 \leq \ldots$ in P recursively. Let $p = p_0$. For the induction step, we assume that we have defined p_n and we define p_{n+1}. Notice that there are unique natural numbers $i, j \in \omega$ such that $n = \text{pair}(i, j)$ and $i, j \leq n$.

- If there is $q \geq p_n$ such that $q \Vdash \psi_{p_i}(j)$, then let $p_{n+1} := q$.
- Otherwise, $p_{n+1} := p_n$, which means that $p_{n+1} \Vdash \neg\psi_{p_i}(j)$.

Then $G := \{q \in P \mid q \leq p_n \text{ for some } n \in \omega\}$ is generic and contains p. ◀

Lemma 22 is the key for a modular approach to forcing and it is the equivalent of the Lindenbaum's lemma from Henkin's method for proving completeness.

▶ Remark 23. Since $\Delta : (G, \leq) \to \mathsf{Sig}$ is a directed diagram of signature inclusions, one can construct a co-limit $\mu : \Delta \Rightarrow \Delta_G$ of the functor $\Delta : (G, \leq) \to \mathsf{Sig}$ such that $\mu_p : \Delta_p \to \Delta_G$ is an inclusion for all $p \in G$.

The results which leads to completeness are developed over the signature Δ_G. If $\mathbb{P}$ is the syntactic forcing described in Example 18, then Δ_G is obtain from the base signature Σ by adding all Henkin constants from $\{\Delta_p\}_{p \in G}$. In general, Δ_G does not contain all Henkin constants from C, which is one of the major differences between the classical approach and the present developments.

▶ **Definition 24** (Generic model). *Let $\mathbb{P} = (P, \leq, \Delta, f)$ be a forcing property and $G \subseteq P$ a generic set. A model $\mathfrak{A}$ defined over Δ_G is a* generic model *for G iff for every sentence $\phi \in \bigcup_{p \in G} \mathsf{Sen}(\Delta_p)$, we have $\mathfrak{A} \models \phi$ iff $G \Vdash \phi$.*

The notion of generic model is the semantic counterpart of the definition of generic set. The following result shows that every generic set has a generic model.

▶ **Theorem 25** (Generic Model Theorem). *Let $\mathbb{P} = (P, \leq, \Delta, f)$ be a forcing property and $G \subseteq P$ a generic set. Then there is a generic model $\mathfrak{A}$ for G which is countable and reachable.*

Proof sketch. We define the set of all atomic sentences $B \coloneqq \{\phi \in \mathsf{Sen}_b(\Delta_G) \mid G \Vdash \phi\}$ forced by the generic set G. The basic model $\mathfrak{A}^B$ given by Lemma 12 is the generic model for G. ◀

5 Completeness

The logical framework in which the results are developed in this section is the fragment TA^c obtained from TA by restricting the syntax to at most countable signatures. The syntactic forcing property defined in Example 18 is the starting point for proving completeness. Therefore, throughout this section, we let $\mathbb{P} = (P, \leq, \Delta, f)$ be a syntactic forcing property in TA^c as described in Example 18. In particular, all signatures in $\{\Delta\}_{p \in P}$ are at most countable. For the sake of simplicity, we write $p \vdash \phi$ iff $\Gamma_p \vdash_{\Delta_p} \phi$, for all conditions $p = (\Delta_p, \Gamma_p)$ in P.

▶ **Theorem 26.** *For all $p \in P$ and all $\phi \in \mathsf{Sen}(\Delta_p)$, we have $p \Vdash^w \phi$ iff $p \vdash \phi$.*

The above theorem says that a sentence is entailed by a condition if and only if it is weakly forced by that condition. In other words, the entailment relation is the weak forcing relation. Now, we can interpret Lemma 22 in the present context given by the syntactic forcing property $\mathbb{P}$ set above. The following result is a direct consequence of Theorem 26 and Lemma 22.

▶ **Corollary 27** (Lindenbaum's lemma). *Assume the following:*
- *a condition $p^\circ = (\Delta_{p^\circ}, \Gamma_{p^\circ})$ from P, and*
- *a generic set G which contains p° (by Lemma 22).*

Let Δ_G be the vertex of the co-limit $\mu : \Delta \to \Delta_G$ of the functor $\Delta : (G, \leq) \to \mathsf{Sig}$ defined in Remark 23. Then $\Gamma_G = \bigcup_{p \in G} \Gamma_p$ is a maximally consistent set which includes Γ_{p°.

The following example shows that Δ_G does not contain all Henkin constants defined for the base signature.

▶ **Example 28.** Let Σ be the signature defined by: $S \coloneqq \{s_i \mid i \in \omega\}$, $F \coloneqq \{c :\to s_0, d :\to s_0\}$, $M \coloneqq \emptyset$, and $P \coloneqq \{\lambda\}$. Let Γ be the set of sentences over Σ which consists of: (a) $c \overset{\lambda^*}{\Longrightarrow} d$, and (b) $(\exists x_n \cdot \top) \to \neg(c \overset{\lambda^n}{\Longrightarrow} d)$ for all $n \in \omega$, where x_n is a variable of sort s_n.

The first sentence says that there is a transition from c to d in a finite number of steps. For each natural number n, the sentence $(\exists x_n \cdot \top) \to \neg(c \overset{\lambda^n}{\Longrightarrow} d)$ says that if the sort s_n is not empty then there is no transition from c to d in exactly n steps. Recall that $C = \{C_{s_n}\}_{n \in \omega}$ is the set of all Henkin constants, and $\mathsf{card}(C_{s_n}) = \omega$ for all natural numbers n. Notice that $p^\circ = (\Sigma, \Gamma)$ is consistent, but $q = (\Sigma[C], \Gamma)$ is not consistent. By Corollary 27, one can extend Γ to a maximally consistent set of sentences Γ_G. Unlike in classical first-order logic, Γ_G does not contain all Henkin constants from C.

▶ **Theorem 29** (Downwards Löwenheim-Skolem Theorem). *For any consistent set of sentences Γ defined over a countable signature Σ, there exists a countable Σ-model $\mathfrak{A}$ that satisfies Γ.*

Proof. Let $\mathbb{P} = (P, \leq, \Delta, f)$ be the forcing property described in Definition 17. Notice that $p := (\Sigma, \Gamma)$ is a condition from P. Since all signatures are countable, by Lemma 22, p belongs to a generic set G. By Theorem 25, G has a generic model $\mathfrak{B}$ which is countable and reachable. In particular, $\mathfrak{B} \models_{\Delta_G} \Gamma$. Let $\mathfrak{A} := \mathfrak{B}\restriction_{\Sigma}$, and by the satisfaction condition, $\mathfrak{A} \models_{\Sigma} \Gamma$. ◀

▶ **Theorem 30** (Completeness). *For all sets of sentences Γ and all sentences ϕ defined over a countable signature Σ, we have: $\Gamma \vdash_{\Sigma} \phi$ iff $\Gamma \models_{\Sigma} \phi$.*

Proof. The forward implication holds because all proof rules are sound. For the backwards implication assume $\Gamma \nvdash_{\Sigma} \phi$. We have $\Gamma \cup \{\neg\phi\} \nvdash_{\Sigma} \bot$. By Theorem 29, there is a countable Σ-model $\mathfrak{A}$ such that $\mathfrak{A} \models_{\Sigma} \Gamma \cup \{\neg\phi\}$. Therefore, $\Gamma \not\models_{\Sigma} \phi$. ◀

6 Conclusions

In this study, we have defined an extension of many-sorted first-order logic, called transition algebra, that offers explicit support for state transitions; furthermore, we have investigated its logical properties in order to apply the institutional model theory approach to new algebraic specification languages based on this logic, and with a greater expressivity than Maude and CafeOBJ. Transition algebra satisfies desirable properties such as truth invariance under change of signature, and has an expressive power that goes beyond that of ordinary first-order logic, which is important for formal-verification purposes. Our efforts have focused on two main aspects of transition algebra: first, on its formal-specification capabilities, i.e., to show that it forms a proper extension of first-order equational logic; and second, on support for formal verification, for which we have studied a number of model-theoretic properties, syntactic entailment and, most importantly, soundness and completeness results.

Concerning its formal-specification capabilities, transition algebra blends features of dynamic logic with features of many-sorted first-order logic. From the former, it borrows the idea of expressing the dynamics of a system by means of actions, which are built from atomic transitions using composition, iteration, and so on. The iteration of an action is a key feature because it allows us to express reachability, which is not possible in ordinary first-order logic. From the latter, our logic borrows term-building operators and quantifiers. This allows us to capture system states as terms, and hence to reason about the structure of states more freely and in a more complex manner than it is possible in dynamic logic.

For verification purposes, our contribution is twofold: on one hand, we have introduced a sound proof system for transition algebra; and on the other hand, we have developed a new general method for proving completeness based on forcing. The latter is highly important, because it has enabled us to circumvent the lack of compactness of transition algebra, which prevents the use of readily available methods for proving completeness. Moreover, it also overcomes a significant limitation of existing forcing techniques, namely their reliance on models with non-empty carriers, which is another basic property (like compactness) that does not hold for transition algebra. We have demonstrated the use of this extended forcing technique to show that the proof system for transition algebra is complete. We aim to further develop and apply this technique to extensions of transition algebra that take into account, for example, subsorting – to which we have already alluded in this paper. Furthermore, future research includes applying forcing to prove omitting types theorem for logical systems that interpret sorts as sets, possibly empty, thus upgrading the results from [17, 20]. Subsequently, the application of omitting types theorem to Robinson consistency property and interpolation, as demonstrated in [19], remains a feasible avenue for exploration.

References

1 Egidio Astesiano, Michel Bidoit, Hélène Kirchner, Bernd Krieg-Brückner, Peter D. Mosses, Donald Sannella, and Andrzej Tarlecki. CASL: The common algebraic specification language. *Theoretical Computer Science*, 286(2):153–196, 2002. doi:10.1016/S0304-3975(01)00368-1.

2 Tomasz Borzyszkowski. Generalized interpolation in CASL. *Information Processing Letters*, 76(1-2):19–24, 2000. doi:10.1016/S0020-0190(00)00120-4.

3 Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott, editors. *All About Maude – A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *LNCS*. Springer, 2007. doi:10.1007/978-3-540-71999-1.

4 Mihai Codescu, Till Mossakowski, Adrián Riesco, and Christian Maeder. Integrating Maude into Hets. In Michael Johnson and Dusko Pavlovic, editors, *Algebraic Methodology and Software Technology - 13th International Conference, AMAST 2010, Lac-Beauport, QC, Canada, June 23-25, 2010. Revised Selected Papers*, volume 6486 of *LNCS*, pages 60–75. Springer, 2010. doi:10.1007/978-3-642-17796-5_4.

5 Paul J. Cohen. The independence of the continuum hypothesis. *Proceedings of the National Academy of Sciences of the United States of America*, 50(6):1143–1148, December 1963. doi:10.1073/pnas.50.6.1143.

6 Paul J. Cohen. The independence of the continuum hypothesis, II. *Proceedings of the National Academy of Sciences of the United States of America*, 51(1):105–110, January 1964. doi:10.1073/pnas.51.1.105.

7 William Craig. Linear reasoning. A new form of the Herbrand-Gentzen theorem. *Journal of Symbolic Logic*, 22(3):250–268, 1957. doi:10.2307/2963593.

8 Pierpaolo Degano, Fabio Gadducci, and Corrado Priami. A causal semantics for CCS via rewriting logic. *Theoretical Computer Science*, 275(1-2):259–282, 2002. doi:10.1016/S0304-3975(01)00165-7.

9 Răzvan Diaconescu. *Institution-Independent Model Theory*. Studies in Universal Logic. Birkhäuser, 2008.

10 Răzvan Diaconescu. Three decades of institution theory. In Jean-Yves Béziau, editor, *Universal Logic: An Anthology*, pages 309–322. Springer, 2012. doi:10.1007/978-3-0346-0145-0_25.

11 Răzvan Diaconescu and Kokichi Futatsugi. Logical foundations of CafeOBJ. *Theoretical Computer Science*, 285(2):289–318, 2002. doi:10.1016/S0304-3975(01)00361-9.

12 Steven Eker, Narciso Martí-Oliet, José Meseguer, Rubén Rubio, and Alberto Verdejo. The Maude strategy language. *Journal of Logical and Algebraic Methods in Programming*, 134:100887, 2023. doi:10.1016/j.jlamp.2023.100887.

13 Joseph Goguen and José Meseguer. Completeness of many-sorted equational logic. *ACM SIGPLAN Notices*, 17(1):9–17, 1982. doi:10.1145/947886.947887.

14 Joseph Goguen and José Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992. doi:10.1016/0304-3975(92)90302-V.

15 Joseph Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. *Introducing OBJ*, pages 3–167. Springer, 2000. doi:10.1007/978-1-4757-6541-0_1.

16 Joseph A. Goguen and Rod M. Burstall. Institutions: Abstract model theory for specification and programming. *J. ACM*, 39(1):95–146, 1992. doi:10.1145/147508.147524.

17 Daniel Găină. Forcing, downward Löwenheim-Skolem and omitting types theorems, institutionally. *Logica Universalis*, 8(3-4):469–498, 2014. doi:10.1007/S11787-013-0090-0.

18 Daniel Găină. Forcing and calculi for hybrid logics. *Journal of the Association for Computing Machinery*, 67(4):1–55, 2020. doi:10.1145/3400294.

19 Daniel Găină, Guillermo Badia, and Tomasz Kowalski. Robinson consistency in many-sorted hybrid first-order logics. In David Fernández-Duque, Alessandra Palmigiano, and Sophie Pinchinat, editors, *Advances in Modal Logic, AiML 2022, Rennes, France, August 22-25, 2022*, pages 407–428. College Publications, 2022. URL: http://www.aiml.net/volumes/volume14/25-Gaina-Badia-Kowalski.pdf.

20 Daniel Găină, Guillermo Badia, and Tomasz Kowalski. Omitting types theorem in hybrid dynamic first-order logic with rigid symbols. *Annals of Pure and Applied Logic*, 174(3):103212, 2023. doi:10.1016/J.APAL.2022.103212.

21 Daniel Găină and Marius Petria. Completeness by forcing. *Journal of Logic and Computation*, 20(6):1165–1186, 2010. doi:10.1093/LOGCOM/EXQ012.

22 Daniel Găină and Andrei Popescu. An institution-independent proof of the Robinson consistency theorem. *Studia Logica*, 85(1):41–73, 2007. doi:10.1007/S11225-007-9022-4.

23 Go Hashimoto, Daniel Găină, and Ionuţ Ţuţu. Forcing, transition algebras, and calculi (extended version), 2024. arXiv:2404.16111.

24 Narciso Martí-Oliet and José Meseguer. Rewriting logic as a logical and semantic framework. In José Meseguer, editor, *First International Workshop on Rewriting Logic and its Applications, RWLW 1996, Asilomar Conference Center, Pacific Grove, CA, USA, September 3-6, 1996*, volume 4 of *Electronic Notes in Theoretical Computer Science*, pages 190–225. Elsevier, 1996. doi:10.1016/S1571-0661(04)00040-4.

25 José Meseguer. Conditional rewriting logic: Deduction, models and concurrency. In Stéphane Kaplan and Mitsuhiro Okada, editors, *Conditional and Typed Rewriting Systems, 2nd International CTRS Workshop, Montreal, Canada, June 11-14, 1990, Proceedings*, volume 516 of *LNCS*, pages 64–91. Springer, 1990. doi:10.1007/3-540-54317-1_81.

26 Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980. doi:10.1007/3-540-10235-3.

27 Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.

28 J. Donald Monk. *Mathematical Logic*, volume 37 of *Graduate Texts in Mathematics*. Springer-Verlag New York, 1976.

29 Marius Petria. An institutional version of Gödel's completeness theorem. In Till Mossakowski, Ugo Montanari, and Magne Haveraaen, editors, *Algebra and Coalgebra in Computer Science, Second International Conference, CALCO 2007, Bergen, Norway, August 20-24, 2007, Proceedings*, volume 4624 of *LNCS*, pages 409–424. Springer, 2007. doi:10.1007/978-3-540-73859-6_28.

30 Abraham Robinson. Forcing in model theory. *Symposia Mathematica*, 5:69–82, 1971.

31 Donald Sannella and Andrzej Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2012. doi:10.1007/978-3-642-17336-3.

32 Bruce Schechter. *My Brain is Open: The Mathematical Journeys of Paul Erdös*. Simon & Schuster, 2000.

33 Andrzej Tarlecki. Bits and pieces of the theory of institutions. In Klaus Drosten, Hans-Dieter Ehrich, Martin Gogolla, and Udo W. Lipeck, editors, *Proceedings of the 4st Workshop on Abstract Data Type, 1986. University of Braunschweig, Germany*, 1986.

34 Alberto Verdejo and Narciso Martí-Oliet. Implementing CCS in Maude 2. In Fabio Gadducci and Ugo Montanari, editors, *Fourth International Workshop on Rewriting logic and Its Applications, WRLA2002, Pisa, Italy, 19-21, 2002*, volume 71 of *Electronic Notes in Theoretical Computer Science*, pages 282–300. Elsevier, 2002. doi:10.1016/S1571-0661(05)82540-X.

On Transcendence of Numbers Related to Sturmian and Arnoux-Rauzy Words

Pavol Kebis
Department of Computer Science, University of Oxford, UK

Florian Luca
Mathematics Division, Stellenbosch University, Stellenbosch, South Africa

Joël Ouaknine
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Andrew Scoones
Department of Computer Science, University of Oxford, UK

James Worrell
Department of Computer Science, University of Oxford, UK

Abstract

We consider numbers of the form $S_\beta(\boldsymbol{u}) := \sum_{n=0}^{\infty} \frac{u_n}{\beta^n}$, where $\boldsymbol{u} = \langle u_n \rangle_{n=0}^{\infty}$ is an infinite word over a finite alphabet and $\beta \in \mathbb{C}$ satisfies $|\beta| > 1$. Our main contribution is to present a combinatorial criterion on $\boldsymbol{u}$, called echoing, that implies that $S_\beta(\boldsymbol{u})$ is transcendental whenever β is algebraic. We show that every Sturmian word is echoing, as is the Tribonacci word, a leading example of an Arnoux-Rauzy word. We furthermore characterise $\overline{\mathbb{Q}}$-linear independence of sets of the form $\{1, S_\beta(\boldsymbol{u}_1), \ldots, S_\beta(\boldsymbol{u}_k)\}$, where $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$ are Sturmian words having the same slope. Finally, we give an application of the above linear independence criterion to the theory of dynamical systems, showing that for a contracted rotation on the unit circle with algebraic slope, its limit set is either finite or consists exclusively of transcendental elements other than its endpoints 0 and 1. This confirms a conjecture of Bugeaud, Kim, Laurent, and Nogueira.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Computing methodologies → Algebraic algorithms

Keywords and phrases Transcendence, Subspace Theorem, Fibonacci Word, Tribonacci Word

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.144

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding *Joël Ouaknine*: Also affiliated with Keble College, Oxford as emmy.network Fellow, and supported by DFG grant 389792660 as part of TRR 248 (see `https://perspicuous-computing.science`).
James Worrell: Work supported by UKRI Frontier Research Grant EP/X033813/1.

1 Introduction

A well-known conjecture of Hartmanis and Stearns asserts that for an integer $b \geq 2$ and a sequence $\boldsymbol{u} \in \{0, \ldots, b-1\}^\omega$ that is computable by linear-time Turing machine (in the sense that given input n in unary, the machine outputs the first n elements of $\boldsymbol{u}$ in time $O(n)$), the number $S_b(\boldsymbol{u}) := \sum_{n=0}^{\infty} \frac{u_n}{b^n}$ is either rational or transcendental. This conjecture remains open and is considered to be very difficult [4]. Among many other consequences, the conjecture implies that integer multiplication cannot be done in linear time [7].

A weaker version of the Hartmanis-Stearns conjecture was formulated in 1968 by Cobham, who conjectured that every irrational automatic number is transcendental [9]. In other words, if $b \geq 2$ is an integer and $\boldsymbol{u}$ is an automatic word, then the number $S_b(\boldsymbol{u})$ is either rational

or transcendental. Note that every automatic word is morphic, and that morphic words are precisely those that can be generated by so-called tag machines, a restricted class of linear-time Turing machines [4]. The transcendence of irrational automatic numbers over an integer base was proven in 2004 by Adamczewski, Bugeaud, and Luca [3]. It is noted in [1] that extending this result from the class of automatic words to the more general class of morphic words (specifically, for those generated by morphisms with polynomial growth) encompasses recognised open problems in transcendence theory. In another direction, Adamczewski and Faverjon [5] proved a generalisation of Cobham's conjecture to algebraic number bases. Their result entails that for an algebraic number β, with $|\beta| > 1$ and automatic sequence $\boldsymbol{u}$ over a finite alphabet $\{0, 1, \ldots, k-1\}$, the number $S_\beta(\boldsymbol{u})$ either lies in the field $\mathbb{Q}(\beta)$ or is transcendental.

Closely connected with the class of morphic words, one has Sturmian words and, more generally, Arnoux-Rauzy words [6]. A Sturmian word is an infinite word over the alphabet $\{0, 1\}$ that has $n + 1$ factors of length n for all $n \in \mathbb{N}$. In terms of factor complexity, Sturmian words are thereby the simplest non-periodic infinite words. Arnoux-Rauzy words are a generalisation of Sturmian words to the alphabet $\{0, 1, \ldots, k-1\}$ for arbitrary k. Among other properties, an Arnoux-Rauzy word on a k-letter alphabet has factor complexity $(k-1)n+1$. We refer to [6, Definition 3.3] for the precise definition. Perhaps the best-known example of a Sturmian word is the Fibonacci word, while the best-known example of an Arnoux-Rauzy word that is not Sturmian is the Tribonacci word. It so happens that both these words are morphic, although not automatic. The Fibonacci word is the fixed point of the morphism $0 \mapsto 01, 1 \mapsto 0$, while the Tribonacci word is the fixed point of the morphism $0 \mapsto 01, 1 \mapsto 02, 2 \mapsto 0$. More generally, Arnoux-Rauzy words can be generated by iterating a finite set of morphisms via so-called S-adic generation. Sturmian and Arnoux-Rauzy words are also intimately connected with dynamical systems. In their pioneering work [20, 21], Morse and Hedlund showed that every Sturmian word arises as the coding of a translation of the one-dimensional torus and, following the work of Rauzy [23], a subclass of Arnoux-Rauzy words can be realised as natural codings of toral translations in higher dimension [6].

There is an extensive literature on transcendence of Sturmian and Arnoux-Rauzy words over an integer base $b \geq 2$. Danilov [10] proved the transcendence of $S_b(\boldsymbol{u})$ for $\boldsymbol{u}$ the Fibonacci word. This result was significantly strengthened by Ferenczi and Mauduit [11], who proved the transcendence of $S_b(\boldsymbol{u})$, for $\boldsymbol{u}$ either a Sturmian word or an Arnoux-Rauzy word on alphabet $\{0, 1, 2\}$. This result was extended to Arnoux-Rauzy words over alphabet $\{0, 1, \ldots, k-1\}$ for any k in [24]. Meanwhile, Bugeaud *et al.* [8] showed the $\overline{\mathbb{Q}}$-linear independence of sets of the form $\{1, S_b(\boldsymbol{u}_1), S_b(\boldsymbol{u}_2)\}$, where $\boldsymbol{u}_1, \boldsymbol{u}_2$ are Sturmian words having the same slope (where the slope of a Sturmian word is the limiting frequency of 1's, which always exists).

Our interest in this paper is in proving transcendence results for Sturmian and Arnoux-Rauzy words over an algebraic-number base β, with $|\beta| > 1$. Here the picture is less complete compared to the case that β is an integer. Laurent and Nogueira [14] observe that if $\boldsymbol{u}$ is a *characteristic* Sturmian word (cf. Section 3.3), then the transcendence of $S_\beta(\boldsymbol{u})$ follows from a classical result of Loxton and Van der Poorten [17, Theorem 7] concerning transcendence of Hecke-Mahler series. For $\boldsymbol{u}$ having linear subword complexity (which includes all Arnoux-Rauzy words), it follows from [2, Theorem 1] that $S_\beta(\boldsymbol{u})$ is either transcendental or lies in the field $\mathbb{Q}(\beta)$, subject to a non-trivial inequality between the height of β and a combinatorial parameter of $\boldsymbol{u}$ called the *Diophantine exponent*. Most closely related to the present work, recently [18], introduced a criterion that can be used to show transcendence of $S_\beta(\boldsymbol{u})$ for a Sturmian word $\boldsymbol{u}$ and any β.

The main contribution of the present paper is to give a new combinatorial criterion on an infinite word $\boldsymbol{u}$, called *echoing*, that implies that $S_\beta(\boldsymbol{u}) := \sum_{n=0}^{\infty} \frac{u_n}{\beta^n}$ is transcendental for any algebraic number β. The echoing condition is an evolution of the above-mentioned criterion of [18] that allows both handling certain Arnoux-Rauzy words over non-binary alphabets as well as giving a considerably simplified treatment of Sturmian words. To illustrate the utility of the echoing notion, we show that every Sturmian word is echoing, as is the Tribonacci word, a leading example of an Arnoux-Rauzy word. We anticipate that the notion will find further applications among Arnoux-Rauzy words, and note that the thesis [13] contains further examples of such that are echoing. We also employ the echoing condition to give sufficient and necessary conditions for the $\overline{\mathbb{Q}}$-linear independence of a set of Sturmian numbers $\{1, S_\beta(\boldsymbol{u}_1), \ldots, S_\beta(\boldsymbol{u}_k)\}$, where $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$, are Sturmian words that have the same slope.

In Section 7 we give an application of our results to the theory of dynamical systems. We consider the set C of limit points of a contracted rotation f on the unit interval, where f is assumed to have an algebraic contraction factor. The set C is finite if f has a periodic orbit and is otherwise a Cantor set, that is, it is homeomorphic to the Cantor ternary set (equivalently, it is compact, nowhere dense, and has no isolated points). In the latter case we show that all elements of C except its endpoints 0 and 1 are transcendental. Our result confirms a conjecture of Bugeaud, Kim, Laurent, and Nogueira, who proved a special case of this result in [8]. We remark that it is a longstanding open question whether the actual Cantor ternary set contains any irrational algebraic elements.

2 Preliminaries

This section contains some number-theoretic preliminaries that will be used in Section 6.

For functions f and g, we use the Vinogradov notation $f \ll g$ to mean $f = O(g)$.

Let K be a number field of degree d and let $M(K)$ be the set of *places* of K. We divide $M(K)$ into the collection of *infinite places*, which are determined either by an embedding of K in $\mathbb{R}$ or a complex-conjugate pair of embeddings of K in $\mathbb{C}$, and the set of *finite places*, which are determined by prime ideals in the ring $\mathcal{O}_K$ of integers of K.

For $x \in K$ and $v \in M(K)$, define the absolute value $|x|_v$ as follows: $|x|_v := |\sigma(x)|^{1/d}$ in case v corresponds to a real embedding $\sigma : K \to \mathbb{R}$; $|x|_v := |\sigma(x)|^{2/d}$ in case v corresponds to a complex-conjugate pair of embeddings $\sigma, \overline{\sigma} : K \to \mathbb{C}$; finally, $|x|_v := N(\mathfrak{p})^{-\mathrm{ord}_\mathfrak{p}(x)/d}$ if v corresponds to a prime ideal $\mathfrak{p}$ in $\mathcal{O}$ and $\mathrm{ord}_\mathfrak{p}(x)$ is the order of $\mathfrak{p}$ as a divisor of the ideal $x\mathcal{O}$. With the above definitions we have the *product formula*: $\prod_{v \in M(K)} |x|_v = 1$ for all $x \in K^*$. Given a set of places $S \subseteq M(K)$, the ring $\mathcal{O}_S$ of *S-integers* is the subring comprising all $x \in K$ such $|x|_v \leq 1$ for all finite places $v \in S$.

For $m \geq 2$ the *absolute Weil height* of $\boldsymbol{x} = (x_1, \ldots, x_m) \in K^m$ is defined to be

$$H(\boldsymbol{x}) := \prod_{v \in M(K)} \max(|x_1|_v, \ldots, |x_m|_v)\,.$$

This definition is independent of the choice of field K containing $x_1, \ldots, x_m$. Note the restriction $m \geq 2$ in the above definition. For $x \in K$ we define its height $H(x)$ to be $H(1, x)$. For a non-zero polynomial $f = \sum_{i=0}^{s} a_i X^i \in K[X]$, where $s \geq 1$, we define its height $H(f)$ to be the height of its coefficient vector $(a_0, \ldots, a_s)$.

The following special case of the p-adic Subspace Theorem of Schlickewei is one of the main ingredients of our approach.

▶ **Theorem 1.** *Let $S \subseteq M(K)$ be a finite set of places of K that contains all infinite places. Let $v_0 \in S$ be a distinguished place. Given $m \geq 2$, let $L(x_1, \ldots, x_m)$ be a linear form with algebraic coefficients and let $i_0 \in \{1, \ldots, m\}$. Then for any $\varepsilon > 0$ the set of solutions $\boldsymbol{a} = (a_1, \ldots, a_m) \in (\mathcal{O}_S)^m$ of the inequality*

$$|L(\boldsymbol{a})|_{v_0} \cdot \left(\prod_{\substack{(i,v) \in \{1,\ldots,m\} \times S \\ (i,v) \neq (i_0,v_0)}} |a_i|_v \right) \leq H(\boldsymbol{a})^{-\varepsilon}$$

is contained in a finite union of proper linear subspaces of K^m.

We will also need the following additional proposition about roots of univariate polynomials.

▶ **Proposition 2** ([15, Proposition 2.3])**.** *Let $f \in K[X]$ be a polynomial with at most $k+1$ terms. Assume that f can be written as the sum of two polynomials g and h, where every monomial of g has degree at most d_0 and every monomial of h has degree at least d_1. Let β be a root of f that is not a root of unity. If $d_1 - d_0 > \frac{\log(k\,H(f))}{\log H(\beta)}$ then β is a common root of g and h.*

3 Echoing Words

In this section we present the main definition of the paper, the notion of echoing word. Before we present this, by way of motivation we present an informal analysis of the periodicity properties of the Fibonacci and Tribonacci words.

3.1 The Fibonacci Word

Let $\Sigma = \{0, 1\}$ and consider the morphism $\sigma : \Sigma^* \to \Sigma^*$ given by $\sigma(0) = 01$ and $\sigma(1) = 0$. The *Fibonacci word* $F_\infty \in \Sigma^\omega$ is the morphic word

$$F_\infty := \lim_{n \to \infty} \sigma^n(0) = 01001010010010100 \ldots .$$

In more detail, the Fibonacci word F_∞ is the limit of the sequence of finite words $(F_n)_{n=0}^\infty$ given by $F_n = \sigma^n(0)$ for all n (observe that F_n is a prefix of F_{n+1} for all n, so the limit is well defined). Note that the sequence $(F_n)_{n=0}^\infty$ satisfies the recurrence

$$F_n = F_{n-1}F_{n-2} \quad (n \geq 2)$$

analogous to that satisfied by the sequence of Fibonacci numbers.

The Fibonacci word is not periodic and hence F_∞ is not equal to any its tails $\mathrm{tl}^n(F_\infty)$ for $n > 0$. However, if the shift n is judiciously chosen then, intuitively speaking, the mismatches between F_∞ and $\mathrm{tl}^n(F_\infty)$ are *few and far between*. This intuition will be formalised in the definition of echoing word. It turns out that a particularly good choice of shifts is to take them from the sequence $\langle 1, 2, 3, 5, 8 \ldots \rangle$ of Fibonacci numbers: for example, juxtaposing F_∞ and $\mathrm{tl}^5(F_\infty)$ and writing mismatches in bold we see:

$$\begin{aligned} F_\infty &:= 0100101\mathbf{10}0100101001010\mathbf{10}0100101\mathbf{10}0100101001 \ldots \\ \mathrm{tl}^5(F_\infty) &:= 0100101\mathbf{01}0100101001010\mathbf{01}0100101\mathbf{01}0100101001 \ldots \end{aligned}$$

Here, we see that each mismatch involves a factor 10 of F_∞ for which the corresponding factor in $\mathrm{tl}^5(F_\infty)$ is the reverse, 01. In fact, we see the same phenomenon for all shifts of F_∞ by an element of the Fibonacci sequence. Furthermore, it turns out that for each successive such shift, the distance between the mismatching factors increases. This is formalised below as the *expanding gaps property*. We will show that the preceding observations about the Fibonacci word generalise to arbitrary Sturmian words.

3.2 The Tribonacci Word

Recall that Sturmian words have factor complexity $p(n) = n + 1$ and thus can be considered as the simplest non-periodic infinite words. A natural candidate for the next simplest such class is the set of Arnoux-Rauzy words. Over a ternary alphabet such words have factor complexity $p(n) = 2n+1$. A prototypical example of an Arnoux-Rauzy word is the Tribonacci word, which we introduce next.

Let $\Sigma = \{0, 1, 2\}$ and consider the morphism $\sigma : \Sigma^* \to \Sigma^*$ given by $\sigma(0) = 01$, $\sigma(1) = 02$, and $\sigma(2) = 0$. The *Tribonacci word* $W_\infty \in \Sigma^\omega$ is the morphic word

$$W_\infty := \sigma^\omega(0) = 0102010010201\ldots .$$

In more detail, the Tribonacci word W_∞ is the limit of the sequence of finite words $(W_n)_{n=0}^\infty$ given by $W_n = \sigma^n(0)$ for all n (again we have that W_n is a prefix of W_{n+1} for all n, so the limit is well defined). Observe that the sequence of words $(W_n)_{n=0}^\infty$ satisfies recurrence

$$W_n = W_{n-1}W_{n-2}W_{n-3} \quad (n \geq 3)\,.$$

Associated with the Tribonacci word we have the sequence $\langle t_n\rangle_{n=0}^\infty$ of Tribonacci numbers, defined by the recurrence $t_n = t_{n-1} + t_{n-2} + t_{n-3}$ and initial conditions $t_0 = 1, t_1 = 2, t_2 = 4$. Clearly the word W_n has length t_n for all $n \in \mathbb{N}$.

In the spirit of our analysis of the Fibonacci word, we match the Tribonacci word against shifts of itself by elements of the Tribonacci sequence $\langle 1, 2, 4, 7, 13, \ldots\rangle$. By way of example, below we compare W_∞ and $\mathrm{tl}^{13}(W_\infty)$:.

$$\begin{aligned} T_\infty &:= 0102010010201\mathbf{02}\mathbf{01}\mathbf{00}\mathbf{102}0102010010201\mathbf{02}\mathbf{01}\mathbf{0}0102010010201\mathbf{02}\ldots \\ \mathrm{tl}^{13}(T_\infty) &:= 0102010010201\mathbf{02}\mathbf{01}\mathbf{0}\mathbf{02}\mathbf{01}0102010010201\mathbf{0}\mathbf{02}\mathbf{01}0102010010201\mathbf{02}\ldots \end{aligned}$$

Similar to the example of the Fibonacci word, the mismatches above appear as a fixed set of factors (either 10,20, or 102) in T_∞ that get reversed in $\mathrm{tl}^{13}(T_\infty)$. Unlike with the Fibonacci word, this time the factors may appear close to each other. Nevertheless, by suitably grouping these factors, we recover a form of the expanding gaps property and we are moreover able to show that the mismatches between T_∞ and its shifts are relatively sparse.

3.3 Definition of Echoing Words

Inspired by the respective examples of the Fibonacci and Tribonacci words, we give in this section the formal definition of echoing word.

Given two non-empty intervals $I, J \subseteq \mathbb{N}$, write $I < J$ if $a < b$ for all $a \in I$ and $b \in J$, and define the distance of I and J to be $d(I, J) := \min\{|a - b| : a \in I, b \in J\}$.

▶ **Definition 3.** *Let $\Sigma \subseteq \overline{\mathbb{Q}}$ be a finite alphabet. An infinite word $\boldsymbol{u} = u_0u_1u_2\ldots \in \Sigma^\omega$ is said to be* echoing *if for all $c, \varepsilon_1 > 0$, there exists $d \geq 2$ and for all $n \in \mathbb{N}$ there exist positive integers r_n, s_n and intervals $\{0\} < I_{1,n} < \cdots < I_{d,n} < \{s_n + 1\}$ of total length ℓ_n, such that:*

1. *the sequence $\langle r_n\rangle_{n=0}^\infty$ is unbounded and $s_n \geq cr_n$ for all n;*
2. *for all n it holds that $\{i \in \{0, \ldots, s_n\} : u_i \neq u_{i+r_n}\} \subseteq \bigcup_{j=1}^d I_{j,n}$ and $\ell_n \leq \varepsilon_1 s_n$;*
3. *as $n \to \infty$ we have $d(\{0\}, I_{1,n}) = \omega(\log(r_n + \ell_n))$ and $d(I_{j,n}, I_{j+1,n}) = \omega(\log \ell_n)$ for $1 \leq j \leq d - 1$;*
4. *for all $\beta \in \overline{\mathbb{Q}}$ such that $|\beta| > 1$ and all $n \in \mathbb{N}$ there exist at least two $j \in \{1, \ldots, d\}$ such that $\sum_{i \in I_{j,n}} (u_i - u_{i+r_n})\beta^{-i} \neq 0$.*

Properties 1–4 concern the factors $\langle u_0, \ldots, u_{s_n}\rangle$ and $\langle u_{r_n}, \ldots, u_{r_n+s_n}\rangle$ of $\boldsymbol{u}$. The inequality in Property 1 allows us to take the prefix length s_n to be an arbitrarily large multiple of the shift r_n. We call this the *Long-Overlap Property.* Informally speaking, Property 2 says that mismatches between the above two factors can be grouped into a fixed number d of intervals whose total length ℓ_n is small in proportion to s_n. We call this the *Short-Intervals Property.* Property 3 gives lower bounds on the length of the gaps between the above-mentioned intervals. We call this the *Expanding-Gaps Property.* Property 4 will be used to show, in case β is algebraic, that the words $\langle u_0, \ldots, u_{s_n}\rangle$ and $\langle u_{r_n}, \ldots, u_{r_n+s_n}\rangle$ denote different numbers base β for infinitely many n. We call this the *Non-Vanishing Property.*

The Fibonacci and Tribonacci words are both echoing. The formal proofs will be given respectively in Section 4 and Section 5. As suggested by the examples above, in the case of the Fibonacci word a suitable choice for the sequence $\langle r_n\rangle_{n=0}^{\infty}$ of shifts will be the Fibonacci sequence, while in the case of the Tribonacci word it will be the Tribonacci sequence. In the case of the Fibonacci word (and other Sturmian words) all of the intervals $I_{j,n}$ will be doubletons, whereas in the case of the Tribonacci word their total length ℓ_n grows linearly with s_n.

We conclude this section with some remarks about related work. The notion of a echoing word is reminiscent of the transcendence conditions of [1, 8, 11] in that it concerns periodicity in an infinite word. The ability to choose the parameter c to be arbitrarily large (the Long-Overlap property) is key to our being able to prove transcendence results over an arbitrary algebraic base β. In compensation, we allow for a small number of mismatches, as detailed in the Short-Interval property. This should be contrasted with the notion of stammering word in [1, 3, Section 4], where there is no allowance for such discrepancies and in which the quantity corresponding to c is determined in advance by the word (cf. the notion of the *Diophantine exponent* of a word in [2]).

4 Sturmian Words are Echoing

In this section we show that Sturmian words are echoing and, more generally, that a pointwise linear combination of a collection of Sturmian words having the same slope is echoing.

We will work with a characterisation of Sturmian words in terms of dynamical systems. Write $I := [0, 1)$ for the unit interval and given $x \in \mathbb{R}$ denote the integer part of x by $\lfloor x\rfloor$ and the fractional part of x by $\{x\} := x - \lfloor x\rfloor \in I$. Let $0 < \theta < 1$ be an irrational number and define the *rotation map* $R = R_\theta : I \to I$ by $R(y) = \{y + \theta\}$. Given $x \in I$, the *θ-coding of x* is the infinite word $\boldsymbol{u} = u_1u_2u_3\ldots$ defined by $u_n := 1$ if $R^n(x) \in [0, \theta)$ and $u_n := 0$ otherwise. As shown by Morse and Hedlund, $\boldsymbol{u}$ is a Sturmian word and, up to changing at most two letters, all Sturmian words over a binary alphabet arise as codings of the above type for some choice of θ and x. In particular, for the purposes of establishing our transcendence results we may work exclusively with codings as defined above. The number θ is equal to the slope of the Sturmian word, as defined in Section 1. The θ-coding of 0 is in particular called the *characteristic (or standard)* Sturmian word of slope θ.

The main result of this section is as follows:

▶ **Theorem 4.** *Let $\theta \in (0, 1)$ be irrational. Given a positive integer k, let $c_0, \ldots, c_k \in \mathbb{C}$ and $x_1, \ldots, x_k \in I$ with $c_1, \ldots, c_k$ non-zero. Suppose that $x_i - x_j \notin \mathbb{Z}\theta + \mathbb{Z}$ for all $i \neq j$. Writing $\langle u_n^{(i)}\rangle_{n=0}^{\infty}$ for the θ-coding of x_i, for $i = 1, \ldots, k$, define $u_n := c_0 + \sum_{i=1}^{k} c_i u_n^{(i)}$ for all $n \in \mathbb{N}$. Then $\boldsymbol{u} = \langle u_n\rangle_{n=0}^{\infty}$ is echoing.*

Proof. We start by recalling some basic notions concerning continued fractions (see [12, Chapter 10] for details). Let

$$\theta = \cfrac{1}{a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \ddots}}}$$

be the simple continued-fraction expansion of θ. Given $n \in \mathbb{N}$, we write $\frac{p_n}{q_n}$ for the n-th convergent of the above continued fraction, which is obtained by truncating it at a_n. Then $\langle q_n \rangle_{n=0}^{\infty}$ is a strictly increasing sequence of positive integers such that $\|q_n\theta\| = |q_n\theta - p_n|$, where $\|\alpha\|$ denotes the distance of $\alpha \in \mathbb{R}$ to the nearest integer. We moreover have that $q_n\theta - p_n$ and $q_{n+1}\theta - p_{n+1}$ have opposite signs for all n. Finally we have the *law of best approximation*: $q \in \mathbb{N}$ occurs as one of the q_n just in case $\|q\theta\| < \|q'\theta\|$ for all q' with $0 < q' < q$.

To establish that $\boldsymbol{u}$ is echoing, given $c > 0$ we define $\langle r_n \rangle_{n=0}^{\infty}$ to be the subsequence of $\langle q_n \rangle_{n=0}^{\infty}$ comprising all terms q_n such that $\|q_n\theta\| = q_n\theta - p_n > 0$. We thereby have that either $r_n = q_{2n}$ for all n or $r_n = q_{2n+1}$ for all n, so $\langle r_n \rangle_{n=0}^{\infty}$ is an infinite sequence that diverges to infinity. Next, define $d := (k+1)c$ and for all $n \in \mathbb{N}$ define s_n to be the greatest number such that the words $u_0 \dots u_{s_n}$ and $u_{r_n} \cdots u_{r_n+s_n}$ have Hamming distance $2d$. Since $\boldsymbol{u}$ is not ultimately periodic, s_n is well-defined.

Short-Intervals Property. Given $n \in \mathbb{N}$, denote the set of positions at which $u_0 \dots u_{s_n}$ and $u_{r_n} \dots u_{s_n+r_n}$ differ by

$$\Delta_n := \left\{ m \in \{0, \dots, s_n\} : u_m \neq u_{m+r_n} \right\}. \tag{1}$$

We claim that for n sufficiently large, $m \in \Delta_n$ if and only if there exists $\ell \in \{1, \dots, k\}$ such that one of the following two conditions holds:

- **(i)** $R^m(x_\ell) \in [1 - \|r_n\theta\|, 1)$,
- **(ii)** $R^m(x_\ell) \in [\theta - \|r_n\theta\|, \theta)$.

We moreover claim that for all m there is at most one ℓ such that Condition (i) or (ii) holds.

Assuming the claim, since $R^m(x_\ell) \in [1 - \|r_n\theta\|, 1)$ if and only if $R^{m+1}(x_\ell) \in [\theta - \|r_n\theta\|, \theta)$, it follows that the elements of Δ_n come in consecutive pairs, i.e., we can write

$$\Delta_n = \bigcup_{j=1}^{d} \{i_{j,n}, i_{j,n} + 1\},$$

where $i_{1,n} < \cdots < i_{d,n}$ are the elements $m \in \Delta_n$ that satisfy Condition (i) above for some ℓ, while $i_{1,n} + 1 < \cdots < i_{d,n} + 1$ are those that satisfy Condition (ii). Defining $I_{j,n} := \{i_{j,n}, i_{j,n} + 1\}$ for $j \in \{1, \dots, d\}$, we have that Item 2 of Defintion 3 is satisfied. Indeed, since the intervals $I_{1,n}, \dots, I_{d,n}$ have total length $\ell_n = 2d$, for any choice of $\varepsilon_1 > 0$ we have $\ell_n \leq \varepsilon_1 s_n$ for n sufficiently large.

It remains to prove the claim. To this end note that for a fixed $\ell \in \{1, \dots, k\}$, for all m we have that $u_m^{(\ell)} \neq u_{m+r_n}^{(\ell)}$ iff exactly one of $R^m(x_\ell)$ and $R^{m+r_n}(x_\ell)$ lies in the interval $[0, \theta)$ iff one of Condition (i) or Condition (ii), above, holds. Moreover, since $x_\ell - x_{\ell'} \neq \theta \pmod 1$ for $\ell \neq \ell'$, we see that for n sufficiently large there is at most one $\ell \in \{1, \dots, k\}$ such that $u_m^{(\ell)} \neq u_{m+r_n}^{(\ell)}$. We deduce that $u_m \neq u_{m+r_n}$ if and only if $u_m^{(\ell)} \neq u_{m+r_n}^{(\ell)}$ for some $\ell \in \{1, \dots, k\}$. This concludes the proof of the claim.

Long-Overlap Property. Our objective is to show that $s_n \geq cr_n$ for all $n \in \mathbb{N}$. We have already established that there are $d = (k+1)c$ distinct $m \in \Delta_n$ that satisfy Condition (i), above, for some $\ell \in \{1, \ldots, k\}$. Thus there exists $\ell_0 \in \{1, \ldots, k\}$ and $\Delta'_n \subseteq \Delta_n$ such that $|\Delta'_n| \geq c$ and all $m \in \Delta'_n$ satisfy Condition (i) for $\ell = \ell_0$. In this case we have $\|(m_1 - m_2)\theta\| < \|r_n\theta\|$ for all $m_1, m_2 \in \Delta'_n$. By the law of best approximation it follows that every two distinct elements of Δ'_n have difference strictly greater than r_n. But this contradicts $|\Delta'_n| = c$, given that $\Delta'_n \subseteq \{0, 1, \ldots, cr_n\}$.

Expanding-Gaps Property. By definition of $i_{1,n}, \ldots, i_{d,n}$, for all $1 \leq j_1 < j_2 \leq d$ there exists $\ell_1, \ell_2 \in \{1, \ldots, k\}$ with $R^{i_{j_1,n}}(x_{\ell_1}), R^{i_{j_2,n}}(x_{\ell_2}) \in [1 - \|r_n\theta\|, 1)$. We deduce that

$$\|(i_{j_2,n} - i_{j_1,n})\theta + x_{\ell_1} - x_{\ell_2}\| \leq \|r_n\theta\| \,. \tag{2}$$

We claim that the left-hand side of (2) is non-zero. Indeed, the claim holds if $\ell_1 = \ell_2$ because θ is irrational, while the claim also holds in case $\ell_1 \neq \ell_2$ since in this case we have $x_{\ell_1} - x_{\ell_2} \notin \mathbb{Z}\theta + \mathbb{Z}$ by assumption. Since moreover the right-hand side of (2) tends to zero as n tends to infinity, we have that $i_{j_2,n} - i_{j_1,n} = \omega(1)$ as $n \to \infty$. Since $\ell_n = 2d$ is constant, independent of n, we have $i_{j_2,n} - i_{j_1,n} = \omega(\ell_n)$. Finally, we have $i_{1,n} > r_n$ by the requirement that $||R^{i_{n,1}}(x_{\ell_{n,1}})|| < ||r_n\theta||$ and the best-approximation property of r_n. Clearly this entails that $i_{1,n} = \omega(\log(r_n + \ell_n))$. This completes the verification of Item 3 of Definition 3(3).

Non-Vanishing Property. Consider $m \in \Delta_n$ satisfying Condition (i) above, i.e., such that $R^m(x_\ell) \in [1 - \|r_n\theta\|, 1)$ for some $\ell \in \{1, \ldots, k\}$. Then we have

$$u_m^{(\ell)} = 0,\ u_{m+1}^{(\ell)} = 1 \quad \text{and} \quad u_{m+r_n}^{(\ell)} = 1,\ u_{m+r_n+1}^{(\ell)} = 0 \,. \tag{3}$$

Moreover for all $\ell' \neq \ell$ and n sufficiently large we have

$$u_m^{(\ell')} = u_{m+r_n}^{(\ell')} \quad \text{and} \quad u_{m+1}^{(\ell')} = u_{m+r_n+1}^{(\ell')} \,. \tag{4}$$

From Equations (3) and (4) we deduce that $u_m \neq u_{m+r_n}$, $u_{m+1} \neq u_{m+1+r_n}$, and $u_m + u_{m+1} = u_{m+r_m} + u_{m+r_n+1}$. But this implies that $\beta u_m + u_{m+1} \neq \beta u_{m+r_n} + u_{m+r_n+1}$ for all $\beta \neq 1$. This establishes Item 4 of Definition 3. ◀

5 The Tribonacci Word is Echoing

5.1 The Matching Morphism

In this section we define a morphism in order to understand how the Tribonacci word aligns with shifts of itself. This is an instance of a construction that is used elsewhere to show that for certain morphisms, the associated shift dynamical system has pure discrete spectrum (see the notion of *balanced pairs* in [16] and [22, Definition 6.8]).

Recall that the Tribonacci word W_∞ is defined over the alphabet $\Sigma = \{0, 1, 2\}$ as a fixed point of the morphism $\sigma(0) = 01, \sigma(1) = 02, \sigma(2) = 0$. We define an alphabet $\Delta = \{a_0, \ldots, a_{10}\}$ whose elements are certain ordered pairs of words in Σ^+ having the same Parikh image. For intuition we represent the elements of Δ as tiles as follows:

$$a_0 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad a_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad a_2 = \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix} \quad a_3 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \quad a_4 = \begin{bmatrix} 1 & 0 & 2 \\ 2 & 0 & 1 \end{bmatrix} \quad a_5 = \begin{bmatrix} 2 & 0 & 1 \\ 1 & 0 & 2 \end{bmatrix}$$

$$a_6 = \begin{bmatrix} 0 & 1 & 0 & 2 \\ 2 & 0 & 1 & 0 \end{bmatrix} \quad a_7 = \begin{bmatrix} 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 \end{bmatrix} \quad a_8 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad a_9 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad a_{10} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

We partition Δ into a set $\Delta_0 := \{a_0, \ldots, a_7\}$ of *mismatches* and a set $\Delta_1 := \{a_8, a_9, a_{10}\}$ of *matches*.

Define morphisms $\mathrm{top}, \mathrm{bot} : \Delta^* \to \Sigma^*$ such that top extracts the word on the top of each tile and bottom extracts the word on the bottom, e.g.,

$$\mathrm{top}\left(\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\right) = 01 \quad \text{and} \quad \mathrm{bot}\left(\begin{bmatrix}1 & 0 & 2\\ 2 & 0 & 1\end{bmatrix}\right) = 201\,.$$

Below we define the *matching morphism* $\mu : \Delta^* \to \Delta^*$, which is characterised by the following properties:

$$\mathrm{top} \circ \mu = \sigma \circ \mathrm{top} \qquad \text{and} \qquad \mathrm{bot} \circ \mu = \sigma \circ \mathrm{bot}\,. \tag{5}$$

Specifically we have

$$\mu\left(\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\right) := \begin{bmatrix}0\\ 0\end{bmatrix}\begin{bmatrix}1 & 0 & 2\\ 2 & 0 & 1\end{bmatrix} \qquad \mu\left(\begin{bmatrix}1 & 0 & 2\\ 2 & 0 & 1\end{bmatrix}\right) := \begin{bmatrix}0\\ 0\end{bmatrix}\begin{bmatrix}2 & 0 & 1 & 0\\ 0 & 1 & 0 & 2\end{bmatrix}$$

$$\mu\left(\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\right) := \begin{bmatrix}0\\ 0\end{bmatrix}\begin{bmatrix}2 & 0 & 1\\ 1 & 0 & 2\end{bmatrix} \qquad \mu\left(\begin{bmatrix}2 & 0 & 1\\ 1 & 0 & 2\end{bmatrix}\right) := \begin{bmatrix}0\\ 0\end{bmatrix}\begin{bmatrix}0 & 1 & 0 & 2\\ 2 & 0 & 1 & 0\end{bmatrix}$$

$$\mu\left(\begin{bmatrix}0 & 2\\ 2 & 0\end{bmatrix}\right) := \begin{bmatrix}0\\ 0\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \qquad \mu\left(\begin{bmatrix}2 & 0 & 1 & 0\\ 0 & 1 & 0 & 2\end{bmatrix}\right) := \begin{bmatrix}0\\ 0\end{bmatrix}\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}0 & 2\\ 2 & 0\end{bmatrix}\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$$

$$\mu\left(\begin{bmatrix}2 & 0\\ 0 & 2\end{bmatrix}\right) := \begin{bmatrix}0\\ 0\end{bmatrix}\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} \qquad \mu\left(\begin{bmatrix}0 & 1 & 0 & 2\\ 2 & 0 & 1 & 0\end{bmatrix}\right) := \begin{bmatrix}0\\ 0\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}2 & 0\\ 0 & 2\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$$

and

$$\mu\left(\begin{bmatrix}0\\ 0\end{bmatrix}\right) := \begin{bmatrix}0\\ 0\end{bmatrix}\begin{bmatrix}1\\ 1\end{bmatrix} \qquad \mu\left(\begin{bmatrix}1\\ 1\end{bmatrix}\right) := \begin{bmatrix}0\\ 0\end{bmatrix}\begin{bmatrix}2\\ 2\end{bmatrix} \qquad \mu\left(\begin{bmatrix}2\\ 2\end{bmatrix}\right) := \begin{bmatrix}0\\ 0\end{bmatrix}.$$

For later use we remark that the morphism $\iota : \Sigma^* \to \Delta^*$, defined by $\iota(0) = a_0, \iota(1) = a_2, \iota(2) = a_8$ satisfies $\mathrm{top} \circ \iota = \sigma$, while $\mathrm{bot} \circ \iota$ and $\mathrm{tl} \circ \sigma$ agree on Σ^ω. It follows that

$$\mathrm{top}(\iota(W_\infty)) = W_\infty \quad \text{and} \quad \mathrm{bot}(\iota(W_\infty)) = \mathrm{tl}(W_\infty)\,. \tag{6}$$

Associated with the morphism μ we have its *incidence matrix* $M(\mu) \in \mathbb{N}^{11\times 11}$, where $M(\mu)_{i,j} := |\mu(a_i)|_j$ is the number of occurrences of a_j in $\mu(a_i)$ for all $i, j \in \{0, \ldots, 10\}$. It is straightforward that $M(\mu)^n_{i,j} = |\mu^n(a_i)|_j$ for all n and all $i, j \in \{0, \ldots, 10\}$. Matrix $M(\mu)$ admits a block decomposition

$$M(\mu) = \begin{pmatrix} B_1 & B_2 \\ 0 & B_3 \end{pmatrix},$$

where B_1 is the restriction of $M(\mu)$ to the set of mismatch symbols Δ_0 and B_3 is the restriction to the set of match symbols Δ_1. By direct calculation one see that both B_1 and B_3 are primitive and have respective spectral radii $\rho' \approx 1.395$ and $\rho \approx 1.839$. [1]

We hence have that for all $n \in \mathbb{N}$ and $i \in \{0, \ldots, 7\}$,

$$\frac{\sum_{j=0}^{7} |\mu^n(a_i)|_j}{|\mu^n(a_i)|} = \frac{\sum_{j=0}^{7} M(\mu)^n_{i,j}}{\sum_{j=0}^{10} M(\mu)^n_{i,j}} \leq \frac{\sum_{j=0}^{7} (B_1^n)_{i,j}}{\sum_{j=0}^{2}\sum_{k=0}^{n-1} (B_1^k B_2 B_3^{n-k})_{i,j}} \ll \left(\frac{\rho'}{\rho}\right)^n. \tag{7}$$

We deduce that the frequency of mismatch symbols in $\mu^n(a_0)$ converges to 0 as n tends to infinity. The above reasoning shows, *inter alia* that for all $a \in \Delta$ the sequence $|\mu^n(a)| = \Theta(\rho^n)$ and hence there exists a constant κ such that $|\mu^n(a)| \leq \kappa |\mu^n(b)|$ for all $a, b \in \Delta$ and all n sufficiently large.

[1] The inequality $\rho' < \rho$ implies that the Tribonacci morphism has pure discrete spectrum (see [22, Section 6.3] and the references therein).

5.2 Matching Polynomials

Define eval : $\Sigma^* \to \mathbb{Z}[x]$ by $\mathrm{eval}(u_0 \dots u_n) := \sum_{i=0}^{n} u_i x^i$. For all $i \in \{0, \dots, 7\}$ and $n \in \mathbb{N}$ define the *matching polynomial* $P_{i,n}(x) \in \mathbb{Z}[x]$ by

$$P_{i,n} := \mathrm{eval}(\mathrm{top}(\mu^n(a_i))) - \mathrm{eval}(\mathrm{bot}(\mu^n(a_i)))\,. \tag{8}$$

By inspection, the only common root of $P_{i,0}(x)$ for $i \in \{0, \dots, 7\}$ is $x = 1$.

Observe that for all $n \in \mathbb{N}$ and $i \in \{0, 1, 2, 3\}$ we have

$$\mathrm{top}(\mu^n(a_{2i})) = \mathrm{bot}(\mu^n(a_{2i+1})) \quad \text{and} \quad \mathrm{bot}(\mu^n(a_{2i})) = \mathrm{top}(\mu^n(a_{2i+1}))\,,$$

and hence $P_{2i,n}(x) = -P_{2i+1,n}(x)$. As a consequence we can focus our attention on the even-index polynomials $P_{0,n}$, $P_{2,n}$, $P_{4,n}$, and $P_{6,n}$. Indeed, writing $\boldsymbol{P}_n := (P_{0,n}, P_{2,n}, P_{4,n}, P_{6,n}) \in \mathbb{Z}[x]^4$ and

$$M_n := (-1) \cdot \begin{pmatrix} 0 & 0 & -x^{t_n} & 0 \\ x^{t_n} & 0 & 0 & 0 \\ 0 & 0 & 0 & x^{t_n} \\ x^{t_n} + x^{t_{n+1}} & x^{t_n + t_{n+1}} & 0 & 0 \end{pmatrix},$$

then we have $\boldsymbol{P}_{n+1} = M_n \boldsymbol{P}_n$ for all $n \in \mathbb{N}$. But now, since $\boldsymbol{P}_0(\beta) \neq \boldsymbol{0}$ for all $\beta \neq 1$ and $\det(M_n) = -x^{t_{n+1}+4t_n}$, it follows that $\boldsymbol{P}_n(\beta) \neq \boldsymbol{0}$ for all $\beta \in \mathbb{C} \setminus \{0, 1\}$.

5.3 Putting Things Together

▶ **Theorem 5.** *The Tribonacci word $\boldsymbol{u} := W_\infty$ is echoing.*

Proof. We refer to Definition 3. Let $c, \varepsilon_1 > 0$ be given and write $\boldsymbol{w} := \iota(u_0 u_1 \dots u_{c-1}) \in \Delta^*$. It follows from (6) that $\mathrm{top}(\boldsymbol{w})$ is a prefix of $\boldsymbol{u}$ and $\mathrm{bot}(\boldsymbol{w})$ is a prefix of $\mathrm{tl}(\boldsymbol{u})$. Given $n_0 \in \mathbb{N}$, write

$$\mu^{n_0}(\boldsymbol{w}) = \boldsymbol{w}_0 a_{i_1} \boldsymbol{w}_1 \cdots \boldsymbol{w}_{d-1} a_{i_d} \boldsymbol{w}_d\,, \tag{9}$$

where $\boldsymbol{w}_0, \dots, \boldsymbol{w}_d \in \Delta_1^*$ are sequences of match symbols and $a_{i_1}, \dots, a_{i_d} \in \Delta_0$ are mismatch symbols. For $n_0 \in \mathbb{N}$ sufficiently large it holds that $\mu^{n_0}(\boldsymbol{w})$ contains at least two occurrences of every mismatch symbol and the proportion of mismatch symbols in $\mu^{n_0}(\boldsymbol{w})$ is at most ε_1/κ, for κ as in Section 5.1.

For all $n \in \mathbb{N}$, referring to Equation (9), we have

$$\mu^{n+n_0}(\boldsymbol{w}) = \mu^n(\boldsymbol{w}_0) \underbrace{\mu^n(a_{i_1})}_{I_{1,n}} \mu^n(\boldsymbol{w}_1) \cdots \mu^n(\boldsymbol{w}_{d-1}) \underbrace{\mu^n(a_{i_d})}_{I_{d,n}} \mu^n(\boldsymbol{w}_d)\,. \tag{10}$$

The data to show the echoing property are as follows. For all $n \in \mathbb{N}$ define $r_n := |\sigma^n(0)| = t_{n+n_0}$, $s_n := |\mathrm{top}(\mu^{n+n_0}(\boldsymbol{w}))|$, and we take d as in (9). For all $j \in \{1, \dots, d\}$ we define $I_{j,n} \subseteq \mathbb{N}$ to be the interval of positions in $\mu^{n+n_0}(a_0)$ corresponding to the shortest suffix of $\mu^n(a_{i_j})$ that contains all mismatch symbols that occur therein (see (10)).

From Equation (5) we have $\mathrm{top}(\mu^{n+n_0}(\boldsymbol{w})) = \langle u_0, \dots, u_{s_n} \rangle$ and $\mathrm{bot}(\mu^{n+n_0}(\boldsymbol{w})) = \langle u_{r_n}, \dots, u_{s_n+r_n} \rangle$. Since $\mathrm{top}(\boldsymbol{w})$ contains at least c occurrences of the letter 0, we get that $s_n \geq c r_n$, establishing the Long-Overlap Property.

By construction, the intervals $I_{1,n}, \dots, I_{d,n}$ contain all indices where the above two strings differ and they have total length at most $\varepsilon_1 s_n$, establishing the Short-Intervals Property.

By definition of the matching polynomials, it holds that $P_{i_j,n}(\beta^{-1})$ is the product of $\sum_{i\in I_{j,n}}(u_i - u_{i+r_n})\beta^{-1}$ and a power of β for all $j \in \{1,\dots,d\}$. We have shown in Section 5.2 that the vector $(P_{0,n}(\beta^{-1}),\dots,P_{7,n}(\beta^{-1}))$ is non-zero for all n. Since all mismatch symbols occur at least twice among $a_{i_1},\dots,a_{i_d}$, we have that $P_{i_j,n}(\beta^{-1})$ is non-zero for at least two different choices of $j \in \{1,\dots,d\}$. It follows that $\sum_{i\in I_{j,n}}(u_i - u_{i+r_n})\beta^{-1}$ is non-zero for at least two different values of j, establishing the Non-Vanishing Property.

Finally, note that there is at least one letter between each pair of intervals $I_{j,n}$ in (10). Hence the inequality $|\mu(a^n)| \le \kappa|\mu^n(b)|$ for $a,b \in \Delta$ implies that $d(\{0\},I_{1,n}) \gg s_n$ and $d(I_{j,n},I_{j+1,n}) \gg s_n$ for all $j \in \{0,\dots,d-1\}$, which implies the Expanding-Gaps Property. ◀

6 Transcendence Results

6.1 Transcendence for Echoing Words

▶ **Theorem 6.** *Let Σ be a finite set of algebraic numbers and let $\boldsymbol{u} \in \Sigma^\omega$ be an echoing word. Then for any algebraic number β such that $|\beta| > 1$, the sum $\alpha := \sum_{n=0}^{\infty} \frac{u_n}{\beta^n}$ is transcendental.*

Proof. Suppose for a contradiction that α is algebraic. By scaling, we can assume without loss of generality that Σ consists solely of algebraic integers. Let K be the field generated over $\mathbb{Q}$ by $\{\beta\} \cup \Sigma$ and write $S \subseteq M(K)$ for the set comprising all infinite places of K and those finite places of K arising from prime-ideal divisors of elements of $\{\beta\} \cup \Sigma$. Let $v_0 \in S$ be the place corresponding to the inclusion of K in $\mathbb{C}$. Recall that $|a|_{v_0} = |a|^{1/[K:\mathbb{Q}]}$, where $|a|$ denotes the usual absolute value on $\mathbb{C}$.

Applying the definition of echoing sequence (as given in Definition 3) for values of c and ε_1 to be specified later, we obtain $d \ge 2$ such that for all $n \in \mathbb{N}$ there are $r_n, s_n \in \mathbb{N}$ and intervals $\{0\} < I_{1,n} < \cdots < I_{d,n} < \{s_n + 1\}$, of total length ℓ_n, satisfying Items 1–4 of Definition 3.

For $n \in \mathbb{N}$, define $\boldsymbol{a}_n = (a_{1,n},\dots,a_{d+3,n}) \in (\mathcal{O}_S)^{d+3}$ by

$$a_{1,n} := \beta^{r_n},\ a_{2,n} := \sum_{i=0}^{r_n} u_i \beta^{r_n - i},\ a_{3,n} := 1,\ a_{j+3,n} := \sum_{i\in I_{n,j}} (u_{i+r_n} - u_i)\beta^{-i}\ (j = 1,\dots,d)$$

The Non-Vanishing Property (Definition 3(4)) implies that for all $n \in \mathbb{N}$ we have $a_{j+3,n} \neq 0$ for at least two elements $j \in \{1,\dots,d\}$. By passing to a subsequence we henceforth assume without loss of generality that there exists $J \subseteq \{1,\dots,d\}$, of cardinality at least two, such that for all j and n, $a_{j+3,n} \neq 0$ if and only if $j \in J$.

▷ Claim 7. If $F(x_1,\dots,x_{d+3}) = \sum_{i\in\{1,2,3\}\cup J} \alpha_i x_i$ is a linear form with coefficients in K such that $F(\boldsymbol{a}_n) = 0$ for infinitely many n, then $\alpha_j = 0$ for all $j \in J$.

The proof of the claim is as follows. For all $n \in \mathbb{N}$ we have $F(\boldsymbol{a}_n) = P_n(\beta)$ for the polynomial $P_n(x) := P_{0,n}(x) + \sum_{j\in J} P_{j,n}(x)$, where

$$P_{0,n}(x) := \alpha_1 x^{r_n} + \alpha_2 \sum_{i=0}^{r_n} u_i x^{r_n - i} + \alpha_3 \quad \text{and} \quad P_{j,n}(x) := \alpha_j \sum_{i\in I_{j,n}} (u_i - u_{i+r_n})x^{-i}\,.$$

Polynomial P_n has at most $r_n + \ell_n$ monomials. From Proposition 2 and the property $d(\{0\},I_{1,n}) = \omega(\log(r_n + \ell_n))$ (see Definition 3(3)), we deduce that $\sum_{j\in J} P_{j,n}(\beta) = 0$ for infinitely many n. Now $\sum_{j\in J} P_{j,n}(x)$ comprises at most ℓ_n monomials. Thus Proposition 2

and the assumption that $d(I_{j,n}, I_{j+1,n}) = \omega(\log \ell_n)$ for all $j \in \{1, \dots, d-1\}$, entail that for infinitely many n we have $P_{j,n}(\beta) = 0$ for all $j \in J$. But $P_{j,n}(\beta) = \alpha_j a_{j,n}$ and so, since $a_{j,n} \neq 0$ for all $j \in J$, we have $\alpha_j = 0$ for all $j \in J$. This concludes the proof of the claim.

Consider the linear form $L_0(x_1, \dots, x_{d+3}) := \alpha x_1 - x_2 - \alpha x_3 - \sum_{j\in J} x_{j+3}$. Then there exists $c_1 > 1$ such that for all n,

$$\begin{array}{rcl} 0 < |L_0(\boldsymbol{a}_n)| & = & \left|\beta^{r_n}\alpha - \sum_{i=0}^{r_n} u_i \beta^{r_n - i} + \alpha + \sum_{j\in J}\sum_{i\in I_{j,n}} (u_i - u_{i+r_n})\beta^i\right| \\ & = & \left|\sum_{i=s_n+1}^{\infty} (u_i - u_{i+r_n})\beta^{-i}\right| < c_1|\beta|^{-s_n}, \end{array} \tag{11}$$

where the left-hand inequality follows from an application of Claim 7 to L_0. Consider a linear form $L(x_1, \dots, x_{d+3})$ with the following properties: (i) L has coefficients in K; (ii) L has support $\{x_i : i \in I \cup J\}$ for some $I \subseteq \{1, 2, 3\}$; (iii) $0 < |L(\boldsymbol{a}_n)| < c_1|\beta|^{-s_n}$ for all $n \in \mathbb{N}$; (iv) the set I is minimal with respect to set inclusion among linear forms satisfying (i)–(iii). We have just exhibited a form, namely L_0, that satisfies Conditions (i)–(iii), so L is well-defined.

Let $c_2 \geq 2$ be an upper bound of the set of numbers $\{|\gamma|_v : \gamma \in \{\beta\} \cup A \cup A - A,\ v \in S\}$. Then for $v \in S$, by the assumption that $s_n \geq cr_n$ we have

$$|a_{2,n}|_v \leq \sum_{i=0}^{r_n} c_2^{i+1} \leq c_2^{r_n+2} \leq c_2^{(c^{-1}s_n+2)}. \tag{12}$$

We moreover have

$$\prod_{j\in J}\prod_{v\in S} |a_{j+3,n}|_v \leq \prod_{j\in J}\prod_{v\in S}\sum_{i=0}^{|I_{j,n}|} c_2^{i+1} \leq \prod_{j\in J} c_2^{|S|(2+|I_{j,n}|)} \leq c_2^{|S|(2d+\varepsilon_1 s_n)}, \tag{13}$$

where we use the assumption that the intervals $I_{1,n}, \dots, I_{d,n}$ have total length $\ell_n \leq \varepsilon_1 s_n$. We also have $\prod_{v\in S} |a_{1,n}|_v = \prod_{v\in S} |\beta^{r_n}|_v = 1$ by the product formula and, obviously, $\prod_{v\in S} |a_{3,n}|_v = 1$.

Pick $i_0 \in I \cup J$. Then, combining (12) and (13) and the bound $|L(\boldsymbol{a}_n)| < c_1|\beta|^{-s_n}$, we have

$$|L(\boldsymbol{a}_n)|_{v_0} \cdot \prod_{\substack{(i,v)\in(I\cup J)\times S \\ (i,v)\neq(i_0,v_0)}} |a_{i,n}|_v \leq c_2^{\left(s_n(\varepsilon_1+c^{-1})+2d+2\right)|S|} \cdot (c_1|\beta|^{-s_n})^{1/[K:\mathbb{Q}]}. \tag{14}$$

For c sufficiently large, ε_1 sufficiently small, and all but finitely many n, the right-hand side of (14) is less than $|\beta|^{-s_n/2[K:\mathbb{Q}]}$. On the other hand, there exists a constant $c_3 > 0$ such that the height of $\boldsymbol{a}_n$ satisfies the bound $H(\boldsymbol{a}_n) \leq |\beta|^{c_3 s_n}$ for all n. Thus there exists $\varepsilon > 0$ such that the right-hand side of (14) is at most $H(\boldsymbol{a}_n)^{-\varepsilon}$ for infinitely many n.

Given (14), we can apply Theorem 1 to obtain a non-zero linear form $L'(x_1, \dots, x_{3+d})$ that has coefficients in K and support in $\{x_i : i \in I \cup J\}$, such that for infinitely many $n \in \mathbb{N}$ we have both $0 < |L(\boldsymbol{a}_n)| < c_1|\beta|^{-s_n}$ and $L'(\boldsymbol{a}_n) = 0$. By Claim 7, the support of L' is in fact contained in I. Hence, by subtracting a suitable multiple of L' from L we obtain a linear form $L''(x_1, \dots, x_{3+d})$ with strictly fewer coefficients than L such that $0 < |L''(\boldsymbol{a}_n)| < c_1|\beta|^{-s_n}$ for infinitely many $n \in \mathbb{N}$. But this contradicts the minimality of the support of L. ◀

6.2 Transcendence for Sturmian Words and the Tribonacci Word

Combining the transcendence result for echoing words (Theorem 6) with the fact that Sturmian words and the Tribonacci word are echoing (Theorem 4 and Theorem 5), we obtain:

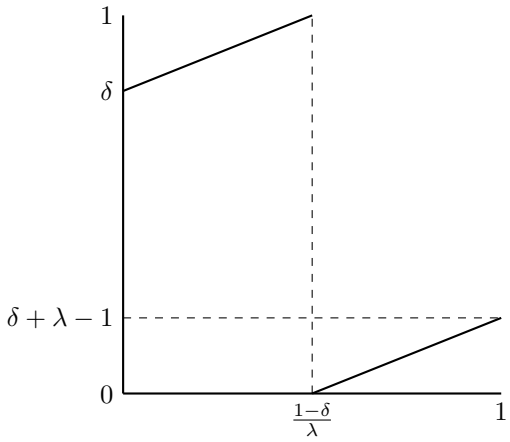


Figure 1 A plot of $f_{\lambda,\delta} : I \to I$.

▶ **Theorem 8.** *Let β be an algebraic number with $|\beta| > 1$.*

1. *Let $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$ be Sturmian words with the same slope such that $\boldsymbol{u}_i$ is not a suffix of $\boldsymbol{u}_j$ for all $i \neq j$. Then $\{1, S_\beta(\boldsymbol{u}_1), \ldots, S_\beta(\boldsymbol{u}_k)\}$ is linearly independent over $\overline{\mathbb{Q}}$.*
2. *Let $\boldsymbol{u}$ be the Tribonacci word. Then $S_\beta(\boldsymbol{u})$ is transcendental.*

7 Application to Limit Sets of Contracted Rotations

Let $0 < \lambda, \delta < 1$ be real numbers such that $\lambda + \delta > 1$. We call the map $f = f_{\lambda,\delta} : I \to I$ given by $f(x) := \{\lambda x + \delta\}$ a *contracted rotation* with slope λ and *offset* δ. Associated with f we have the map $F = F_{\lambda,\delta} : \mathbb{R} \to \mathbb{R}$, given by $F(x) = \lambda\{x\} + \delta + \lfloor x \rfloor$. We call F a *lifting* of f: it is characterised by the properties that $F(x+1) = F(x) + 1$ and $\{F(x)\} = f(\{x\})$ for all $x \in \mathbb{R}$. The *rotation number* $\theta = \theta_{\lambda,\delta}$ of f is defined by

$$\theta := \lim_{n\to\infty} \frac{F^n(x_0)}{n},$$

where the limit exists and is independent of the initial point $x_0 \in \mathbb{R}$.

If the rotation number θ is irrational then the restriction of f to the *limit set* $\bigcap_{n\geq 0} f^n(I)$ is topologically conjugated to the rotation map $R = R_\theta : I \to I$ with $R(y) = \{y + \theta\}$. The closure of the limit set is a Cantor set $C = C_{\lambda,\delta}$, that is, C is compact, nowhere dense, and has no isolated points. On the other hand, if θ is rational then the limit set C is the unique periodic orbit of f. For each choice of slope $0 < \lambda < 1$ and irrational rotation number $0 < \theta < 1$, there exists a unique offset δ such that $\delta + \lambda > 1$ and the map f has rotation number θ. It is known that such δ must be transcendental if λ is algebraic [14].

The main result of this section is as follows:

▶ **Theorem 9.** *Let $0 < \lambda, \theta < 1$ be such that λ is algebraic and θ is irrational. Let δ be the unique offset such that the contracted rotation $f_{\lambda,\delta}$ has rotation number θ. Then every element of the Cantor set $C_{\lambda,\delta}$ other than 0 and 1 is transcendental.*

A special case of Theorem 9, in which λ is assumed to be the reciprocal of an integer, was proven in [8, Theorem 1.2]. In their discussion of the latter result the authors conjecture the truth of Theorem 9, i.e., the more general case in which λ may be algebraic. As noted in [8], while $C_{\lambda,\delta}$ is homeomorphic to the Cantor ternary set, it is a longstanding open problem, formulated by Mahler [19], whether the Cantor ternary set contains irrational algebraic elements.

Proof of Theorem 9. For a real number $0 < x < 1$ define

$$\begin{aligned} \xi_x &:= \sum_{n\geq 1} \left(\lceil x+(n+1)\theta\rceil - \lceil x+n\theta\rceil\right)\lambda^n \\ \xi'_x &:= \sum_{n\geq 1} \left(\lfloor x+(n+1)\theta\rfloor - \lfloor x+n\theta\rfloor\right)\lambda^n . \end{aligned}$$

Note that for all x the binary sequence $\langle \lceil x+(n+1)\theta\rceil - \lceil x+n\theta\rceil : n\in\mathbb{N}\rangle$ is the coding of $-x-\theta$ by $1-\theta$ (as defined in Section 4) and hence is Sturmian of slope $1-\theta$. Similarly, the binary sequence $\langle \lfloor x+(n+1)\theta\rfloor - \lfloor x+n\theta\rfloor : n\in\mathbb{N}\rangle$ is the coding of $x+\theta$ by θ and hence is Sturmian of slope θ. Thus for all x, both ξ_x and ξ'_x are Sturmian numbers.

It is shown in [8, Lemma 4.2][2] that for every element of $y \in C_{\lambda,\delta}\setminus\{0,1\}$, either there exists $z\in\mathbb{Z}$ and $0<x<1$ with $x\notin\mathbb{Z}\theta+\mathbb{Z}$ such that

$$y = z + \xi_0 - \xi_{-x}$$

or else there exists a strictly positive integer m and $\gamma\in\mathbb{Q}(\beta)$ such that

$$y = \gamma + (1-\beta^{-m})\,\xi'_0 .$$

In either case, transcendence of y follows from Theorem 8. ◀

References

1 B. Adamczewski and Y. Bugeaud. On the complexity of algebraic numbers i. expansions in integer bases. *Annals of Mathematics*, 165:547–565, 2005.

2 B. Adamczewski and Y. Bugeaud. Dynamics for β-shifts and diophantine approximation. *Ergodic Theory and Dynamical Systems*, 27:1695–1711, 2007.

3 B. Adamczewski, Y. Bugeaud, and F. Luca. Sur la complexité des nombres algébriques. *Comptes Rendus Mathematique*, 339:11–14, 2004.

4 B. Adamczewski, J. Cassaigne, and M. Le Gonidec. On the computational complexity of algebraic numbers: the hartmanis–stearns problem revisited. *Transactions of the American Mathematical Society*, 373(5):3085–3115, 2020.

5 B. Adamczewski and C. Faverjon. Mahler's method in several variables and finite automata. *arXiv preprint*, 2020. `arXiv:2012.08283`.

6 V. Berthe, S. Ferenczi, and L. Q. Zamboni. Interactions between dynamics, arithmetics and combinatorics: The good, the bad, and the ugly. *Algebraic and Topological Dynamics*, 385, 2005.

7 J. Borwein and P. B. Borwein. On the complexity of familiar functions and numbers. *SIAM Review*, 30(4):589–601, 1988.

8 Y. Bugeaud, D. H. Kim, M. Laurent, and A. Nogueira. On the diophantine nature of the elements of cantor sets arising in the dynamics of contracted rotations. *Annali Scuola Normale Superiore di Pisa - Classe Di Scienze*, XXII:1681–1704, 2021.

9 A. Cobham. Uniform tag seqences. *Math. Syst. Theory*, 6(3):164–192, 1972.

10 L. V. Danilov. Some classes of transcendental numbers. *Mathematical notes of the Academy of Sciences of the USSR*, 12(2):524–527, 1972.

11 S. Ferenczi and C. Mauduit. Transcendence of numbers with a low complexity expansion. *Journal of Number Theory*, 67(2):146–161, 1997.

12 G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, fifth edition, 1978.

[2] The proof of the lemma is stated for β an integer but carries over without change for β algebraic.

13 P. Kebis. *Transcendence of numbers related to Episturmian words*. PhD thesis, University of Oxford, 2023.

14 M. Laurent and A. Nogueira. Rotation number of contracted rotations. *Journal of Modern Dynamics*, 12:175–191, 2018.

15 Hendrik W Lenstra Jr. Finding small degree factors of lacunary polynomials. *Number theory in progress*, 1:267–276, 1999.

16 A. N. Livshits. On the spectra of adic transformations of markov compacta. *Russian Mathematical Surveys*, 42(3):222, 1987.

17 J. H. Loxton and A. J. Van der Poorten. Arithmetic properties of certain functions in several variables iii. *Bulletin of the Australian Mathematical Society*, 16(1):15–47, 1977.

18 F. Luca, J. Ouaknine, and J. Worrell. On the transcendence of a series related to sturmian words, 2022. To appear in Annali della Scuola Normale Superiore di Pisa. `arXiv:2204.08268`.

19 K. Mahler. Some suggestions for further research. *Bulletin of the Australian Mathematical Society*, 29:101–108, 1984.

20 M. Morse and G. A. Hedlund. Symbolic dynamics: Sturmian trajectories. *American Journal of Mathematics*, 60:815–866, 1938.

21 M. Morse and G. A. Hedlund. Symbolic dynamics ii: Sturmian trajectories. *American Journal of Mathematics*, 62:1–42, 1940.

22 M. Queffélec. *Substitution dynamical systems-spectral analysis*, volume 1294. Springer, 2010.

23 G. Rauzy. Nombres algébriques et substitutions. *Bulletin de la Société Mathématique de France*, 110:147–178, 1982.

24 R. Risley and L. Zamboni. A generalization of sturmian sequences: Combinatorial structure and transcendence. *Acta Arithmetica*, 95, January 2000.

The Threshold Problem for Hypergeometric Sequences with Quadratic Parameters

George Kenison
School of Computer Science and Mathematics, Liverpool John Moores University, UK

Abstract

Hypergeometric sequences are rational-valued sequences that satisfy first-order linear recurrence relations with polynomial coefficients; that is, $\langle u_n \rangle_{n=0}^{\infty}$ is hypergeometric if it satisfies a first-order linear recurrence of the form $p(n)u_{n+1} = q(n)u_n$ with polynomial coefficients $p, q \in \mathbb{Z}[x]$ and $u_0 \in \mathbb{Q}$.

In this paper, we consider the Threshold Problem for hypergeometric sequences: given a hypergeometric sequence $\langle u_n \rangle_{n=0}^{\infty}$ and a threshold $t \in \mathbb{Q}$, determine whether $u_n \geq t$ for each $n \in \mathbb{N}_0$. We establish decidability for the Threshold Problem under the assumption that the coefficients p and q are monic polynomials whose roots lie in an imaginary quadratic extension of $\mathbb{Q}$. We also establish conditional decidability results; for example, under the assumption that the coefficients p and q are monic polynomials whose roots lie in any number of quadratic extensions of $\mathbb{Q}$, the Threshold Problem is decidable subject to the truth of Schanuel's conjecture. Finally, we show how our approach both recovers and extends some of the recent decidability results on the Membership Problem for hypergeometric sequences with quadratic parameters.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Computing methodologies → Symbolic and algebraic algorithms; Computing methodologies → Algebraic algorithms

Keywords and phrases Threshold Problem, Membership Problem, Hypergeometric Sequences

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.145

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding The initial stages of this research were conducted at TU Wien where the author was financially supported by the ERC consolidator grant ARTIST 101002685 and the WWTF grant ProbInG ICT19-018.

Acknowledgements I am grateful to both Mahsa Shirmohammadi and James Worrell for many helpful discussions. I also thank the anonymous reviewers for their constructive feedback.

1 Introduction

Background

The *Threshold Problem* is a fundamental open decision problem in automated verification that asks to determine whether every term in a recursively defined sequence is bounded from below by a given value (commonly, the *threshold*). The Threshold Problem appears under many guises across the computational and mathematical sciences with applications in fields as diverse as software verification, probabilistic model checking, combinatorics, and formal languages (we refer the interested reader to the discussion in [26] and the references therein).

The inputs for the Threshold Problem are a recursively defined sequence $\langle u_n \rangle_{n=0}^{\infty} \subseteq \mathbb{Q}$ and a threshold $t \in \mathbb{Q}$. (Hereafter, we shall use tuple notation $(\langle u_n \rangle_{n=0}^{\infty}, t)$ as shorthand for a given problem instance.) Threshold then asks to determine whether $u_n \geq t$ for each $n \in \mathbb{N}_0$. Arguably, the variant of the Threshold Problem that has received the most attention in automated verification is the Positivity Problem for *C-finite sequences* (those sequences that obey a linear recurrence relation with constant coefficients). Therein, Positivity sets as a threshold $t = 0$ and so asks whether every term in a C-finite sequence is non-negative.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 145; pp. 145:1–145:20


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Herein we consider the hypergeometric subclass of *P-finite sequences* (those sequences that satisfy a linear recurrence relation with polynomial coefficients) [11]. Recall that a *hypergeometric sequence* is a rational-valued first-order linear recurrence sequence with polynomial coefficients; that is to say, a sequence $\langle u_n \rangle_{n=0}^{\infty} \subseteq \mathbb{Q}$ that satisfies a relation of the form

$$p(n)u_{n+1} = q(n)u_n \tag{1}$$

where $p, q \in \mathbb{Z}[x]$ and $p(x)$ has no non-negative integer zeros. By the latter assumption on $p(x)$, the recurrence relation (1) uniquely defines an infinite sequence of rational numbers once the initial value $u_0 \in \mathbb{Q}$ is specified. For a hypergeometric sequence $\langle u_n \rangle_{n=0}^{\infty}$ satisfying (1), we call the roots of the polynomial pq the sequence's *parameters*. Hypergeometric sequences and their associated generating functions, the hypergeometric series, are commonplace in fields such as numerical analysis and analytic combinatorics [8, 11].

In this paper, we consider the Threshold Problem for hypergeometric sequences. Naïvely, we might construe that decidability of the Threshold Problem in this setting is easily settled. Consider an instance of the Threshold Problem $(\langle u_n \rangle_{n=0}^{\infty}, t)$. Without loss of generality, we can assume that $\langle u_n \rangle_{n=0}^{\infty}$ either diverges to infinity or converges to a non-zero limit (further explanation behind this assumption is given in the Preliminaries). Suppose that $\langle u_n \rangle_{n=0}^{\infty}$ converges to a limit not equal to t. From the form of the recurrence relation in (1), we can compute a bound B such that if $n > B$ then $u_n > t$ or $u_n < t$. Similar deductions handle the case that $\langle u_n \rangle_{n=0}^{\infty}$ diverges to infinity. In the case that the limit of $\langle u_n \rangle_{n=0}^{\infty}$ is the threshold t, we can compute a similar bound based on the fact that the convergence to t is eventually monotonic. It follows that, in each case, the Threshold Problem reduces to exhaustively checking whether $u_n \geq t$ for each $n \in \{0, 1, \ldots, B\}$. Unfortunately this reasoning does not suffice to decide the Threshold Problem. Indeed, we do not know how to decide whether a generic hypergeometric sequence converges to a given rational limit. Further, such convergence questions are intricately linked to open problems concerning algebraic relations for the gamma function (we give further details below).

Contributions

Our primary contributions are:

(a) The Threshold Problem for hypergeometric sequences whose polynomial coefficients are monic and split over an imaginary quadratic field are decidable (Theorem 15).

(b) The Threshold Problem for hypergeometric sequences whose polynomial coefficients are monic and each irreducible factor of pq is either linear or quadratic is decidable subject to the truth of Schanuel's conjecture (Corollary 18).

We delay a formal statement of Schanuel's conjecture to the Preliminaries. For our conditional decidability results, we note that only termination is conditional on Schanuel's conjecture and that correctness of our procedure is unconditional (Remark 24). Corollary 18 follows from the more general result:

(c) The Threshold Problem for hypergeometric sequences whose monic polynomial coefficients possess Property S is decidable subject to the truth of Schanuel's conjecture (Theorem 17 in Subsection 4.1).

Polynomials with Property S (Subsection 4.1) lead to classes of hypergeometric sequences with unnested radical and cyclotomic parameters.

Our secondary contribution concerns the *Membership Problem* for hypergeometric sequences. Given a hypergeometric sequence $\langle u_n \rangle_{n=0}^{\infty}$ and target $t \in \mathbb{Q}$, Membership asks to determine whether there is an $n \in \mathbb{N}_0$ for which $u_n = t$.

(d) For classes of hypergeometric sequences where we establish (un)conditional decidability of the Threshold Problem, we also obtain (un)conditional decidability of the Membership Problem. This contribution is a straightforward corollary of the following observation: for hypergeometric sequences, decidability of the Membership Problem reduces to that of the Threshold Problem (Proposition 7).

We note this secondary contribution both recovers and extends some of the recent results in work by Kenison et al. [15]. For the avoidance of doubt, we break new ground for the Membership Problem. For example, we have conditional decidability of the Membership Problem for hypergeometric sequences with unnested radical and cyclotomic parameters (both classes fall outside of the remit of the previous works). A concrete subclass is given by those hypergeometric sequences whose polynomial coefficients are of the form $(x^2 - \ell_1)(x^2 - \ell_2) \cdots (x^2 - \ell_d)$ where $\ell_1, \ldots, \ell_d \in \mathbb{Z}$.

Approach

As previously mentioned, an obstacle that prevents us from settling decidability of the Threshold Problem for hypergeometric sequences is determining whether a given hypergeometric sequence converges to some rational limit.

To each hypergeometric sequence $\langle u_n \rangle_{n=0}^{\infty}$ satisfying (1), we associate the *shift quotient* $r(x) := q(x)/p(x) \in \mathbb{Q}(x)$. It is clear that the terms of $\langle u_n \rangle_{n=0}^{\infty}$ are given by a sequence of partial products such that the nth term is given thus: $u_n = u_0 \cdot \prod_{k=0}^{n} r(k)$. Without loss of generality, we can a normalise a sequence with $u_0 \neq 0$ by assuming that $u_0 = 1$. (We note that the Threshold Problem is trivial to decide when $u_0 = 0$.) Thus our consideration of the Threshold Problem reduces to analysing the sequence of partial products $\langle \prod_{k=0}^{n} r(k) \rangle_{n=0}^{\infty}$. In all problem instances where the Threshold Problem is not trivial to determine, we employ a classical theorem in analysis (Theorem 5) that permits us to write the limit of the sequence $\langle \prod_{k=0}^{n} r(k) \rangle_{n=0}^{\infty}$ as a quotient of two finite products involving the gamma function. Thus the Threshold Problem for hypergeometric sequences reduces to testing an equality between gamma products. Our novel approach leverages algebraic and transcendental properties to settle such equality tests. For example, we frequently employ the algebraic independence of transcendental constants π and e^{π} (a celebrated consequence of Nesterenko's work on modular functions [20]).

There is a large corpus of research connecting hypergeometric sequences and the gamma function (often under the guise of infinite product identities [1, 5, 7]). This is particularly relevant for our approach (as described above) and sets us apart from previous papers on the Membership Problem in this setting [15, 22]. As a nod to the wider appeal of our approach, let us consider two examples from the literature on numeric and symbolic computation.

▶ **Example 1** (Numerical evaluation of the Kepler–Bouwkamp constant [5, Section 4])**.** This example closely follows work by Chamberland and Straub [5]. Those authors demonstrate the use of hypergeometric sequences to efficiently approximate certain numerical constants given by infinite products. One such example is the Kepler–Bouwkamp constant $\prod_{k=3}^{\infty} \cos(\pi/k)$. The convergence of this infinite product is notoriously slow: the error bound between the approximation $\prod_{k=3}^{10^4} \cos(\pi/k)$ and the Kepler–Bouwkamp constant is 10^{-4}.

Let us consider a Padé approximation that uses hypergeometric sequences with quadratic parameters. We recall that the $[2,2]$-*Padé approximant* of a function $f(x)$ is the rational function $q(x)/p(x)$ where $p, q \in \mathbb{Z}[x]$ are quadratic polynomials for which the Maclaurin series of $q(x)/p(x)$ agrees with that of $f(x)$ up to order 4. For example, the $[2,2]$-Padé approximant of $\cos(x)$ is

$$r_2(x) := \frac{12 - 5x^2}{12 + x^2} = \cos(x) + O(x^6).$$

Now consider

$$\prod_{k=3}^{\infty} r_2(\pi/k) = \prod_{k=3}^{\infty} \frac{12k^2 - 5\pi^2}{12k^2 + \pi^2} = \frac{\Gamma(3 - \frac{\mathrm{i}}{6}\sqrt{3}\pi)\Gamma(3 + \frac{\mathrm{i}}{6}\sqrt{3}\pi)}{\Gamma(3 - \frac{1}{6}\sqrt{15}\pi)\Gamma(3 + \frac{1}{6}\sqrt{15}\pi)}.$$

Here the evaluation as a quotient of two gamma products follows from Theorem 5. We pass this evaluation to any modern computer algebra system and determine that the error between $\prod_{k=3}^{\infty} r_2(\pi/k)$ and the Kepler–Bouwkamp constant is bounded by 10^{-3}.

Loosely speaking, the $[2, 2]$-Padé approximant leverages a hypergeometric sequence with quadratic parameters.[1] Higher-order approximants will give a closer numerical estimate via hypergeometric sequences with higher-degree parameters.

▶ **Example 2** (Evaluation of a gamma product with cyclotomic parameters)**.** Here we include a simple (yet concrete) example [4, pages 4–6] of the difficulty in determining whether a hypergeometric sequence converges to a rational limit. Consider

$$\prod_{k=2}^{\infty} \frac{k^5 - 1}{k^5 + 1} = \frac{2 \cdot \Gamma(-\omega_{10})\Gamma(\omega_{10}^2)\Gamma(-\omega_{10}^3)\Gamma(\omega_{10}^4)}{5 \cdot \Gamma(\omega_{10})\Gamma(-\omega_{10}^2)\Gamma(\omega_{10}^3)\Gamma(-\omega_{10}^4)} \tag{2}$$

where $\omega_{10} = \mathrm{e}^{2\pi\mathrm{i}/10}$ and, once again, the right-hand side is derived from Theorem 5. As noted by the authors of [4], it is not known whether the limit in (2) is even algebraic.

Whilst the state of the art cannot generally handle the evaluation of expressions given by gamma products, many works in the literature have established identities for restricted classes of products (a non-exhaustive list includes [5, 18, 21, 27, 34, 35]). Our connection to such interests stems from our approach herein: our reduction-step leaves us to determine whether the ratio of two gamma products (as above) is rational.

Related Work

Membership for Hypergeometric Sequences. Two recent works consider the Membership Problem for hypergeometric sequences [15, 22]. In both of these works, the authors use p-adic techniques and divisibility arguments (in stark contrast to the approach herein). It is worth noting that such techniques seem appropriate only for the Membership Problem and not the Threshold Problem.

The authors of [22] establish decidability of the Membership Problem for the class of hypergeometric sequences with rational parameters. Closer to our setting, the authors of [15] establish decidability of the Membership Problem for the class of hypergeometric sequences whose polynomial coefficients (as in (1)) are both monic and split over a quadratic field. By comparison to [15], we establish decidability of not only the Membership Problem, but also the Threshold Problem (Theorem 15) for hypergeometric sequences whose monic polynomial coefficients split over an imaginary quadratic field. We note our result for the Membership Problem is weaker for sequences whose monic polynomial coefficients split over a real quadratic field: in this setting we are limited to conditional decidability (Proposition 7 and Corollary 18).

[1] The quadratics $12k^2 - 5\pi^2$ and $12k^2 + \pi^2$ do not have rational coefficients, but in Appendix B we demonstrate how our approach handles decidability of the Threshold Problem in this example.

Positivity for P-finite sequences. Identities for P-finite sequences are frequent in the literature; however, as noted by Kauers and Pillwein, "in contrast,... almost no algorithms are available for inequalities" in this setting [12]. Determining whether the terms of a P-finite sequence are non-negative has garnered much attention in recent works (see, for example, [10, 12, 13, 28]). On the one hand, these works handle higher-order P-finite sequences than the hypergeometric sequences we consider. On the other hand, the algorithms described in the above studies are restricted in their applicability (placing syntactic restrictions on the polynomial coefficients) and termination is not guaranteed for all initial values. Indeed, genericity of initial conditions (in the sense that, the growth rate of a recurrence sequence is determined by a positive dominant eigenvalue) is required for the algorithms in [10, 12]. Additionally, determining whether the initial conditions of a given sequence are generic is an open problem (even at low orders) [13].

Positivity for C-finite sequences. It is easily seen that the Threshold Problem for C-finite sequences reduces to the Positivity Problem for C-finite sequences. Recall that *Positivity* asks to determine whether all terms in a sequence lie above the threshold zero (so the variant of the Threshold Problem where $t = 0$). This reduction is straightforward: to determine whether $u_n \geq t$ for each $n \in \mathbb{N}_0$ we can equivalently ask whether $v_n := u_n - t \geq 0$ for each $n \in \mathbb{N}_0$. Observe that $\langle v_n \rangle_{n=0}^{\infty}$ is C-finite since it is given by the difference of two C-finite sequences $\langle u_n \rangle_{n=0}^{\infty}$ and $\langle t \rangle_{n=0}^{\infty}$ and we are done.

Decidability of the Positivity Problem for C-finite sequences is considered a challenging open problem. Further, Positivity and its variants have garnered much research interest in recent works [9, 14, 23, 24, 25, 26]. Akin to our focus on restricted classes of hypergeometric sequences herein, the authors of [14] and [23] consider restricted classes of C-finite sequences: both of those works place restrictions on the algebraic properties of the associated recurrence relations.

Structure and Outline

This paper is structured as follows. In the next section we gather together relevant preliminary material. In Sections 3 and 4, we establish (un)conditional decidability of the Threshold Problem for classes of hypergeometric sequences. In one sense, Section 3 gives an overview of our approach in the setting of hypergeometric sequences with quadratic parameters. In Section 4, we introduce the class of polynomials with Property S (Subsection 4.1) and then show that the Threshold Problem for hypergeometric sequences whose monic polynomial coefficients possess Property S is decidable subject to the truth of Schanuel's conjecture (Theorem 17 in Subsection 4.2). We make suggestions for future avenues of research in the conclusion (Section 5). Proofs omitted from the main text are given in Appendix A. Appendix B contains a worked example related to the Kepler–Bouwkamp (Example 1) and demonstrates an application of Schanuel's conjecture.

2 Preliminaries

The Gamma Function. The approach herein relies on transcendence theory for the gamma function Γ where

$$\Gamma(z) = \int_0^{\infty} x^{z-1} \mathrm{e}^{-x}\, dx \quad \text{for } z \in \mathbb{C} \text{ with } \mathrm{Re}(z) > 0.$$

It is possible to analytically extend the domain of Γ to the whole complex plane minus the non-positive integers where the function has simple poles. We briefly recall standard results for the gamma function. Further details and historical accounts are given in a number of sources (cf. [2, 37]).

The standard relations for the gamma function give the functional identities: the *recurrence* (or *translation*) *property* $\Gamma(z+1) = z\Gamma(z)$ for $z \notin \mathbb{Z}$ and the *reflection property* $\Gamma(z)\Gamma(1-z) = \pi/\sin(\pi z)$ for $z \notin \mathbb{Z}$. In the domain of the gamma function, repeated application of the translation property leads to the following "rising factorial" identity. For $n \in \{1, 2, \ldots\}$, we have

$$\frac{\Gamma(z+n)}{\Gamma(z)} = z(z+1)\cdots(z+n-1).$$

Similarly, the "falling factorial" identity is given by

$$\frac{\Gamma(z+1)}{\Gamma(z-n+1)} = z(z-1)\cdots(z-n+1).$$

The next technical lemma is derived from the aforementioned properties of the gamma function. We employ the notation $\frac{1}{2}\mathbb{Z}$ for the set of integers and half-integers.

▶ **Lemma 3.** *Let $\rho \in \frac{1}{2}\mathbb{Z}$. Suppose that $w \in \mathbb{C}$ is an algebraic number such that both $\rho + w$ and $\rho - w$ lie in the domain of the gamma function and $w \notin \frac{1}{2}\mathbb{Z}$. Up to multiplication by an algebraic number, we have the following equalities:*

$$\Gamma(\rho+w)\Gamma(\rho-w) = \begin{cases} \dfrac{2\pi \mathrm{i} e^{\pi w \mathrm{i}}}{w(1-e^{2\pi w \mathrm{i}})} & \text{if } \rho \text{ is an integer, or} \\ \dfrac{2\pi e^{\pi w \mathrm{i}}}{e^{2\pi w \mathrm{i}}+1} & \text{if } \rho \text{ is a half-integer.} \end{cases}$$

Proof. Let us apply the rising and falling factorial identities (as appropriate to the sign of ρ). Then, up to multiplication by an algebraic number, we have the following equalities:

$$\Gamma(\rho+w)\Gamma(\rho-w) = \begin{cases} \Gamma(w)\Gamma(-w) & \text{if } \rho \text{ is an integer, or} \\ \Gamma(1/2+w)\Gamma(1/2-w) & \text{if } \rho \text{ is a half-integer.} \end{cases}$$

Consider the first of the two cases above. The reflection and recurrence formulas lead to

$$\Gamma(w)\Gamma(-w) = \frac{\Gamma(w)\Gamma(1-w)}{-w} = \frac{\pi}{-w\sin(\pi w)} = -\frac{2\pi \mathrm{i}}{w(e^{\pi w \mathrm{i}} - e^{-\pi w \mathrm{i}})}.$$

For the second case, we employ the cosine variant of Euler's reflection formula to obtain

$$\Gamma(1/2+w)\Gamma(1/2-w) = \frac{\pi}{\cos(\pi w)} = \frac{2\pi}{e^{\pi w \mathrm{i}} + e^{-\pi w \mathrm{i}}}.$$

The equalities in the statement of the lemma quickly follow. ◀

Decidability and Reduction Results. Recall that a rational-valued sequence $\langle u_n \rangle_{n=0}^{\infty}$ is *hypergeometric* if it satisfies a first-order recurrence relation of the form (1) with polynomial coefficients $p, q \in \mathbb{Z}[x]$. Due to space restrictions, we omit the proofs of Lemma 4, Proposition 6, and Proposition 7 from the main text. Each proof is included in Appendix A.

The following straightforward lemma appears in previous works [13, 22].

▶ **Lemma 4.** *Consider the class of hypergeometric sequences $\langle u_n \rangle_n$ whose shift quotients $r(n)$ either diverge to $\pm\infty$ or converge to a limit ℓ with $|\ell| \neq 1$. For this class, the Membership and Threshold Problems are both decidable.*

Thus to decide the Membership and Threshold Problems for hypergeometric sequences, we need only consider the sequences whose shift quotients $r(n)$ converge to ± 1 as $n \to \infty$. We say an infinite product $\prod_{k=0}^{\infty} r(k)$ *converges* if the sequence of partial products converges to a finite non-zero limit (otherwise the product is said to diverge). Recall the following classical theorem ([37, §12] and [5]).

▶ **Theorem 5.** *Consider the rational function*

$$r(k) := \frac{c(k+\alpha_1)\cdots(k+\alpha_m)}{(k+\beta_1)\cdots(k+\beta_{m'})}$$

where we suppose that each $\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_{m'}$ is a complex number that is neither zero nor a negative integer. The infinite product $\prod_{k=0}^{\infty} r(k)$ converges to a finite non-zero limit only if $c = 1$, $m = m'$, and $\sum_j \alpha_j = \sum_j \beta_j$. Further, the value of the limit is given by

$$\prod_{k=0}^{\infty} r(k) = \prod_{j=1}^{m} \frac{\Gamma(\beta_j)}{\Gamma(\alpha_j)}.$$

With Theorem 5 in mind, it is useful to introduce the following terminology for shift quotients. We call a rational function $r(k)$ (as above) *harmonious* if $r(k)$ satisfies the assumptions $c = 1$, $m = m'$, and $\sum_j \alpha_j = \sum_j \beta_j$. From Theorem 5, it is immediately apparent that a hypergeometric sequence $\langle u_n \rangle_n$ with shift quotient r converges to a finite non-zero limit only if r is harmonious.

Related to the assumptions in Theorem 5 (and Proposition 6 below), when considering Membership and Threshold we can assume without loss of generality that the roots $\alpha_1, \ldots, \alpha_m$ of the coefficient q (as in (1)) are neither zero nor negative integers. For otherwise, a hypergeometric sequence eventually hits zero and is identically zero thereafter. In [22], Nosan et al. establish Propositions 6 and 7 for rational parameters. The proof of Proposition 6 given in Appendix A is all but identical to the proof of Proposition 2 in [22].

▶ **Proposition 6.** *Let $\langle u_n \rangle_n$ be a hypergeometric sequence whose shift quotient is given by a ratio of two polynomials with real coefficients. For such sequences, the Membership and Threshold Problems are both Turing-reducible to the following decision problem. Given $d \in \mathbb{N}$, $\alpha_1, \ldots, \alpha_d \in \mathbb{C} \setminus \mathbb{Z}_{<0}$ (the roots of some $P(x) \in \mathbb{R}[x]$), and $\beta_1, \ldots, \beta_d \in \mathbb{C} \setminus \mathbb{Z}_{<0}$ (the roots of some $Q(x) \in \mathbb{R}[x]$), determine whether*

$$\frac{\Gamma(\beta_1)\cdots\Gamma(\beta_d)}{\Gamma(\alpha_1)\cdots\Gamma(\alpha_d)} = t$$

for $t \in \mathbb{Q} \setminus 0$.

To be absolutely clear, the fact that $t \in \mathbb{Q}$ is non-zero in Proposition 6 follows directly from the infinite product in Theorem 5 converging to a non-zero limit.

Our (un)conditional decidability results for the Membership Problem follow from the next proposition. This proposition can be deduced from the work in [22] (a straightforward proof is given in Appendix A).

▶ **Proposition 7.** *For hypergeometric sequences, decidability of the Membership Problem Turing-reduces to that of the Threshold Problem.*

Number Fields. We recall standard results for quadratic fields below (cf. [33, Chapter 3]). A number field K is *quadratic* if $[K : \mathbb{Q}] = 2$. A field K is quadratic if and only if there is a square-free integer d such that $K = \mathbb{Q}(\sqrt{d})$. Further, a quadratic field $\mathbb{Q}(\sqrt{d})$ is *imaginary* if $d < 0$.

▶ **Theorem 8.** *Suppose that $d \in \mathbb{Z}$ is square-free. Then the algebraic integers of $\mathbb{Q}(\sqrt{d})$ are given by $\mathbb{Z}[\sqrt{d}]$ if $d \not\equiv 1 \pmod 4$ or $\mathbb{Z}[1/2 + \sqrt{d}/2]$ if $d \equiv 1 \pmod 4$.*

We include the following straightforward lemma for ease of reference.

▶ **Lemma 9.** *Let $\mathbb{L}/\mathbb{Q}$ be a finite Galois extension. Suppose that $\mathcal{P} \in \mathbb{L}(X_1, \ldots, X_m)$ is a polynomial such that $\mathcal{P}(s_1, \ldots, s_m) = 0$ with $(s_1, \ldots, s_m) \in \mathbb{C}^m$. Then there is a polynomial $\mathcal{Q} \in \mathbb{Q}(X_1, \ldots, X_m)$ such that $\mathcal{Q}(s_1, \ldots, s_m) = 0$.*

Proof. Let $\mathcal{P} = \sum_{(t_1,\ldots,t_m)} c_{(t_1,\ldots,t_m)} X_1^{t_1} X_2^{t_2} \cdots X_m^{t_m}$ and for each $\sigma \in G$ (the Galois group of $\mathbb{L}/\mathbb{Q}$) let

$$\sigma(\mathcal{P}) = \sum_{(t_1,\ldots,t_m)} \sigma(c_{(t_1,\ldots,t_m)}) X_1^{t_1} X_2^{t_2} \cdots X_m^{t_m}.$$

Let $\mathcal{Q} = \mathrm{N}_{\mathbb{L}/\mathbb{Q}}(\mathcal{P}) := \prod_{\sigma \in G} \sigma(\mathcal{P})$. It is clear that each of the coefficients of the polynomial $\mathcal{Q}$ is rational since the coefficients are invariant under the action of the group G. Further,

$$\mathcal{Q}(s_1, \ldots, s_m) = \mathcal{P}(s_1, \ldots, s_m) \prod_{\sigma \in G \setminus e_G} \sigma(\mathcal{P})(s_1, \ldots, s_m) = 0,$$

as desired. ◀

Transcendental Number Theory. The transcendence degree of a field extension is a measure of the size of the extension. In fact, for finitely generated extensions of $\mathbb{L}/\mathbb{Q}$ (such as those that we consider), the transcendence degree indicates the largest cardinality of an algebraically independent subset of $\mathbb{L}$ over $\mathbb{Q}$. For a field extension $\mathbb{L}/\mathbb{Q}$, a subset $\{\xi_1, \ldots, \xi_n\} \subset \mathbb{L}$ is *algebraically independent* over $\mathbb{Q}$ if for each polynomial $P(X_1, \ldots, X_n) \in \mathbb{Q}[X_1, \ldots, X_n]$ we have that $P(\xi_1, \ldots, \xi_n) = 0$ only if P is identically zero.

It is useful to recall the Gelfond–Schneider Theorem that establishes the transcendentality of α^β for algebraic numbers α and β except for the cases where $\alpha = 0, 1$ or β is rational.

Schanuel's conjecture is a unifying prediction in transcendental number theory. If Schanuel's conjecture is true, then it generalises several of the principal results in transcendental number theory such as: the Gelfond–Schneider Theorem, the Lindemann–Weierstrass Theorem, and Baker's theorem (cf. [16, 3, 36]). The conjecture makes the following prediction: for $\xi_1, \ldots, \xi_n$ rationally linearly independent complex numbers, there is a subset of $\{\xi_1, \ldots, \xi_n, \mathrm{e}^{\xi_1}, \ldots, \mathrm{e}^{\xi_n}\}$ of size at least n that is algebraically independent over $\mathbb{Q}$.

▶ **Conjecture 10** (Schanuel). *Suppose that $\xi_1, \ldots, \xi_n \in \mathbb{C}$ are linearly independent over the rationals $\mathbb{Q}$. Then the transcendence degree of the field extension $\mathbb{Q}(\xi_1, \ldots, \xi_n, \mathrm{e}^{\xi_1}, \ldots, \mathrm{e}^{\xi_n})$ over $\mathbb{Q}$ is at least n.*

3 Hypergeometric Sequences with Quadratic Parameters

As an appetiser to the proofs of Theorems 15 and 17, we introduce our approach by first establishing decidability of the Threshold Problem for hypergeometric sequences with Gaussian integer parameters (Proposition 11 below). We also include a worked example in Example 12. Recall that the *Gaussian integers* $\mathbb{Z}[\mathrm{i}]$ are those complex numbers of the form $a + b\mathrm{i}$ for which $a, b \in \mathbb{Z}$.

▶ **Proposition 11.** *The Threshold Problem for hypergeometric sequences whose polynomial coefficients are monic and split over* $\mathbb{Q}(\mathrm{i})$ *are decidable.*

Proof. By Theorem 5 and Proposition 6, we need only consider hypergeometric sequences with harmonious shift quotients. Since the polynomial coefficients $p, q \in \mathbb{Z}[x]$ are monic, the roots and poles of the shift quotient are integers in $\mathbb{Q}(i)$; that is to say, they lie in $\mathbb{Z}[i]$. It follows from Lemma 3 that each such instance $(\langle u_n \rangle_n, t)$ of the Threshold Problem reduces to testing an equality of the form

$$\theta \pi^{\ell} \prod_m (\mathrm{e}^{b_m \pi} - \mathrm{e}^{-b_m \pi})^{\varepsilon_m} = t. \tag{3}$$

Here θ is rational and non-zero, $\ell \in \mathbb{Z}$, each pair $\Gamma(a_m + b_m \mathrm{i})\Gamma(a_m - b_m \mathrm{i})$ from Proposition 6 contributes a term $(\mathrm{e}^{b_m \pi} - \mathrm{e}^{-b_m \pi})^{\varepsilon_m}$ in the finite product, and $\varepsilon_m = \pm 1$.

We break the remainder of the proof into several subcases. Without loss of generality, we can assume that not all the roots and poles of r are rational integers, for otherwise testing (3) reduces to the decidable task of testing equality between two rational numbers.

We continue under the assumption that not all the roots and poles of r are rational integers. Let us now consider the product in (3). Up to multiplication by a rational, we can write the left-hand side of (3) in the form

$$\theta \pi^{\ell} \prod_m (\mathrm{e}^{b_m \pi} - \mathrm{e}^{-b_m \pi})^{\varepsilon_m} = \theta \pi^{\ell} \frac{f(\mathrm{e}^{\pi})}{g(\mathrm{e}^{\pi})}$$

where $f, g \in \mathbb{Q}[X]$ are non-trivial polynomials. Observe that $\mathrm{e}^{\pi} = (\mathrm{e}^{\pi \mathrm{i}})^{-\mathrm{i}} = (-1)^{-\mathrm{i}}$; thus, by the Gelfond–Schneider theorem, e^{π} is transcendental. We break the remainder of the proof into two cases depending on the rationality of $f(\mathrm{e}^{\pi})/g(\mathrm{e}^{\pi})$.

Suppose that $f(\mathrm{e}^{\pi})/g(\mathrm{e}^{\pi}) \in \mathbb{Q}$. There are two further subcases to consider: if $\ell = 0$, then, once again, the equality test (3) reduces to deciding whether two rationals are equal; and if $\ell \neq 0$, then the equality test (3) reduces to testing whether π^{ℓ} is equal to a given rational number, which cannot hold for then π is necessarily algebraic.

All that remains is to consider the case where $f(\mathrm{e}^{\pi})/g(\mathrm{e}^{\pi}) \notin \mathbb{Q}$, which we again split into two subcases. If $\ell = 0$, then it is trivial to see that (3) cannot hold as the right-hand side is rational. If $\ell \neq 0$ and we assume, for a contradiction, that (3) holds, then a simple rearrangement of (3) shows that there is a non-trivial polynomial $\mathcal{P} \in \mathbb{Q}[X, Y]$ such that $\mathcal{P}(\pi, \mathrm{e}^{\pi}) = 0$. This contradicts Nesterenko's theorem [20] that π and e^{π} are algebraically independent. We have dispatched each of the subcases and conclude the desired result. ◀

▶ **Example 12.** Suppose that $\langle u_n \rangle_{n=0}^{\infty}$ is the hypergeometric sequence defined by

$$u_n = \frac{n^2 - 4n + 5}{n^2 - 4n + 13} u_{n-1} \text{ with } u_0 = 1.$$

For the Threshold Problem, let us consider the problem instance $(\langle u_n \rangle_n, t)$ with $t \in \mathbb{Q}$. First, we evaluate the associated infinite product:

$$\prod_{k=0}^{\infty} \frac{k^2 - 4k + 5}{k^2 - 4k + 13} = \frac{\Gamma(-2 - 3\mathrm{i})\Gamma(-2 + 3\mathrm{i})}{\Gamma(-2 - \mathrm{i})\Gamma(-2 + \mathrm{i})} = \frac{\sinh(\pi)}{39 \sinh(3\pi)} = \frac{\mathrm{e}^{3\pi}(\mathrm{e}^{2\pi} - 1)}{39\mathrm{e}^{\pi}(\mathrm{e}^{6\pi} - 1)}.$$

Second, as directed by the proof of Proposition 11, decidability of the Threshold Problem in this instance reduces to determining whether the following equality holds:

$$\frac{\mathrm{e}^{3\pi}(\mathrm{e}^{2\pi} - 1)}{39\mathrm{e}^{\pi}(\mathrm{e}^{6\pi} - 1)} = t.$$

A simple rearrangement shows that if the above equality holds, then there is a non-trivial polynomial $p \in \mathbb{Q}[X]$ such that $p(\mathrm{e}^\pi) = 0$, from which we deduce that e^π is algebraic. However, by the Gelfond–Schneider theorem, e^π is transcendental. We have reached a contradiction and deduce that the aforementioned equality cannot hold.

Finally, as described in Proposition 6, the Threshold Problem reduces to an exhaustive search of a computable number of initial terms in the sequence $\langle u_n \rangle_n$. As an aside, by Proposition 7, we also obtain decidability of the Membership Problem for problem instances $(\langle u_n \rangle_n, t)$.

▶ Remark 13. Subject to appropriate changes and by employing Lemma 9, we can extend the result in Proposition 11 from instances of the Membership and Threshold Problems $(\langle u_n \rangle_n, t)$ with $u_0, t \in \mathbb{Q}$ to problem instances with $u_0, t \in \mathbb{L}(\pi, \mathrm{e}^\pi)$ where $\mathbb{L}$ is any finite Galois extension of $\mathbb{Q}$. This extension similarly holds for Theorem 15 (below).

▶ Remark 14. Analogous decision procedures to the proof of Proposition 11 also hold for other famous rings of integers: the Eisenstein, Kummer, and Kleinian integers. Recall that the *Eisenstein integers* are the elements of $\mathbb{Z}[\zeta_3] = \{a + b\zeta_3 : a, b \in \mathbb{Z}\}$ where $\zeta_3 := \mathrm{e}^{2\pi \mathrm{i}/3}$. Similarly, the *Kummer integers* are the elements of $\mathbb{Z}[\sqrt{-5}] = \{a + b\sqrt{-5} : a, b \in \mathbb{Z}\}$. Finally, the *Kleinian integers* are the elements of $\mathbb{Z}[\mu] = \{a + b\mu : a, b \in \mathbb{Z}\}$ where $\mu = -1/2 + \sqrt{-7}/2$.

The claims for decidability in Remark 14, follow from the next theorem. We establish decidability of the Threshold Problem for hypergeometric sequences whose parameters are drawn from the ring of integers of an imaginary quadratic number field.

▶ **Theorem 15.** *The Threshold Problem for hypergeometric sequences whose polynomial coefficients are monic and split over an imaginary quadratic number field is decidable.*

Proof. Mutatis mutandis, the proof of Theorem 15 follows the approach in Proposition 11. For the sake of brevity, we shall indicate only the major changes to Proposition 11 here. Consider the ring of integers of an imaginary quadratic field $\mathbb{Q}(\sqrt{d})$ where $-d \in \mathbb{N}$ is square-free. By Theorem 8, there are two cases to consider: first, when $d \not\equiv 1 \pmod 4$ and second, when $d \equiv 1 \pmod 4$.

We concentrate on the changes to the proof of Proposition 11 when $d \not\equiv 1 \pmod 4$. Like before, we can use the recurrence formula to write $\Gamma(a + b\sqrt{d}) = \theta\Gamma(b\sqrt{d})$ where $\theta \in \mathbb{N}$. Thus all that remains is to evaluate products $\Gamma(b\sqrt{d})\Gamma(-b\sqrt{d})$ of conjugate elements. By the reflection formula, we have

$$\Gamma(b\sqrt{d})\Gamma(-b\sqrt{d}) = -\frac{\pi}{b\sqrt{d}\sin(b\pi\sqrt{d})} = \frac{2\pi}{b\sqrt{-d}(\mathrm{e}^{\pi b\sqrt{-d}} - \mathrm{e}^{-\pi b\sqrt{-d}})}.$$

The important update is the product in (3). In our new setting, the product takes the form

$$\prod_m (\mathrm{e}^{\pi b\sqrt{-d}} - \mathrm{e}^{-\pi b\sqrt{-d}})^{\varepsilon_m}.$$

Observe that $\mathrm{e}^{\pi\sqrt{-d}}$ is transcendental (once again by Gelfond–Schneider) and that for each $-d \in \mathbb{N}$ the numbers π and $\mathrm{e}^{\pi\sqrt{-d}}$ are algebraically independent over $\mathbb{Q}$ [20, Corollary 6]. The rest of the proof in this case follows as before.

In the second case where $d \equiv 1 \pmod 4$ we must additionally deal with contributions of the form $\Gamma(b/2 + b\sqrt{d}/2)\Gamma(b/2 - b\sqrt{d}/2)$. This setting introduces cases where $2 \nmid b$, which we resolve by repeated application of the recurrence formula and the cosine variant of Euler's reflection formula. Indeed, we have

$$\Gamma(1/2 + b\sqrt{d}/2)\Gamma(1/2 - b\sqrt{d}/2) = \frac{\pi}{\cos(\pi b\sqrt{d}/2)} = \frac{2\pi}{\mathrm{e}^{\pi b\sqrt{-d}/2} + \mathrm{e}^{-\pi b\sqrt{-d}/2}},$$

and so we can construct an updated version of the product in (3). This update and analogous arguments for the transcendental properties of $e^{\pi\sqrt{-d}/2}$ let us conclude decidability in this case too. ◀

4 Conditional Decidability Subject to Schanuel's conjecture

In this section we shall give a generalisation (Theorem 17) of the decidability results in Section 3. The result applies to a strictly larger class of hypergeometric sequences; however, we sacrifice unconditional decidability. Briefly, termination of the decidability procedure in Theorem 17 is dependent on the truth of Schanuel's conjecture (further details are given in Remark 24). The motivation for studying reachability problems for this larger class arises from interest in the literature for limits of hypergeometric sequences with unnested radical and cyclotomic parameters (see Subsection 4.2).

Let us first describe the class of hypergeometric sequences for which we establish conditional decidability of the Membership and Threshold Problems. This will require us to introduce some notations and Property S below.

Consider $\mathcal{R}_f$ the multiset of roots of a given monic polynomial $f \in \mathbb{Z}[x]$ and $\mathcal{V}_f$ the multiset of irrational roots of f. We define the graph $\mathcal{G}_f := (\mathcal{V}_f, \mathcal{E}_f)$ with vertex set $\mathcal{V}_f$ and edge set $\mathcal{E}_f$. Here $\mathcal{E}_f$ is encoded using an adjacency matrix A (whose rows and columns are indexed by $\mathcal{V}_f$) as follows. For distinct $u, v \in \mathcal{V}_f$, let $A(u,v) = 1$ if $u = \rho + w$ and $v = \rho - w$ for some $\rho \in \frac{1}{2}\mathbb{Z}$ and w an algebraic number not in $\frac{1}{2}\mathbb{Z}$. Otherwise, let $A(u,v) = 0$.

▶ **Property S.** *We say that f has Property S if $\mathcal{G}_f$ admits a perfect matching.*

The motivation for introducing Property S is as follows: in combination with Lemma 3 and subject to the truth of Schanuel's conjecture, we can circumvent certain hard problems concerning the evaluation of gamma products if the input parameters possess certain symmetries. The main result of this section is a conditional decidability result for the class of hypergeometric sequences whose polynomial coefficients are monic and possess Property S.

▶ **Theorem 17.** *The Threshold Problem for hypergeometric sequences whose polynomial coefficients (as in* (1)*) are both monic and have Property S is decidable subject to the truth of Schanuel's conjecture.*

The remainder of this section is structured as follows. In Subsection 4.1 we list classes of polynomials with Property S and highlight straightforward corollaries (Corollaries 18, 22, and 23) of Theorem 17. In Subsection 4.2 we prove Theorem 17.

4.1 Polynomials with Property S

Even Polynomials. It is immediate that even polynomials with at least one irrational root possess Property S. Suppose that $f \in \mathbb{Z}[x]$ is a monic even polynomial. Then $f(-x) = f(x)$ and so $\mathcal{G}_f$ admits a perfect matching, as desired. More generally, we note that a horizontal translation of such a polynomial, say $\tilde{f}$, for which $\tilde{f}(\rho - x) = \tilde{f}(\rho + x)$ where $\rho \in \frac{1}{2}\mathbb{Z}$, also has Property S.

Quadratic Polynomials. Every irreducible monic quadratic polynomial in $\mathbb{Z}[x]$ possesses Property S. This observation is straightforward: consider an irreducible monic quadratic polynomial $x^2 + bx + c \in \mathbb{Z}[x]$. The roots of said quadratic satisfy $-\frac{b}{2} \pm \frac{\sqrt{b^2-4c}}{2}$. We note that $-\frac{b}{2} \in \frac{1}{2}\mathbb{Z}$ and $b^2 - 4c \neq 0$.

The following corollary is a straightforward consequence of Theorem 17: we have conditional decidability of the Threshold Problem for hypergeometric sequences whose parameters are drawn from the rings of integers of quadratic number fields.

▶ **Corollary 18.** *The Threshold Problem for hypergeometric sequences whose polynomial coefficients are monic with irreducible factors that are either linear or quadratic is decidable subject to the truth of Schanuel's conjecture.*

We note that the assumption in Corollary 18 permits us to draw sequence parameters from the integers of any number of quadratic fields in order to establish conditional decidability of both the Threshold and Membership Problems. This is in stark contrast to the work in [15] that establishes decidability of Membership for hypergeometric sequences whose parameters are drawn from the integers of a single quadratic field.

Algebraic Numbers with Rational Real Part. Consider the class $\mathcal{C}$ of monic irreducible polynomials in $\mathbb{Q}[x]$ that possess a root with rational real part. Trivially, a linear polynomial with rational coefficients is always a member of $\mathcal{C}$ since the single root of the polynomial is rational. It is straightforward to see that an irreducible quadratic polynomial $x^2+bx+c \in \mathbb{Q}[x]$ is in $\mathcal{C}$ if and only if $b^2-4c<0$. Dilcher, Noble, and Smyth [6] achieve a complete classification of the class $\mathcal{C}$ with the following result.

▶ **Theorem 19** ([6, Theorem 1])**.** *Let f be a polynomial of degree at least three. Then $f \in \mathcal{C}$ if and only if $f(x) = g((x-\rho)^2)$ for some $\rho \in \mathbb{Q}$ and monic irreducible $g \in \mathbb{Q}[X]$ that has a negative real root. In this case, f has a root with a rational real part ρ.*

The following corollary is key to understanding the connection to Property S and demonstrates the 2-fold rotational symmetry of the set of roots of a polynomial in $\mathcal{C}$.

▶ **Corollary 20** ([6, Corollary 2])**.** *Suppose that $f \in \mathcal{C}$ has degree at least three. The roots of f that have rational real part have the same real part ρ. Further, we have $f(\rho-x) = f(\rho+x)$.*

The trivial lemma below shows that if an algebraic integer has rational real part, then said real part lies in $\frac{1}{2}\mathbb{Z}$.

▶ **Lemma 21.** *Let α be an algebraic integer with $\mathrm{Re}(\alpha) \in \mathbb{Q}$. Then $\mathrm{Re}(\alpha) \in \frac{1}{2}\mathbb{Z}$.*

Proof. Since $\mathrm{Re}(\alpha) = \frac{1}{2}(\alpha+\overline{\alpha})$, we deduce that $\alpha+\overline{\alpha} \in \mathbb{Q}$. Observe that both α and $\overline{\alpha}$ are algebraic integers and so $\alpha+\overline{\alpha}$ is too due to the closure of the ring of algebraic integers. We deduce that $\alpha+\overline{\alpha} \in \mathbb{Z}$, from which the desired result follows. ◀

When we combine the result in Theorem 17 with the observations in Corollary 20 and Lemma 21 we obtain the following corollary.

▶ **Corollary 22.** *The Threshold Problem for hypergeometric sequences whose polynomial coefficients are monic with irreducible factors in $\mathcal{C}$ is decidable subject to the truth of Schanuel's conjecture.*

Unnested Radicals and Cyclotomic Polynomials. We now highlight the class of hypergeometric sequences with parameters determined by unnested radicals and cyclotomic polynomials. The limits of such sequences are considered in both [29, pp. 753–757] and [7]. In the sequel we use Φ_d to denote the dth cyclotomic polynomial. We state and prove the following corollary of Theorem 17.

▶ **Corollary 23.** *The Threshold Problem for hypergeometric sequences whose polynomial coefficients have irreducible factors of the form $x^d - a$ with $d \in 2\mathbb{Z}$, or Φ_d with $d \in 4\mathbb{Z}$ is decidable subject to the truth of Schanuel's conjecture.*

Proof. In light of Theorem 17, it is sufficient to prove that non-linear polynomials of the form $x^d - a$ with $d \in 2\mathbb{Z}$, or Φ_d with $d \in 4\mathbb{Z}$ have Property S.

We first consider irreducible factors of the form $x^d - a \in \mathbb{Z}[x]$. The roots of $x^d - a$ are unnested radicals of the form $\sqrt[n]{a}\omega_m^j$ for $j \in \{0, \ldots, m-1\}$ where $\omega_d := \mathrm{e}^{2\pi \mathrm{i}/d}$. Recall that $x^d - a \in \mathbb{Z}[x]$ is irreducible if a is not the Nth power of an element of $\mathbb{Q}$ for some $N > 1$ with $N \mid d$ (cf. [19]). When d is even, it follows that $x^d - a$ is even and so possesses Property S.

The cyclotomic polynomial Φ_d is irreducible and its roots are the primitive dth roots of unity. Under the assumption that d is a multiple of four, it is straightforward to show that Φ_d is even. It follows that for such d, Φ_d has Property S. ◀

We note with our approach we cannot lift the additional assumption that $d \in 4\mathbb{Z}$ for cyclotomic factors: the set of primitive 18th roots of unity $\{\omega_{18}, \omega_{18}^5, \omega_{18}^7, \omega_{18}^{11}, \omega_{18}^{13}, \omega_{18}^{17}\}$ show that Φ_{18} does not have Property S.

4.2 Proof of Theorem 17

We now prove our main result. A worked example, demonstrating our approach, is given in Appendix B.

▶ **Theorem 17.** *The Threshold Problem for hypergeometric sequences whose polynomial coefficients (as in* (1)*) are both monic and have Property S is decidable subject to the truth of Schanuel's conjecture.*

Proof. Let $\langle u_n \rangle_{n=0}^{\infty}$ be a hypergeometric sequence whose polynomial coefficients are both monic and satisfy Property S. We assume without loss of generality that the associated shift quotient $r(x) := q(x)/p(x)$ is harmonious. As previously noted, each instance of the Threshold Problem $(\langle u_n \rangle_n, t)$ with $t \in \mathbb{Q}$ reduces to checking an equality of the form

$$\Gamma(\beta_1) \cdots \Gamma(\beta_d) = t\Gamma(\alpha_1) \cdots \Gamma(\alpha_d) \tag{4}$$

where $\{\alpha_1, \ldots, \alpha_d\} =: \mathcal{R}_p$ and $\{\beta_1, \ldots, \beta_d\} =: \mathcal{R}_q$ are the multisets of the roots of the respective polynomial coefficients p and q. The proof is split into two parts: a reduction to an equality testing problem and a proof of decidability subject to the truth of Schanuel's conjecture.

Reduction to Equality Testing. Since pq is a monic polynomial, the rational elements of the multisets $\mathcal{R}_p$ and $\mathcal{R}_q$ are rational integers (so we can assume their contributions are absorbed into the rational parameter t in (4)). Further, under Property S there are perfect matchings on both of the graphs $\mathcal{G}_p$ and $\mathcal{G}_q$, which we denote by $\mathcal{M}_p$ and $\mathcal{M}_q$ respectively. Indeed, we recall that in $\mathcal{G}_p$ the edges in $\mathcal{M}_p$ are of the form $e(\alpha_-, \alpha_+)$ such that $\alpha_\pm = \rho_e \pm w_e$ where $\rho_e \in \frac{1}{2}\mathbb{Z}$ and w_e is an algebraic number not in $\frac{1}{2}\mathbb{Z}$ (and similarly for $\mathcal{G}_q$ and $\mathcal{M}_q$). In this setting, we repeatedly apply the recurrence formula and absorb the resulting algebraic factors into a single term θ in order to rewrite (4) as

$$\prod_{i \in \mathcal{M}_q} \Gamma(w_i)\Gamma(-w_i) = \theta \prod_{j \in \mathcal{M}_p} \Gamma(w_j)\Gamma(-w_j). \tag{5}$$

Consider the set of algebraic numbers $\{w_1, \ldots, w_M\}$ determined by the parameters in (5). We denote by $S' := \{s'_1, \ldots, s'_m\}$ a maximal subset of $\{w_1, \ldots, w_M\}$ for which the elements of $\{\pi, \pi\mathrm{i}\} \cup \pi S'$ are $\mathbb{Q}$-linearly independent (here $\pi S' := \{\pi s'_1, \ldots, \pi s'_m\}$). Then for each $k \in \{1, \ldots, M\}$, write w_k as a $\mathbb{Q}$-linear sum of elements in $\{s'_1, \ldots, s'_m\}$ so that

$$w_k = \frac{x_{k1}}{y_{k1}} s'_1 + \cdots + \frac{x_{km}}{y_{km}} s'_m.$$

We define $s_j := s'_j / \operatorname{lcm}(y_{1j}, y_{2j}, \ldots, y_{Mj})$ for each $j \in \{1, \ldots, m\}$. Now we can write each $w_k \in \{w_1, \ldots, w_M\}$ as a $\mathbb{Z}$-linear sum of elements in the normalised set $S := \{s_1, \ldots, s_m\}$.

We apply Lemma 3 to (5) and, by the preceding paragraph, determine that the problem of testing the equality in (5) reduces to that of determining whether a certain non-trivial polynomial with coefficients in $\mathbb{Q}(\theta)$ vanishes at a given point (this is analogous to the process in Proposition 11 and Example 12). More specifically, we want to test whether a given non-trivial polynomial $P \in \mathbb{Q}(\theta)[X_1, \ldots, X_{4m+4}]$ satisfies

$$P\big(\pi, \pi\mathrm{i}, \pi S, \pi S\mathrm{i}, \mathrm{e}^{\pi}, \mathrm{e}^{\pi\mathrm{i}}, \mathrm{e}^{\pi S}, \mathrm{e}^{\pi S\mathrm{i}}\big) = 0. \tag{6}$$

Here $\pi S\mathrm{i} := \{\pi s_1\mathrm{i}, \ldots, \pi s_m\mathrm{i}\}$, $\mathrm{e}^{\pi S} := \{\mathrm{e}^{\pi s_1}, \ldots, \mathrm{e}^{\pi s_m}\}$, and likewise for $\mathrm{e}^{\pi S\mathrm{i}}$. Note that we need only consider a polynomial in $4m + 4$ variables as the parameters $\{w_1, \ldots, w_M\}$ of our problem instance are given by $\mathbb{Z}$-linear combinations of the elements of $S \cup S\mathrm{i}$. We claim that the equality in (6) cannot hold if Schanuel's conjecture is true. We prove this claim below.

Conditional Decidability Subject to Schanuel's conjecture. Consider the set

$$\mathfrak{S} := \big\{\pi, \pi\mathrm{i}, \pi S, \pi S\mathrm{i}, \mathrm{e}^{\pi}, \mathrm{e}^{\pi\mathrm{i}}, \mathrm{e}^{\pi S}, \mathrm{e}^{\pi S\mathrm{i}}\big\}$$

with cardinality $4m + 4$. We observe that the elements in the subset $\{\pi, \pi\mathrm{i}, \pi S, \pi S\mathrm{i}\} \subset \mathfrak{S}$ are $\mathbb{Q}$-linearly independent. It follows that if Schanuel's conjecture is true, then $\mathfrak{S}$ possesses a subset of cardinality at least $2m + 2$ whose elements are algebraically independent. By construction, this algebraically independent subset is necessarily $\{\pi, \mathrm{e}^{\pi}, \mathrm{e}^{\pi S}, \mathrm{e}^{\pi S\mathrm{i}}\}$ since the $2m + 2$ elements of $\{\pi, \pi\mathrm{i}, \pi S, \pi S\mathrm{i}\}$ are pairwise algebraically dependent and $\mathrm{e}^{\pi\mathrm{i}} = -1$.

We now rewrite the equality in (6) in terms of the (obvious) polynomial $\hat{P}$ that absorbs the algebraically dependent parameters of $S \cup S\mathrm{i}$ into the coefficients. That is to say, we employ a polynomial $\hat{P} \in \mathbb{L}(X_1, \ldots, X_{2m+2})$ where $\mathbb{L}$ is the Galois closure of the number field $\mathbb{Q}(\theta)(S, S\mathrm{i})$ and evaluate $\hat{P}$ on the algebraically independent subset $\{\pi, \mathrm{e}^{\pi}, \mathrm{e}^{\pi S}, \mathrm{e}^{\pi S\mathrm{i}}\}$ of $\mathfrak{S}$. It follows that the equality in (6) holds only if

$$\hat{P}(\pi, \mathrm{e}^{\pi}, \mathrm{e}^{\pi S}, \mathrm{e}^{\pi S\mathrm{i}}) = 0. \tag{7}$$

By Lemma 9, the equality in (7) holds only if there exists a non-trivial polynomial $Q \in \mathbb{Q}[X_1, \ldots, X_{2m+2}]$ such that $Q(\pi, \mathrm{e}^{\pi}, \mathrm{e}^{\pi S}, \mathrm{e}^{\pi S\mathrm{i}}) = 0$. Recall that if Schanuel's conjecture is true, then the elements of the set $\{\pi, \mathrm{e}^{\pi}, \mathrm{e}^{\pi S}, \mathrm{e}^{\pi S\mathrm{i}}\}$ are algebraically independent over $\mathbb{Q}$, from which we deduce that the preceding equality cannot hold.

Subject to the truth of Schanuel's conjecture, we can determine equality tests of the above form. Thus we have conditional decidability of the Threshold Problem for the desired class of hypergeometric sequences. ◀

▶ Remark 24. We note that the equality test $\mathcal{Q}(\pi, \mathrm{e}^{\pi}, \mathrm{e}^{\pi S}, \mathrm{e}^{\pi S\mathrm{i}}) = 0$ can be realised as a proposition in the first-order theory of the reals with exponentiation. Macintyre and Wilkie [17] established decidability of said theory subject to the truth of Schanuel's conjecture. As noted in previous works, careful inspection of Macintyre and Wilkie's algorithm reveals

that correctness is independent of the truth of Schanuel's conjecture. Indeed, Schanuel's conjecture is only used to prove termination. Thus if we apply Macintyre and Wilkie's algorithm to determine whether the equality $\mathcal{Q}(\pi, \mathrm{e}^{\pi}, \mathrm{e}^{\pi S}, \mathrm{e}^{\pi S \mathrm{i}}) = 0$ holds and find the procedure terminates, then the output is certainly correct.

We note that Macintyre and Wilkie's algorithm terminates unless the inputs constitute a counterexample to Schanuel's conjecture. Thus, the process underlying the proof of Theorem 17 presents an interesting prospect in the sense described by Richardson in [31] (see also [32]) "A failure of the [process] to terminate would be even more interesting than [its] success."

▶ Remark 25. Recall Remark 13 where we extended our class of problem instances to include setups with $u_0, t \in \mathbb{L}(\pi, \mathrm{e}^{\pi})$. Under the assumption that Schanuel's conjecture is true we can include a broader range of setups. Notice, for instance, that the algebraic independence of π and e is currently unknown; however, if Schanuel's conjecture is true, then it follows that π and e are algebraically independent. Thus, subject to the truth of Schanuel's conjecture, we can extend our results in Proposition 11, Theorem 15, and Theorem 17 to instances where $u_0, t \in \mathbb{L}(\pi, \mathrm{e})$. In fact, we can go further in this direction since the truth of Schanuel's conjecture implies the algebraic independence of the numbers e, e^{π}, e^{e}, π, π^{π}, π^{e}, 2^{π}, 2^{e}, $\log \pi$, $\log 2$, $\log 3$, $\log \log 2$, $(\log 2)^{\log 3}$, $2^{\sqrt{2}}$, and many more (cf. [30, Conjecture S_7]).

5 Conclusion

Summary. In this paper we establish (un)conditional decidability results for the Threshold Problem for hypergeometric sequences and, as a side-effect, (un)conditional decidability results for the Membership Problem for hypergeometric sequences. Previous works have considered the Membership Problem for hypergeometric sequences [13, 22]; however, the approach in those works cannot handle instances of the Threshold Problem. The novelty of our approach is the combination of a classical convergence result (Theorem 5) with results on the algebraic independence of common mathematical constants.

Obstacles. Let us illustrate an immediate obstacle to the methods herein. We cannot handle parameters drawn from biquadratic fields because the monic polynomials that split over such fields are not necessarily amenable to our approach. Recall that biquadratic fields are a particularly well-behaved class of quartic fields (such as $\mathbb{Q}(\sqrt{5}, \sqrt{13})$ and $\mathbb{Q}(\sqrt{21}, \sqrt{33})$). For example the minimal polynomial $x^4 - 5x^3 - 71x^2 + 120x + 1044$ of $(5 + 3\sqrt{5} + \sqrt{13} + 3\sqrt{65})/4 \in \mathbb{Q}(\sqrt{5}, \sqrt{13})$ does not have Property S. Similarly, the minimal polynomial $x^4 - x^3 - 16x^2 + 37x - 17$ of $(1 + \sqrt{21} + \sqrt{33} - \sqrt{77})/4 \in \mathbb{Q}(\sqrt{21}, \sqrt{33})$ does not have Property S. Both of these examples are taken from [38].

We also note that the class of sequences we can handle does not permit standard operations on the parameters such as addition. Consider, for example, that $\sqrt{2}$ and $\sqrt[4]{2}$ are both unnested radicals whose minimal polynomials satisfy Property S; however, the minimal polynomial $x^4 - 4x^2 - 8x + 2$ of $\sqrt{2} + \sqrt[4]{2}$ does not possess Property S.

It is not clear how to extend our approach to hypergeometric sequences with larger classes of parameters. For example, the parameters herein are all algebraic integers. Even in the restricted setting of hypergeometric sequences with rational parameters (as in the work of Nosan et al. [22]) it is beyond the state of the art to evaluate equalities between associated gamma products. Indeed, for $s \in \{1/6, 1/4, 1/3, 2/3, 3/4\}$ and $n \in \mathbb{N}$, it is known that $\Gamma(n + s)$ is a transcendental number and algebraically independent of π (cf. [36]). However, transcendence of the gamma function at other rational points is not known. It is notable that

for rational parameters determining equality between gamma products is decidable subject to the truth of the Rohrlich–Lang conjecture (which itself concerns multiplicative relations for the gamma function [16, 36]).

Directions for Future Work. We give one class of hypergeometric sequences whose parameters link the related works. Consider the class of hypergeometric sequences whose parameters lie in $\mathbb{Q}(\mathrm{i})$. For such sequences, decidability of both the Membership and Threshold Problems is open.

The sequences in this class generalise the setting discussed here (and in work by Kenison et al. [15]) by removing the condition that the polynomial coefficients of the defining recurrence relation are both monic. Further, results in this direction would extend the discussion of the Membership Problem for hypergeometric sequences with rational parameters in work by Nosan et al. [22] as well as those sequences whose polynomial coefficients have irreducible factors in $\mathcal{C}$ (i.e., each irreducible factor has a root with rational real part) discussed herein.

References

1 Jean-Paul Allouche. Paperfolding infinite products and the gamma function. *Journal of Number Theory*, 148:95–111, March 2015. doi:10.1016/j.jnt.2014.09.012.

2 George E. Andrews, Richard Askey, and Ranjan Roy. *Special Functions*, volume 71 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1999. doi:10.1017/CBO9781107325937.

3 Alan Baker. *Transcendental Number Theory*. Cambridge Mathematical Library. Cambridge University Press, 1975. doi:10.1017/CBO9780511565977.

4 Jonathan M. Borwein, David H. Bailey, and Roland Girgensohn. *Experimentation in Mathematics*. A K Peters/CRC Press, April 2004. doi:10.1201/9781439864197.

5 Marc Chamberland and Armin Straub. On gamma quotients and infinite products. *Advances in Applied Mathematics*, 51(5):546–562, 2013. doi:10.1016/j.aam.2013.07.003.

6 Karl Dilcher, Rob Noble, and Chris Smyth. Minimal polynomials of algebraic numbers with rational parameters. *Acta Arithmetica*, 148(3):281–308, 2011. doi:10.4064/aa148-3-5.

7 Karl Dilcher and Christophe Vignat. Infinite products involving Dirichlet characters and cyclotomic polynomials. *Advances in Applied Mathematics*, 100:43–70, September 2018. doi:10.1016/j.aam.2018.05.003.

8 P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.

9 Vesa Halava, Tero Harju, and Mika Hirvensalo. Positivity of second order linear recurrent sequences. *Discrete Applied Mathematics*, 154(3):447–451, 2006. doi:10.1016/j.dam.2005.10.009.

10 Alaa Ibrahim and Bruno Salvy. Positivity certificates for linear recurrences. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 982–994, 2024. doi:10.1137/1.9781611977912.37.

11 Manuel Kauers and Peter Paule. *The Concrete Tetrahedron*. Springer Vienna, 2011. doi:10.1007/978-3-7091-0445-3.

12 Manuel Kauers and Veronika Pillwein. When can we detect that a p-finite sequence is positive? In *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, ISSAC '10, pages 195–201, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1837934.1837974.

13 George Kenison, Oleksiy Klurman, Engel Lefaucheux, Florian Luca, Pieter Moree, Joël Ouaknine, Whiteland Markus, and James Worrell. On Inequality Decision Problems for Low-Order Holonomic Sequences. preprint.

14 George Kenison, Joris Nieuwveld, Joël Ouaknine, and James Worrell. Positivity problems for reversible linear recurrence sequences. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 130:1–130:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.130.

15 George Kenison, Klara Nosan, Mahsa Shirmohammadi, and James Worrell. The membership problem for hypergeometric sequences with quadratic parameters. In *Proceedings of the 2023 International Symposium on Symbolic and Algebraic Computation*, ISSAC '23, pages 407–416. Association for Computing Machinery, 2023. doi:10.1145/3597066.3597121.

16 S. Lang. *Introduction to transcendental numbers*. Addison-Wesley series in mathematics. Addison-Wesley Pub. Co., 1966.

17 Angus Macintyre and A. J. Wilkie. On the decidability of the real exponential field. In *Kreiseliana*, pages 441–467. A K Peters, Wellesley, MA, 1996.

18 Greg Martin. A product of Gamma function values at fractions with the same denominator, December 2009. arXiv:0907.4384.

19 L. J. Mordell. On the linear independence of algebraic numbers. *Pacific J. Math.*, 3:625–630, 1953.

20 Yu V Nesterenko. Modular functions and transcendence questions. *Sbornik: Mathematics*, 187(9):1319–1348, October 1996. doi:10.1070/sm1996v187n09abeh000158.

21 Albert Nijenhuis. Short gamma products with simple values. *The American Mathematical Monthly*, 117(8):733, 2010. doi:10.4169/000298910x515802.

22 Klara Nosan, Amaury Pouly, Mahsa Shirmohammadi, and James Worrell. The membership problem for hypergeometric sequences with rational parameters. In *Proceedings of the 2022 International Symposium on Symbolic and Algebraic Computation*, ISSAC '22, pages 381–389. Association for Computing Machinery, 2022. doi:10.1145/3476446.3535504.

23 Joël Ouaknine and James Worrell. On the positivity problem for simple linear recurrence sequences,. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 318–329. Springer, 2014. doi:10.1007/978-3-662-43951-7_27.

24 Joël Ouaknine and James Worrell. Ultimate positivity is decidable for simple linear recurrence sequences. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 330–341. Springer, 2014. doi:10.1007/978-3-662-43951-7_28.

25 Joël Ouaknine and James Worrell. On linear recurrence sequences and loop termination. *ACM SIGLOG News*, 2(2):4–13, 2015. doi:10.1145/2766189.2766191.

26 Joël Ouaknine and James Worrell. *Positivity Problems for Low-Order Linear Recurrence Sequences*, pages 366–379. Society for Industrial and Applied Mathematics, 2014. doi:10.1137/1.9781611973402.27.

27 William Paulsen. Gamma triads. *The Ramanujan Journal*, 50(1):123–133, October 2019. doi:10.1007/s11139-018-0114-8.

28 Veronika Pillwein and Miriam Schussler. An efficient procedure deciding positivity for a class of holonomic functions. *ACM Commun. Comput. Algebra*, 49(3):90–93, November 2015. doi:10.1145/2850449.2850458.

29 A.P. Prudnikov, Y.A. Brychkov, and O.I. Marichev. *Integrals and Series*, volume 1: Elementary Functions. Gordon & Breach, 1986.

30 Paulo Ribenboim. *My numbers, my friends*. Springer-Verlag, New York, 2000. Popular lectures on number theory.

31 Daniel Richardson. The elementary constant problem. In *Papers from the International Symposium on Symbolic and Algebraic Computation*, ISSAC '92, pages 108–116, New York, NY, USA, 1992. Association for Computing Machinery. doi:10.1145/143242.143284.

32 Daniel Richardson. How to recognize zero. *Journal of Symbolic Computation*, 24(6):627–645, 1997. doi:10.1006/jsco.1997.0157.

33 I. Stewart and D. Tall. *Algebraic number theory and Fermat's last theorem*. CRC Press, Boca Raton, FL, fourth edition, 2016.

34 J. Sándor and L. Tóth. A remark on the gamma function. *Elemente der Mathematik*, 44(3):73–76, 1989.

35 Raimundas Vidunas. Expressions for values of the gamma function. *Kyushu Journal of Mathematics*, 59(2):267–283, 2005. doi:10.2206/kyushujm.59.267.

36 Michel Waldschmidt. Transcendence of periods: the state of the art. *Pure Appl. Math. Q.*, 2(2, Special Issue: In honor of John H. Coates. Part 2):435–463, 2006. doi:10.4310/PAMQ.2006.v2.n2.a3.

37 E. T. Whittaker and G. N. Watson. *A Course of Modern Analysis*. Cambridge University Press, September 1996. doi:10.1017/CBO9780511608759.

38 Kenneth S. Williams. Integers of Biquadratic Fields. *Canadian Mathematical Bulletin*, 13(4):519–526, 1970. doi:10.4153/CMB-1970-094-8.

A Appendixed Proofs

▶ **Lemma 4.** *Consider the class of hypergeometric sequences $\langle u_n \rangle_n$ whose shift quotients $r(n)$ either diverge to $\pm\infty$ or converge to a limit ℓ with $|\ell| \neq 1$. For this class, the Membership and Threshold Problems are both decidable.*

Proof. When $r(n) \equiv 0$ decidability of both problems is trivial. So we assume that this is not the case for the remainder of the proof. Write the shift quotient $r(n) = c\frac{q(n)}{p(n)}$ where $p, q \in \mathbb{Q}[x]$ are monic polynomials and $c \in \mathbb{Q}$. Without loss of generality, we can assume that $c > 0$; for otherwise, sequence $\langle u_n \rangle_{n=0}^{\infty}$ is given by the interlacing of two hypergeometric sequences with this property. Let us assume that $r(n)$ diverges to $+\infty$. In this case it is easily seen that, for each $t \in \mathbb{Q}$, there exists a computable $N_0 \in \mathbb{N}$ such that if $n \geq N_0$ then $|u_n| = |u_0 \cdot \prod_{k=0}^{n} r(k)| > |t|$. Thus to determine the Threshold Problem in this instance, we need only determine the ultimate sign of $\langle u_n \rangle_{n=0}^{\infty}$ (which is straightforward). Moreover, we can compute a bound N_1 after which the sign of $\langle u_n \rangle_{n=0}^{\infty}$ is constant. Thus the Threshold Problem in such instances reduces to an exhaustive search that asks whether $u_n \geq t$ for each $n \in \{0, 1, \ldots, \max\{N_0, N_1\}\}$. Mutatis mutandis, decidability is similarly established for instances of the Threshold Problem where $r(n)$ converges to a limit ℓ with $|\ell| \neq 1$.

The argument for the Membership Problem is similar and given in full in [22]. ◀

▶ **Proposition 6.** *Let $\langle u_n \rangle_n$ be a hypergeometric sequence whose shift quotient is given by a ratio of two polynomials with real coefficients. For such sequences, the Membership and Threshold Problems are both Turing-reducible to the following decision problem. Given $d \in \mathbb{N}$, $\alpha_1, \ldots, \alpha_d \in \mathbb{C} \setminus \mathbb{Z}_{<0}$ (the roots of some $P(x) \in \mathbb{R}[x]$), and $\beta_1, \ldots, \beta_d \in \mathbb{C} \setminus \mathbb{Z}_{<0}$ (the roots of some $Q(x) \in \mathbb{R}[x]$), determine whether*

$$\frac{\Gamma(\beta_1) \cdots \Gamma(\beta_d)}{\Gamma(\alpha_1) \cdots \Gamma(\alpha_d)} = t$$

for $t \in \mathbb{Q} \setminus 0$.

Proof. From Lemma 4, we need only consider cases where the associated shift quotient $r(k)$ converges to ± 1 and, by Theorem 5, we can assume without loss of generality that $r(k)$ is harmonious. We treat the case that the sequence of partial products $\langle \prod_{k=0}^{n} r(k) \rangle_n$ is eventually strictly increasing. The case where the sequence of partial products is eventually strictly decreasing follows *mutatis mutandis*.

Consider an instance $(\langle u_n\rangle_n, t)$ of the Threshold Problem with $r(k)$ as above and recall our running assumption that $u_0 = 1$. Let $\tau := \prod_{k=0}^{\infty} r(k)$. We assume that the sequence $\langle u_n\rangle_n$, whose terms $u_n = \prod_{k=0}^{n} r(k)$ are given by partial products, is eventually strictly increasing. Then there exists a computable $N \in \mathbb{N}$ such that $u_n < \tau$ for each $n \geq N$. There are two subcases to consider. First, if $\tau \leq t$ then it is clear that $u_n < t$ for each $n \geq N$ and so we return the answer `no` to the Threshold Problem. Second, if $\tau > t$ then there exists an $N_1 \in \mathbb{N}$ such that $u_n > t$ for each $n \geq N_1$. So decidability of Threshold in this instance reduces to an exhaustive check that asks whether $u_n \geq t$ for each $n \in \{0, 1, \ldots, N-1\}$.

All that remains is to decide whether $\tau \leq t$. It is clear that, by computing τ to sufficient precision, the problem of determining whether $\tau < t$ or $\tau > t$ is recursively enumerable. Thus we need only test whether the equality $\tau = t$ holds. By Theorem 5, we know that $\tau = \prod_{j=1}^{m} \Gamma(\beta_j)/\Gamma(\alpha_j)$, from which we deduce the desired result.

For the sake of brevity, we omit the argument for the reduction from the Membership Problem, which is near identical to the reasoning displayed above. ◀

▶ **Proposition 7.** *For hypergeometric sequences, decidability of the Membership Problem Turing-reduces to that of the Threshold Problem.*

Proof. By Theorem 5 and Proposition 6, we need only consider hypergeometric sequences with harmonious shift quotients. Thus, we continue under this assumption.

Let $(\langle u_n\rangle_n, t)$ be an instance of the Membership Problem as above and, in addition, assume that $\langle u_n\rangle_n$ is eventually decreasing. We note there is a computable bound $N_0 \in \mathbb{N}$ such that $u_{n+1} \leq u_n$ for all $n \geq N_0$. Now let τ be the limit of the the sequence $\langle u_n\rangle_n$. Either we have that $\tau = t$ or $\tau \neq t$. We note that the Membership Problem is decidable when $\tau \neq t$ and so it remains to test cases when $\tau = t$.

- Suppose that an oracle for the Threshold Problem returns the answer `yes` to the problem instance $(\langle u_n\rangle_{n=N_0}^{\infty}, \tau)$. Since $u_{n+1} < u_n$ for all $n \geq N_0$, we deduce that τ is not a member of the sequence $\langle u_n\rangle_{n=N_0}^{\infty}$. Thus all that remains is to test whether $\tau \in \{u_0, u_1, \ldots, u_{N_0-1}\}$.
- Suppose that an oracle for the Threshold Problem returns the answer `no` to the problem instance $(\langle u_n\rangle_{n=N_0}^{\infty}, \tau)$. Then there is a computable bound N_1 such that $u_n < \tau$ for all $n \geq N_1$. Thus we can decide the Membership Problem by testing whether $\tau \in \{u_0, u_1, \ldots, u_{N_1-1}\}$.

We note a similar argument to that given above holds for testing the Membership Problem for hypergeometric sequences that are eventually increasing. Thus we deduce the desired result: that decidability of the Membership Problem for hypergeometric sequences Turing-reduces to that of the Threshold Problem for hypergeometric sequences. ◀

B Threshold for the Kepler–Bouwkamp Constant Approximation

In this section we demonstrate that we can conditionally determine the Threshold Problem for instances $(\langle v_n\rangle_{n=3}^{\infty}, t)$ where sequence $\langle v_n\rangle_{n=3}^{\infty}$ is the recurrence sequence associated with the approximation of the Kepler–Bouwkamp Constant (Example 1) by the $[2,2]$-Padé approximant of $\cos(\,\cdot\,)$.

Let us begin. Recall that the sequence $\langle v_n\rangle_{n=3}^{\infty}$ has terms given by $v_n := \prod_{k=3}^{n} \frac{12k^2-5\pi^2}{12k^2+\pi^2}$. By Proposition 6, decidability of problem instance $(\langle v_n\rangle_{n=3}^{\infty}, t)$ reduces to determining whether

$$\prod_{k=3}^{\infty} \frac{12k^2-5\pi^2}{12k^2+\pi^2} = \frac{\Gamma(3-\frac{\mathrm{i}}{6}\sqrt{3}\pi)\Gamma(3+\frac{\mathrm{i}}{6}\sqrt{3}\pi)}{\Gamma(3-\frac{1}{6}\sqrt{15}\pi)\Gamma(3+\frac{1}{6}\sqrt{15}\pi)} \tag{8}$$

is equal to t. (We note that the formulation as a gamma product is given to us by Theorem 5.)

We consider the numerator and denominator in turn. First, we use the translation and reflection properties of the gamma function to write the numerator of (8) as

$$\begin{aligned}\Gamma\left(3-\frac{\sqrt{3}\pi\mathrm{i}}{6}\right)\Gamma\left(3+\frac{\sqrt{3}\pi\mathrm{i}}{6}\right) &= \Gamma\left(-\frac{\sqrt{3}\pi\mathrm{i}}{6}\right)\Gamma\left(\frac{\sqrt{3}\pi\mathrm{i}}{6}\right)\prod_{k=0}^{2}\left(k^2+\frac{\pi^2}{12}\right)\\ &= \frac{4\sqrt{3}}{\mathrm{e}^{\frac{\pi^2}{2\sqrt{3}}}-\mathrm{e}^{-\frac{\pi^2}{2\sqrt{3}}}}\prod_{k=0}^{2}\left(k^2+\frac{\pi^2}{12}\right)\\ &= \frac{4\sqrt{3}\mathrm{e}^{\frac{\pi^2}{2\sqrt{3}}}}{\mathrm{e}^{\frac{\pi^2}{\sqrt{3}}}-1}\prod_{k=0}^{2}\left(k^2+\frac{\pi^2}{12}\right). \qquad (9)\end{aligned}$$

Second, we likewise write the denominator of (8) as

$$\begin{aligned}\Gamma\left(3-\frac{\sqrt{15}\pi}{6}\right)\Gamma\left(3+\frac{\sqrt{15}\pi}{6}\right) &= \Gamma\left(-\frac{\sqrt{15}\pi}{6}\right)\Gamma\left(\frac{\sqrt{15}\pi}{6}\right)\prod_{k=0}^{2}\left(k^2-\frac{5\pi^2}{12}\right)\\ &= -\frac{12\mathrm{i}}{\sqrt{15}\left(\mathrm{e}^{\frac{\sqrt{15}\pi^2\mathrm{i}}{6}}-\mathrm{e}^{-\frac{\sqrt{15}\pi^2\mathrm{i}}{6}}\right)}\prod_{k=0}^{2}\left(k^2-\frac{5\pi^2}{12}\right)\\ &= -\frac{12\mathrm{i}\mathrm{e}^{\frac{\pi^2\sqrt{15}\mathrm{i}}{6}}}{\sqrt{15}\left(\mathrm{e}^{\frac{\pi^2\sqrt{15}\mathrm{i}}{3}}-1\right)}\prod_{k=0}^{2}\left(k^2-\frac{5\pi^2}{12}\right). \qquad (10)\end{aligned}$$

Let us consider the decision problem at hand. Taken together, we are tasked to determine whether the ratio of (9) and (10) is equal to t. For this equality to hold, a simple rearrangement argument leads us to the following: there is a non-trivial polynomial $\mathcal{Q} \in \mathbb{Q}[x,y,z] = 0$ such that $\mathcal{Q}\big(\pi^2, \mathrm{e}^{\frac{\pi^2}{\sqrt{3}}}, \mathrm{e}^{\frac{\sqrt{15}\pi^2\mathrm{i}}{3}}\big) = 0$ (the fact that such a polynomial with rational coefficients exists is guaranteed by Lemma 9). However, we claim that no such polynomial $\mathcal{Q}$ exists if Schanuel's conjecture is true.

Let us prove the above claim. Consider the set

$$\mathfrak{S} := \left\{\frac{\pi^2}{\sqrt{3}}, \frac{\sqrt{15}\pi^2\mathrm{i}}{3}, \pi\mathrm{i}, \mathrm{e}^{\frac{\pi^2}{\sqrt{3}}}, \mathrm{e}^{\frac{\sqrt{15}\pi^2\mathrm{i}}{3}}, \mathrm{e}^{\pi\mathrm{i}}\right\}.$$

We note that the elements of $\left\{\frac{\pi^2}{\sqrt{3}}, \frac{\sqrt{15}\pi^2\mathrm{i}}{3}, \pi\mathrm{i}\right\}$ are $\mathbb{Q}$-linearly independent. Thus, Schanuel's conjecture predicts that there is a subset of $\mathfrak{S}$ of size at least 3 whose elements are algebraically independent. Since $\frac{\pi^2}{\sqrt{3}}$ and $\frac{\sqrt{15}\pi^2\mathrm{i}}{3}$ are algebraically dependent and $\mathrm{e}^{\pi\mathrm{i}} = -1$, we deduce that, subject to the truth of Schanuel's conjecture, the elements of $\left\{\pi^2, \mathrm{e}^{\frac{\pi^2}{\sqrt{3}}}, \mathrm{e}^{\frac{\sqrt{15}\pi^2\mathrm{i}}{3}}\right\}$ are algebraically independent. From the preceding work, we deduce that there is no non-trivial polynomial $\mathcal{Q}$ for which $\mathcal{Q}\big(\pi^2, \mathrm{e}^{\frac{\pi^2}{\sqrt{3}}}, \mathrm{e}^{\frac{\sqrt{15}\pi^2\mathrm{i}}{3}}\big) = 0$. It follows that the desired equality cannot hold. Thus we can conditionally decide the Threshold Problem for instances $(\langle v_n\rangle_{n=0}^{\infty}, t)$ with $t \in \mathbb{Q}$.

Solving Promise Equations over Monoids and Groups

Alberto Larrauri
Department of Computer Science, University of Oxford, UK

Stanislav Živný
Department of Computer Science, University of Oxford, UK

Abstract

We give a complete complexity classification for the problem of finding a solution to a given system of equations over a fixed finite monoid, given that a solution over a more restricted monoid exists. As a corollary, we obtain a complexity classification for the same problem over groups.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Constraint and logic programming

Keywords and phrases constraint satisfaction, promise constraint satisfaction, equations, minions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.146

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2402.08434` [40]

Funding This research was funded in whole by UKRI EP/X024431/1. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission. All data is provided in full in the results section of this paper.

1 Introduction

Constraint satisfaction problems (CSPs) form a large class of fundamental computational problems studied in artificial intelligence, database theory, logic, graph theory, and computational complexity. Since CSPs (with infinite domains) capture, up to polynomial-time Turing reductions, *all* computational problems [11], some restrictions need to be imposed on CSPs in order to have a chance to obtain complexity classifications. One line of work, pioneered in the database theory [36], restricts the interactions of the constraints in the instance [30, 41].

Another line of work, pioneered in [34, 26], restricts the types of relations used in the instance; these CSPs are known as nonuniform CSPs, or as having a fixed template/constraint language. Such CSPs with infinite domains capture graph acyclicity, systems of linear equations over the rationals, and many other problems [10]. Already fixed-template CSPs with finite domains form a large class of fundamental problems, including graph colourings [32], variants of the Boolean satisfiability problem, and, more generally, systems of equations over different types of finite algebraic structures. Even then, the class of finite-domain CSPs avoided a complete complexity classification for two decades despite a sustained effort.

In 2017, Bulatov [20] and, independently, Zhuk [47] classified all finite-domain CSPs as either solvable in polynomial time or NP-hard, thus answering in the affirmative the Feder-Vardi dichotomy conjecture [26]. In the effort to answer the Feder-Vardi conjecture, many complexity dichotomies were established in restricted fragments of CSPs. This included conservative CSPs [19], or equations over finite algebraic structures such as semigroups, groups, and monoids [29, 35]. In particular, while systems of equations[1] over Abelian groups are solvable in polynomial time, they are NP-hard over non-Abelian groups [29].

[1] Some papers use the term a *linear* equation.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 146; pp. 146:1–146:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

One of the recent research directions in constraint satisfaction that has attracted a lot of attention is the area of *promise* CSPs (PCSPs) [3, 13, 5]. The idea is that each constraint has two versions, a strong version and a weak version. Given an instance, one is promised that a solution satisfying all strict constraints exists and the goal is to find a solution satisfying all weak constraints, which may be an easier task. The prototypical example is the approximate graph colouring problem [28]: Given a 3-colourable graph, can one find a 6-colouring? The complexity of this problem is open (but believed to be NP-hard). Despite a flurry of papers on PCSPs, e.g., [27, 1, 4, 17, 43, 44, 6, 2, 21, 14, 15, 24, 22], the PCSP complexity landscape is widely open and unexplored. It is not even clear whether a dichotomy should be expected. Even the case of Boolean PCSPs remain open, the state-of-the-art being a dichotomy for Boolean *symmetric* PCSPs [27]. This should be compared with Boolean (non-promise) CSPs, which were classified by Schaefer in 1978 [45]. Schaefer's tractable cases include the classic and well-known examples of CSPs: equations and graph colouring. Both have been studied on non-Boolean domains and their complexity is well understood. However, the complexity of the promise variant of these fundamental problems is open. The first problem, graph colouring, leads to the already mentioned approximate graph colouring problem, which is a notorious open problem, despite recent progress [5, 38]. In this paper, we look at the second problem, and study PCSPs capturing systems of equations.

Contributions

The precise statements of all our main results are presented in Section 3.

As our most important contribution, in Section 5 we establish a complexity dichotomy for PCSPs capturing *promise* systems of equations over finite monoids, and over finite groups as a special case. Perhaps unsurprisingly, the tractability boundary is linked to the notion of Abelianness, just like in the non-promise setting, but the result is non-trivial and requires some care. Our main tool is the "the algebraic approach to PCSPs" [5]. The influential paper [5] identified *minions* as an important concept. Minions generalise the notion of "a family of functions that is closed under permuting arguments, identifying arguments, and adding dummy arguments". A crucial example is the polymorphism minion of a PCSP template. Polymorphisms can be seen as high-dimensional symmetries of a PCSP template and capture the complexity of the underlying computational problem [13, 5]. Following the algebraic approach [5], hardness of a PCSP is established by showing that the associated polymorphism minion is, in some sense, limited. Conversely, if this minion is rich enough then the PCSP can be shown to be solvable via some efficient algorithm [5, 16, 22, 15].

To prove our main result, we study a class of minions that arise naturally from monoids, which we call monoidal minions. In Section 4 we show a complexity dichotomy for PCSPs whose polymorphism minions are homomorphically equivalent to some monoidal minion. This is our second contribution, which may be of independent interest. In particular, the concept of monoidal minions captures studied minions, cf. Remark 14 in Section 3.

All our tractability results use solvability via the BLP + AIP algorithm [16]. In fact, tractable PCSPs corresponding to promise systems of equations over monoids are finitely tractable in the sense of [13, 1]. In the special case of promise systems of equations over groups, the affine integer programming (AIP) algorithm [13, 5] suffices, rather than BLP + AIP. However, AIP is provably not enough to solve promise equations over general monoids.

As our final contribution, in Section 6 we show that our dichotomy for systems of equations over monoids cannot be easily extended to semigroups, as this would imply a dichotomy for all PCSPs. We do so by showing that every PCSP is polynomial-time equivalent to a PCSP capturing systems of equations over semigroups, a phenomenon observed for CSPs in [35].

Related work

PCSPs are a qualitative approximation of CSPs; the goal is still to satisfy all constraints, but in a weaker form. A recent related line of work includes the series [7, 8, 9]. A traditional approach to approximation is quantitative: maximising the number of satisfied constraints. Regarding approximation of equations, Håstad showed that, for any Abelian group G and any $\varepsilon > 0$, it is NP-hard to find a solution satisfying $1/|G| + \varepsilon$ constraints [31] even if $1 - \varepsilon$ constraints can be satisfied. Hence, the random assignment, which satisfies $1/|G|$ constraints, is optimal! Håstad's result has been extended to non-Abelian groups in [25, 7]. Systems of equations have been studied, e.g., over semigroups in [46], over monoids and semigroups in [35], and over arbitrary finite algebras in [39, 37, 12, 42].

The full version of this paper [40] contains all details and proofs.

2 Preliminaries

We denote by $[k]$ the set $\{1, 2, \dots, k\}$. We write id_X for the identity map on a set X. We use the lowercase boldface font for tuples; e.g., we write $\boldsymbol{b}$ for a tuple $(b_1, \dots, b_n)$. We say that a function f *extends* another function g if $\mathrm{dom}(g) \subseteq \mathrm{dom}(f)$, and $f|_{\mathrm{dom}(g)} = g$.

Algebraic structures

A *semigroup* S is a set equipped with an associative binary operation, for which we use multiplicative notation. Two elements $a, b \in S$ commute if $ab = ba$. An *Abelian* semigroup is a semigroup in which every two elements commute. A *semigroup homomorphism* from a semigroup S_1 to a semigroup S_2 is a map $\varphi : S_1 \to S_2$ satisfying $\varphi(s \cdot_{S_1} t) = \varphi(s) \cdot_{S_2} \varphi(t)$.[2] Given two elements $s, t \in S$ we write $s \sqsubseteq t$ if s can be expressed as a product of elements in S including t. Note that $\sqsubseteq$ constitutes a preorder over any semigroup. We define the equivalence relation $\sim$ by $s \sim t$ whenever $s \sqsubseteq t$ and $t \sqsubseteq s$.

A *monoid* is a semigroup containing an identity element for its binary operation, denoted by e. A *monoid homomorphism* from a monoid M_1 to a monoid M_2 is a map $\varphi : M_1 \to M_2$ satisfying $\varphi(x \cdot_{M_1} y) = \varphi(x) \cdot_{M_2} \varphi(y)$ and $\varphi(e_{M_1}) = e_{M_2}$. We say that φ is *Abelian* if its image $\mathrm{Im}(\varphi)$ is an Abelian monoid. A *group* is a monoid in which each element has an inverse. A *group homomorphism* from a group G_1 to a group G_2 is a map $\varphi : G_1 \to G_2$ satisfying $\varphi(x \cdot_{G_1} y) = \varphi(x) \cdot_{G_2} \varphi(y)$ (which implies that also the inverses and the identity element are preserved).

Given a semigroup S, a subset $G \subseteq S$ is called a *subgroup* if G equipped with S's binary operation is a group, meaning that there is a distinguished element $e_G \in G$ satisfying that (1) $e_G \cdot_M g = g \cdot_M e_G = g$ for each $g \in G$, and (2) for each element $g \in G$ there exists $h \in G$ satisfying $g \cdot_M h = h \cdot_M g = e_G$. We say that S is a *union of subgroups* if every element $s \in S$ belongs to a subgroup of S. We call an element s of S *regular* if $s^2 t = s$ and $ts = st$ for some t in S.[3] Intuitively, t acts as some type of inverse of s. It is known that s belongs to a subgroup of S if and only if s is regular [33, Theorem 2.2.5].

We use the standard product (and also the power) of a semigroup (monoid, group), where the operation is defined componentwise. We use the symbol $\preceq$ for a substructure; e.g., if S is a semigroup then we write $T \preceq S$ to indicate that T is a subsemigroup of S (and similarly for monoids and groups).

Unless stated explicitly otherwise, all semigroups, monoids, and groups in this paper are finite.

[2] I.e., the multiplication on the LHS is in S_1, whereas the multiplication on the RHS is in S_2.
[3] The usual definition of a regular element in a semigroup just requires that $sts = s$ for some t [33].

Relational structures

A *relational signature* σ consists of a finite set of relation symbols R, each with a finite arity $\mathrm{ar}(R) \in \mathbb{N}$. A *relational structure* $\boldsymbol{A}$ over the signature σ, or a σ-structure, consists of a finite set A and a relation $R^{\boldsymbol{A}} \subseteq A^k$ of arity $k = \mathrm{ar}(R)$ for every $R \in \sigma$. Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be two σ-structures. A map $h : A \to B$ is called a *homomorphism* from $\boldsymbol{A}$ to $\boldsymbol{B}$ if h preserves all relations in $\boldsymbol{A}$; i.e., if, for every $R \in \sigma$, $h(\mathbf{x}) \in R^{\boldsymbol{B}}$ whenever $\mathbf{x} \in R^{\boldsymbol{A}}$, where h is applied componentwise. We denote the existence of a homomorphism from $\boldsymbol{A}$ to $\boldsymbol{B}$ by writing $\boldsymbol{A} \to \boldsymbol{B}$. A *template* is a pair $(\boldsymbol{A}, \boldsymbol{B})$ of relational structures such that $\boldsymbol{A} \to \boldsymbol{B}$.

A k-ary *polymorphism* of a template $(\boldsymbol{A}, \boldsymbol{B})$ over signature σ is a map $p : A^k \to B$ that preserves all relations $R^{\boldsymbol{A}}$ from $\boldsymbol{A}$ in the following sense: For any $\mathrm{ar}(R) \times k$ matrix whose columns belong to $R^{\boldsymbol{A}}$, applying p row-wise results in a tuple that belongs to $R^{\boldsymbol{B}}$. We denote by $\mathrm{Pol}(\boldsymbol{A}, \boldsymbol{B})$ the set of all polymorphisms of $(\boldsymbol{A}, \boldsymbol{B})$.[4]

Minions

A *minion* $\mathcal{M}$ consists of a set $\mathcal{M}(n)$ for each positive number n, and a map $\pi^{\mathcal{M}} : \mathcal{M}(n) \to \mathcal{M}(m)$ for each map $\pi : n \to m$ satisfying (1) $\mathrm{id}_{[n]}^{\mathcal{M}} = \mathrm{id}_{\mathcal{M}(n)}$ for every $n > 0$, and (2) $\pi^{\mathcal{M}} \circ \tau^{\mathcal{M}} = (\pi \circ \tau)^{\mathcal{M}}$ for every pair of suitable maps π, τ. When the minion is clear from the context, we write $p^{(\pi)}$ for $\pi^{\mathcal{M}}(p)$. Elements $p \in \mathcal{M}(n)$ are called n-ary, and whenever $p^{(\pi)} = q$ we say that q is a *minor* of p. A *minion homomorphism* $\xi : \mathcal{M} \to \mathcal{N}$ is a map from a minion $\mathcal{M}$ to another minion $\mathcal{N}$ that preserves arities and minor operations. I.e., $\xi(p^{(\pi)}) = (\xi(p))^{(\pi)}$ for every minor $p^{(\pi)}$.

Given a template $(\boldsymbol{A}, \boldsymbol{B})$, its set of polymorphisms $\mathrm{Pol}(\boldsymbol{A}, \boldsymbol{B})$ can be equipped with a minion structure in a natural way. That is, n-ary elements of $\mathrm{Pol}(\boldsymbol{A}, \boldsymbol{B})$ are just n-ary polymorphisms $p : A^n \to B$. Additionally, given some n-ary polymorphism p, and some map $\pi : [n] \to [m]$, the minor $p^{(\pi)}$ is the polymorphism $q : A^m \to B$ given by $(a_1, \ldots, a_m) \mapsto p(b_1, \ldots, b_n)$, where $b_i = a_{\pi(i)}$ for each $i \in [n]$.

Given a minion $\mathcal{M}$, we define two special types of elements. An element $p \in \mathcal{M}(2k+1)$ is called *alternating* if $p^{(\pi)} = p$ for any permutation $\pi : [2m+1] \to [2m+1]$ that preserves parity, and $p^{(\pi_1)} = p^{(\pi_2)}$, where for each $i = 1, 2$ the map π_i is given by $1 \mapsto i$, $2 \mapsto i$ and $j \mapsto j$ for all $j > 2$. An element $p \in \mathcal{M}(2k+1)$ is called 2-*block-symmetric* if the set $[2k+1]$ can be partitioned into two blocks of size $k+1$ and k in such a way that $p^{(\pi)} = p$ for any map $\pi : [2m+1] \to [2m+1]$ that preserves each block.

Constraint satisfaction

Let $(\boldsymbol{A}, \boldsymbol{B})$ be a template with common signature σ. The *promise constraint satisfaction problem* (PCSP) with template $(\boldsymbol{A}, \boldsymbol{B})$ is the following computational problem, denoted by $\mathrm{PCSP}(\boldsymbol{A}, \boldsymbol{B})$. Given a σ-structure $\boldsymbol{X}$, output YES if $\boldsymbol{X} \to \boldsymbol{A}$ and output NO if $\boldsymbol{X} \not\to \boldsymbol{B}$. This is the decision version. In the search version, one is given a σ-structure $\boldsymbol{X}$ with the promise that $\boldsymbol{X} \to \boldsymbol{A}$; the goal is to find a homomorphism from $\boldsymbol{X}$ to $\boldsymbol{B}$ (which necessarily exists, as $\boldsymbol{X} \to \boldsymbol{A}$ and $\boldsymbol{A} \to \boldsymbol{B}$, and homomorphisms compose). It is known that the decision version polynomial-time reduces to the search version (but it is not known whether the two variants are polynomial-time equivalent) [5]. In our results, the positive (tractability) results are for the search version, whereas the hardness (intractability) results are for the decision version. We denote by $\mathrm{CSP}(\boldsymbol{A})$ the problem $\mathrm{PCSP}(\boldsymbol{A}, \boldsymbol{A})$; this is the standard (non-promise) constraint satisfaction problem (CSP). For CSPs, the decision version and the search version are polynomial-time equivalent [18].

[4] Equivalently, p is a polymorphism of $(\boldsymbol{A}, \boldsymbol{B})$ if p is a homomorphism from the k-th power of $\boldsymbol{A}$ to $\boldsymbol{B}$.

We need two existing algorithms for PCSPs, namely the AIP algorithm [5] and the strictly more powerful BLP + AIP algorithm [16]. Their power is captured by the following results.

▶ **Theorem 1** ([5]). *Let* $(\boldsymbol{A}, \boldsymbol{B})$ *be a template. Then* $\mathrm{PCSP}(\boldsymbol{A}, \boldsymbol{B})$ *is solved by AIP if and only if* $\mathrm{Pol}(\boldsymbol{A}, \boldsymbol{B})$ *contains alternating maps of all odd arities.*

▶ **Theorem 2** ([16]). *Let* $(\boldsymbol{A}, \boldsymbol{B})$ *be a template. Then* $\mathrm{PCSP}(\boldsymbol{A}, \boldsymbol{B})$ *is solved by BLP + AIP if and only if* $\mathrm{Pol}(\boldsymbol{A}, \boldsymbol{B})$ *contains 2-block-symmetric maps of all odd arities.*

3 Overview of Results

Promise equations over monoids and groups

Our first and main result is a dichotomy theorem for solving promise equations over finite monoids and thus also, as a special case, over finite groups. We first define equations in the standard, non-promise setting as it is useful for mentioning previous work and for our own proofs.

An *equation* over a semigroup S is an expression of the form $x_1 \dots x_n = y_1 \dots y_m$, where each x_i, y_i is either a variable or some element from S, referred to as a *constant*. A *system of equations* over S is just a set of equations. A solution to such a system is an assignment of elements of S to the variables of the system that makes all equations hold. Equations and systems of equations are defined similarly for monoids and groups. The only difference is that for groups we allow "inverted variables" x^{-1} in the equations, which are interpreted as inverses of the elements assigned to x.

In the context of CSPs, it is common to consider only restricted "types" of equations that can then express all other equations. The following definition captures systems of equations where each equation is either of the form $x_1 x_2 = x_3$, for three variables, or $x = c$, fixing a variable to a constant. It is well known that restricting to systems of equations of this kind is without loss of generality [40].

▶ **Definition 3.** *Let* S *be a semigroup and* $T \preceq S$ *a subsemigroup. The relational structure* $\mathrm{Eqn}(S, T)$ *has universe* S*, and the following relations:*

- *A ternary relation* $R_\times = \{(s_1, s_2, s_3) \in S^3 \mid s_1 s_2 = s_3\}$*, and*
- *a singleton unary relation* $R_t = \{t\}$ *for each* $t \in T$*.*

This template captures systems of equations of the kind described above when we allow only constants in a subsemigroup T of the ambient semigroup S. Similarly, we define the templates $\mathrm{Eqn}(M, N)$, $\mathrm{Eqn}(G, H)$ in the same way when M is a monoid and $N \preceq M$ a submonoid, and when G is a group and $H \preceq G$ is a subgroup. Observe that the definition of subgroup is more restrictive than the one of submonoid and this in turn is more restrictive than the notion of subsemigroup. We abuse the notation and write $\mathrm{Eqn}(S, T)$ for $\mathrm{CSP}(\mathrm{Eqn}(S, T))$.

Previous works focused on problems $\mathrm{Eqn}(G, G)$ and $\mathrm{Eqn}(M, M)$. Given a group G, it is known that $\mathrm{Eqn}(G, G)$ is solvable in polynomial time (by AIP) if G is Abelian, and NP-hard otherwise [29]. Similarly, when M is a monoid, $\mathrm{Eqn}(M, M)$ is solvable in polynomial time if M is Abelian and it is the union of its subgroups, and NP-hard otherwise [35].

We now define promise equations.

▶ **Definition 4.** *Let* S_1, S_2 *be semigroups, and let* φ *be a semigroup homomorphism with* $\mathrm{dom}(\varphi) \preceq S_1$ *and* $\mathrm{Im}(\varphi) \preceq S_2$*. The* promise system of equations over semigroups *problem* $\mathrm{PEqn}(S_1, S_2, \varphi)$ *is the* $\mathrm{PCSP}(\boldsymbol{A}, \boldsymbol{B})$*, where* $A = S_1$*,* $B = S_2$*, and the relations are defined as follows:*

- *A ternary relation* $R_\times^{\boldsymbol{A}} = \{(s_1, s_2, s_3) \in S_1^3 \mid s_1 s_2 = s_3\}$*, and* $R_\times^{\boldsymbol{B}} = \{(s_1, s_2, s_3) \in S_2^3 \mid s_1 s_2 = s_3\}$.
- *For each* $t \in \mathrm{dom}(\varphi)$*, a unary relation given by* $R_t^{\boldsymbol{A}} = \{t\}$*, and* $R_t^{\boldsymbol{B}} = \{\varphi(t)\}$.

For this template to be well defined there should be a homomorphism from $\boldsymbol{A}$ *to* $\boldsymbol{B}$*, which is equivalent to the existence of a semigroup homomorphism* $\psi : S_1 \to S_2$ *that extends* φ.

Analogously, we also define the *promise system of equations over monoids* problem and the *promise system of equations over groups* problem by replacing semigroup-related notions with monoid-related notions and group-related notions respectively. Observe that the problem $\mathrm{Eqn}(S, T)$ described before corresponds precisely to $\mathrm{PEqn}(S, S, \mathrm{id}_T)$.

We can now state our main result.

▶ **Theorem 5** (Main). $\mathrm{PEqn}(M_1, M_2, \varphi)$ *is solvable in polynomial time by BLP + AIP if there is an Abelian homomorphism* $\psi : M_1 \to M_2$ *extending* φ *and* $\mathrm{Im}(\psi)$ *is a union of subgroups. Otherwise,* $\mathrm{PEqn}(M_1, M_2, \varphi)$ *is* NP*-hard.*

For the special case of groups, we get a simpler tractability criterion and a simpler algorithm.

▶ **Corollary 6.** $\mathrm{PEqn}(G_1, G_2, \varphi)$ *is solvable in polynomial time via AIP if there is an Abelian homomorphism* $\psi : G_1 \to G_2$ *extending* φ*. Otherwise,* $\mathrm{PEqn}(G_1, G_2, \varphi)$ *is* NP*-hard.*

As easy corollaries, Theorem 5 applies in the special case of non-promise setting.

▶ **Corollary 7.** *Given two monoids* $N \preceq M$*,* $\mathrm{Eqn}(M, N)$ *is solvable in polynomial time by BLP + AIP if there is an Abelian endomorphism of* M *extending* id_N *whose image is a union of subgroups, and is* NP*-hard otherwise.*

▶ **Corollary 8.** *Given two groups* $H \preceq G$*,* $\mathrm{Eqn}(G, H)$ *is solvable in polynomial time by AIP if there is an Abelian endomorphism of* G *that extends* id_H*, and is* NP*-hard otherwise.*

▶ **Example 9.** Let G be the dihedral group on four elements, and H be the symmetric group on four elements. Observe that G can be seen as a subgroup of H in a natural way: H consists of all permutations on four elements, while G contains only those that are symmetries of the square. The group G is generated by the right 90-degree rotation r and an arbitrary reflection f that leaves no element fixed. We consider two group homomorphisms φ_1, φ_2 with $\mathrm{dom}(\varphi_i) \preceq G$ and $\mathrm{Im}(\varphi_i) \preceq H$. The domain of both homomorphism is the subgroup $\{e, r, r^2, r^3\} \preceq G$. Then, φ_1 is given by $r \mapsto r^2$, and φ_2 is given by $r \mapsto r$. The following hold:

- $\mathrm{PEqn}(G, H, \varphi_1)$ is tractable, and solvable via AIP. However both $\mathrm{Eqn}(G, \mathrm{dom}(\varphi_1))$ and $\mathrm{Eqn}(H, \mathrm{Im}(\varphi_1))$ are NP-hard.
- $\mathrm{PEqn}(G, H, \varphi_2)$ is NP-hard.

To see the first item, observe that the group homomorphism $\psi : G \to H$ given by $r \mapsto r^2$ and $f \mapsto f$ is Abelian (its image is isomorphic to the direct product $Z_2 \times Z_2$) and extends φ_1. Hardness of $\mathrm{Eqn}(G, \mathrm{dom}(\varphi_1))$ is a consequence of the fact that the commutator subgroup of G is $\{e, r, r^2, r^3\} \succeq \mathrm{dom}(\varphi_1)$, so $\mathrm{dom}(\varphi_1)$ is included in the kernel of any Abelian endomorphism of G. Similarly, hardness of $\mathrm{Eqn}(H, \mathrm{Im}(\varphi_1))$ follows from the fact that the commutator subgroup of H is the alternating group on four elements, and has $\mathrm{Im}(\varphi_1)$ as a subgroup.

The second item can be proved by observing that the only normal subgroup of G that does not intersect $\mathrm{dom}(\varphi_2)$ is the trivial subgroup, so any homomorphism $\psi : G \to H$ that extends φ_2 needs to be injective, and thus non-Abelian.

We say that $\mathrm{PCSP}(\boldsymbol{A}, \boldsymbol{B})$ is *finitely tractable* if there is $\boldsymbol{C}$ such that $\boldsymbol{A} \to \boldsymbol{C} \to \boldsymbol{B}$ and $\mathrm{CSP}(\boldsymbol{C})$ is solvable in polynomial time. The tractable cases in Theorem 5 are in fact finitely tractable; for details, cf. [40].

The power of BLP + AIP is necessary in Theorem 5 in the sense that AIP does not suffice for all monoids, even for (non-promise) CSPs, unlike in the case of groups. Indeed, adding a fresh element to a group that serves as the monoid identity fools AIP; for details, cf. [40].

Promise equations over semigroups

As our next result, we prove that every PCSP is polynomial-time equivalent to a problem of the form $\mathrm{PEqn}(S_1, S_2, \varphi)$ over some semigroups S_1, S_2. Hence, extending our classification of promise equations beyond monoids is difficult in the sense that understanding the computational complexity of promise equations over semigroups is as hard as classifying all PCSPs. This result is analogous to the one known in the non-promise setting obtained in [35], whose proof we closely follow. One difficulty in lifting the result from [35] is the lack of constants in the promise setting. The details can be found in Section 6.

▶ **Theorem 10.** *Let $(\boldsymbol{A}, \boldsymbol{B})$ be a template. Then there are semigroups S_1, S_2 and a semigroup homomorphism φ with $\mathrm{dom}(\varphi) \preceq S_1$ and $\mathrm{Im}(\varphi) \preceq S_2$ such that $\mathrm{PCSP}(\boldsymbol{A}, \boldsymbol{B})$ is polynomial-time equivalent to $\mathrm{PEqn}(S_1, S_2, \varphi)$.*

Monoidal minions

As our third result, we investigate minions based on monoids. For PCSPs whose polymorphism minions are homomorphically equivalent to such minions, we establish a dichotomy. This is a building block in the proof of our main result, but may be interesting in its own right. In this direction, we show that for each monoidal minion $\mathcal{M}$, there are PCSP templates whose polymorphism minions are isomorphic to $\mathcal{M}$. For a finite set $[n]$, a tuple $(a_i)_{i\in[n]} \in M^n$ is called commutative if each pair of its elements commute.

▶ **Definition 11.** *Given an element $a \in M$ the* monoidal minion $\mathcal{M}_{M,a}$ *is the one where for each $n \in \mathbb{N}$ the elements $\boldsymbol{b} \in \mathcal{M}_{M,a}(n)$ are commutative tuples $\boldsymbol{b} \in M^n$ with $\prod_{i\in[n]} b_i = a$, and where for each $m > 0$ and each $\pi : [n] \to [m]$ the minor $\boldsymbol{b}^{(\pi)}$ is the tuple $\boldsymbol{c} \in M^m$ given by $c_j = \prod_{i\in\pi^{-1}(j)} b_i$, and the empty product equals the identity element e.*

▶ **Theorem 12.** *Let M be a finite monoid and let $a \in M$. Consider a template $(\boldsymbol{A}, \boldsymbol{B})$ with $\mathrm{Pol}(\boldsymbol{A}, \boldsymbol{B})$ homomorphically equivalent to $\mathcal{M}_{M,a}$. If a is regular in M then $\mathrm{PCSP}(\boldsymbol{A}, \boldsymbol{B})$ is solvable in polynomial time by BLP + AIP. Otherwise, $\mathrm{PCSP}(\boldsymbol{A}, \boldsymbol{B})$ is* NP*-complete.*

Next, we show that there are templates whose polymorphism minions are of the considered type (up to isomorphism).

▶ **Theorem 13.** *Let M be a monoid, and $a \in M$ an arbitrary element. Then the template $(\boldsymbol{A}, \boldsymbol{B})$ described below satisfies that $\mathrm{Pol}(\boldsymbol{A}, \boldsymbol{B}) \simeq \mathcal{M}_{M,a}$.[5] The signature σ of $\boldsymbol{A}$ and $\boldsymbol{B}$ contains three relation symbols: a ternary symbol R, and two unary ones C_0, C_1. We define $A = \{0, 1\}$, $R^{\boldsymbol{A}} = \{(1,0,0), (0,1,0), (0,0,1)\}$, $C_0^{\boldsymbol{A}} = \{0\}$ and $C_1^{\boldsymbol{A}} = \{1\}$. The universe B of $\boldsymbol{B}$ is $\mathcal{M}_{M,a}(2)$. We define $R^{\boldsymbol{B}}$ as the set of triples in $(\mathcal{M}_{M,a}(2))^3$ of the form $((c_1, c_2c_3), (c_2, c_1c_3), (c_3, c_1c_2))$, where $c_1, c_2, c_3 \in M$ commute pairwise, and $c_1c_2c_3 = a$. Finally, the unary relations $C_0^{\boldsymbol{B}}$ and $C_1^{\boldsymbol{B}}$ are the singleton sets containing the tuples (e, a) and (a, e) respectively.*[6]

[5] We use $\simeq$ to denote the isomorphism relation, i.e., the existence of a bijection between the minions that preserves arities and minor operations.

[6] The map $f : A \to B$ given by $0 \mapsto (e, a)$ and $1 \mapsto (a, e)$ is a homomorphism from $\boldsymbol{A}$ to $\boldsymbol{B}$. The structure $\boldsymbol{A}$ corresponds to the "1-in-3" template, where both constants are added, and $\boldsymbol{B}$ is the so-called "free structure" [5] of $\mathcal{M}_{M,a}$ generated by $\boldsymbol{A}$.

Finally, we remark that monoidal minions are natural objects of study, as they include other relevant previously studied minions.

▶ Remark 14. Consider the Abelian monoid $M = \{0, 1, \epsilon\}$, whose multiplicative identity is 0, and where $1 \cdot 1 = 1 \cdot \epsilon = \epsilon \cdot \epsilon = \epsilon$. The elements of $\mathcal{M}_{M,1}$ are tuples with all zero entries except for a single 1 entry. Hence $\mathcal{M}_{M,1}$, corresponds to the so-called trivial minion $\mathcal{T}$ consisting of all projections (also known as dictators) on a two-element set. This minion represents the hardness boundary for CSPs, in the sense that a CSP is NP-hard if and only if its polymorphism minion maps homomorphically to $\mathcal{T}$ [18, 47].

Another example of a monoidal minion is the one capturing the power of arc consistency from [24]. In fact, every *linear* minion (in the sense of [23]) is a union of monoidal minions.[7]

If we allow infinite monoids to be considered, then monoidal minions include important minions that capture solvability via relevant algorithms. Consider the monoid $M = \{(r, z) \in \mathbb{Q} \times \mathbb{Z} \mid r \in [0, 1], \text{ and } r = 0 \text{ implies } z = 0\}$, where the binary operation is given by coordinate-wise addition, and the identity is $(0, 0)$. Then $\mathcal{M}_{M,(1,1)}$ is precisely the minion $\mathcal{M}_{\mathrm{BLP+AIP}}$ described in [16], which expresses the power of BLP + AIP. Similarly, the minions described in [5] to capture the power of BLP and AIP are monoidal minions as well.

4 Monoidal Minions: Proof of Theorem 12

Tractability. We use the characterisation of the power of BLP + AIP from Theorem 2 for the tractability part of Theorem 12. Observe that if there is a minion homomorphism $\xi : \mathcal{M}_{M,a} \to \mathrm{Pol}(\boldsymbol{A}, \boldsymbol{B})$ and $p \in \mathcal{M}_{M,a}$ is a $(2k+1)$-ary 2-block-symmetric element, then so is $\xi(p)$. Hence, showing that $\mathcal{M}_{M,a}$ has 2-block-symmetric elements of all arities proves that $\mathrm{PCSP}(\boldsymbol{A}, \boldsymbol{B})$ is solvable in polynomial time via BLP + AIP. Let $b \in M$ witness that a is regular. For each $k > 0$ consider the $(2k+1)$-ary element of $\mathcal{M}_{M,a}$ consisting of $k+1$ consecutive a's followed by k consecutive b's. To see that this this is indeed an element of $\mathcal{M}_{M,a}$ we need to check that $a^{k+1}b^k = a$. This follows from the assumption that b witnesses that a is regular and using $a^{k+1}b^k = a(ba)^k$. This tuple is 2-block-symmetric, with the blocks corresponding to a and b (of sizes k and $k+1$, respectively).

Intractability. We prove the intractability part of Theorem 12 (as well as other hardness results later in this paper) using the following result.

▶ **Theorem 15** ([5]). *Let $\mathcal{M} = \mathrm{Pol}(\boldsymbol{A}, \boldsymbol{B})$, and let $K, L > 0$ be any fixed integers. Suppose that M satisfies the following condition:*

$\mathcal{M} = \bigcup_{\ell \in [L]} \mathcal{M}_\ell$, and for each $\ell \in [L]$ there is a map $p \mapsto \mathcal{I}_\ell(p)$ that sends each $p \in \mathcal{M}_\ell$ to a set of its coordinates $\mathcal{I}_\ell(p)$ of size at most K. Furthermore, suppose that for each $\ell \in [L]$ and for each minor $p^{(\pi)} = q$ where $p, q \in \mathcal{M}_\ell$ it holds that $\pi(\mathcal{I}_\ell(p)) \bigcap \mathcal{I}_\ell(q) \neq \varnothing$.

Then $\mathrm{PCSP}(\boldsymbol{A}, \boldsymbol{B})$ is NP-complete.

Given a template $(\boldsymbol{A}, \boldsymbol{B})$, if there is a minion homomorphism $\xi : \mathrm{Pol}(\boldsymbol{A}, \boldsymbol{B}) \to \mathcal{M}_{M,a}$ and $\mathcal{M}_{M,a}$ satisfies the condition in the previous theorem, so does $\mathrm{Pol}(\boldsymbol{A}, \boldsymbol{B})$. Indeed, if $\mathcal{M}_{M,a} = \bigcup_{\ell \in [L]} \mathcal{M}_\ell$, then we can write $\mathrm{Pol}(\boldsymbol{A}, \boldsymbol{B}) = \bigcup_{\ell \in [L]} \xi^{-1}(\mathcal{M}_\ell)$. Additionally, if the map $\mathcal{I}_\ell$ witnesses the condition in the theorem for $\mathcal{M}_\ell$, then the map $\mathcal{I}'_\ell$ given by $p \mapsto \mathcal{I}_\ell(\xi(p))$ witnesses the same condition for $\xi^{-1}(\mathcal{M}_\ell)$. Hence, we show the hardness part of Theorem 12 by proving that $\mathcal{M}_{M,a}$ satisfies the assumptions in Theorem 15 when $a \in M$ is not regular.

[7] We thank Lorenzo Ciardo for this observation.

For a monoid M, we define a refinement $\sqsubseteq^{\mathrm{A}}$ of the preorder $\sqsubseteq$ introduced in Section 2. In detail, we write $a \sqsubseteq^{\mathrm{A}} b$ for $a, b \in M$ if there is a third element $c \in M$ that commutes with b such that $bc = a$. We put $a \sim^{\mathrm{A}} b$ when both $a \sqsubseteq^{\mathrm{A}} b$ and $b \sqsubseteq^{\mathrm{A}} a$ hold, and $a \sqsubsetneq^{\mathrm{A}} b$ when $a \sqsubseteq^{\mathrm{A}} b$ holds but $b \sqsubseteq^{\mathrm{A}} a$ does not. We use the following simple observation.

▶ **Observation 16.** *Let M be a monoid and $a, b, c \in M$ be three elements that commute pairwise. Suppose that $abc \sqsubsetneq^{\mathrm{A}} ab$. Then $ac \sqsubsetneq^{\mathrm{A}} a$.*

Proof. We prove the contrapositive. Suppose that $a \sqsubseteq^{\mathrm{A}} ac$. That is, there is some $d \in M$ that commutes with ac and satisfies $acd = a$. We have $dabc = (dac)b = (acd)b = ab$ $abcd = c(abd) = ca = ac$, and thus $dabc = abcd = ab$, proving that $abc \sqsupseteq^{\mathrm{A}} ab$. ◀

Assume that a is not regular. That is, that $a^2 b \neq a$ for every $b \in M$ that commutes with a. Let $\boldsymbol{b} \in \mathcal{M}_{M,a}(n)$ for some number $n > 0$. A coordinate $j \in [n]$ is called *relevant* in $\boldsymbol{b}$ if $\prod_{i \in [m] \setminus j} b_i \sqsupsetneq^{\mathrm{A}} \prod_{i \in [n]} b_i$. Consider the map $\mathcal{I}$ that assigns to each $\boldsymbol{b} \in \mathcal{M}_{M,a}$ its set of relevant coordinates. Claims 1 through 3 proved below establish the required assumptions in Theorem 15 with $L = 1$ and $K = |m|$, thus showing NP-hardness of $\mathrm{PCSP}(\boldsymbol{A}, \boldsymbol{B})$.

Claim 1: b has at most $|M|$ relevant coordinates. Let $\{i_1, \ldots, i_h\} \subseteq [n]$ be the set of relevant coordinates of $\boldsymbol{b}$. Given $k \in [h]$ we define

$$c_k = \prod_{j \in [k-1]} b_{i_j}, \quad \text{and} \quad d_k = \prod_{j \in [n] \setminus \{i_1, \ldots, i_k\}} b_j.$$

The following hold: (1) $a = d_k c_k b_{i_k}$, (2) b_{i_k}, c_k and d_k commute pairwise, and (3) as i_k is a relevant coordinate, it holds that $d_k c_k b_{i_k} \sqsubsetneq^{\mathrm{A}} d_k c_k$. Applying Observation 16, we obtain that $c_k b_{i_k} \sqsubsetneq^{\mathrm{A}} c_k$. Expanding the definition of c_k this means that

$$\prod_{j \in [k]} b_{i_j} \sqsubsetneq^{\mathrm{A}} \prod_{j \in [k-1]} b_{i_j}.$$

This holds for all $k \in [h]$, so in particular the products $\prod_{j \in [k]} b_{i_j}$ must be pairwise different and the number h of relevant coordinates is at most $|M|$, proving the claim.

Claim 2: Minors preserve relevant coordinates. Let $\boldsymbol{c} = \boldsymbol{b}^{(\pi)}$, where $\pi : [n] \to [m]$ is a map and let $i \in [n]$ be a relevant coordinate of $\boldsymbol{b}$. We want to show that $j = \pi(i)$ is a relevant coordinate of $\boldsymbol{c}$. Indeed, if that was not the case we would have that

$$\prod_{k \in [n] \setminus \pi^{-1}(j)} b_k \sqsubseteq^{\mathrm{A}} a.$$

However, $i \in \pi^{-1}(j)$, so we know that $\prod_{k \in [n] \setminus \{i\}} b_k \sqsubseteq^{\mathrm{A}} \prod_{k \in [n] \setminus \pi^{-1}(j)} b_k$. Putting this together with the previous identity shows that

$$\prod_{k \in [n] \setminus \pi^{-1}(j)} b_k \sqsubseteq^{\mathrm{A}} a,$$

contradicting the fact that i was a relevant coordinate of $\boldsymbol{b}$.

Claim 3: b has at least one relevant coordinate. Suppose otherwise for the sake of contradiction. Then for each $i \in [n]$ there is an element $c_i \in M$ that commutes with a such that $ac_i = \prod_{i \in [n] \setminus \{j\}} b_i$. Let $c = \prod_{i \in [n]} c_i$. Observe that c itself commutes with a. However, one can check that that $a^2 c = a$, contradicting our assumption that a was not regular. Indeed,

$$
\begin{aligned}
a^2c &= \left(\prod_{i=1}^{n} b_i\right)(ac_1)\left(\prod_{i=2}^{n} c_i\right) = \left(\prod_{i=1}^{n} b_i\right)\left(\prod_{i\in[n]\setminus\{1\}} b_i\right)\left(\prod_{i=2}^{n} c_i\right) \\
&= \left(\prod_{i=2}^{n} b_i\right)(ac_2)\left(\prod_{i=3}^{n} c_i\right) = \left(\prod_{i=2}^{n} b_i\right)\left(\prod_{i\in[n]\setminus\{2\}} b_i\right)\left(\prod_{i=3}^{n} c_i\right) \\
&= \left(\prod_{i=3}^{n} b_i\right)(ac_3)\left(\prod_{i=4}^{n} c_i\right) = \left(\prod_{i=3}^{n} b_i\right)\left(\prod_{i\in[n]\setminus\{3\}} b_i\right)\left(\prod_{i=4}^{n} c_i\right) \\
&= \cdots = \left(\prod_{i=n}^{n} b_i\right)\left(\prod_{i\in[n]\setminus\{n\}} b_i\right) = a.
\end{aligned}
$$

5 Equations Over Monoids and Groups: Proofs of Theorem 5 and Corollary 6

We need a simple characterisation of the polymorphisms of promise equation templates, and various characterisations of regularity; both are proved in [40].

▶ **Lemma 17.** *Consider a template* $\mathrm{PEqn}(Z_1, Z_2, \varphi)$ *of promise equations over semigroups/-monoids/groups. A map* $p : Z_1^n \to Z_2$ *is a polymorphism of* $\mathrm{PEqn}(Z_1, Z_2, \varphi)$ *if and only if* p *is a semigroup/monoid/group homomorphism and* $p(s, s, \ldots, s) = \varphi(s)$ *for all* $s \in \mathrm{dom}(\varphi)$.

▶ **Lemma 18.** *Let* M *be a monoid and* $s \in M$. *Then the following are equivalent:*

1. s *is regular,*
2. $s^k = s$ *for some* $k > 1$,
3. s *belongs to a subgroup of* M,
4. $s \sqsubseteq s^2$.

With these two lemmas, we can now prove our main result.

▶ **Theorem 5** (Main). $\mathrm{PEqn}(M_1, M_2, \varphi)$ *is solvable in polynomial time by BLP + AIP if there is an Abelian homomorphism* $\psi : M_1 \to M_2$ *extending* φ *and* $\mathrm{Im}(\psi)$ *is a union of subgroups. Otherwise,* $\mathrm{PEqn}(M_1, M_2, \varphi)$ *is* NP*-hard.*

Proof. We prove both implications. Suppose that such homomorphism ψ exists. As $\mathrm{Im}(\psi)$ is a union of subgroups, by Lemma 18 there is some number $k > 1$ such that $s^k = s$ for all $s \in \mathrm{Im}(\psi)$. Let $n > 0$ be arbitrary. Consider the map $p : M_1^{2n+1} \to M_2$ given by

$$
(s_i)_{i\in[2n+1]} \mapsto \left(\prod_{i\in[n+1]} \psi(s_i)\right)\left(\prod_{i\in[n]} \psi(s_{i+n+1})^{k-2}\right),
$$

where the convention is that the zero-th power of an element equals the identity of the monoid. We claim that p is a 2-block-symmetric polymorphism of $\mathrm{PEqn}(M_1, M_2, \varphi)$ with the first block consisting of the first $n+1$ coordinates, and the second block consisting of the rest. The fact that p is a 2-block-symmetric map with the blocks as claimed follows from the fact that ψ is Abelian. To complete the argument, we show that p is a polymorphism of $\mathrm{PEqn}(M_1, M_2, \varphi)$ using the characterisation from Lemma 17. First, observe that the fact that ψ is Abelian implies that p is a monoid homomorphism. Indeed,

$$\begin{aligned} &p(s_1,\dots,s_{2n+1})p(t_1,\dots,t_{2n+1}) \\ &\qquad = \left(\prod_{i\in[n+1]} \psi(s_i)\psi(t_i)\right)\left(\prod_{i\in[n]} \psi(s_{i+n+1})^{k-1}\psi(t_{i+n+1})^{k-1}\right) \\ &\qquad = p(s_1t_1,\dots,s_{2n+1}t_{2n+1}), \end{aligned}$$

so p preserves products. Now we only need to prove that $p(s,\dots,s) = \varphi(s)$ for all $s \in \mathrm{dom}(\varphi)$ in order to show that p is a polymorphism. To see that this holds, observe that

$$p(s,\dots,s) = \psi(s)^{n(k-1)+1} = \psi(s) = \varphi(s),$$

where the last equality uses the fact that ψ extends φ. This completes the proof of the first implication via Theorem 2.

In the other direction, we show that $\mathrm{PEqn}(M_1, M_2, \varphi)$ is NP-hard assuming there is no Abelian homomorphism $\psi : M_1 \to M_2$ extending φ whose image is a union of subgroups. Let $\mathcal{M}$ be the polymorphism minion of $\mathrm{PEqn}(M_1, M_2, \varphi)$. Given a polymorphism $p \in \mathcal{M}$, we define $\mathcal{N}(p)$ as the submonoid $\{p(s,\dots,s) \mid s \in M_1\} \preceq M_2$. Observe that by assumption, for a given polymorphism p it holds that the monoid $\mathcal{N}(p)$ is non-Abelian or that $\mathcal{N}(p)$ is not a union of subgroups. Define Ω as the set of monoid homomorphisms $\psi : M_1 \to M_2$ for which $\mathrm{Im}(\psi)$ is not a union of subgroups. By Lemma 18, this happens precisely when $\mathrm{Im}(\psi)$ contains some non-regular element $a \in M_2$. Let $L = |\Omega| + 1$, and let $K = \max(|M_2|, |\{N \preceq M_2 \mid N \text{ is non-Abelian }\}|$. We use Theorem 15 with the constants L, K to show NP-hardness. We define the following subminions of $\mathcal{M}$.

$$\mathcal{M}_{\mathrm{A}} = \{p \in \mathcal{M}, \mid \mathcal{N}(p) \text{ is not Abelian}\},$$

and given any monoid homomorphism $\psi \in \Omega$ we set

$$\mathcal{M}_{\psi} = \{p \in \mathcal{M}, \mid p(s,\dots,s) = \psi(s) \text{ for all } s \in M_1\}.$$

By the previous observation it holds that

$$\mathcal{M} = \mathcal{M}_{\mathrm{A}} \bigcup_{\psi\in\Omega} \mathcal{M}_{\psi}.$$

We give selection functions $\mathcal{I}$ for each of these sub-minions satisfying the assumptions of Theorem 15. Suppose that $\mathcal{M}_{\mathrm{A}}$ is not empty. Otherwise we are done defining $\mathcal{I}_{\mathrm{A}}$. Let p be any n-ary polymorphism in $\mathcal{M}_{\mathrm{A}}$. Given $i \in [n]$ we define $\mathcal{N}(p,i) \preceq M_2$ as the submonoid

$$\{p(s_1,\dots,s_n) \mid s_i \in M_1, \text{ and } s_j = e \text{ when } j \neq i\}.$$

We give some facts about these submonoids.

Fact 1: The map $\phi : \prod_{i\in[n]} \mathcal{N}(p,i) \to M_2$ given by $(s_1,\dots s_n) \mapsto \prod_{i\in[n]} s_i$ is a monoid homomorphism. In particular, given $1 \leqslant i < j \leqslant n$, any two elements $t_1 \in \mathcal{N}(p,i)$, $t_2 \in \mathcal{N}(p,j)$ commute.

Fact 2: If $\mathcal{N}(p,i) = \mathcal{N}(p,j)$ for some $i \neq j \in [n]$ then $\mathcal{N}(p,i)$ is Abelian.

Fact 3: The submonoid $\mathcal{N}(p)$ is contained in $\mathrm{Im}(\phi)$, where ϕ is as defined in Fact 1. In particular, given that $\mathcal{N}(p)$ is not Abelian, some $\mathcal{N}(p,i)$ must be non-Abelian.

Given an n-ary polymorphism $p \in \mathcal{M}_{\mathrm{A}}$, we define $\mathcal{I}_{\mathrm{A}}(p) \subseteq [n]$ as the set of coordinates i for which $\mathcal{N}(p,i)$ is non-Abelian. We claim that $\mathcal{I}_{\mathrm{A}}$ satisfies the assumptions of Theorem 15. Indeed, given some n-ary p:

- $\mathcal{I}_{\mathrm{A}}(p)$ is non empty by Fact 3.
- $|\mathcal{I}_{\mathrm{A}}(p)| \leqslant K$. Otherwise it would be that $\mathcal{N}(p,i) = \mathcal{N}(p,j)$ for some different $i, j \in \mathcal{I}_{\mathrm{A}}(p)$, contradicting the fact that $\mathcal{N}(p,i)$ is non-Abelian (by Fact 2).
- Suppose that $p = q^{(\pi)}$ for some m-ary q and some $\pi : [m] \to [n]$. Let $i \in \mathcal{I}_{\mathrm{A}}(p)$, then

$$\mathcal{N}(p,i) \subseteq \left\{ \prod_{j \in \pi^{-1}(i)} s_j \mid s_j \in \mathcal{N}(s,j) \text{ for all } j \in \pi^{-1}(i) \right\}.$$

  As $\mathcal{N}(p,i)$ is non-Abelian, some submonoid $\mathcal{N}(q,j)$ with $j \in \pi^{-1}(i)$ must be non-Abelian as well. This means that $\mathcal{I}_{\mathrm{A}}(p) \subseteq \pi(\mathcal{I}_{\mathrm{A}}(q))$.

Now consider an arbitrary homomorphism $\psi \in \Omega$ for which $\mathcal{M}_\psi$ is non-empty. We define a selection function $\mathcal{I}_\psi$ satisfying the assumptions of Theorem 15. Let $t \in \mathrm{Im}(\psi)$ be a non-regular element, and let $s \in M_1$ be such that $\psi(s) = t$. Let $\mathcal{M}_{M_2,t}$ be the monoidal minion defined in Definition 11. Consider the map $\xi : \mathcal{M}_\psi \to \mathcal{M}_{M_2,a}$ that sends any n-ary polymorphism $p \in \mathcal{M}_\psi$ to the tuple $(r_1, \dots, r_n) \in \mathcal{M}_{M_2,a}(n)$ where for each $i \in [n]$

$$r_i = p(s_1, \dots, s_n), \quad \text{where } s_i = s, \text{ and } s_j = e \text{ for all } j \neq i \ .$$

Observe that this is a well-defined minion homomorphism from $\mathcal{M}_\psi$ to $\mathcal{M}_{M_2,t}$. Indeed, first observe that $(r_1, \dots, r_n)$ belongs to the second minion. This holds because $r_1 r_2 \dots r_n = p(s, \dots, s) = \psi(s) = t$, and for each $i \in [n]$ the element r_i belongs to $\mathcal{N}(p,i)$, so the r_i's commute pairwise by Fact 1 above. One can also check that ξ preserves minors.

From the proof of Theorem 12 there is some selection function $\mathcal{I}$ on $\mathcal{M}_{M_2,t}$ satisfying the hypotheses of Theorem 15 for some constant $K' = |M_2| \leqslant K$ and $L = 1$. Thus, we can define $\mathcal{I}_\psi$ on $\mathcal{M}_\psi$ simply by setting $\mathcal{I}_\psi(p) = \mathcal{I}(\xi(p))$ for each polymorphism $p \in \mathcal{M}_\psi$.

Hence, have defined selection functions $\mathcal{I}_{\mathrm{A}}$ and $\mathcal{I}_\psi$ for each $\psi \in \Omega$ that satisfy the requirements of Theorem 15, showing that $\mathrm{PEqn}(M_1, M_2, \varphi)$ is NP-hard. ◀

▶ **Corollary 6.** $\mathrm{PEqn}(G_1, G_2, \varphi)$ *is solvable in polynomial time via AIP if there is an Abelian homomorphism* $\psi : G_1 \to G_2$ *extending* φ*. Otherwise,* $\mathrm{PEqn}(G_1, G_2, \varphi)$ *is* NP*-hard.*

Proof. We prove both directions. The hardness case follows from Theorem 5. Indeed, $\mathrm{PEqn}(G_1, G_2, \varphi)$ is a template of promise equations over monoids (where the monoids just happen to be groups). Suppose that there is no Abelian group homomorphism $\psi : G_1 \to G_2$ that extends φ. Observe that a monoid homomorphism between two groups must also be a group homomorphism, so there is no Abelian monoid homomorphism $\psi : G_1 \to G_2$ that extends φ. Thus, by Theorem 5, $\mathrm{PEqn}(G_1, G_2, \varphi)$ is NP-hard.

In the other direction, suppose that such a ψ exists. We show that $\mathrm{PEqn}(G_1, G_2, \varphi)$ is solved by AIP using Theorem 1. Let n be any odd arity and let $p : G_1^n \to G_2$ be the map given by $p(g_1, \dots, g_n) \mapsto \prod_{i \in [n]} t_i$, where $t_i = \psi(g_i)$ for every odd i, and $t_i = \psi(g_i)^{-1}$ for every even i. Then p is an alternating polymorphism of $\mathrm{PEqn}(G_1, G_2, \varphi)$. ◀

6 Equations over Semigroups: Proof of Theorem 10

A *digraph* $\boldsymbol{D}$ is a relational structure whose signature consists of a single binary relation $E^{\boldsymbol{D}}$.

We follow closely the ideas from [35, Theorem 7]. That result states that every CSP is polynomial-time equivalent to a problem of the form $\mathrm{Eqn}(S, S)$ for some semigroup S. Their proof uses the fact that every CSP is polynomial-time equivalent to another CSP whose template is a digraph $\boldsymbol{D}$ with all singleton unary relations [26]. The fact that they consider these unary relations on $\boldsymbol{D}$ yields equations in $\mathrm{Eqn}(S, S)$ where all constants are allowed. For PCSPs, however, this is our starting point.

▶ **Theorem 19** ([13]). *For every template* $(\boldsymbol{A}_1, \boldsymbol{A}_2)$ *there is a template* $(\boldsymbol{D}_1, \boldsymbol{D}_2)$ *of digraphs such that* $\mathrm{PCSP}(\boldsymbol{A}_1, \boldsymbol{A}_2)$ *is polynomial-time equivalent to* $\mathrm{PCSP}(\boldsymbol{D}_1, \boldsymbol{D}_2)$.

The fact that we lack singleton unary relations in the templates $(\boldsymbol{D}_1, \boldsymbol{D}_2)$ is the main obstacle for applying the techniques from [35]. We overcome this by extending our digraphs with an additional edge joining two fresh distinguished vertices. The relational signature σ^+ contains one binary relation symbol E, and two unary relation symbols P, Q. Given a digraph $\boldsymbol{D}$, we write $\boldsymbol{D}^+$ for the σ^+ structure defined by $D^+ = D \cup \{p, q\}$, where p and q are fresh vertices, $E^{\boldsymbol{D}^+} = E^{\boldsymbol{D}} \cup \{(p, q)\}$, $P^{\boldsymbol{D}^+} = \{p\}$, and $Q^{\boldsymbol{D}^+} = \{q\}$.

▶ **Lemma 20.** *Let* $(\boldsymbol{D}_1, \boldsymbol{D}_2)$ *be a template of digraphs. Then* $\mathrm{PCSP}(\boldsymbol{D}_1, \boldsymbol{D}_2)$ *is polynomial-time equivalent to* $\mathrm{PCSP}(\boldsymbol{D}_1^+, \boldsymbol{D}_2^+)$.

Proof. We give polynomial-time Turing reductions in both directions. First, we reduce from $\mathrm{PCSP}(\boldsymbol{D}_1, \boldsymbol{D}_2)$ to $\mathrm{PCSP}(\boldsymbol{D}_1^+, \boldsymbol{D}_2^+)$. We consider two cases. Suppose that $E^{\boldsymbol{D}_2}$ is empty. Then $\mathrm{PCSP}(\boldsymbol{D}_1, \boldsymbol{D}_2)$ amounts to deciding whether a given instance $\boldsymbol{I}$ has an edge or not, which takes polynomial time. Otherwise, assume that $E^{\boldsymbol{D}_2}$ is non-empty. Then our reduction takes any instance $\boldsymbol{I}$ of $\mathrm{PCSP}(\boldsymbol{D}_1, \boldsymbol{D}_2)$ and considers it as an instance of $\mathrm{PCSP}(\boldsymbol{D}_1^+, \boldsymbol{D}_2^+)$. Clearly, if $\boldsymbol{I}$ maps homomorphically to $\boldsymbol{D}_1$ then it also maps homomorphically to $\boldsymbol{D}_1^+$ using the same homomorphism. Otherwise, if $\boldsymbol{I}$ does not map homomorphically to $\boldsymbol{D}_2$ then it cannot map homomorphically to $\boldsymbol{D}_2^+$. Indeed, to see this observe that the digraph resulting from of $\boldsymbol{D}_2^+$ (by forgetting about the P, Q relations) maps homomorphically to $\boldsymbol{D}_2$: it suffices to send the edge (p, q) to an arbitrary edge in $E^{\boldsymbol{D}_2}$, which is non-empty by assumption.

Now we describe a polynomial-time reduction from $\mathrm{PCSP}(\boldsymbol{D}_1^+, \boldsymbol{D}_2^+)$ to $\mathrm{PCSP}(\boldsymbol{D}_1, \boldsymbol{D}_2)$. The reduction considers an instance $\boldsymbol{I}$ of $\mathrm{PCSP}(\boldsymbol{D}_1^+, \boldsymbol{D}_2^+)$ and checks in polynomial time whether every connected component of $\boldsymbol{I}$ that intersects $P^{\boldsymbol{I}}$ or $Q^{\boldsymbol{I}}$ maps homomorphically to the edge structure $\boldsymbol{W}$ with $W = \{p, q\}$, $E^{\boldsymbol{W}} = \{(p, q)\}$, $P^{\boldsymbol{W}} = \{p\}$, and $Q^{\boldsymbol{W}} = \{q\}$. If this is not the case, $\boldsymbol{I}$ is rejected. Otherwise, we remove from $\boldsymbol{I}$ the components that intersect $P^{\boldsymbol{I}}$ or $Q^{\boldsymbol{I}}$. Next, we check in polynomial time whether each remaining component of $\boldsymbol{I}$ can be mapped homomorphically to $\boldsymbol{W}$, and removes the ones that do. If the resulting structure $\boldsymbol{I}'$ is empty, then our reduction accepts $\boldsymbol{I}$. Otherwise, observe that the resulting instance $\boldsymbol{I}'$ is equivalent to the original $\boldsymbol{I}$, in the sense that $\boldsymbol{I}$ maps to $\boldsymbol{D}_i^+$ if and only if $\boldsymbol{I}'$ does so as well. Furthermore, observe that a homomorphism from $\boldsymbol{I}'$ to $\boldsymbol{D}_i^+$ cannot include p and q in its image, as there are no components in $\boldsymbol{I}'$ that map homomorphically to $\boldsymbol{W}$. This means that $\boldsymbol{I}'$ maps to $\boldsymbol{D}_i^+$ if and only if it maps to $\boldsymbol{D}_i$. Hence, as the last step in our reduction we simply use $\boldsymbol{I}'$ as an instance of $\mathrm{PCSP}(\boldsymbol{D}_1, \boldsymbol{D}_2)$. ◀

A semigroup S is a *right-normal band* if $ss = s$ for all $s \in S$ and $rst = srt$ for all $r, s, t \in S$. Recall that we write $s \sim r$ if $s \sqsubseteq r$ and $r \sqsubseteq s$ hold. It is easy to see that the quotient $\widehat{S} = S/\sim$ inherits the semigroup structure from S. Moreover, $\widehat{S}$ is a *semilattice*, meaning that it is an Abelian semigroup where every element is idempotent. Given an instance $\boldsymbol{I}$ of $\mathrm{Eqn}(S, S)$ we denote by $\widehat{\boldsymbol{I}}$ the corresponding instance over $\widehat{S}$, where every constant s is substituted by its $\sim$ class $\hat{s}$.

We need two lemmas from [35] and a simple observation.

▶ **Lemma 21** ([35]). *Let* S *be a semilattice. Then* $\mathrm{Eqn}(S, S)$ *can be solved in polynomial time. Moreover, if an instance* $\boldsymbol{I}$ *has a solution, it also has a unique minimal one (with respect to the* $\sqsubseteq$ *preorder) that can be obtained in polynomial time.*

▶ **Lemma 22** ([35]). *Let* S *be a right-normal band. Then an instance* $\boldsymbol{I}$ *of* $\mathrm{Eqn}(S, S)$ *is solvable if it has a solution satisfying* $f(x) \in \hat{s}_x$*, for all* $x \in I$*, where the map* $x \mapsto \hat{s}_x$ *is the minimal solution of* $\widehat{\boldsymbol{I}}$ *in* $\mathrm{Eqn}(\widehat{S}, \widehat{S})$.

▶ **Observation 23.** *Let S be a right-normal band, and let $s, s', t \in S$ be three arbitrary elements with $s \sim s'$. Then $st = s't$.*

Proof. As $s \sim s'$ and S is right-normal, it must hold that $s = s'r'$ and $s' = sr$ for some $r, r' \in S$. Thus, $st = s'r't = srr't$, and $s't = srt = s'r'rt = srr'rt = srr't$, where the last equality holds since S is a right-normal band. ◀

Let $\boldsymbol{D}$ be a digraph. We define a semigroup S_D related to $\boldsymbol{D}$ in a similar fashion as [35]. The main difference is that we need to "plant" a special subsemigroup S_W inside S_D that is used later as the set of constants in our promise equations. The semigroup $S = S_D$ is a right-normal band. It has the following $\sim$-classes: $V^{\mathrm{L}}, V^{\mathrm{R}}, V^{\mathrm{LC}}, V^{\mathrm{LR}}, V^{\mathrm{CR}}, E^{\mathrm{C}}, 0$, described as follows. Given $\square \in \{\mathrm{L}, \mathrm{R}, \mathrm{LC}, \mathrm{LR}, \mathrm{CR}\}$ the class $V^{\square}$ is a copy of $D \cup \{p, q\}$. That is, $V^{\square} = \{v^{\square} \mid v \in D\} \cup \{p^{\square}, q^{\square}\}$. The class E^{C} is a copy of $E^D \cup \{(p, q)\}$, meaning that $E^{\mathrm{C}} = \{(u, v)^{\mathrm{C}} \mid (u, v) \in E^D\} \cup \{(p, q)^{\mathrm{C}}\}$. Finally, the class 0 contains a single element 0. By Observation 23, in a right-normal band T it must hold that $st = s't$ for all $s, s', t \in T$ with $s \sim s'$. Hence, given a $\sim$-class $C \subseteq T$ and an element t we abuse the notation and write Ct to denote the product of an arbitrary element from C with t. The product operation in S is given by the following rules:

$$\begin{aligned} V^{\mathrm{R}}v^{\mathrm{L}} = V^{\mathrm{L}}v^{\mathrm{R}} = V^{\mathrm{LR}}v^{\mathrm{R}} = V^{\mathrm{LR}}v^{\mathrm{L}} = V^{\mathrm{L}}v^{\mathrm{LR}} = V^{\mathrm{R}}v^{\mathrm{LR}} = v^{\mathrm{LR}} \\ V^{\mathrm{L}}v^{\mathrm{LC}} = V^{\mathrm{LC}}v^{\mathrm{L}} = E^{\mathrm{C}}v^{\mathrm{L}} = E^{\mathrm{C}}v^{\mathrm{LC}} = v^{\mathrm{LC}} \\ V^{\mathrm{R}}v^{\mathrm{CR}} = V^{\mathrm{CR}}v^{\mathrm{R}} = E^{\mathrm{C}}v^{\mathrm{R}} = E^{\mathrm{C}}v^{\mathrm{CR}} = v^{\mathrm{CR}}, \end{aligned}$$

where v is an arbitrary element in $D \cup \{p, q\}$. Additionally,

$$V^{\mathrm{L}}(u, v)^{\mathrm{C}} = V^{\mathrm{LC}}(u, v)^{\mathrm{C}} = u^{\mathrm{LC}}, \quad \text{and} \quad V^{\mathrm{R}}(u, v)^{\mathrm{C}} = V^{\mathrm{CR}}(u, v)^{\mathrm{C}} = v^{\mathrm{CR}},$$

where (u, v) belongs to $E^D \cup \{(p, q)\}$. Finally, all other products not described above have 0 as their result.

We define the subsemigroup $S_W \preceq S_D$ as the one containing the elements $0, (p, q)^{\mathrm{C}}, p^{\square}, q^{\square}$ for $\square \in \{\mathrm{L}, \mathrm{R}, \mathrm{LC}, \mathrm{LR}, \mathrm{CR}\}$. Observe that for any digraph D, the quotient $\widehat{S_D} = S_D/\sim$ is isomorphic to $\widehat{S_W} = S_W/\sim$.

▶ **Lemma 24.** *There is a polynomial-time algorithm Φ that takes as an input a σ^+-structure $\boldsymbol{I}$ and outputs a system of equations $\Phi(\boldsymbol{I})$ with constants in S_W satisfying that for any digraph $\boldsymbol{D}$, $\boldsymbol{I}$ maps into $\boldsymbol{D}^+$ if and only if $\Phi(\boldsymbol{I})$ has a solution over S_D.*

Proof. This follows the first reduction in [35, Theorem 7] while making sure that all constants remain in S_W. We construct the system $\Phi(\boldsymbol{I})$. For every vertex $v \in I$ we include variables $v^{\mathrm{L}}, v^{\mathrm{R}}, v^{\mathrm{LR}}$. For each $\square \in \{\mathrm{L}, \mathrm{R}, \mathrm{LR}\}$ we include the constraint $v^{\square} \in V^{\square}$, which is a shorthand for the equations $p^{\square}v^{\square} = v^{\square}$ and $v^{\square}p^{\square} = p^{\square}$. We also include the equations $p^{\mathrm{LR}}v^{\mathrm{L}} = v^{\mathrm{LR}}$ and $p^{\mathrm{LR}}v^{\mathrm{R}} = v^{\mathrm{LR}}$. If $v \in P^{\boldsymbol{I}}$ we include all constraints $v^{\square} = p^{\square}$ for $\square \in \{\mathrm{L}, \mathrm{R}, \mathrm{LR}\}$. Similarly, if $v \in Q^{\boldsymbol{I}}$, then we include the constraints of the form $v^{\square} = q^{\square}$. For each edge $(u, v) \in E^{\boldsymbol{I}}$ we include a variable $(u, v)^{\mathrm{C}}$ in $\Phi(\boldsymbol{I})$, together with the constraint $(u, v)^{\mathrm{C}} \in E^{\mathrm{C}}$, which is a shorthand for the equations $(u, v)^{\mathrm{C}}(p, q)^{\mathrm{C}} = (p, q)^{\mathrm{C}}$ and $(p, q)^{\mathrm{C}}(u, v)^{\mathrm{C}} = (u, v)^{\mathrm{C}}$. Finally, we also add the equations $p^{\mathrm{LC}}(u, v)^{\mathrm{C}} = p^{\mathrm{LC}}u^{\mathrm{L}}$ and $p^{\mathrm{CR}}(u, v)^{\mathrm{C}} = p^{\mathrm{CR}}v^{\mathrm{R}}$. The resulting system $\Phi(\boldsymbol{I})$ satisfies the statement of the theorem. ◀

▶ **Lemma 25.** *There is a polynomial-time algorithm Ψ that takes as an input a system of equations $\boldsymbol{X}$ with constants in S_W and produces one of the following outcomes:*

- **(I)** *It outputs a σ^+-structure $\Psi(\boldsymbol{X})$ that maps into $\boldsymbol{D}^+$ for any digraph $\boldsymbol{D}$ if and only if $\boldsymbol{X}$ has a solution over S_D, or*
- **(II)** *it rejects $\boldsymbol{X}$ and $\boldsymbol{X}$ has no solution over S_D for any digraph $\boldsymbol{D}$.*

Proof. We describe the algorithm Ψ. This algorithm is meant to transform the system $\boldsymbol{X}$ into a system of the form $\Phi(\boldsymbol{I})$, for the algorithm Φ given in Lemma 24 and some σ^+-structure $\boldsymbol{I}$. This time we follow the second reduction in [35, Theorem 7] while making sure that all constants in $\boldsymbol{X}$ remain in S_W throughout all the transformations.

Without loss of generality, we may assume that every equation in $\boldsymbol{X}$ is initially of the form $x_1x_2 = x_3$, for some variables x_1, x_2, x_3, or of the form $x = s$, for some variable x and some element $s \in S_W$. Consider the system $\widehat{\boldsymbol{X}}$ with constants in $\widehat{S_W} = S_W/\sim$. By Lemma 21 we can find a minimal solution of $\widehat{\boldsymbol{X}}$ in polynomial time. If such a solution does not exist, then the system $\boldsymbol{X}$ is not satisfiable over S_D for any digraph $\boldsymbol{D}$, and the algorithm Ψ just rejects it. Otherwise, suppose that the system $\widehat{\boldsymbol{X}}$ has some minimal solution. This solution maps each variable $x \in X$ to a $\sim$-class C_x of S_W. Consider an arbitrary digraph $\boldsymbol{D}$. Using the observation that $\widehat{S_W} \simeq \widehat{S_D}$ and Lemma 22, we deduce that $\boldsymbol{X}$ has a solution over S_D if and only if it has a solution where the value of each variable $x \in X$ belongs to the class C_x. Given a class C_x, we define the constant $c_x \in S_W$ as

- $p^\square$ if C_x is the class $V^\square$ for $\square \in \{\mathrm{L}, \mathrm{R}, \mathrm{LC}, \mathrm{LR}, \mathrm{CR}\}$,
- $(p, q)^{\mathrm{C}}$ if $C_x = E^{\mathrm{C}}$, or
- 0 if $C_x = 0$.

For each variable $x \in X$ we add the equations $c_x x = x$ and $xc_x = c_x$. These equations are equivalent to the constraint that $x \in C_x$ (and we use $x \in C_x$ as a shorthand for those equations), so the resulting system is satisfiable over a semigroup S_D if and only if the original one was. Additionally, once every variable x is constrained to take values inside C_x, we can replace every equation of the form $x_1x_2 = x_3$ in $\boldsymbol{X}$ with the equation $c_{x_3}x_2 = c_{x_3}x_3$ to yield an equivalent system. Indeed, it must hold that $c_{x_i}x_i = x_i$, so the equation $x_1x_2 = x_3$ is equivalent to $c_{x_1}x_1c_{x_2}x_2 = c_{x_3}x_3$. Not only that, but S_D is a normal band and $x_1c_{x_1} = c_{x_1}$, so last equation is equivalent to $c_{x_1}c_{x_2}x_2 = c_{x_3}x_3$. Finally, the classes $C_{x_1}, C_{x_2}, C_{x_3}$ were part of a solution to $\widehat{\boldsymbol{X}}$, so it must be that $c_{x_1}c_{x_2} \sim c_{x_3}$, and by Observation 23 it holds that $c_{x_1}c_{x_2}x_1 = c_{x_3}x_1$.

Every resulting equation of the form $0x_1 = 0x_2$ is trivially satisfied and can be discarded. Consider a variable $x \in X$ whose corresponding class C_x is 0. As we have removed every equation of the form $0x_1 = 0x_2$, x can only appear in constraints of the form $x \in 0$, and $x = 0$. These are trivially satisfiable by any assignment that maps x to 0, so we can remove the variable x and all equations containing it.

We are left with a system $\boldsymbol{X}$ where each variable is bound to a class $V^\square$ for $\square \in \{\mathrm{L}, \mathrm{R}, \mathrm{LC}, \mathrm{LR}, \mathrm{CR}\}$ or E^{C}. Consider a variable $x \in X$ bound to the class V^{LC}. Suppose this variable appears in some equation of the form $c_1x = c_1y$, and consider the class C of c_1. By construction, it must be that $C \supseteq V^{\mathrm{LC}}$ in $\widehat{S_W}$. However, we have removed all equations containing 0, so the only possibility left is that $C = V^{\mathrm{LC}}$. Suppose that we replace the requirement $x \in V^{\mathrm{LC}}$ with $x \in V^{\mathrm{L}}$ and every equation of the form $x = v^{\mathrm{LC}}$, where $v^{\mathrm{LC}} \in S_W$ is a constant, with $x = v^{\mathrm{L}}$. We claim the system $\boldsymbol{X}$ remains equivalent after these changes. Indeed, this results from the observation that $V^{\mathrm{LC}}v^{\mathrm{L}} = V^{\mathrm{LC}}v^{\mathrm{LC}}$ in any semigroup S_D for any vertex $v \in D^+$. By the same logic we can also replace any requirement of the kind $x \in V^{\mathrm{LR}}$ or $x \in V^{\mathrm{CR}}$ with $x \in V^{\mathrm{CR}}$.

Consider any equation of the form $x = (u, v)^{\mathrm{C}}$ for a constant $(u, v)^{\mathrm{C}}$. This equation is equivalent to the constraints $p^{\mathrm{LC}}x = p^{\mathrm{LC}}y$, $p^{\mathrm{CR}}x = p^{\mathrm{CR}}z$, $y = u^{\mathrm{L}}$ and $z = v^{\mathrm{R}}$, where y and z are fresh variables.

Consider an equation of the form $cx = cy$, where both x, y are constrained to be in c's $\sim$-class. $(p, q)^{\mathrm{C}}x = (p, q)^{\mathrm{C}}y$, both This equation holds if and only if $x = y$. Hence, we may remove this equation and identify both variables x, y together.

This far we have obtained a system $\boldsymbol{X}$ where each variable is bound to either $V^{\mathrm{L}}, V^{\mathrm{R}}$ or E^{C}, and the only constants are among $p^{\mathrm{L}}, p^{\mathrm{R}}, q^{\mathrm{L}}, q^{\mathrm{R}}$. Identifying variables and adding dummy variables if necessary we can assume the following hold:

- For each variable $x \in X$ constrained by $x \in E^{\mathrm{C}}$ there is exactly one variable x_{L} constrained by $x_{\mathrm{L}} \in V^{\mathrm{L}}$ in an equation of the form $p^{\mathrm{LC}}x = p^{\mathrm{LC}}x_{\mathrm{L}}$, and exactly one variable x_{R} constrained by $x_{\mathrm{R}} \in V^{\mathrm{R}}$ that appears in an equation of the form $p^{\mathrm{CR}}x = p^{\mathrm{CR}}x_{\mathrm{R}}$.
- There are no two variables $x, y \in X$ constrained by $x, y \in E^{\mathrm{C}}$ with $x_{\mathrm{L}} = y_{\mathrm{L}}$ and $x_{\mathrm{R}} = y_{\mathrm{L}}$.
- Not considering equations that are part of the constraints $x \in C$ for some $\sim$-class C, each equation is of the form (i) $p^{\mathrm{LR}}x = p^{\mathrm{LR}}y$ with $x \in V^{\mathrm{L}}$ and $y \in V^{\mathrm{R}}$, (ii) $p^{\mathrm{LC}}x = p^{\mathrm{LC}}x_{\mathrm{L}}$ or $p^{\mathrm{CR}}x = p^{\mathrm{CR}}x_{\mathrm{R}}$ for some $x \in E^{\mathrm{C}}$, or (iii) $x = p^{\square}$ or $x = q^{\square}$ for $\square \in \{\mathrm{L}, \mathrm{R}\}$.

One can see that such a system corresponds to $\Phi(\boldsymbol{I})$ for some σ^+-structure $\boldsymbol{I}$ that can be built in polynomial time. Then Ψ returns $\boldsymbol{I}$, which satisfies our requirements by Lemma 24. ◀

▶ **Corollary 26.** *Let $(\boldsymbol{D}_1, \boldsymbol{D}_2)$ be a template of digraphs. Then $\mathrm{PCSP}(\boldsymbol{D}_1, \boldsymbol{D}_2)$ is polynomial-time equivalent to $\mathrm{PEqn}(S_{D_1}, S_{D_2}, \varphi)$, where $\varphi = \mathrm{id}_{S_W}$.*

Proof. We show that $\mathrm{PEqn}(S_{D_1}, S_{D_2}, \varphi)$ is polynomial-time equivalent to $\mathrm{PCSP}(\boldsymbol{D}_1^+, \boldsymbol{D}_2^+)$, which suffices by Lemma 20. Observe that algorithm Φ given in Lemma 25 is a polynomial-time Turing reduction from $\mathrm{PCSP}(\boldsymbol{D}_1^+, \boldsymbol{D}_2^+)$ to $\mathrm{PEqn}(S_{D_1}, S_{D_2}, \varphi)$, and algorithm Ψ, given in Lemma 24 is a polynomial-time Turing reduction in the other direction. ◀

Corollary 26 and Theorem 19 establish Theorem 10.

References

1. Kristina Asimi and Libor Barto. Finitely tractable promise constraint satisfaction problems. In *Proc. 46th International Symposium on Mathematical Foundations of Computer Science (MFCS'21)*, volume 202 of *LIPIcs*, pages 11:1–11:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.11.
2. Albert Atserias and Víctor Dalmau. Promise constraint satisfaction and width. In *Proc. 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA'22)*, pages 1129–1153, 2022. doi:10.1137/1.9781611977073.48.
3. Per Austrin, Venkatesan Guruswami, and Johan Håstad. $(2+\epsilon)$-Sat is NP-hard. *SIAM J. Comput.*, 46(5):1554–1573, 2017. doi:10.1137/15M1006507.
4. Libor Barto, Diego Battistelli, and Kevin M. Berg. Symmetric Promise Constraint Satisfaction Problems: Beyond the Boolean Case. In *Proc. 38th International Symposium on Theoretical Aspects of Computer Science (STACS'21)*, volume 187 of *LIPIcs*, pages 10:1–10:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.10.
5. Libor Barto, Jakub Bulín, Andrei A. Krokhin, and Jakub Opršal. Algebraic approach to promise constraint satisfaction. *J. ACM*, 68(4):28:1–28:66, 2021. doi:10.1145/3457606.
6. Libor Barto and Marcin Kozik. Combinatorial Gap Theorem and Reductions between Promise CSPs. In *Proc. 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA'22)*, pages 1204–1220, 2022. doi:10.1137/1.9781611977073.50.
7. Amey Bhangale and Subhash Khot. Optimal Inapproximability of Satisfiable k-LIN over Non-Abelian Groups. In *Proc. 53rd Annual ACM Symposium on Theory of Computing (STOC'21)*, pages 1615–1628. ACM, 2021. doi:10.1145/3406325.3451003.
8. Amey Bhangale, Subhash Khot, and Dor Minzer. On Approximability of Satisfiable k-CSPs: II. In *Proc. 55th Annual ACM Symposium on Theory of Computing (STOC'23)*, pages 632–642. ACM, 2023. doi:10.1145/3564246.3585120.

9 Amey Bhangale, Subhash Khot, and Dor Minzer. On Approximability of Satisfiable k-CSPs: III. In *Proc. 55th Annual ACM Symposium on Theory of Computing (STOC'23)*, pages 643–655. ACM, 2023. doi:10.1145/3564246.3585121.

10 Manuel Bodirsky. *Complexity of infinite-domain constraint satisfaction*, volume 52. Cambridge University Press, 2021.

11 Manuel Bodirsky and Martin Grohe. Non-dichotomies in Constraint Satisfaction Complexity. In *Proc. 35th International Colloquium on Automata, Languages and Programming (ICALP'08)*, volume 5126 of *Lecture Notes in Computer Science*, pages 184–196. Springer, 2008. doi:10.1007/978-3-540-70583-3_16.

12 Manuel Bodirsky and Thomas Quinn-Gregson. Solving equation systems in ω-categorical algebras. *J. Math. Log.*, 21(3), 2021. doi:10.1142/S0219061321500203.

13 Joshua Brakensiek and Venkatesan Guruswami. Promise Constraint Satisfaction: Algebraic Structure and a Symmetric Boolean Dichotomy. *SIAM J. Comput.*, 50(6):1663–1700, 2021. doi:10.1137/19M128212X.

14 Joshua Brakensiek, Venkatesan Guruswami, and Sai Sandeep. Conditional Dichotomy of Boolean Ordered Promise CSPs. *TheoretiCS*, 2, 2023. doi:10.46298/theoretics.23.2.

15 Joshua Brakensiek, Venkatesan Guruswami, and Sai Sandeep. SDPs and robust satisfiability of promise CSP. In *Proc. 55th Annual ACM Symposium on Theory of Computing (STOC'23)*, pages 609–622. ACM, 2023. doi:10.1145/3564246.3585180.

16 Joshua Brakensiek, Venkatesan Guruswami, Marcin Wrochna, and Stanislav Živný. The power of the combined basic LP and affine relaxation for promise CSPs. *SIAM J. Comput.*, 49:1232–1248, 2020. doi:10.1137/20M1312745.

17 Alex Brandts and Stanislav Živný. Beyond PCSP(1-in-3,NAE). *Information and Computation*, 2022. doi:10.1016/j.ic.2022.104954.

18 Andrei Bulatov, Peter Jeavons, and Andrei Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34(3):720–742, 2005. doi:10.1137/S0097539700376676.

19 Andrei A. Bulatov. Complexity of conservative constraint satisfaction problems. *ACM Trans. Comput. Log.*, 12(4):24:1–24:66, 2011. doi:10.1145/1970398.1970400.

20 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *Proc. 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS'17)*, pages 319–330, 2017. doi:10.1109/FOCS.2017.37.

21 Lorenzo Ciardo and Stanislav Živný. Approximate graph colouring and the hollow shadow. In *Proc. 55th Annual ACM Symposium on Theory of Computing (STOC'23)*, pages 623–631. ACM, 2023. doi:10.1145/3564246.3585112.

22 Lorenzo Ciardo and Stanislav Živný. CLAP: A new algorithm for promise CSPs. *SIAM J. Comput.*, 52(1):1–37, 2023. doi:10.1137/22M1476435.

23 Lorenzo Ciardo and Stanislav Živný. Hierarchies of minion tests for PCSPs through tensors. In *Proc. 2023 ACM-SIAM Symposium on Discrete Algorithms (SODA'23)*, pages 568–580, 2023. doi:10.1137/1.9781611977554.ch25.

24 Victor Dalmau and Jakub Opršal. Local consistency as a reduction between constraint satisfaction problems. *CoRR*, 2023. arXiv:2301.05084.

25 Lars Engebretsen, Jonas Holmerin, and Alexander Russell. Inapproximability results for equations over finite groups. *Theor. Comput. Sci.*, 312(1):17–45, 2004. doi:10.1016/S0304-3975(03)00401-8.

26 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998. doi:10.1137/S0097539794266766.

27 Miron Ficak, Marcin Kozik, Miroslav Olšák, and Szymon Stankiewicz. Dichotomy for Symmetric Boolean PCSPs. In *Proc. 46th International Colloquium on Automata, Languages, and Programming (ICALP'19)*, volume 132, pages 57:1–57:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.57.

28 M. R. Garey and David S. Johnson. The complexity of near-optimal graph coloring. *J. ACM*, 23(1):43–49, 1976. doi:10.1145/321921.321926.

29 Mikael Goldmann and Alexander Russell. The complexity of solving equations over finite groups. *Inf. Comput.*, 178(1):253–262, 2002. doi:10.1006/INCO.2002.3173.

30 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1–24, 2007. doi:10.1145/1206035.1206036.

31 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001. doi:10.1145/502090.502098.

32 Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990. doi:10.1016/0095-8956(90)90132-J.

33 John M Howie. *Fundamentals of semigroup theory*. Oxford University Press, 1995.

34 Peter G. Jeavons, David A. Cohen, and Marc Gyssens. Closure properties of constraints. *J. ACM*, 44(4):527–548, 1997. doi:10.1145/263867.263489.

35 Ondřej Klíma, Pascal Tesson, and Denis Thérien. Dichotomies in the complexity of solving systems of equations over finite semigroups. *Theory Comput. Syst.*, 40(3):263–297, 2007. doi:10.1007/S00224-005-1279-2.

36 Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.*, 61(2):302–332, 2000. doi:10.1006/jcss.2000.1713.

37 Michael Kompatscher. The equation solvability problem over supernilpotent algebras with Mal'cev term. *International Journal of Algebra and Computation*, 28(06):1005–1015, 2018. doi:10.1142/S0218196718500443.

38 Andrei Krokhin, Jakub Opršal, Marcin Wrochna, and Stanislav Živný. Topology and adjunction in promise constraint satisfaction. *SIAM J. Comput.*, 52(1):37–79, 2023. doi:10.1137/20M1378223.

39 Benoît Larose and László Zádori. Taylor terms, constraint satisfaction and the complexity of polynomial equations over finite algebras. *Int. J. Algebra Comput.*, 16(3):563–582, 2006. doi:10.1142/S0218196706003116.

40 Alberto Larrauri and Stanislav Živný. Solving promise equations over monoids and groups. *CoRR*, 2024. arXiv:2402.08434.

41 Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60(6), 2013. Article No. 42. doi:10.1145/2535926.

42 Peter Mayr. On the complexity dichotomy for the satisfiability of systems of term equations over finite algebras. In *Proc. 48th International Symposium on Mathematical Foundations of Computer Science (MFCS'23)*, volume 272 of *LIPIcs*, pages 66:1–66:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.MFCS.2023.66.

43 Tamio-Vesa Nakajima and Stanislav Živný. Linearly ordered colourings of hypergraphs. *ACM Trans. Comput. Theory*, 13(3–4), 2022. doi:10.1145/3570909.

44 Tamio-Vesa Nakajima and Stanislav Zivný. Boolean symmetric vs. functional PCSP dichotomy. In *Proc. 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'23)*, 2023. doi:10.1109/LICS56636.2023.10175746.

45 Thomas Schaefer. The complexity of satisfiability problems. In *Proc. 10th Annual ACM Symposium on the Theory of Computing (STOC'78)*, pages 216–226, 1978. doi:10.1145/800133.804350.

46 Steve Seif and Csaba Szabó. Algebra complexity problems involving graph homomorphism, semigroups and the constraint satisfaction problem. *J. Complex.*, 19(2):153–160, 2003. doi:10.1016/S0885-064X(02)00027-4.

47 Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *J. ACM*, 67(5):30:1–30:78, 2020. doi:10.1145/3402029.

Smoothed Analysis of Deterministic Discounted and Mean-Payoff Games

Bruno Loff
LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal

Mateusz Skomra
LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

Abstract

We devise a policy-iteration algorithm for deterministic two-player discounted and mean-payoff games, that runs in polynomial time with high probability, on any input where each payoff is chosen independently from a sufficiently random distribution and the underlying graph of the game is ergodic.

This includes the case where an arbitrary set of payoffs has been perturbed by a Gaussian, showing for the first time that deterministic two-player games can be solved efficiently, in the sense of smoothed analysis.

More generally, we devise a *condition number* for deterministic discounted and mean-payoff games played on ergodic graphs, and show that our algorithm runs in time polynomial in this condition number.

Our result confirms a previous conjecture of Boros et al., which was claimed as a theorem [18] and later retracted [19]. It stands in contrast with a recent counter-example by Christ and Yannakakis [24], showing that Howard's policy-iteration algorithm does *not* run in smoothed polynomial time on *stochastic* single-player mean-payoff games.

Our approach is inspired by the analysis of random optimal assignment instances by Frieze and Sorkin [39], and the analysis of bias-induced policies for mean-payoff games by Akian, Gaubert and Hochart [6].

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases Mean-payoff games, discounted games, policy iteration, smoothed analysis

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.147

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2402.03975`

Funding Funded by the European Union (ERC, HOFGA, 101041696). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. Also supported by FCT through the LASIGE Research Unit, ref. UIDB/00408/2020 and ref. UIDP/00408/2020.

Acknowledgements MS would like to thank Xavier Allamigeon, Stéphane Gaubert, and Ricardo D. Katz for many useful discussions on mean-payoff games, policy iteration, and the operator approach, for exchanging ideas about the problem of smoothed analysis, for their remarks on a preliminary version of this paper, and for being a perpetual source of friendship and inspiration.

1 Extended Abstract

1.1 A history of discounted and mean-payoff games

John von Neumann proved his minimax theorem in 1928, founding game theory by showing the existence of optimal strategies in zero-sum matrix games. In 1953, Lloyd Shapley [75] considered what happened if two players repeatedly played a zero-sum matrix game. The overall game proceeds as follows. We have n states, and to each state $i \in [n]$ corresponds


a zero-sum matrix game G_i. At each round, the two players are in some state $i \in [n]$ and play the corresponding game G_i, with each player simultaneously choosing an action out of a finite set of possible actions. The two players' choice of actions determines not just the payoff, but also the state at the next round. This is repeated ad-infinitum.

In these games, randomness is possible in two different ways. First, the state at the next round can be chosen stochastically, or deterministically, based on the current state and on the players' chosen actions. This gives us two variants: *stochastic* games, and *deterministic* games, with the latter being a special case of the former. Second, the players' choice of action can itself be *pure* (a single action), or *mixed* (a distribution over the possible actions).

In an infinite game such as this there are two natural ways of determining the winning player. In the *discounted* variant, payoffs received at round t are multiplied by a *discount factor* of γ^t (for some $0 \leqslant \gamma < 1$), and we wish to know the total discounted payoff in the limit as the number of rounds goes to infinity. This is equivalent to saying that the game is forced to stop after every round with probability $1 - \gamma$, and asking for the expected payoff at the limit. In the *mean-payoff* variant, we measure the liminf or limsup, as the number of rounds goes to infinity, of the average payoff received so far (i.e. total payoff divided by the number of rounds). Shapley [75] proved the existence of a value and optimal (mixed) strategies for the stochastic, discounted variant.

Concurrently to Shapley's work, Bellman [16] studied a class of problems which he termed *Markov Decision Processes* (MDPs). MDPs model decision making when the result of one's actions can be partly random, and they can be seen as *single-player* variant of stochastic games. At each round, the player finds himself in a given state out of a finite number of states, and chooses an action. Depending on his choice he receives a payoff, and transitions to a different state. This transition can be either deterministic or stochastic. The player's goal is to maximize the discounted payoff or mean payoff at the limit, as the number of rounds goes to infinity. Bellman provided a method to find an optimal pure strategy in the discounted variant.

In both MDPs and in discounted games the optimal strategies can be made *memoryless*, in that the choice of what to do only depends on the current state i, and not on the past history. In the general case of discounted stochastic games where the players play simultaneously, the optimal memoryless strategies must be mixed. In the case of MDPs, the optimal memoryless strategy can further be made *pure*.

As for the mean-payoff variant, Gillette [42] gave an example of a mean-payoff two-player game, where the players play simultaneously at each round, whose optimal strategies cannot be memoryless.[1] With this in mind, Gillette introduced a *turn-based* variant of Shapley's infinite game, where two players play in turns. At each round, the game is in some state i, and one of the players (depending on i) chooses an action, which determines the next state (stochastically or deterministically) and a resulting payoff. One player is trying to maximize the payoff at the limit, and the other tries to minimize it. Gillette claimed that turn-based two-player games have a value and that optimal strategies exist for both players which are both memoryless and pure. Gillette's proof was actually wrong, but it was later corrected by Liggett and Lippman [60], so the statement *is* true. It also implies the corresponding statement for the mean-payoff variant of MDPs, as a special case.

[1] In fact, it was only in the 1980s [63] that simultanous-move mean-payoff games were proven to have a value. For every $\varepsilon > 0$, optimal (not memoryless) strategies exist for each player ensuring that the payoff is ε-close to the value.

A pure, memoryless strategy for such games is called a *policy*, and it is a finite object: one can represent it as a finite function $\sigma : \mathsf{States} \to \mathsf{Actions}$ specifying the chosen action at each state. In this paper, we will not concern ourselves with simultaneous two-player games, and only consider single-player games and turn-based two-player games. We will use the informal nomenclature "deterministic/stochastic single-player/two-player discounted/mean-payoff game" to denote each of the eight variants of (non-simultaneous) games just mentioned. Let us also use the term "discounted and mean-payoff games" to refer to these games as whole.

1.2 Algorithms

The above results show that all eight variants have a value, and that optimal strategies are policies, hence finite objects. It then makes sense to consider the algorithmic problem of *solving* such a game: given as input a specification of the game with rational weights (and with rational discount factor, if applicable), compute the value of the game, and optimal policies for the players.[2] The study of discounted and mean-payoff games has always been accompanied by the development of algorithms for solving them. Most algorithms for solving discounted and mean-payoff games can be broadly classified into three families: value-iteration algorithms[3], algorithms for MDPs that use linear programming[4], and, of particular importance to us, policy iteration algorithms.

Policy iteration algorithms have been invented for solving all variants of games described above. These algorithms maintain a policy in memory, and proceed by repeatedly modifying the policy, so that its quality improves monotonically according to some measure, until it can no longer be improved, at which point the measure must guarantee that we have found an optimal strategy for both players.

The first policy iteration algorithm was invented by Howard [52], and finds an optimal strategy for deterministic (and some stochastic) Markov Decision Processes. This was later extended by Denardo and Fox [30] to work on all stochastic MDPs. The method was first extended to two-player mean-payoff games by Hoffman and Karp [50], and two-player discounted games by Denardo [29], with later developments by many other authors [70, 67, 25, 40, 69, 26]. A good historical overview with more technical details appears in [3], where a policy iteration algorithm first appeared that can handle all the variants of mean-payoff games. In the case of discounted games, an optimal strategy can be found in time polynomial in $\frac{1}{1-\gamma}$ [78, 47]. Otherwise, for mean-payoff games, or for discount factors γ exponentially close to 1, no upper-bound is known on the number of iterations, significantly better than the number of policies, which is exponential in the number n of states. More precisely, the best upper-bound on the number of iterations is $2^{\tilde{O}(\sqrt{n})}$ [46, 48].

1.3 Policy iteration versus the simplex method

When one first studies policy iteration algorithms, one gets a sense of familiarity, as if policy iteration algorithms are analogous to the simplex method for linear programming. The intuitive sense is that the choice of policy plays the same role as the choice of basic feasible solution in the simplex method, with a change in policy being analogous to a pivot operation.

[2] It can be shown that the value of such a game with rational weights (and discount factor) is a rational number of comparable size.

[3] The first algorithm ever invented was a value iteration algorithm [16]. For a modern value-iteration algorithm for single-player games, see [76], which contains a historical overview in Section 2. For two players see, e.g., [79, 54, 23, 13, 8].

[4] See, e.g., Chapter 2 of [37], or various sections of [68].

In fact, in some cases, this analogy can be formally established. It is possible to express a MDP by a particular linear program, and in this particular case the connection is perfect: a simplex pivoting rule gives us a policy iteration algorithms for MDPs, and any policy iteration algorithm that switches a single node at a time gives us a pivoting rule for applying simplex on this particular program.

As a result, many known counter-examples for the simplex method, showing that certain pivoting rules require an exponential number of pivots, were devised by first finding examples of MDPs for which certain policy-iteration algorithms need an exponential number of iterations, and then *translating* the counter-example to work for the simplex algorithm, by the above connection [38, 33].

More broadly, it turns out that deterministic two-player mean-payoff games are exactly equivalent to tropical linear programming, i.e., solving systems of "linear" inequalities over the tropical $(\min, +)$ semiring. This was first explicitly shown by Akian, Gaubert, and Guterman [4], strengthening earlier connections between these problems that were made in the literature on tropical algebra (such as [40, 32, 58]) and in works on scheduling problems [64].[5]

Furthermore, tropical linear programs can be reduced to linear programs over the non-Archimedean field of convergent generalized power series [31, 11].[6] This characterization has been exploited to show that, if there exists a strongly-polynomial-time pivoting rule for the simplex algorithm, where the choice of basis element to pivot is semialgebraic in a certain technical sense (and this is the case for many pivoting rules), then the entire algorithm can be tropicalized, to get a polynomial-time algorithm for deterministic two-player mean-payoff games [10].

The analogy between policy iteration and the simplex method is also seen in practice. The aforementioned counter-examples show that policy-iteration algorithms run in exponential time in the worst-case. And yet, various benchmarks have shown that policy-iteration algorithms are very efficient at solving real-world instances, both for single-player [41, 27, 59] and two-player games [32, 21]. This difference between worst-case and real-life performance is also what happens with the simplex method. And in both cases it begs the question: *why?*

1.4 Smoothed analysis

In the case of the simplex method, the generally accepted explanation was proposed by Spielman and Teng [77]. They have shown that, if one takes any linear programming instance $\max\{c \cdot x \mid A \cdot x \geqslant b\}$ of dimension n, and perturbs each entry of A, b and c by a Gaussian with mean 0 and standard deviation $\frac{1}{\phi}$, then the simplex method, with a particular choice of pivoting rule, will solve the resulting perturbed system in time $\text{poly}(\phi \cdot n)$ [77, 28]. It is then reasonable to expect the simplex method to work efficiently on real-world instances, since they incorporate real-world data which is prone to such perturbations. It was this result of Spielman and Teng that founded the area of *smoothed analysis*, where one studies the efficiency of algorithms on such perturbed inputs.

---

[5] Stochastic two-player mean-payoff games, on the other hand, are equivalent to tropical semidefinite programming [14].

[6] A linear program over such a field can be thought of as a parametric family of linear programs over $\mathbb{R}$. It follows from the above reduction that deterministic mean-payoff games can be encoded as linear programs with coefficients of exponential bit-length. Such an encoding was first derived by Schewe [73], without reference to non-Archimedean fields.

The question then naturally follows: are policy-iteration algorithms efficient in the sense of smoothed analysis?

Recent evidence seems to indicate that no, they are not. The first policy-iteration algorithm for single-player games (MDPs), by Howard [52], determines for the current policy $\sigma : \mathsf{States} \to \mathsf{Actions}$, and for each state i of the game, if a local improvement is possible: *would a different choice of action at i improve the value of the game when starting at i, if the game were to be played according to σ at every other state?* The algorithm then changes the action $\sigma(i)$ at every state i where such a local improvement is possible, to the best possible local improvement. This is sometimes called *Howard's policy iteration*, or the *greedy all-switches* rule.

Last year, Christ and Yannakakis [24] showed a remarkable lower bound. They showed that $2^{\Omega(n^c)}$ iterations are necessary on a certain family of stochastic MDPs (single-player games), even when the payoffs are perturbed. In fact, the lower bound holds not only probabilistically, where each payoff is independently perturbed by a Gaussian with standard deviation $\frac{1}{\mathrm{poly}(n)}$, but even adversarially, where each payoff is perturbed by any value within $\pm\frac{1}{\mathrm{poly}(n)}$.

It is surprising that such a bound can be proven at all. However, their result only holds for *stochastic* games, and does not necessarily apply to deterministic games, where it has been previously conjectured that Howard's rule is efficient [49]. Also, this result shows that a particular way of improving the policy, the greedy all-switches rule, does not give us an efficient algorithm (in the sense of smoothed analysis). This is analogous to saying that a particular pivoting rule in the simplex algorithm is not efficient, and does not exclude the possibility that other ways of improving the policy might work.

Exploiting any one of these caveats could in principle allow for a smoothed analysis for policy iteration. Our main result exploits both: we show that for *deterministic* two-player games, a *slightly different* policy-improvement method will be efficient, in the sense of smoothed analysis.

▶ **Theorem 1** (our main theorem). *There exists a policy-iteration algorithm for solving n-state deterministic two-player (discounted or mean-payoff) games played on ergodic graphs, which runs in time* $\mathrm{poly}(\phi \cdot n)$ *with high probability, on an input where normalized payoffs in* $[-1, 1]$ *have been independently perturbed by a Gaussian with mean* 0 *and standard deviation* $\frac{1}{\phi}$.

It should be emphasized that the lower bound of Christ and Yannakakis holds even for the single-player case, and our result could be contrasted with theirs, even if we only had proved it for the single-player case. However, our policy-iteration algorithm (our upper-bound) works even for two-player games, which are much harder. Our policy-improvement rule is similar to the greedy all-switches rule, except that the choice of switches is allowed to depend on an additional parameter (a discount factor) which evolves over time.

1.5 Computational complexity

Our result should also be contrasted with the case of the simplex algorithm for linear programming. Recall that we *do* know polynomial-time algorithms for linear programming, it is only the simplex algorithm which fails to run efficiently in the worst case. However, it should be emphasized that we do *not* know any polynomial-time algorithms for solving two-player discounted or mean-payoff games.

Indeed, solving one-player discounted and mean-payoff games reduces to linear programming, so we have polynomial-time algorithms. But the complexity of solving *two-player* discounted and mean-payoff games is one of the great unsolved problems in computational

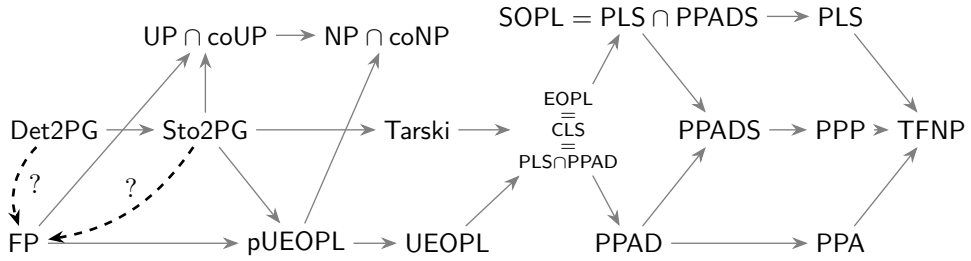


Figure 1 A hierarchy of NP search problems. Det2PG and Sto2PG refer to deterministic, respectively stochastic, two-player games. Arrows denote inclusion or containment. By inclusion of a search problem in the classes UP ∩ coUP and NP ∩ coUP of decision problems, we mean that the problem of deciding each bit of the unique answer can be computed in these classes.

complexity, first posed by Gurvich, Karzanov, and Khachiyan [44]. For any of the two-player variants, the respective decision problem (what is the i-th bit of the value) is in UP∩coUP [56], and the search problem (find an optimal strategy) is in the computational complexity class UEOPL (*Unique End-of-Potential-Line*) [53, 35]. In fact it is in the promise version of this class, which we denote pUEOPL. Even within the class pUEOPL, the problem of solving two-player discounted and mean-payoff games seems to be only a special, simple case: the sought optimal policies can be obtained from the unique fixed point of a simple monotone operator. This places the search problem in the class Tarski. The two complexity classes pUEOPL and Tarski sit at the bottom of a large hierarchy of complexity classes [34, 43]. This hierarchy stratifies the broad class TFNP of NP *search problems* [55, 62, 66]. See Figure 1.

In this sense, the problem of solving two-player discounted and mean-payoff games is the simplest known problem in NP, which is not yet known to be solvable in polynomial time (or even in time $2^{n^{o(1)}}$). For any problem which is not known to be in P, one may ask the question: *is the problem still hard on a random instance?* Many hard problems have been conjectured to have this property, of being hard to solve even for a random instance. This is the case for SAT [74, 36] and subset sum [72], but also for other, not necessarily NP-hard, problems in NP, such as lattice problems [2]. There is a broad belief that natural problems which are hard, remain hard on random inputs.[7] In contrast, our main theorem generalizes to show that solving deterministic two-player games is easy, for any sufficiently random input distribution.

▶ **Theorem 2** (generalization). *Consider distributions on n-state deterministic two-player discounted or mean-payoff games played on an ergodic graph, where each payoff is chosen independently according to a (not necessarily identical) distribution with mean in $[-1, 1]$ and standard deviation $\leqslant \frac{1}{\phi}$ and with probability density functions satisfying $f(y) \leqslant \phi$ for all $y \in \mathbb{R}$. (This includes, for example, payoffs perturbed by a Gaussian, or payoffs sampled from a uniform interval of length $\geqslant \frac{1}{\phi}$.)*

There exists a policy-iteration algorithm which runs in time $\mathrm{poly}(n, \phi)$*, with high probability, on games sampled according to any such distribution.*

[7] The distributions which are considered hard must be chosen carefully to avoid trivial cases, e.g. CNF formulas with too many or too few clauses, but there are often simple and natural distributions.

1.6 A previous approach and our approach

A first, naive attempt at proving Theorem 1 could proceed along the same lines as the Mulmuley, Vazirani and Vazirani's isolation lemma [65]. We think of what happens to the game when all of the payoffs of all the actions are fixed, except for one, which is sampled independently according to some distribution as above. Let us suppose that the free payoff is for an action of some state i. As it turns out, when every other payoff is fixed, the value of the game at state i is a piecewise linear function of the free payoff, and one can then hope that it has few break points. If this were indeed the case, that there were only $\text{poly}(n)$ break points, one could then argue similarly to the MVV isolation lemma, to show that approximating the random payoffs using $O(\log(n))$ bits of precision is enough to isolate the linear piece (in the piecewise linear function). From this it would follow that optimal policies for the truncated payoffs are also optimal for the untruncated payoffs. One could then invoke a pseudo-polynomial-time value-iteration algorithm [79] on the truncated payoffs, and this would run in polynomial time.

The conference version of the paper of Boros, Elbassioni, Fouz, Gurvich, Makino and Manthey [18] outlines a similar proof strategy. Among other results, a result similar to our Theorem 1 was claimed [18, Theorem 4.6]. Their proof works for the one-player case, and the authors claimed, without a careful proof, that the two-player case also follows. This claim was later retracted in the journal version [19].[8] Indeed, it turns out that the two-player case is significantly more subtle.

In the one-player case it can be shown that for every action there exist at most n break points, and hence an isolation lemma can be proven. One can then conjecture, for the two-player case, a $\text{poly}(n)$ upper-bound on the number of break points. As it turns out, this conjecture is wrong. An exponential example can be created using the construction of [17] that was also used in [12] to prove that interior point methods for linear programming are not strongly polynomial. This construction gives us a deterministic two-player game with n states such that, leaving the payoffs of all but one of actions fixed, as the free payoff varies between -1 and 1, the value of the game is a piecewise-linear function with $2^{\Omega(n)}$ break points.

So what do we do instead? One natural thing to try is to show that the number of break points *is* $\text{poly}(n)$, with very high probability, for randomly-chosen payoffs. This could well be true, and an argument in the style of the MVV isolation lemma would then follow. But we were unable to show it.

Instead, our results depend on a deeper analysis of deterministic two-player games. We also prove an isolation lemma, but using an approach different to MVV. Instead of attempting to isolate an optimal policy among all possible policies, we show that sufficiently random payoffs, with high probability, isolate a Blackwell-optimal policy. Blackwell-optimal policies are policies that arise in discounted games with discount factor γ close to 1. Blackwell-optimal policies are part of a family of policies which are induced by an object called a *bias*. Not all optimal policies are Blackwell-optimal, or even bias-induced. Nonetheless, every two-player discounted game has a Blackwell-optimal policy [60]. Furthermore, there exist policy-iteration algorithms for finding a Blackwell-optimal policy [57, 51] (that are inefficient in the worst case).

[8] In their paper, the breakpoints are chosen so that between any two breakpoints the value of the optimal strategy and the value of the second-best strategy are sufficiently far apart. This works for the single player-case, but the argument we just presented, where we only keep track of the number of break points in the value function, is simpler and also works.

We are then able to show that, if the payoffs are sufficiently random, then with high probability it will happen that there is a unique bias-induced policy, which must then be the Blackwell-optimal policy. Furthermore, from the proof of this theorem we devise a *condition number* $\Delta(r)$, associating a number in $[0, +\infty]$ to any given choice r of payoffs. The uniqueness proof generalizes to show that a deterministic two-player game with sufficiently random payoffs will have a small condition number ($\Delta(r) \leqslant \text{poly}(n)$) with high probability.

This is a condition number in the same sense as the known condition numbers that govern the complexity of algorithms for solving linear equations, semidefinite feasibility, *etc*, and broadly measure the *inverse-distance to ill-posedness* (see [20]). Although, strictly speaking, our condition number does not measure inverse-distance to some set, it does have the property that $\Delta(r)$ is finite if and only if the game with payoffs r has a unique bias-induced policy, and that every payoff $\tilde{r} \in B_\infty(r, \delta)$, within a ball of size $\delta \leqslant \frac{1}{\text{poly}(\Delta(r))}$ around r, will also have a unique bias-induced policy. So, at least intuitively, we can think of $\Delta(r)$ as an inverse-distance between r and the set of games with multiple bias-induced policies.

Finally, it can be shown that, taking a discount rate $\gamma \geqslant 1 - \frac{1}{\text{poly}(n,\Delta)}$ (i.e., sufficiently close to 1), the only optimal policy is the Blackwell-optimal policy. We can then use the results of [78, 47] to obtain a policy iteration algorithm for finding the Blackwell optimal policy in time $\text{poly}(n, \Delta)$. This algorithm runs in polynomial smoothed time, because, as mentioned above, sufficiently random payoffs have small condition number. This includes any fixed choice of payoffs that has been randomly perturbed by a Gaussian.

1.7 Related work

There is not a lot of work on the complexity of discounted and mean-payoff games with random payoffs. Besides the papers of Boros et al. [18, 19] and Christ and Yannakakis [24], which we mentioned above, we only know of a paper by Mathieu and Wilson [61]. They do not provide an algorithm, but they analyze the distribution of the value of a deterministic single-player mean-payoff game (deterministic MDP) played on a complete graph with i.i.d. exponentially distributed payoffs. Our algorithm will also work on such a payoff distribution.

A paper of Allamigeon, Benchimol, and Gaubert [9] analyzes efficiency in a different random model. The shadow vertex rule is known to be efficient on average under any distribution over linear feasibility problems, which is symmetric up to changing of the sign in each linear inequality [1]. Allamigeon et al. tropicalize this result, to show that a certain tropical analogue of the shadow vertex rule will solve deterministic two-player mean-payoff games in a bipartite graph in expected polynomial time, if the distribution over the payoff matrix is invariant under transposition (this is the tropical analogue of the above symmetry). In particular, if the payoffs of some given fixed input obey the same symmetry, their algorithm runs in polynomial time. Of course, in general, perturbed payoffs need not be symmetric in this way.

It was a paper of Frieze and Sorkin [39] that gave us the first idea of how to approach the problem. Frieze and Sorkin analyse the gap between optimal and second-optimal assignment in the assignment problem, using a bound on the reduced costs of the associated linear program at the optimum solution [39, Theorem 3]. In the simplex algorithm, the reduced cost works as a gradient, telling us the improvement in the objective function obtained by changing the current basic solution in a given direction. Frieze and Sorkin show that, at an optimum basic solution of a random assignment problem, every reduced cost is large, which implies that there is a large difference between the optimum and second-best solution. This large difference implies that the optimum solution is stable under perturbations. In the case of deterministic two-player games, biases will play the role of dual variables, which

allows us define an analogous notion of reduced costs at the optimal solution. We then show, analogously, that, with high probability for a random instance, every reduced cost is large at the optimum policy, which also implies stability. Our condition number is the (normalized) inverse of the smallest reduced cost.

Our analysis of discounted and mean-payoff games is based on the operator approach to study these games. Using this approach, Akian, Gaubert, and Hochart [6] have previously shown that a generic two-player mean-payoff game has a unique bias (which must then equal the Blackwell bias). More precisely, they show that for any stochastic or deterministic two-player mean-payoff game, the set of payoffs where the bias is not unique has measure zero. We give a more precise version of this result, for deterministic two-player mean-payoff games, by showing that the policies induced by the unique bias are also unique. This further allows us the measure "how far from having multiple bias-induced policies" is a given choice of payoffs.

The operator approach was also used by Allamigeon, Gaubert, Katz and Skomra [13], who define a condition number for the value iteration algorithm. Computer experiments indicate that value iteration converges quickly for random games [14], which strongly suggests that the condition number of [13] is small for random games, but there is currently no formal proof of this claim. Even though our condition number and the one from [13] are based on the bias vector, it is not clear how these two conditions numbers compare to each other. In particular, we do not know if the value iteration algorithm has polynomial smoothed complexity and we leave this problem as an open question.

2 Technical summary

For the sake of simplicity, we restrict our attention to deterministic mean-payoff games played on an *ergodic* weighted directed graph $\vec{\mathcal{G}} = ([n], E, r)$, where $|E| = m$ and $[n]$ is split into vertices controlled by players Max and Min, $[n] = V_{\text{Max}} \uplus V_{\text{Min}}$. The ergodicity is taken in the sense of [45, 5, 7]: a graph is called ergodic if the value of any mean-payoff game played on this graph is independent of the initial state of the game.[9] A typical example of such a graph is a complete bipartite graph, in which the bipartition is formed by $V_{\text{Max}}, V_{\text{Min}}$. Ergodic graphs are representative for the difficulty of mean-payoff games, because solving games on general graphs reduces to solving games on complete bipartite graphs [22]. We note however that it is not clear if this reduction can be done in the smoothed analysis setting. We leave the problem of extending our results to non-ergodic graphs as a question for future research.

The weights r_{ij} of the edges in our model are chosen randomly: we suppose that (r_{ij}) are independent absolutely continuous variables with densities f_{ij}. We further suppose that weights are normalized – $\mathbb{E}(r_{ij}) \in [-1, 1]$ – and that there exists a number $\phi > 0$ such that $f_{ij}(y) \leqslant \phi$ for all i, j, y and $\text{Var}(r_{ij}) \leqslant 1/\phi^2$. As an example, if the weights r are taken by perturbing some fixed weights $\bar{r}_{ij} \in [-1, 1]$ by Gaussian noise, so that $r_{ij} \sim \mathcal{N}(\bar{r}_{ij}, \rho^2)$, then we can take $\phi := 1/\rho$.

Under the ergodicity assumption, it is known [44, 7] that the following *ergodic equation* has a solution $(\lambda, u) \in \mathbb{R}^{n+1}$ for all choices of weights:

$$\begin{cases} \forall i \in V_{\text{Max}},\ \lambda + u_i = \max_{(i,j)\in E}\{r_{ij} + u_j\}\,, \\ \forall i \in V_{\text{Min}},\ \lambda + u_i = \min_{(i,j)\in E}\{r_{ij} + u_j\}\,. \end{cases}$$

[9] This is a notion similar to strong connectivity, but for two-player games. Intuitively, a graph is ergodic if no player can play in such a way as to force the game to get stuck on a sub-graph.

Furthermore, the number λ is unique and it is the value of the game (which does not depend on the initial state because of ergodicity)[10]. The vector $u \in \mathbb{R}^n$, called a *bias*, is never unique because the set of solutions contains at least one line: we can always add a constant to all coordinates of u. In general, the set of biases may consist of more than one line. We say that a pair of policies $\sigma\colon V_{\mathrm{Max}} \to V$ (of Max) and $\tau\colon V_{\mathrm{Min}} \to V$ (of Min) is *bias-induced* if there is a bias u such that the edges used by σ, τ achieve the maxima and minima in the ergodic equation. Bias-induced policies are optimal [44], but not every optimal policy is bias-induced. To study the behavior of random games, we introduce the sets

$$\mathcal{P}^{\sigma,\tau} \coloneqq \{r \in \mathbb{R}^m\colon (\sigma,\tau) \text{ is the only pair of bias-induced policies in the MPG with weights } r\},$$

for any pair (σ, τ) such that the resulting graph $\vec{\mathcal{G}}^{\sigma,\tau}$ has a single directed cycle. We denote by Ξ the set of all such pairs of policies. We also put $\mathcal{U} \coloneqq \cup\mathcal{P}^{\sigma,\tau}$. Using the techniques from [6] we are then able to show the following proposition. This proposition strengthens [6, Theorem 3.2] for deterministic games by showing that each maximum and minimum in the ergodic equation is generically achieved by a single edge.

▶ **Proposition 3** (cf. [6, Theorem 3.2])**.** *The sets $\mathcal{P}^{\sigma,\tau}$ are open polyhedral cones. Moreover, these cones are disjoint and $\mathbb{R}^m \setminus \mathcal{U}$ is included in a finite union of hyperplanes. In particular, this set has Lebesgue measure zero. Furthermore, if $r \in \mathcal{U}$, then the ergodic equation has a single solution (up to adding a constant to the bias), and each maximum and minimum in the ergodic equation is achieved by a single edge.*

This motivates the introduction of the following *condition number* Δ, which measures the difference between the edge that achieves a maximum or minimum in the ergodic equation and the "second best" edge, relatively to the spread of the weights around the value:

▶ **Definition 4.** *Given $r \in \mathcal{U}$, we put*

$$\Delta(r) \coloneqq \frac{\max\{|r_{ij} - \lambda|\colon (i,j) \in E\}}{\min\{|r_{ij} - \lambda + u_j - u_i|\colon (i,j) \in E, r_{ij} - \lambda + u_j - u_i \neq 0\}}\,.$$

When defined in such a way, the condition number does not change when the weights are multiplied by a positive constant, or when the same constant is added to all the weights.

To analyze the behavior of random games, we introduce the following random variables. If i is a vertex controlled by Min, then for any edge $(i,j) \in E$ we put

$$Z_{ij} = \inf\{x \in \mathbb{R}\colon (x, r_{-ij}) \in \mathcal{U} \text{ and the MPG with weights } (x, r_{-ij}) \text{ has a pair of bias-induced policies } (\sigma,\tau) \in \Xi \text{ such that } \tau(i) \neq j\}\,.$$

Here, (x, r_{-ij}) is the vector obtained from r by replacing the ijth coordinate with x. We analogously define the variables Z_{ij} for vertices controlled by Max, changing inf to sup. Since the variable Z_{ij} does not depend on r_{ij}, we get the estimate

▶ **Lemma 5.** *For any $\alpha > 0$, $\mathbb{P}(\exists ij, |r_{ij} - Z_{ij}| \leqslant \alpha) \leqslant 2\alpha m\phi$.*

[10] This is a fundamental result which appeared already in [44]: in the paper's only theorem, $p(v)$ is the value and $c'_{ij} = r_{ij} + u_j - u_i$ are the payoffs modified by u. In the reference [7], the existence of λ and u is line (iii) of the much more general Theorem 2.1, which applies to additive eigenvectors of additively homogeneous monotone operators.

Furthermore, the variables Z_{ij} are related to the ergodic equation in the following way.

▶ **Lemma 6.** *Suppose that $r \in \mathcal{P}^{\sigma,\tau}$ for some $(\sigma,\tau) \in \Xi$. Then, for every $(i,j) \in E$ that is not used in (σ,τ) we have $Z_{ij} = \lambda + u_i - u_j$.*

The two lemmas above improve the conclusion of Proposition 3: not only each maximum and minimum in the ergodic equation is achieved by a single edge, but with high probability the difference between the best edge and the second best edge is large. In particular, this shows that bias-induced policies do not change when the random weights are truncated, and it gives an estimate of the condition number.

▶ **Theorem 7.** *Let $\delta := 1/(4n(2n+1)m\phi)$. Then, with probability at least $1-1/n$, the whole ℓ_∞ ball $B_\infty(r,\delta)$ is included in a single polyhedron $\mathcal{P}^{\sigma,\tau}$.*

▶ **Theorem 8.** *Random mean payoff games are well conditioned with high probability. More formally, for every $\varepsilon > 0$ we have $\mathbb{P}\big(\Delta \geqslant \frac{8m}{\varepsilon}(\phi + \sqrt{\frac{2m}{\varepsilon}})\big) \leqslant \varepsilon$.*

To propose an algorithm that exploits the condition number, we use the fact that every mean-payoff game has a pair of *Blackwell-optimal* policies, i.e., policies that are optimal for all discount factors γ close to 1. Such policies are induced by the *Blackwell bias*, which is defined as $u^* := \lim_{\gamma\to 1}(\lambda^{(\gamma)} - \lambda)/(1-\gamma)$, where $\lambda^{(\gamma)}$ is the value of the discounted game with discount factor γ. We then show that, for well-conditioned games, the Blackwell-optimal policies can be already found when the discount factor is low.

▶ **Theorem 9.** *Suppose that $r \in \mathcal{P}^{\sigma,\tau}$ and fix $1 > \gamma > 1 - \frac{1}{6n^2\Delta(r)}$. Then, (σ,τ) is the unique pair of optimal policies in the discounted game with discount factor γ.*

Combining Theorems 8 and 9 with the results of [78, 47] showing that policy iteration has polynomial complexity for discount factor $\gamma < 1 - \frac{1}{\mathrm{poly}(n)}$, we get our final result.[11]

▶ **Theorem 10.** *The greedy-all switches policy iteration rule combined with increasing discount factor solves random instances of deterministic discounted or mean-payoff games in polynomial smoothed complexity.*

In the theorem above, "polynomial smoothed complexity" is defined as in [15, 71]: there exists a polynomial $\mathrm{poly}(x_1,x_2,x_3,x_4)$ such that for all $\varepsilon \in]0,1]$ the probability that the number of iterations of our algorithm exceeds $\mathrm{poly}(n,m,\phi,\frac{1}{\varepsilon})$ is at most ε.

3 Conclusion and future work

We gave an analysis of two-player discounted and mean-payoff games, that led to a condition number, and a policy-iteration algorithm which is efficient on well-conditioned inputs. We showed that random inputs are well-conditioned with high probability. A few remarks are in order.

1. Our techniques work for two-player games played on ergodic graphs. In non-ergodic graphs, the value λ_i is not necessarily the same at each vertex i. A folklore reduction, appearing for example in [22], shows that computing the value vector of a non-ergodic

[11] To wit: since the pair of optimal policies is unique, we can find them using an algorithm for discount factor $\gamma < 1 - \frac{1}{\mathrm{poly}(n,\phi)}$, and the same policies will be optimal for any higher γ and also for the mean-payoff game.

game reduces to computing the value of an ergodic game. So one can ask if our algorithms can be used on non-ergodic games. The answer is not obvious. The reduction proceeds in rounds, where in the first round one finds, say, the largest coordinate λ_i of the value vector, and then discards the node i (which requires some care) and repeats. Now, if one takes a non-ergodic game $\vec{\mathcal{G}}$ with sufficiently random payoffs, and applies this reduction, the resulting game is sufficiently random at the first round, but it is not clear what happens in the succeeding rounds. So, as far as we can tell, the question remains open: *Do deterministic two-player discounted and mean-payoff games have polynomial smoothed complexity, when played on non-ergodic graphs?* A possible way of answering this question is by doing an analysis of Blackwell-optimal policies in the non-ergodic case, similar to what we have done here for the ergodic case.

2. Allamigeon, Gaubert, Katz and Skomra [13] show that a certain value-iteration algorithm runs efficiently on all ergodic instances with value λ bounded away from zero. They use $\frac{\max_i u_i - \min_i u_i}{|\lambda|}$ as a condition number. Can we use their result to show that value iteration has polynomial smoothed complexity? I.e., is a sufficiently-random instance well-conditioned as per their condition number? This was the central question left unanswered in their paper, and we tried to solve it, or provide a counter-example, but have so far failed to do so.
3. Our policy-iteration rule is not one of the standard rules (Howard, lexicographic, RandomFacet, *etc*). Do these standard rules also have polynomial smoothed complexity on deterministic two-player games? How about other "combinatorial" algorithms?
4. Can we extend our results to *stochastic* two-player games? The counter-example of Christ and Yannakakis shows that the Howard all-switches rule does not have polynomial smoothed complexity on stochastic two-player games. This seems to indicate that the stochastic setting is more delicate. On the other hand, our policy iteration rule is different to Howard's. So one could tentatively ask: is there a smoothed counter-example to the Howard rule also in the deterministic (say, two-player) setting? This would show that our policy-iteration rule cannot be replaced by the Howard rule.
5. How about other problems in UEOPL? Some of these problems are combinatorial, and do not seem to be amenable to smoothed analysis. But one can consider, for example, the P-Matrix Linear Complementarity Problem (P-LCP, see [35, Section 4.3]), and ask: *does it have polynomial smoothed complexity?* More broadly speaking, one can make the conjecture that *every problem in* UEOPL *becomes easy under a suitable notion of perturbation*. This conjecture is broad and imprecise, but it might be an interesting starting point for further research.

References

1 Ilan Adler, Richard M. Karp, and Ron Shamir. A simplex variant solving an $m \times d$ linear program in $O(\min(m^2, d^2))$ expected number of pivot steps. *Journal of Complexity*, 3(4):372–387, 1987.

2 Miklós Ajtai. Generating hard instances of the short basis problem. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 1–9, 1999.

3 M. Akian, J. Cochet-Terrasson, S. Detournay, and S. Gaubert. Policy iteration algorithm for zero-sum multichain stochastic games with mean payoff and perfect information, 2012. `arXiv:1208.0446`.

4 M. Akian, S. Gaubert, and A. Guterman. Tropical polyhedra are equivalent to mean payoff games. *Int. J. Algebra Comput.*, 22(1):125001 (43 pages), 2012. doi:10.1142/S0218196711006674.

5 M. Akian, S. Gaubert, and A. Hochart. Ergodicity conditions for zero-sum games. *Discrete Contin. Dyn. Syst.*, 35(9):3901–3931, 2015. doi:10.3934/dcds.2015.35.3901.

6 M. Akian, S. Gaubert, and A. Hochart. Generic uniqueness of the bias vector of finite zero-sum stochastic games with perfect information. *J. Math. Anal. Appl.*, 457:1038–1064, 2018. doi:10.1016/j.jmaa.2017.07.017.

7 M. Akian, S. Gaubert, and A. Hochart. A game theory approach to the existence and uniqueness of nonlinear Perron-Frobenius eigenvectors. *Discrete & Continuous Dynamical Systems - A*, 40:207–231, 2020. doi:10.3934/dcds.2020009.

8 M. Akian, S. Gaubert, U. Naepels, and B. Terver. Solving irreducible stochastic mean-payoff games and entropy games by relative Krasnoselskii-Mann iteration. In *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*, volume 272 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2023.10.

9 X. Allamigeon, P. Benchimol, and S. Gaubert. The tropical shadow-vertex algorithm solves mean payoff games in polynomial time on average. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 8572 of *Lecture Notes in Comput. Sci.*, pages 89–100. Springer, 2014. doi:10.1007/978-3-662-43948-7_8.

10 X. Allamigeon, P. Benchimol, S. Gaubert, and M. Joswig. Combinatorial simplex algorithms can solve mean payoff games. *SIAM J. Optim.*, 24(4):2096–2117, 2014. doi:10.1137/140953800.

11 X. Allamigeon, P. Benchimol, S. Gaubert, and M. Joswig. Tropicalizing the simplex algorithm. *SIAM J. Discrete Math.*, 29(2):751–795, 2015. doi:10.1137/130936464.

12 X. Allamigeon, P. Benchimol, S. Gaubert, and M. Joswig. Log-barrier interior point methods are not strongly polynomial. *SIAM J. Appl. Algebra Geom.*, 2(1):140–178, 2018. doi:10.1137/17M1142132.

13 X. Allamigeon, S. Gaubert, R. D. Katz, and M. Skomra. Universal complexity bounds based on value iteration and application to entropy games. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 126:1–126:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

14 X. Allamigeon, S. Gaubert, and M. Skomra. Solving generic nonarchimedean semidefinite programs using stochastic game algorithms. *J. Symbolic Comput.*, 85:25–54, 2018. doi:10.1016/j.jsc.2017.07.002.

15 R. Beier and B. Vöcking. Typical properties of winners and losers in discrete optimization. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 343–352. ACM, 2004. doi:10.1145/1007352.1007409.

16 Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

17 M. Bezem, R. Nieuwenhuis, and E. Rodríguez-Carbonell. Exponential behaviour of the Butkovič–Zimmermann algorithm for solving two-sided linear systems in max-algebra. *Discrete Appl. Math.*, 156(18):3506–3509, 2008. doi:10.1016/j.dam.2008.03.016.

18 E. Boros, K. Elbassioni, M. Fouz, V. Gurvich, K. Makino, and B. Manthey. Stochastic mean payoff games: smoothed analysis and approximation schemes. In *Proceedings of the 38th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 6755 of *Lecture Notes in Comput. Sci.*, pages 147–158. Springer, 2011. doi:10.1007/978-3-642-22006-7_13.

19 Endre Boros, Khaled Elbassioni, Mahmoud Fouz, Vladimir Gurvich, Kazuhisa Makino, and Bodo Manthey. Approximation schemes for stochastic mean payoff games with perfect information and few random positions. *Algorithmica*, 80:3132–3157, 2018.

20 Peter Bürgisser and Felipe Cucker. *Condition: The geometry of numerical algorithms*, volume 349. Springer Science & Business Media, 2013.

21 Jakub Chaloupka. Parallel algorithms for mean-payoff games: An experimental evaluation. In *European Symposium on Algorithms*, pages 599–610. Springer, 2009.

22 K. Chatterjee, M. Henzinger, S. Krinninger, and D. Nanongkai. Polynomial-time algorithms for energy games with special weight structures. *Algorithmica*, 70(3):457–492, 2014. doi:10.1007/s00453-013-9843-7.

23 K. Chatterjee and R. Ibsen-Jensen. The complexity of ergodic mean-payoff games. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 8573 of *Lecture Notes in Comput. Sci.*, pages 122–133. Springer, 2014. doi:10.1007/978-3-662-43951-7_11.

24 Miranda Christ and Mihalis Yannakakis. The smoothed complexity of policy iteration for Markov decision processes. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1890–1903, 2023.

25 J. Cochet-Terrasson, S. Gaubert, and J. Gunawardena. A constructive fixed point theorem for min-max functions. *Dyn. Stab. Syst.*, 14(4):407–433, 1999. doi:10.1080/026811199281967.

26 Jean Cochet-Terrasson and Stéphane Gaubert. A policy iteration algorithm for zero-sum stochastic games with mean payoff. *Comptes Rendus Mathematique*, 343(5):377–382, 2006.

27 Jean Cochet-Terrasson Guy Cohen, Stephane Gaubert, Michael Mc Gettrick, and Jean-Pierre Quadrat. Numerical computation of spectral elements in max-plus algebra. In *Proceedings of the IFAC Conference on System Structure and Control*, 1998.

28 Daniel Dadush and Sophie Huiberts. A friendly smoothed analysis of the simplex method. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 390–403, 2018.

29 Eric V. Denardo. Contraction mappings in the theory underlying dynamic programming. *Siam Review*, 9(2):165–177, 1967.

30 Eric V. Denardo and Bennett L. Fox. Multichain Markov renewal programs. *SIAM Journal on Applied Mathematics*, 16(3):468–487, 1968.

31 M. Develin and J. Yu. Tropical polytopes and cellular resolutions. *Exp. Math.*, 16(3):277–291, 2007. doi:10.1080/10586458.2007.10129009.

32 Vishesh Dhingra and Stéphane Gaubert. How to solve large scale deterministic games with mean payoff by policy iteration. In *Proceedings of the 1st International Conference on Performance Evaluation Methodologies and Tools (ValueTools)*, pages 12–es, 2006. doi:10.1145/1190095.1190110.

33 Y. Disser and N. Mosis. A unified worst case for classical simplex and policy iteration pivot rules. In *Proceedings of the 34th International Symposium on Algorithms and Computation (ISAAC)*, pages 27:1–27:17, 2023.

34 John Fearnley, Paul Goldberg, Alexandros Hollender, and Rahul Savani. The complexity of gradient descent: $\mathsf{CLS} = \mathsf{PPAD} \cap \mathsf{PLS}$. *Journal of the ACM*, 70(1):1–74, 2022.

35 John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *Journal of Computer and System Sciences*, 114:1–35, 2020.

36 Uriel Feige. Relations between average case complexity and approximation complexity. In *Proceedings of the 34th annual ACM Symposium on Theory of Computing (STOC)*, pages 534–543, 2002.

37 J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer, New York, 2007. doi:10.1007/978-1-4612-4054-9.

38 Oliver Friedmann. *Exponential lower bounds for solving infinitary payoff games and linear programs*. PhD thesis, Ludwig Maximilian University of Munich, 2011.

39 Alan Frieze and Gregory B. Sorkin. The probabilistic relationship between the assignment and asymmetric traveling salesman problems. *SIAM Journal on Computing*, 36(5):1435–1452, 2007.

40 S. Gaubert and J. Gunawardena. The duality theorem for min-max functions. *C. R. Acad. Sci.*, 326(1):43–48, 1998. doi:10.1016/S0764-4442(97)82710-3.

41 Loukas Georgiadis, Andrew V Goldberg, Robert E Tarjan, and Renato F Werneck. An experimental study of minimum mean cycle algorithms. In *2009 Proceedings of the Eleventh Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 1–13. SIAM, 2009.

42 D. Gillette. Stochastic games with zero stop probabilities. In M. Dresher, A. W. Tucker, and P. Wolfe, editors, *Contributions to the Theory of Games III*, volume 39 of *Ann. of Math. Stud.*, pages 179–188. Princeton University Press, Princeton, NJ, 1957.

43 Mika Göös, Alexandros Hollender, Siddhartha Jain, Gilbert Maystre, William Pires, Robert Robere, and Ran Tao. Further collapses in TFNP. In *Proceedings of the 37th Computational Complexity Conference (CCC)*, pages 1–15, 2022.

44 V. A. Gurvich, A. V. Karzanov, and L. G. Khachiyan. Cyclic games and finding minimax mean cycles in digraphs. *Zh. Vychisl. Mat. Mat. Fiz.*, 28(9):1406–1417, 1988. doi:10.1016/0041-5553(88)90012-2.

45 V. A. Gurvich and V. N. Lebedev. A criterion and verification of the ergodicity of cyclic game forms. *Russian Math. Surveys*, 44(1):243–244, 1989. doi:10.1070/RM1989v044n01ABEH002010.

46 N. Halman. Simple stochastic games, parity games, mean payoff games and discounted payoff games are all LP-type problems. *Algorithmica*, 49(1):37–50, 2007. doi:10.1007/s00453-007-0175-3.

47 T. D. Hansen, P. B. Miltersen, and U. Zwick. Strategy iteration is strongly polynomial for 2-player turn-based stochastic games with a constant discount factor. *J. ACM*, 60(1):1–16, 2013. doi:10.1145/2432622.2432623.

48 T. D. Hansen and U. Zwick. An improved version of the Random-Facet pivoting rule for the simplex algorithm. In *Proceedings of the 47th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 209–218. ACM, 2015. doi:10.1145/2746539.2746557.

49 Thomas Dueholm Hansen and Uri Zwick. Lower bounds for Howard's algorithm for finding minimum mean-cost cycles. In *International Symposium on Algorithms and Computation*, pages 415–426. Springer, 2010.

50 A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Manag. Sci.*, 12(5):359–370, 1966. doi:10.1287/mnsc.12.5.359.

51 A. Hordijk and A. A. Yushkevich. Blackwell optimality. In E. A. Feinberg and A. Shwartz, editors, *Handbook of Markov Decision Processes: Methods and Applications*, volume 40 of *Internat. Ser. Oper. Res. Management Sci.*, pages 231–267. Springer, Boston, MA, 2002. doi:10.1007/978-1-4615-0805-2_8.

52 Ronald A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.

53 Pavel Hubácek and Eylon Yogev. Hardness of continuous local search: Query complexity and cryptographic lower bounds. *SIAM Journal on Computing*, 49(6):1128–1172, 2020.

54 R. Ibsen-Jensen and P. B. Miltersen. Solving simple stochastic games with few coin toss positions. In *Proceedings of the 20th Annual European Symposium on Algorithms (ESA)*, volume 7501 of *Lecture Notes in Comput. Sci.*, pages 636–647. Springer, 2012. doi:10.1007/978-3-642-33090-2_55.

55 David S Johnson, Christos H Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of computer and system sciences*, 37(1):79–100, 1988.

56 M. Jurdziński. Deciding the winner in parity games is in UP ∩ co-UP. *Inform. Process. Lett.*, 68(3):119–124, 1998. doi:10.1016/S0020-0190(98)00150-1.

57 L. Kallenberg. Finite state and action MDPs. In E. A. Feinberg and A. Shwartz, editors, *Handbook of Markov Decision Processes: Methods and Applications*, volume 40 of *Internat. Ser. Oper. Res. Management Sci.*, pages 21–87. Springer, Boston, MA, 2002. doi:10.1007/978-1-4615-0805-2_2.

58 Ricardo David Katz. Max-plus (A, B)-invariant spaces and control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, 52(2):229–241, 2007.

59 Jan Křetínskỳ and Tobias Meggendorfer. Efficient strategy iteration for mean payoff in Markov decision processes. In *International Symposium on Automated Technology for Verification and Analysis*, pages 380–399. Springer, 2017.

60 T. M. Liggett and S. A. Lippman. Stochastic games with perfect information and time average payoff. *SIAM Rev.*, 11(4):604–607, 1969. doi:10.1137/1011093.

61 C. Mathieu and D. B. Wilson. The min mean-weight cycle in a random network. *Combin. Probab. Comput.*, 22(5):763–782, 2013. doi:10.1017/S0963548313000229.

62 Nimrod Megiddo and Christos H Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991.

63 J.-F. Mertens and A. Neyman. Stochastic games. *Internat. J. Game Theory*, 10(2):53–66, 1981. doi:10.1007/BF01769259.

64 Rolf H. Möhring, Martin Skutella, and Frederik Stork. Scheduling with AND/OR precedence constraints. *SIAM Journal on Computing*, 33(2):393–415, 2004.

65 Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the 19th annual ACM Symposium on Theory of Computing (STOC)*, pages 345–354, 1987.

66 Christos H Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and system Sciences*, 48(3):498–532, 1994.

67 Anuj Puri. *Theory of hybrid systems and discrete event systems*. University of California at Berkeley, 1995.

68 M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Ser. Probab. Stat. Wiley, Hoboken, NJ, 2005.

69 T.E.S. Raghavan and Zamir Syed. A policy-improvement type algorithm for solving zero-sum two-person stochastic games of perfect information. *Mathematical Programming*, 95(3):513–532, 2003.

70 S. S. Rao, R. Chandrasekaran, and K.P.K. Nair. Algorithms for discounted stochastic games. *Journal of Optimization Theory and Applications*, 11(6):627–637, 1973.

71 H. Röglin and B. Vöcking. Smoothed analysis of integer programming. *Math. Program.*, 110(1):21–56, 2007. doi:10.1007/s10107-006-0055-7.

72 Steven Rudich. Super-bits, demi-bits, and NP/qpoly-natural proofs. In *Proceedings of the International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM/APPROX)*, pages 85–93, 1997.

73 Sven Schewe. From parity and payoff games to linear programming. In *Proceedings of the 34th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 675–686, 2009.

74 Bart Selman, David G Mitchell, and Hector J Levesque. Generating hard satisfiability problems. *Artificial intelligence*, 81(1-2):17–29, 1996.

75 L. S. Shapley. Stochastic games. *Proc. Natl. Acad. Sci. USA*, 39(10):1095–1100, 1953. doi:10.1073/pnas.39.10.1095.

76 Aaron Sidford, Mengdi Wang, Xian Wu, and Yinyu Ye. Variance reduced value iteration and faster algorithms for solving Markov decision processes. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 770–787, 2018.

77 Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, 2004.

78 Y. Ye. The simplex and policy-iteration methods are strongly polynomial for the Markov decision problem with a fixed discount rate. *Math. Oper. Res.*, 36(4):593–603, 2011.

79 U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoret. Comput. Sci.*, 158(1–2):343–359, 1996. doi:10.1016/0304-3975(95)00188-3.

An Order out of Nowhere: A New Algorithm for Infinite-Domain CSPs

Antoine Mottet
Research Group on Theoretical Computer Science, Hamburg University of Technology, Germany

Tomáš Nagy
Theoretical Computer Science Department, Jagiellonian University, Kraków, Poland

Michael Pinsker
Institut für Diskrete Mathematik und Geometrie, Technische Universität Wien, Austria

Abstract

We consider the problem of satisfiability of sets of constraints in a given set of finite uniform hypergraphs. While the problem under consideration is similar in nature to the problem of satisfiability of constraints in graphs, the classical complexity reduction to finite-domain CSPs that was used in the proof of the complexity dichotomy for such problems cannot be used as a black box in our case. We therefore introduce an algorithmic technique inspired by classical notions from the theory of finite-domain CSPs, and prove its correctness based on symmetries that depend on a linear order that is external to the structures under consideration. Our second main result is a P/NP-complete complexity dichotomy for such problems over many sets of uniform hypergraphs. The proof is based on the translation of the problem into the framework of constraint satisfaction problems (CSPs) over infinite uniform hypergraphs. Our result confirms in particular the Bodirsky-Pinsker conjecture for CSPs of first-order reducts of some homogeneous hypergraphs. This forms a vast generalization of previous work by Bodirsky-Pinsker (STOC'11) and Bodirsky-Martin-Pinsker-Pongrácz (ICALP'16) on graph satisfiability.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases Constraint Satisfaction Problems, Hypergraphs, Polymorphisms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.148

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://doi.org/10.48550/arXiv.2301.12977` [29]

Funding *Tomáš Nagy*: This research was funded in whole or in part by the Austrian Science Fund (FWF) [P 32337, I 5948]. This research was funded in whole or in part by National Science Centre, Poland 2021/03/Y/ST6/00171. For the purpose of Open Access, the authors have applied a CC-BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission.
Michael Pinsker: This research was funded in whole or in part by the Austrian Science Fund (FWF) [P 32337, I 5948]. For the purpose of Open Access, the authors have applied a CC BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission. This research is also funded by the European Union (ERC, POCOCOP, 101071674). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

1 Introduction

In [15], Bodirsky and the third author introduced the computational problem Graph-SAT as a generalization of systematic restrictions of the Boolean satisfiability problem studied by Schaefer [36]. A *graph formula* is a formula formed from the atomic formulas $E(x, y)$ and $x = y$ using negation, conjunction and disjunction, where E is interpreted as the edge

relation of a simple undirected graph. Given a finite set Ψ of graph formulas, the *graph satisfiability problem* Graph-SAT(Ψ) gets as an input a finite set $\mathcal{V}$ of variables and a graph formula $\Phi = \phi_1 \wedge \cdots \wedge \phi_n$, where every ϕ_i is obtained from a formula $\psi \in \Psi$ by substituting the variables of ψ by variables from $\mathcal{V}$; the goal is to decide the existence of a graph satisfying Φ. Any instance of a Boolean satisfiability problem can indeed easily be reduced to a problem of this form, roughly by replacing Boolean variables by pairs of variables which are to be assigned vertices in a graph, and by translating the potential Boolean values 0 and 1 into the non-existence or existence of an edge between these two variables. The main result of [15] states that this computational problem is either solvable in polynomial time or is NP-complete. This can be put in contrast with the theorem of Ladner [28] stating that if P $\neq$ NP, then there exist computational problems that are neither solvable in polynomial-time nor NP-complete. Similar *dichotomy theorems* have been established for related problems concerning the satisfaction of constraints by linear orders [8], partially ordered sets [26], tournaments [30], or phylogenetic trees [6]. It is conjectured that such dichotomies defying Ladner's theorem are common; we refer to Section 1.2 and Conjecture 4 for a precise statement.

In order to develop our understanding of such natural generalizations of the classical Boolean satisfiability problem, we consider in this article the complexity of Graph-SAT where graph formulas are replaced by *ℓ-hypergraph formulas* for some fixed $\ell \geq 2$. More precisely, we consider formulas where E is an ℓ-ary symbol denoting the edge relation of an ℓ-uniform undirected hypergraph; in the following, since all our hypergraphs are uniform and undirected, we simply write *ℓ-hypergraph*. The problem ℓ-Hypergraph-SAT is then defined in the same way as the problem Graph-SAT above. We also study the complexity of the natural variant of the ℓ-Hypergraph-SAT problem, investigated in [12] for the special case of graphs, where we ask for the existence of a satisfying hypergraph that belongs to a prescribed set $\mathcal{K}$ of finite ℓ-hypergraphs. This corresponds to imposing structural restrictions on the possible satisfying hypergraph solutions. For example, it is natural to ask for the existence of a solution in the class $\mathcal{K}_r^\ell$ of all finite ℓ-hypergraphs omitting a generalized clique on r vertices. We use the notation ℓ-Hypergraph-SAT($\Psi, \mathcal{K}$) to denote this problem.

Surprisingly, it turns out that ℓ-hypergraph problems behave very differently from the corresponding graph problems, requiring in particular genuinely novel algorithmic methods to handle them. A natural attempt to solve hypergraph satisfiability in polynomial time is to use a generic reduction to constraint satisfaction problems (CSPs) whose domain consists of the hypergraphs with at most ℓ elements [13]. While this reduction always works in the (graph) case of $\ell = 2$ [30], it can happen for $\ell > 2$ that the resulting finite-domain CSP is an NP-complete problem, although the original hypergraph satisfiability problem is solvable in polynomial time. Our main result, Theorem 8, is an algorithm running in polynomial time and solving the ℓ-Hypergraph-SAT($\Psi, \mathcal{K}$) problem under some general algebraic assumptions.

The next example illustrates that in this setting our algorithm is strictly more powerful than the reduction of [13].

▶ **Example 1.** Let $\ell = 3$, let ψ be a formula with 4 free variables that holds precisely for the hypergraphs in Figure 1, and let Ψ be the set consisting of ψ. This is an example where the reduction to the finite from [13] cannot be applied to prove the tractability of ℓ-Hypergraph-SAT(Ψ). However, the problem is solvable in polynomial time, as it can be solved by the algorithm introduced in Section 3.

Building on Theorem 8, our second contribution is a full complexity dichotomy for the problems ℓ-Hypergraph-SAT($\Psi, \mathcal{K}$) where $\mathcal{K}$ is the class $\mathcal{K}_{\text{all}}^\ell$ of all finite ℓ-hypergraphs, or $\mathcal{K}_r^\ell$.

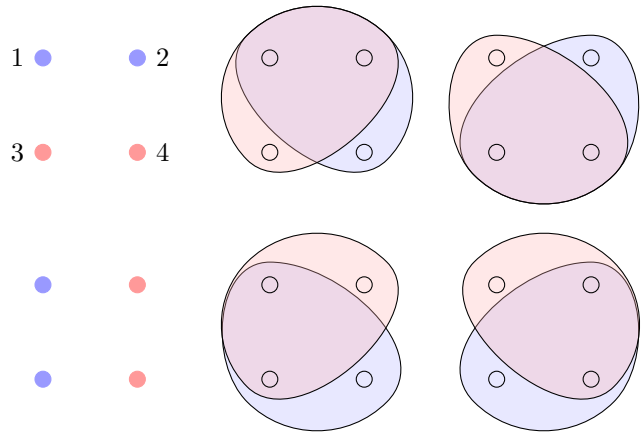


Figure 1 Hypergraphs on at most 4 elements satisfying ψ in Example 1. The vertices are labeled to represent each of the four free variables of ψ (only shown on one of the hypergraphs for readability). The two 3-hypergraphs on the left have 2 vertices, and the color coding denotes vertices that are equal. The other four 3-hypergraphs have four vertices each and precisely two hyperedges.

▶ **Theorem 2.** *Let $\ell \geq 3$, let $\mathcal{K}$ be either the class $\mathcal{K}^{\ell}_{\text{all}}$ of all finite ℓ-hypergraphs or the class $\mathcal{K}^{\ell}_{r}$ for some $r > \ell$, and let Ψ be a set of ℓ-hypergraph formulas. Then ℓ-Hypergraph-SAT$(\Psi, \mathcal{K})$ is either in P, or it is NP-complete. Moreover, given Ψ, one can algorithmically decide which of the cases holds.*

In fact, our polynomial-time algorithm in Theorem 8 solves the hypergraph satisfiability problem for all classes $\mathcal{K}$ of hypergraphs satisfying certain assumptions that we introduce in Section 3. Likewise, our results imply a dichotomy result as in Theorem 2 for every class $\mathcal{K}$ satisfying certain structural assumptions. For more details, see the full version of the article [29].

1.1 Connection to Constraint Satisfaction Problems

The *constraint satisfaction problem* with template $\mathbb{A} = (A; R_1, \dots, R_n)$ is the computational problem CSP($\mathbb{A}$) of deciding, given an instance with variables $\mathcal{V}$ and constraints $\phi(x_{i_1}, \dots, x_{i_r})$ with $\phi \in \{R_1, \dots, R_n\}$ and $x_{i_1}, \dots, x_{i_r} \in \mathcal{V}$, whether there exists an assignment $f\colon \mathcal{V} \to A$ that satisfies all the constraints.

Note how the problem ℓ-Hypergraph-SAT(Ψ) is similar in nature to a constraint satisfaction problem, where the difference lies in the fact that we are not asking for a labelling of variables to elements of a structure $\mathbb{A}$, but rather for a consistent labelling of ℓ-tuples of variables to a finite set describing all the possible ℓ-hypergraphs on at most ℓ elements. For example, in the case of $\ell = 2$, this set contains 3 elements (for the graph on a single vertex, and the two undirected graphs on 2 vertices), while for $\ell = 3$ this set contains 6 elements (there is one labeled 3-hypergraph on a single element, three on 2 elements, and two on 3 elements).

It was already noticed in [15] that it is possible to design a structure $\mathbb{A}$ (which is necessarily infinite) such that Graph-SAT(Ψ) is equivalent to CSP($\mathbb{A}$), and this observation also carries out to the hypergraph setting as follows.

Fix $\ell \geq 3$ and a class $\mathcal{K}$ of finite ℓ-hypergraphs. Let us assume that $\mathcal{K}$ is an *amalgamation class*: an isomorphism-closed class that is closed under induced sub-hypergraphs and with the property that for any two hypergraphs $\mathbb{H}_1, \mathbb{H}_2 \in \mathcal{K}$ having a common hypergraph $\mathbb{H}$ as

intersection, there exists $\mathbb{H}' \in \mathcal{K}$ and embeddings of $\mathbb{H}_1, \mathbb{H}_2$ into $\mathbb{H}'$ which agree on $\mathbb{H}$. A classical result of Fraïssé [22] yields that there exists an infinite limit hypergraph $\mathbb{H}_\mathcal{K}$, called the *Fraïssé limit of* $\mathcal{K}$, with the property that the finite induced sub-hypergraphs of $\mathbb{H}_\mathcal{K}$ are precisely the hypergraphs in $\mathcal{K}$. Moreover, this limit can be taken to be *homogeneous*, i.e., highly symmetric in a certain precise sense – see Section 2 for precise definitions of these concepts. If Ψ is a set of ℓ-hypergraph formulas, then it defines in $\mathbb{H}_\mathcal{K}$ a set of relations that one can view as a CSP template $\mathbb{A}_{\mathcal{K},\Psi}$. It follows that the problem ℓ-Hypergraph-SAT$(\mathcal{K}, \Psi)$ is precisely the same as CSP$(\mathbb{A}_{\mathcal{K},\Psi})$. The assumption that $\mathcal{K}$ is an amalgamation class is rather mild and is for example fulfilled by the classes of interest for Theorem 2, namely by the class $\mathcal{K}^\ell_{\text{all}}$ of all finite ℓ-hypergraphs, or for any $\ell < r$ by the class $\mathcal{K}^\ell_r$. We refer to [1] for a discussion of amalgamation classes of 3-hypergraphs; importantly, while in the case of $\ell = 2$ all such classes are known [27], it seems very difficult to obtain a similar classification in general since there are uncountably-many such classes already for $\ell = 3$. The latter fact obliges us to build on and refine abstract methods rather than relying on the comfort of a classification in our general dichotomy result, which contrasts with the approach for graphs in [12].

Using the reformulation of ℓ-hypergraph problems as constraint satisfaction problems, we show that the border between tractability and NP-hardness in Theorem 2 can be described algebraically by properties of the *polymorphisms* of the structures $\mathbb{A}_{\mathcal{K},\Psi}$, i.e., by the functions preserving all relations of $\mathbb{A}_{\mathcal{K},\Psi}$. This implies, in particular, the above-mentioned decidability of this border. Roughly speaking, the tractable case corresponds to the CSP template enjoying some non-trivial algebraic invariants in the form of polymorphisms, whereas the hard case is characterized precisely by the absence of such invariants.

▶ **Theorem 3.** *Let $\ell \geq 3$, let $\mathcal{K}$ be either the class $\mathcal{K}^\ell_{\text{all}}$ of all finite ℓ-hypergraphs or the class $\mathcal{K}^\ell_r$ for some $r > \ell$, and let Ψ be a set of ℓ-hypergraph formulas. Then precisely one of the following applies.*

1. *The clone of polymorphisms of $\mathbb{A}_{\mathcal{K},\Psi}$ has no uniformly continuous minion homomorphism to the clone of projections $\mathscr{P}$, and* CSP$(\mathbb{A}_{\mathcal{K},\Psi})$ *is in* P.
2. *The clone of polymorphisms of $\mathbb{A}_{\mathcal{K},\Psi}$ has a uniformly continuous minion homomorphism to the clone of projections $\mathscr{P}$, and* CSP$(\mathbb{A}_{\mathcal{K},\Psi})$ *is* NP*-complete.*

The algebraic assumptions in the second item of the theorem correspond to the clone of polymorphisms being trivial in a certain sense (i.e., containing only polymorphisms that imitate the behaviour of projections when restricted to a certain set). For the precise definitions, see [5].

1.2 Related work on constraint satisfaction problems

In the framework of CSPs, it is natural to consider not only classes of finite ℓ-hypergraphs but also classes of different finite structures in a fixed relational signature. If such a class $\mathcal{K}$ is an amalgamation class, then there exists a countably infinite homogeneous structure $\mathbb{B}_\mathcal{K}$ whose finite substructures are precisely the structures in $\mathcal{K}$. However, CSP$(\mathbb{B}_\mathcal{K})$ is not guaranteed to be contained in the complexity class NP since the class $\mathcal{K}$ does not have to be algorithmically enumerable (as mentioned above, there are uncountably many amalgamation classes of 3-hypergraphs; hence there exists such a $\mathcal{K}$ such that CSP$(\mathbb{B}_\mathcal{K})$ is undecidable). A natural way of achieving the algorithmical enumerability of $\mathcal{K}$ is to require that there exists a natural number $b_\mathcal{K}$ such that a structure is contained in $\mathcal{K}$ if, and only if, all its substructures of size at most $b_\mathcal{K}$ are in $\mathcal{K}$. In this case, we say that $\mathcal{K}$ (or its Fraïssé limit $\mathbb{B}_\mathcal{K}$) is *finitely bounded*. For every set Ψ of formulas in the language of the structures at hand, one then

gets as in the previous section a structure $\mathbb{A}_{\mathcal{K},\Psi}$ whose domain is the same as $\mathbb{B}_{\mathcal{K}}$ and whose relations are definable in first-order logic from the relations of $\mathbb{B}_{\mathcal{K}}$ – we say that $\mathbb{A}_{\mathcal{K},\Psi}$ is a *first-order reduct* of $\mathbb{B}_{\mathcal{K}}$.

Thus, for every set Ψ of formulas and every finitely bounded amalgamation class $\mathcal{K}$, the generalized satisfiability problem parameterized by Ψ and $\mathcal{K}$ is the CSP of a first-order reduct $\mathbb{A}_{\mathcal{K},\Psi}$ of a finitely bounded homogeneous structure. It is known that the complexity of the CSP over any such template depends solely on the polymorphisms of $\mathbb{A}_{\mathcal{K},\Psi}$ [16]. This motivates the following conjecture generalizing the dichotomy for Graph-SAT which was formulated by Bodirsky and Pinsker in 2011 (see [17]). The modern formulation of the conjecture based on recent progress [2, 3, 5] is the following:

▶ **Conjecture 4.** *Let $\mathbb{A}$ be a CSP template which is a first-order reduct of a finitely bounded homogeneous structure. Then one of the following applies.*

1. *The clone of polymorphisms of $\mathbb{A}$ has no uniformly continuous minion homomorphism to the clone of projections $\mathscr{P}$, and* CSP($\mathbb{A}$) *is in* P.
2. *The clone of polymorphisms of $\mathbb{A}$ has a uniformly continuous minion homomorphism to the clone of projections $\mathscr{P}$, and* CSP($\mathbb{A}$) *is* NP*-complete.*

It follows that Theorem 3 is a special case of Conjecture 4. It is known that if the clone of polymorphisms of any CSP template within the range of Conjecture 4 has a uniformly continuous minion homomorphism to $\mathscr{P}$, then the CSP of such template is NP-hard [5]. Already before Conjecture 4 was introduced, a similar conjecture was formulated by Feder and Vardi [23] for CSPs over templates with finite domains and it was confirmed independently by Bulatov and Zhuk [19, 39, 40] recently. Conjecture 4 itself has been confirmed for many subclasses: for example for CSPs of all structures first-order definable in finitely bounded homogeneous graphs [15, 12], in $(\mathbb{Q};<)$ [8], in any unary structure [14], in the random poset [26], in the random tournament [30], or in the homogeneous branching C-relation [6], in ω-categorical monadically stable structures [18], as well as for all CSPs in the class MMSNP [11], and for CSPs of representations of some relational algebras [9, 10].

1.3 Novelty of the methods and significance of the results

We prove that under the algebraic assumption in item (1) of Theorem 3, $\mathbb{A}_{\mathcal{K},\Psi}$ admits non-trivial symmetries that can be seen as operations acting on the set of *linearly ordered* ℓ-hypergraphs with at most ℓ elements. We moreover know from [34] that the introduction of a linear order "out of nowhere" is unavoidable, in the sense that the symmetries of $\mathbb{A}_{\mathcal{K},\Psi}$ acting on unordered ℓ-hypergraphs can be trivial even if CSP($\mathbb{A}_{\mathcal{K},\Psi}$) is solvable in polynomial time. This is rather surprising (in model-theoretic terms, hypergraphs form a class having the *non-strict order property*, and thus have no ability to encode linear orders) and is to date the only example of such a phenomenon. As a consequence, the aforementioned "reduction to the finite" introduced in [13], which is enough to prove the tractability part of most of the complexity dichotomies mentioned in the previous section, cannot be used in the hypergraph satisfiability setting. In order to prove the tractability part of Theorem 3, we thus introduce new algorithmic techniques inspired by results in the theory of constraint satisfaction problems with *finite* domains, in particular by absorption theory [4] and Zhuk's theory [39, 40]. More precisely, let $\mathcal{I}$ be an instance of CSP($\mathbb{A}_{\mathcal{K},\Psi}$). Our algorithm transforms $\mathcal{I}$ into an equi-satisfiable instance $\mathcal{I}'$ that is sufficiently locally consistent, such that the solution set of a certain relaxation of $\mathcal{I}'$ does not imply any restrictions on the solution set of the whole instance, and that satisfies an additional condition resembling Zhuk's notion of irreducibility [39, 40]. We then prove that any non-trivial instance satisfying those

properties has an injective solution. This step is to be compared with the case of *absorbing reductions* in Zhuk's algorithm. The existence of an injective solution can then be checked by the aforementioned reduction to the finite from [13, 14]. In this way, we resolve the trouble with hitherto standard methods pointed out in [34]. A positive resolution of the general Conjecture 4 will likely have to proceed in a similar spirit, albeit at a yet higher level of sophistication. In the case of graphs, the above described algorithm is not necessary since every instance can be immediately reduced to a finite-domain CSP by the black box reduction.

We use the recently developed theory of smooth approximations [30] to prove the dichotomy, i.e., that $\mathbb{A}_{\mathcal{K},\Psi}$ satisfies one of the two items of Theorem 3, for all Ψ. The classification of the complexity of graph-satisfiability problems from [15] used a demanding case distinction over the possible automorphisms groups of the structures $\mathbb{A}_{\mathcal{K},\Psi}$ (where Ψ is a set of graph formulas, and $\mathcal{K}$ is the class of all finite simple undirected graphs) – it was known previously that there are exactly 5 such groups [37]. Our result relies neither on such a classification of the automorphism groups of the structures under consideration, nor on the classification of the hypergraphs of which they are first-order reducts; as mentioned above, no such classification is available. While Thomas [38] obtained a classification of the mentioned automorphism groups for every fixed ℓ and for $\mathcal{K}$ consisting of all finite ℓ-hypergraphs, this number grows with ℓ and makes an exhaustive case distinction impossible. To overcome the absence of such classifications, we rely on the scalability of the theory of smooth approximations, i.e., on the fact that the main results of the theory can be used without knowing the base structures under consideration. This was claimed to be one of the main contributions of this theory; Theorem 3 and its generalization [29, Theorem 21] are the first complexity classification using smooth approximations that truly exemplifies this promise.

1.4 Bonus track: local consistency

Our structural analysis of ℓ-hypergraph problems allows us to obtain as an easy consequence a description of the hypergraph satisfiability problems ℓ-Hypergraph-SAT$(\Psi, \mathcal{K})$ that are solvable by local consistency methods, assuming that Ψ contains the atomic formula E. Similar classifications, and general results on the amount of local consistency needed in those cases, had previously obtained for various other problems (including Graph-SAT problems) which can be modeled as CSPs of first-order reducts of finitely bounded homogeneous structures [31, 30].

▶ **Theorem 5.** *Let $r > \ell \geq 3$, let $\mathcal{K}$ be either the class $\mathcal{K}^{\ell}_{\text{all}}$ of all finite ℓ-hypergraphs or the class $\mathcal{K}^{\ell}_{r}$, and let Ψ be a set of ℓ-hypergraph formulas containing $E(x_1, \dots, x_\ell)$. Then precisely one of the following applies.*

1. *The clone* $\text{Pol}(\mathbb{A})$ *has no uniformly continuous minion homomorphism to the clone of affine maps over a finite module, and* $\text{CSP}(\mathbb{A})$ *has relational width* $(2\ell, \max(3\ell, r))$.
2. *The clone* $\text{Pol}(\mathbb{A})$ *has a uniformly continuous minion homomorphism to the clone of affine maps over a finite module.*

1.5 Future work

This work is concerned with the complexity of the decision version of constraint satisfaction problems whose study is motivated by Conjecture 4. A natural variant of such problems is the optimisation version, where one is interested in finding a solution to an instance of the CSP that minimizes the number of unsatisfied constraints. The complexity of such problems (called MinCSPs) has mostly been investigated for finite templates, but recently also in the

case of infinite templates falling within the scope of Conjecture 4 from the point of view of exact optimisation and approximation [21, 20, 24], as well as from the point of view of parameterized complexity [33].

Our complexity classification for the decision CSP (Theorem 3) can be seen as a foundation for a systematic structural study of optimisation problems over hypergraphs.

1.6 Organisation of the present article

After introducing a few notions needed for the formulation of the main algorithm in Section 2, we introduce the algorithm and prove its correctness in Section 3. For lack of space, we only present the proof of one of the main correctness arguments (Theorem 13) for the algorithmic part of Theorem 3 and otherwise illustrate the main concepts that we introduce using examples. In Section 4, we give an outline of the proof of Theorem 3. The rest of the proofs can be found in the appendix.

2 Preliminaries

For any $k \geq 1$, we write $[k]$ to denote the set $\{1, \dots, k\}$. A tuple is called *injective* if its entries are pairwise distinct. In the entire article, we consider only relational structures in a finite signature.

A *primitive-positive* (pp-)formula is a first-order formula built only from atomic formulas, existential quantification, and conjunction. A relation $R \subseteq A^n$ is *pp-definable* in a relational structure $\mathbb{A}$ if there exists a pp-formula $\phi(x_1, \dots, x_n)$ such that the tuples in R are precisely the tuples satisfying ϕ.

Let $\ell \geq 2$. A structure $\mathbb{H} = (H; E)$ is an *ℓ-hypergraph* if the relation E is of arity ℓ, contains only injective tuples (called *hyperedges*), and is *fully symmetric*, i.e., every tuple obtained by permuting the components of a hyperedge is a hyperedge as well. Given any ℓ-hypergraph $\mathbb{H} = (H; E)$, we write N for the set of all injective ℓ-tuples in H that are not hyperedges, and we call this set the *non-hyperedge* relation.

2.1 CSPs and Relational Width

A *CSP instance* over a set A is a pair $\mathcal{I} = (\mathcal{V}, \mathcal{C})$, where $\mathcal{V}$ is a non-empty finite set of variables, and $\mathcal{C}$ is a set of *constraints*; each constraint $C \in \mathcal{C}$ is a subset of A^U for some non-empty $U \subseteq \mathcal{V}$ (U is called the *scope* of C). For a relational structure $\mathbb{A}$, we say that $\mathcal{I}$ is an *instance of* $\mathrm{CSP}(\mathbb{A})$ if for every $C \in \mathcal{C}$ with scope U, there exists an enumeration $u_1, \dots, u_k$ of the elements of U and a k-ary relation R of $\mathbb{A}$ such that for all $f \colon U \to A$ we have $f \in C \Leftrightarrow (f(u_1), \dots, f(u_k)) \in R$. A mapping $s \colon \mathcal{V} \to A$ is a *solution* of the instance $\mathcal{I}$ if we have $s|_U \in C$ for every $C \in \mathcal{C}$ with scope U. Given a constraint $C \subseteq A^U$ and a tuple $\mathbf{v} \in U^k$ for some $k \geq 1$, the *projection of C onto* $\mathbf{v}$ is defined by $\mathrm{proj}_{\mathbf{v}}(C) := \{f(\mathbf{v}) \colon f \in C\}$. Let $U \subseteq \mathcal{V}$. We define the *restriction* of $\mathcal{I}$ to U to be an instance $\mathcal{I}|_U = (U, \mathcal{C}|_U)$ where the set of constraints $\mathcal{C}|_U$ contains for every $C \in \mathcal{C}$ the constraint $C|_U = \{g|_U \mid g \in C\}$.

We denote by $\mathrm{CSP}_{\mathrm{Inj}}(\mathbb{A})$ the restriction of $\mathrm{CSP}(\mathbb{A})$ to those instances of $\mathrm{CSP}(\mathbb{A})$ where for every constraint C and for every pair of distinct variables u, v in its scope, $\mathrm{proj}_{(u,v)}(C) \subseteq \{(a, b) \in A^2 \mid a \neq b\}$.

▶ **Definition 6.** *Let $1 \leq m \leq n$. We say that an instance $\mathcal{I} = (\mathcal{V}, \mathcal{C})$ is* (m, n)-minimal *if both of the following hold:*

- *every non-empty subset of at most n variables in $\mathcal{V}$ is contained in the scope of some constraint in $\mathcal{I}$;*
- *for every at most m-element tuple of variables $\mathbf{v}$ and any two constraints $C_1, C_2 \in \mathcal{C}$ whose scopes contain all variables of $\mathbf{v}$, the projections of C_1 and C_2 onto $\mathbf{v}$ coincide.*

For $m \geq 1$, we say that an instance is *m-minimal* if it is (m, m)-minimal. We say that an instance $\mathcal{I}$ of the CSP is *non-trivial* if it does not contain any empty constraint. Otherwise, $\mathcal{I}$ is *trivial*.

For all $1 \leq m \leq n$ and for every instance $\mathcal{I}$ of a CSP($\mathbb{A}$) for some finite-domain structure $\mathbb{A}$, an (m, n)-minimal instance with the same solution set as $\mathcal{I}$ can be computed from $\mathcal{I}$ in polynomial time. The same holds for any ω-categorical structure $\mathbb{A}$ (see Section 2.2 for the definition of ω-categoricity, and see e.g., Section 2.3 in [32] for a description of the (m, n)-minimality algorithm in this setting). The resulting instance $\mathcal{I}'$ is called the *(m, n)-minimal instance equivalent to $\mathcal{I}$* and the algorithm that computes this instance is called the *(m, n)-minimality algorithm*. Note that the instance $\mathcal{I}'$ is not necessarily an instance of CSP($\mathbb{A}$). However, $\mathcal{I}'$ is an instance of CSP($\mathbb{A}'$) where $\mathbb{A}'$ is the expansion of $\mathbb{A}$ by all at most n-ary relations pp-definable in $\mathbb{A}$. Moreover, CSP($\mathbb{A}'$) has the same complexity as CSP($\mathbb{A}$).

If $\mathcal{I}$ is m-minimal and $\mathbf{v}$ is a tuple of variables of length at most m, then by definition there exists a constraint of $\mathcal{I}$ whose scope contains all variables in $\mathbf{v}$, and all the constraints who do have the same projection on $\mathbf{v}$. We write $\mathrm{proj}_{\mathbf{v}}(\mathcal{I})$ for this projection, and call it the *projection of $\mathcal{I}$ onto $\mathbf{v}$*.

▶ **Definition 7.** *Let $1 \leq m \leq n$, and let $\mathbb{A}$ be a relational structure. We say that* CSP($\mathbb{A}$) *has relational width (m, n) if every non-trivial (m, n)-minimal instance equivalent to an instance of* CSP($\mathbb{A}$) *has a solution.* CSP($\mathbb{A}$) *has* bounded width *if it has relational width (m, n) for some natural numbers $m \leq n$.*

2.2 Basic model-theoretic definitions

Let $\mathbb{B}$ and $\mathbb{C}$ be relational structures in the same signature. A *homomorphism* from $\mathbb{B}$ to $\mathbb{C}$ is a mapping $f\colon B \to C$ with the property that for every relational symbol R from the signature of $\mathbb{B}$ and for every $\mathbf{b} \in R^{\mathbb{B}}$, it holds that $f(\mathbf{b}) \in R^{\mathbb{C}}$. An embedding of $\mathbb{B}$ into $\mathbb{C}$ is an injective homomorphism $f\colon \mathbb{B} \to \mathbb{C}$ such that f^{-1} is a homomorphism from the structure induced by the image of f in $\mathbb{C}$ to $\mathbb{B}$, and an *isomorphism* from $\mathbb{B}$ to $\mathbb{C}$ is a bijective embedding of $\mathbb{B}$ into $\mathbb{C}$. An endomorphism of $\mathbb{B}$ is a homomorphism from $\mathbb{B}$ to $\mathbb{B}$, an *automorphism* of $\mathbb{B}$ is an isomorphism from $\mathbb{B}$ to $\mathbb{B}$. We denote the set of endomorphisms of $\mathbb{B}$ by End($\mathbb{B}$) and the set of its automorphisms by Aut($\mathbb{B}$).

Let $\ell \geq 2$, and let $\mathcal{K}$ be an isomorphism-closed class of finite ℓ-hypergraphs. We say that $\mathcal{K}$ is an *amalgamation class* if the following two conditions are satisfied: It is closed under induced substructures, and for any ℓ-hypergraphs $\mathbb{H}, \mathbb{H}_1, \mathbb{H}_2 \in \mathcal{K}$ and for any embeddings f_i of $\mathbb{H}$ into $\mathbb{H}_i$ ($i \in \{1, 2\}$), there exists an ℓ-hypergraph $\mathbb{H}'$ and embeddings g_i of $\mathbb{H}_i$ into $\mathbb{H}'$ ($i \in \{1, 2\}$) such that $g_1 \circ f_1 = g_2 \circ f_2$. We write $\overrightarrow{\mathcal{K}}$ for the class which contains for every ℓ-hypergraph $\mathbb{H}$ from $\mathcal{K}$ all ordered ℓ-hypergraphs obtained by linearly ordering $\mathbb{H}$.

A relational structure $\mathbb{B}$ is *homogeneous* if every isomorphism between finite induced substructures of $\mathbb{B}$ extends to an automorphism of $\mathbb{B}$. The class of finite substructures of a homogeneous structure $\mathbb{B}$ is an amalgamation class; and conversely, for every amalgamation class $\mathcal{K}$ there exists a homogeneous structure $\mathbb{B}_{\mathcal{K}}$ whose finite induced substructures are exactly the structures in $\mathcal{K}$ (see e.g. [25] for this as well as the other claims in this section). The structure $\mathbb{B}_{\mathcal{K}}$ is called the Fraïssé limit of $\mathcal{K}$. The *universal homogeneous ℓ-hypergraph* is the Fraïssé limit of $\mathcal{K}^{\ell}_{\mathrm{all}}$.

A *first-order reduct* of a structure $\mathbb{B}$ is a structure $\mathbb{A}$ on the same domain whose relations are definable over $\mathbb{B}$ by first-order formulas without parameters. Recall that for any amalgamation class $\mathcal{K}$ and for any set Ψ of ℓ-hypergraph formulas, $\mathbb{A}_{\mathcal{K},\Psi}$ denotes the first-order reduct of

the Fraïssé limit $\mathbb{H}_{\mathcal{K}}$ of $\mathcal{K}$ whose relations are defined by the formulas in Ψ. We remark that if $\mathbb{B}$ is the Fraïssé limit of a finitely bounded class, then every first-order formula is equivalent to one without quantifiers.

A countable relational structure is *ω-categorical* if its automorphism group has finitely many orbits in its componentwise action on n-tuples of elements for all $n \geq 1$. This is equivalent to saying that there are only finitely many relations of any fixed arity $n \geq 1$ that are first-order definable from $\mathbb{A}$. Every first-order reduct of a finitely bounded homogeneous structure is ω-categorical.

2.3 Polymorphisms

A *polymorphism* of a relational structure $\mathbb{A}$ is a function from A^n to A for some $n \geq 1$ which *preserves* all relations of $\mathbb{A}$, i.e., for every such relation R of arity m and for all tuples $(a_1^1, \dots, a_m^1), \dots, (a_1^n, \dots, a_m^n) \in R$, it holds that $(f(a_1^1, \dots, a_1^n), \dots, f(a_m^1, \dots, a_m^n)) \in R$. We also say that a polymorphism of $\mathbb{A}$ preserves a constraint $C \subseteq A^U$ if for all $g_1, \dots, g_n \in C$, it holds that $f \circ (g_1, \dots, g_n) \in C$. The set of all polymorphisms of a structure $\mathbb{A}$, denoted by $\mathrm{Pol}(\mathbb{A})$, is a *function clone*, i.e., a set of finitary operations on a fixed set which contains all projections and which is closed under arbitrary compositions. Every relation that is pp-definable in a relational structure $\mathbb{A}$ is preserved by all polymorphisms of $\mathbb{A}$.

Let $S \subseteq R \subseteq A^n$ be relations pp-definable in a structure $\mathbb{A}$. We say that S *is a binary absorbing subuniverse of R in* $\mathbb{A}$ if there exists a binary operation $f \in \mathrm{Pol}(\mathbb{A})$ such that for every $\mathbf{s} \in S, \mathbf{r} \in R$, we have that $f(\mathbf{s}, \mathbf{r}), f(\mathbf{r}, \mathbf{s}) \in S$. In this case, we write $S \trianglelefteq_{\mathbb{A}} R$, and we say that f *witnesses* the binary absorption.

Let $\mathbb{A}$ be a relational structure, and let $\mathscr{G} = \mathrm{Aut}(\mathbb{A})$ be the group of its automorphisms. For $n \geq 1$, a k-ary operation f defined on the domain of $\mathbb{A}$ is *n-canonical* with respect to $\mathbb{A}$ if for all $\mathbf{a}_1, \dots, \mathbf{a}_k \in A^n$ and all $\alpha_1, \dots, \alpha_k \in \mathscr{G}$, there exists $\beta \in \mathscr{G}$ such that $f(\mathbf{a}_1, \dots, \mathbf{a}_k) = \beta \circ f(\alpha_1(\mathbf{a}_1), \dots, \alpha_k(\mathbf{a}_k))$. A function f that is n-canonical with respect to $\mathbb{A}$ for all $n \geq 1$ is called *canonical* with respect to $\mathbb{A}$. In particular, f induces an operation on the set $A^n/\mathscr{G}$ of orbits of n-tuples under $\mathscr{G}$ for every $n \geq 1$. In our setting, we are interested in operations that are canonical with respect to a homogeneous ℓ-hypergraph $\mathbb{H}$ or to a homogeneous linearly ordered ℓ-hypergraph $(\mathbb{H}, <)$. In this case, an operation canonical with respect to $\mathbb{H}$ can simply be seen as an operation on labeled ℓ-hypergraphs with at most n elements, while an operation canonical with respect to $(\mathbb{H}, <)$ can be seen as an operation on labeled ℓ-hypergraphs with at most n elements which carry a weak linear order.

3 Polynomial-Time Algorithms From Symmetries

In this section, we fix $\ell \geq 3$ and a finitely bounded class $\mathcal{K}$ of ℓ-hypergraphs such that $\overrightarrow{\mathcal{K}}$ is an amalgamation class. We write $(\mathbb{H}, <)$ for the Fraïssé limit of $\overrightarrow{\mathcal{K}}$, I_n for the set of injective n-tuples of elements from H for any $n \geq 1$, I for I_ℓ, and $b_{\mathbb{H}}$ for an integer witnessing that $\mathcal{K}$ is finitely bounded. We also fix a first-order reduct $\mathbb{A}$ of $\mathbb{H}$. We say that $\mathbb{A}$ *admits an injective linear symmetry* if it has a ternary injective polymorphism m which is canonical with respect to $(\mathbb{H}, <)$, and which has the property that for any $\mathbf{a}, \mathbf{b} \in I$, the orbits under $\mathrm{Aut}(\mathbb{H})$ of $m(\mathbf{a}, \mathbf{a}, \mathbf{b}), m(\mathbf{a}, \mathbf{b}, \mathbf{a}), m(\mathbf{b}, \mathbf{a}, \mathbf{a})$ and $\mathbf{b}$ agree. Note that in this case, m induces an operation on the set $\{E, N\}$ of orbits of injective ℓ-tuples under $\mathrm{Aut}(\mathbb{H})$. We say that m *acts as a minority operation on* $\{E, N\}$ since the second condition on m can be equivalently written as $m(X, X, Y) = m(X, Y, X) = m(Y, X, X) = Y$ for all $X, Y \in \{E, N\}$.

We prove the following.

▶ **Theorem 8.** *Let $\ell \geq 3$, let $\mathcal{K}$ be a finitely bounded class of ℓ-hypergraphs such that $\overrightarrow{\mathcal{K}}$ is an amalgamation class. Let $\mathbb{A}$ be a first-order reduct of the Fraïssé limit $\mathbb{H}$ of $\mathcal{K}$. Suppose that $I \trianglelefteq_{\mathbb{A}} H^\ell$, and that $\mathbb{A}$ admits an injective linear symmetry or is such that $\mathrm{CSP}_{\mathrm{Inj}}(\mathbb{A})$ has bounded width. Then $\mathrm{CSP}(\mathbb{A})$ is solvable in polynomial time.*

If $\mathrm{CSP}_{\mathrm{Inj}}(\mathbb{A})$ has bounded width, $\mathrm{CSP}(\mathbb{A})$ has bounded width as well by general principles, and is therefore in particular solvable in polynomial time (see [29, Section 3.2]). In the rest of this section, we will therefore focus on the case when $\mathbb{A}$ admits an injective linear symmetry.

Let $\mathbb{A}$ be a first-order reduct of $\mathbb{H}$ admitting an injective linear symmetry. Set $p_1(x, y) := m(x, y, y)$. It follows that p_1 is canonical with respect to $(\mathbb{H}, <)$ and that it acts as the first projection on $\{E, N\}$, i.e., it satisfies for any $\mathbf{a}, \mathbf{b} \in I$ that the orbit of $p_1(\mathbf{a}, \mathbf{b})$ under $\mathrm{Aut}(\mathbb{H})$ is equal to the orbit of $\mathbf{a}$. Moreover, by composing p_1 with a suitable endomorphism of $\mathbb{H}$, we can assume that $p_1(y, x)$ acts lexicographically on the order, i.e., $p_1(x, y) < p_1(x', y')$ if $y < y'$ or $y = y'$ and $x < x'$ (for more details, see [29, Section 5.2]).

In the remainder of this section, we present an algorithm solving $\mathrm{CSP}(\mathbb{A})$ in polynomial time, given that $\mathbb{A}$ has among its polymorphisms operations p_1 and m with the properties derived above. Before giving the technical details, we give here an overview of the methods we employ. Let $\mathcal{I}$ be an instance of $\mathrm{CSP}(\mathbb{A})$. Our algorithm transforms $\mathcal{I}$ into an equi-satisfiable instance $\mathcal{I}'$ that is sufficiently minimal, such that the solution set of a certain relaxation of $\mathcal{I}'$ is subdirect on all projections to an ℓ-tuple $\mathbf{v}$ of pairwise distinct variables (i.e., for every tuple $\mathbf{a}$ in this projection, this relaxation of $\mathcal{I}'$ has a solution where the variables from $\mathbf{v}$ are assigned values from $\mathbf{a}$), and that additionally satisfies a condition which we call *inj-irreducibility*, inspired by Zhuk's notion of irreducibility [39, 40]. We then prove that any non-trivial instance satisfying those properties has an injective solution. This step is to be compared with the case of *absorbing reductions* in Zhuk's algorithm, and in particular with Theorem 5.5 in [40], in which it is proved that any sufficiently minimal and irreducible instance that has a solution also has a solution where an arbitrary variable is constrained to belong to an absorbing subuniverse. Since in our setting I is an absorbing subuniverse of H^ℓ in $\mathbb{A}$, this fully establishes a parallel between the present work and [40]. The algorithm that we are going to introduce will work with infinite sets which are however always unions of orbits of ℓ-tuples under $\mathrm{Aut}(\mathbb{H})$. $\mathrm{Aut}(\mathbb{H})$ is oligomorphic, i.e., it has only finitely many orbits in its action on H^k for every $k \geq 1$; in particular, there are only finitely many orbits of ℓ-tuples under $\mathrm{Aut}(\mathbb{H})$, whence we can represent every union of such orbits by listing all orbits included in this union.

We now show that the structure defined by the formula from Example 1 has polymorphisms satisfying our assumptions on p_1 and on m, and hence it falls into the scope of this section.

▶ **Example 1** (continued)**.** Let $R \subseteq H^4$ be the relation defined by ψ, and let $\mathbb{A} := (H; R)$. We define the canonical behaviour of a binary injection (i.e., a binary injective function) p_1 with respect to $(\mathbb{H}, <)$ as follows. We require that p_1 acts as the first projection on $\{E, N\}$, if O is a non-injective orbit of triples under $\mathrm{Aut}(\mathbb{H})$, and P is an injective orbit, then we require that $p_1(O, P) = p_1(P, O) = P$. Finally, for two non-injective orbits O_1, O_2 of triples under $\mathrm{Aut}(\mathbb{H}, <)$ such that $p_1(O_1, O_2)$ needs to be injective, we require that $p_1(O_1, O_2) = E$ if the minimum of any triple in O_1 appears only once in this triple, and $p_1(O_1, O_2) = N$ otherwise. Now, we take a ternary injection m' canonical with respect to $(\mathbb{H}, <)$ which behaves like a minority on $\{E, N\}$, and we define $m := m'(p_1(x, p_1(y, z)), p_1(y, p_1(z, x)), p_1(z, p_1(x, y)))$. It is easy to see that p_1 and m preserve R, hence $\mathbb{A}$ satisfies the assumptions from Theorem 8.

Let $\sim$ denote the 6-ary relation containing the tuples $(\mathbf{a}, \mathbf{b})$ where $\mathbf{a}, \mathbf{b}$ are triples that are in the same orbit under $\mathrm{Aut}(\mathbb{H})$. Note that the relation T defined by

$$(x_1, x_2, x_3, x_4) \in T :\Longleftrightarrow R(x_1, x_2, x_3, x_4) \wedge (x_1, x_3, x_2) \sim (x_4, x_2, x_3)$$

is preserved by all polymorphisms of $\mathbb{A}$ that are canonical with respect to $\mathbb{H}$ and that it contains precisely those tuples of the form (a,a,b,b) and (a,b,a,b) for arbitrary $a \neq b$. It can be seen (e.g., from [7]) that $\mathrm{Pol}(H;T)$ only contains *essentially unary* operations, of the form $(x_1,\dots,x_n) \mapsto \alpha(x_i)$ for arbitrary permutations α of H, and therefore the polymorphisms of $\mathbb{A}$ that are canonical with respect to $\mathbb{H}$ are also essentially unary. It follows that the finite-domain CSP used in the reduction from [13] is NP-complete.

3.1 Finitisation of instances

Let $\mathbb{A}$ be a first-order reduct of $\mathbb{H}$. Let $\mathcal{I} = (\mathcal{V},\mathcal{C})$ be an instance of $\mathrm{CSP}(\mathbb{A})$. In this section, we always assume that the variable set $\mathcal{V}$ is equipped with an arbitrary linear order; this assumption is however inessential and only used to formulate the statements and proofs in a more concise way. We denote by $[\mathcal{V}]^\ell$ the set of injective increasing ℓ-tuples of variables from $\mathcal{V}$. Given any instance $\mathcal{I}$ of $\mathrm{CSP}(\mathbb{A})$, consider the following CSP instance $\mathcal{I}_{\mathrm{fin}}$ over the set $\mathcal{O}$ of orbits of ℓ-tuples under $\mathrm{Aut}(\mathbb{H})$, called the *finitisation of* $\mathcal{I}$:

- The variable set of $\mathcal{I}_{\mathrm{fin}}$ is the set $[\mathcal{V}]^\ell$.
- For every constraint $C \subseteq A^U$ in $\mathcal{I}$, $\mathcal{I}_{\mathrm{fin}}$ contains the constraint C' containing the maps $g\colon [U]^\ell \to \mathcal{O}$ such that there exists $f \in C$ satisfying $f(\mathbf{v}) \in g(\mathbf{v})$ for every $\mathbf{v} \in [U]^\ell$.

This instance corresponds to the instance $\mathcal{I}_{\mathrm{Aut}(\mathbb{H}),\ell}$ from [32, Definition 3.1], with the difference that there the ℓ-element subsets of $\mathcal{V}$ were used as variables, and the domain consisted of orbits of maps. However, the translation between the two definitions is straightforward. Note that if a mapping $f\colon \mathcal{V} \to A$ is a solution of $\mathcal{I}$, then the mapping $h\colon [\mathcal{V}]^\ell \to \mathcal{O}$, where $h(\mathbf{v})$ is the orbit of $f(\mathbf{v})$ under $\mathrm{Aut}(\mathbb{H})$ for every $\mathbf{v} \in [\mathcal{V}]^\ell$ is a solution of $\mathcal{I}_{\mathrm{fin}}$.

Let $\mathcal{J} = (S,\mathcal{C})$ be an instance over the set $\mathcal{O}$ of orbits of ℓ-tuples under $\mathrm{Aut}(\mathbb{H})$, e.g., $\mathcal{J} = \mathcal{I}_{\mathrm{fin}}$ for some $\mathcal{I}$. The *injectivisation* of $\mathcal{J}$, denoted by $\mathcal{J}^{(\mathrm{inj})}$, is the instance obtained by removing from all constraints all maps taking some value outside the two injective orbits E and N.

Let $\mathcal{I} = (\mathcal{V},\mathcal{C})$ be an instance of $\mathrm{CSP}(\mathbb{A})$; the *injective finitisation of* $\mathcal{I}$ is the instance $(\mathcal{I}_{\mathrm{fin}})^{(\mathrm{inj})}$. Let $S \subseteq [\mathcal{V}]^\ell$. The *injective finitisation of* $\mathcal{I}$ *on* S is the restriction of the injective finitisation of $\mathcal{I}$ to S. For any constraint $C \in \mathcal{C}$, the corresponding constraint in the injective finitisation of $\mathcal{I}$ is called the injective finitisation of C. Note that if $\mathbb{A}$ admits an injective linear symmetry, then for any instance $\mathcal{I} = (\mathcal{V},\mathcal{C})$ of $\mathrm{CSP}(\mathbb{A})$ and for any $S \subseteq [\mathcal{V}]^\ell$, the injective finitisation of $\mathcal{I}$ on S is solvable in polynomial time. This follows from Lemma 3.4 in [32] and from the dichotomy theorem for finite-domain CSPs [39, 40, 19].

Let $\mathbb{A}$ be a first-order reduct of $\mathbb{H}$ preserved by m and by p_1. We can assume that $\mathbb{A}$ has among its relations all unions of orbits of ℓ-tuples under $\mathrm{Aut}(\mathbb{H})$ that are preserved by p_1 and by the ternary injection m. Otherwise, we expand $\mathbb{A}$ by these finitely many relations and we prove that the CSP of this expanded structure is solvable in polynomial time. Note that in particular, every orbit of ℓ-tuples under $\mathrm{Aut}(\mathbb{H})$ is a relation of $\mathbb{A}$. Moreover, we suppose that $\mathbb{A}$ has the property that for every instance $\mathcal{I}$ of $\mathrm{CSP}(\mathbb{A})$, the $(2\ell, \max(3\ell, b_{\mathbb{H}}))$-minimal instance equivalent to $\mathcal{I}$ is again an instance of $\mathrm{CSP}(\mathbb{A})$. This can be achieved without loss of generality since it is enough to expand $\mathbb{A}$ by finitely many pp-definable relations, which are also preserved by m and p_1. Note that if $\mathcal{I}$ is a $(2\ell, \max(3\ell, b_{\mathbb{H}}))$-minimal instance of $\mathrm{CSP}(\mathbb{A})$, then its injective finitisation $\mathcal{I}_{\mathrm{fin}}$ is $(2,3)$-minimal by [32, Lemma 3.2]; in particular, $\mathcal{I}_{\mathrm{fin}}$ is *cycle consistent*, i.e., it satisfies one of the basic consistency notions used in Zhuk's algorithm [39]. Moreover, if $\mathcal{I}_{\mathrm{fin}}$ is $(2,3)$-minimal, then for any solution $h\colon [\mathcal{V}]^\ell \to \mathcal{O}$ of $\mathcal{I}_{\mathrm{fin}}$, any mapping $f\colon \mathcal{V} \to A$ with $f(\mathbf{v}) \in h(\mathbf{v})$ for every $\mathbf{v} \in [\mathcal{V}]^\ell$ is a solution of $\mathcal{I}$ by [32, Lemma 3.3].

Let $\mathcal{I} = (\mathcal{V}, \mathcal{C})$ be an instance of CSP($\mathbb{A}$), and let $C \in \mathcal{C}$. Since $\mathbb{A}$ is preserved by m, there exists a set of linear equations over $\mathbb{Z}_2$ associated with the injective finitisation of C. By abuse of notation, we write every linear equation as $\sum_{\mathbf{v} \in S} X_{\mathbf{v}} = P$, where $P \in \{E, N\}$ and $S \subseteq [\mathcal{V}]^\ell$ is a set of injective ℓ-tuples of variables from the scope of C. In these linear equations, we identify E with 1 and N with 0, so that e.g. $E + E = N$ and $N + E = E$. Using this notation, the canonical behaviour of the function m on $\{E, N\}$ can be written as $m(X, Y, Z) = X + Y + Z$ which justifies the notion of $\mathbb{A}$ admitting linear symmetries.

For an instance $\mathcal{I} = (\mathcal{V}, \mathcal{C})$ of CSP($\mathbb{A}$), we define an instance $\mathcal{I}_{\text{eq}} = (\mathcal{V}, \mathcal{C}_{\text{eq}})$ of the equality-CSP (i.e., CSP over structures first-order definable over $(H; =)$) over the same base set H corresponding to the closure of the constraints under the full symmetric group on H. Formally, for every constraint $C \in \mathcal{C}$, the corresponding constraint $C_{\text{eq}} \in \mathcal{C}_{\text{eq}}$ contains all functions αh for all $h \in C$ and $\alpha \in \text{Sym}(H)$. Since $\mathbb{A}$ is preserved by a binary injection, the constraints of $\mathcal{I}_{\text{eq}}$ are preserved by the same or indeed any binary injection and hence, its CSP has relational width $(2, 3)$ by the classification of equality CSPs [7].

Let $\mathcal{I}$ be an ℓ-minimal instance of CSP($\mathbb{A}$), let $\mathbf{v} \in [\mathcal{V}]^\ell$, and let $R \subseteq \text{proj}_{\mathbf{v}}(\mathcal{I})$ be an ℓ-ary relation from the signature of $\mathbb{A}$. Let $\mathcal{I}^{\mathbf{v} \in R}$ be the instance obtained from $\mathcal{I}$ by replacing every constraint C containing all variables from $\mathbf{v}$ by $\{g \in C \mid g(\mathbf{v}) \in R\}$.

We call an ℓ-minimal instance of CSP($\mathbb{A}$) *eq-subdirect* if for every $\mathbf{v} \in [\mathcal{V}]^\ell$ and for every non-injective orbit $O \subseteq \text{proj}_{\mathbf{v}}(\mathcal{I})$ under Aut($\mathbb{H}$), the instance $(\mathcal{I}^{\mathbf{v} \in O})_{\text{eq}}$ has a solution. Note that by ℓ-minimality and since all constraints of the instance are preserved by a binary injection, the instance $(\mathcal{I}^{\mathbf{v} \in O})_{\text{eq}}$ has a solution for every injective orbit $O \subseteq \text{proj}_{\mathbf{v}}(\mathcal{I})$ under Aut($\mathbb{H}$). Indeed, any injective mapping from $\mathcal{V}$ to H is a solution of $(\mathcal{I}^{\mathbf{v} \in O})_{\text{eq}}$.

▶ **Example 9.** Let $\mathbb{H}$ be the universal homogeneous ℓ-hypergraph. Let $\mathbf{u} = (u_1, \ldots, u_\ell), \mathbf{v} = (v_1, \ldots, v_\ell)$ be disjoint ℓ-tuples of variables, and let $\mathcal{V}$ be the set of all variables contained in these tuples. We define a CSP instance $\mathcal{I} = (\mathcal{V}, \mathcal{C})$ over the set H as follows. Let $\mathbf{u}' = (u_2, \ldots, u_\ell, u_1)$. We set $\mathcal{C}$ to contain two constraints C, C' such that C contains all mappings $f \colon \mathcal{V} \to H$ such that $f(\mathbf{u})$ and $f(\mathbf{v})$ belong to the same orbit under Aut($\mathbb{H}$), and C' contains all mappings $f \colon \mathcal{V} \to H$ such that $f(\mathbf{u}')$ and $f(\mathbf{v})$ belong to the same orbit. Note that the constraints C, C' are preserved by any function which is canonical with respect to $\mathbb{H}$. It is easy to see that $\mathcal{I}$ is non-trivial and ℓ-minimal, but it is not eq-subdirect. Indeed, for any non-injective and non-constant mapping g from the set of variables of $\mathbf{u}$ to H, it holds that $g(\mathbf{u})$ and $g(\mathbf{u}')$ are contained in different orbits under Aut($\mathbb{H}$).

We can obtain an eq-subdirect instance out of an ℓ-minimal instance in polynomial time by the algorithm introduced in [29, Section 3.1]. The algorithm successively shrinks for every $\mathbf{v} \in [\mathcal{V}]^\ell$ the projection $\text{proj}_{\mathbf{v}}(\mathcal{I})$ to the union of those orbits $O \subseteq \text{proj}_{\mathbf{v}}(\mathcal{I})$ under Aut($\mathbb{H}$) for which the instance $(\mathcal{I}^{\mathbf{v} \in O})_{\text{eq}}$ has a solution; it stops when no more orbits can be removed from $\text{proj}_{\mathbf{v}}(\mathcal{I})$ for any $\mathbf{v} \in [\mathcal{V}]^\ell$.

Note that for any $1 \leq m \leq n$ and any instance $\mathcal{I}$ of CSP($\mathbb{A}$), we can compute an instance that is both eq-subdirect and (m, n)-minimal and that has the same solution set as $\mathcal{I}$ in polynomial time. Indeed, it is enough to repeat the above-mentioned algorithm and the (m, n)-minimality algorithm until no orbits under Aut($\mathbb{H}$) are removed from any constraint.

3.2 Inj-irreducibility

For any $\mathbf{a} \in H^\ell$, we write $O(\mathbf{a})$ for the orbit of $\mathbf{a}$ under Aut($\mathbb{H}$) and $O_<(\mathbf{a})$ for the orbit of $\mathbf{a}$ under Aut($\mathbb{H}, <$). Recall that being canonical with respect to $(\mathbb{H}, <)$, the function p_1 acts naturally on orbits under Aut($\mathbb{H}, <$); we can therefore abuse the notation and write $p_1(O, P)$ for orbits O, P under Aut($\mathbb{H}, <$). We will say that a non-injective orbit O of ℓ-tuples under Aut($\mathbb{H}$) is:

- *deterministic* if for every $\mathbf{a} \in O$, there exists $\alpha \in \mathrm{Aut}(\mathbb{H})$ such that $p_1(O_<(\alpha(\mathbf{a})), O_<(\mathbf{e})) = p_1(O_<(\alpha(\mathbf{a})), O_<(\mathbf{n}))$, where $\mathbf{e}, \mathbf{n}$ are arbitrary strictly increasing ℓ-tuples of elements of $\mathbb{H}$ with $\mathbf{e} \in E$, $\mathbf{n} \in N$,
- *non-deterministic* otherwise.

For a tuple $\mathbf{a}$ contained in a deterministic orbit O, we call any $\alpha \in \mathrm{Aut}(\mathbb{H})$ with the property that $p_1(O_<(\alpha(\mathbf{a})), E) = p_1(O_<(\alpha(\mathbf{a})), N)$ for the strictly increasing ordering in the second coordinate *deterministic for* $\mathbf{a}$. Note that for any $\beta \in \mathrm{Aut}(\mathbb{H}, <)$, $\beta\alpha \in \mathrm{Aut}(\mathbb{H})$ is deterministic for $\mathbf{a}$ as well since p_1 is canonical with respect to $(\mathbb{H}, <)$.

Let $\mathcal{J} = (\mathcal{V}, \mathcal{C})$ be a CSP instance over a set B. A sequence $v_1, C_1, v_2, \ldots, C_k, v_{k+1}$, where $k \geq 1$, $v_i \in \mathcal{V}$ for every $i \in [k+1]$, $C_i \in \mathcal{C}$ for every $i \in [k]$, and v_i, v_{i+1} are contained in the scope of C_i for every $i \in [k]$, is called a *path* in $\mathcal{J}$. We say that two elements $a, b \in B$ are *connected* by a path $v_1, C_1, v_2, \ldots, C_k, v_{k+1}$ if there exists a tuple $(c_1, \ldots, c_{k+1}) \in B^{k+1}$ such that $c_1 = a, c_{k+1} = b$, and such that $(c_i, c_{i+1}) \in \mathrm{proj}_{(v_i, v_{i+1})}(C_i)$ for every $i \in [k]$. Suppose that $\mathcal{J}$ is 1-minimal. The *linkedness congruence* on $\mathrm{proj}_v(\mathcal{J})$ is the equivalence relation λ on $\mathrm{proj}_v(\mathcal{J})$ defined by $(a, b) \in \lambda$ if there exists a path $v_1, C_1, v_2, \ldots, C_k, v_{k+1}$ from a to b in $\mathcal{J}$ such that $v_1 = v_{k+1} = v$. Note that for a finite relational structure $\mathbb{B}$, for a $(2,3)$-minimal instance $\mathcal{J} = (\mathcal{V}, \mathcal{C})$ of $\mathrm{CSP}(\mathbb{B})$, and for any $v \in \mathcal{V}$, the linkedness congruence λ on $\mathrm{proj}_v(\mathcal{J})$ is a relation pp-definable in $\mathbb{B}$. Indeed, it is easy to see that the binary relation containing precisely the pairs $(a, b) \in B^2$ that are connected by a particular path in $\mathcal{I}$ is pp-definable in $\mathbb{B}$. If we concatenate all paths that connect two elements $(a, b) \in \lambda$, the resulting path connects every pair $(a, b) \in \lambda$ since by the $(2,3)$-minimality of $\mathcal{J}$, every path from v to v connects c to c for every $c \in \mathrm{proj}_v(\mathcal{J})$. It follows that λ is pp-definable.

▶ **Definition 10.** *Let $\mathbb{A}$ be a first-order reduct of $\mathbb{H}$, and let $\mathcal{I} = (\mathcal{V}, \mathcal{C})$ be a non-trivial ℓ-minimal instance of* $\mathrm{CSP}(\mathbb{A})$*. We call $\mathcal{I}$* inj-irreducible *if for every set $S \subseteq [\mathcal{V}]^\ell$, one of the following holds for the instance $\mathcal{J} = \mathcal{I}_{fin}|_S$:*

- *$\mathcal{J}^{(\mathrm{inj})}$ has a solution,*
- *for some $\mathbf{v} \in S$, $\mathrm{proj}_{\mathbf{v}}(\mathcal{J})$ contains the two injective orbits and the linkedness congruence on $\mathrm{proj}_{\mathbf{v}}(\mathcal{J})$ does not connect them,*
- *for some $\mathbf{v} \in S$, the linkedness congruence on $\mathrm{proj}_{\mathbf{v}}(\mathcal{J})$ links an injective orbit to a non-deterministic orbit.*

▶ **Example 11.** We illustrate the concept of inj-irreducibility on the following instance. Let $\ell = 3$, and let $\mathbb{H}$ be the universal homogeneous ℓ-hypergraph. Let $a \neq b \in H$ be arbitrary; we call the orbits $O(a,a,b), O(a,b,a), O(b,a,a)$ under $\mathrm{Aut}(\mathbb{H})$ *half-injective*. Let us define an instance $\mathcal{I} = (\mathcal{V}, \mathcal{C})$ over the set H as follows. Above, we identified E with 1 and N with 0 in the linear equations associated to injective finitisations of constraints. In this example, we identify also all half-injective orbits under $\mathrm{Aut}(\mathbb{H})$ with 1. Hence, we can write, e.g., $E + O(a,a,b) = 0$.

Let $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ be increasing triples of pairwise disjoint variables, and set $\mathcal{V}$ to be the union of all variables of these tuples. We define $\mathcal{C}$ to be a set consisting of two constraints, C_0 and C_1, defined as follows. For $i \in \{0, 1\}$, we set C_i to contain all mappings $f \in H^{\mathcal{V}}$ such that both of the following hold:

- either $O(f(\mathbf{v}_1)) + O(f(\mathbf{v}_2)) + O(f(\mathbf{v}_3)) = i$, or $O(f(\mathbf{v}_1)) = O(f(\mathbf{v}_2)) = O(f(\mathbf{v}_3)) = \{(a,a,a) \mid a \in H\}$,
- $f(x) \neq f(y)$ for all $x, y \in \mathcal{V}$ belonging to different triples from $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$.

We show that the constraints C_0, C_1 are preserved by a binary injection p_1 and a ternary injection m that are both canonical with respect to $\mathbb{H}$. To this end, we define the canonical behaviours of p_1 and m on the orbits of triples under $\mathrm{Aut}(\mathbb{H})$ as follows. We set p_1 to act

as the first projection on the numbers associated to the respective orbits, and to satisfy $p_1(O(a,a,a),P) = p_1(P,O(a,a,a)) = P$ for an arbitrary $a \in H$ and for an arbitrary orbit P. Note that these assumptions together with the requirement that p_1 is an injection uniquely determine the behaviour of p_1 – e.g., $p_1(P,N) = E$ for an arbitrary half-injective orbit P. We set m to act as an idempotent minority on the numbers associated to the orbits, and to act as $p_1(x,p_1(y,z))$ on the orbits where the action is not determined by the previous condition. It is easy to verify that the constraints C_0 and C_1 are preserved by both p_1 and m, and hence $\mathcal{I}$ is an instance of CSP($\mathbb{A}$) for some first-order reduct $\mathbb{A}$ of $\mathbb{H}$ which falls into the scope of this section.

It immediately follows that all the half-injective orbits under Aut($\mathbb{H}$) are deterministic since $p_1(O_<(\mathbf{c}),E) = p_1(O_<(\mathbf{c}),N) = E$ for any $\mathbf{c} \in H^3$ contained in a half-injective orbit; the orbit P of the constant tuples is non-deterministic since $p_1(P,E) = E$, and $p_1(P,N) = N$. It is also easy to see that $\mathcal{I}$ is non-trivial and $(6,9)$-minimal. Moreover, $\mathcal{I}$ is not inj-irreducible. Indeed, setting $S := \{\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3\}$, the linkedness congruence on $\mathrm{proj}_{\mathbf{v}_1}(\mathcal{I}_{\mathrm{fin}}|_S)$ connects precisely all injective and half-injective orbits, and the injective finitisation of $\mathcal{I}$ on S does not have a solution.

Note that if $\mathbb{A}$ is the structure from Example 11, its CSP can be solved by the reduction to the finite from [13]. For simplicity, we choose to illustrate the concepts that we have just introduced on this example rather than on an example where the canonical behaviour of the functions p_1 and m depends on the linear order. However, Example 1 provides us with a structure admitting linear symmetries where the canonical behaviour of any polymorphism satisfying the assumptions on m or on p_1 depends on the additional linear order.

▶ **Lemma 12.** *Let C be a constraint of an instance of* CSP($\mathbb{A}$) *which contains an injective mapping, and let S be a set of variables appearing together in an unsplittable linear equation associated with the injective finitisation of C. Then for every $g \in C$, either $g(\mathbf{v})$ is in an injective or deterministic orbit for all $\mathbf{v} \in S$, or $g(\mathbf{v})$ is in a non-deterministic orbit for all $\mathbf{v} \in S$.*

▶ **Theorem 13.** *Let $\mathbb{A}$ be a first-order reduct of $\mathbb{H}$ that admits linear symmetries. Let $\mathcal{I}$ be a $(2\ell, \max(3\ell, b_{\mathbb{H}}))$-minimal, inj-irreducible instance of* CSP($\mathbb{A}$) *with variables $\mathcal{V}$ such that for every distinct $u,v \in \mathcal{V}$, $\mathrm{proj}_{(u,v)}(\mathcal{I}) \cap I_2 \neq \emptyset$. Then $\mathcal{I}$ has an injective solution.*

Proof. Note that if $\mathcal{I}$ has fewer than ℓ variables, it has an injective solution by the assumption on binary projections of $\mathcal{I}$ and since all constraints of $\mathcal{I}$ are preserved by the binary injection p_1. Let us therefore suppose that $\mathcal{I}$ has at least ℓ variables. Let us assume for the sake of contradiction that $\mathcal{I}$ does not have an injective solution. Let $\mathcal{J}$ be $\mathcal{I}_{\mathrm{fin}}$, and let $\mathcal{C}$ be the set of its constraints. By assumption, $\mathcal{J}^{(\mathrm{inj})}$ does not have a solution. Note that $\mathcal{J}^{(\mathrm{inj})}$ corresponds to a system of linear equations over $\mathbb{Z}_2$, which is therefore unsatisfiable. In case this system can be written as a diagonal block matrix, there exists a set $S \subseteq [\mathcal{V}]^\ell$ of variables such that the system of equations associated with the injectivisation of $\mathcal{L} := \mathcal{J}|_S = (S,\mathcal{C}')$ corresponds to a minimal unsatisfiable block. By definition, this means that $\mathcal{L}^{(\mathrm{inj})}$ is unsatisfiable. The instance $\mathcal{L}$ has the property that for every non-trivial partition of S into parts S_1, S_2, there exists an unsplittable equation associated with the injectivisation of a constraint $C \in \mathcal{C}'$ which contains variables from both S_1 and S_2.

Since $\mathcal{I}$ is inj-irreducible, there exists $\mathbf{v} \in S$ such that the two injective orbits are elements of $\mathrm{proj}_{\mathbf{v}}(\mathcal{L})$ and are not linked, or some injective orbit in $\mathrm{proj}_{\mathbf{v}}(\mathcal{L})$ is linked to a non-deterministic orbit in $\mathrm{proj}_{\mathbf{v}}(\mathcal{L})$.

In the first case, we note that for *all* $\mathbf{w} \in S$ such that $\mathrm{proj}_{\mathbf{w}}(\mathcal{L})$ contains the two injective orbits, the two injective orbits are not linked. Indeed, suppose that there exists $\mathbf{w} \in S$ such that $E, N \in \mathrm{proj}_{\mathbf{w}}(\mathcal{L})$ are linked, i.e., there exists a path $\mathbf{v}_1 = \mathbf{w}, C_1, \dots, C_k, \mathbf{v}_{k+1} = \mathbf{w}$ in $\mathcal{L}$ connecting E and N. Since $\mathcal{I}$ is $(2\ell, \max(3\ell, b_{\mathbb{H}}))$-minimal, Lemma 3.2 in [32] yields that $\mathcal{J}$ and hence also $\mathcal{L}$ is $(2,3)$-minimal. In particular, there exists $C \in \mathcal{C}'$ containing in its scope both $\mathbf{v}$ and $\mathbf{w}$. Let $O_1, O_2 \in \{E, N\}$ be disjoint such that there exist $g_1, g_2 \in C$ with $g_1(\mathbf{v}) = O_1, g_1(\mathbf{w}) = E, g_2(\mathbf{v}) = O_2, g_2(\mathbf{w}) = N$. It follows that the path $\mathbf{v}, C, \mathbf{w} = \mathbf{v}_1, C_1, \dots, C_k, \mathbf{v}_{k+1} = \mathbf{w}, C, \mathbf{v}$ connects O_1 with O_2 in $\mathrm{proj}_{\mathbf{v}}(\mathcal{J})$, a contradiction. Let $g\colon S \to \{E, N\}$ be defined as follows. For a fixed $\mathbf{v} \in S$, let $g(\mathbf{v})$ be an arbitrary element of $\mathrm{proj}_{\mathbf{v}}(\mathcal{L}^{(\mathrm{inj})})$. Next, for $\mathbf{w} \in S$, define $g(\mathbf{w})$ to be the unique injective orbit O such that there exists a constraint $C \in \mathcal{C}'$ containing both $\mathbf{v}$ and $\mathbf{w}$ in its scope and such that there exists $g' \in C$ with $g'(\mathbf{v}) = g(\mathbf{v})$ and $g'(\mathbf{w}) = O$. This g is a solution of $\mathcal{L}^{(\mathrm{inj})}$, a contradiction.

Thus, it must be that a non-deterministic orbit in $\mathrm{proj}_{\mathbf{v}}(\mathcal{L})$ is linked to an injective orbit in $\mathrm{proj}_{\mathbf{v}}(\mathcal{L})$. Hence, there exists a path in $\mathcal{L}$ from $\mathbf{v}$ to $\mathbf{v}$ and connecting an injective orbit to a non-deterministic one. Moreover, up to composing this path with additional constraints, one can assume that this path goes through all the variables in S. This follows by the $(2,3)$-minimality of $\mathcal{J}$. Define a partition of S where $\mathbf{w} \in S_1$ if the first time that $\mathbf{w}$ appears in the path, the element associated with $\mathbf{w}$ is in an injective orbit, and $\mathbf{w} \in S_2$ otherwise. Since the system of unsplittable equations associated with $\mathcal{L}^{(\mathrm{inj})}$ cannot be decomposed as a diagonal block matrix, some constraint $C \in \mathcal{C}'$ gives an equation in that system containing $\mathbf{u}_1 \in S_1$ and $\mathbf{u}_2 \in S_2$. Thus, there exists $g \in C$ with $g(\mathbf{u}_1)$ injective, and $g(\mathbf{u}_2)$ non-deterministic. This contradicts Lemma 12. ◀

3.3 Establishing inj-irreducibility

We introduce a polynomial-time algorithm which produces, given an instance $\mathcal{I}$ of $\mathrm{CSP}(\mathbb{A})$, an instance $\mathcal{I}'$ of $\mathrm{CSP}(\mathbb{A})$ that is either inj-irreducible or trivial and that has a solution if, and only if, $\mathcal{I}$ has a solution. It uses the fact that the injective finitisation of an instance $\mathcal{I}$ of $\mathrm{CSP}(\mathbb{A})$ on S is solvable in polynomial time for any set $S \subseteq [\mathcal{V}]^{\ell}$. This follows from the fact that the constraints of the injective finitisation of $\mathcal{I}$ are preserved by a ternary minority by Lemma 3.4 from [32].

We give a brief description of the algorithm. It gradually ensures that the instance is $(2\ell, \max(3\ell, b_{\mathbb{H}}))$-minimal, eq-subdirect, and so that for no distinct variables $u, v \in \mathcal{V}$, it holds that $\mathrm{proj}_{(u,v)}(\mathcal{I}) = \{(a,a) \mid a \in H\}$. If the instance satisfies these assumptions, we consider for every $\mathbf{u} \in [\mathcal{V}]^{\ell}$ every partition $\{E^1_{\mathbf{u}}, \dots, E^s_{\mathbf{u}}\}$ on $\mathrm{proj}_{\mathbf{u}}(\mathcal{I})$ with pp-definable classes satisfying that $E^1_{\mathbf{u}}$ contains no non-deterministic orbit. We find a subset $S \subseteq [\mathcal{V}]^{\ell}$ such that for every $\mathbf{w} \in S$, the set $\{E^1_{\mathbf{w}}, \dots, E^s_{\mathbf{w}}\}$ defined by $E^i_{\mathbf{w}} := \{\mathbf{a} \in H^{\ell} \mid \exists \mathbf{b} \in H^{\ell}\colon (\mathbf{b}, \mathbf{a}) \in \mathrm{proj}_{(\mathbf{u},\mathbf{w})}(\mathcal{I})\}$ for every $i \in [s]$ forms a partition on $\mathrm{proj}_{\mathbf{w}}(\mathcal{I})$ with the property that $E^1_{\mathbf{w}}$ contains no non-deterministic orbit, and such that this partition cannot be extended to any other tuple in $[\mathcal{V}]^{\ell}$. For every such partition, the algorithm checks if the injective finitisation of $\mathcal{I}$ on S has a solution; if not, we constrain every $\mathbf{w} \in S$ to not take any value from $E^1_{\mathbf{w}}$. It is *a priori* unclear that adding these constraints on the one hand yields an instance of $\mathrm{CSP}(\mathbb{A})$, and on the other hand that we do not transform a satisfiable instance into an unsatisfiable one in this way. The following technical result is that, in fact, this does not happen.

▶ **Theorem 14.** *The instance $\mathcal{I}'$ produced by the procedure* InjIrreducibility *in [29, Section 3.1] is an instance of* $\mathrm{CSP}(\mathbb{A})$ *and it has a solution if, and only if, the original instance has a solution. Moreover, $\mathcal{I}'$ is either trivial or inj-irreducible.*

4 Final Arguments for the Complexity Dichotomy

Finally, we briefly sketch the proof of Theorem 3. If the algebraic assumptions in the second item are met, it is known that CSP($\mathbb{A}$) is NP-complete [5]. On the other hand, if I is a binary absorbing subuniverse of H^ℓ and $\mathbb{A}$ either admits an injective linear symmetry or is such that $\mathrm{CSP}_{\mathrm{Inj}}(\mathbb{A})$ has bounded width, then CSP($\mathbb{A}$) is polynomial-time solvable by Theorem 8. It therefore remains to prove that the assumptions in the first item of Theorem 3 (i.e., the clone of polymorphisms of $\mathbb{A}$ being non-trivial) imply that $\mathbb{A}$ falls into the scope of Theorem 8.

By [35], if some polymorphism of $\mathbb{A}$ acts on $\{E, N\}$ in a non-trivial way, then $\mathbb{A}$ either admits an injective linear symmetry or is such that $\mathrm{CSP}_{\mathrm{Inj}}(\mathbb{A})$ has bounded width. Moreover, by results from [30, Proposition 25], the polymorphisms of $\mathbb{A}$ being non-trivial imply that $\mathbb{A}$ contains a binary injection witnessing that I is a binary absorbing subuniverse of H^ℓ. On the other hand, if the action of the polymorphisms of $\mathbb{A}$ on $\{E, N\}$ is trivial, then we can use the theory of smooth approximations [30] to show that the polymorphisms of $\mathbb{A}$ are trivial and $\mathbb{A}$ thus falls into the scope of the second item of Theorem 3.

References

1 Reza Akhtar and Alistair H. Lachlan. On countable homogeneous 3-hypergraphs. *Archive for Mathematical Logic*, 34:331–344, 1995. doi:10.1007/BF01387512.

2 Libor Barto, Michael Kompatscher, Miroslav Olšák, Trung Van Pham, and Michael Pinsker. The equivalence of two dichotomy conjectures for infinite domain constraint satisfaction problems. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science – LICS'17*, 2017. doi:10.1109/LICS.2017.8005128.

3 Libor Barto, Michael Kompatscher, Miroslav Olšák, Trung Van Pham, and Michael Pinsker. Equations in oligomorphic clones and the constraint satisfaction problem for ω-categorical structures. *Journal of Mathematical Logic*, 19(02):1950010, 2019. doi:10.1142/S0219061319500107.

4 Libor Barto and Marcin Kozik. Absorbing subalgebras, cyclic terms, and the constraint satisfaction problem. *Log. Methods Comput. Sci.*, 8(1), 2012. doi:10.2168/LMCS-8(1:7)2012.

5 Libor Barto, Jakub Opršal, and Michael Pinsker. The wonderland of reflections. *Israel Journal of Mathematics*, 223(1):363–398, 2018. doi:10.1007/s11856-017-1621-9.

6 Manuel Bodirsky, Peter Jonsson, and Trung Van Pham. The complexity of phylogeny constraint satisfaction problems. *ACM Transactions on Computational Logic*, 18(3), 2017. An extended abstract appeared in the conference STACS 2016. doi:10.1145/3105907.

7 Manuel Bodirsky and Jan Kára. The complexity of equality constraint languages. *Theory of Computing Systems*, 3(2):136–158, 2008. A conference version appeared in the proceedings of Computer Science Russia (CSR'06). doi:10.1007/S00224-007-9083-9.

8 Manuel Bodirsky and Jan Kára. The complexity of temporal constraint satisfaction problems. *Journal of the ACM*, 57(2), 2010. An extended abstract appeared in the Proceedings of the Symposium on Theory of Computing (STOC). doi:10.1145/1667053.1667058.

9 Manuel Bodirsky and Simon Knäuer. Network satisfaction for symmetric relation algebras with a flexible atom. *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, 35(7):6218–6226, 2021. doi:10.1609/AAAI.V35I7.16773.

10 Manuel Bodirsky and Simon Knäuer. The complexity of network satisfaction problems for symmetric relation algebras with a flexible atom. *J. Artif. Intell. Res.*, 75:1701–1744, 2022. doi:10.1613/JAIR.1.14195.

11 Manuel Bodirsky, Florent R. Madelaine, and Antoine Mottet. A universal-algebraic proof of the complexity dichotomy for monotone monadic SNP. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science – LICS'18*, 2018. doi:10.1145/3209108.3209156.

12 Manuel Bodirsky, Barnaby Martin, Michael Pinsker, and András Pongrácz. Constraint satisfaction problems for reducts of homogeneous graphs. *SIAM Journal on Computing*, 48(4):1224–1264, 2019. A conference version appeared in the Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, pages 119:1–119:14. doi:10.1137/16M1082974.

13 Manuel Bodirsky and Antoine Mottet. Reducts of finitely bounded homogeneous structures, and lifting tractability from finite-domain constraint satisfaction. In *Proceedings of the 31th Annual ACM/IEEE Symposium on Logic in Computer Science – LICS'16*, 2016. doi:10.1145/2933575.2934515.

14 Manuel Bodirsky and Antoine Mottet. A dichotomy for first-order reducts of unary structures. *Logical Methods in Computer Science*, 14(2), 2018. doi:10.23638/LMCS-14(2:13)2018.

15 Manuel Bodirsky and Michael Pinsker. Schaefer's theorem for graphs. *Journal of the ACM*, 62(3):19:1–19:52, 2015. A conference version appeared in the Proceedings of the Symposium on Theory of Computing (STOC 2011), pages 655–664. doi:10.1145/2764899.

16 Manuel Bodirsky and Michael Pinsker. Topological Birkhoff. *Transactions of the American Mathematical Society*, 367:2527–2549, 2015. doi:10.1090/S0002-9947-2014-05975-8.

17 Manuel Bodirsky, Michael Pinsker, and András Pongrácz. Projective clone homomorphisms. *Journal of Symbolic Logic*, 86(1):148–161, 2021. doi:10.1017/jsl.2019.23.

18 Bertalan Bodor. CSP dichotomy for ω-categorical monadically stable structures. PhD dissertation, Institute of Algebra, Technische Universität Dresden, 2021. URL: https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-774379.

19 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 319–330. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.37.

20 Moses Charikar, Venkatesan Guruswami, and Rajsekar Manokaran. Every permutation CSP of arity 3 is approximation resistant. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009*, pages 62–73. IEEE Computer Society, 2009. doi:10.1109/CCC.2009.29.

21 Vaggos Chatziafratis and Konstantin Makarychev. Phylogenetic CSPs are approximation resistant. *CoRR*, abs/2212.12765, 2022. doi:10.48550/arXiv.2212.12765.

22 Roland Fraïssé. Une hypothèse sur l'extension des relations finies et sa vérification dans certaines classes particulières (deuxième partie). *Synthese*, 16(1):34–46, 1966. URL: http://www.jstor.org/stable/20114493.

23 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal on Computing*, 28(1):57–104, 1998. doi:10.1137/S0097539794266766.

24 Venkatesan Guruswami, Johan Håstad, Rajsekar Manokaran, Prasad Raghavendra, and Moses Charikar. Beating the random ordering is hard: Every ordering CSP is approximation resistant. *SIAM J. Comput.*, 40(3):878–914, 2011. doi:10.1137/090756144.

25 Wilfrid Hodges. *Model theory*, volume 42 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 1993.

26 Michael Kompatscher and Trung Van Pham. A complexity dichotomy for poset constraint satisfaction. *IfCoLog Journal of Logics and their Applications (FLAP)*, 5(8):1663–1696, 2018. A conference version appeared in the Proceedings of the 34th Symposium on Theoretical Aspects of Computer Science (STACS 2017), pages 47:1–47:12. URL: https://www.collegepublications.co.uk/downloads/ifcolog00028.pdf.

27 Alistair H. Lachlan and Robert E. Woodrow. Countable ultrahomogeneous undirected graphs. *Transactions of the American Mathematical Society*, 262, 1980. doi:10.1090/S0002-9947-1980-0583847-2.

28 Richard Emil Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, January 1975. doi:10.1145/321864.321877.

29 Antoine Mottet, Tomáš Nagy, and Michael Pinsker. An order out of nowhere: a new algorithm for infinite-domain csps, 2023. arXiv:2301.12977.

30 Antoine Mottet and Michael Pinsker. Smooth approximations and CSPs over finitely bounded homogeneous structures. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science – LICS'22*, 2022. doi:10.1145/3531130.3533353.

31 Antoine Mottet, Tomáš Nagy, Michael Pinsker, and Michał Wrona. Smooth approximations and relational width collapses. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 138:1–138:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.138.

32 Antoine Mottet, Tomáš Nagy, Michael Pinsker, and Michał Wrona. When symmetries are enough: collapsing the bounded width hierarchy for infinite-domain CSPs. arxiv:2102.07531, 2022. doi:10.48550/arXiv.2102.07531.

33 George Osipov and Magnus Wahlström. Parameterized Complexity of Equality MinCSP. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 86:1–86:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.86.

34 Michael Pinsker. Current challenges in infinite-domain constraint satisfaction: Dilemmas of the infinite sheep. In *2022 IEEE 52nd International Symposium on Multiple-Valued Logic (ISMVL)*, pages 80–87, Los Alamitos, CA, USA, 2022. IEEE Computer Society. doi:10.1109/ISMVL52857.2022.00019.

35 Emil L. Post. The two-valued iterative systems of mathematical logic. *Annals of Mathematics Studies*, 5, 1941.

36 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM Symposium on Theory of Computing (STOC '78)*, pages 216–226, New York, NY, USA, 1978. Association for Computing Machinery. doi:10.1145/800133.804350.

37 Simon Thomas. Reducts of the random graph. *The Journal of Symbolic Logic*, 56(1):176–181, 1991. doi:10.2307/2274912.

38 Simon Thomas. Reducts of random hypergraphs. *Annals of Pure and Applied Logic*, 80(2):165–193, 1996. doi:10.1016/0168-0072(95)00061-5.

39 Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 331–342. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.38.

40 Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *Journal of the ACM*, 67(5):30:1–30:78, 2020.

A Complete Quantitative Axiomatisation of Behavioural Distance of Regular Expressions

Wojciech Różowski
Department of Computer Science, University College London, UK

Abstract

Deterministic automata have been traditionally studied through the point of view of language equivalence, but another perspective is given by the canonical notion of *shortest-distinguishing-word* distance quantifying the of states. Intuitively, the longer the word needed to observe a difference between two states, then the closer their behaviour is. In this paper, we give a sound and complete axiomatisation of *shortest-distinguishing-word* distance between regular languages. Our axiomatisation relies on a recently developed quantitative analogue of equational logic, allowing to manipulate rational-indexed judgements of the form $e \equiv_\varepsilon f$ meaning *term e is approximately equivalent to term f within the error margin of ε*. The technical core of the paper is dedicated to the completeness argument that draws techniques from order theory and Banach spaces to simplify the calculation of the behavioural distance to the point it can be then mimicked by axiomatic reasoning.

2012 ACM Subject Classification Theory of computation → Regular languages

Keywords and phrases Regular Expressions, Behavioural Distances, Quantitative Equational Theories

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.149

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2404.13352` [28]

Funding This work was partially supported by ERC grant Autoprobe (grant agreement 101002697).

Acknowledgements The author wishes to thank Giorgio Bacci, Leo Lobski, Alexandra Silva and Mateo Torres-Ruiz for discussions and feedback, as well as anonymous reviewers for their comments.

1 Introduction

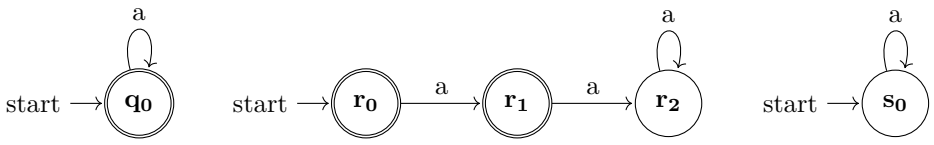
Transition systems have been widely employed to model computational phenomena. In theoretical computer science, it is customary to model computations as transition systems and subsequently reason about their equivalence or similarity. Classical examples include checking language equivalence of deterministic finite automata using Hopcroft and Karp's algorithm [18] or constructing bisimulations between labelled transition systems [25]. Throughout the years, especially in the concurrency theory community, researchers have studied a plethora of different notions of behavioural equivalences and preorders one could impose on a transition system [41]. However, in many practical applications, especially when dealing with probabilistic or quantitative transition systems, asking about such classical notions of equivalence (or similarity) could be too strict, and it might be more reasonable to ask quantitative questions about how far apart the behaviour of the two states is.

A growing line of work on *behavioural distances* [38, 5, 39, 40, 13] answers this problem by equipping state-spaces of transition systems with (pseudo)metric structures quantifying the dissimilarity of states. In such a setting, states at distance zero are not necessarily the same, but rather equivalent with respect to some classical notion of behavioural equivalence. In a nutshell, equipping transition systems with such a notion of distance crucially relies on the possibility of *lifting* the distance between the states to the distance on the observable

behaviour of the transition system. Behavioural distances were originally studied in the context of probabilistic transition systems [16, 39], where observable behaviour is in the form of probability distribution among possible transitions. In such a case, the necessary lifting of distances between states to distances between probability distributions of possible outcomes relies on the famous Kantorovich/Wasserstein liftings, studied traditionally in transportation theory [42]. In general, transition systems can be viewed more abstractly through a well-established category-theoretic framework of coalgebras for an endofunctor [27]. Recent work [5] generalised the Kantorovich/Wassertstein lifting to lifting endofunctors (modelling one-step behaviour of transition systems) from the category of sets and functions to the category of (pseudo)metric spaces and nonexpansive functions, thus allowing equipping a multitude of different kinds of transition systems with a sensible notion of behavioural distance.

Traditionally, besides looking at behavioural equivalence/similarity purely from the algorithmic point of view, one can look at those problems axiomatically, by describing behaviours of transition systems as expressions and by providing formal systems based on (in)equational logic for reasoning about equivalence/similarity of the transition systems described by the expressions. Classic examples include reasoning about language equivalence of Kleene's regular expressions representing deterministic finite automata using inference systems of Salomaa [30] or Kozen [19], or reasoning about bisimilarity of finite-state labelled transition systems through Milner's calculus of finite-state behaviours [24].

In this paper, we are interested in a similar axiomatic point of view, but in the case of behavioural distances. Unfortunately, the classical (in)equational logic cannot be applied here, as it has no way of talking about approximate equivalence. Instead, we rely on the quantitative analogue of equational logic [22], which deals with the statements of the form $e \equiv_\varepsilon f$, intuitively meaning *term e is within the distance of at most $\varepsilon \in \mathbb{Q}^+$ from the term f*. While the existing work [4, 2, 1] looked at quantitative axiomatisations of behavioural distance for probabilistic transition systems calculated through the Kantorovich/Wasserstein lifting, which can be thought of as a special case of the abstract coalgebraic framework relying on lifting endofunctors to the category of pseudometric spaces, the notions of behavioural distance for other kinds of transition systems have not been axiomatised before. It turns out that the approach to completeness used in [2] relies on properties which are not unique to distances obtained through the Kantorovich/Wasserstein lifting and can be employed to give complete axiomatisations of behavioural distances for other kinds of transition systems obtained through the coalgebraic framework [5]. In this paper, as a starting point, we look at one of the simplest instantiations of that abstract framework in the case of deterministic automata, yielding *shortest-distinguishing-word* distance. To illustrate that notion of distance, consider the following three deterministic finite automata:



Neither of the above automata are language equivalent. Their languages are respectively $\{\epsilon, a, aa, aaa, \dots\}$, $\{\epsilon, a\}$ and $\emptyset$. However, one could argue that the behaviour of the middle automaton is closer to the one on the left rather than the one on the right. In particular, languages of the left and middle automaton agree on all of the words of length less than two, while the left and right one disagree on all words. One can make this idea precise, by providing 1-bounded metric $d_{\mathcal{L}} : \mathcal{P}(A^*) \times \mathcal{P}(A^*) \to [0, 1]$ on the set of all formal languages over some fixed alphabet A given by the following formula, where $\lambda \in]0, 1[$ and $L, M \subseteq A^*$:

$$d_{\mathcal{L}}(L,M) = \begin{cases} \lambda^{|w|} & w \text{ is the shortest word that belongs to only one of } L \text{ and } M \\ 0 & \text{if } L = M \end{cases} \tag{1}$$

If we set $\lambda = \frac{1}{2}$, then $d_{\mathcal{L}}(\{\epsilon, a, aa, aaa, \dots\}, \{\epsilon, a\}) = \frac{1}{4}$ and $d_{\mathcal{L}}(\{\epsilon, a, aa, aaa, \dots\}, \emptyset) = 1$, which allows to formally state that the behaviour of the middle automaton is a better approximation of the left one, rather than the right one. Observe, that we excluded $\lambda = 0$ and $\lambda = 1$, as in both cases $d_{\mathcal{L}}$ would become a pseudometric setting all languages to be at distance zero or one. Automata in the example above correspond to the regular expressions a^*, $a + 1$ and 0 respectively. In order to determine the distance between arbitrary regular expressions e and f one would have to construct corresponding deterministic finite automata and calculate (or approximate) the distance between their languages. Instead, as a main contribution of this paper, we present a sound and complete quantitative inference system for reasoning about the shortest-distinguishing-word distance of languages denoted by regular expressions in question. Formally speaking, if $[\![-]\!] : \mathsf{Exp} \to \mathcal{P}(A^*)$ is a function taking regular expressions to their languages, then our inference system satisfies the following:

$$\vdash e \equiv_\varepsilon f \iff d_{\mathcal{L}}([\![e]\!], [\![f]\!]) \leq \varepsilon$$

Although much of our development is grounded in category theory and coalgebra, we spell out all the definitions and results concretely, without the need for specialised language. We organise the paper as follows:

1. In Section 2 we review basic definitions from automata theory and recall the semantics of regular expressions through Brzozowski derivatives [10]. Then, in order to talk about distances, we state basic definitions and properties surrounding (pseudo)metric spaces.
2. In Section 3 we instantiate the framework of coalgebraic behavioural metrics [5] to the concrete case of deterministic automata. We recall the abstract results from [5] in simple automata-theoretic terms.
3. In Section 4 we start by recalling the definitions surrounding the quantitative equational theories [22] from the literature. We then present the axioms of our inference system for the shortest-distinguishing-word distance of regular expressions, give soundness result and provide a discussion about the axioms. The interesting insight is that when relying on quantitative equational theories which contain an infinitary rule capturing the notion of convergence, there is no need for any fixpoint introduction rule. We illustrate this by axiomatically deriving Salomaa's fixpoint rule for regular expressions [30].
4. The key result of our paper is contained in Section 5, where we prove completeness of our inference system. The heart of the argument relies on showing that the behavioural distance of regular expressions can be approximated from above using Kleene's fixpoint theorem, which can be then mimicked through the means of axiomatic reasoning. This part of the paper makes heavy use of the order-theoretic and Banach space structures carried by the sets of pseudometrics over a given set.
5. We conclude in Section 6, review related literature, and sketch directions for future work.

Omitted proofs appear in the full version [28].

2 Preliminaries

We start by recalling basic definitions surrounding deterministic automata, regular expressions and (pseudo)metric spaces from the literature.

Deterministic automata. A deterministic automaton $\mathcal{M}$ with inputs in a finite alphabet A is a pair $(M, \langle o_M, t_M \rangle)$ consisting of a set of states M and a pair of functions $\langle o_M, t_M \rangle$, where $o_M : M \to \{0,1\}$ is the *output* function which determines whether a state m is final ($o_M(m) = 1$) or not ($o_M(m) = 0$), and $t : M \to M^A$ is the *transition* function, which, given an input letter a determines the next state. If the set M of states is finite, then we call an automaton $\mathcal{M}$ a deterministic finite automaton (DFA). We will frequently write m_a to denote $t_M(m)(a)$ and refer to m_a as the derivative of m for the input a. Definition of derivatives can be inductively extended to words $w \in A^*$, by setting $m_\varepsilon = m$ and $m_{aw'} = (m_a)_{w'}$. Note that our definition of deterministic automaton slightly differs from the most common one in the literature, by not explicitly including the initial state. Instead of talking about the language of the automaton, we will talk about the languages of particular states of the automaton. Given a state $m \in M$, we write $L_{\mathcal{M}}(m) \subseteq A^*$ for its language, which is formally defined by $L_{\mathcal{M}}(m) = \{w \in A^* \mid o(m_w) = 1\}$. Given two deterministic automata $(M, \langle o_M, t_M \rangle)$ and $(N, \langle o_N, t_N \rangle)$, a function $h : M \to N$ is a homomorphism if it preserves outputs and input derivatives, that is $o_N(h(m)) = o_M(m)$ and $h(m)_a = h(m_a)$. The set of all languages $\mathcal{P}(A^*)$ over an alphabet A can be made into a deterministic automaton $(\mathcal{P}(A^*), \langle o_L, t_L \rangle)$, where for $l \in \mathcal{P}(A^*)$ the output function is given by $o_L(l) = [\epsilon \in l]$ and for all $a \in A$ the input derivative is defined to be $l_a = \{w \mid aw \in l\}$. This automaton is *final*, that is for any other automaton $\mathcal{M} = (M, \langle o_M, t_M \rangle)$ there exists a unique homomorphism from M to $\mathcal{P}(A^*)$, which is precisely given by the map $L_{\mathcal{M}} : M \to \mathcal{P}(A^*)$ taking each state $m \in S$ to its language. Given a set of states $M' \subseteq M$, we write $\langle M' \rangle_{\mathcal{M}} \subseteq M$ for the smallest set of states reachable from M' through the transition function of the automaton $\mathcal{M}$. Clearly, $(\langle M' \rangle_{\mathcal{M}}, \langle o_M, t_M \rangle)$ is a deterministic automaton. We will abuse the notation and write $\langle M' \rangle_{\mathcal{M}}$ for $(\langle M' \rangle_{\mathcal{M}}, \langle o_M, t_M \rangle)$. The canonical inclusion map $\iota : \langle M' \rangle_{\mathcal{M}} \hookrightarrow M$ given by $\iota(m) = m$ for all $m \in \langle M' \rangle_{\mathcal{M}}$ is a homomorphism from $\langle M' \rangle_{\mathcal{M}}$ to $\mathcal{M}$. In the case of singleton and two-element sets of states, we will simplify the notation and write $\langle m \rangle_{\mathcal{M}}$ and $\langle m, m' \rangle_{\mathcal{M}}$.

Regular expressions. We let e, f range over *regular expressions over* A generated by the following grammar:

$$e, f \in \mathsf{Exp} ::= 0 \mid 1 \mid a \in A \mid e + f \mid e\,;f \mid e^*$$

The standard interpretation of regular expressions $[\![-]\!] : \mathsf{Exp} \to \mathcal{P}(A^*)$ is inductively defined by the following:

$$[\![0]\!] = \emptyset \quad [\![1]\!] = \{\epsilon\} \quad [\![a]\!] = \{a\} \quad [\![e + f]\!] = [\![e]\!] \cup [\![f]\!] \quad [\![e\,;f]\!] = [\![e]\!] \diamond [\![f]\!] \quad [\![e^*]\!] = [\![e]\!]^*$$

We write ϵ for the empty word. Given $L, M \subseteq A^*$, we define $L \diamond M = \{lm \mid l \in L, m \in M\}$, where mere juxtaposition denotes concatenation of words. L^* denotes the *asterate* of the language L defined as $L^* = \bigcup_{i \in \mathbb{N}} L^i$ with $L^0 = \{\epsilon\}$ and $L^{n+1} = L \diamond L^n$.

Brzozowski derivatives. The famous Kleene's theorem states that the formal languages accepted by DFA are in one-to-one correspondence with formal languages definable by regular expressions. One direction of this theorem involves constructing a DFA for an arbitrary regular expression. The most common way is via Thompson construction, ϵ-transition removal and determinisation. Instead, we recall a direct construction due to Brzozowski [10], in which the set Exp of regular expressions is equipped with a structure of deterministic automaton $\mathcal{R} = (\mathsf{Exp}, \langle o_{\mathcal{R}}, t_{\mathcal{R}} \rangle)$ through so-called Brzozowski derivatives [10]. The output derivative $o_{\mathcal{R}} : \mathsf{Exp} \to \{0,1\}$ is defined inductively by the following for $a \in A$ and $e, f \in \mathsf{Exp}$:

$$o_{\mathcal{R}}(0) = 0 \quad o_{\mathcal{R}}(1) = 1 \quad o_{\mathcal{R}}(a) = 0$$
$$o_{\mathcal{R}}(e + f) = o_{\mathcal{R}}(e) \vee o_{\mathcal{R}}(f) \quad o_{\mathcal{R}}(e\,;f) = o_{\mathcal{R}}(e) \wedge o_{\mathcal{R}}(f) \quad o_{\mathcal{R}}(e^*) = 1$$

Similarly, the transition derivative $t_{\mathcal{R}} \in \mathsf{Exp} \to A \to \mathsf{Exp}$ denoted $t_{\mathcal{R}}(e)(a) = (e)_a$ is defined by the following:

$$(0)_a = 0 \quad (1)_a = 0 \quad (a')_a = \begin{cases} 1 & a = a' \\ 0 & a \neq a' \end{cases}$$
$$(e + f)_a = (e)_a + (f)_a \quad (e\,;f)_a = (e_a)\,;f + o_{\mathcal{R}}(e)\,;f \quad (e^*) = (e)_a\,;e^*$$

The canonical language-assigning homomorphism from $\mathcal{R}$ to $\mathcal{L}$, happens to coincide with the semantics map $[\![-]\!]$ assigning a language to each regular expression.

▶ **Lemma 1** ([33, Theorem 3.1.4]). *For all $e \in \mathsf{Exp}$, $[\![e]\!] = L_{\mathcal{R}}(e)$*

Instead of looking at infinite-state automaton defined on the state-space of all regular expressions, we can restrict ourselves to the subautomaton $\langle e \rangle_{\mathcal{R}}$ of $\mathcal{R}$ while obtaining the semantics of e.

▶ **Lemma 2.** *For all $e \in \mathsf{Exp}$, $[\![e]\!] = L_{\langle e \rangle_{\mathcal{R}}}(e)$*

Unfortunately, for an arbitrary regular expression $e \in \mathsf{Exp}$, the automaton $\langle e \rangle_{\mathcal{R}}$ is not guaranteed to have a finite set of states. However, simplifying the transition derivatives by removing duplicates in the expressions in the form $e_1 + \cdots + e_n$, guarantees a finite number of reachable states from any expression. Formally speaking, let $\doteqdot \subseteq \mathsf{Exp} \times \mathsf{Exp}$ be the least congruence relation closed under $(e + f) + g \doteqdot e + (f + g)$ (Associativity), $e + f \doteqdot f + e$ (Commutativity) and $e \doteqdot e + e$ (Idempotence) for all $e, f, g \in \mathsf{Exp}$. We will write $\mathsf{Exp}/\doteqdot$ for the quotient of Exp by the relation $\doteqdot$ and $[-]_{\doteqdot} : \mathsf{Exp} \to \mathsf{Exp}/\doteqdot$ for the canonical map taking each expression $e \in \mathsf{Exp}$ into its equivalence class $[e]_{\doteqdot}$ modulo $\doteqdot$. Because of [27, Proposition 5.8], $\mathsf{Exp}/\doteqdot$ can be equipped with a structure of deterministic automaton $\mathcal{Q} = (\mathsf{Exp}/\doteqdot, \langle o_{\mathcal{Q}}, t_{\mathcal{Q}} \rangle)$, where for all $e \in \mathsf{Exp}, a \in A$, $o_{\mathcal{Q}}([e]_{\doteqdot}) = o_{\mathcal{R}}(e)$ and $([e]_{\doteqdot})_a = [e_a]_{\doteqdot}$, which makes the quotient map $[-]_{\doteqdot} : \mathsf{Exp} \to \mathsf{Exp}/\doteqdot$ into an automaton homomorphism from the Brzozowski automaton $\mathcal{R}$ into $\mathcal{Q}$. This automaton enjoys the following property:

▶ **Lemma 3** ([10, Theorem 4.3]). *For any $e \in \mathsf{Exp}$, the set $\langle e \rangle_{\mathcal{Q}} \subseteq \mathsf{Exp}/\doteqdot$ is finite.*

Through an identical line of reasoning as before (Lemma 2), we can show that:

▶ **Lemma 4.** *For all $e \in \mathsf{Exp}$, $L_{\langle [e]_{\doteqdot} \rangle_{\mathcal{Q}}}([e]_{\doteqdot}) = [\![e]\!]$*

(Pseudo)metric spaces. Let $\top \in]0, \infty]$ be a fixed maximal element. A $\top$-bounded *pseudometric* on a set X (equivalently $\top$-*pseudometric* or even just a *pseudometric* if $\top$ is clear from the context) is a function $d : X \times X \to [0, \top]$ satisfying $d(x, x) = 0$ (*reflexivity*), $d(x, y) = d(y, x)$ (*symmetry*) and $d(x, z) \leq d(x, y) + d(y, z)$ (*triangle inequality*) for all $x, y, z \in X$. If additionally $d(x, y) = 0$ implies $x = y$, d is called a $\top$-*metric*. A *(pseudo)metric* space is a pair (X, d) where X is a set and d is a (pseudo)metric on X. Given pseudometric spaces (X, d_X) and (Y, d_Y), we call a map $f : X \to Y$ *nonexpansive*, if for all $x, x' \in X$, $d_Y(f(x), f(x')) \leq d_X(x, x')$ and an *isometry* if $d_Y(f(x), f(x')) = d_X(x, x')$. A simple example of a pseudometric is the discrete metric which can be defined on any set X as $d_X(x, x) = 0$ for all $x \in X$ and $d(x, y)_X = \top$ for $x, y \in X$ such that $x \neq y$. The set D_X of (pseudo)metrics over some fixed set X can be equipped with a partial order structure given by the pointwise order, i.e. $d \sqsubseteq d' \iff \forall x, x' \in X. d(x, y) \leq d'(x, y)$.

▶ **Lemma 5** ([5, Lemma 3.2]). *$(D_X, \sqsubseteq)$ is a complete lattice. The join of an arbitrary set of pseudometrics $D \subseteq D_X$ is taken pointwise, ie.* $(\sup D)(x,y) = \sup\{d(x,y) \mid d \in D\}$ *for $x, y \in X$. The meet of D is defined to be* $\inf D = \sup\{d \mid d \in D_X, \forall d' \in D, d \sqsubseteq d'\}$.

Crucially for our completeness proof, if we are dealing with descending chains, that is sequences $\{d_i\}_{i\in\mathbb{N}}$, such that $d_i \sqsupseteq d_{i+1}$ for all $i \in \mathbb{N}$, then we can also calculate infima in the pointwise way[1].

▶ **Lemma 6.** *Let $\{d_i\}_{i\in\mathbb{N}}$ be an infinite descending chain in the lattice $(D_X, \sqsubseteq)$ of pseudometrics over some fixed set X. Then* $(\inf\{d_i \mid i \in \mathbb{N}\})(x,y) = \inf\{d_i(x,y) \mid i \in \mathbb{N}\}$ *for any $x, y \in X$.*

Proof. It suffices to argue that $d(x,y) = \inf\{d_i(x,y) \mid i \in \mathbb{N}\}$ is a pseudometric. For reflexivity, observe that $d(x,x) = \inf\{d_i(x,x) \mid i \in \mathbb{N}\} = \inf\{0\} = 0$ for all $x \in X$. For symmetry, we have that $d(x,y) = \inf\{d_i(x,y) \mid i \in \mathbb{N}\} = \inf\{d_i(y,x) \mid i \in \mathbb{N}\} = d(y,x)$ for any $x, y \in X$. The only difficult case is triangle inequality. First, let $i, j \in \mathbb{N}$ and define $k = \max(i,j)$. Since $d_k \sqsubseteq d_i$ and $d_k \sqsubseteq d_j$, we have that $d_k(x,y)+d_k(y,z) \leq d_i(x,y)+d_j(y,z)$. Therefore $\inf\{d_l(x,y) + d_l(y,z) \mid l \in \mathbb{N}\}$ is a lower bound of $d_i(x,y) + d_j(y,z)$ for any $i, j \in \mathbb{N}$ and hence it is below the greatest lower bound, that is $\inf\{d_l(x,y) + d_l(y,z) \mid l \in \mathbb{N}\} \leq \inf\{d_i(x,y) + d_j(y,z) \mid i, j \in \mathbb{N}\}$. We can use that property to show the following $d(x,y) = \inf\{d_i(z,y) \mid i \in \mathbb{N}\} \leq \inf\{d_i(x,y) + d_i(y,z) \mid i \in \mathbb{N}\} \leq \inf\{d_i(x,y) + d_j(y,z) \mid i, j \in \mathbb{N}\} = \inf\{d_i(x,y) \mid i \in \mathbb{N}\} + \inf\{d_j(y,z) \mid j \in \mathbb{N}\} = d(x,y) + d(y,z)$, which completes the proof. ◀

Additionally, the set of pseudometrics can be equipped with a norm. We write $\overline{\mathbb{R}} = [-\infty, \infty]$ for the set of extended reals. For any set X, the set of functions $\overline{\mathbb{R}}^{X\times X}$, which is a superset of D_X, can be seen as a Banach space [26] (complete normed vector space) by means of the sup-norm $\|d\| = \sup_{x,y\in X} |d(x,y)|$. This structure will implicitly underly some of the claims used as intermediate steps in the proof of completeness.

3 Behavioural distance

We now instantiate the abstract coalgebraic framework [5] to the case of deterministic automata relying on the lifting described in [5, Example 5.33]. We concretise the generic results from that paper and spell them in simple automata-theoretic terms.

Lifting pseudometrics. Let $\mathcal{M} = (M, \langle o_M, t_M\rangle)$ be a deterministic automaton. Its one-step observable behaviour (after applying the output and transition derivatives) can be seen as pairs of the type $\{0,1\} \times M^A$, where the first component determines whether the given state is accepting or not and the second one gives successor state for each letter from the input alphabet. Let's say we have two such observations $\langle o_1, f_1\rangle, \langle o_2, f_2\rangle \in \{0,1\} \times M^A$. If we had had some notion of a distance defined on the state-space of our automaton, or speaking more formally a 1-pseudometric $d : M \times M \to [0,1]$, then we can *lift* this notion of distance, to a distance on observations, given by the following:

$$d_{\{0,1\}\times M^A}(\langle o_1, f_1\rangle, \langle o_2, f_2\rangle) = \max\{d_2(o_1,o_2), \lambda \cdot \max_{a\in A} d(f_1(a), f_2(a))\} \qquad \lambda \in]0,1[$$

[1] Lemma 6 is one of the intermediate results used in the proof of [2, Lemma 5.6] that was communicated to us by the authors of [2]. As this result was excluded in the mentioned paper, we incorporated it along with its proof for the sake of completeness.

The definition above involves d_2, the discrete metric on the set $\{0,1\}$. One can observe that $d_{\{0,1\}\times M^A}$ is again a 1-pseudometric, but this time defined on the set $\{0,1\}\times M^A$ instead. We now move on to showing how one could use this lifting in order to equip a state-space of automaton $\mathcal{M}$ with a sensible notion of behavioural pseudometric.

Behavioural pseudometric. If one gave us a 1-metric $d : M \times M \to [0,1]$ on the state-space of the automaton, we could use our lifting to produce a new pseudometric $\Phi_{\mathcal{M}}(d) : M \times M \to [0,1]$ on the same set, which would calculate a distance between an arbitrary pair of states by first applying the output and transition derivatives to obtain a pair of observations and then by calculating the distance between them using the aforementioned lifting of pseudometric d to the pseudometric defined on the set $\{0,1\}\times M^A$. Formally speaking, define a map $\Phi_{\mathcal{M}} : D_M \to D_M$ on the lattice of 1-pseudometrics on M given by the following:

$$\Phi_{\mathcal{M}}(d)(m,m') = \max\{d_2(o_M(m), o_M(m')), \lambda \cdot \max_{a\in A} d(m_a, m'_a)\} \qquad \lambda \in]0,1[$$

The construction above only tells us how to construct new pseudometrics on the state-space of the automaton out of existing ones, but does not give one to start with. It turns out, that the map $\Phi_{\mathcal{M}}$ is a monotone mapping on the lattice of 1-psuedometrics on the set M [5, Lemma 6.1]. Because of that, one can use the Knaster-Tarski fixpoint theorem [36] and construct its least fixed point, explicitly given by $d_{\mathcal{M}} = \inf\{d \mid d \in D_M \wedge \Phi_{\mathcal{M}}(d) \sqsubseteq d\}$. Pseudometrics, which are fixpoints of $\Phi_{\mathcal{M}}$ intuitively interact well with the automaton structure, as they satisfy the property that the distance between two states is the same as the distance between their observable behaviour calculated using the lifting. Taking the least such pseudometric satisfies several desirable properties [5] and thus we will call $d_{\mathcal{M}}$ a behavioural pseudometric on the automaton $\mathcal{M}$. First of all, preserving automaton transitions also preserves behavioural distances.

▶ **Proposition 7.** *Let $\mathcal{M} = (M, \langle o_M, t_M\rangle)$ and $\mathcal{N} = (N, \langle o_N, t_N\rangle)$ be deterministic automata. If $h : M \to N$ is a homomorphism, then it is also an isometric mapping between pseudometric spaces $(M, d_{\mathcal{M}})$ and $(N, d_{\mathcal{N}})$.*

If we look at the final automaton on the set of all formal languages over a fixed finite alphabet A, then one can easily verify that the behavioural distance given by the least fixpoint construction precisely corresponds to the Equation (1) defining the shortest-distinguishing-word distance we stated in Section 1. In general, states of an arbitrary deterministic automaton characterised by the behavioural pseudometric to be in distance zero are language equivalent. When we look at $d_{\mathcal{L}}$ defined on the states of the final automaton, whose state-space consists of formal languages, then the language equivalence corresponds to the equality of states. In other words $d_{\mathcal{L}}$ becomes a metric space.

▶ **Lemma 8.** *Let $\mathcal{M} = (M, \langle o_M, t_M\rangle)$ be an arbitrary deterministic automaton and let $\mathcal{L} = (\mathcal{P}(A^*), \langle o_L, t_L\rangle)$ be a deterministic automaton structure on the set of all languages over an alphabet A.*

1. *$(\mathcal{P}(A^*), d_{\mathcal{L}})$ is a metric space.*
2. *For any $m, m' \in M$, $d_{\mathcal{M}}(m,m') = 0 \iff L_{\mathcal{M}}(m) = L_{\mathcal{M}}(m')$.*

4 Quantitative Axiomatisation

In order to provide a quantitative inference system for reasoning about the behavioural distance of languages denoted by regular expressions, we first recall the definition of quantitative equational theories from the existing literature [22, 2] following the notational conventions

from [2]. We then present our axiomatisation and demonstrate its soundness. The interesting thing about our axiomatisation is the lack of any fixpoint introduction rule. We show that in the case of quantitative analogue of equational logic [22] containing the infinitary rule capturing the notion of convergence, we can use our axioms to derive Salomaa's fixpoint rule from his axiomatisation of language equivalence of regular expressions [30].

Quantitative equational theories. Let Σ be an algebraic signature (in the sense of universal algebra [11]) consisting of operation symbols $f_n \in \Sigma$ of arity $n \in \mathbb{N}$. If we write X for the countable set of *metavariables*, then $\mathbb{T}(\Sigma, X)$ denotes a set of freely generated terms over X built from the signature Σ. As a notational convention, we will use letters $t, s, u, \ldots \in \mathbb{T}(\Sigma, X)$ to denote terms. By a *substitution* we mean a function of the type $\sigma\colon X \to \mathbb{T}(\Sigma, X)$ allowing to replace metavariables with terms. Each substitution can be inductively extended to terms in a unique way by setting $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$ for each operation symbol $f_n \in \Sigma$ from the signature. We will write $\mathcal{S}(\Sigma)$ for the set of all substitutions. Given two terms $t, s \in \mathbb{T}(\Sigma, X)$ and a nonnegative rational number $\varepsilon \in \mathbb{Q}^+$ denoting the distance between the terms, we call $t \equiv_\varepsilon s$ a *quantitative equation (of type Σ)*. Notation-wise, we will write $\mathcal{E}(\Sigma)$ to denote the set of all quantitative equations (of type Σ) and we will use the capital Greek letters $\Gamma, \Theta, \ldots \subseteq \mathcal{E}(\Sigma)$ to denote the subsets of $\mathcal{E}(\Sigma)$. By a *deducibility relation* we mean a binary relation denoted $\vdash \subseteq \mathcal{P}(\mathcal{E}(\Sigma)) \times \mathcal{E}(\Sigma)$. Similarly, to the classical equational logic, we will use the following notational shorthands $\Gamma \vdash t \equiv_\varepsilon s \iff (\Gamma, t \equiv_\varepsilon s) \in \vdash$ and $\vdash t \equiv_\varepsilon s \iff \emptyset \vdash t \equiv_\varepsilon s$. Furthermore, following the usual notational conventions, we will write $\Gamma \vdash \Theta$ as a shorthand for the situation when $\Gamma \vdash t \equiv_\varepsilon s$ holds for all $t \equiv_\varepsilon s \in \Theta$. To call $\vdash$ a *quantitative deduction system (of type Σ)* it needs to satisfy the following rules of inference:

(Refl) $\vdash t \equiv_0 t$,

(Symm) $\{t \equiv_\varepsilon s\} \vdash s \equiv_\varepsilon t$,

(Triang) $\{t \equiv_\varepsilon u, u \equiv_{\varepsilon'} s\} \vdash t \equiv_{\varepsilon+\varepsilon'} s$,

(Max) $\{t \equiv_\varepsilon s\} \vdash t \equiv_{\varepsilon+\varepsilon'} s$, for all $\varepsilon' > 0$,

(Cont) $\{t \equiv_{\varepsilon'} s \mid \varepsilon' > \varepsilon\} \vdash t \equiv_\varepsilon s$,

(NExp) $\{t_1 \equiv_\varepsilon s_1, \ldots, t_n \equiv_\varepsilon s_n\} \vdash f(t_1, \ldots, t_n) \equiv_\varepsilon f(s_1, \ldots, s_n)$, for all $f_n \in \Sigma$,

(Subst) If $\Gamma \vdash t \equiv_\varepsilon s$, then $\sigma(\Gamma) \vdash \sigma(t) \equiv_\varepsilon \sigma(s)$, for all $\sigma \in \mathcal{S}(\Sigma)$,

(Cut) If $\Gamma \vdash \Theta$ and $\Theta \vdash t \equiv_\varepsilon s$, then $\Gamma \vdash t \equiv_\varepsilon s$,

(Assum) If $t \equiv_\varepsilon s \in \Gamma$, then $\Gamma \vdash t \equiv_\varepsilon s$.

where $\sigma(\Gamma) = \{\sigma(t) \equiv_\varepsilon \sigma(s) \mid t \equiv_\varepsilon s \in \Gamma\}$. Finally, by a *quantitative equational theory* we mean a set $\mathcal{U}$ of universally quantified *quantitative inferences* $\{t_1 \equiv_{\varepsilon_1} s_1, \ldots, t_n \equiv_{\varepsilon_n} s_n\} \vdash t \equiv_\varepsilon s$, with *finitely many premises*, closed under $\vdash$-derivability.

Quantitative algebras. Quantitative equational theories lie on the syntactic part of the picture. On the semantic side, we have their models called *quantitative algebras*, defined as follows.

▶ **Definition 9** ([22, Definition 3.1])**.** *A quantitative algebra is a tuple $\mathcal{A} = (A, \Sigma^{\mathcal{A}}, d^{\mathcal{A}})$, such that $(A, \Sigma^{\mathcal{A}})$ is an algebra for the signature Σ and $(A, d^{\mathcal{A}})$ is an ∞-pseudometric such that for all operation symbols $f_n \in \Sigma$, for all $1 \leq i \leq n$, $a_i, b_i \in A$, $d^{\mathcal{A}}(a_i, b_i) \leq \varepsilon$ implies $d^{\mathcal{A}}(f^{\mathcal{A}}(a_1, \ldots, a_n), f^{\mathcal{A}}(b_1, \ldots, b_n)) \leq \varepsilon$.*

Consider a quantitative algebra $\mathcal{A} = (A, \Sigma^{\mathcal{A}}, d^{\mathcal{A}})$. Given an assignment $\iota\colon X \to A$ of meta-variables from X to elements of carrier A, one can inductively extend it to Σ-terms $t \in \mathbb{T}(\Sigma, X)$ in a unique way. We will abuse the notation and just write $\iota(t)$ for the interpretation of the term t in quantitative algebra $\mathcal{A}$. We will say that $\mathcal{A}$ *satisfies* the quantitative inference $\Gamma \vdash t \equiv_\varepsilon s$, written $\Gamma \models_{\mathcal{A}} t \equiv_\varepsilon s$, if for any assignment of the meta-variables $\iota\colon X \to A$ it is the case that for all $t' \equiv_{\varepsilon'} s' \in \Gamma$ we have that $d^{\mathcal{A}}(\iota(t'), \iota(s')) \leq \varepsilon'$ implies $d^{\mathcal{A}}(\iota(t), \iota(s)) \leq \varepsilon$. Finally, we say that a quantitative algebra $\mathcal{A}$ *satisfies* (or is a *model* of) the quantitative theory $\mathcal{U}$, if whenever $\Gamma \vdash t \equiv_\varepsilon s \in \mathcal{U}$, then $\Gamma \models_{\mathcal{A}} t \equiv_\varepsilon s$.

Quantitative algebra of regular expressions. From now on, let's focus on the signature $\Sigma = \{0_0, 1_0, +_2, ;_2, (-)^*{}_1\} \cup \{a_0 \mid a \in A\}$, where A is a finite alphabet. This signature consists of all operations of regular expressions. We can easily interpret all those operations in the set Exp of all regular expressions, using trivial interpretation functions eg. $+^{\mathcal{B}}(e, f) = e + f$, which interpret the operations by simply constructing the appropriate terms. Formally speaking, we can do this because the set Exp is the carrier of initial algebra [11] (free algebra over the empty set of generators) for the signature Σ.

To make this algebra into a quantitative algebra, we first equip the set Exp with a ∞-pseudometric, given by $d^{\mathcal{B}}(e, f) = d_{\mathcal{L}}([\![e]\!], [\![f]\!])$ for all $e, f \in \mathsf{Exp}$. Recall that $d_{\mathcal{L}}$ used in the definition above is a behavioural pseudometric on the final deterministic automaton carried by the set $\mathcal{P}(A^*)$ of all formal languages over an alphabet A. In other words, we define the distance between arbitrary expressions e and f to be the distance between formal languages $[\![e]\!]$ and $[\![f]\!]$ calculated through the shortest-distinguishing-word metric. It turns out, that in such a situation all the interpretation functions of Σ-algebra structure on Exp are non-expansive with respect to to the pseudometric defined above. In other words, we have that:

▶ **Lemma 10.** *$\mathcal{B} = (\mathsf{Exp}, \Sigma^{\mathcal{B}}, d^{\mathcal{B}})$ is a quantitative algebra.*

Axiomatisation. In order to talk about the quantitative algebra $\mathcal{B}$ of the behavioural distance of regular expressions in an axiomatic way, we introduce the quantitative equational theory REG (Figure 1). The first group of axioms capture properties of the nondeterministic

	Nondeterministic choice		**Sequential composition**
(SL1)	$\vdash e + e \equiv_0 e$,	(1S)	$\vdash 1\,;e \equiv_0 e$,
(SL2)	$\vdash e + f \equiv_0 f + e$,	(S)	$\vdash e\,;(f\,;g) \equiv_0 (e\,;f)\,;g$,
(SL3)	$\vdash (e + f) + g \equiv_0 e + (f + g)$,	(S1)	$\vdash e\,;1 \equiv_0 e$,
(SL4)	$\vdash e + 0 \equiv_0 e$,	(0S)	$\vdash 0\,;e \equiv_0 0$,
(SL5)	$\{e \equiv_\varepsilon g, f \equiv_{\varepsilon'} h\}$	(S0)	$\vdash e\,;0 \equiv_0 0$,
	$\vdash e + f \equiv_{\max(\varepsilon,\varepsilon')} g + h$,	(D1)	$\vdash e\,;(f + g) \equiv_0 e\,;f + e\,;g$,
		(D2)	$\vdash (e + f)\,;g \equiv_0 e\,;g + f\,;g$,
	Loops		**Behavioural pseudometric**
(Unroll)	$\vdash e^* \equiv_0 e\,;e^* + 1$,	(Top)	$\vdash e \equiv_1 f$,
(Tight)	$\vdash (e + 1)^* \equiv_0 e^*$,	(λ-Pref)	$\{e \equiv_\varepsilon f\} \vdash a\,;e \equiv_{\varepsilon'} a\,;f$, for $\varepsilon' \geq \lambda \cdot \varepsilon$

Figure 1 Axioms of the quantitative equational theory REG for $e, f, g \in \mathsf{Exp}$ and $a \in A$.

choice operator $+$ (SL1-SL5). The first four axioms (SL1-SL4) are the usual laws of semilattices with bottom element 0. (SL5) is a quantitative axiom allowing one to reason about distances between sums of expressions in terms of distances between expressions being summed. Moreover, (SL1-SL5) are axioms of so-called *Quantitative Semilattices with zero*, which

have been shown to axiomatise the Hausdorff metric [22]. The sequencing axioms (1S), (S1), (S) state that the set Exp of regular expressions has the structure of a monoid (with neutral element 1) with absorbent element 0 (0S), (S0). Additionally, (D1-D2) talk about interaction of the nondeterministic choice operator + with sequential composition. The loop axioms (Unroll) and (Tight) are directly inherited from Salomaa's axiomatisation of language equivalence of regular expressions [30]. (Unroll) axiom associates loops with their intuitive behaviour of choosing, at each step, between successful termination and executing the loop body once. (Tight) states that the loop whose body might instantly terminate, causing the next loop iteration to be executed immediately is provably equivalent to a different loop, whose body does not contain immediate termination. The last remaining group are behavioural pseudometric axioms. (Top) states that any two expressions are at most in distance one from each other. Finally, (λ-Pref) captures the fact that prepending the same letter to arbitrary expressions shrinks the distance between them by the factor of $\lambda \in]0,1[$ (used in the definition of $d^{\mathcal{B}}$). This axiom is adapted from the axiomatisation of discounted probabilistic bisimilarity distance [2].

Through a simple induction on the length of derivation, one can verify that indeed $\mathcal{B}$ is a model of the quantitative theory REG.

▶ **Theorem 11.** *(Soundness) The quantitative algebra $\mathcal{B} = (\mathsf{Exp}, \Sigma^{\mathcal{B}}, d^{\mathcal{B}})$ is a model of the quantitative theory* REG. *In other words, for any $e, f \in \mathsf{Exp}$ and $\varepsilon \in \mathbb{Q}^+$, if $\Gamma \vdash e \equiv_\varepsilon f \in \mathsf{REG}$, then $\Gamma \models_{\mathcal{B}} e \equiv_\varepsilon f$*

Proof. By the structural induction on the judgement $\Gamma \vdash e \equiv_\varepsilon f \in \mathsf{REG}$. (Subst), (Cut) and (Assum) deduction rules from classical logic hold immediately. The soundness of (Refl), (Symm), (Triang), (Cont) and (Max) follows from the fact that $d^{\mathcal{B}}$ is a pseudometric. (NExp) follows from the fact that interpretations of symbols from the algebraic signature are nonexpansive (Lemma 10). Recall that $d^{\mathcal{B}} = d_{\mathcal{L}} \circ (\llbracket - \rrbracket \times \llbracket - \rrbracket)$. The soundness of (Top) follows from the fact that $d_{\mathcal{L}}$ is a 1-pseudometric. Additionally, for all axioms in the form $\vdash e \equiv_0 f$ it suffices to show that $\llbracket e \rrbracket = \llbracket f \rrbracket$. (SL1), (SL2), (SL3), (SL4), (1S), (S), (S1), (0S), (S0), (D1), (D2), (Unroll) and (Tight) are taken from Salomaa's axiomatisation of language equivalence of regular expressions [30] and thus both sides of those equations denote the same formal languages [43, Theorem 5.2]. For (λ-Pref) assume that the premise is satisfied in the model, that is $d_{\mathcal{L}}(\llbracket e \rrbracket, \llbracket f \rrbracket) \leq \varepsilon$. Let $\varepsilon' \geq \lambda \cdot \varepsilon$. We show the following:

$$
\begin{aligned}
d^{\mathcal{B}}(a\,;e, a\,;f) &= d_{\mathcal{L}}(\llbracket a\,;e \rrbracket, \llbracket a\,;f \rrbracket) && \text{(Def. of } d^{\mathcal{B}}\text{)} \\
&= \Phi_{\mathcal{L}}(d_L)(\llbracket a\,;e \rrbracket, \llbracket a\,;f \rrbracket) && \text{(d_L is a fixpoint of } \Phi_{\mathcal{L}}\text{)} \\
&= \max\{d_2(o_{\mathcal{L}}(a\,;e), o_{\mathcal{L}}(a\,;f)), \lambda \cdot \max_{a' \in A} d_{\mathcal{L}}(\llbracket a\,;e \rrbracket_{a'}, \llbracket a\,;f \rrbracket_{a'})\} \\
&= \lambda \cdot d_{\mathcal{L}}(\llbracket e \rrbracket, \llbracket f \rrbracket) && \text{(Def. of final automaton)} \\
&\leq \lambda \cdot \epsilon \leq \varepsilon' && \text{(Assumptions)}
\end{aligned}
$$

Finally, (SL5) is derivable from other axioms[2]. If $\varepsilon = \max(\varepsilon, \varepsilon')$ then $\{e \equiv_\epsilon g\} \vdash e \equiv_{\max(\varepsilon,\varepsilon')} g$ holds by (Assum). If $\epsilon < \max(\varepsilon, \varepsilon')$, then we can derive the quantitative judgement above using (Max). By a similar line of reasoning, we can show that $\{f \equiv_{\epsilon'} h\} \vdash f \equiv_{\max(\varepsilon,\varepsilon')} h$. Finally, using (Cut) and (NExp), we can show that $\{e \equiv_\varepsilon g, f \equiv_{\varepsilon'} h\} \vdash e + f \equiv_{\max(\varepsilon,\varepsilon')} g + h$ as desired. ◀

[2] We included (SL5) as an axiom to highlight the similarity of our inference system with axiomatisations of language equivalence of regular expressions [30, 19] containing the axioms of semilattices with bottom. In the previous work [22], (SL1 – SL5) are precisely the axioms of *Quantitative Semilattices with zero* axiomatising the Hausdorff distance.

We now revisit the example from Section 1. Recall that states marked as initial of the left and middle automata can be respectively represented as a^* and $a+1$. The shortest word distinguishing languages representing those expressions is aa. If we fix $\lambda = \frac{1}{2}$, then $d_{\mathcal{L}}([\![a^*]\!], [\![a+1]\!]) = \frac{1}{4} = \left(\frac{1}{2}\right)^{|aa|}$. We can derive this distance through the means of axiomatic reasoning using the quantitative equational theory REG in the following way:

▶ **Example 12.**

$$\begin{aligned}
\vdash a^* &\equiv_1 0 && \text{(Top)}\\
\vdash a\,;a^* &\equiv_{\frac{1}{2}} a\,;0 && (\lambda\text{-Pref})\\
\vdash a\,;a^*+1 &\equiv_{\frac{1}{2}} a\,;0+1 && (\vdash 1 \equiv_0 1 \text{ and SL5})\\
\vdash a^* &\equiv_{\frac{1}{2}} 1 && \text{(Triang, Unroll, S0 and SL4)}\\
\vdash a\,;a^* &\equiv_{\frac{1}{4}} a\,;1 && (\lambda\text{-Pref})\\
\vdash a\,;a^*+1 &\equiv_{\frac{1}{4}} a\,;1+1 && (\vdash 1 \equiv_0 1 \text{ and SL5})\\
\vdash a^* &\equiv_{\frac{1}{4}} a+1 && \text{(Triang, Unroll and S1)}
\end{aligned}$$

(The lack of) the fixpoint axiom. Traditionally, completeness of inference systems for behavioural equivalence of languages of expressions featuring recursive constructs such as Kleene star or μ-recursion [24] rely crucially on fixpoint introduction rules. Those allow showing that an expression is provably equivalent to a looping construct if it exhibits some form of self-similarity, typically subject to productivity constraints. As an illustration, Salomaa's axiomatisation of language equivalence of regular expressions incorporates the following inference rule:

$$\frac{g \equiv e\,;g+f \qquad \epsilon \notin [\![e]\!]}{g \equiv e^*\,;f} \tag{2}$$

The side condition on the right states that the loop body is *productive*, that is a deterministic automaton corresponding to an expression e cannot immediately reach acceptance without performing any transitions. This is simply equivalent to the language $[\![e]\!]$ not containing the empty word. It would be reasonable for one to expect REG to contain a similar rule to be complete, especially since it should be able to prove language equivalence of regular expressions (by proving that they are in distance zero from each other). Furthermore, all axioms of Salomaa except Equation (2) are contained in REG as rules for distance zero.

It turns out that in the presence of the infinitary continuity (Cont) rule of quantitative deduction systems and the (λ-Pref) axiom of REG, the Salomaa's inference rule (Equation (2)) becomes a derivable fact for distance zero. First of all, one can show that (λ-Pref) can be generalised from prepending single letters to prepending any regular expression satisfying the side condition from Equation (2).

▶ **Lemma 13.** *Let $e, f, g \in \mathsf{Exp}$, such that $\epsilon \notin [\![e]\!]$. Then, $\{f \equiv_\varepsilon g\} \vdash e\,;f \equiv_{\varepsilon'} e\,;g$ is derivable using the axioms of REG for all $\varepsilon' \geq \lambda \cdot \varepsilon$.*

With the above lemma in hand, one can inductively show that if $g \equiv_0 e\,;g+f$ and $\epsilon \notin [\![e]\!]$, then g gets arbitrarily close to $e^*\,;f$. Intuitively, the more we unroll the loop in $e^*\,;f$ using (Unroll) and the more we unroll the definition of g, then the closer both expressions become.

▶ **Lemma 14.** *Let $e, f, g \in \mathsf{Exp}$, such that $\epsilon \notin [\![e]\!]$ and let $n \in \mathbb{N}$. Then, $\{g \equiv_0 e\,;\, g + f\} \vdash g \equiv_\varepsilon e^*\,;\, f$ is derivable using the axioms of* REG *for all $\varepsilon \geq \lambda^n$.*

Having the result above, we can now use the infinitary (Cont) rule capturing the limiting property of decreasing chain of overapproximations to the distance and show the derivability of Salomaa's inference rule.

▶ **Lemma 15.** *Let $e, f, g \in \mathsf{Exp}$, such that $\epsilon \notin [\![e]\!]$. Then, $\{g \equiv_0 e\,;\, g + f\} \vdash g \equiv_0 e^*\,;\, f$ is derivable using the axioms of REG.*

Proof. To deduce that $\vdash g \equiv_0 e^*\,;\, f$ using (Cont) it suffices to show that $\vdash g \equiv_\varepsilon e^*\,;\, f$ for all $\varepsilon > 0$. To do so, pick an arbitrary $\varepsilon > 0$ and let $N = \lceil \log_\lambda \varepsilon \rceil$. Observe that $\lambda^N = \lambda^{\lceil \log_\lambda \varepsilon \rceil} \leq \lambda^{\log_\lambda \varepsilon} = \varepsilon$. Because of Lemma 14 we have that $\vdash g \equiv_\varepsilon e^*\,;\, f$, which completes the proof. ◀

5 Completeness

We now move on to the central result of this paper, which is the completeness of REG with respect to the shortest-distinguishing-word metric on languages denoting regular expressions. We use the strategy from the proof of completeness of quantitative axiomatisation of probabilistic bisimilarity distance [2]. It turns out that the results from [2] rely on properties that are not unique to the Kantorovich/Wassertstein lifting and can be also established for instances of the abstract coalgebraic framework [5].

The heart of our argument relies on the fact that the distance between languages denoting regular expressions can be calculated in a simpler way than applying the Knaster-Tarski fixpoint theorem while looking at the infinite-state final automaton of all formal languages over some fixed alphabet. In particular, regular expressions denote the behaviour of finite-state deterministic automata. Since automata homomorphisms are non-expansive mappings, the distance between languages $[\![e]\!]$ and $[\![f]\!]$ of some arbitrary regular expressions $e, f \in \mathsf{Exp}$ is the same as the distance between states in some DFA whose languages corresponds to $[\![e]\!]$ and $[\![f]\!]$. To be precise, we will look at the finite subautomaton $\langle [e]_{\doteq}, [f]_{\doteq} \rangle_{\mathcal{Q}}$ of the $\doteq$ quotient of the Brzozowski automaton. The reason we care about deterministic finite automata is that it turns out that one can calculate the behavioural distance between two states through iterative approximation from above, which can be also derived axiomatically using the (Cont) rule of quantitative deduction systems. We start by showing how this simplification works and then we move on to establishing completeness.

Behavioural distance on finite-state automata. Consider a deterministic automaton $\mathcal{M} = (M, \langle o_M, t_M \rangle)$. The least fixpoint of a monotone endomap $\Phi_{\mathcal{M}} : D_M \to D_M$ on the complete lattice of 1-pseudometrics on the set M results in $d_{\mathcal{M}}$, which is a behavioural pseudometric on the states of the automaton $\mathcal{M}$. It is noteworthy that $\Phi_{\mathcal{M}}$ exhibits two generic properties. Firstly, $\Phi_{\mathcal{M}}$ behaves well within the Banach space structure defined by the supremum norm.

▶ **Lemma 16.** *$\Phi_{\mathcal{M}} : D_M \to D_M$ is nonexpansive with respect to the supremum norm. In other words, for all $d, d' \in D_M$ we have that $\|\Phi_{\mathcal{M}}(d') - \Phi_{\mathcal{M}}(d)\| \leq \|d' - d\|$.*

Proof. We can safely assume that $d \sqsubseteq d'$, as other case will be symmetric. It sufices to show that for all $m, m' \in M$, $\Phi_{\mathcal{M}}(d')(m, m') - \Phi_{\mathcal{M}}(d)(m, m') \leq \|d' - d\|$. First, let's consider the case when $o_M(m) \neq o_M(m')$ and hence $d_2(m, m') = 1$. In such a scenario, it holds that $\Phi_{\mathcal{M}}(d')(m, m') - \Phi_{\mathcal{M}}(d)(m, m') = 0 \leq \|d' - d\|$. From now on, we will assume that $o_M(m) = o_M(m)$ and hence $d_2(m, m') = 0$. We have the following

$$\begin{aligned}
\Phi_{\mathcal{M}}(d')(m,m') - \Phi_{\mathcal{M}}(d)(m,m') &= \lambda \cdot \max_{a\in A} d'(m_a,m'_a) - \lambda \cdot \max_{a\in A} d(m_a,m'_a) \\
&= \lambda \cdot \left(\max_{a\in A} d'(m_a,m'_a) - \max_{a\in A} d(m_a,m'_a) \right) \\
&\leq \lambda \cdot \left(\max_{a\in A} \{d'(m_a,m'_a) - d(m_a,m'_a)\} \right) \\
&\leq \lambda \cdot \sup_{n,n'\in M} \{d'(n,n') - d(n,n')\} \\
&= \lambda \cdot \|d' - d\| \leq \|d' - d\|
\end{aligned}$$ ◀

Secondly, it turns out that $\Phi_{\mathcal{M}}$ has only one fixpoint. This means that if we want to calculate $d_{\mathcal{M}}$ it suffices to look at any fixpoint of $\Phi_{\mathcal{M}}$. This will enable a simpler characterisation, than the one given by the Knaster-Tarski fixpoint theorem.

▶ **Lemma 17.** *$\Phi_{\mathcal{M}}$ has a unique fixed point.*

Proof. Let $d, d' \in D_M$ be two fixed points of $\Phi_{\mathcal{M}}$, that is $\Phi_{\mathcal{M}}(d) = d$ and $\Phi_{\mathcal{M}}(d') = d'$. We can safely assume that $d \sqsubseteq d'$, as the other case is symmetric. We wish to show that $d = d'$ and to do so we will use proof by contradiction.

Assume that $d \neq d'$, and hence there exist $m, m' \in M$, such that $d(m,m') < d'(m,m')$ and $\|d' - d\| = d'(m,m') - d(m,m') \neq 0$. First, consider the case when $o_M(m) \neq o_M(m')$. In such a case both $d(m,m')$ and $d(m,m')$ are equal to 1 and hence $\|d' - d\| = 0$, which leads to contradiction. From now, we can safely assume that $o_M(m) = o_M(m')$. Through an identical line of reasoning to the proof of Lemma 16, we can show that $\|\Phi_{\mathcal{M}}(d') - \Phi_{\mathcal{M}}(d)\| \leq \lambda \cdot \|d' - d\|$. Since both d and d' are fixed points, this would mean that $\|d' - d\| \leq \lambda \cdot \|d' - d\|$. Since $\lambda \in]0,1[$, this would imply that $\|d' - d\| = 0$ leading again to contradiction. ◀

In particular, we will rely on the characterisation given by the Kleene fixpoint theorem [31, Theorem 2.8.5], which allows to obtain the greatest fixpoint of an endofunction on the lattice as the infimum of the decreasing sequence of finer approximations obtained by repeatedly applying the function to the top element of the lattice.

▶ **Theorem 18** (Kleene fixpoint theorem)**.** *Let $(X, \sqsubseteq)$ be a complete lattice with a top element $\top$ and $f : X \to X$ an endofunction that is ω-cocontinuous or in other words for any decreasing chain $\{x_i\}_{i\in\mathbb{N}}$ it holds that $\inf_{i\in\mathbb{N}}\{f(x_i)\} = f\left(\inf_{i\in\mathbb{N}} x_i\right)$. Then, f possesses a greatest fixpoint, given by $\mathrm{gfp}(f) = \inf_{i\in\mathbb{N}}\{f^{(i)}(\top)\}$ where $f^{(n)}$ denotes n-fold self-composition of f given inductively by $f^{(0)}(x) = x$ and $f^{(n+1)}(x) = f^{(n+1)}(f(x))$ for all $x \in X$.*

The theorem above requires the endomap to be ω-cocontinuous. Luckily, it is the case for $\Phi_{\mathcal{M}}$ if we restrict our attention to DFA. To show that, we directly follow the line of reasoning from [2, Lemma 5.6] generalising the similar line of reasoning for ω-continuity from [37, Theorem 1]. First, using Lemma 6 we show that decreasing chains of pseudometrics over a finite set converge to their infimum. That result is a minor re-adaptation of [37, Theorem 1] implicitly used in [2, Lemma 5.6].

▶ **Lemma 19.** *Let $\{d_i\}_{i\in\mathbb{N}}$ be an infinite descending chain in the lattice $(D_X, \sqsubseteq)$, where X is a finite set. The sequence $\{d_i\}_{i\in\mathbb{N}}$ converges (in the sense of convergence in the Banach space) to $d(x,y) = \inf_{i\in\mathbb{N}} d_i(x,y)$.*

Proof. Let $\varepsilon > 0$ and let $x, y \in X$. Since $d(x,y) = \inf_{i\in\mathbb{N}} d_i(x,y)$ there exists an index $m_{x,y} \in \mathbb{N}$ such that for all $n \geq m_{x,y}$, $|d_n(x,y) - d(x,y)| < \varepsilon$. Now, let $N = \max\{m_{x,y} \mid x, y \in X\}$. This is well-defined because X is finite. Therefore, for all $n \geq N$ and $x, y \in X$, $|d_n(x,y) - d(x,y)| < \varepsilon$ and hence $\|d_n - d\| < \varepsilon$. ◀

We can now use the above to show the desired property, by re-adapting [37, Theorem 1].

▶ **Lemma 20.** *If $\mathcal{M}$ is a deterministic finite automaton, then $\Phi_{\mathcal{M}}$ is ω-cocontinuous.*

Proof. By Lemma 19, the chain $\{d_i\}_{i\in\mathbb{N}}$ converges to $\inf_{i\in\mathbb{N}} d_i$. Since $\Phi_{\mathcal{M}}$ is nonexpansive (Lemma 16) it is also continuous (in the sense of the Banach space continuity) and therefore $\{\Phi_{\mathcal{M}}(d_i)\}_{i\in\mathbb{N}}$ converges to $\Phi_{\mathcal{M}}(\inf_{i\in\mathbb{N}} d_i)$. Recall that $\Phi_{\mathcal{M}}$ is monotone, which makes $\{\Phi_{\mathcal{M}}(d_i)\}_{i\in\mathbb{N}}$ into a chain, which by Lemma 6 and Lemma 19 converges to $\inf_{i\in\mathbb{N}}\{\Phi_{\mathcal{M}}(d_i)\}$. Since limit points are unique, $\inf_{i\in\mathbb{N}}\{\Phi_{\mathcal{M}}(d_i)\} = \Phi_{\mathcal{M}}(\inf_{i\in\mathbb{N}} d_i)$. ◀

We can combine the preceding results and provide a straightforward characterisation of the distance between languages represented by arbitrary regular expressions, denoted as $e, f \in \mathsf{Exp}$. Utilising a simple argument based on Proposition 7, which asserts that automata homomorphisms are nonexpansive, one can demonstrate that the distance between $[\![e]\!]$ and $[\![f]\!]$ in the final automaton is equivalent to the distance between $[e]_{\doteq}$ and $[f]_{\doteq}$ in $\langle [e]_{\doteq}, [f]_{\doteq}\rangle_{\mathcal{Q}}$. This is the least subautomaton of $\mathcal{Q}$ that contains the derivatives (modulo $\doteq$) reachable from $[e]_{\doteq}$ and $[f]_{\doteq}$. Importantly, this automaton is finite (Lemma 3), allowing us to apply the Kleene fixpoint theorem to calculate the distance.

Let $\Psi^{(0)}_{e,f}$ denote the discrete metric on the set $\langle [e]_{\doteq}, [f]_{\doteq}\rangle_{\mathcal{Q}}$ (the top element of the lattice of pseudometrics over that set). Define $\Psi^{(n+1)}_{e,f} = \Phi_{\langle [e]_{\doteq},[f]_{\doteq}\rangle_{\mathcal{Q}}}(\Psi^{(n)}_{e,f})$. Additionally, leveraging the fact that infima of decreasing chains are calculated pointwise (Lemma 6), we can conclude with the following:

▶ **Lemma 21.** *For all $e, f \in \mathsf{Exp}$, the underlying pseudometric of the quantitative algebra $\mathcal{B}$ can be given by $d^{\mathcal{B}}(e,f) = \inf_{i\in\mathbb{N}} \left\{ \Psi^{(i)}_{e,f}([e]_{\doteq}, [f]_{\doteq}) \right\}$*

In simpler terms, we have demonstrated that the behavioural distance between a pair of arbitrary regular expressions can be calculated as the infimum of decreasing approximations of the actual distance from above. Alternatively, one could calculate the same distance as the supremum of increasing approximations from below using the Kleene fixpoint theorem for the least fixpoint. We chose the former approach because our proof of completeness relies on the (Cont) rule of quantitative deduction systems. This rule essentially states that to prove two terms are at a specific distance, we should be able to prove that for all approximations of that distance from above. This allows us to replicate the fixpoint calculation through axiomatic reasoning.

Completeness result. We start by recalling that regular expressions satisfy a certain decomposition property, stating that each expression can be reconstructed from its small-step semantics, up to $\equiv_0$. This property, often referred to as the fundamental theorem of Kleene Algebra/regular expressions (in analogy with the fundamental theorem of calculus and following the terminology of Rutten [27] and Silva [33]) is useful in further steps of the proof of completeness.

▶ **Theorem 22** (Fundamental Theorem). *For any $e \in \mathsf{Exp}$, $\vdash e_i \equiv_0 \sum_{a\in A} a\,;(e_i)_a + o_{\mathcal{R}}(e_i)$ is derivable using the axioms of REG.*

The theorem above makes use of the n-ary generalised sum operator, which is well defined because of (SL1-SL4) axioms of REG. Let's now say that we are interested in the distance between some expressions $e, f \in \mathsf{Exp}$. As mentioned before, we will rely on $\langle [e]_{\doteq}, [f]_{\doteq} \rangle_{\mathcal{Q}}$, the least subautomaton of the $\doteq$ quotient of the Brzozowski automaton containing states reachable from $[e]_{\doteq}$ and $[f]_{\doteq}$. Recall that by Lemma 3 its state space is finite. It turns out that the approximations from above (from Lemma 21) to the distance between any pair of states in that automaton can be derived through the means of axiomatic reasoning.

▶ **Lemma 23.** *Let $e, f \in \mathsf{Exp}$ be arbitrary regular expressions and let $[g]_{\doteq}, [h]_{\doteq} \in \langle [e]_{\doteq}, [f]_{\doteq} \rangle_{\mathcal{Q}}$. For all $i \in \mathbb{N}$, and $\varepsilon \geq \Psi^{(i)}_{e,f}([g]_{\doteq}, [h]_{\doteq})$, one can derive $\vdash g \equiv_{\varepsilon} h$ using the axioms of* REG.

Proof. We proceed by induction on i. For the base case, observe that $\Psi^{(0)}_{e,f}$ is the discrete 1-pseudometric on the set $\langle [e]_{\doteq}, [f]_{\doteq} \rangle_{\mathcal{Q}}$ such that $\Psi^{(0)}_{e,f}([g]_{\doteq}, [h]_{\doteq}) = 0$ if and only if $g \doteq h$, or otherwise $\Psi^{(0)}_{e,f}([g]_{\doteq}, [h]_{\doteq}) = 1$. In the first case, we immediately have that $g \equiv_0 h$, because $\doteq$ is contained in distance zero axioms of REG. In the latter case, we can just use (Top), to show that $g \equiv_1 h$. Then, in both cases, we can apply (Max) to obtain $\vdash g \equiv_{\varepsilon} h$, since $\varepsilon \geq \Psi^{(0)}_{e,f}([g]_{\doteq}, [h]_{\doteq})$. For the induction step, let $i = j + 1$ and derive the following:

$$\begin{aligned}
&\varepsilon \geq \Psi^{(j+1)}_{e,f}([g]_{\doteq}, [h]_{\doteq}) \iff \varepsilon \geq \Phi_{\langle [e]_{\doteq}, [f]_{\doteq} \rangle_{\mathcal{Q}}} \left(\Psi^{(j)}_{e,f} \right) ([g]_{\doteq}, [h]_{\doteq}) && \text{(Def. of } \Psi^{j+1}_{e,f}\text{)} \\
&\iff \varepsilon \geq \max \left\{ d_2(o_{\mathcal{Q}}([g]_{\doteq}), o_{\mathcal{Q}}([h]_{\doteq})), \lambda \cdot \max_{a \in A} \left\{ \Psi^{(j)}_{e,f}([g]_{\doteq a}, [h]_{\doteq a}) \right\} \right\} && \text{(Def. of } \Phi\text{)} \\
&\iff \varepsilon \geq \max \left\{ d_2\left(o_{\mathcal{R}}(g), o_{\mathcal{R}}(h)\right), \lambda \cdot \max_{a \in A} \left\{ \Phi^{(j)}_{\langle [e]_{\doteq}, [f]_{\doteq} \rangle_{\mathcal{Q}}}([(g)_a]_{\doteq}, [(h)_a]_{\doteq}) \right\} \right\} \\
& && \text{(Def. of } \mathcal{Q}\text{)} \\
&\iff \varepsilon \geq d_2(o_{\mathcal{R}}(g), o_{\mathcal{R}}(h)) \text{ and for all } a \in A,\ \varepsilon \cdot \lambda^{-1} \geq \Psi^{(j)}_{e,f}([(g)_a]_{\doteq}, [(h)_a]_{\doteq})
\end{aligned}$$

Firstly, since d_2 is the discrete 1-pseudometric on the set $\{0, 1\}$, we can use (Refl) or (Top) depending on whether $o_{\mathcal{R}}(g) = o_{\mathcal{R}}(h)$ and then apply (Max) to derive $\vdash o_{\mathcal{R}}(g) \equiv_{\varepsilon} o_{\mathcal{R}}(h)$.

Let $a \in A$. We will show that $\vdash a\,;(g)_a \equiv_{\varepsilon} a\,;(h)_a$. Since $\varepsilon \cdot \lambda^{-1}$ is not guaranteed to be rational, we cannot immediately apply the induction hypothesis. Instead, we rely on (Cont) rule. First, pick an arbitrary rational ε' strictly greater than ε and fix $\{r_n\}_{n \in \mathbb{N}}$ to be any decreasing sequence of rationals that converges to λ^{-1}. Let r_N be an element of that sequence such that $\varepsilon' \geq \lambda \cdot \varepsilon \cdot r_N$. It is always possible to pick such element because $\{\lambda \cdot r_n\}_{n \in \mathbb{N}}$ is a decreasing sequence that converges to 1 and $\varepsilon' > \varepsilon$. Since $\epsilon \cdot r_N \geq \epsilon \cdot \lambda^{-1}$ and $\varepsilon \cdot r_N \in \mathbb{Q}^+$, we can use the induction hypothesis and derive $\vdash (g)_a \equiv_{\epsilon \cdot r_N} (h_a)$. Then, by ($\lambda$-Pref) axiom we have that $\vdash a\,;(g)_a \equiv_{\varepsilon'} a\,;(h)_a$. Since we have shown it for arbitrary $\varepsilon' > \varepsilon$, by (Cont) rule we have that $\vdash a\,;(g)_a \equiv_{\varepsilon} a\,;(h)_a$. Using (SL5), we can combine all subexpressions involving the output and transition derivatives into the following:

$$\vdash \sum_{a \in A} a\,;(g)_a + o_{\mathcal{R}}(g) \equiv_{\varepsilon} \sum_{a \in A} a\,;(h)_a + o_{\mathcal{R}}(h)$$

Since both sides are normal forms of g and h existing because of Theorem 22, we can apply (Triang) on both sides and obtain $\vdash g \equiv_{\varepsilon} h$ thus completing the proof. ◀

At this point, we have done all the hard work, and establishing completeness involves a straightforward argument that utilises the (Cont) rule and the lemma above.

▶ **Theorem 24** (Completeness). *For any $e, f \in \mathsf{Exp}$ and $\varepsilon \in \mathbb{Q}^+$, if $\models_{\mathcal{B}} e \equiv_{\varepsilon} f$, then $\vdash e \equiv_{\varepsilon} f \in$ REG*

Proof. Assume that $\models_{\mathcal{B}} e \equiv_\varepsilon f$, which by the definition of $\models_{\mathcal{B}}$ is equivalent to $d^{\mathcal{B}}(e,f) \leq \varepsilon$. In order to use (Cont) axiom to derive $\vdash e \equiv_\varepsilon f$, we need to be able to show $\vdash e \equiv_{\varepsilon'} f$ for all $\varepsilon' > \varepsilon$. Because of iterative characterisation of $d^{\mathcal{B}}$ from Lemma 21, we have that $\inf_{i\in I}\{\Psi^{(i)}_{e,f}([e]_{\doteq},[f]_{\doteq})\} < \varepsilon'$. Since ε' is strictly above the infimum of the descending chain of approximants, there exists a point $i \in \mathbb{N}$, such that $\varepsilon' > \Psi^{(i)}_{e,f}([e]_{\doteq},[f]_{\doteq})$. We can show this by contradiction.

Assume that for all $i \in \mathbb{N}$, $\varepsilon' \leq \Psi^{(i)}_{e,f}([e]_{\doteq},[f]_{\doteq})$. This would make ε' into the lower bound of the chain $\left\{\Psi^{(i)}_{e,f}([e]_{\doteq},[f]_{\doteq})\right\}_{i\in\mathbb{N}}$ and in such a case ε' would be less than or equal to the infimum of that chain, which by assumption is less than or equal to ε. By transitivity, we could obtain $\varepsilon' \leq \varepsilon$. Since $\varepsilon' > \varepsilon$, by antisymmetry we could derive that $\varepsilon' = \varepsilon$, which would lead to the contradiction.

Using the fact shown above, we can use Lemma 23 to obtain $\vdash e \equiv_{\varepsilon'} f \in \mathsf{REG}$ for any $\varepsilon' > \varepsilon$, which completes the proof. ◀

In simpler terms, the (Cont) rule enables us to demonstrate that two terms are at a specific distance by examining all strict overapproximations of that distance. Due to the iterative nature outlined in Lemma 21, this implies that we only need to consider finite approximants used in the Kleene fixpoint theorem. Each of those finite approximants can be axiomatically derived using Lemma 23.

6 Discussion

We have presented a sound and complete axiomatisation of the shortest-distinguishing word distance between languages representing regular expressions through a quantitative analogue of equational logic [22]. Before our paper, only axiomatised behavioural distances of probabilistic/weighted transition systems existed, through (variants of) the Kantorovich/Wasserstein lifting [21, 12, 4, 2, 1], while we looked at a behavioural distance obtained through a more general coalgebraic framework [5]. Outside of the coalgebra community, the shortest-distinguishing word distance and its variants also appear in the model checking [20] and in the automata learning [14] literature.

We have followed the strategy for proving completeness from [2]. The interesting insight about that strategy is that it relies on properties that are not exclusive to distances obtained through the Kantorovich/Wasserstein lifting and can be established for notions of behavioural distance for other kinds of transition systems stemming from the coalgebraic framework. In particular, one needs to show that the monotone map on the lattice of pseudometrics used in defining the distance of finite-state systems is nonexpansive with respect to the sup norm (and hence ω-cocontinuous) and has a unique fixpoint, thus allowing to characterise the behavioural distance as the greatest fixpoint obtained through the Kleene fixpoint theorem. This point of view allows one to reconstruct the fixpoint calculation in terms of axiomatic manipulation involving the (Cont) rule, eventually leading to completeness.

We have additionally observed that in the presence of the infinitary (Cont) rule and the (λ-Pref) axiom, there is no need for a fixpoint rule, which is common place in all axiomatisations of regular expressions but also in other work on distances. In particular, the previous work on axiomatising a discounted probabilistic bisimilarity distance from [2] includes both (λ-Pref) and the fixpoint introduction rule, but its proof of completeness [2, Theorem 6.4] does not involve the fixpoint introduction rule at any point. We are highly confident that in the case of that axiomatisation, the fixpoint introduction rule could be derived from other axioms in a similar fashion to the way we derived Salomaa's rule for introducing the Kleene star [30]. Additionally, we are interested in how much this argument relates to the recent study of fixpoints in quantitative equational theories [23].

Moreover, the axiomatisations from [1, 2] rely on a slight modification of quantitative equational theories, which drop the requirement of all operations from the signature to be nonexpansive. This is dictated by the fact that the interpretation of μ-recursion in Stark and Smolka's probabilistic process algebra [35] can increase the behavioural distances in the case of unguarded recursion, while in regular expressions recursive behaviour is introduced through Kleene's star, whose interpretation is non-expansive with respect to the shortest-distinguishing-word distance. This allowed us to fit instantly into the original framework of quantitative equational theories. The earlier work [4] focusing on Markov processes [8] also relies on quantitative equational theories, but its syntax does not involve any recursive primitives. Instead, the recursive behaviour is introduced through Cauchy completion of a pseudometric induced by the axioms. The earliest works on axiomatising behavioural distances of weighted [21] and probabilistic [12] transition systems, studied before the introduction of quantitative equational theories, rely on ad-hoc inference systems that cannot be easily generalised.

The pioneering works [13, 39, 37, 40, 38, 3] laid foundations for behavioural (pseudo)metrics of various flavours of probabilistic transition systems. The coalgebraic point of view [5] allowed to generalise these ideas to a wide range of transition systems by moving from the Kantorovich/Wasserstein lifting to the abstract setting of lifting endofunctors from the category of sets to the category of pseudometric spaces. Building upon this theory, further lines of work were dedicated to asymmetric distances (called hemimetrics) through the theory of quasi-lax liftings [44], fuzzy analogues of Hennessy-Milner logic characterising behavioural distance [17, 6], fibrational generalisations involving quantale-enriched categories [7], up-to techniques allowing for efficient approximation of behavioural distances [9] and quantitative analogues of van Glabbek's linear-time branching-time spectrum [15].

In this paper, we have focused on the simplest and most intuitive instantiation of the coalgebraic framework in the case of deterministic automata, but the natural next step would be to generalise our results to a wider class of transition systems. A good starting point could be to consider coalgebras for *polynomial* endofunctors, in the fashion of the framework of *Kleene Coalgebra* [33]. Alternatively, it would be interesting to look at recent work on a family of process algebras parametric on an equational theory representing the branching constructs [32] and study its generalisations to quantitative equational theories. A related and interesting avenue for future work are equational axiomatisations of behavioural equivalence of Guarded Kleene Algebra with Tests (GKAT) [34, 32] and its probabilistic extension (ProbGKAT) [29], whose completeness results rely on a powerful uniqueness of solutions axiom (UA). The soundness of UA in both cases is shown through an involved argument relying on equipping the transition systems giving the operational semantics with a form of behavioural distance and showing that recursive specifications describing finite-state systems correspond to certain contractive mappings. It may be more sensible, particularly for ProbGKAT to consider quantitative axiomatisations in the first place and give the proofs of completeness through the pattern explored in this paper.

References

1 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. Complete Axiomatization for the Total Variation Distance of Markov Chains. In Sam Staton, editor, *Proceedings of the Thirty-Fourth Conference on the Mathematical Foundations of Programming Semantics, MFPS 2018, Dalhousie University, Halifax, Canada, June 6-9, 2018*, volume 341 of *Electronic Notes in Theoretical Computer Science*, pages 27–39. Elsevier, 2018. doi:10.1016/J.ENTCS.2018.03.014.

2 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. A Complete Quantitative Deduction system for the Bisimilarity Distance on Markov Chains. *Log. Methods Comput. Sci.*, 14(4), 2018. doi:10.23638/LMCS-14(4:15)2018.

3 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. Converging from branching to linear metrics on Markov chains. *Math. Struct. Comput. Sci.*, 29(1):3–37, 2019. doi:10.1017/S0960129517000160.

4 Giorgio Bacci, Radu Mardare, Prakash Panangaden, and Gordon D. Plotkin. An Algebraic Theory of Markov Processes. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 679–688. ACM, 2018. doi:10.1145/3209108.3209177.

5 Paolo Baldan, Filippo Bonchi, Henning Kerstan, and Barbara König. Coalgebraic Behavioral Metrics. *Log. Methods Comput. Sci.*, 14(3), 2018. doi:10.23638/LMCS-14(3:20)2018.

6 Harsh Beohar, Sebastian Gurke, Barbara König, and Karla Messing. Hennessy-Milner Theorems via Galois Connections. In Bartek Klin and Elaine Pimentel, editors, *31st EACSL Annual Conference on Computer Science Logic, CSL 2023, February 13-16, 2023, Warsaw, Poland*, volume 252 of *LIPIcs*, pages 12:1–12:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CSL.2023.12.

7 Harsh Beohar, Sebastian Gurke, Barbara König, Karla Messing, Jonas Forster, Lutz Schröder, and Paul Wild. Expressive Quantale-valued Logics for Coalgebras: an Adjunction-based Approach. *CoRR*, abs/2310.05711, 2023. doi:10.48550/arXiv.2310.05711.

8 Richard Blute, Josée Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for Labelled Markov Processes. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*, pages 149–158. IEEE Computer Society, 1997. doi:10.1109/LICS.1997.614943.

9 Filippo Bonchi, Barbara König, and Daniela Petrisan. Up-To Techniques for Behavioural Metrics via Fibrations. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, volume 118 of *LIPIcs*, pages 17:1–17:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.CONCUR.2018.17.

10 Janusz A. Brzozowski. Derivatives of Regular Expressions. *J. ACM*, 11(4):481–494, 1964. doi:10.1145/321239.321249.

11 Stanley Burris and H P Sankappanavar. *A Course in Universal Algebra*. Lecture Notes in Statistics. Springer, New York, NY, November 1981.

12 Pedro R. D'Argenio, Daniel Gebler, and Matias David Lee. Axiomatizing Bisimulation Equivalences and Metrics from Probabilistic SOS rules. In Anca Muscholl, editor, *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8412 of *Lecture Notes in Computer Science*, pages 289–303. Springer, 2014. doi:10.1007/978-3-642-54830-7_19.

13 Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labelled markov processes. *Theor. Comput. Sci.*, 318(3):323–354, 2004. doi:10.1016/J.TCS.2003.09.013.

14 Tiago Ferreira, Gerco van Heerdt, and Alexandra Silva. Tree-Based Adaptive Model Learning. In Nils Jansen, Mariëlle Stoelinga, and Petra van den Bos, editors, *A Journey from Process Algebra via Timed Automata to Model Learning - Essays Dedicated to Frits Vaandrager on the Occasion of His 60th Birthday*, volume 13560 of *Lecture Notes in Computer Science*, pages 164–179. Springer, 2022. doi:10.1007/978-3-031-15629-8_10.

15 Jonas Forster, Lutz Schröder, and Paul Wild. Quantitative Graded Semantics and Spectra of Behavioural Metrics. *CoRR*, abs/2306.01487, 2023. doi:10.48550/arXiv.2306.01487.

16 Alessandro Giacalone, Chi-Chang Jou, and Scott A. Smolka. Algebraic Reasoning for Probabilistic Concurrent Systems. In Manfred Broy and Cliff B. Jones, editors, *Programming concepts and methods: Proceedings of the IFIP Working Group 2.2, 2.3 Working Conference on Programming Concepts and Methods, Sea of Galilee, Israel, 2-5 April, 1990*, pages 443–458. North-Holland, 1990.

17 Sergey Goncharov, Dirk Hofmann, Pedro Nora, Lutz Schröder, and Paul Wild. Kantorovich Functors and Characteristic Logics for Behavioural Distances. In Orna Kupferman and Pawel Sobocinski, editors, *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13992 of *Lecture Notes in Computer Science*, pages 46–67. Springer, 2023. doi:10.1007/978-3-031-30829-1_3.

18 John E. Hopcroft and Richard M. Karp. A Linear Algorithm for Testing Equivalence of Finite Automata, 1971. URL: https://api.semanticscholar.org/CorpusID:120207847.

19 Dexter Kozen. A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. *Inf. Comput.*, 110(2):366–390, 1994. doi:10.1006/INCO.1994.1037.

20 Marta Z. Kwiatkowska. A Metric for Traces. *Inf. Process. Lett.*, 35(3):129–135, 1990. doi:10.1016/0020-0190(90)90061-2.

21 Kim G. Larsen, Uli Fahrenberg, and Claus R. Thrane. Metrics for weighted transition systems: Axiomatization and complexity. *Theor. Comput. Sci.*, 412(28):3358–3369, 2011. doi:10.1016/J.TCS.2011.04.003.

22 Radu Mardare, Prakash Panangaden, and Gordon D. Plotkin. Quantitative Algebraic Reasoning. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 700–709. ACM, 2016. doi:10.1145/2933575.2934518.

23 Radu Mardare, Prakash Panangaden, and Gordon D. Plotkin. Fixed-points for Quantitative Equational Logics. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470662.

24 Robin Milner. A Complete Inference System for a Class of Regular Behaviours. *J. Comput. Syst. Sci.*, 28(3):439–466, 1984. doi:10.1016/0022-0000(84)90023-0.

25 David Michael Ritchie Park. Concurrency and Automata on Infinite Sequences. In *Theoretical Computer Science*, 1981. URL: https://api.semanticscholar.org/CorpusID:206841958.

26 Walter Rudin. *Functional Analysis*. International Series in Pure & Applied Mathematics. McGraw Hill Higher Education, Maidenhead, England, 2 edition, October 1990.

27 Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000. doi:10.1016/S0304-3975(00)00056-6.

28 Wojciech Różowski. A Complete Quantitative Axiomatisation of Behavioural Distance of Regular Expressions, 2024. arXiv:2404.13352.

29 Wojciech Różowski, Tobias Kappé, Dexter Kozen, Todd Schmid, and Alexandra Silva. Probabilistic Guarded KAT Modulo Bisimilarity: Completeness and Complexity. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 136:1–136:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.136.

30 Arto Salomaa. Two Complete Axiom Systems for the Algebra of Regular Events. *J. ACM*, 13(1):158–169, 1966. doi:10.1145/321312.321326.

31 Davide Sangiorgi. *Coinduction and the duality with induction*, pages 28–88. Cambridge University Press, 2011.

32 Todd Schmid, Wojciech Różowski, Alexandra Silva, and Jurriaan Rot. Processes Parametrised by an Algebraic Theory. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 132:1–132:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICALP.2022.132.

33 A.M Silva. *Kleene coalgebra*. PhD thesis, Radboud Universiteit Nijmegen, 2010.

34 Steffen Smolka, Nate Foster, Justin Hsu, Tobias Kappé, Dexter Kozen, and Alexandra Silva. Guarded Kleene algebra with tests: verification of uninterpreted programs in nearly linear time. *Proc. ACM Program. Lang.*, 4(POPL):61:1–61:28, 2020. doi:10.1145/3371129.

35 Eugene W. Stark and Scott A. Smolka. A complete axiom system for finite-state probabilistic processes. In Gordon D. Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 571–596. The MIT Press, 2000.

36 Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

37 Franck van Breugel. On behavioural pseudometrics and closure ordinals. *Inf. Process. Lett.*, 112(19):715–718, 2012. doi:10.1016/J.IPL.2012.06.019.

38 Franck van Breugel. Probabilistic bisimilarity distances. *ACM SIGLOG News*, 4(4):33–51, 2017. doi:10.1145/3157831.3157837.

39 Franck van Breugel and James Worrell. Towards Quantitative Verification of Probabilistic Transition Systems. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 421–432. Springer, 2001. doi:10.1007/3-540-48224-5_35.

40 Franck van Breugel and James Worrell. Approximating and computing behavioural distances in probabilistic transition systems. *Theor. Comput. Sci.*, 360(1-3):373–385, 2006. doi:10.1016/J.TCS.2006.05.021.

41 Rob J. van Glabbeek. The Linear Time-Branching Time Spectrum (Extended Abstract). In Jos C. M. Baeten and Jan Willem Klop, editors, *CONCUR '90, Theories of Concurrency: Unification and Extension, Amsterdam, The Netherlands, August 27-30, 1990, Proceedings*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990. doi:10.1007/BFB0039066.

42 Cédric Villani. *Optimal Transport*. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-540-71050-9.

43 Jana Wagemaker, Marcello M. Bonsangue, Tobias Kappé, Jurriaan Rot, and Alexandra Silva. Completeness and Incompleteness of Synchronous Kleene Algebra. In Graham Hutton, editor, *Mathematics of Program Construction - 13th International Conference, MPC 2019, Porto, Portugal, October 7-9, 2019, Proceedings*, volume 11825 of *Lecture Notes in Computer Science*, pages 385–413. Springer, 2019. doi:10.1007/978-3-030-33636-3_14.

44 Paul Wild and Lutz Schröder. Characteristic Logics for Behavioural Hemimetrics via Fuzzy Lax Extensions. *Log. Methods Comput. Sci.*, 18(2), 2022. doi:10.46298/LMCS-18(2:19)2022.

Homogeneity and Homogenizability: Hard Problems for the Logic SNP

Jakub Rydval
Technische Universität Wien, Austria

Abstract

The infinite-domain CSP dichotomy conjecture extends the finite-domain CSP dichotomy theorem to reducts of finitely bounded homogeneous structures. Every countable finitely bounded homogeneous structure is uniquely described by a universal first-order sentence up to isomorphism, and every reduct of such a structure by a sentence of the logic SNP. By Fraïssé's Theorem, testing the existence of a finitely bounded homogeneous structure for a given universal first-order sentence is equivalent to testing the amalgamation property for the class of its finite models. The present paper motivates a complexity-theoretic view on the classification problem for finitely bounded homogeneous structures. We show that this meta-problem is EXPSPACE-hard or PSPACE-hard, depending on whether the input is specified by a universal sentence or a set of forbidden substructures. By relaxing the input to SNP sentences and the question to the existence of a structure with a finitely bounded homogeneous expansion, we obtain a different meta-problem, closely related to the question of homogenizability. We show that this second meta-problem is already undecidable, even if the input SNP sentence comes from the Datalog fragment and uses at most binary relation symbols. As a byproduct of our proof, we also get the undecidability of some other properties for Datalog programs, e.g., whether they can be rewritten in the logic MMSNP, whether they solve some finite-domain CSP, or whether they define a structure with a homogeneous Ramsey expansion in a finite relational signature.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Computational complexity and cryptography

Keywords and phrases constraint satisfaction problems, finitely bounded, homogeneous, amalgamation property, universal, SNP, homogenizable

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.150

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2108.00452` [47]

Funding *Jakub Rydval*: This research was funded in whole or in part by the Austrian Science Fund (FWF) [I 5948]. For the purpose of Open Access, the authors have applied a CC BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission.

Acknowledgements The author thanks Manuel Bodirsky, Simon Knäuer, and Jakub Opršal for many inspiring discussions on the topic, and the anonymous reviewers for many helpful suggestions.

1 Introduction

Strict NP (SNP) is an expressive fragment of existential second-order logic and thus, by Fagin's Theorem, of the complexity class NP. If one only considers structures over a finite relational signature, then SNP can be obtained from the universal fragment of first-order logic simply by allowing existential quantification over relation symbols at the beginning of the quantifier prefix. In particular, universal first-order formulas themselves are SNP formulas. Despite the name, SNP already has the full power of NP, in the sense that every problem in NP is equivalent to a problem in SNP under polynomial-time reductions [30]. In addition, this logic class has many connections to *Constraint Satisfaction Problems* (CSPs), which we use as the primary source of motivation for the present article. The CSP of a

relational structure $\mathfrak{B}$, denoted by CSP($\mathfrak{B}$), is (the membership problem for) the class of all finite structures which homomorphically map to $\mathfrak{B}$. Every computational decision problem is polynomial-time equivalent to a CSP [12]. Many practically relevant problems like Boolean satisfiability or graph colorability can even be formulated as a *finite-domain* CSP, i.e., where the *template* $\mathfrak{B}$ can be chosen finite. The basic link from SNP to CSP is that every sentence of the monotone fragment of this logic defines a finite disjoint union of CSPs of (possibly infinite) relational structures [8]. There are, however, some more nuanced connections, such as the one that led to the formulation of the *Feder-Vardi conjecture*, now known as the finite-domain CSP dichotomy theorem [52]. In their seminal work [30], Feder and Vardi showed that the *Monotone Monadic* fragment of SNP (MMSNP) exhibits a dichotomy between P and NP-completeness if and only if the seemingly less complicated class of all finite-domain CSPs exhibits such a dichotomy,[1] they also conjectured the latter to be true. The logic class MMSNP contains all finite-domain CSPs, and many other interesting combinatorial problems, e.g., the problem of deciding whether the vertices of a given graph can be 2-coloured without obtaining any monochromatic triangle [42]. The Feder-Vardi conjecture was confirmed in 2017 independently by Bulatov and Zhuk [23, 51].

There is a yet unconfirmed generalization of the Feder-Vardi conjecture, to CSPs of *reducts of finitely bounded homogeneous structures*, formulated by Bodirsky and Pinsker in 2011 [18]. Here we refer to it as the *Bodirsky-Pinsker conjecture*. A structure is *finitely bounded* if it has a finite relational signature and the class of all finite structures embeddable into it is definable by a universal first-order sentence, and *homogeneous* if every isomorphism between two of its finite substructures extends to an automorphism. *Reducts* of such structures are obtained simply by removing some of the original relations. A prototypical example of a structure that satisfies both finite boundedness and homogeneity is $(\mathbb{Q}; <)$, the set of rational numbers equipped with the usual countable dense linear order without endpoints. It is a folklore fact that the class of reducts of finitely bounded homogeneous structures is closed under taking expansions of structures by first-order definable relations [8]. Roughly said, the condition imposed on the structures within the scope of the Bodirsky-Pinsker conjecture ensures that the CSP is in NP and that its template enjoys some of the universal-algebraic properties that have played an essential role in the proofs of the Feder-Vardi conjecture [6]. At the same time, it covers CSP-reformulations of many natural problems in qualitative reasoning, as well as all problems definable in MMSNP.

Every countable finitely bounded homogeneous structure is uniquely described by a universal first-order sentence up to isomorphism, and every reduct of such a structure by a sentence of the logic SNP. The CSPs of both kinds of structures are always definable in the monotone fragment of SNP. By Fraïssé's theorem, asking whether a given universal first-order sentence describes a finitely bounded homogeneous structure is equivalent to asking whether the class of its finite models has the *Amalgamation Property* (AP). This question has been considered many times in the context of the Lachlan-Cherlin classification programme for homogeneous structures [3, 36, 26], and is known to be decidable in the case of binary signatures [38, 15]. It also appears as an open problem in Bodirsky's book on infinite-domain constraint satisfaction [8]. Whether a given SNP sentence describes a reduct of a finitely bounded homogeneous structure is a different question, closely related to *homogenizability* [2, 4, 28, 35]. To the best of our knowledge, neither of the two questions is known to be decidable in general. Hence, it is unclear which CSPs actually fall within

[1] The correspondence between MMSNP and finite-domain CSP was initially only achieved up to randomized reductions, but it was later derandomized by Kun [37].

the scope of the Bodirsky-Pinsker conjecture. Besides CSPs, they are also relevant to other areas of theoretical computer science such as verification of database-driven systems [21] or description logics with concrete domains [39, 5]. Below, we state the two questions explicitly.

1. **The amalgamation meta-problem.** Given a universal sentence Φ over a finite relational signature, does there exist a finitely bounded homogeneous structure $\mathfrak{B}$ such that the finite models of Φ are precisely the finite substructures of $\mathfrak{B}$ up to isomorphism?
2. **The homogenizability meta-problem.** Given an SNP sentence Φ over a finite relational signature, does there exist a reduct $\mathfrak{B}$ of a finitely bounded homogeneous structure such that the finite models of Φ are precisely the finite substructures of $\mathfrak{B}$ up to isomorphism?

Contributions. In the present paper, we prove the intractability of the two meta-problems. More specifically, we show that the amalgamation meta-problem is EXPSPACE-hard (Theorem 6) or PSPACE-hard (Theorem 7), depending on the encoding of the input, and that the homogenizability meta-problem is undecidable (Theorem 13). Theorem 6 and Theorem 7 are proved in Section 3.1 by taking a proof-theoretic perspective on the AP for classes defined by universal Horn sentences. We show that, for some of these classes, the failures of the AP are in a 1:1 correspondence with the rejecting runs of certain Datalog programs verifying instances of the rectangle tiling problem. Here, by Datalog we mean the monotone Horn fragment of SNP. Theorem 13 is proved in Section 4, by analyzing model-theoretic properties of a very natural encoding of context-free grammars into Datalog sentences. As a byproduct of the proof, we also get the undecidability of some other properties for Datalog programs, e.g., whether they can be rewritten in the logic MMSNP, whether they solve some finite-domain CSP, or whether they define a structure with a homogeneous Ramsey expansion in a finite relational signature.

It is known that, from every finite structure $\mathfrak{A}$ over a finite relational signature one can construct in polynomial time a finite structure $\mathfrak{B}$ over a finite binary relational signature such that CSP($\mathfrak{A}$) and CSP($\mathfrak{B}$) are polynomial-time equivalent [24, 30]. By our results, such a reduction is unlikely to exist for universal sentences representing finitely bounded homogeneous structures, unless it avoids the amalgamation meta-problem. The reason is that, for binary relational signatures, the amalgamation meta-problem can be decided in coNEXPTIME (Proposition 3). Our results provide evidence for the need for a fundamentally new language-independent approach to the Bodirsky-Pinsker conjecture. First steps in this direction were taken in the recent works of Mottet and Pinsker [44] and Bodirsky and Bodor [9], but they do not fully address the issues stemming from the two meta-problems. We elaborate on this claim below. To keep our results as general as possible, we formulate them for some reasonable promise relaxations of the two meta-problems, i.e., where a subclass and a superclass of the positive instances are being separated from each other with the promise that the input never belongs to the complement of the subclass within the superclass.

The subtleties of the Bodirsky-Pinsker conjecture. In 2016, Bodirsky and Mottet presented an elegant tool for lifting tractability from finite-domain constraint satisfaction to the infinite [17], hereby establishing the first general link between the Feder-Vardi and the Bodirsky-Pinsker conjecture. Since then, their method has been used numerous times to prove new or reprove old complexity classification results for infinite-domain CSPs. One prominent such example is the universal-algebraic proof of the complexity dichotomy for MMSNP [16]. Conveniently enough, every MMSNP sentence defines a finite union of CSPs of structures within the scope of the Bodirky-Pinsker conjecture, so the two meta-problems were not relevant in this context. There is a prospect that the methods from [17] will also

prove useful in proving a dichotomy for the even more general logic class *Guarded Monotone* SNP (GMSNP) introduced in [7]. Also GMSNP enjoys the above mentioned property of MMSNP [15], and hence avoids the two meta-problems.

However, outside of GMSNP there exists a regime where the methods from [17] definitely fall short, and where the two meta-problems become relevant. Consider for instance the dichotomy for *temporal CSPs*, i.e., for CSPs of structures with domain $\mathbb{Q}$ and whose relations are definable by a Boolean combination of formulas of the form $(x = y)$ or $(x < y)$, obtained by Bodirsky and Kára in 2010 [14]. At the present time, these problems are already very well understood; tractable temporal CSPs can always be solved by an algorithm that repeatedly searches for a set of potential minimal elements among the input variables, where each instance of the search is performed using an oracle for a tractable finite-domain CSP. The latter is generally determined by the shape of the Boolean combinations. E.g., in the case of $\mathrm{CSP}(\mathbb{Q}; \{(x, y, z) \in \mathbb{Q}^3 \mid (x = y < z) \vee (y = z < x) \vee (z = x < y)\})$, solving the finite-domain CSP in question amounts to solving linear equations modulo 2 [14, 19]. It is known that the tractability results from [14] cannot be obtained using the reduction from [17].

In 2022, Mottet and Pinsker introduced the machinery of *smooth approximations* [44], which vastly generalizes the methods in [17]. The last section of their paper is devoted to temporal CSPs, and the authors manage to reprove a significant part of the dichotomy on just a few pages. They achieve this by applying some of their general results to first-order expansions of $(\mathbb{Q}; <)$ and obtaining either NP-hardness for the CSP, or one of the two types of symmetry that played a fundamental role in the original proof from [14]. This symmetry can then be used to prove correctness of the reduction to a finite-domain CSP described above, but only under an explicit usage of the homogeneity of $(\mathbb{Q}; <)$ (see Proposition 3.1 in [19] and the last section of [44]). In contrast to the methods in [17] which only use homogeneity as a blackbox, this approach can be described as *language-dependent*.

A similar situation occurs in the case of phylogeny CSPs [13], which capture decision problems concerning the existence of a binary tree satisfying certain constraints imposed on its leaves. Tractable phylogeny CSPs are strikingly similar to tractable temporal CSPs; they can always be solved by an algorithm that repeatedly searches for a subdivision of the input variables into two parts, representing the two different branches below the root of a binary tree, where each instance of this search is performed using an oracle for a tractable finite-domain CSP. However, for tractable phylogeny CSPs, already the homogeneity of the infinite-domain CSP template is both sufficient and necessary for proving the correctness of the reduction to the finite-domain CSP (Theorem 6.13 and Lemma 6.12 in [13]). We can therefore speak of a case of extreme language-dependency. Temporal and phylogeny CSPs are special cases of CSPs of structures obtainable from the universal homogeneous binary tree [10] by specifying relations using first-order formulas. Achieving a complexity dichotomy in this context will require a non-trivial combination of the methods from [14] and [13].

An optimal way of approaching the conjecture would be gaining a very good understanding of the class of reducts of finitely bounded homogeneous structures, e.g., through some sort of a classification. However, it is unclear how realistic this prospect is as model-theoretic properties often tend to be undecidable [25]. We remark that homogeneity is a vital part of the Bodirsky-Pinsker conjecture; this assumption can be weakened or strengthened but not dropped entirely, as otherwise we get a class that provably does not have a dichotomy [30, 8].

2 Preliminaries

Relational structures. The set $\{1, \ldots, n\}$ is denoted by $[n]$, and we use the bar notation $\bar{t}$ for tuples. A (*relational*) *signature* τ is a set of *relation symbols*, each $R \in \tau$ with an associated natural number called *arity*. We say that τ is *binary* if it consists of symbols of

arity ≤ 2. A (*relational*) τ*-structure* $\mathfrak{A}$ consists of a set A (the *domain*) together with the relations $R^{\mathfrak{A}} \subseteq A^k$ for each $R \in \tau$ with arity k. An *expansion* of $\mathfrak{A}$ is a σ-structure $\mathfrak{B}$ with $A = B$ such that $\tau \subseteq \sigma$, $R^{\mathfrak{B}} = R^{\mathfrak{A}}$ for each relation symbol $R \in \tau$. Conversely, we call $\mathfrak{A}$ a *reduct* of $\mathfrak{B}$. The *union* of two τ-structures $\mathfrak{A}$ and $\mathfrak{B}$ is the τ-structure $\mathfrak{A} \cup \mathfrak{B}$ with domain $A \cup B$ and relations of the form $R^{\mathfrak{A}\cup\mathfrak{B}} := R^{\mathfrak{A}} \cup R^{\mathfrak{B}}$ for every $R \in \tau$.

A *homomorphism* $h\colon \mathfrak{A} \to \mathfrak{B}$ for τ-structures $\mathfrak{A}, \mathfrak{B}$ is a mapping $h\colon A \to B$ that *preserves* each relation of $\mathfrak{A}$, i.e., if $\bar{t} \in R^{\mathfrak{A}}$ for some k-ary relation symbol $R \in \tau$, then $h(\bar{t}) \in R^{\mathfrak{B}}$. We write $\mathfrak{A} \to \mathfrak{B}$ if $\mathfrak{A}$ maps homomorphically to $\mathfrak{B}$. The *Constraint Satisfaction Problem* (CSP) of $\mathfrak{A}$, denoted by $\mathrm{CSP}(\mathfrak{A})$, is defined as the class of all finite structures which homomorphically map to $\mathfrak{A}$. An *embedding* is a homomorphism $h\colon \mathfrak{A} \to \mathfrak{B}$ that additionally satisfies the following condition: for every k-ary relation symbol $R \in \tau$ and $\bar{t} \in A^k$ we have $h(\bar{t}) \in R^{\mathfrak{B}}$ only if $\bar{t} \in R^{\mathfrak{A}}$. We write $\mathfrak{A} \hookrightarrow \mathfrak{B}$ if $\mathfrak{A}$ embeds to $\mathfrak{B}$. The *age* of $\mathfrak{A}$, denoted by $\mathrm{age}(\mathfrak{A})$, is the class of all finite structures which embed to $\mathfrak{A}$. A *substructure* of $\mathfrak{A}$ is a structure $\mathfrak{B}$ over $B \subseteq A$ such that the inclusion map $i\colon B \to A$ is an embedding. An *isomorphism* is a surjective embedding. Two structures $\mathfrak{A}$ and $\mathfrak{B}$ are *isomorphic* if there exists an isomorphism from $\mathfrak{A}$ to $\mathfrak{B}$. An *automorphism* is an isomorphism from $\mathfrak{A}$ to $\mathfrak{A}$. The *orbit* of a tuple $\bar{t} \in A^k$ in $\mathfrak{A}$ is the set $\{g(\bar{t}) \mid g \text{ is an automorphism of } \mathfrak{A}\}$. A countable structure $\mathfrak{A}$ is ω*-categorical* if, for every $k \geq 1$, there are only finitely many orbits of k-tuples in $\mathfrak{A}$.

First-order logic. We assume that the reader is familiar with classical *first-order* logic as well as with basic preservation properties of first-order formulas, e.g., that every first-order formula ϕ is preserved by isomorphisms; by embeddings if ϕ is existential, and by homomorphisms if ϕ is existential positive. For a first-order sentence Φ, we denote the class of all its finite models by $\mathrm{fm}(\Phi)$. We say that a first-order formula is k*-ary* if it has k free variables. For a first-order formula ϕ, we use the notation $\phi(\bar{x})$ to indicate that the free variables of ϕ are among $\bar{x}$. This does not mean that the truth value of ϕ depends on each entry in $\bar{x}$. We assume that equality $=$ as well as the nullary predicate symbol $\bot$ for falsity are always available when building first-order formulas. Thus, *atomic* τ*-formulas*, or τ*-atoms* for short, over a relational signature τ are of the form $\bot$, $(x = y)$, and $R(\bar{x})$ for some $R \in \tau$. We say that a formula is *equality-free* if it does not contain any occurrence of the default equality predicate. If ϕ is a disjunction of possibly negated τ-atoms, then we define the *Gaifman graph* of ϕ as the undirected graph whose vertex set consists of all free variables of ϕ and where two distinct variables x, y form an edge if and only if they appear jointly in a negative atom of ϕ. Let Φ be a universal τ-sentence in prenex normal form whose quantifier-free part ϕ is in CNF. We call Φ *Horn* if every clause of ϕ is Horn, i.e., contains at most one positive disjunct. We call Φ *complete* if the Gaifman graph of each clause of ϕ is complete. It is a folklore fact that, if Φ is complete, then $\mathrm{fm}(\Phi)$ is preserved by unions.

SNP and its fragments. An *SNP* τ*-sentence* is a second-order sentence Φ of the form $\exists X_1, \ldots, X_n \forall \bar{x}.\, \phi$ where ϕ is a quantifier-free formula in CNF over $\tau \cup \{X_1, \ldots, X_n\}$. We call Φ *monadic* if X_i is unary for every $i \in [n]$; *monotone* if ϕ does not contain any positive τ-atoms (in particular no positive equality atoms); and *guarded* if, for every positive atom β there exists a negative atom α containing all variables of β. Note that all notions from the previous paragraph easily transfer to SNP sentences viewed as universal sentences in an extended signature. The monadic monotone and the guarded monotone fragments of SNP are denoted by MMSNP and GMSNP, respectively. The monotone Horn fragment of SNP is commonly known as the logic programming language *Datalog*. When we say that a Datalog program Φ *solves* the CSP of a structure $\mathfrak{B}$, we simply mean that $\mathrm{fm}(\Phi) = \mathrm{CSP}(\mathfrak{B})$.

Homogeneity, homogenizability, and finite boundedness. A countable structure $\mathfrak{S}$ is *homogeneous* if every isomorphism between two finite substructures of $\mathfrak{S}$ extends to an automorphism of $\mathfrak{S}$. Clearly, every homogeneous structure in a finite relational signature is ω-categorical, and so are the reducts of such structures. Homogeneous structures arise as limit objects of well-behaved classes of finite structures in the sense of Theorem 1.

Let $\mathcal{K}$ be a class of finite structures in a finite relational signature τ closed under isomorphisms and substructures. We say that $\mathcal{K}$ has the *amalgamation property* (AP) if, for all $\mathfrak{B}_1, \mathfrak{B}_2 \in \mathcal{K}$ whose substructures on $B_1 \cap B_2$ are identical, there exists $\mathfrak{C} \in \mathcal{K}$ together with embeddings $f_1 \colon \mathfrak{B}_1 \hookrightarrow \mathfrak{C}$ and $f_2 \colon \mathfrak{B}_2 \hookrightarrow \mathfrak{C}$ such that $f_1|_{B_1 \cap B_2} = f_2|_{B_1 \cap B_2}$. We refer to $\mathfrak{C}$ as an *amalgam* of $\mathfrak{B}_1$ and $\mathfrak{B}_2$ in $\mathcal{K}$. Note that, for a class closed under isomorphisms and substructures, the AP is implied by the property of being closed under unions $\mathfrak{B}_1 \cup \mathfrak{B}_2$, also called *free amalgams*.

▶ **Theorem 1** (Fraïssé). *For a class $\mathcal{K}$ of finite structures in a finite relational signature τ, the following are equivalent:*

- *$\mathcal{K}$ is the age of an up to isomorphism unique countable homogeneous τ-structure;*
- *$\mathcal{K}$ is closed under isomorphisms, substructures, and has the AP.*

As already mentioned in the introduction, the structure $(\mathbb{Q}; <)$ is homogeneous because every local isomorphism can be extended to an automorphism using a piecewise affine transformation. Its age is the class of all finite strict linear orders.

A countable structure $\mathfrak{S}$ is *homogenizable* if it is a reduct of a homogeneous structure $\mathfrak{H}$ over a finite relational signature such that $\mathfrak{S}$ and $\mathfrak{H}$ have the same sets of automorphisms [28]. Whenever this happens, by the theorem of Ryll-Nardzewski, all relations of $\mathfrak{H}$ are first-order definable in $\mathfrak{S}$ [33]. One might say that $\mathfrak{S}$ already has all the relations necessary for homogeneity but they perhaps do not all have names. A prototypical example of this phenomenon is the universal "homogeneous" binary tree, which is homogenizable but not homogeneous, see, e.g., Proposition 3.2 in [10]. We call a class $\mathcal{K}$ of finite structures in a finite relational signature τ *homogenizable* if it forms the age of a homogenizable structure.

For a class $\mathcal{N}$ of finite structures in a finite relational signature τ, the class $\mathrm{Forb}_e(\mathcal{N})$ consists of all finite τ-structures which do not embed any member of $\mathcal{N}$. Following the terminology in [40], we say that a class $\mathcal{K}$ of finite structures in a finite relational signature is *finitely bounded* if there exists a finite $\mathcal{N}$ such that $\mathcal{K} = \mathrm{Forb}_e(\mathcal{N})$. We refer to $\mathcal{N}$ as a set of *bounds* for $\mathcal{K}$, and define the *size* of $\mathcal{N}$ as the sum of the cardinalities of the domain and the relations of all structures in $\mathcal{N}$. A structure $\mathfrak{S}$ is *finitely bounded* if its age is finitely bounded. We say that a class $\mathcal{K}$ is *finitely bounded homogenizable* if it forms the age of a reduct $\mathfrak{R}$ of a finitely bounded homogeneous structure $\mathfrak{H}$ such that $\mathfrak{R}$ and $\mathfrak{H}$ have the same sets of automorphisms. Sufficient conditions for finitely bounded homogenizability were provided by Hubička and Nešetřil [34], generalizing previous work of Cherlin, Shelah, and Shi [27].

3 The Amalgamation Meta-Problem

By Theorem 1, every homogeneous structure is uniquely described by its age (up to isomorphism). Consequently, every finitely bounded homogeneous structure is uniquely described by a finite set of bounds. It is known that the question whether $\mathrm{Forb}_e(\mathcal{N})$ has the AP for a given finite set of bounds $\mathcal{N}$ can be tested algorithmically in the case where the signature is binary [38]. This decidability result is based on the following observation. A *one-point amalgamation diagram* is an input $\mathfrak{B}_1, \mathfrak{B}_2$ to the AP where $|B_1 \setminus B_2| = |B_2 \setminus B_1| = 1$.

▶ **Proposition 2** ([38]). *A class of finite relational τ-structures that is closed under isomorphisms and substructures has the AP if and only if it has the AP restricted to one-point amalgamation diagrams.*

As a consequence of Proposition 2, if τ is binary and $\mathrm{Forb}_e(\mathcal{N})$ does not have the AP, then the size of a smallest counterexample to the AP is polynomial in the size of $\mathcal{N}$ [15]. Such a counterexample can be non-deterministically guessed and verified using a coNP-oracle, which places the problem at the second level of the polynomial hierarchy (Theorem 15 in [5]).

There is a second, arguably more practical, equivalent definition of finite boundedness. Namely, a class $\mathcal{K}$ of finite structures in a finite relational signature is finitely bounded if and only if there exists a universal sentence Φ such that $\mathcal{K} = \mathrm{fm}(\Phi)$. Using this definition, it is easy to see that $(\mathbb{Q}; <)$ is finitely bounded because its age, the class of all finite strict linear orders, admits a finite universal axiomatization (irreflexivity, transitivity, and totality). From a complexity-theoretical perspective, the two definitions are equivalent only up to a single-exponential blow-up in one direction. Given a finite set of bounds $\mathcal{N}$, we can obtain a universal sentence Φ of size polynomial in the size of $\mathcal{N}$ satisfying $\mathrm{fm}(\Phi) = \mathrm{Forb}_e(\mathcal{N})$ by describing each structure in $\mathcal{N}$ up to isomorphism using a quantifier-free formula. However, given a universal sentence Φ, it can be the case that a smallest $\mathcal{N}$ satisfying $\mathrm{fm}(\Phi) = \mathrm{Forb}_e(\mathcal{N})$ is of size single-exponential in the size of Φ. The reason is that obtaining $\mathcal{N}$ from Φ is comparable to rewriting Φ in DNF. Consequently, the algorithm from [15] only gives us a relatively weak upper bound for the case where the inputs are specified by universal sentences.

▶ **Proposition 3.** *Let Φ be a universal sentence over a finite binary relational signature τ. If $\mathrm{fm}(\Phi)$ does not have the AP, then the size of a smallest counterexample to the AP is at most single-exponential in the size of Φ. Consequently, the question whether $\mathrm{fm}(\Phi)$ has the AP is decidable in* coNEXPTIME.

The upper bound provided by Proposition 3 is not unreasonable since a smallest counterexample to the AP might be of size exponential in the size of the input sentence even if the signature is binary. This is demonstrated in Example 4.

▶ **Example 4.** Let τ be the signature consisting of the unary symbols $\{L, R\} \cup \{X_i \mid i \in [n]\}$ and the binary symbols $\{E\} \cup \{Y_i \mid i \in [n]\}$ for some $n \in \mathbb{N}$. Consider the universal sentence

$$\Phi \coloneqq \forall x, y_1, y_2 \Big(L(y_1) \wedge R(y_2) \wedge E(x, y_1) \wedge E(x, y_2) \wedge \big(\textstyle\bigwedge_{i \in [n]} Y_i(y_1, y_2) \Leftrightarrow X_i(x) \big) \Rightarrow \bot \Big).$$

Our first claim is that $\mathrm{fm}(\Phi)$ does not have the AP. We define the one-point amalgamation diagram $\mathfrak{B}_1, \mathfrak{B}_2 \in \mathrm{fm}(\Phi)$ as follows. The domains are $B_i \coloneqq \{b_i\} \cup \{b_S \mid S \subseteq [n]\}$, $i \in \{1, 2\}$, and the relations are given by the following conjunction of atomic formulas:

$$L(b_1) \wedge R(b_2) \wedge \textstyle\bigwedge_{S \subseteq [n]} E(b_S, b_1) \wedge E(b_S, b_2) \wedge \bigwedge_{i \in S} X_i(b_S).$$

We have that $\mathfrak{B}_1$ and $\mathfrak{B}_2$ satisfy Φ because $R^{\mathfrak{B}_1} = \emptyset$ and $L^{\mathfrak{B}_2} = \emptyset$. Clearly, no amalgam for $\mathfrak{B}_1$ and $\mathfrak{B}_2$ can be obtained by identifying b_1 and b_2 because $L^{\mathfrak{B}_1} = \{b_1\}$ and $R^{\mathfrak{B}_2} = \{b_2\}$. The free amalgam $\mathfrak{B}_1 \cup \mathfrak{B}_2$ does not satisfy Φ because of the assignment $y_1 \coloneqq b_1$, $y_2 \coloneqq b_2$, and $x \coloneqq b_\emptyset$. But since we can assign $x \coloneqq b_S$ for any $S \subseteq [n]$, also no amalgam satisfying Φ can be obtained by adding the pair (b_1, b_2) to any subset of the relations $Y_1^{\mathfrak{B}_1 \cup \mathfrak{B}_2}, \ldots, Y_n^{\mathfrak{B}_1 \cup \mathfrak{B}_2}$ of the free amalgam. We conclude that $\mathrm{fm}(\Phi)$ does not have the AP.

Our second claim is that every one-point amalgamation diagram $\mathfrak{B}_1, \mathfrak{B}_2 \in \mathrm{fm}(\Phi)$ satisfying $|B_1 \cap B_2| < 2^n$ has an amalgam in $\mathrm{fm}(\Phi)$. Let b_1 and b_2 be the unique elements contained in $B_1 \setminus B_2$ and $B_2 \setminus B_1$, respectively. If $\mathfrak{B}_1 \cup \mathfrak{B}_2 \models \Phi$, then we are done because $\mathfrak{B}_1 \cup \mathfrak{B}_2$ is

an amalgam for $\mathfrak{B}_1$ and $\mathfrak{B}_2$. So suppose that $\mathfrak{B}_1 \cup \mathfrak{B}_2 \not\models \Phi$. Consider any evaluation of the quantifier-free part of Φ witnessing the fact that $\mathfrak{B}_1 \cup \mathfrak{B}_2 \not\models \Phi$. Since b_1 and b_2 do not appear together in any relation of $\mathfrak{B}_1 \cup \mathfrak{B}_2$ and $\mathfrak{B}_1, \mathfrak{B}_2 \models \Phi$, by the shape of Φ, it must be the case that x is assigned some element $b \in B_1 \cap B_2$, y_1 is assigned b_1, and y_2 is assigned b_2 (or vice versa). Since $|B_1 \cap B_2| < 2^n$, there must exist $S \subseteq [n]$ such that, for every $b \in B_1 \cap B_2$, it is not the case that $b \in X_i^{\mathfrak{B}_1 \cup \mathfrak{B}_2}$ if and only if $i \in S$. Consequently, we can obtain an amalgam $\mathfrak{C} \in \mathrm{fm}(\Phi)$ for $\mathfrak{B}_1$ and $\mathfrak{B}_2$ by adding the pairs (b_1, b_2) and (b_2, b_1) to $Y_i^{\mathfrak{B}_1 \cup \mathfrak{B}_2}$ for every $i \in S$. We conclude that a smallest counterexample to the AP for $\mathrm{fm}(\Phi)$ is of size $> 2^n$.

Very little progress has been done on signatures containing symbols of arities larger than 2. In particular, it is not even known whether the AP is decidable for finitely bounded classes in general. The scenario where this is not the case is not unrealistic since the closely related *joint embedding property* (JEP) is undecidable already for finitely bounded classes of graphs [22]. The JEP determines whether a finitely bounded class forms the age of any structure, without the requirement of homogeneity [33]. Note that the undecidability of the JEP does not necessarily have any consequences for the Bodirsky-Pinsker conjecture, similarly as it did not have any for the Feder-Vardi conjecture.

If the AP turns out to be undecidable as well (for finitely bounded classes), then the Bodirsky-Pinsker conjecture addresses a class of structures with an undecidable membership problem, at least under the currently best known input to the meta-problem. It seems that, to some extent, the decidability issue can be ignored by only using homogeneity as a blackbox. This was demonstrated in the recent work [45] on the complexity of CSPs of homogeneous uniform hypergraphs, whose classification remains an open problem [3]. We remark that the complexity of the amalgamation meta-problem is already open in the following case, where we can only prove PSPACE-hardness.

▶ **Theorem 5.** *Given a universal Horn sentence Φ over a finite relational signature that is binary except for one ternary symbol, the question whether* $\mathrm{fm}(\Phi)$ *has the AP is* PSPACE*-hard.*

The next theorem states that testing the AP becomes properly harder than in the binary case if we do not impose any restrictions on the input (unless coNEXPTIME = EXPSPACE). The fact that this is also true for the strong version of the AP might be of independent interest to model-theorists. The *strong* version of the AP is when $\mathfrak{C} \in \mathcal{K}$ and $f_i \colon \mathfrak{B}_i \hookrightarrow \mathfrak{C}$ for $i \in \{1, 2\}$ can always be chosen so that $f_1(B_1) \cap f_2(B_2) = f_1(B_1 \cap B_2)$. Note that the theorem is formulated as a statement of the form "the question whether X or not even Y is hard." This is a compact way for writing that both X and Y (and every property in between) are hard, the formulation tacitly assumes that the inputs never satisfy "Y and not X."

▶ **Theorem 6.** *Given a universal sentence Φ over a finite signature, the question whether* $\mathrm{fm}(\Phi)$ *has the strong AP or not even the AP is* EXPSPACE*-hard.*

Theorem 7 is a variant of Theorem 6 in the setting where the input is specified by a set of bounds instead of a universal sentence. This setting can be compared to the situation where, in Theorem 6, the quantifier-free part of Φ is required to be in DNF. As a consequence, we cannot profit from succinctness of general universal sentences, which leads to a weaker PSPACE lower bound on the complexity. On the other hand, it turns out that PSPACE-hardness is witnessed even by instances whose domain size remains constant.

▶ **Theorem 7.** *Given a finite set $\mathcal{N}$ of finite structures over a finite signature, the question whether* $\mathrm{Forb}_e(\mathcal{N})$ *has the strong AP or not even the AP is* PSPACE*-hard. The statement is true even when the domain size for the structures in $\mathcal{N}$ is bounded by a constant.*

3.1 Proofs of Theorem 6 and Theorem 7

Our proofs of Theorem 6 and Theorem 7 are based on the fact that, if Φ is a universal Horn sentence such that $\mathrm{fm}(\Phi)$ does not have the AP, then every counterexample to the AP has the form of a particular Horn clause which can be derived from Φ in a syntactical manner. By padding each Horn clause in Φ with auxiliary negative atoms and hereby increasing its "degree of completeness," we gain some control over the form of the counterexamples to the AP. When this is performed in a careful and systematic way, the counterexamples to the AP can be brought into a 1:1 correspondence with the rejecting runs of certain Datalog programs. In our case, such programs verify the validity of tilings w.r.t. given input parameters to a bounded tiling problem. This is the main technical contribution of the present article.

▶ **Definition 8.** *Let Φ be an equality-free universal Horn sentence over a relational signature τ. Additionally, let $\phi(\bar{x})$ and $\psi(\bar{x},\bar{y})$ be equality-free conjunctions of atomic τ-formulas. We write $\psi(\bar{x},\bar{y}) \leq_\Phi \phi(\bar{x})$ if, for every atomic τ-formula $\chi(\bar{x})$ other than equality,*

$$\Phi \models \forall \bar{x},\bar{y}\big(\psi(\bar{x},\bar{y}) \Rightarrow \chi(\bar{x})\big) \quad \textit{implies} \quad \Phi \models \forall \bar{x}\big(\phi(\bar{x}) \Rightarrow \chi(\bar{x})\big).$$

In the next lemma, we reformulate the (strong) AP using Definition 8.

▶ **Lemma 9.** *Let Φ be an equality-free universal Horn sentence over a relational signature τ. Then the following are equivalent:*

1. $\mathrm{fm}(\Phi)$ *has the strong AP.*
2. $\mathrm{fm}(\Phi)$ *has the AP.*
3. *If $\phi(\bar{x})$, $\phi_1(\bar{x},y_1)$, and $\phi_2(\bar{x},y_2)$ are equality-free conjunctions of atomic formulas, where y_1 and y_2 are distinct variables not contained in $\bar{x}$, such that, for both $i \in \{1,2\}$, every atom in $\phi_i(\bar{x},y_i)$ contains the variable y_i and $\phi(\bar{x}) \wedge \phi_i(\bar{x},y_i) \leq_\Phi \phi(\bar{x})$, then*

$$\phi(\bar{x}) \wedge \phi_1(\bar{x},y_1) \wedge \phi_2(\bar{x},y_2) \leq_\Phi \phi(\bar{x}) \wedge \phi_1(\bar{x},y_1).$$

Let Φ be an equality-free universal Horn sentence and ψ a Horn clause over a relational signature τ. An *SLD-derivation of ψ from Φ* is a finite sequence of Horn clauses $\psi_0,\dots,\psi_s = \psi$ such that ψ_0 is a conjunct in Φ and, for every $i \in [s]$, there exists a Horn clause ϕ_i which is, up to renaming of variables, a conjunct in Φ, and such that ψ_i is obtained from ψ_{i-1} by replacing a negative atom of ψ_{i-1} that appears positively in ϕ_i with all negative atoms of ϕ_i. We say that ψ_i is a *resolvent* of ψ_{i-1} and ϕ_i. We call ψ a *weakening* of a clause ψ' if ψ' can be obtained from ψ by removing any amount of atoms. In particular, ψ is a weakening of itself. There exists an *SLD-deduction* of ψ from Φ, written as $\Phi \vdash \psi$, if ψ is a tautology or a weakening of a Horn clause ψ' that has an SLD-derivation from Φ. The following theorem presents a fundamental property of equality-free universal Horn sentences.

▶ **Theorem 10** (Theorem 7.10 in [46]). *Let Φ be an equality-free universal Horn sentence and ψ an equality-free Horn clause, both in a fixed signature τ. Then $\Phi \models \psi$ if and only if $\Phi \vdash \psi$.*

Our hardness proofs are by polynomial-time reductions from the complements of two well-known bounded versions of the tiling problem. Consider the signature σ consisting of the two binary symbols P_h, P_v, as well as the four unary symbols P_ℓ, P_r, P_t, P_b. For natural numbers $m,n \geq 1$, the σ-structure $\mathfrak{R}_{m,n}$ has the domain $[m] \times [n]$ and the relations

$$P_h^{\mathfrak{R}_{m,n}} := \{((i,j),(i+1,j)) \mid i \in [n-1], j \in [m]\},$$
$$P_v^{\mathfrak{R}_{m,n}} := \{((i,j),(i,j+1)) \mid i \in [n], j \in [m-1]\},$$
$$P_\ell^{\mathfrak{R}_{m,n}} := [m] \times \{1\},\ P_r^{\mathfrak{R}_{m,n}} := [m] \times \{n\},$$
$$P_b^{\mathfrak{R}_{m,n}} := \{1\} \times [n],\ P_t^{\mathfrak{R}_{m,n}} := \{m\} \times [n].$$

The *rectangle tiling problem* asks whether, given a natural number n and a finite σ-structure $\mathfrak{T}$, there exists a natural number m such that $\mathfrak{R}_{m,n} \to \mathfrak{T}$. Note that this is just a reformulation of the usual statement using the language of homomorphisms.[2] In contrast to the better-known NP-complete square tiling problem, one dimension of the tiling grid is not part of the input and is existentially quantified instead. As a result, the problem becomes PSPACE-complete [50].[3] One can further increase the complexity by allowing a succinct encoding of the space bound. The input remains the same but now we ask for a a rectangle tiling with 2^n columns. Analogously to the natural complete problems based on Turing machines, this yields a decision problem that is complete for the complexity class EXPSPACE [48].

Inputs specified by universal sentences. Theorem 6 is proved by polynomial-time reduction from the complement of the exponential rectangle tiling problem. From every input, we construct a universal sentence Φ of polynomial size such that $\mathrm{fm}(\Phi)$ has the AP if and only if there exists no exponential rectangle tiling satisfying the given parameters. The sentence Φ is almost Horn but disjunctions of non-negated atoms are used in premises of implications to represent exponentially many Horn clauses in a universal sentence of polynomial size. In the text that follows, we allow ourselves to still call such sentences Horn. Our encoding is very compact; each row, i.e., an ordered sequence of 2^n-many tiles, is represented using a constant amount of variables. This is achieved by storing the information about each individual row in binary using $(n+1)$-ary atoms whose entries always contain at most three variables. We refer to the variables representing rows of the tiling as *path* nodes. In order to check the tiling from bottom to top, i.e., parse a chain of path nodes, we require each pair of subsequent path nodes to be verified by a set of 2^n-many *verifier* nodes. This process ensures the vertical consistency of the tiling as well as the presence of 2^n-many tiles in every row. The precise number of verifier nodes is achieved using combinations of n pairs of unary atoms.

To control the occurrence of amalgamation failures, we first introduce a binary symbol E and two unary symbols L, R. Atoms with these symbols serve no other purpose than to ensure that *almost* each conjunct in Φ is complete, i.e., defines a class of structures that is preserved by taking unions and hence has the AP. More concretely, the premise of almost every Horn clause in Φ has a subformula of the form

$$L(y_1) \wedge R(y_2) \wedge \bigwedge\nolimits_{i\in[k]} E(y_1, x_i) \wedge E(y_2, x_i) \wedge \bigwedge\nolimits_{j\in[k]} E(x_i, x_j)$$

making the Horn clause almost complete, with the exception of one potentially missing edge in the Gaifman graph between y_1 and y_2. Our intention is to make this missing edge the only place at which potential faulty one-point amalgamation diagrams can be built (see Figure 1). The sentence Φ is defined as $\Phi_1 \wedge \Phi_2$, where the two parts are described below.

The first part Φ_1 does not yet explain how our reduction works, but ensures that it does not fall apart, e.g., due to ill-behaved identifications of variables. For every $\alpha \in T$, the signature τ contains an $(n+1)$-ary symbol T_α. The first n arguments in a T_α-atom serve as binary counters, and the last argument carries a given path node p. Suppose that the variables $\mathtt{0}$ and $\mathtt{1}$ represent the bits 0 and 1, respectively. Then each atomic formula $T_\alpha(c_1, \dots, c_n, p)$ with $c_1, \dots, c_n \in \{\mathtt{0}, \mathtt{1}\}$ represents the situation in which a tile α is present in the p-th row and in the $\big(1 + \sum_{k\in[n]} (c_k = \mathtt{1}) \cdot 2^{n-k}\big)$-th column. First, we want to ensure the horizontal consistency of the tiling. To this end, for every pair $(\alpha, \beta) \in T^2 \setminus P_h^{\mathfrak{T}}$, we

[2] For comparison, see, e.g., Section 4 in [32].
[3] In [50], the rectangle tiling problem is called the corridor tiling problem.

include in Φ_1 a complete Horn sentence without positive atoms ensuring that two horizontally adjacent positions in the p-th row cannot be tiled with α and β. Here, we encode the successor relation w.r.t. binary addition using a combination of equalities: $(c_{n+1}, \dots, c_{2n})$ is the successor of $(c_1, \dots, c_n)$ if and only if there exists $j \in [n]$ such that $c_i = c_{i+n} \in \{0, 1\}$ for every $i \in [j-1]$, $c_j = 0$ and $c_{n+j} = 1$, and $c_i = 1$ and $c_{i+n} = 0$ for every $i \in [n] \setminus [j]$. This encoding only makes sense if 0 and 1 truly represent the bits 0 and 1, so we introduce a simple mechanism (in terms of a Horn sentence) for distinguishing between the variables 0 and 1. Next, we want to ensure that every position in the p-th row is occupied by at most one tile. To this end, for every $(\alpha, \beta) \in T^2$ with $\alpha \neq \beta$, we include

$$\forall p, 0, 1, c_1, \dots, c_n \big(T_\alpha(c_1, \dots, c_n, p) \wedge T_\beta(c_1, \dots, c_n, p) \wedge \bigwedge\nolimits_{i \in [n]} (c_i = 0 \vee c_i = 1) \Rightarrow \bot \big)$$

as a conjunct in Φ_1. Finally, we want to ensure that each verifier node v represents at most one number from $[2^n]$. For every $i \in [n]$, τ contains two unary symbols $\mathbf{0}_i$ and $\mathbf{1}_i$ which will be used to encode numbers in binary. We include $\forall v \big(\bigvee_{i \in [n]} \mathbf{0}_i(v) \wedge \mathbf{1}_i(v) \Rightarrow \bot \big)$ as the last conjunct in Φ_1. Now the idea is that the combinations of atomic formulas $\mathbf{0}_i(v)$ and $\mathbf{1}_i(v)$ at a verifier node v will be compared with the combinations of 0 and 1 in atomic formulas $T_\alpha(c_1, \dots, c_n, p)$. The Horn sentences in the second part of Φ will be formulated so that verifying the presence of all 2^n atoms of the form $T_\alpha(c_1, \dots, c_n, p_1)$ at a path node p_1 using verifier nodes $v_1, \dots, v_{2^n}$ is the only possible way to progress to a next path node p_2. Consequently, we do not need to add an explicit requirement for rows, represented by path nodes, to be completely tiled from left to right. For the same reason, we also do not need to add an explicit requirement for verifier nodes to represent at least one number from $[2^n]$.

We now proceed with the sentence Φ_2, which explains how the parsing of a tiling actually works. The parsing of a tiling starts from a path node p representing a row whose leftmost position contains a tile that can be present in the bottom left corner of a tiling grid. This must be confirmed by a verifier node, in which case a 6-ary Q_b-atom is derived, representing the fact that the leftmost column of the p-th row has been checked. To this end, we include in Φ_2 suitable Horn sentences for every $\alpha \in P_\ell^{\mathfrak{T}} \cap P_b^{\mathfrak{T}}$. These sentences form *the* non-complete part of Φ; we intentionally leave a missing edge between y_1 and y_2 in the Gaifman graph to enable the formation of potential AP-counterexamples. Moreover, the variables y_1 and y_2 appear in the 1st and the 2nd entry of the derived atom, respectively, and this invariant is maintained throughout the whole construction of Φ_2.

Using 2^n-many verifier nodes and propagation of Q_b-atoms, the whole bottom row is checked for the presence of tiles. Their horizontal consistency already follows from the conditions imposed on path nodes by Φ_2 and needs not to be checked during this step. To this end, we include in Φ_2 suitable Horn sentences for every $\alpha \in P_b^{\mathfrak{T}}$. After the p-th row has been checked by a 2^n-th verifier node, we mark p with a Q-atom indicating that the parsing can progress to a successor path node. To this end, we include in Φ_2 suitable Horn sentences for every $\alpha \in P_b^{\mathfrak{T}} \cap P_r^{\mathfrak{T}}$. The successor relation for path nodes is represented by the binary symbol S, and the certificate of vertical verification for pairs (p_1, p_2) of successive path nodes is represented by the 7-ary symbol Q_v. For every $(\alpha, \beta) \in P_v^{\mathfrak{T}} \cap (P_\ell^{\mathfrak{T}})^2$, we include in Φ_2 a Horn sentence verifying the vertical consistency of the leftmost positions in the rows p_1 and p_2 and deriving the first Q_v-atom containing p_1 and p_2, but only if there is a Q-atom containing p_1. Next, for every $(\alpha, \beta) \in P_v^{\mathfrak{T}}$, we include in Φ_2 a Horn sentence verifying the vertical consistency for the intermediate positions by deriving further Q_v-atoms containing p_1 and p_2. And finally, for every $(\alpha, \beta) \in P_v^{\mathfrak{T}} \cap (P_r^{\mathfrak{T}})^2$, we include in Φ_2 a Horn sentence verifying the vertical consistency of the leftmost positions in the rows p_1 and p_2 and deriving the first Q-atom containing p_2 only. The top row is verified using a 6-ary symbol Q_t similarly as the bottom row; however, the verification of the rightmost position in the top row results in the derivation of $\bot$.

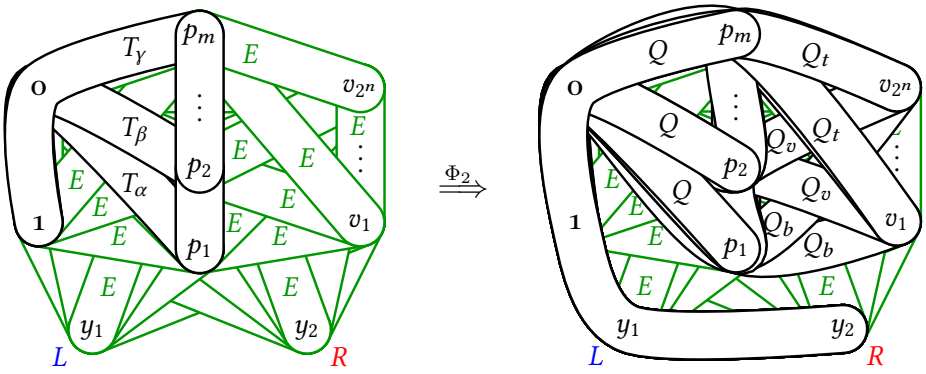


Figure 1 An illustration of an AP-counterexample representing a verification of a valid tiling of an exponential rectangle with m rows and 2^n columns.

Proof of Theorem 6. We first argue that Φ is equivalent to a particular Horn sentence, to which we can apply Lemma 9. By construction, the quantifier-free part of each conjunct in Φ has the form of an implication where the premise possibly also contains instances of disjunction, which are not allowed in Horn clauses, but no instances of negation. Therefore, it can be rewritten as a conjunction of Horn clauses by converting the premise into positive DNF and then considering each disjunct as a separate premise. As a result, the size of the sentence increases exponentially, but this does not matter for the purpose of the proof. Subsequently, all equality atoms can be eliminated by replacing each variable c_i with either o or $\mathrm{1}$. We denote the resulting Horn sentence by $\overline{\Phi}$ and the two parts stemming from Φ_1 and Φ_2 by $\overline{\Phi}_1$ and $\overline{\Phi}_2$, respectively.

"$\Rightarrow$" Suppose that there exists a tiling $f\colon \mathfrak{R}_{m,2^n} \to \mathfrak{T}$. Guided by f, we define a one-point amalgamation diagram $\mathfrak{B}_1, \mathfrak{B}_2 \in \mathrm{fm}(\overline{\Phi})$ which has no amalgam in $\mathrm{fm}(\overline{\Phi})$ (see Figure 1). The domains are $B_i \coloneqq \{y_i, p_1, \dots, p_m, v_1, \dots, v_{2^n}, \mathrm{o}, \mathrm{1}\}$ for $i \in \{1, 2\}$, and the relations are given by the following conjunctions of atomic formulas. We require $T_\alpha(c_1, \dots, c_n, p_j)$ for $c_1, \dots, c_n \in \{\mathrm{o}, \mathrm{1}\}$ if and only if $f(1 + \sum_{k\in[n]}(c_k = \mathrm{1}) \cdot 2^{n-k}, j) = i$. Next, we require all of the L, R, and E-atoms necessary for enabling the Horn clauses in $\mathfrak{B}_1 \cup \mathfrak{B}_2$. Finally, we require $\bigwedge_{i\in[m-1]} S(p_i, p_{i+1})$ to define a successor chain through path nodes, and $\mathrm{O}_i(v_j)$ or $\mathrm{1}_i(v_j)$ if and only if $j = 1 + \sum_{k\in[n]} \lambda_k \cdot 2^{n-k}$ for $\lambda_1, \dots, \lambda_n \in \{0, 1\}$ and $\lambda_i = 0$ or $\lambda_i = 1$, respectively. Clearly, the tiling atoms are placed correctly and the verifier nodes correctly represent values in $[2^n]$. Since $R^{\mathfrak{B}_1} = \emptyset$ and $L^{\mathfrak{B}_2} = \emptyset$, we have $\mathfrak{B}_1, \mathfrak{B}_2 \models \overline{\Phi}_2$. Since f is horizontally consistent, we have $\mathfrak{B}_1, \mathfrak{B}_2 \models \overline{\Phi}_1$, i.e., $\mathfrak{B}_1, \mathfrak{B}_2 \in \mathrm{fm}(\overline{\Phi})$. But since f is also vertically consistent and $\overline{\Phi}_2$ is a universal Horn sentence, we have $\mathfrak{C} \not\models \overline{\Phi}_2$ for every τ-structure $\mathfrak{C}$ with a homomorphism from $\mathfrak{B}_1 \cup \mathfrak{B}_2$. Hence, $\mathrm{fm}(\overline{\Phi})$ does not have the AP.

"$\Leftarrow$" Suppose that $\mathrm{fm}(\overline{\Phi})$ does not have the AP. Then there exists a counterexample to item (3) in Lemma 9, i.e., there exists a Horn clause ψ of the form $\phi(\bar{x}) \wedge \phi_1(\bar{x}, y_1) \wedge \phi_2(\bar{x}, y_2) \Rightarrow \chi$, where ϕ, ϕ_1, and ϕ_2 satisfy the prerequisites of item (3) in Lemma 9 and $\chi(\bar{x}, y_1)$ is an atomic τ-formula other than equality, such that

$$\overline{\Phi} \models \forall \bar{x}, y_1, y_2 \big(\phi \wedge \phi_1 \wedge \phi_2 \Rightarrow \chi\big), \tag{1}$$

$$\overline{\Phi} \not\models \forall \bar{x}, y_1 \big(\phi \wedge \phi_1 \Rightarrow \chi\big). \tag{2}$$

We choose ψ minimal with respect to the number of its atomic subformulas. By Theorem 10, ψ has an SLD-deduction from $\overline{\Phi}$. Note that, by (2), $\chi(\bar{x}, y_1)$ cannot be a subformula of $\phi(\bar{x}) \wedge \phi_1(\bar{x}, y_1)$. Also, $\chi(\bar{x}, y_1)$ cannot be a subformula of $\phi_2(\bar{x}, y_2)$ because every atom in

$\phi_2(\bar{x}, y_2)$ contains the variable y_2 which does not appear in $\chi(\bar{x}, y_1)$. Hence, $\chi(\bar{x}, y_1)$ is not a subformula of $\phi(\bar{x}) \wedge \phi_1(\bar{x}, y_1) \wedge \phi_2(\bar{x}, y_2)$, i.e., ψ is not a tautology. Consequently, ψ is a weakening of a Horn clause ψ' which has an SLD-derivation $\psi'_0, \dots, \psi'_s = \psi'$ from $\overline{\Phi}$. Recall that every atom from ψ' appears in ψ.

▷ Claim 11. $\overline{\Phi}_2 \vdash \psi'$.

Proof. We start by showing that ψ'_0 is a conjunct of $\overline{\Phi}_2$. Suppose, on the contrary, that ψ'_0 is a conjunct of $\overline{\Phi}_1$. Then ψ' does not contain any positive atom because, by construction, Horn sentences in $\overline{\Phi}_1$ do not contain any positive atoms. By construction, in $\overline{\Phi}$ there is no Horn clause containing a positive atom that occurs negatively in a Horn clause from $\overline{\Phi}_1$, i.e., it is impossible to take resolvents of ψ'_0 and Horn clauses from $\overline{\Phi}$. It follows that $s = 0$. Also, every Horn clause from $\overline{\Phi}_1$ is complete. Since there is no edge between y_1 and y_2 in the Gaifman graph of ψ, and, for $i \in \{1, 2\}$, each atom in ϕ_i contains the variable y_i, either ϕ_1 or ϕ_2 must be empty. Since ψ' does not contain any positive atom and $\phi(\bar{x}) \wedge \phi_i(\bar{x}, y_i) \leq_{\overline{\Phi}} \phi(\bar{x})$ for both $i \in \{1, 2\}$, we get a contradiction to (2). Thus, ψ'_0 must be a conjunct of $\overline{\Phi}_2$. Since no Horn clause in $\overline{\Phi}_1$ contains a positive atom, no Horn clause from $\overline{\Phi}_1$ can be used as a resolvent. We conclude that ψ' has an SLD-derivation from $\overline{\Phi}_2$. ◁

▷ Claim 12. ψ' contains no positive atoms, and no atoms with a symbol from $\{Q_b, Q_v, Q_t, Q\}$.

Proof. By the construction of $\overline{\Phi}_2$, for every $i \in [s]$, if ψ'_{i-1} contains variables z_1, z_2 such that

$$\begin{array}{l}\text{every atomic subformula with a symbol from } \{Q_b, Q_v, Q_t, Q\} \\ \text{contains } z_1 \text{ in its 1st and } z_2 \text{ in its 2nd argument, respectively,}\end{array} \tag{3}$$

then this is also the case for ψ'_i, for the same variables z_1, z_2 up to renaming. Since every possible choice of ψ'_0 from $\overline{\Phi}_2$ initially satisfies (3), it follows via induction that (3) holds for $\psi' = \psi'_s$ for some variables z_1, z_2. Next, we show that $\{z_1, z_2\} = \{y_1, y_2\}$ holds for the pair z_1, z_2 satisfying (3) for ψ'. Suppose, on the contrary, that both z_1 and z_2 are among $\bar{x}, y_1$ or $\bar{x}, y_2$. By construction, the only Horn clauses in $\overline{\Phi}_2$ that are not complete have the property that the incompleteness is only due to one missing edge in the Gaifman graph between two distinguished variables satisfying (3). Therefore, for every $i \in [s]$, ψ'_i is a resolvent of ψ'_{i-1} and a Horn clause from $\overline{\Phi}_2$ which is almost complete except possibly for one missing edge in the Gaifman graph between a pair of variables which must be substituted for the pair (z_1, z_2) satisfying (3) for ψ'_{i-1}. Since the variables y_1 and y_2 do not appear together in any atom in ψ' and $\{z_1, z_2\} \neq \{y_1, y_2\}$, they also do not appear together in any atom during the SLD-derivation. Then it follows from the fact that ϕ, ϕ_1, and ϕ_2 satisfy the prerequisites of item (3) in Lemma 9 that we already have $\overline{\Phi}_2 \vdash \forall \bar{x}, y_1 \big(\phi \wedge \phi_1 \Rightarrow \chi\big)$, a contradiction to (2). Since $\{z_1, z_2\} = \{y_1, y_2\}$ holds for the pair z_1, z_2 satisfying (3) for ψ', ψ' cannot contain any negative atoms with a symbol from $\{Q_b, Q_v, Q_t, Q\}$. Suppose that the conclusion of ψ'_0 is not $\bot$. Then, by construction, ψ' contains a positive atom with a symbol from $\{Q_b, Q_v, Q_t, Q\}$. But then, since z_1, z_2 with $\{z_1, z_2\} = \{y_1, y_2\}$ satisfy (3) for ψ', the said positive atom in ψ' contains both variables y_1 and y_2. This leads to a contradiction to (1) where we assume that the positive atom in ψ may may only contain variables from $\bar{x}, y_1$. Thus the conclusion of ψ'_0 is $\bot$, which means that χ equals $\bot$ due to the minimality assumption. ◁

It remains to show that the existence of such ψ' implies the existence of a tiling $f \colon \mathfrak{R}_{m,2^n} \to \mathfrak{T}$. By the first and the second claim, the conclusion of ψ'_0 is $\bot$. Since ψ' does not contain any atoms with a symbol from $\{Q_b, Q_v, Q_t, Q\}$, the last such atom must have been eliminated from ψ'_{s-1} by taking a resolvent with one of the incomplete Horn clauses in $\overline{\Phi}_2$. By the

construction of $\overline{\Phi}_2$, to obtain ψ' through an SLD-derivation $\psi'_0, \ldots, \psi'_s = \psi'$ from $\overline{\Phi}_2$, all Horn clauses introduced in the definition of $\overline{\Phi}_2$ must have been used in the intended order. Recall that we have replaced each variable c_i in Φ_2 with either 0 or 1 while rewriting Φ_2 as an Horn sentence. Every Horn clause from $\overline{\Phi}_2$ has the property that every positive atomic subformula has a symbol from $\{Q_b, Q_v, Q_t, Q\}$ and contains all variables that appear in a negative atomic subformula, with the following two exceptions. First, verifier nodes are not carried over in any atoms because their only contribution is the encoding of a unique number. Second, after a pair of successive rows has been checked by deriving a Q_v-atom containing a 2^n-th verifier node, the variable representing the lower row is not carried over in any atom because it is no longer needed. Since ϕ, ϕ_1, and ϕ_2 satisfy the prerequisites of item (3) in Lemma 9, no ill-behaved variable identifications might have occurred during the SLD-derivation above as otherwise, we would have $\overline{\Phi}_1 \models \forall \bar{x}, y_1\big(\phi \wedge \phi_1 \Rightarrow \chi\big)$, a contradiction to (2). Consequently, the SLD-derivation must have the full intended length $(2^n + 1) \cdot m$ for some $m \geq 1$, because every intermediate stage starts and ends with verifier nodes encoding the numbers 2^n and 1, respectively, and one can only progress in steps which decrement the encoded number by one. Clearly, the SLD-derivation witnesses the existence of $f\colon \mathfrak{R}_{m,2^n} \to \mathfrak{T}$. ◀

Inputs specified by sets of bounds. We continue with the proof of Theorem 7. This time, we reduce from the complement of the basic rectangle tiling problem. The proof strategy is similar. In particular, we include in τ the two auxiliary unary symbols L, R and the binary auxiliary symbol E. However, the encoding of the tiles is different. We include in τ a symbol I of arity $\lceil \log_2 |T| \cdot n \rceil + 1$. The first $\lceil \log_2 |T| \cdot n \rceil$ entries serve as binary counters to represent the pairs $(i, \alpha) \in [n] \times T$ in binary, and the last entry carries a path node representing a row of the tiling grid. Each pair $(i, \alpha) \in [n] \times T$ is to be interpreted as the fact that the i-th column in the row represented by a particular path node contains the tile α. The reason for this choice of encoding is that we aim to construct a universal sentence Φ which is equivalent to a set of forbidden substructures $\mathcal{N}$ of size polynomial in $|T| \cdot n$. To achieve this, we use a constant number of symbols whose arity is logarithmic in the size of the input.

Suppose that 0 and 1 are two variables representing the bits 0 and 1, respectively. For each pair $(i, \alpha) \in [n] \times T$, the ternary formula $\mathrm{TILE}_{i,\alpha}(0, 1, p)$ is the I-atom whose last entry contains the variable p and the first $\lceil \log_2 |T| \cdot n \rceil$ entries contain the variables 0 and 1 in the unique way that represents the number $i \cdot \alpha$ in binary when read from left to right. Note that the number of such formulas is polynomial in the size of the input to the tiling problem. In contrast to the proof of Theorem 6, it is not necessary to introduce any verifier nodes as the number of columns in the tiling grid is polynomial in the size of the input. The sentence Φ is defined similarly as in the proof of Theorem 6, so we only provide a general overview and highlight the main differences. We want each row to be horizontally consistent. For all $i \in [n-1]$ and $(\alpha, \beta) \in T^2 \setminus P_h^{\mathfrak{T}}$, we include the following sentence as a conjunct in Φ_1:

$$\forall p, 0, 1\big(\psi(0, 1) \wedge \mathrm{TILE}_{i,\alpha}(0, 1, p) \wedge \mathrm{TILE}_{i+1,\beta}(0, 1, p) \Rightarrow \bot\big),$$

where $\psi(0, 1)$ represents a simple mechanism for distinguishing between 0 and 1. We also want each position in a given row to be occupied by at most one tile. For all $i \in [n], \alpha, \beta \in T$, we include the following sentence as a conjunct in Φ_1:

$$\forall p, 0, 1\big(\mathrm{TILE}_{i,\alpha}(0, 1, p) \wedge \mathrm{TILE}_{i,\beta}(0, 1, p) \Rightarrow \bot\big).$$

As in the proof of Theorem 7, we do *not* need to include an explicit condition stating that each row must be completely tiled from left to right. For the purpose of verifying the validity

of a tiling, we include in τ a 5-ary symbol Q, a $(\lceil \log_2 |T| \cdot n \rceil + 4)$-ary symbol Q_v, and two $(\lceil \log_2 |T| \cdot n \rceil + 3)$-ary symbols Q_b, Q_t. For each pair $(i, \alpha) \in [n] \times T$, the formulas

$$\mathrm{BOT}_{i,\alpha}(0, 1, p, y_1, y_2), \quad \mathrm{TOP}_{i,\alpha}(0, 1, p, y_1, y_2), \quad \text{and} \quad \mathrm{VERT}_{i,\alpha}(0, 1, p_1, p_2, y_1, y_2)$$

are defined analogously to $\mathrm{TILE}_{i,\alpha}(0, 1, p)$ but using the Q_b, Q_t and Q_v-atoms instead. We use them to verify the bottom row, top row, and the vertical consistency of a given tiling. In contrast to $\mathrm{TILE}_{i,\alpha}$, the parameter α is not important in $\mathrm{BOT}_{i,\alpha}$, $\mathrm{TOP}_{i,\alpha}$, and $\mathrm{VERT}_{i,\alpha}$.

We now explain how to convert Φ into a set of forbidden substructures. Let $\mathcal{N}$ be the class of all τ-structures with the domain $[i]$ for some $i \in [6]$ that do not satisfy Φ. Since Φ only uses six variables, we have $\mathrm{fm}(\Phi) = \mathrm{Forb}_e(\mathcal{N})$. It remains to show that there exists a polynomial that bounds the size of $\mathcal{N}$. Since there is a constant number of domains of structures in $\mathcal{N}$ and their sizes are also constant, it is enough to show that there exists a polynomial that bounds the number of structures in $\mathcal{N}$. The only non-constant parameters in the construction are the four symbols I, Q_b, Q_t, and Q_v whose arity grows logarithmically with $|T| \cdot n$. Thus, there exists a constant c such that the number of structures in $\mathcal{N}$ is bounded by $c \cdot \left(2^{\lceil \log_2 |T| \cdot n \rceil}\right)^4 \leq c \cdot (2 \cdot |T| \cdot n)^4$. The rest is analogous to the proof of Theorem 6.

4 The Homogenizability Meta-Problem

Every reduct $\mathfrak{R}$ of a finitely bounded homogeneous structure $\mathfrak{H}$ is uniquely described by an SNP sentence, which can be obtained from a universal sentence for $\mathrm{age}(\mathfrak{H})$ by existentially quantifying all the surplus predicates upfront. This is (arguably) the most natural representation for such structures. The homogenizability meta-problem asks whether a given SNP τ-sentence Φ is logically equivalent to an SNP τ-sentence $\Psi = \exists Y_1, \ldots, Y_m \forall \bar{y}. \psi$ such that $\mathrm{fm}(\forall \bar{y}. \psi)$ has the AP in the signature $\tau \cup \{Y_1, \ldots, Y_m\}$. We are additionally interested in the refinement of the question where we require the homogeneous structure from Theorem 1 associated to $\mathrm{fm}(\forall \bar{y}. \psi)$ to have the same set of automorphisms as its reduct to the original signature τ. This amounts to asking whether $\mathrm{fm}(\Phi)$ is finitely bounded homogenizable.

MMSNP was presented in [30] as a large subclass of SNP which has a dichotomy between P and NP-completeness if and only if the class of all finite-domain CSPs has one. The latter has been confirmed, and the dichotomy for MMSNP has received a new universal-algebraic proof within the programme attacking the Bodirsky-Pinsker conjecture [16]. The new proof relies on the observation that every MMSNP sentence Φ is equivalent to a finite disjunction $\Phi_1 \vee \cdots \vee \Phi_n$ of MMSNP sentences such that, for every $i \in [n]$, there exists a reduct $\mathfrak{R}_i$ of a finitely bounded homogeneous structure $\mathfrak{H}_i$ such that $\mathrm{fm}(\Phi_i) = \mathrm{CSP}(\mathfrak{R}_i)$.[4] Moreover, the structure $\mathfrak{H}$ can be chosen so that its age has the *Ramsey property*, which plays an essential role in an argument in [16] showing that the authors correctly identified all of the tractable cases. The exact definition of this property is not essential to the present article and is therefore omitted. GMSNP was first introduced in [41] in its seemingly weaker form MMSNP_2, as a generalization of MMSNP where "monadic" second-order variables may also range over atomic formulas. It was later shown that relaxing the above requirement for monotone SNP to guardedness does not result in a more expressive logic [7]. There is a prospect that GMSNP will also have dichotomy between P and NP-completeness since it enjoys similar model-theoretic properties as MMSNP [15].

Theorem 13 is the most general version of our undecidability result. It applies not only to the original formulation of the homogenizability meta-problem, but also to its generalization to ω-categorical structures. The second item of the theorem might give the impression that

[4] This observation was first made in the the proof of Theorem 7 in [11].

one cannot effectively distinguish between CSPs of reducts of finitely bounded structures and CSPs of reducts of finitely bounded homogeneous structures. Recall that the former class does not have a dichotomy [30, 8]. However, as indicated by the formulation of the second item, all CSPs of reducts of finitely bounded homogeneous structures in the proof of Theorem 13 are in fact finite-domain CSPs, for which there is a dichotomy. Therefore, Theorem 13 merely shows that SNP sentences are an exceptionally bad choice of an input to the question, albeit one that is often used [4, 43].

▶ **Theorem 13.** *For a given a Datalog sentence Φ using at most binary relation symbols, it is undecidable whether:*

1. *Φ is logically equivalent to a monadic Datalog sentence,*
   or Φ is not even logically equivalent to any GMSNP sentence;
2. *Φ simultaneously satisfies the following three conditions:*
   - *$\mathrm{fm}(\Phi)$ is the CSP of a finite structure,*
   - *$\mathrm{fm}(\Phi)$ is a finitely bounded homogenizable class,*
   - *$\mathrm{fm}(\Phi)$ is the age of a reduct of a finitely bounded homogeneous Ramsey structure,*

   or $\mathrm{fm}(\Phi)$ is not even the CSP or the age of any ω-categorical structure.

The following corollary extracts the statement originally announced in the introduction.

▶ **Corollary 14.** *It is undecidable whether a given SNP sentence defines the age of a reduct of a finitely bounded homogeneous structure. The statement is true even if the SNP sentence comes from the Datalog fragment and uses at most binary relation symbols.*

4.1 A proof of Theorem 13

As usual, the *Kleene plus* and the *Kleene star* of a finite set of symbols Σ, denoted by Σ^+ and Σ^*, are the sets of all finite words over Σ of lengths ≥ 1 and ≥ 0, respectively.

A *context-free grammar* (CFG) is a 4-tuple $\mathcal{G} = (N, \Sigma, P, S)$ where N is a finite set of *non-terminal symbols*, Σ is a finite set of *terminal symbols*, P is a finite set of *production rules* of the form $A \to w$ where $A \in N$ and $w \in (N \cup \Sigma)^+$, $S \in N$ is the *start symbol*. For $u, v \in (N \cup \Sigma)^+$ we write $u \to_G v$ if there are $x, y \in (N \cup \Sigma)^+$ and $(A \to w) \in P$ such that $u = xAy$ and $v = xwy$. The *language of* $\mathcal{G}$ is $L(\mathcal{G}) \coloneqq \{w \in \Sigma^+ \mid S \to^*_{\mathcal{G}} w\}$, where $\to^*_{\mathcal{G}}$ denotes the transitive closure of $\to_{\mathcal{G}}$. Note that with this definition the *empty word* ϵ can never be an element of $L(\mathcal{G})$; some authors use a modified definition that also allows rules that derive ϵ, but for our purposes the difference is not essential. A context-free grammar is called *(left-)regular* if its production rules are always of the form $A \to a$ or $A \to Ba$ for non-terminal symbols A, B and a terminal symbol a. For a finite set Σ, we call a set $L \subseteq \Sigma^+$ *regular* if it is the language of a regular grammar with terminal symbols Σ.

▶ **Example 15.** Consider the CFG $\mathcal{G}$ with a single terminal symbol a, non-terminal symbols S, A, B, C, and production rules $S \to a$, $S \to aa$, $S \to aaa$, $S \to Aa$, $A \to Ba$, $B \to Ca$, $C \to Ca$, and $C \to a$. Clearly, $\mathcal{G}$ is not regular. However, $L(\mathcal{G}) = \{a\}^+$ is regular.

Let $\mathcal{G} = (N, \Sigma, P, S)$ be a CFG. The signature τ_Σ consists of the unary symbols I, T and the binary symbols R_a for every $a \in \Sigma$, and the signature τ_N consists of a binary symbol R_a for every element $a \in N$. For $a_1 \dots a_n \in \Sigma^+$, we set $\phi_{a_1 \dots a_n}(x_1, \dots, x_{n+1}) \coloneqq \bigwedge_{i \in [n]} R_{a_i}(x_i, x_{i+1})$. Let $\overline{\Phi}_{\mathcal{G}}$ be the universal Horn sentence over the signature $\tau_\Sigma \cup \tau_N$ whose quantifier-free part contains, for every $(A, w) \in P$, the Horn clause $\phi_w(x_1, \dots, x_{|w|+1}) \Rightarrow R_A(x_1, x_{|w|+1})$, and additionally the Horn clause $I(x_1) \wedge R_S(x_1, x_2) \wedge T(x_2) \Rightarrow \bot$. Then $\Phi_{\mathcal{G}}$ is the Datalog sentence obtained from $\overline{\Phi}_{\mathcal{G}}$ by existentially quantifying all symbols from τ_N

upfront. This encoding of CFGs into Datalog programs is standard (Exercise 12.26 in [1]), and the correspondence provided by the next lemma can be shown via a straightforward induction. For a proof, we refer the reader to [20].

▶ **Lemma 16.** *For a τ_Σ-structure $\mathfrak{A}$, we have $\mathfrak{A} \models \Phi_{\mathcal{G}}$ if and only if, for every $w \in L(\mathcal{G})$,*

$$\mathfrak{A} \models \forall x_1, \dots, x_{|w|+1} \big(I(x_1) \wedge \phi_w(x_1, \dots, x_{|w|+1}) \wedge T(x_{|w|+1}) \Rightarrow \bot \big).$$

The following lemma is proved by establishing a connection between the well-known Myhill-Nerode correspondence and ω-categoricity, under the addition of several auxiliary results from [29, 15].

▶ **Lemma 17.** *Let $\mathcal{G}$ be a context-free grammar. Then the following are equivalent:*

1. *$L(\mathcal{G})$ is regular.*
2. *$\Phi_{\mathcal{G}}$ is equivalent to a monadic Datalog sentence.*
3. *$\Phi_{\mathcal{G}}$ is equivalent to a GMSNP sentence.*
4. *$\mathrm{fm}(\Phi_{\mathcal{G}})$ is the CSP of a finite structure.*
5. *$\mathrm{fm}(\Phi_{\mathcal{G}})$ is the CSP of an ω-categorical structure.*
6. *$\mathrm{fm}(\Phi_{\mathcal{G}})$ is the age of a reduct of a finitely bounded homogeneous Ramsey structure.*
7. *$\mathrm{fm}(\Phi_{\mathcal{G}})$ is the age of an ω-categorical structure.*
8. *$\mathrm{fm}(\Phi_{\mathcal{G}})$ is finitely bounded homogenizable.*

Proof of Theorem 13. It is well-known that the questions whether $L(\mathcal{G})$ is regular for a given context-free grammar $\mathcal{G}$ is undecidable, see, e.g., Theorem 6.6.6 in [49]. Hence, all eight equivalent conditions in Lemma 17 are undecidable for $\mathcal{G}$. ◀

5 Open Questions

We proved the EXPSPACE-hardness of the amalgamation meta-problem. However, our methods rely heavily on the following three facts. First, symbols of arity > 2 allow us to simulate a restricted form of Datalog computation within one-point amalgamation diagrams. Second, Boolean combinations of atoms enable succinct representations of the Datalog rules. And third, symbols of unbounded arity enable storing exponential amount of information on a constant number of variables. We do not know how to extend our hardness result beyond EXPSPACE. In particular, it does not seem to be possible to reduce from any of the standard undecidable problems, which can be done for the closely related joint embedding property [22, 20]. Intuitively, the reason is that every representation of a run of a Turing machine in a finitely bounded class requires some sort of a successor predicate (see, e.g., [31]), and the successor predicate is never definable in any ω-categorical structure [8].

Open question: Is the amalgamation meta-problem decidable in EXPSPACE?

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley, 1995. URL: http://webdam.inria.fr/Alice/.

2 Ove Ahlman. Homogenizable structures and model completeness. *Arch. Math. Log.*, 55(7-8):977–995, 2016. doi:10.1007/s00153-016-0507-6.

3 Reza Akhtar and Alistair H. Lachlan. On countable homogeneous 3-hypergraphs. *Arch. Math. Log.*, 34(5):331–344, 1995. doi:10.1007/BF01387512.

4 Albert Atserias and Szymon Torunczyk. Non-homogenizable classes of finite structures. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 – September 1, 2016, Marseille, France*, volume 62 of *LIPIcs*, pages 16:1–16:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CSL.2016.16.

5 Franz Baader and Jakub Rydval. Using model theory to find decidable and tractable description logics with concrete domains. *J. Autom. Reason.*, 66(3):357–407, 2022. doi:10.1007/s10817-022-09626-2.

6 Libor Barto, Michael Kompatscher, Miroslav Olsák, Trung Van Pham, and Michael Pinsker. The equivalence of two dichotomy conjectures for infinite domain constraint satisfaction problems. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. available at arXiv:1612.07551. doi:10.1109/LICS.2017.8005128.

7 Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive datalog, csp, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44, 2014. doi:10.1145/2661643.

8 Manuel Bodirsky. *Complexity of Infinite-Domain Constraint Satisfaction*. Cambridge University Press, 2021. doi:10.1017/9781107337534.

9 Manuel Bodirsky and Bertalan Bodor. Canonical polymorphisms of ramsey structures and the unique interpolation property. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 – July 2, 2021*, pages 1–13. IEEE, IEEE, 2021. doi:10.1109/LICS52264.2021.9470683.

10 Manuel Bodirsky, David Bradley-Williams, Michael Pinsker, and András Pongrácz. The universal homogeneous binary tree. *J. Log. Comput.*, 28(1):133–163, 2018. doi:10.1093/logcom/exx043.

11 Manuel Bodirsky and Víctor Dalmau. Datalog and constraint satisfaction with infinite templates. *J. Comput. Syst. Sci.*, 79(1):79–100, 2013. A preliminary version appeared in the proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS'05). doi:10.1016/j.jcss.2012.05.012.

12 Manuel Bodirsky and Martin Grohe. Non-dichotomies in constraint satisfaction complexity. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II – Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 184–196. Springer, July 2008. doi:10.1007/978-3-540-70583-3_16.

13 Manuel Bodirsky, Peter Jonsson, and Van Trung Pham. The complexity of phylogeny constraint satisfaction problems. *ACM Trans. Comput. Log.*, 18(3):23:1–23:42, 2017. doi:10.1145/3105907.

14 Manuel Bodirsky and Jan Kára. The complexity of temporal constraint satisfaction problems. *J. ACM*, 57(2):9:1–9:41, 2010. doi:10.1145/1667053.1667058.

15 Manuel Bodirsky, Simon Knäuer, and Florian Starke. ASNP: A tame fragment of existential second-order logic. In Marcella Anselmo, Gianluca Della Vedova, Florin Manea, and Arno Pauly, editors, *Beyond the Horizon of Computability – 16th Conference on Computability in Europe, CiE 2020, Fisciano, Italy, June 29 – July 3, 2020, Proceedings*, volume 12098 of *Lecture Notes in Computer Science*, pages 149–162. Springer, Springer, 2020. doi:10.1007/978-3-030-51466-2_13.

16 Manuel Bodirsky, Florent R. Madelaine, and Antoine Mottet. A proof of the algebraic tractability conjecture for monotone monadic SNP. *SIAM J. Comput.*, 50(4):1359–1409, 2021. doi:10.1137/19M128466X.

17 Manuel Bodirsky and Antoine Mottet. Reducts of finitely bounded homogeneous structures, and lifting tractability from finite-domain constraint satisfaction. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 623–632. ACM, 2016. available at arXiv:1601.04520. doi:10.1145/2933575.2934515.

18 Manuel Bodirsky, Michael Pinsker, and András Pongrácz. Projective clone homomorphisms. *J. Symb. Log.*, 86(1):148–161, 2021. doi:10.1017/jsl.2019.23.

19 Manuel Bodirsky and Jakub Rydval. On the descriptive complexity of temporal constraint satisfaction problems. *J. ACM*, 70(1):2:1–2:58, 2023. doi:10.1145/3566051.

20 Manuel Bodirsky, Jakub Rydval, and André Schrottenloher. Universal horn sentences and the joint embedding property. *Discret. Math. Theor. Comput. Sci.*, 23(2), 2021. doi:10.46298/dmtcs.7435.

21 Mikolaj Bojanczyk, Luc Segoufin, and Szymon Torunczyk. Verification of database-driven systems via amalgamation. In Richard Hull and Wenfei Fan, editors, *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA – June 22–27, 2013*, pages 63–74. ACM, 2013. doi:10.1145/2463664.2465228.

22 Samuel Braunfeld. The undecidability of joint embedding and joint homomorphism for hereditary graph classes. *Discret. Math. Theor. Comput. Sci.*, 21(2), 2019. available at arXiv:1903.11932. doi:10.23638/DMTCS-21-2-9.

23 Andrei A. Bulatov. A dichotomy theorem for nonuniform csps. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 319–330. IEEE, IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.37.

24 Jakub Bulin, Dejan Delic, Marcel Jackson, and Todd Niven. A finer reduction of constraint problems to digraphs. *Log. Methods Comput. Sci.*, 11(4), 2015. doi:10.2168/LMCS-11(4:18)2015.

25 Gregory L. Cherlin. Forbidden substructures and combinatorial dichotomies: WQO and universality. *Discret. Math.*, 311(15):1543–1584, 2011. doi:10.1016/j.disc.2011.03.014.

26 Gregory L. Cherlin. *Homogeneous ordered graphs, metrically homogeneous graphs, and beyond*, volume 1. Cambridge University Press, 2022. doi:10.1017/9781009229661.

27 Gregory L. Cherlin, Saharon Shelah, and Niandong Shi. Universal graphs with forbidden subgraphs and algebraic closure. *Advances in Applied Mathematics*, 22(4):454–491, 1999. doi:10.1006/aama.1998.0641.

28 Jacinta Covington. Homogenizable relational structures. *Illinois Journal of Mathematics*, 34(4):731–743, 1990. doi:10.1215/ijm/1255988065.

29 Péter L. Erdös, Claude Tardif, and Gábor Tardos. Caterpillar dualities and regular languages. *SIAM J. Discret. Math.*, 27(3):1287–1294, 2013. doi:10.1137/120879270.

30 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998. doi:10.1137/S0097539794266766.

31 Haim Gaifman, Harry G. Mairson, Yehoshua Sagiv, and Moshe Y. Vardi. Undecidable optimization problems for database logic programs. *J. ACM*, 40(3):683–713, 1993. doi:10.1145/174130.174142.

32 Nicola Gigante, Andrea Micheli, Angelo Montanari, and Enrico Scala. Decidability and complexity of action-based temporal planning over dense time. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 9859–9866. AAAI Press, 2020. doi:10.1609/aaai.v34i06.6539.

33 Wilfrid Hodges. *A shorter model theory*. Cambridge University Press, Cambridge, 1997.

34 Jan Hubička and Jaroslav Nešetřil. Homomorphism and embedding universal structures for restricted classes. *J. Multiple Valued Log. Soft Comput.*, 27(2-3):229–253, 2016. available at arXiv:0909.4939. URL: http://www.oldcitypublishing.com/journals/mvlsc-home/mvlsc-issue-contents/mvlsc-volume-27-number-2-3-2016/mvlsc-27-2-3-p-229-253/.

35 Jan Hubička and Jaroslav Nešetřil. All those ramsey classes (ramsey classes with closures and forbidden homomorphisms). *Advances in Mathematics*, 356:1–89, 2019. doi:10.1016/j.aim.2019.106791.

36 Julia F. Knight and Alistair H. Lachlan. Shrinking, stretching, and codes for homogeneous structures. In *Classification Theory: Proceedings of the US-Israel Workshop on Model Theory in Mathematical Logic held in Chicago, Dec. 15–19, 1985*, volume 1292, pages 192–229. Springer, Springer, Berlin, Heidelberg, 2006. doi:10.1007/BFb0082239.

37 Gábor Kun. Constraints, MMSNP and expander relational structures. *Comb.*, 33(3):335–347, 2013. doi:10.1007/s00493-013-2405-4.

38 Alistair H. Lachlan. Homogeneous structures. In *Proceedings of the International Congress of Mathematicians*, volume 1, pages 314–321, Berkeley, 1986. AMS.

39 Carsten Lutz and Maja Milicic. A tableau algorithm for description logics with concrete domains and general tboxes. *J. Autom. Reason.*, 38(1-3):227–259, 2007. doi:10.1007/s10817-006-9049-7.

40 Dugald Macpherson. A survey of homogeneous structures. *Discret. Math.*, 311(15):1599–1634, 2011. doi:10.1016/j.disc.2011.01.024.

41 Florent R. Madelaine. Universal structures and the logic of forbidden patterns. *Log. Methods Comput. Sci.*, 5(2), 2009. doi:10.2168/LMCS-5(2:13)2009.

42 Florent R. Madelaine and Iain A. Stewart. Some problems not definable using structure homomorphisms. *Ars Comb.*, 67:153–160, 2003.

43 Antoine Mottet. Promise and infinite-domain constraint satisfaction. In Aniello Murano and Alexandra Silva, editors, *32nd EACSL Annual Conference on Computer Science Logic, CSL 2024, February 19-23, 2024, Naples, Italy*, volume 288 of *LIPIcs*, pages 41:1–41:19. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.CSL.2024.41.

44 Antoine Mottet and Michael Pinsker. Smooth approximations and csps over finitely bounded homogeneous structures. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2–5, 2022*, pages 36:1–36:13. ACM, 2022. available at arXiv:2011.03978. doi:10.1145/3531130.3533353.

45 Antoine Mottet, Michael Pinsker, and Tomáš Nagy. An order out of nowhere: a new algorithm for infinite-domain CSPs. In *51st EATCS International Colloquium on Automata, Languages and Programming (ICALP 2024)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. to appear, available at arXiv:2301.12977. doi:10.48550/arXiv.2301.12977.

46 Shan-Hwei Nienhuys-Cheng and Ronald de Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Computer Science*. Springer, 1997. doi:10.1007/3-540-62927-0.

47 Jakub Rydval. Homogeneity and homogenizability: Hard problems for the logic snp. *CoRR*, abs/2108.00452, 2021. doi:10.48550/arXiv.2108.00452.

48 François Schwarzentruber. The complexity of tiling problems. *CoRR*, abs/1907.00102, 2019. doi:10.48550/arXiv.1907.00102.

49 Jeffrey O. Shallit. *A second course in formal languages and automata theory*. Cambridge University Press, 2008. URL: http://www.cambridge.org/gb/knowledge/isbn/item1173872/?site_locale=en_GB.

50 Peter van Emde Boas. The convenience of tilings. In *Complexity, Logic, and Recursion Theory*, pages 331–363. CRC Press, 2019. doi:10.1201/9780429187490.

51 Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 331–342. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.38.

52 Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *J. ACM*, 67(5):30:1–30:78, 2020. doi:10.1145/3402029.

Identifying Tractable Quantified Temporal Constraints Within Ord-Horn

Jakub Rydval
Technische Universität Wien, Austria

Žaneta Semanišinová
Technische Universität Dresden, Germany

Michał Wrona
Jagiellonian University, Kraków, Poland

Abstract

The constraint satisfaction problem, parameterized by a relational structure, provides a general framework for expressing computational decision problems. Already the restriction to the class of all finite structures forms an interesting microcosm on its own, but to express decision problems in temporal reasoning one has to take a step beyond the finite-domain realm. An important class of templates used in this context are temporal structures, i.e., structures over $\mathbb{Q}$ whose relations are first-order definable using the usual countable dense linear order without endpoints.

In the standard setting, which allows only existential quantification over input variables, the complexity of finite and temporal constraints has been fully classified. In the quantified setting, i.e., when one also allows universal quantifiers, there is only a handful of partial classification results and many concrete cases of unknown complexity. This paper presents a significant progress towards understanding the complexity of the quantified constraint satisfaction problem for temporal structures. We provide a complexity dichotomy for quantified constraints over the Ord-Horn fragment, which played an important role in understanding the complexity of constraints both over temporal structures and in Allen's interval algebra. We show that all problems under consideration are in P or coNP-hard. In particular, we determine the complexity of the quantified constraint satisfaction problem for $(\mathbb{Q}; x = y \Rightarrow x \geq z)$, hereby settling a question open for more than ten years.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Logic; Theory of computation → Computational complexity and cryptography

Keywords and phrases constraint satisfaction problems, quantifiers, dichotomy, temporal reasoning, Ord-Horn

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.151

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2402.09187` [17]

Funding *Jakub Rydval*: This research was funded in whole or in part by the Austrian Science Fund (FWF) [I 5948]. For the purpose of Open Access, the authors have applied a CC BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission.
Žaneta Semanišinová: The author has been funded by the European Research Council (Project POCOCOP, ERC Synergy Grant 101071674) and by the DFG (Project FinHom, Grant 467967530). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.
Michał Wrona: The author is partially supported by National Science Centre, Poland grant number 2020/37/B/ST6/01179.

Acknowledgements The authors thank Dmitriy Zhuk for many inspiring discussions on the topic, and the anonymous reviewers for many helpful suggestions.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 151; pp. 151:1–151:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The constraint satisfaction problem (CSP) of a structure $\mathfrak{B}$ in a finite relational signature τ, denoted by CSP($\mathfrak{B}$), is the problem of deciding whether a given primitive positive τ-sentence holds in $\mathfrak{B}$. The class of all *finite-domain* CSPs, i.e., where $\mathfrak{B}$ can be chosen finite, famously constitutes a large fragment of NP that admits a dichotomy between P and NP-completeness [19]. Quantified constraint satisfaction problems (QCSPs) generalize CSPs by allowing both existential *and* universal quantification over input variables. The complexity of such problems is much less understood already for finite structures, the state of the art being a complexity classification for QCSPs of finite structures with all unary relations and three-element structures with all singleton unary relations [21]. For infinite structures, the investigations essentially follow the CSP programme, which was initiated by the study of the CSPs of structures over $\mathbb{N}$ (or $\mathbb{Q}$) whose relations are definable by Boolean combinations of equalities and disequalities, the so-called *equality structures* [6]. The full complexity classification for quantified equality constraints was completed quite recently [22], by resolving the long-standing question of determining the complexity of QCSP($\mathbb{Q}$; D), where

$$\mathrm{D} \coloneqq \{(x, y, z) \in \mathbb{Q}^3 \mid x = y \Rightarrow x = z\}.$$

This question was left open in [3], where all the remaining results have been provided. The next in line are *temporal structures*, which are structures with domain $\mathbb{Q}$ whose relations are first-order definable over $\{<\}$, where $<$ interprets as the usual unbounded dense linear order. The relations of such structures are called *temporal*.

By definition, temporal structures form a richer class than equality structures. While the complexity of temporal CSPs has been classified more than a decade ago [7], there is only a handful of partial classification results regarding the complexity of temporal QCSPs [4, 10, 11, 12, 18]. Yet, already from this limited amount of available data it is apparent that the majority of the pathological cases is concentrated in the *Ord-Horn* (OH) fragment, we elaborate on this below. The OH fragment comprises all temporal structures whose relations are definable by an OH formula, i.e., a conjunction of clauses of the form

$$(x_1 \neq y_1 \vee \cdots \vee x_k \neq y_k \vee x_{k+1} \geq y_{k+1}) \tag{1}$$

for $k \geq 0$, where the last disjunct is optional and some variables might be identified [2].

1.1 Ord-Horn

OH was first introduced and used by Nebel and Bürckert to describe a maximally tractable constraint language containing all basic relations on Allen's interval algebra [16]. For a full classification of maximally tractable subalgebras of Allen, see [15]. In the context of CSPs over temporal structures, OH is not even a maximally tractable language as it is properly contained in two of the nine maximally tractable fragments characterized by the eight binary operations *min*, *max*, *mx*, *dual mx*, *mi*, *dual mi*, $\ell\ell$, *dual* $\ell\ell$, and a constant operation [7]. The *dual* of an operation f on $\mathbb{Q}$ is the operation $(x_1, \ldots, x_n) \mapsto -f(-x_1, \ldots, -x_n)$, e.g., *max* is the dual of *min*. The description of maximally tractable languages by operations is typical for the so-called algebraic approach to constraint satisfaction problems. For the sake of the reader unfamiliar with this approach, we simply refer to a maximally tractable temporal language characterized by an operation *op* as the *op* fragment and refrain from defining the operations. For example, we write "the *min* fragment" or "the *max* fragment." The question which of the nine fragments are also maximal w.r.t. tractability of the QCSP

was investigated in [4], and answered positively in the first four cases. The answer is negative in the last three cases [22], and the question remains open for *mi* and *dual mi* fragments. In the intersections of $\ell\ell$, *mi* and *dual mi* fragments lie the OH structures $(\mathbb{Q}; \mathrm{M}^+)$ and $(\mathbb{Q}; \mathrm{M}^-)$, respectively, where

$$\mathrm{M}^+ \coloneqq \{(x,y,z) \in \mathbb{Q}^3 \mid x = y \Rightarrow x \geq z\} \quad \text{and} \quad \mathrm{M}^- \coloneqq \{(x,y,z) \in \mathbb{Q}^3 \mid x = y \Rightarrow x \leq z\}.$$

Determining the complexity of $\mathrm{QCSP}(\mathbb{Q}; \mathrm{M}^+)$ was posed as an open question in [4]; it could have been anywhere between PTIME and PSPACE. Note that its counterpart $\mathrm{QCSP}(\mathbb{Q}; \mathrm{M}^-)$ is essentially the same problem with the order reversed.

Apart from temporal structures preserved by a constant operation, OH captures precisely those temporal structures whose CSP is solvable by local consistency checking [8]. This well-known generic preprocessing algorithm can be formulated for any CSP satisfying some reasonable structural assumptions [5], and thus OH constraints are fairly well understood from the CSP perspective. However, the analysis of OH constraints in the quantified setting requires a surprisingly large amount of creativity. As a simple example, already $\mathrm{QCSP}(\mathbb{Q}; R)$ for the OH relation R defined by $(x_1 \neq x_2 \vee x_3 \geq x_4) \wedge \phi$ is in PTIME if ϕ equals $(x_3 \geq x_1) \wedge (x_1 \geq x_3) \wedge (x_3 \neq x_4)$ [12], coNP-complete if ϕ equals $(\bigwedge_{i,j \in \{1,2\}} x_i \neq x_{j+2})$ [20], and PSPACE-complete if ϕ is the empty conjunction [22].

The class of *Guarded Ord-Horn* (GOH) formulas [12] is defined inductively. In the base case we are allowed to take OH formulas of the form $(x \leq y)$, $(x_1 \neq y_1 \vee \cdots \vee x_k \neq y_k)$, or $(x \neq x_1 \vee \cdots \vee x \neq x_k) \vee (x < y) \vee (y \neq y_1 \vee \cdots \vee y \neq y_\ell)$. In the induction step we can form formulas of the form $\psi_1 \wedge \psi_2$ or $(x_1 \leq y_1 \vee \cdots \vee x_k \leq y_k) \wedge (x_1 \neq y_1 \vee \cdots \vee x_k \neq y_k \vee \psi)$, where ψ, ψ_1, ψ_2 are GOH formulas. Thus, newly added disequalities are guarded by atomic $\{\leq\}$-formulas. A GOH structure may only contain temporal relations definable by GOH formulas. Observe that the tractable template from the previous paragraph is GOH.

▶ **Theorem 1** ([12]). *Let $\mathfrak{B}$ be a GOH structure. Then* $\mathrm{QCSP}(\mathfrak{B})$ *is in* PTIME.

The tractability result from [12] is conceptually simple and based on pebble games generalizing local consistency methods. At the same time, all quantified OH constraints outside of GOH are coNP-hard or admit a LOGSPACE reduction from $\mathrm{QCSP}(\mathbb{Q}; \mathrm{M}^+)$.

▶ **Theorem 2** ([18]). *Let $\mathfrak{B}$ be an OH structure. Then one of the following holds.*

- *$\mathfrak{B}$ is GOH.*
- $\mathrm{QCSP}(\mathfrak{B})$ *is* coNP*-hard.*
- *$\mathfrak{B}$ primitively positively defines* M^+ *or* M^-.

There was a prospect that $\mathrm{QCSP}(\mathbb{Q}; \mathrm{M}^+)$ would be PSPACE-hard, because the PSPACE-hardness proof from [22] for $\mathrm{QCSP}(\mathbb{Q}; \mathrm{D})$, when adjusted appropriately, almost yields a proof of PSPACE-hardness for this QCSP. In that case, Theorems 1 and 2 would immediately yield a dichotomy between P and coNP-hardness for quantified OH constraints. However, it turns out that the situation is more complicated, as we explain below.

1.2 Contributions

On the one hand, we prove tractability for $\mathrm{QCSP}(\mathbb{Q}; \mathrm{M}^+)$, and thereby provide a positive answer to an open question from [4]. This is the main technical contribution of the present paper, and the proof stretches over the entirety of Section 3.

In a certain sense, the presented algorithm generalizes local consistency methods. We iteratively expand a given instance Φ of $\mathrm{QCSP}(\mathfrak{B})$ by constraints associated to relations whose arity is bounded by the size of Φ and which have short primitive positive definitions

in $\mathfrak{B}$, until a fixed-point is reached. The condition for the expansion by these constraints is tested using an oracle for $\mathrm{CSP}(\mathfrak{B}, <)$. The algorithm is thus not very far from the well-known framework of Datalog with existential rules [1, 9].

▶ **Theorem 3.** $\mathrm{QCSP}(\mathbb{Q}; \mathrm{M}^+)$ *is in* PTIME.

Our tractability result naturally extends to the QCSPs of those OH structures which can be expressed in $(\mathbb{Q}; \mathrm{M}^+, \neq)$ using primitive positive definitions (see Proposition 6). We show that the set of these structures coincides with the intersection of the OH fragment with the $\pi\pi$ fragment. Here by $\pi\pi$ [1] we refer to the "*projection-projection*" operation from [7], which played an important role in identifying the maximally tractable temporal CSP languages covered by *min*, *mi* and *mx*. In the present paper, we introduce the $\pi\pi$ fragment using the syntactic description obtained in [4]. For a definition of the operation $\pi\pi$, see [7]. The $\pi\pi$ fragment consists of all temporal relations definable by a conjunction of clauses of the form

$$(x \neq y_1 \vee \cdots \vee x \neq y_k \vee x \geq z_1 \vee \cdots \vee x \geq z_\ell) \tag{2}$$

for $k, l \geq 0$. The *dual* $\pi\pi$ fragment is obtained by replacing every instance of $\geq$ in (2) by $\leq$.

▶ **Corollary 4.** $\mathrm{QCSP}(\mathfrak{B})$ *is in* PTIME *if* $\mathfrak{B}$ *is an OH structure in which every relation is definable by a conjunction of clauses of the form*

$$(x \neq y_1 \vee \cdots \vee x \neq y_k \vee x \geq z) \tag{3}$$

for $k \geq 0$ *and where the last disjunct* $(x \geq z)$ *may be omitted. The above condition is satisfied if and only if* $\mathfrak{B}$ *is contained in the intersection of the OH fragment and the* $\pi\pi$ *fragment.*

On the other hand, we confirm that $\mathrm{QCSP}(\mathbb{Q}; \mathrm{M}^+)$ indeed walks a very fine line between tractability and hardness. We show that, if M^+ is combined with any OH relation R that is not contained in the $\pi\pi$ fragment, then the resulting QCSP becomes coNP-hard, even if $\mathrm{QCSP}(\mathbb{Q}; R)$ is tractable. Intuitively, either $(\mathbb{Q}; \mathrm{M}^+, R)$ already primitively positively defines D and we use the PSPACE-hardnees proof from [22] directly, or we replace each constraint of the form $\mathrm{D}(x, y, z)$ in the proof by $\mathrm{M}^+(x, y, z) \wedge \mathrm{M}^+(z, z, x)$. The latter, however, is not entirely conditional, and certain issues arise due to the transitivity of $\leq$. These issues can be partially (but not entirely) resolved using constraints associated to

$$\check{\mathrm{Z}} \coloneqq \{(x_1, y_1, x_2, y_2) \in \mathbb{Q}^4 \mid (x_1 \neq y_1 \vee x_2 \neq y_2) \wedge (y_1 < y_2)\},$$

which is quantified primitively positively definable in $(\mathbb{Q}; \mathrm{M}^+, R)$, ultimately leaving us with a proof of coNP-hardness.

By a careful combination of syntactic pruning arguments, Theorem 2, and a new coNP-hardness proof inspired by the PSPACE-hardness proof from [22], we prove coNP-hardness in all cases for which tractability does not follow from Theorem 1, Corollary 4 or its analogue for *dual* $\pi\pi$, i.e., where $\geq$ is replaced with $\leq$ in (3). This leads to the following dichotomy for quantified OH constraints.

▶ **Theorem 5.** *Let* $\mathfrak{B}$ *be an OH structure. Then* $\mathrm{QCSP}(\mathfrak{B})$ *is solvable in polynomial time if* $\mathfrak{B}$ *is GOH, contained in the* $\pi\pi$ *fragment, or in the dual* $\pi\pi$ *fragment. Otherwise,* $\mathrm{QCSP}(\mathfrak{B})$ *is* coNP*-hard.*

[1] In contrast to previous literature on temporal CSPs, we deviate from the notation pp from [7] that clashes with the shortcut for "primitive positive" and use $\pi\pi$ instead.

We believe that the methods used in this paper will also prove useful in identifying the complexity of quantified temporal constraints outside of OH, e.g., in the case of *mi* or $\pi\pi$. Omitted proofs can be found in the long version of the article available on arXiv [17], where we also provide more details on the algebraic approach and relevant operations on $\mathbb{Q}$.

2 Preliminaries

2.1 First-order structures

The set $\{1, \dots, n\}$ is denoted by $[n]$. In the present paper, we consider structures $\mathfrak{A} = (A; R_1, \dots, R_k)$ over a finite relational signature τ. For the sake of simplicity, we often use the same symbol R for both the relation $R^{\mathfrak{A}}$ and the relational symbol R. An *expansion* of $\mathfrak{A}$ is a σ-structure $\mathfrak{B}$ with $A = B$ such that $\tau \subseteq \sigma$ and $R^{\mathfrak{B}} = R^{\mathfrak{A}}$ for each $R \in \tau$. We write $(\mathfrak{A}, R)$ for the expansion of $\mathfrak{A}$ by the relation R over A.

We assume that the reader is familiar with classical first-order logic; we allow the first-order formulas $x = y$ and $\bot$ (the nullary falsity predicate). Let T be a set of first-order τ-sentences over a common signature τ and ϕ, ψ τ-formulas whose free variables are among $\bar{x}$. We say that ϕ *entails* ψ w.r.t. T if $\mathfrak{A} \models \forall \bar{x}(\phi \Rightarrow \psi)$ holds for all models $\mathfrak{A}$ of T. We do not explicitly mention T if it is clear from the context, e.g., the theory of linear orders. A first-order τ-formula ϕ is *primitive positive* (pp) if it is of the form $\exists x_1, \dots, x_m(\phi_1 \wedge \dots \wedge \phi_n)$, where each ϕ_i is *atomic*, i.e., of the form $\bot$, $(x_i = x_j)$, or $R(x_{i_1}, \dots, x_{i_\ell})$ for some $R \in \tau$. *Quantified primitive positive* (qpp) formulas generalize pp-formulas by allowing both existential *and* universal quantification. If ϕ and ψ are (q)pp-formulas, then $\phi \wedge \psi$ can be rewritten into an equivalent (q)pp-formula, so we treat such formulas as (q)pp-formulas as well.

The *(quantified) constraint satisfaction problem* for a structure $\mathfrak{B}$, denoted by (Q)CSP($\mathfrak{B}$), is the computational problem of deciding whether a given (q)pp τ-sentence holds in $\mathfrak{B}$. By *constraints*, we refer to the conjuncts in the quantifier-free part of a given (Q)CSP instance of (Q)CSP($\mathfrak{B}$). In the QCSP framework, we usually think of an instance as a game between two players: an *existential player* (EP) and a *universal player* (UP) who assign values to the existentially and universally quantified variables, respectively. To every moment of the game we associate a partial function $[\![\cdot]\!]$ from the variables into the domain of the parametrizing structure describing values assigned to the variables by either of the players. The instance is true if and only if the EP has a winning strategy in this game, i.e., can respond to all moves of the UP while keeping all constraints satisfied. Otherwise, the instance is false and the UP has a winning strategy, i.e., can violate a constraint regardless of the moves of the EP.

If $\mathfrak{A}$ is a τ-structure and $\phi(x_1, \dots, x_n)$ is a τ-formula with free variables $x_1, \dots, x_n$, then the relation $\{(a_1, \dots, a_n) \in A^n \mid \mathfrak{A} \models \phi(a_1 \dots, a_n)\}$ is the *relation defined by* ϕ *in* $\mathfrak{A}$, and denoted by $\phi^{\mathfrak{A}}$. Let S be a set of τ-formulas. We say that a relation R has a *S-definition* in $\mathfrak{A}$, or that $\mathfrak{A}$ *S-defines* R, if R equals $\phi^{\mathfrak{A}}$ for some $\phi \in S$. For instance, S can be the set of all quantifier-free or primitive positive formulas over τ. We might also say that a relation R S-defines another relation R' if the structure $(A; R)$ S-defines R'. The next proposition is folklore in the constraint satisfaction literature.

▶ **Proposition 6** ([2, 3]). *Let $\mathfrak{A}, \mathfrak{B}$ be structures with the same domain. If every relation of $\mathfrak{B}$ is (q)pp-definable in $\mathfrak{A}$, then* (Q)CSP($\mathfrak{B}$) *reduces to* (Q)CSP($\mathfrak{A}$) *in* LOGSPACE.

2.2 Temporal structures

Since $(\mathbb{Q}; <)$ has quantifier-elimination [13], every temporal relation is in fact quantifier-free-definable in $(\mathbb{Q}; <)$. We may further assume that every quantifier-free definition is in *conjunctive normal form* (CNF). We might sometimes refer to temporal relations directly

using their CNF-definitions. Also, it will sometimes be convenient to work with formulas over the structure $(\mathbb{Q};\leq,\neq)$ instead of the structure $(\mathbb{Q};<)$, e.g., in the definitions of OH or $\pi\pi$ from the introduction. The following lemma is folklore.

▶ **Lemma 7** ([2]). *The OH and the (dual) $\pi\pi$ fragments are closed under expansions by pp-definable relations.*

From the syntactic descriptions (1) and (2) it is apparent that the fragments OH and $\pi\pi$ are incomparable. Their intersection consists of all temporal relations definable by a conjunction of clauses of the form (3). This can be shown using the following lemma.

▶ **Lemma 8.** *Let R be an OH relation defined by a quantifier-free formula ϕ in CNF over the signature $\{\leq,\neq\}$ containing a clause $\psi_1\vee\psi_2$, where ψ_1 is equivalent to $(x\geq z_1\vee\cdots\vee x\geq z_\ell)$ for some variables x and $z_1,\dots,z_\ell$. Then we may replace ψ_1 in ϕ by $(x\geq z_i)$ for some $i\in[\ell]$ so that the resulting formula still defines R.*

In the present article, the intersection of OH and the (*dual*) $\pi\pi$ fragment is the sole source of all newly identified tractable QCSPs. It is convenient to work with a finite relational basis. Recall the relations M^+ and M^- from the introduction. By Lemma 8 and Lemma 9 below, a temporal relation is OH and contained in the $\pi\pi$ fragment if and only if it is pp-definable in $(\mathbb{Q};\mathrm{M}^+,\neq)$. An analogous statement holds for *dual* $\pi\pi$ and $(\mathbb{Q};\mathrm{M}^-,\neq)$.

▶ **Lemma 9.** *The $(k+2)$-ary temporal relation defined by (3) has the pp-definition*

$$\exists h_1,\dots,h_{k+1}\Big((x=h_1)\wedge\big(\textstyle\bigwedge_{i\in[k]}\mathrm{M}^+(h_i,h_i,x)\wedge\mathrm{M}^+(h_i,y_i,h_{i+1})\big)\wedge(h_{k+1}=z)\Big).$$

Note that the length of the above pp-definition is linear in k, this will be relevant later in the proof of Theorem 3.

3 QCSP$(\mathbb{Q};\mathrm{M}^+)$ is in PTIME

In this section, we prove that QCSP$(\mathbb{Q};\mathrm{M}^+)$ can be solved in polynomial time using Algorithm 1. In the formulation of the algorithm, we view instances of QCSP$(\mathbb{Q};\mathrm{M}^+)$ as sentences over $\{\geq,\neq\}$ in prenex normal form whose quantifier-free part is in CNF.

We first need to fix some terminology. For the remainder of Section 3, Φ always denotes an arbitrary or explicitly specified instance of QCSP$(\mathbb{Q};\mathrm{M}^+)$, ϕ its quantifier-free part, and V the set if its variables. Furthermore, we denote the universal variables by $\mathrm{V}_\forall$ and the existential variables by $\mathrm{V}_\exists$. Let $\prec$ be the linear order on all variables of Φ in which they appear in the quantifier prefix of Φ. When we write $A\prec B$ for $A,B\subseteq V$, we mean $x\prec y$ for all $x\in A,y\in B$. In particular, this condition is trivially true if one of the two sets is empty.

▶ **Definition 10.** *For $x,z\in\mathrm{V}$, we define*
- *$x\equiv z$ if and only if x,z refer to the same variable,*
- *$x\preceq z$ if and only if $x\equiv z$ or $x\prec z$,*
- *$x\preceq_\forall z$ if and only if $x\equiv z$, or $x\prec z$ and $z\in\mathrm{V}_\forall$.*

For $u\in V$ and $A\subseteq V$, we define
- *$\uparrow_u\coloneqq\{y\in\mathrm{V}_\forall\mid u\preceq y\}$,*
- *$\uparrow_A\coloneqq\bigcup_{u\in A}\uparrow_u$ (recall that the empty union is empty).*

Note that the three binary relations in Definition 10 are transitive.

▶ **Definition 11.** *For every pair $x,z\in\mathrm{V}$, we define x-z-cut $\coloneqq\{u\in\mathrm{V}_\forall\mid\mathrm{V}_\exists\cap\{x,z\}\prec u\}\setminus\{z\}$.*

Observe that the definition of the x-z-cut depends on how x and z are quantified. The idea is that x-z-cut represents the universal variables that the UP can always make equal to x to trigger the condition $(x \geq z)$ via an entailed constraint of the form $\big((\bigwedge_{v \in A} x = v) \Rightarrow x \geq z\big)$. Since the UP has full control over the values of these variables with respect to x and z, they can be removed from the clauses added in the second last line in Algorithm 1.

Note that, by Lemma 9, the constraints added by Algorithm 1 correspond to relations which have pp-definitions in $(\mathbb{Q}; \mathrm{M}^+)$ of length linear in their arity. This means that satisfiability in Algorithm 1 can be tested using an oracle for $\mathrm{CSP}(\mathbb{Q}; \mathrm{M}^+, <)$, because we can simply replace each constraint by its pp-definition in $\mathrm{CSP}(\mathbb{Q}; \mathrm{M}^+)$ while only changing the size of Φ by a polynomial factor.

Algorithm 1 An algorithm for $\mathrm{QCSP}(\mathbb{Q}; \mathrm{M}^+)$.

Input: An instance Φ of $\mathrm{QCSP}(\mathbb{Q}; \mathrm{M}^+)$ with the quantifier-free part ϕ
Output: *true* or *false*
while ϕ *changes* **do**
  for $x, z, u \in \mathrm{V}$ **do**
    if ϕ *contains the clause* $(x \geq z)$ *or* $(z \geq x)$*, where* $x \prec z$ *and* $z \in \mathrm{V}_\forall$ **then**
      return *false*;
    if $\phi \wedge (\bigwedge_{v \in \uparrow_u \setminus \{x,z\}} x = v) \wedge (x < z)$ *is unsatisfiable* **then**
      expand ϕ by the clause $\big((\bigwedge_{v \in \uparrow_u \setminus (\{x,z\} \cup x\text{-}z\text{-cut})} x = v) \Rightarrow x \geq z\big)$;
return *true*;

▶ **Example 12.** Consider the instance Φ of $\mathrm{QCSP}(\mathbb{Q}; \mathrm{M}^+)$ defined by

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \exists x_5 \big((x_1 = x_2 \Rightarrow x_1 \geq x_5) \wedge (x_3 = x_2 \Rightarrow x_3 \geq x_4) \\ \wedge\ (x_5 = x_4 \Rightarrow x_5 \geq x_3) \wedge (x_3 \geq x_1) \wedge (x_5 \geq x_1)\big).$$

We claim that Algorithm 1 derives $(x_1 \geq x_4)$, and thereby rejects on Φ. We first observe that the formula $\phi \wedge (\bigwedge_{v \in \uparrow_u \setminus \{x_1, x_4\}} x_1 = v) \wedge (x_1 < x_4)$ is satisfiable for every $u \in \{x_1, \ldots, x_5\}$. Since $x_3, x_5 \in \mathrm{V}_\exists$, it is enough to show that $\phi \wedge (x_1 = x_2) \wedge (x_1 < x_4)$ is satisfiable, which is witnessed by any assignment satisfying $(x_5 = x_1 = x_2 < x_3 < x_4)$. On the other hand, $\phi \wedge (\bigwedge_{v \in \uparrow_{x_2} \setminus \{x_1, x_3\}} x_1 = v) \wedge (x_1 < x_3)$ is not satisfiable. Therefore, the algorithm expands ϕ by $(x_1 = x_2 \Rightarrow x_1 \geq x_3)$, because $x_4 \in x_1$-x_3-cut. But now $\phi \wedge (\bigwedge_{v \in \uparrow_{x_2} \setminus \{x_1, x_4\}} x_1 = v) \wedge (x_1 < x_4)$ is not satisfiable anymore. Since $x_2 \in x_1$-x_4-cut, the algorithm expands ϕ by $(x_1 \geq x_4)$.

As mentioned below Definition 11, Algorithm 1 rejects correctly because all constraints added during the run of the algorithm are entailed by Φ, see Lemma 13.

▶ **Lemma 13.** *Suppose that Algorithm 1 derives from Φ a constraint ψ. Then Φ is true if and only if Φ expanded by ψ is true.*

Proof. Denote by Ψ and Ψ' the sentences obtained from Φ by replacing ϕ with $\phi \wedge \psi$ and $\phi \wedge \psi'$, respectively, where

$$\psi \coloneqq \Big(\bigvee\nolimits_{v \in \uparrow_u \setminus (\{x,z\} \cup x\text{-}z\text{-cut})} x \neq v\Big) \vee (x \geq z) \quad \text{and} \quad \psi' \coloneqq \Big(\bigvee\nolimits_{v \in \uparrow_u \setminus \{x,z\}} x \neq v\Big) \vee (x \geq z).$$

Since $\phi \wedge \neg\psi'$ is unsatisfiable, we have that ϕ entails ψ'. It follows that Φ is true iff Ψ' is true. To complete the proof, we have to show that if Ψ' is true, then Ψ is true. We prove the contraposition and assume that the UP has a winning strategy on Ψ. If the UP wins

on Ψ by falsifying any clause different from ψ, then the very same choices lead the UP to falsifying the same clause in Ψ'. Otherwise, the UP falsifies ψ while playing on Ψ. Then the UP can play in the same way on Ψ' when it comes to the variables that occur both in ψ and ψ' and set all variables in x-z-cut to the same value as x. Note that this is possible since either x is universal or x precedes all variables in x-z-cut. It remains to show that ψ' is falsified. Clearly, all $\{\neq\}$-disjuncts are falsified. Since z is either universal or precedes all variables in x-z-cut, the disjunct $(x \geq z)$ is falsified as well, because it is falsified in ψ. ◀

Example 14 showcases how a winning strategy of the UP, obtainable implicitly from Lemma 13, might in fact be uniquely determined.

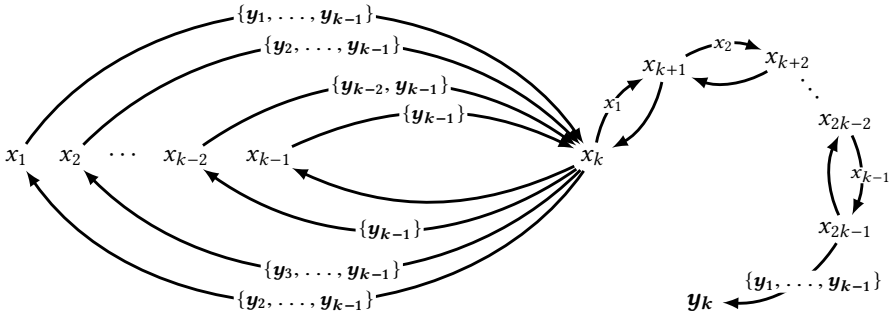


Figure 1 The quantifier-free part of Φ from Example 14.

▶ **Example 14.** Consider the instance $\Phi := \exists x_1 \forall y_1 \exists x_2 \forall y_2 \cdots \exists x_k \forall y_k \exists x_{k+1} \ldots \exists x_{2k-1}\ \phi$ with ϕ described by Figure 1, where an edge from x to z labeled with y stands for $\mathrm{M}^+(x, y, z)$. An edge from x to z labeled with some subset A of the universal variables stands for a constraint of the form $\big((\bigwedge_{v \in A} x = v) \Rightarrow x \geq z\big)$ already derived by Algorithm 1. Using Lemma 9, these edges can be appropriately replaced with pp-definitions, and thus Φ is well-defined.

We claim that the UP has the unique winning strategy on Φ of playing $[\![y_i]\!]$ equal to an arbitrary number $> [\![x_i]\!]$ if $i = k$ and $[\![x_1]\!] = \cdots = [\![x_k]\!]$, and equal to $min\{[\![x_1]\!], \ldots, [\![x_i]\!]\}$ otherwise. We start by showing that this is a winning strategy.

Suppose, on the contrary, that there exists an assignment $[\![\cdot]\!] \colon \mathrm{V} \to \mathbb{Q}$ of values to the variables witnessing that the EP has a counter-strategy to the strategy of the UP from above. First, consider the case where $[\![x_1]\!], \ldots, [\![x_k]\!]$ are not all equal. Suppose that $[\![x_k]\!] = min\{[\![x_1]\!], \ldots, [\![x_k]\!]\}$ and let $j \in [k]$ be the largest index such that $[\![x_j]\!] > [\![x_k]\!]$. Recall that the algorithm already derived the constraint $\psi_1 := \big((\bigwedge_{v \in \{y_{j+1}, \ldots, y_{k-1}\}} x_k = v) \Rightarrow x_k \geq x_j\big)$ on Φ. By the strategy of the UP, we have $[\![y_{j+1}]\!] = \cdots = [\![y_{k-1}]\!] = [\![x_{j+1}]\!] = [\![x_k]\!]$. But then ψ_1 is clearly not satisfied by $[\![\cdot]\!]$, a contradiction. Suppose now that $[\![x_k]\!] > min\{[\![x_1]\!], \ldots, [\![x_k]\!]\}$. Let $j \in [k]$ be the largest index such that $[\![x_j]\!] = min\{[\![x_1]\!], \ldots, [\![x_k]\!]\}$. Recall that the algorithm already derived the constraint $\psi_2 := \big((\bigwedge_{v \in \{y_j, \ldots, y_{k-1}\}} x_j = v) \Rightarrow x_j \geq x_k\big)$. By the strategy of the UP, we have $[\![y_j]\!] = \cdots = [\![y_{k-1}]\!] = [\![x_j]\!] < [\![x_k]\!]$. But then ψ_2 is clearly not satisfied by $[\![\cdot]\!]$, a contradiction. We conclude that $[\![x_1]\!] = \cdots = [\![x_k]\!]$. In this case, the UP played $[\![y_k]\!] > [\![x_k]\!]$. Since $[\![\cdot]\!]$ is a satisfying assignment, we must have $[\![x_k]\!] = [\![x_{k+1}]\!] = \cdots = [\![x_{2k-1}]\!]$. But then $[\![\cdot]\!]$ does not satisfy $\big((\bigwedge_{v \in \{y_1, \ldots, y_{k-1}\}} x_{2k-1} = v) \Rightarrow x_{2k-1} \geq y_k\big)$ because $[\![y_k]\!] > [\![x_k]\!] = [\![x_{2k-1}]\!] = [\![y_1]\!] = \cdots = [\![y_{k-1}]\!]$, a contradiction. We conclude that the strategy of the UP from above is a winning strategy.

The strategy of the UP is unique in the sense that, no matter what values the EP played for $x_1, \ldots, x_i$, if the UP deviates from his strategy at y_i, then the EP wins by playing $[\![x_{i+1}]\!] = \cdots = [\![x_{2k-1}]\!]$ equal to an arbitrary number $> max([\![x_i]\!], [\![y_i]\!])$ if $[\![x_i]\!] \neq [\![y_i]\!]$ and equal to $min\{[\![x_1]\!], \ldots, [\![x_i]\!]\}$ otherwise.

Table 1 The inference rules of the proof system $\mathcal{P}$. Here I stands for "initialize," S for "simplify," T for "transitivity," R for "reject," A for "alternative transitivity," and C for "constraint".

§I	$\mathcal{P}(x,x;\emptyset) :- x \in V$
§S	$\mathcal{P}(x,z;A \setminus x\text{-}z\text{-cut}) :- \mathcal{P}(x,z;A)$
§T	$\mathcal{P}(x,z;A) :- \mathcal{P}(x,y;A) \wedge \mathcal{P}(y,z;\emptyset)$
§R	$\bot :-$ 1. $\mathcal{P}(x,z;\emptyset)$; 2. $x \prec z$ and $z \in \mathrm{V}_\forall$, or $z \prec x$ and $x \in \mathrm{V}_\forall$
§A	$\mathcal{P}(x_i,z;A \cup B \cup (\{x_1,x_2\} \setminus \{x_i\})) :-$ 1. $\mathcal{P}(x_1,y;A) \wedge \mathcal{P}(y,x_2;\emptyset) \wedge \mathcal{P}(y,z;B)$; 2. $(\{x_1,x_2\} \setminus \{x_i\}) \subseteq \mathrm{V}_\forall$ $(i \in \{1,2\})$
§C	$\mathcal{P}(x_i,z;A \cup B \cup (\{x_1,x_2,x_3,x_4\} \setminus \{x_i\})) :-$ 1. $\mathcal{P}(x_1,u;A) \wedge \mathcal{P}(u,x_2;\emptyset)$; 2. $\mathcal{P}(x_3,v;B) \wedge \mathcal{P}(v,x_4;\emptyset)$; 3. $(\{x_1,x_2,x_3,x_4\} \setminus \{x_i\}) \subseteq \mathrm{V}_\forall$ $(i \in \{1,2,3,4\})$; 4. $(u{=}v \Rightarrow u \geq z)$ or $(v{=}u \Rightarrow v \geq z)$ in ϕ

Finally, we show that the algorithm derives *false*. In the first run of the main loop, we get $(x_{i+1} \geq x_i)$ for all $i \in \{1,\dots,k-2\}$. Assuming previously derived constraints, we get $(x_k \geq x_i)$ for all $i \in \{1,\dots,k-1\}$ (purely by transitivity). Now it is possible to derive $(x_i = y_i \Rightarrow x_i \geq x_{i+1})$ for all $i \in \{1,\dots,k-1\}$. In the final step, we get $(x_1 \geq y_k)$, again, simply by invoking an oracle for $\mathrm{CSP}(\mathbb{Q}; \mathrm{M}^+, <)$, which makes the algorithm reject.

3.1 False instances

The goal of this subsection is to reformulate the condition for rejection by Algorithm 1 within a certain proof system $\mathcal{P}$ operating on Φ, whose rules are given in Table 1 using a Datalog-style syntax. The proof system syntactically derives predicates of the form $\mathcal{P}(x, y; A)$ with $x, y \in V$ and $A \subseteq \mathrm{V}_\forall$ (on the left hand side of $:-$) from other predicates of this form derived earlier and the information encoded in Φ (on the right hand side of $:-$).

We shall now provide some intuition behind the formulation of the proof system. The idea is that an expression $\mathcal{P}(x, z; A)$ should capture a constraint $\left((\bigwedge_{v \in A} x = v) \Rightarrow x \geq z\right)$ entailed by Φ, where A only consists of universal variables. Assuming the adopted semantics, §T and §A just describe natural properties of such expressions, and §S and §R witness consequences of the quantification over the variables. The combination of §I and §C captures precisely the situations where the UP can indirectly enforce the identification of two (potentially existential) variables within a constraint in ϕ. In particular, it can be used to introduce $\mathcal{P}(u, z; \{v\})$ for conjuncts $(u = v \Rightarrow v \geq z)$ in ϕ with v universally quantified as follows. The proof system first derives $\mathcal{P}(u, u; \emptyset)$ and $\mathcal{P}(v, v; \emptyset)$ using §I. Then it uses §C to derive $\mathcal{P}(u, z; \{v\})$ by identifying x_1, x_2 with u and x_3, x_4 with v.

The reader might naturally ask why we cannot obtain a polynomial-time algorithm by just closing Φ under the rules of the proof system with a suitable form of fixed-point semantics. The reason is that, already under the least fixed-point semantics, the proof system might derive exponentially many expressions of the form $\mathcal{P}(x, z; A)$. Such a situation occurs, e.g., in Example 18 and in the case of the constraint paths in ϕ as defined in the proof of Lemma 25.

The precise connection between the proof system and Algorithm 1 is captured by Lemma 15. Note that Lemma 15 in particular implies that Algorithm 1 rejects whenever the proof system derives $\bot$. When combined with Lemma 13 and Lemma 19 (proved later in Section 3.2), we get that this is in fact the only situation in which Algorithm 1 rejects. Lemma 15 can be proved by a straightforward induction on the length of the derivation sequences within $\mathcal{P}$.

▶ **Lemma 15.** *Suppose that $\mathcal{P}(x, z; A)$ is derived by the proof system and $z \notin A$. Then Algorithm 1 expands ϕ by the clause*

$$\Big(\big(\bigwedge\nolimits_{v \in \uparrow_A \setminus (\{x,z\} \cup x\text{-}z\text{-}cut)} x = v\big) \Rightarrow x \geq z\Big).$$

In particular, it expands ϕ by the clause $(x \geq z)$ for every derived $\mathcal{P}(x, z; \emptyset)$.

In the proof of Lemma 15, we use the following simple observation.

▷ Claim 16. For every pair $(x, z) \in V^2$, and every $A \subseteq \mathrm{V}_\forall$, there exists $u \in V$ such that $A \subseteq \uparrow_u$ and $\uparrow_A \setminus (\{x, z\} \cup x\text{-}z\text{-cut}) = \uparrow_u \setminus (\{x, z\} \cup x\text{-}z\text{-cut})$.

Proof. It is easy to see that if $A \neq \emptyset$, then we may choose u to be the variable in A that satisfies $u \preceq y$ for all $y \in A$.

If $A = \emptyset$, then we choose u as the last variable in the quantifier-prefix of Φ. Indeed, if u is existential, then we are done. Otherwise, u is universal. If $u \in \{x, z\}$, then this variable is removed from $\uparrow_A$ and we are done. If $u \notin \{x, z\}$, then $u \in x\text{-}z\text{-cut}$. This completes the proof of the observation. ◁

Proof sketch (Lemma 15). We assume that ϕ is expanded by all derived clauses from the run of Algorithm 1, and show that $\big((\bigwedge_{v \in \uparrow_A \setminus (\{x,z\} \cup x\text{-}z\text{-cut})} x = v) \Rightarrow x \geq z\big)$ is among these clauses.

We prove the lemma by induction on the length of the derivation of $\mathcal{P}(x, z; A)$. Observe that it is enough to show that, if $\mathcal{P}(x, z; A)$ is derived, where $z \notin A$, then $\phi \wedge (\bigwedge_{v \in A} x = v) \wedge (x < z)$ is not satisfiable. Then indeed, by Claim 16, we may choose $u \in V$ such that $A \subseteq \uparrow_u$ and $\uparrow_A \setminus (\{x, z\} \cup x\text{-}z\text{-cut}) = \uparrow_u \setminus (\{x, z\} \cup x\text{-}z\text{-cut})$. Since $z \notin A$ and $(x = x)$ is always satisfied, if $\phi \wedge (\bigwedge_{v \in A} x = v) \wedge (x < z)$ is not satisfiable, then neither is $\phi \wedge (\bigwedge_{v \in \uparrow_u \setminus \{x,z\}} x = v) \wedge (x < z)$, and therefore the algorithm expands ϕ by the desired clause. The rest of the proof consists of a straightforward verification of the base case for §I and the induction step for the remaining rules of $\mathcal{P}$. ◀

We conclude this subsection with two examples, the first one showcasing how the run of Algorithm 1 can be represented within the proof system, and the second one demonstrating that, in general, the proof system cannot be used to verify true instances in polynomial time.

▶ **Example 17.** Consider the instance Φ from Example 12. We show that the proof system derives $\bot$. First, we can derive $\mathcal{P}(x_i, x_i; \emptyset)$ for every $i \in [5]$ using §I. With §C (and suitable identifications of variables), we get $\mathcal{P}(x_3, x_1; \emptyset), \mathcal{P}(x_5, x_1; \emptyset)$, $\mathcal{P}(x_1, x_5; \{x_2\})$, $\mathcal{P}(x_3, x_4; \{x_2\})$, and $\mathcal{P}(x_5, x_3; \{x_4\})$. Next, a single application of §A yields $\mathcal{P}(x_1, x_3; \{x_2, x_4\})$. We can use §S to simplify the latter to $\mathcal{P}(x_1, x_3; \{x_2\})$. Using §A again, we get $\mathcal{P}(x_1, x_4; \{x_2\})$, and finally, §S simplifies the latter to $\mathcal{P}(x_1, x_4; \emptyset)$. Now an application of §R yields $\bot$.

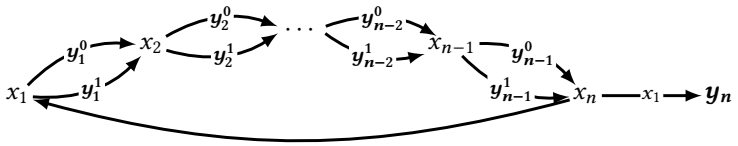


Figure 2 The quantifier-free part of Φ from Example 18.

▶ **Example 18.** Consider $\Phi \coloneqq \exists x_1 \forall y_1^0 \forall y_1^1 \exists x_2 \forall y_2^0 \forall y_2^1 \cdots \exists x_{n-1} \forall y_{n-1}^0 \forall y_{n-1}^1 \exists x_n \forall y_n\ \phi$ with ϕ described by Figure 2, where an edge from x to z labeled with y stands for $\mathrm{M}^+(x, y, z)$. Note that the proof system derives $\mathcal{P}(x_1, x_n; \{y_1^{i_1}, \ldots, y_{n-1}^{i_{n-1}}\})$ for all $i_1, \ldots, i_{n-1} \in \{0, 1\}$. Indeed, this is because it can follow the shortest derivation sequences, of which there are exponentially many. In contrast, Algorithm 1 derives the constraints $(x_{n-1} \geq x_1), \ldots, (x_2 \geq x_1), (x_1 \geq y_n)$ in this order, which leads to rejection. Interestingly enough, constraint paths as in Figure 2 were previously used in [22] to prove PSPACE-hardness of QCSP($\mathbb{Q}$; D).

3.2 True instances

In this subsection we prove Lemma 19, which states that the refutation condition §R from Table 1 is not only sufficient, but also necessary.

▶ **Lemma 19.** *If the proof system does not derive $\bot$ from Φ, then Φ is true.*

Proof. Suppose that the proof system cannot derive $\bot$ from Φ. Consider the following strategy for the EP. Let $x \in \mathrm{V}_\exists$ be such that $[\![x]\!]$ is not yet defined, but $[\![z]\!]$ is defined for every $z \prec x$. Then the EP selects any value for x such that, for every $z \prec x$:

- $[\![x]\!] \geq [\![z]\!]$ *if and only if* there exists $y \prec x$ with $[\![y]\!] \geq [\![z]\!]$ and $\mathcal{P}(x, y; \emptyset)$;
- $[\![x]\!] = [\![z]\!]$ *if and only if* there exist $y_1, y_2 \prec x$ and $y_2 \prec A \prec x$ such that

$$\mathcal{P}(x, y_1; \emptyset) \wedge \mathcal{P}(y_2, x; A) \quad \text{and} \quad [\![z]\!] = [\![y_1]\!] = [\![y_2]\!] = [\![A]\!].$$

We remark that some naïve simplifications of the above strategy fail already on small instances. For example, it is not enough for the EP to set $[\![x]\!] \geq [\![z]\!]$ if and only if $\mathcal{P}(x, z; \emptyset)$. To see this, consider $\Phi = \exists y \forall z \exists x\, \mathrm{M}^+(x, x, y)$. If the UP sets $[\![z]\!] = [\![y]\!]$, then the EP has to respect $(x \geq z)$ even though $\mathcal{P}(x, z; \emptyset)$ is not derived.

▷ Claim 20. The strategy of the EP is well-defined.

Proof. Suppose, on the contrary, that it is not. Let $x \in \mathrm{V}_\exists$ be the smallest variable w.r.t. $\prec$ for which the strategy of the EP is not well-defined. Then it must be the case that there exist $y, y_1, y_2 \prec x$ and $y_2 \prec A \prec x$ such that

$$\mathcal{P}(x, y; \emptyset) \wedge \mathcal{P}(x, y_1; \emptyset) \wedge \mathcal{P}(y_2, x; A) \quad \text{and} \quad [\![y]\!] > [\![y_1]\!] = [\![y_2]\!] = [\![A]\!]. \tag{4}$$

In particular, $y \notin A$. We choose the smallest possible y w.r.t. $\prec$ witnessing a condition of the form (4). By §T, we have $\mathcal{P}(y_2, y; A)$.

Case 1: $y \prec y_2$. Then, by §S, we have $\mathcal{P}(y_2, y; \emptyset)$.

Case 1.1: $y_2 \in \mathrm{V}_\forall$. Then $\bot$ can be derived using §R, a contradiction.

Case 1.2: $y_2 \in \mathrm{V}_\exists$. Then the EP did not follow his strategy because $[\![y]\!] > [\![y_2]\!]$ and we have $\mathcal{P}(y_2, y; \emptyset)$, a contradiction.

Case 2: $y_2 \prec y$.

Case 2.1: $y \in \mathrm{V}_\forall$. Then, by §S, we have $\mathcal{P}(y_2, y; \emptyset)$. But then $\bot$ can be derived using §R, a contradiction.

Case 2.2: $y \in \mathrm{V}_\exists$. Then, by §S, we have $\mathcal{P}(y_2, y; A \setminus y_2\text{-}y\text{-cut})$. Since $[\![y]\!] \geq [\![y_2]\!]$, by the strategy of the EP, there exists a variable $y' \prec y$ such that $[\![y']\!] \geq [\![y_2]\!]$ and $\mathcal{P}(y, y'; \emptyset)$. If $[\![y']\!] = [\![y_2]\!]$, then the EP did not follow his strategy, because he played $[\![y]\!] > [\![y']\!]$ while $y' \prec y$, $y_2 \prec A \setminus y_2\text{-}y\text{-cut} \prec y$, $\mathcal{P}(y, y'; \emptyset) \wedge \mathcal{P}(y_2, y; A \setminus y_2\text{-}y\text{-cut})$, and $[\![y']\!] = [\![y_2]\!] = [\![A \setminus y_2\text{-}y\text{-cut}]\!]$. a contradiction. So it must be the case that $[\![y']\!] > [\![y_2]\!]$. By §T, we have $\mathcal{P}(y_2, y'; A)$. But now y' can assume the role of y in (4), a contradiction to the minimality of y w.r.t. $\prec$. ◁

The next claim characterizes the equality of values for pairs of variables under $[\![\cdot]\!]$ in terms of properties of previously quantified variables, assuming that the EP has followed the strategy above. In particular, we show that if $[\![x]\!] = [\![z]\!]$ if and only if there exists a variable $y \preceq \{x, z\}$ so that $[\![x]\!] = [\![y]\!]$ and $[\![y]\!] = [\![z]\!]$ are enforced by the identifications of values of universal variables with y by the UP. Recall the comparison relations $\preceq$ and $\preceq_\forall$ from Definition 10.

▷ Claim 21. Suppose that the EP follows the strategy above. Then, for all $x, z \in \mathrm{V}$, we have $[\![x]\!] = [\![z]\!]$ if and only if there exist $x_1, x_2, z_1, z_2 \in V$ and $A_{x_2,x}, A_{z_2,z} \subseteq \mathrm{V}_\forall$ such that

1. $\{x_1, x_2\} \preceq x$ and $\{z_1, z_2\} \preceq z$
2. $x_2 \prec A_{x_2,x} \preceq x$ and $z_2 \prec A_{z_2,z} \preceq z$,
3. $y \preceq_\forall \{x_1, x_2, z_1, z_2\}$ for some $y \in \{x_2, z_2\}$,
4. $\mathcal{P}(x_2, x; A_{x_2,x}) \wedge \mathcal{P}(x, x_1; \emptyset) \wedge \mathcal{P}(z_2, z; A_{z_2,z}) \wedge \mathcal{P}(z, z_1; \emptyset)$,
5. $[\![A_{x_2,x}]\!] = [\![x_1]\!] = [\![x_2]\!] = [\![z_1]\!] = [\![z_2]\!] = [\![A_{z_2,z}]\!]$.

Whenever the right-hand side of the equivalence holds, we also have $[\![x]\!] = [\![x_2]\!]$ and $[\![z]\!] = [\![z_2]\!]$.

Proof sketch (Claim 21). "⇐" We show that $[\![x]\!] = [\![x_2]\!]$ and $[\![z]\!] = [\![z_2]\!]$. If $x \equiv x_2$, then clearly $[\![x]\!] = [\![x_2]\!]$. So, w.l.o.g., $x_2 \prec x$. If $x \in \mathrm{V}_\forall$, then §S yields $\mathcal{P}(x_2, x; \emptyset)$ and hence §R produces $\bot$, a contradiction. So we must have $x \in \mathrm{V}_\exists$. Then either $x_1 \equiv x$ or $x_1 \prec x$, and it follows from the strategy of the EP that $[\![x]\!] = [\![x_2]\!]$. Analogously we obtain that $[\![z]\!] = [\![z_2]\!]$. The rest follows by the transitivity of the equality.

"⇒" Whenever the right-hand side of the equivalence in Claim 21 is satisfied, we call $(x, x_1, x_2; A_{x_2,x})$ and $(z, z_1, z_2; A_{z_2,z})$ *witnessing quadruples* for $[\![x]\!] = [\![z]\!]$. If $x \equiv z$, then the statement trivially follows using §I, the witnessing quadruples are $(x, x, x; \emptyset)$ and $(z, z, z; \emptyset)$. So, w.l.o.g., $z \prec x$. If $x \in \mathrm{V}_\forall$, then the claim follows using §I, the witnessing quadruples are again $(x, x, x; \emptyset)$ and $(z, z, z; \emptyset)$. So suppose that $x \in \mathrm{V}_\exists$ and that the claim holds for all pairs of variables preceding x. Since $[\![x]\!] = [\![z]\!]$, by the strategy of the EP, there exist $x_1, x_2 \prec x$ and $x_2 \prec A \prec x$ such that $\mathcal{P}(x, x_1; \emptyset) \wedge \mathcal{P}(x_2, x; A)$ and $[\![z]\!] = [\![x_1]\!] = [\![x_2]\!] = [\![A]\!]$.

Since $[\![x_2]\!] = [\![z]\!]$ and $x_2, z \prec x$, we can apply the induction hypothesis for the pair x_2, z to obtain the witnessing quadruples $(x_2, x_{2_1}, x_{2_2}; A_{x_{2_2},x_2})$ and $(z, z_1, z_2; A_{z_2,z})$. By assumption, there exists $y \in \{z_2, x_{2_2}\}$ such that $y \preceq_\forall \{z_1, z_2, x_{2_1}, x_{2_2}\}$. Note that $[\![y]\!] = [\![x_1]\!]$. Thus, we can apply the induction hypothesis for the pair x_1, y to obtain the witnessing quadruples $(x_1, x_{1_1}, x_{1_2}; A_{x_{1_2},x_1})$ and $(y, y_1, y_2; A_{y_2,y})$. By assumption, there exists $y' \in \{y_2, x_{1_2}\}$ such that $y' \preceq_\forall \{y_1, y_2, x_{1_1}, x_{1_2}\}$. The two cases $y \equiv z_2$ and $y \equiv x_{2_2}$ are illustrated in Figure 3.

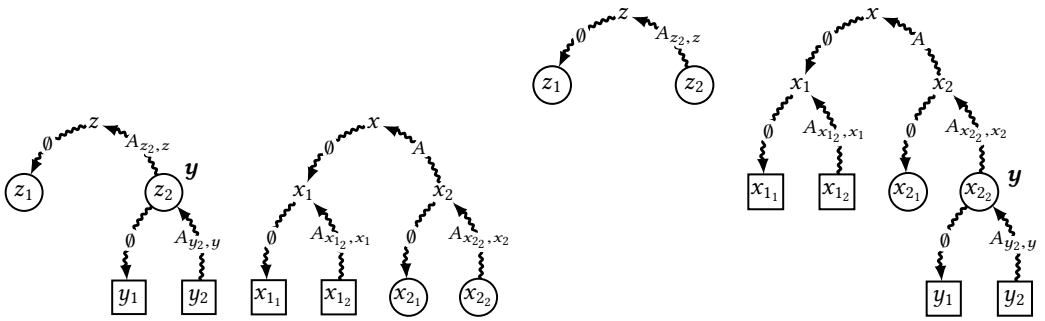


Figure 3 Cases $y \equiv z_2$ and $y \equiv x_{2_2}$ in the proof of Claim 21. The squiggly arrows represent inferences of $\mathcal{P}$.

Our goal is to find witnesses x'_1, x'_2, z'_1, z'_2 for the main statement of the claim, i.e., the witnessing quadruples will be of the form $(x, x'_1, x'_2; A_{x'_2,x})$ and $(z, z'_1, z'_2; A_{z'_2,z})$. The idea is that we want to choose x'_1, x'_2, z'_1, z'_2 from the variables introduced above, which all evaluate

to the value $[\![x]\!] = [\![z]\!]$ in $[\![\cdot]\!]$. To obtain the property in item 3, we want to choose variables that are small enough with respect to the order $\prec$, so that one of them can be compared to the others with respect to $\preceq_\forall$, assuming the properties of y and y' from above.

One suitable choice of witnesses is as follows. First, we choose $x'_1 \coloneqq x_{1_1}$. As visible in Figure 3, we can apply §T to $\mathcal{P}(x, x_1; \emptyset) \wedge \mathcal{P}(x_1, x_{1_1}; \emptyset)$ to derive $\mathcal{P}(x, x_{1_1}; \emptyset)$. Second, we choose $z'_1 \coloneqq z_1$ if $y \not\equiv z_1$ and $z'_1 \coloneqq y_1$ otherwise. Note that we have $\mathcal{P}(z, z_1; \emptyset)$ by assumption and, if $y \equiv z_1$, then we can use §T to derive $\mathcal{P}(z, y_1; \emptyset)$ from $\mathcal{P}(z, z_1; \emptyset) \wedge \mathcal{P}(y, y_1; \emptyset)$. Next, we choose $x'_2 \coloneqq x_{2_2}$ if $y \not\equiv x_{2_2}$, and $x'_2 \coloneqq y_2$ otherwise. A short argument shows that $x_{2_2} \preceq_\forall x_{2_1}$, which allows us to apply §A to $\mathcal{P}(x_{2_2}, x_2; A_{x_{2_2}, x_2}) \wedge \mathcal{P}(x_2, x_{2_1}; \emptyset) \wedge \mathcal{P}(x_2, x; A)$ to obtain an expression of the form $\mathcal{P}(x_{2_2}, x; A_{x_{2_2}, x})$. If $y \equiv x_{2_2}$, then it is necessary to apply §A a second time to obtain the expression of the form $\mathcal{P}(y_2, x; A_{y_2, x})$. Finally, the choice for z'_2 that we need will be $z'_2 \coloneqq z_2$ if $y \not\equiv z_2$ or $z'_2 \coloneqq y_2$ otherwise.

With the above witnessing quadruples, one can verify that items 1, 2, 4, and 5 will be satisfied. Thanks to choosing "small enough candidates" with respect to $\prec$ for each of x'_1, x'_2, z'_1, z'_2, item 3 can be verified as well. A full proof of Claim 21 with a verification of these properties can be found in Appendix A. ◁

▷ Claim 22. The strategy of the EP is a winning strategy.

Proof. Suppose, on the contrary, that this is not the case. Then there has to be a violated constraint of the form $(x = z \Rightarrow x \geq w)$, i.e., $[\![x]\!] = [\![z]\!] < [\![w]\!]$. Since $[\![x]\!] = [\![z]\!]$, by Claim 21, there exist $x_1, x_2, z_1, z_2 \in V$ and $A_{x_2,x}, A_{z_2,z} \subseteq \mathrm{V}_\forall$ such that

- $\{x_1, x_2\} \preceq x$ and $\{z_1, z_2\} \preceq z$
- $x_2 \prec A_{x_2,x} \preceq x$ and $z_2 \prec A_{z_2,z} \preceq z$,
- $y \preceq_\forall \{x_1, x_2, z_1, z_2\}$ for some $y \in \{x_2, z_2\}$,
- $\mathcal{P}(x_2, x; A_{x_2,x}) \wedge \mathcal{P}(x, x_1; \emptyset) \wedge \mathcal{P}(z_2, z; A_{z_2,z}) \wedge \mathcal{P}(z, z_1; \emptyset)$,
- $[\![A_{x_2,x}]\!] = [\![x_1]\!] = [\![x_2]\!] = [\![z_1]\!] = [\![z_2]\!] = [\![A_{z_2,z}]\!]$.

Moreover, we have $[\![x]\!] = [\![y]\!] = [\![z]\!]$. Let $A \coloneqq A_{x_2,x} \cup A_{z_2,z} \cup (\{x_1, x_2, z_1, z_2\} \setminus \{y\})$. By a single application of §C, we get $\mathcal{P}(y, w; A)$. Since $[\![z]\!] < [\![w]\!]$, clearly w is different from all variables which share the value with z. We choose the smallest possible w w.r.t. $\prec$ for which $\mathcal{P}(y, w; A)$ can be derived and such that $[\![z]\!] < [\![w]\!]$. Since $[\![w]\!] \neq [\![z]\!]$, we have $w \notin A$. Now we consider the following cases.

Case 1: $w \prec y$. By §S, we have $\mathcal{P}(y, w; \emptyset)$.

Case 1.1: $y \in \mathrm{V}_\forall$. In this case $\bot$ can be derived using §R, a contradiction.

Case 1.2: $y \in \mathrm{V}_\exists$. Then the EP was supposed to set $[\![y]\!] \geq [\![w]\!]$, however, we have $[\![y]\!] < [\![w]\!]$. Hence, the EP did not follow his strategy, a contradiction.

Case 2: $y \prec w$.

Case 2.1: $w \in \mathrm{V}_\forall$. By §S, we get $\mathcal{P}(y, w; \emptyset)$. But then $\bot$ can be derived using §R, a contradiction.

Case 2.2: $w \in \mathrm{V}_\exists$. Since $[\![w]\!] > [\![y]\!]$, by the strategy of the EP, there must exist $w' \prec w$ with $[\![w']\!] \geq [\![y]\!]$ such that $\mathcal{P}(w, w'; \emptyset)$. By §S, we have $\mathcal{P}(y, w; A \setminus y\text{-}w\text{-cut})$. If $[\![w']\!] = [\![y]\!]$, then the EP did not follow his strategy because he played $[\![w]\!] > [\![w']\!]$ while $y, w' \prec w$ and $y \prec A \setminus y\text{-}w\text{-cut} \prec w$, $[\![w']\!] = [\![y]\!] = [\![A \setminus y\text{-}w\text{-cut}]\!]$, and $\mathcal{P}(y, w; A \setminus y\text{-}w\text{-cut}) \wedge \mathcal{P}(w, w'; \emptyset)$, a contradiction. So it must be the case that $[\![w']\!] > [\![y]\!]$. By an application of §T to $\mathcal{P}(y, w; A) \wedge \mathcal{P}(w, w'; \emptyset)$, we have $\mathcal{P}(y, w'; A)$. But note that now w' can assume the role of w, a contradiction to the minimality of w w.r.t. $\prec$. ◁

This concludes the proof of Lemma 19. ◀

3.3 Putting everything together

Proof (Theorem 3). We show that Algorithm 1 solves $\mathrm{QCSP}(\mathbb{Q}; \mathrm{M}^+)$ in polynomial time. Observe that Algorithm 1 runs in polynomial time with respect to the length of Φ. Indeed, it expands Φ by at most V^3-many constraints, all of which have pp-definitions in $(\mathbb{Q}; \mathrm{M}^+, <)$ of linear length due to Lemma 9, and $\mathrm{CSP}(\mathbb{Q}; \mathrm{M}^+, <)$ is solvable in polynomial time [8]. Note that, if Φ contains a clause $(x \geq z)$ or $(z \geq x)$ such that $x \prec z$ and $z \in \mathrm{V}_\forall$, then Φ is false. Therefore, by Lemma 13, Φ is false whenever the algorithm rejects. Suppose that the algorithm accepts an instance Φ. By Lemma 15, $\bot$ cannot be derived from Φ using the proof system and hence, by Lemma 19, Φ is true. This completes the proof. ◀

4 The Complexity Dichotomy

This section is devoted to the proof of Theorem 5. We start by explaining how coNP-hardness is obtained in the cases that are not covered by Theorem 1, Corollary 4, or its analogue for *dual* $\pi\pi$. Recall Theorem 2 that can be used as a black box.

First, we use a syntactical argument to reduce the arity of the relations that need to considered to 4.

▶ **Lemma 23.** *Let $\mathfrak{B}$ be an OH structure that is not contained in the $\pi\pi$ fragment. Then $\mathfrak{B}$ pp-defines a relation of arity at most 4 that is not contained in the $\pi\pi$ fragment.*

Second, we perform an "educated brute-force" search through all relations of arity at most 4 that are not contained in the $\pi\pi$ fragment in order to classify them. Recall the relations D and $\check{\mathrm{Z}}$ defined in the introduction.

▶ **Lemma 24.** *Let $\mathfrak{B}$ be an OH structure that is not contained in the $\pi\pi$ fragment. Then $\mathfrak{B}$ pp-defines* D *or* $(\mathfrak{B}; \mathrm{M}^+)$ *qpp-defines* $\check{\mathrm{Z}}$.

Third, we show coNP-hardness of the QCSP for said relations combined with M^+.

▶ **Lemma 25.** *Let $\mathfrak{B}$ be an OH structure that is not contained in the $\pi\pi$ fragment and pp-defines M^+. Then $\mathrm{QCSP}(\mathfrak{B})$ is coNP-hard.*

The proof of Lemma 25 below relies almost entirely on constraint paths built using M^+. In Figure 4, edges relate to constraints over M^+, e.g. the two leftmost arrows in the lower chain represent $\mathrm{M}^+(f, y_1^i, z) \wedge \mathrm{M}^+(z, z, f)$ for $i \in \{1, 2\}$ where z corresponds to an unlabelled vertex. These constraint paths are used to generate exponentially many incomparable expressions within the proof system $\mathcal{P}$, but M^+ itself has no mechanism for turning them into a working gadget. This is why such constraint paths can be handled by Algorithm 1. The situation changes already when we add a single constraint associated to the relation $\check{\mathrm{Z}}$.

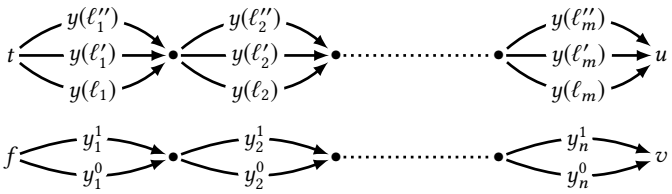


Figure 4 A gadget for the proof of Lemma 25.

Proof (Lemma 25). In the case where $\mathfrak{B}$ pp-defines D, we have that QCSP($\mathfrak{B}$) is PSPACE-hard by Corollary 6 in [22] and Proposition 6. So suppose that $\mathfrak{B}$ does not pp-define D. By Lemma 24, we have that $\mathfrak{B}$ qpp-defines $\check{\mathrm{Z}}$.

We reduce from the complement of the satisfiability problem for propositional 3-CNF. Consider an arbitrary propositional 3-CNF formula ψ, i.e., a conjunction of clauses of the form $\ell_i \vee \ell'_i \vee \ell''_i$ for $i \in [m]$, where $\ell_i, \ell'_i, \ell''_i$ are potentially negated propositional variables from $\{x_1, \dots, x_n\}$. We set $\Phi \coloneqq \exists t \exists f \forall y_1^0 \forall y_1^1 \dots \forall y_n^0 \forall y_n^1 \exists \dots \exists u \exists v\ \phi \wedge \check{\mathrm{Z}}(v, f, u, t)$, where $\exists \dots$ are additional unlabelled existentially quantified variables, and ϕ is defined as in Figure 4. Here $y(x_i) \coloneqq y_i^1$, $y(\neg x_i) \coloneqq y_i^0$, a directed edge from x to z labeled with y stands for $\mathrm{M}^+(x, y, z) \wedge \mathrm{M}^+(z, z, x)$,[2] and unlabelled dots correspond to unlabelled existential variables.

"$\Rightarrow$" Suppose that ψ is not satisfiable. We show that the EP has a winning strategy on Φ. First, the EP chooses $[\![f]\!] < [\![t]\!]$. If there exists $i \in [n]$, such that the UP chose $[\![y_i^0]\!] \neq [\![f]\!]$ and $[\![y_i^1]\!] \neq [\![f]\!]$, then the EP chooses the values for the remaining existential variables as follows: equal $[\![f]\!]$ if they appear in the lower chain in Figure 4 before y_i^0 and y_i^1, and equal $[\![t]\!]$ otherwise. Since $[\![f]\!] < [\![t]\!]$, this choice satisfies $\phi \wedge (v \neq f)$. We may therefore assume that $[\![f]\!] \in \{[\![y_i^0]\!], [\![y_i^1]\!]\}$ for every $i \in [n]$.

We claim that there exists $j \in [m]$ such that $[\![t]\!] \notin \{[\![y(\ell_j)]\!], [\![y(\ell'_j)]\!], [\![y(\ell''_j)]\!]\}$. Suppose, on the contrary, that this is not the case. Let $[\![\cdot]\!]'$ be any map from $\{x_1, \dots, x_n\}$ to $\{0, 1\}$ such that, for every $i \in [n]$, $[\![x_i]\!]' = 0$ if $[\![y_i^0]\!] = t$ and $[\![x_i]\!]' = 1$ if $[\![y_i^1]\!] = t$. Recall that $[\![f]\!] \in \{[\![y_i^0]\!], [\![y_i^1]\!]\}$ for every $i \in [n]$ and thus $[\![\cdot]\!]'$ is well-defined. Observe that $[\![\cdot]\!]'$ is a satisfying assignment to ψ, contradicting our assumption. Hence the claim holds.

The EP can choose the values for the remaining existential variables as follows: equal $[\![t]\!]$ if they appear in the upper chain in Figure 4 before the j-th column, equal to an arbitrary number $q > [\![t]\!]$ if they appear in the upper chain after the j-th column, and equal $[\![f]\!]$ otherwise. Such assignment satisfies $\phi \wedge (t \neq u)$.

"$\Leftarrow$" Suppose that there exists a satisfying assignment $[\![\cdot]\!]'$ to ψ. We show that then the UP has a winning strategy on Φ. If the EP chooses $[\![f]\!] \geq [\![t]\!]$, the UP wins on Φ, suppose therefore that $[\![f]\!] < [\![t]\!]$. If $[\![x_i]\!]' = 0$, the UP plays $[\![y_i^0]\!] = t$ and $[\![y_i^1]\!] = f$, and if $[\![x_i]\!]' = 1$, the UP plays $[\![y_i^0]\!] = f$ and $[\![y_i^1]\!] = t$. It follows from the lower chain in Figure 4 that the EP loses unless $[\![v]\!] = [\![f]\!]$. Moreover, since $[\![\cdot]\!]'$ is a satisfying assignment to ψ, it follows from the upper chain that the EP loses unless $[\![u]\!] = [\![t]\!]$. But then $[\![\cdot]\!]$ violates $(v \neq f \vee u \neq t)$ and the UP wins again. ◀

We are now ready to prove Theorem 5. As an intermediate step, we prove Corollary 4, which extends the tractability result for M^+ to the whole $\pi\pi$ fragment.

Proof (Corollary 4). By Lemma 9, every relation definable by a clause of the form (3) is pp-definable in $(\mathbb{Q}; \mathrm{M}^+)$. For clauses of the form (3) where the last disjunct $(x \geq z)$ is not present, we may use the pp-definition from Lemma 9 and universally quantify over z, which yields a qpp-definition. Hence, if $\mathfrak{B}$ is as in Corollary 4, then every relation of $\mathfrak{B}$ is qpp-definable in $(\mathbb{Q}; \mathrm{M}^+)$. In this case, QCSP($\mathfrak{B}$) reduces in LOGSPACE to QCSP($\mathbb{Q}; \mathrm{M}^+$) due to Proposition 6 and, by Theorem 3, is in PTIME.

For the second part, note that the forward direction immediately follows from the definition of the OH and $\pi\pi$ fragment, since the syntactic form in (3) is a special case of both (2) and (1). For the backward direction, suppose that $\mathfrak{B}$ is an OH structure contained in the $\pi\pi$ fragment. Then every relation has a CNF-definition ϕ over $\{\neq, \geq\}$ where each conjunct is of the form (2) for $k, \ell \geq 0$. By Lemma 8, we can choose ϕ so that every conjunct is of the form (3), possibly with the last disjunct omitted. ◀

[2] Note the difference from the previous interpretation of labeled directed edges, e.g., in Examples 14 and 18. The current interpretation entails $D(x, y, z)$.

Proof (Theorem 5). If $\mathfrak{B}$ is GOH, then QCSP($\mathfrak{B}$) is in PTIME by Theorem 3. If $\mathfrak{B}$ is contained in the $\pi\pi$ fragment, then QCSP($\mathfrak{B}$) is in PTIME by Corollary 4. If $\mathfrak{B}$ is contained in the *dual* $\pi\pi$ fragment, then we reach the same conclusion as in the previous case using the dual version of Corollary 4, which can be obtained by reversing the order in each individual statement used in its proof. Finally, suppose that $\mathfrak{B}$ is not GOH, and also not contained in the $\pi\pi$ or *dual* $\pi\pi$ fragment. Then it either follows directly from Theorem 2 that QCSP($\mathfrak{B}$) is coNP-hard, in which case we are done, or $\mathfrak{B}$ pp-defines M^+ or M^-. If $\mathfrak{B}$ pp-defines M^+, then QCSP($\mathfrak{B}$) is coNP-hard by Lemma 25, because $\mathfrak{B}$ is not contained in the $\pi\pi$ fragment. Otherwise $\mathfrak{B}$ pp-defines M^-. Then we reach the same conclusion as in the previous case using the dual version of Lemma 25, which can again be obtained by reversing the order. ◀

5 The Meta-Problem

For a classification as the one in Theorem 5, one is often interested in the complexity of the following meta-problem: "Does a given structure satisfy the condition for tractability provided by the classification?" In the present section, we give a nondeterministic single-exponential upper bound on the complexity of this meta-problem for Theorem 5.

▶ **Theorem 26.** *The question whether* QCSP($\mathfrak{B}$) *is tractable for a given OH structure* $\mathfrak{B}$ *is decidable in* NEXPTIME.

The most natural way to finitely represent a temporal relation R of arity k is by the set of all weak linear orderings on k variables that are witnessed by the tuples in R. Since the first-order theory of $(\mathbb{Q};<)$ has quantifier-elimination [13], every temporal relation has such a representation. To see this, note that the atomic formula $R(x_1,\dots,x_k)$ has a quantifier-free definition ψ over $\{<\}$, then it is enough to list all weak linear orderings that are compatible with ψ. For instance, the temporal relation defined by $(x_1 \neq x_2 \vee x_1 < x_3) \wedge (x_1 \leq x_2) \wedge (x_2 \leq x_3)$ admits the following three weak linear orderings:

$$(x_1 = x_2 < x_3), \quad (x_1 < x_2 < x_3), \quad \text{and} \quad (x_1 < x_2 = x_3).$$

Proof (Theorem 26). For every relation R of $\mathfrak{B}$, let ϕ_R be the first-order formula over $\{<\}$ defining R that is provided as an input to the meta-problem. Let ψ_R be the disjunction of all k-ary formulas specifying a weak linear order compatible with R. We claim that ψ_R can be computed from ϕ_R in exponential time. Indeed, to test whether a k-ary formula θ specifying a weak linear ordering is compatible with R, we must only test whether $(\mathbb{Q};<) \models \forall\bar{x}\big(\theta(\bar{x}) \Rightarrow \phi_R(\bar{x})\big)$. This can be done in PSPACE (Theorem 21.2 in [14]). Clearly, ψ_R defines R. Now, to test whether R is contained in the (*dual*) $\pi\pi$ fragment, we simply guess a conjunction ϕ of clauses of the form (2) that defines R. Note that ϕ might be of size exponential in the size of the input. To verify whether ϕ defines R, we test whether $(\mathbb{Q};<) \models \forall\bar{x}\big(\phi(\bar{x}) \Leftrightarrow \psi_R(\bar{x})\big)$. This can be done in NEXPTIME because the satisfiability of quantifier-free formulas over $(\mathbb{Q};<)$, such as $\neg(\phi(\bar{x}) \Leftrightarrow \psi_R(\bar{x}))$, can be decided in NP. Whether R is contained in the GOH fragment can be tested similarly. ◀

6 Open Questions

For quantified OH constraints, we leave the following question open:

Question 1: Do OH QCSPs exhibit a dichotomy between coNP and PSPACE-hardness?

We also ask the following questions regarding open cases outside of OH:

Question 2: Is QCSP($\mathfrak{B}$) in P whenever $\mathfrak{B}$ is a temporal structure contained in the *mi* fragment [7]? It is enough to consider QCSP($\mathbb{Q}; x \neq y \vee x \geq z \vee x > w$) [2].

Question 3: Is QCSP($\mathfrak{B}$) in NP whenever $\mathfrak{B}$ is a temporal structure contained in the $\pi\pi$ fragment? It is enough to consider QCSP($\mathbb{Q}; x \neq y \vee x \geq z_1 \vee x \geq z_2$) [2].

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley, Boston, 1995. URL: http://webdam.inria.fr/Alice/.

2 Manuel Bodirsky. *Complexity of Infinite-Domain Constraint Satisfaction*. Cambridge University Press, Cambridge, 2021. doi:10.1017/9781107337534.

3 Manuel Bodirsky and Hubie Chen. Quantified equality constraints. *SIAM J. Comput.*, 39(8):3682–3699, 2010. A preliminary version of the paper appeared in the proceedings of LICS'07. doi:10.1137/080725209.

4 Manuel Bodirsky, Hubie Chen, and Michał Wrona. Tractability of quantified temporal constraints to the max. *Int. J. Algebra Comput.*, 24(8):1141–1156, 2014. doi:10.1142/S0218196714500507.

5 Manuel Bodirsky and Víctor Dalmau. Datalog and constraint satisfaction with infinite templates. *J. Comput. Syst. Sci.*, 79(1):79–100, 2013. A preliminary version appeared in the proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS'05). doi:10.1016/j.jcss.2012.05.012.

6 Manuel Bodirsky and Jan Kára. The complexity of equality constraint languages. *Theory Comput. Syst.*, 43(2):136–158, 2008. doi:10.1007/s00224-007-9083-9.

7 Manuel Bodirsky and Jan Kára. The complexity of temporal constraint satisfaction problems. *J. ACM*, 57(2):9:1–9:41, 2010. doi:10.1145/1667053.1667058.

8 Manuel Bodirsky and Jakub Rydval. On the descriptive complexity of temporal constraint satisfaction problems. *J. ACM*, 70(1):2:1–2:58, 2023. doi:10.1145/3566051.

9 Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic programming and databases*. Surveys in computer science. Springer, Berlin, Heidelberg, 1990. URL: https://www.worldcat.org/oclc/20595273.

10 Witold Charatonik and Michał Wrona. Quantified positive temporal constraints. In Michael Kaminski and Simone Martini, editors, *Computer Science Logic, 22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings*, volume 5213 of *Lecture Notes in Computer Science*, pages 94–108, Berlin, Heidelberg, 2008. Springer. doi:10.1007/978-3-540-87531-4_9.

11 Witold Charatonik and Michał Wrona. Tractable quantified constraint satisfaction problems over positive temporal templates. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008, Doha, Qatar, November 22-27, 2008. Proceedings*, volume 5330 of *Lecture Notes in Computer Science*, pages 543–557, Berlin, Heidelberg, 2008. Springer, Springer. doi:10.1007/978-3-540-89439-1_38.

12 Hubie Chen and Michał Wrona. Guarded ord-horn: A tractable fragment of quantified constraint satisfaction. In Ben C. Moszkowski, Mark Reynolds, and Paolo Terenziani, editors, *19th International Symposium on Temporal Representation and Reasoning, TIME 2012, Leicester, United Kingdom, September 12-14, 2012*, pages 99–106, Leicester, UK, 2012. IEEE Computer Society. doi:10.1109/TIME.2012.19.

13 Wilfrid Hodges. *A shorter model theory*. Cambridge University Press, Cambridge, 1997.

14 Dexter Kozen. *Theory of computation*, volume 170 of *Texts in Computer Science*. Springer, 2006. doi:10.1007/1-84628-477-5.

15 Andrei A. Krokhin, Peter Jeavons, and Peter Jonsson. Reasoning about temporal relations: The tractable subalgebras of allen's interval algebra. *J. ACM*, 50(5):591–640, 2003. doi:10.1145/876638.876639.

16 Bernhard Nebel and Hans-Jürgen Bürckert. Reasoning about temporal relations: A maximal tractable subclass of allen's interval algebra. *J. ACM*, 42(1):43–66, 1995. doi:10.1145/200836.200848.

17 Jakub Rydval, Žaneta Semanišinová, and Michał Wrona. Identifying tractable quantified temporal constraints within ord-horn. *CoRR*, abs/2402.09187, 2024. doi:10.48550/arXiv.2402.09187.

18 Michał Wrona. Tractability frontier for dually-closed ord-horn quantified constraint satisfaction problems. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 535–546, Berlin, Heidelberg, 2014. Springer. doi:10.1007/978-3-662-44522-8_45.

19 Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *J. ACM*, 67(5):30:1–30:78, 2020. doi:10.1145/3402029.

20 Dmitriy Zhuk. Personal communication, 2022.

21 Dmitriy Zhuk and Barnaby Martin. QCSP monsters and the demise of the chen conjecture. *J. ACM*, 69(5):35:1–35:44, 2022. doi:10.1145/3563820.

22 Dmitriy Zhuk, Barnaby Martin, and Michał Wrona. The complete classification for quantified equality constraints. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2746–2760, Florence, Italy, 2023. SIAM. doi:10.1137/1.9781611977554.ch103.

A A Full Proof of Claim 21

▷ Claim 21. Suppose that the EP follows the strategy above. Then, for all $x, z \in \mathrm{V}$, we have $[\![x]\!] = [\![z]\!]$ if and only if there exist $x_1, x_2, z_1, z_2 \in V$ and $A_{x_2,x}, A_{z_2,z} \subseteq \mathrm{V}_\forall$ such that

1. $\{x_1, x_2\} \preceq x$ and $\{z_1, z_2\} \preceq z$
2. $x_2 \prec A_{x_2,x} \preceq x$ and $z_2 \prec A_{z_2,z} \preceq z$,
3. $y \preceq_\forall \{x_1, x_2, z_1, z_2\}$ for some $y \in \{x_2, z_2\}$,
4. $\mathcal{P}(x_2, x; A_{x_2,x}) \wedge \mathcal{P}(x, x_1; \emptyset) \wedge \mathcal{P}(z_2, z; A_{z_2,z}) \wedge \mathcal{P}(z, z_1; \emptyset)$,
5. $[\![A_{x_2,x}]\!] = [\![x_1]\!] = [\![x_2]\!] = [\![z_1]\!] = [\![z_2]\!] = [\![A_{z_2,z}]\!]$.

Whenever the right-hand side of the equivalence holds, we also have $[\![x]\!] = [\![x_2]\!]$ and $[\![z]\!] = [\![z_2]\!]$.

Proof. "⇐" We show that $[\![x]\!] = [\![x_2]\!]$ and $[\![z]\!] = [\![z_2]\!]$. If $x \equiv x_2$, then clearly $[\![x]\!] = [\![x_2]\!]$. So, w.l.o.g., $x_2 \prec x$. If $x \in \mathrm{V}_\forall$, then §S yields $\mathcal{P}(x_2, x; \emptyset)$ and hence §R produces $\bot$, a contradiction. So we must have $x \in \mathrm{V}_\exists$. Then either $x_1 \equiv x$ or $x_1 \prec x$, and it follows from the strategy of the EP that $[\![x]\!] = [\![x_2]\!]$. Analogously we obtain that $[\![z]\!] = [\![z_2]\!]$. The rest follows by the transitivity of the equality.

"⇒" Whenever the right-hand side of the equivalence in Claim 21 is satisfied, we call $(x, x_1, x_2; A_{x_2,x})$ and $(z, z_1, z_2; A_{z_2,z})$ *witnessing quadruples* for $[\![x]\!] = [\![z]\!]$. If $x \equiv z$, then the statement trivially follows using §I, the witnessing quadruples are $(x, x, x; \emptyset)$ and $(z, z, z; \emptyset)$. So, w.l.o.g., $z \prec x$. If $x \in \mathrm{V}_\forall$, then the claim follows using §I, the witnessing quadruples are again $(x, x, x; \emptyset)$ and $(z, z, z; \emptyset)$. So suppose that $x \in \mathrm{V}_\exists$ and that the claim holds for all pairs of variables preceding x. Since $[\![x]\!] = [\![z]\!]$, by the strategy of the EP, there exist $x_1, x_2 \prec x$ and $x_2 \prec A \prec x$ such that $\mathcal{P}(x, x_1; \emptyset) \wedge \mathcal{P}(x_2, x; A)$ and $[\![z]\!] = [\![x_1]\!] = [\![x_2]\!] = [\![A]\!]$.

Since $[\![x_2]\!] = [\![z]\!]$ and $x_2, z \prec x$, we can apply the induction hypothesis for the pair x_2, z to obtain the witnessing quadruples $(x_2, x_{2_1}, x_{2_2}; A_{x_{2_2},x_2})$ and $(z, z_1, z_2; A_{z_2,z})$. By assumption, there exists $y \in \{z_2, x_{2_2}\}$ such that $y \preceq_\forall \{z_1, z_2, x_{2_1}, x_{2_2}\}$. Note that $[\![y]\!] = [\![x_1]\!]$. Thus, we can apply the induction hypothesis for the pair x_1, y to obtain the witnessing quadruples

$(x_1, x_{1_1}, x_{1_2}; A_{x_{1_2},x_1})$ and $(y, y_1, y_2; A_{y_2,y})$. By assumption, there exists $y' \in \{y_2, x_{1_2}\}$ such that $y' \preceq_\forall \{y_1, y_2, x_{1_1}, x_{1_2}\}$. The two cases w.r.t. the variable y that can occur are illustrated in Figure 3, see Case 1 and Case 2 below.

Our goal is to find witnesses x'_1, x'_2, z'_1, z'_2 for the main statement of the claim, i.e., the witnessing quadruples will be of the form $(x, x'_1, x'_2; A_{x'_2,x})$ and $(z, z'_1, z'_2; A_{z'_2,z})$. For the sake of brevity, we will not explicitly write down the precise definitions of $A_{x'_2,x}$ and $A_{z'_2,z}$ as they will be clear from the context. We set $x'_1 \coloneqq x_{1_1}$, and:

$$z'_1 \coloneqq \begin{cases} y_1 & \text{if } y \equiv z_1, \\ z_1 & \text{otherwise,} \end{cases} \qquad z'_2 \coloneqq \begin{cases} y_2 & \text{if } y \equiv z_2, \\ z_2 & \text{otherwise,} \end{cases} \qquad x'_2 \coloneqq \begin{cases} y_2 & \text{if } y \equiv x_{2_2}, \\ x_{2_2} & \text{otherwise.} \end{cases}$$

By the induction hypothesis and the transitivity of $\prec$, x'_1, x'_2, z'_1, z'_2 clearly satisfy item 1. It will also be clear that our implicit choice of $A_{x'_2,x}$ and $A_{z'_2,z}$ leads to satisfaction of item 5. The remaining three items are proved in the case distinction below. In both Cases 1 and 2, we initially start proving items 2 and 4, and then proceed with item 3 in the finer subdivision into Cases 1.1, 1.2, 2.1, and 2.2. For the sake of conciseness, when applying rules of $\mathcal{P}$ to derive new expressions, we often do not state all necessary expressions for the inference, as long as they are clear from the rule and the resulting expression.

To justify the applications of the rule §A that follow, we observe that $y_2 \preceq_\forall y_1$. Indeed, suppose that $y_1 \not\equiv y_2$. By §T, we have $\mathcal{P}(y_2, y_1, A_{y_2,y})$. If $y_1 \in \mathrm{V}_\exists$ or $y_1 \prec y_2$, then $y' \preceq_\forall \{y_1, y_2\}$ implies $y_1 \prec y_2$ and $y_2 \in \mathrm{V}_\forall$. By §S, we obtain $\mathcal{P}(y_2, y_1, \emptyset)$, and then using §R we get $\bot$, a contradiction. By an analogous argument, we have $x_{2_2} \preceq_\forall x_{2_1}$. Now it immediately follows that, by §A, we have

$$\mathcal{P}(x_{2_2}, x; A \cup A_{x_{2_2},x_2} \cup (\{x_{2_1}\} \setminus \{x_{2_2}\}). \tag{5}$$

A subsequent double application of §T yields

$$\mathcal{P}(x_{2_2}, x_{1_1}; A \cup A_{x_{2_2},x_2} \cup (\{x_{2_1}\} \setminus \{x_{2_2}\})). \tag{6}$$

Case 1: $y \equiv z_2$. Then $z_2 \preceq_\forall \{z_1, x_{2_1}, x_{2_2}\}$. We now establish item 4 with a suitable choice of sets $A_{x'_2,x}$ and $A_{z'_2,z}$. It is easy to verify that item 2 is satisfied as well.

- If $y \equiv z_1$, then, by §T, we have $\mathcal{P}(z, z'_1; \emptyset)$ because $z'_1 \equiv y_1$. Otherwise $z_1 \in \mathrm{V}_\forall$ and $y \prec z_1$; we have $\mathcal{P}(z, z'_1; \emptyset)$, because $z'_1 \equiv z_1$.
- Recall that $y_2 \preceq_\forall y_1$. Thus, by §A, we have

  $$\mathcal{P}(z'_2, z; A_{z_2,z} \cup A_{y_2,y} \cup (\{y_1\} \setminus \{y_2\})), \tag{7}$$

  because $z'_2 \equiv y_2$.
- By §T, we have $\mathcal{P}(x, x'_1; \emptyset)$, because $x'_1 = x_{1_1}$.
- By assumption, we have $y \preceq_\forall x_{2_2}$. Recall that we have (5). If $y \equiv x_{2_2}$, then an application of §A yields

  $$\mathcal{P}(x'_2, x; A \cup A_{x_{2_2},x_2} \cup (\{x_{2_1}\} \setminus \{x_{2_2}\}) \cup A_{y_2,y} \cup (\{y_1\} \setminus \{y_2\})), \tag{8}$$

  because $x'_2 \equiv y_2$. Otherwise $y \prec x_{2_2}$ and $x_{2_2} \in \mathrm{V}_\forall$. Then $\mathcal{P}(x'_2, x; A \cup A_{x_{2_2},x_2} \cup (\{x_{2_1}\} \setminus \{x_{2_2}\}))$ follows directly from (5) because $x'_2 \equiv x_{2_2}$.

In the case distinction below, we verify item 3 for x'_1, x'_2, z'_1, z'_2.

Case 1.1: $y' \equiv y_2$. By the choice of y', we have $y_2 \equiv y' \preceq \{y_1, y_2, x_{1_1}\}$. Recall that $y_2 \preceq y \preceq_\forall \{z_1, z_2, x_{2_2}\}$. Hence, if $y \prec z_1$, then $y_2 \preceq_\forall z_1$. Similar applies to z_2 and x_{2_2}. It follows that, for all choices of x'_1, x'_2, z'_1, z'_2 above, we get $z'_2 \equiv y_2 \preceq_\forall \{z'_1, x'_1, x'_2\}$.

Case 1.2: $y' \equiv x_{1_2}$. We show that, as above, $z'_2 \equiv y_2 \preceq_\forall \{z'_1, x'_1, x'_2\}$, starting with x'_2. If $y \equiv z_2 \prec x_{2_2}$ and $x_{2_2} \in \mathrm{V}_\forall$, then we have $z'_2 \equiv y_2 \preceq_\forall x_{2_2} \equiv x'_2$. Otherwise $x_{2_2} \equiv z_2 \equiv y$ and by our choice of x'_2 and z'_2, we have $z'_2 \equiv y_2$ and $x'_2 \equiv y_2$. In particular, $z'_2 \preceq_\forall x'_2$. Next comes z'_1. If $y \prec z_1$ and $z_1 \in \mathrm{V}_\forall$, then $y_2 \preceq_\forall z_1$ because $y_2 \preceq y$. Consequently, $z'_2 \equiv y_2 \preceq_\forall z_1 \equiv z'_1$. Otherwise $y \equiv z_1$ and $z'_1 \equiv y_1$. Recall that we always have $y_2 \preceq_\forall y_1$ and hence $z'_2 \equiv y_2 \preceq_\forall y_1 \equiv z'_1$.

Finally x'_1. Recall that we have $y \equiv z_2 \preceq_\forall x_{2_2} \preceq_\forall x_{2_1}$ (the second $\preceq_\forall$ was derived above equation (5)). We consider the following two cases. First, suppose that $z_2 \prec x_{2_2}$ and $x_{2_2} \in \mathrm{V}_\forall$. Since $x_{2_2} \in \mathrm{V}_\forall$, it cannot be the case that $x_{1_1} \prec x_{2_2}$, otherwise §S applied on (6) yields $\mathcal{P}(x_{2_2}, x_{1_1}; \emptyset)$. Using §R, we obtain $\bot$, a contradiction. Hence $x_{2_2} \preceq x_{1_1}$. Now it follows that $y_2 \preceq z_2 \prec x_{2_2} \preceq x_{1_1}$. Since $y' \preceq_\forall \{y_2, x_{1_1}\}$, we even have $z'_2 \equiv y_2 \preceq_\forall x_{1_1} \equiv x'_1$. Second, suppose that $x_{2_2} \equiv z_2 \equiv y$. Recall that $y_2 \preceq_\forall y_1$ (derived above (5)) and $\mathcal{P}(y_2, y; A_{y_2,y})$. Combining this with (6) and applying §A, we get

$$\mathcal{P}(y_2, x_{1_1}; A \cup A_{x_{2_2},x_2} \cup (\{x_{2_1}\} \setminus \{x_{2_2}\}) \cup A_{y_2,y} \cup (\{y_1\} \setminus \{y_2\})). \tag{9}$$

We cannot have $x_{1_1} \prec y_2$, otherwise $y_2 \in \mathrm{V}_\forall$ in which case §S yields $\mathcal{P}(y_2, x_{1_1}; \emptyset)$ and then §R yields $\bot$, a contradiction. Hence, $y_2 \preceq x_{1_1}$. Since $y' \preceq_\forall \{y_2, x_{1_1}\}$, we get $z'_2 \equiv y_2 \preceq_\forall x_{1_1} \equiv x'_1$.

Case 2: $y \equiv x_{2_2}$. Then $x_{2_2} \preceq_\forall \{z_1, z_2, x_{2_1}\}$. We now show that item 4 holds true with a suitable choice of sets $A_{x'_2,x}$ and $A_{z'_2,z}$; it will be clear that item 2 is satisfied as well.

- If $y \equiv z_1$, then, by §T, we get $\mathcal{P}(z, z'_1; \emptyset)$, because $z'_1 \equiv y_1$. Otherwise $z_1 \in \mathrm{V}_\forall$ and $y \prec z_1$; we have $\mathcal{P}(z, z'_1; \emptyset)$ because $z'_1 \equiv z_1$.
- First, suppose that $y \equiv z_2$. Recall that we have $y_2 \preceq_\forall y_1$. By §A, we have (7) because $z'_2 \equiv y_2$. Second, suppose that $y \prec z_2$ and $z_2 \in \mathrm{V}_\forall$. Then we have $\mathcal{P}(z'_2, z; A_{z_2,z})$ because $z'_2 \equiv z_2$.
- By §T, we have $\mathcal{P}(x, x'_1; \emptyset)$ because $x'_1 \equiv x_{1_1}$.
- Recall that we have (5) and $y_2 \preceq_\forall y_1$. By §A, we have (8) because $x'_2 \equiv y_2$.

Finally, we verify item 3 of the claim.

Case 2.1: $y' \equiv y_2$. By the choice of y', we have $y_2 \equiv y' \preceq \{y_1, y_2, x_{1_1}\}$. Recall that $y_2 \preceq y \preceq_\forall \{z_1, z_2, x_{2_2}\}$. Hence, if $y \prec z_1$, then $y_2 \preceq_\forall z_1$. Similar applies to z_2 and x_{2_2}. It follows that, for all choices of x'_1, x'_2, z'_1, z'_2 above, we get $x'_2 \equiv y_2 \preceq_\forall \{z'_1, z'_2, x'_1\}$.

Case 2.2: $y' \equiv x_{1_2}$. By a double application of §T on (8), we get (9). It cannot be that $x_{1_1} \prec y_2$, as this implies $y_2 \in \mathrm{V}_\forall$, in which case §S yields $\mathcal{P}(y_2, x_{1_1}; \emptyset)$ and then §R yields $\bot$, a contradiction as in Case 1.2. Hence $x'_2 \equiv y_2 \preceq_\forall x_{1_1} \equiv x'_1$. Recall that $y_2 \preceq_\forall y_1$. Either $x_{2_2} \prec z_1$ and $z_1 \in \mathrm{V}_\forall$, in which case $x'_2 \equiv y_2 \preceq_\forall z_1 \equiv z'_1$, or $x_{2_2} \equiv z_1$ in which case $x'_2 \equiv y_2 \preceq_\forall y_1 \equiv z'_1$. Also either $x_{2_2} \prec z_2$ and $z_2 \in \mathrm{V}_\forall$, in which case $x'_2 \equiv y_2 \preceq_\forall z_2 \equiv z'_2$, or $x_{2_2} \equiv z_2$ in which case $x'_2 \equiv y_2 \preceq_\forall y_2 \equiv z'_2$. Hence, $x'_2 \preceq_\forall \{z'_1, z'_2, x'_1\}$. ◁

On Homomorphism Indistinguishability and Hypertree Depth

Benjamin Scheidt
Humboldt-Universität zu Berlin, Germany

Abstract

GC^k is a logic introduced by Scheidt and Schweikardt (2023) to express properties of hypergraphs. It is similar to first-order logic with counting quantifiers (C) adapted to the hypergraph setting. It has distinct sets of variables for vertices and for hyperedges and requires vertex variables to be guarded by hyperedge variables on every quantification.

We prove that two hypergraphs G, H satisfy the same sentences in the logic GC^k with *guard depth* at most k if, and only if, they are homomorphism indistinguishable over the class of hypergraphs of strict hypertree depth at most k. This lifts the analogous result for tree depth $\leq k$ and sentences of first-order logic with counting quantifiers of quantifier rank at most k due to Grohe (2020) from graphs to hypergraphs. The guard depth of a formula is the quantifier rank with respect to hyperedge variables, and strict hypertree depth is a restriction of hypertree depth as defined by Adler, Gavenčiak and Klimošová (2012). To justify this restriction, we show that for every H, the strict hypertree depth of H is at most 1 larger than its hypertree depth, and we give additional evidence that strict hypertree depth can be viewed as a reasonable generalisation of tree depth for hypergraphs.

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Mathematics of computing → Hypergraphs

Keywords and phrases homomorphism indistinguishability, counting logics, guarded logics, hypergraphs, incidence graphs, tree depth, elimination forest, hypertree width

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.152

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2404.10637` [26]

Acknowledgements We thank Nicole Schweikardt for helpful discussions.

1 Introduction

The (k-dimensional) Weisfeiler-Leman algorithm describes a technique to classify the vertices (or k-tuples) of a graph, by iteratively computing a colouring (i.e., a classification) of the vertices (or k-tuples), which gets refined each iteration until it stabilises. While it can be used as a way to (imperfectly) test graphs for isomorphism, it has found many other – seemingly very different – uses, e.g. reducing the cost of solving linear programs [14], as graph kernels [29] or even as an architecture for graph neural networks [30, 22, 13, 12]. For a more in-depth overview on the expressive power of the Weisfeiler-Leman algorithm itself, consult [17] as a starting point. The success of the Weisfeiler-Leman algorithm can in part be explained by its simplicity, but also by the fact that it appears to capture the structure of a graph really well. This can be explained by its connection to first-order logic with counting quantifiers and to homomorphism counts over graphs of bounded tree width. A classical result due to Cai, Fürer and Immerman [5] and Immerman and Lander [16] says that two graphs are indistinguishable by the k-dimensional Weisfeiler-Leman algorithm if, and only if, they satisfy the same sentences of first-order logic with counting quantifiers (C) and k+1 variables (C^{k+1}).

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 152; pp. 152:1–152:18


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Dvořák [9] and Dell, Grohe, Rattan [8] related the Weisfeiler-Leman algorithm to homomorphism counts over graphs of bounded tree width (this was subsequently generalised to relational structures of bounded tree width by Butti and Dalmau [3]). They showed that two graphs are homomorphism indistinguishable over the class TW_k of graphs of tree width at most k if, and only if, they are indistinguishable by the k-dimensional Weisfeiler-Leman algorithm. Here, two graphs G and H are *homomorphism indistinguishable* over a class $\mathfrak{C}$ of graphs, if the number of homomorphisms from F into G equals the number of homomorphisms from F into H for all $F \in \mathfrak{C}$. Dvořák used the previously mentioned connection to C^{k+1} and an inductive characterisation of the graphs of bounded tree width to prove this result, while Dell et al. used elaborate algebraic techniques on vectors containing homomorphism counts.

In recent years, a whole theory has emerged around homomorphism indistinguishability. There are characterisations of homomorphism indistinguishability for classes of graphs other than TW_k (cf. [7, 10, 21, 25]), among which we would like to emphasise the following: A classical result by Lovász [18], stating that two graphs are isomorphic if, and only if, they are homomorphism indistinguishable over all graphs; a well-received result by Mančinska and Roberson [20], stating that two graphs are homomorphism indistinguishable over the class of planar graphs if, and only if, they are quantum isomorphic; and, of importance for this paper, Grohe [11] showed that two graphs are homomorphism indistinguishable over the graphs of tree depth at most m if, and only if, they satisfy the same sentences of C with quantifier rank at most m (C_m). There is also work concerned with a more fundamental analysis of homomorphism counting from restricted classes (cf. [2, 15, 23, 24, 28]).

Some real-world problems can be represented by hypergraphs in a much more natural way than by graphs. The great track record of the Weisfeiler-Leman method poses the question, whether a similar algorithm exists that works on *hypergraphs*. A direct application of the Weisfeiler-Leman algorithm on the incidence structure of a hypergraph is sometimes used. But Böker noted in [4], that this approach does not capture the hypergraph structure well, since the algorithm will mix up hyperedges and vertices. Thus, a proper variant of the Weisfeiler-Leman algorithm that works on hypergraphs is, to the best of our knowledge, still missing. We believe that establishing results analogous to the ones mentioned so far can give valuable insight on how the algorithm should operate on hypergraphs. A first step from this angle is a result by Scheidt and Schweikardt [27], who lift Dvořák's result to hypergraphs by proving the following: two hypergraphs G, H are homomorphism indistinguishable over the class GHW_k of hypergraphs of generalised hypertree width at most k if, and only if, they satisfy the same sentences of the logic GC^k. GC^k is a novel logic introduced in [27]. It has distinct variables for vertices and for hyperedges and counting quantifiers for both variable types. The main feature of GC^k is that it bounds the number of variables for hyperedges by k, and it requires that vertex variables are "guarded by" (i.e., contained in) hyperedge variables on every quantification.

Contributions. As the main contribution of this work, we show that two hypergraphs satisfy the same sentences of the logic GC^k with guard depth at most k if, and only if, they are homomorphism indistinguishable over the class of hypergraphs of strict hypertree depth at most k (Theorem 6.1). The guard depth is the quantifier depth of the hyperedge variables. This theorem follows from an inductive characterisation of the class of hypergraphs of strict hypertree depth $\leq k$, combined with the main technical lemmas of Scheidt and Schweikardt [27]. We believe that this inductive characterisation is interesting on its own, since the same technique combined with the core lemmata in Dvořák's work [9] can be used to give a concise proof of the analogous result on graphs due to Grohe [11], which was

independently recognised and shown by Fluck et al. [10] recently. Strict hypertree depth is a mild restriction of hypertree depth as defined by Adler, Gavenčiak and Klimošová [1]. This (as it turns out only slight) deviation from hypertree depth is surprising at first. Because of the properties and relations between strict hypertree depth and hypertree depth we acquire in this paper, we claim that strict hypertree depth can be viewed as a reasonable generalisation of tree depth for hypergraphs too. In particular, we show that the strict hypertree depth of a hypergraph is at most 1 larger than its hypertree depth (Theorem 2.5). Moreover, we show that the distinguishing power of homomorphism counts from hypergraphs of hypertree depth at most k is different from the distinguishing power of homomorphism counts from their respective incidence graphs (Theorem 2.9). Compared to other hypergraph parameters, this is very unexpected.

Organisation. The remainder of the paper is organised as follows. Section 2 is dedicated to the introduction of the necessary notation and definitions. In particular, we introduce incidence graphs as representations of hypergraphs that will be used throughout the paper, following Böker [4] and Scheidt and Schweikardt [27]. The notions of (strict) hypertree depth are introduced in Section 2.1, followed by Section 2.2 where we handle the differences between homomorphisms between hypergraphs and homomorphisms between incidence graphs. In Section 3 we introduce k-labeled incidence graphs that were the principle tool used in [27] to achieve their result. We utilise them in Section 4 to give an inductive characterisation of the hypergraphs of strict hypertree depth at most k (Theorem 4.1). Section 5 is devoted to the logic GC^k. In Section 6 we combine the results from Section 4 and Section 2.2 with the results from [27] to obtain Theorem 6.1. Section 7 concludes the paper with a summary of the results obtained in this paper, as well as an outlook on further research directions.

2 Preliminaries

Since we heavily rely on the work by Scheidt and Schweikardt [27], we will keep our notation close to theirs. We denote the set of natural numbers *including* 0 by $\mathbb{N}$, the set of *positive* natural numbers by $\mathbb{N}_{\geq 1}$, and we write $[n]$ to denote the set $\{1, 2, \ldots, n\}$. To denote isomorphism of two objects, we use $\cong$. A tuple is denoted using a bar, e.g. $\overline{a}$. For a given ℓ-tuple $\overline{a}$, we use a_i to denote the i-th element of $\overline{a}$, i.e., $\overline{a} = (a_1, a_2, \ldots, a_\ell)$. For any set S, let $\mathcal{P}(S)$ be the set of subsets of S and let $\mathcal{P}_k(S)$ be the subsets of cardinality exactly k. If S is a set of sets, let $\bigcup S = \bigcup_{s \in S} s$.

For a finite set S of cardinality $\ell \in \mathbb{N}$, a total order $<$ on S and any number $d \in \mathbb{N}$, we say that $\langle i_{d+1}, i_{d+2} \ldots, i_{d+\ell} \rangle$ is the *$<$-enumeration of S*, if $i_{d+1} < i_{d+2} < \cdots < i_{d+\ell}$ and $\{i_{d+1}, \ldots, i_{d+\ell}\} = S$. If the order $<$ is clear from the context, we simply say that $\langle i_{d+1}, i_{d+2} \ldots, i_{d+\ell} \rangle$ is the enumeration of S. Note that we usually let $d = 0$, i.e., we usually write $\langle i_1, \ldots, i_\ell \rangle$. Furthermore, the enumeration $\langle i_{d+1}, i_{d+2} \ldots, i_{d+\ell} \rangle$ of S is empty if, and only if, S is empty.

We denote a partial function f from A to B by $f \colon A \rightharpoonup B$, and we let $\mathrm{dom}(f) \coloneqq \{a \in A : f(a) \text{ is defined}\}$ and $\mathrm{img}(f) \coloneqq \{b \in B : \text{ex. } a \in A \text{ s.t. } f(a) = b\}$. We say that two functions f and g are *compatible*, if $f(x) = g(x)$ for all $x \in \mathrm{dom}(f) \cap \mathrm{dom}(g)$. We identify a (partial) function f with the set $\{(x, f(x)) : x \in \mathrm{dom}(f)\}$ whenever we are using set notation on functions. For example, we write $f \subseteq g$ to indicate $\mathrm{dom}(f) \subseteq \mathrm{dom}(g)$ and $f(x) = g(x)$ for all $x \in \mathrm{dom}(f)$. In particular, by $f \cup g$ we denote the function h with $\mathrm{dom}(h) = \mathrm{dom}(f) \cup \mathrm{dom}(g)$ and $h(x) = f(x)$ for all $x \in \mathrm{dom}(f)$ and $h(x) = g(x)$ for all $x \in \mathrm{dom}(g) \setminus \mathrm{dom}(f)$. Note that f has *precedence* over g, but this only matters if f and g are not compatible. For a

(partial) function f and a set $S \subseteq \mathrm{dom}(f)$ we write $f(S)$ to denote $\{f(x) : x \in S\}$, and we call the function $g \subseteq f$ with $\mathrm{dom}(g) \coloneqq S$ the *restriction of f to S*. Finally, we define partial functions inline like this: $\{a \to 3, b \to 2, c \to 5\}$. In particular, the empty set $\varnothing$ denotes a partial function with empty domain.

Graphs, Trees and Forests. An (undirected) graph is a tuple $G = (V(G), E(G))$, where $V(G)$ is a finite set and $E(G) \subseteq \mathcal{P}_2(V(G))$. For a set $S \subseteq V(G)$, $G[S]$ denotes the subgraph induced by S, i.e., $V(G[S]) \coloneqq S$ and $E(G[S]) \coloneqq E(G) \cap \mathcal{P}_2(S)$. A connected component of a graph is a maximal induced subgraph that is connected. A tree is a connected acyclic graph and a forest is a graph were each connected component is a tree. A rooted tree T is a tree with some distinguished node that we call its *root*, which we denote by ω_T. A rooted forest F is the disjoint union of a collection of rooted trees. It therefore has a set of roots denoted by Ω_F. We may omit the index if it is clear from the context. Note that a rooted tree is also a rooted forest and that every node n in a rooted forest is contained in a unique connected component which is a tree that we call the *tree for n* and whose root is the *root for n*.

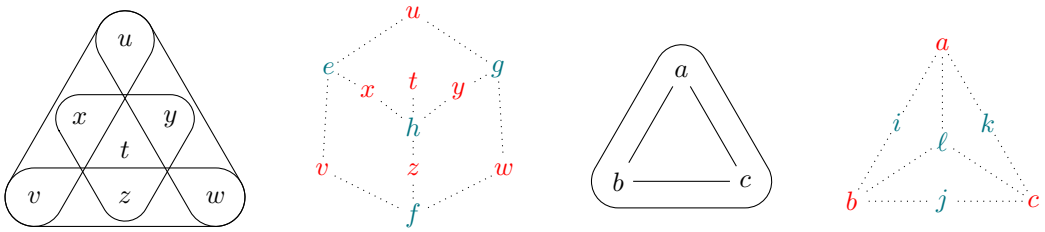
For a rooted forest F we let $\leq_F$ be the induced partial order on the nodes, i.e., the roots are the minimal elements and $s \leq_F t$ if s is on the unique path from t to its root in Ω. By $\mathrm{P}(s,t)$ we denote the set of nodes on the path from s to t (including s and t). In particular, if no path from s to t exists, $\mathrm{P}(s,t) = \varnothing$. By $\mathrm{P}(s)$ we denote the set of nodes on the path from s to the root for s and by $\wedge(s,t)$ we denote the unique element, if it exists, where the paths $\mathrm{P}(s)$, $\mathrm{P}(t)$ join, i.e., $\wedge(s,t) \coloneqq \max_{\leq_F}(\mathrm{P}(s) \cap \mathrm{P}(t))$. Notice that $\wedge(s,t)$ is undefined iff s and t are not in the same tree, and that $\wedge(s,t) = s$, iff $s \leq_F t$ (and conversely, $\wedge(s,t) = t$ iff $t \leq_F s$).

The subtree T_t induced by $t \in V(F)$ is the tree $F[V]$ with root t and $V \coloneqq \{s \in V(T) : t \leq_F s\}$. The *level* of a node $s \in V(F)$ is defined as the number of elements on the path from s to its root, i.e., $\mathrm{level}(s) \coloneqq |\mathrm{P}(s)|$. The *height* of a rooted tree T is the maximal level, i.e., $\mathrm{height}(T) \coloneqq \max\{\mathrm{level}(t) : t \in V(T)\}$ and the height of a node $t \in V(T)$ is the height of its induced subtree T_t, i.e., $\mathrm{height}(t) \coloneqq \mathrm{height}(T_t)$.

Hyper- and Incidence Graphs. A *hypergraph* is a tuple $\mathcal{H} = (V, E, \beta)$, where V and E are disjoint finite sets and β is a total function from E to $\mathcal{P}(V)$ with $V = \bigcup_{e \in E} \beta(e)$. We call the elements of V *vertices* and the elements of E *hyperedges* and for every $e \in E$, we call $\beta(e)$ its *contents*. We denote V by $V(\mathcal{H})$, E by $E(\mathcal{H})$ and β by $\beta_{\mathcal{H}}$, though we may omit the index if there is no ambiguity. Notice that, in general, multiple hyperedges with the same content and hyperedges without content are allowed. We call $\mathcal{H}$ *simple* if β is injective.

An *incidence graph* is a tuple $I = (R, B, E)$ consisting of two disjoint finite sets R and B of *red* and *blue* vertices and a relation $E \subseteq B \times R$. We denote R by $\mathcal{R}(I)$, B by $\mathcal{B}(I)$ and E by $E(I)$. For every $e \in \mathcal{B}(I)$, we let $\beta(e) \coloneqq \{v \in \mathcal{R}(I) : (e,v) \in E(I)\}$. Notice that β is equivalent in its function to the map β for a hypergraph, hence we denote them similarly. We only consider incidence graphs where for every $v \in \mathcal{R}(I)$ there is an $e \in \mathcal{B}(I)$ such that $(e,v) \in E(I)$. It is easy to see that we can assign an incidence graph to every hypergraph and the other way around: For every hypergraph $\mathcal{H}$ we let $I_{\mathcal{H}} \coloneqq (V(\mathcal{H}), E(\mathcal{H}), E)$ where $E \coloneqq \{(e,v) : e \in E(\mathcal{H}), v \in \beta(e)\}$. Conversely, for every incidence graph I we let $\mathcal{H}_I \coloneqq (\mathcal{R}(I), \mathcal{B}(I), \beta)$ where $\beta(e) \coloneqq \{v \in \mathcal{R}(I) : (e,v) \in E(I)\}$ for all $e \in \mathcal{B}(I)$.

For every set $S \subseteq E(\mathcal{H})$ we define the *induced subhypergraph* $\mathcal{H}[S]$ as $(V', S, \beta'_{\mathcal{H}})$ where $V' \coloneqq \bigcup_{e \in S} \beta(e)$ and $\beta'_{\mathcal{H}}$ is the restriction of $\beta_{\mathcal{H}}$ to S. We say that a hypergraph is connected if its incidence graph is connected. An induced subhypergraph is a connected component, if its corresponding incidence graph is a connected component.



(a) The hypergraph $\mathcal{H}$. **(b)** The IG $I_{\mathcal{H}}$ for $\mathcal{H}$. **(c)** The hypergraph $\mathcal{G}$. **(d)** The IG $I_{\mathcal{G}}$ for $\mathcal{G}$.

Figure 1 Examples for hypergraphs and their corresponding incidence graphs.

By $\mathcal{P}_n$ we denote the path of n hyperedges, where each hyperedge contains 2 vertices. I.e., we let $V(\mathcal{P}_n) = [n+1]$, $E(\mathcal{P}_n) = \{e_i : i \in [n]\}$ and $\beta(e_i) = \{i, i+1\}$ for all $i \in [n]$. We may use different names for the vertices if it is convenient.

▶ **Example 2.1.** The hypergraph $\mathcal{H}$ illustrated in Figure 1a is defined as $V(\mathcal{H}) = \{u, v, w, x, y, z, t\}$, $E(\mathcal{H}) = \{e, f, g, h\}$ and $\beta_{\mathcal{H}} = \{e \to \{u, v, x\}, f \to \{v, w, z\}, g \to \{u, w, y\}, h \to \{t, x, y, z\}\}$. Its incidence graph $I_{\mathcal{H}}$, depicted in Figure 1b, is defined by $\mathcal{R}(I_{\mathcal{H}}) = V(\mathcal{H})$, $\mathcal{B}(I_{\mathcal{H}}) = E(\mathcal{H})$ and $E(I_{\mathcal{H}}) = \{(e,u), (e,v), (e,x), (f,v), (f,w), (f,z), (g,u), (g,w), (g,y), (h,t), (h,x), (h,y), (h,z)\}$.

The hypergraph $\mathcal{G}$ depicted in Figure 1c is defined as $V(\mathcal{G}) = \{a, b, c\}$, $E(\mathcal{G}) = \{i, j, k, \ell\}$ and $\beta_{\mathcal{G}} \coloneqq \{i \to \{a, b\}, j \to \{b, c\}, k \to \{a, c\}, \ell \to \{a, b, c\}\}$. Its incidence graph $I_{\mathcal{G}}$, depicted in Figure 1d, is defined as $\mathcal{R}(I_{\mathcal{G}}) = V(\mathcal{G})$, $\mathcal{B}(I_{\mathcal{G}}) = E(\mathcal{G})$ and $E(I_{\mathcal{G}}) = \{(i,a), (i,b), (j,b), (j,c), (k,a), (k,c), (\ell,a), (\ell,b), (\ell,c)\}$.

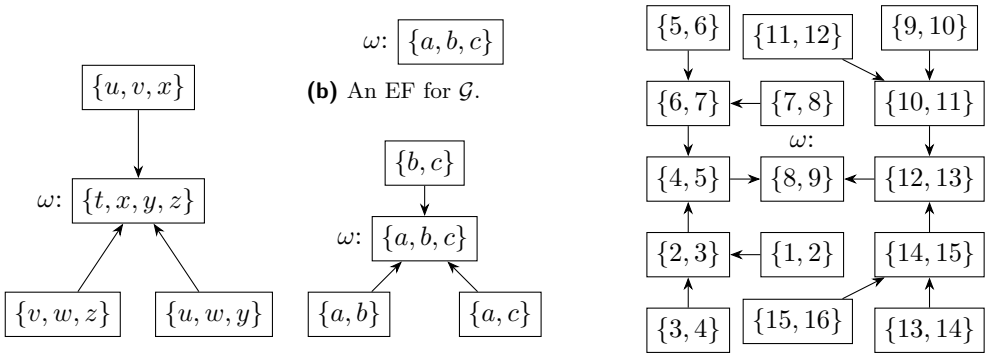
2.1 Hypertree Depth

The following definition of *elimination forest* and *hypertree depth* is due to Adler, Gavenčiak and Klimošová [1], though they refer to elimination forests as "decomposition forests". We call them elimination forests, since this reflects their conceptual similarity to elimination forests for graphs and avoids confusion with hypertree decompositions. Further, we define this notion in terms of incidence graphs, because we mainly work on those. Do notice however, that this definition easily translates to hypergraphs and that it is equivalent to the one given by Adler, Gavenčiak and Klimošová.

▶ **Definition 2.2** (Hypertree Depth and Elimination Forests, [1])**.** Let I be an incidence graph. An *elimination forest* (F, Γ) for I consists of a forest F and a mapping $\Gamma\colon V(F) \to \mathcal{B}(I)$ such that conditions 1–3 hold. We write $\widehat{\Gamma}(t)$ as shorthand for $\beta(\Gamma(t))$.

1. *Completeness for vertices*: For every red vertex $v \in \mathcal{R}(I)$, there is a $t \in V(F)$ such that $v \in \widehat{\Gamma}(t)$.
2. *Hyperedge-Containment*: For every blue vertex $e \in \mathcal{B}(I)$ there are nodes $s, t \in V(F)$ such that $s \leq_F t$ and $\beta(e) \subseteq \bigcup \widehat{\Gamma}(\mathrm{P}(s,t))$.
3. *Shared heritage*: For all $s, t \in V(F)$, if $\widehat{\Gamma}(s) \cap \widehat{\Gamma}(t) \neq \varnothing$, then $\wedge(s,t)$ is defined and $\widehat{\Gamma}(s) \cap \widehat{\Gamma}(t) \subseteq \bigcup \widehat{\Gamma}(\mathrm{P}(\wedge(s,t)))$.

The intuition behind condition 3 is that hyperedges can only share the vertices contained in their common ancestors in the elimination forest. The height of an elimination forest (F, Γ) is simply the height of F. The *hypertree depth of* I is defined as the minimal height over all elimination forests for I, and we denote it by $\mathrm{hd}(I)$. Analogously, we let $\mathrm{hd}(\mathcal{H}) \coloneqq \mathrm{hd}(I_{\mathcal{H}})$ for all hypergraphs. We write IHD_k to denote the class of incidence graphs of hypertree depth at most k and HD_k to denote the corresponding class of hypergraphs. ⌟



(a) A (strict) EF for $\mathcal{H}$. **(b)** An EF for $\mathcal{G}$. **(c)** A strict EF for $\mathcal{G}$. **(d)** A (strict) elimination forest for $\mathcal{P}_{15}$.

Figure 2 Elimination forests for various (hyper)graphs.

We call an elimination forest (F, Γ) *strict* if Γ is bijective. It is easy to see that the first and second condition are trivially satisfied when Γ is bijective, thus we can redefine the notion of strict elimination forest in a more succinct manner.

▶ **Definition 2.3** (Strict Elimination Forest)**.** Let I be an incidence graph. A *strict elimination forest* (F, Γ) for I consists of a forest F and a *bijective* function $\Gamma \colon V(F) \to \mathcal{B}(I)$ satisfying condition 3 of Definition 2.2. The *strict hypertree depth of* I, denoted by $\mathrm{shd}(I)$, is defined as the minimal height over all strict elimination forests for I. Again, we let $\mathrm{shd}(\mathcal{H}) \coloneqq \mathrm{shd}(I_{\mathcal{H}})$ for every hypergraph $\mathcal{H}$. We write ISHD_k to denote the class of incidence graphs of strict hypertree depth at most k and SHD_k to denote the corresponding class of hypergraphs. ⌟

▶ **Example 2.4.** Consider the hypergraphs $\mathcal{G}$ and $\mathcal{H}$ as well as their incidence graphs $I_{\mathcal{G}}$, $I_{\mathcal{H}}$ from Example 2.1 (see Figure 1). Let (F_1, Γ_1) be defined as follows. F_1 is a tree defined by $V(F_1) = \{t_1, t_2, t_3, t_4\}$ and $E(F_2) \coloneqq \{\{t_1, t_2\}, \{t_1, t_3\}, \{t_1, t_4\}\}$ with root t_1. Γ_1 is a map defined as $\{t_1 \to h, t_2 \to e, t_3 \to f, t_4 \to g\}$. (F_1, Γ_1) is an elimination forest of height 2 for $I_{\mathcal{H}}$. It is depicted in Figure 2a, where we labeled the node t_i with $\widehat{\Gamma}_1(t_i)$. Notice, that (F_1, Γ_1) is strict.

Analogously, we depicted two elimination forests for $I_{\mathcal{G}}$ of height 1 and 2 in Figures 2b and 2c. Notice how the elimination forest depicted in Figure 2b, witnessing $\mathrm{hd}(\mathcal{G}) = 1$, is not strict. It is easy to see that $\mathrm{shd}(\mathcal{G}) \geq 2$ since a bijective map implies that there are as many nodes in the forest as there are edges. Thus, Figure 2c witnesses that $\mathrm{shd}(\mathcal{G}) = 2$.

Finally, Figure 2d depicts a (strict) elimination forest for $\mathcal{P}_{15}$ witnessing $\mathrm{shd}(\mathcal{P}_{15}) \leq 4$. Recall that $\mathcal{P}_{15}$ is defined by $V(\mathcal{P}_{15}) = [16]$ and $E(\mathcal{P}_{15}) = \{\{i, i{+}1\} : i \in [15]\}$. ⌟

One can show that the strict hypertree depth of a hypergraph is at most its hypertree depth increased by one. The main idea is that we can turn every elimination forest into a strict one, if we add a leaf for every hyperedge that is currently not being mapped to below the path that contains it according to condition 2 in Definition 2.2. The proof can be found in the full version.

▶ **Theorem 2.5.** *For all hypergraphs* $\mathcal{H}$, $\mathrm{hd}(\mathcal{H}) \leq \mathrm{shd}(\mathcal{H}) \leq \mathrm{hd}(\mathcal{H}){+}1$.

Finally, it is easy to see that, due to condition 3 (shared heritage), every elimination forest of a connected incidence graph is also an *elimination tree*.

▶ **Lemma 2.6.** *Let* I *be a connected incidence graph. For every elimination forest* (F, Γ) *of* I*,* F *is a tree.*

2.2 Homomorphisms

While hypergraphs and incidence graphs are conceptually close, their "natural" notions of homomorphisms are not the same. Since our interest lies in hypergraphs, but we are mainly working on incidence graphs in this paper, we have to relate these notions. Following Scheidt and Schweikardt, we use the same definitions as Böker [4].

A homomorphism from a hypergraph $\mathcal{H}$ into another hypergraph $\mathcal{G}$ is a pair of functions $(h_V\colon V(\mathcal{H}) \to V(\mathcal{G}), h_E\colon E(\mathcal{H}) \to E(\mathcal{G}))$ such that for every $e \in E(\mathcal{H})$ the equality $h_V(\beta(e)) = \beta(h_E(e))$ holds.

A homomorphism from an incidence graph I into another incidence graph J is a pair of mappings $(h_V\colon \mathcal{R}(I) \to \mathcal{R}(J), h_E\colon \mathcal{B}(I) \to \mathcal{B}(J))$ such that $(h_E(e), h_V(v)) \in E(J)$ holds for every edge $(e, v) \in E(I)$. This is equivalent to the requirement $h_V(\beta(e)) \subseteq \beta(h_E(e))$. Thus, the equality that we require for a hypergraph homomorphism is relaxed to an inclusion for incidence graphs.

Let A, B and $\mathfrak{C}$ be two hypergraphs and a class of hypergraphs or two incidence graphs and a class of incidence graphs. We denote the number of homomorphisms from A to B by $\hom(A, B)$, and we let $\mathrm{Hom}(\mathfrak{C}, A)$ be the "vector" that has a row for every $F \in \mathfrak{C}$ containing $\hom(F, A)$. We say that A and B are *homomorphism indistinguishable* over $\mathfrak{C}$ ($A \equiv_{\mathfrak{C}} B$), if $\mathrm{Hom}(\mathfrak{C}, A) = \mathrm{Hom}(\mathfrak{C}, B)$, i.e., if $\hom(F, A) = \hom(F, B)$ for all $F \in \mathfrak{C}$.

The following crucial theorem relates homomorphism indistinguishability over a class of hypergraphs to homomorphism indistinguishability over the corresponding class $\mathfrak{C}_{\mathsf{I}}$ of incidence graphs. As noted in [27], this theorem is implicit in [4], consult Appendix A of the full version of [27] for details.

▶ **Theorem 2.7** ([4, 27]). *Let $\mathfrak{C}$ be a class of hypergraphs and let $\mathfrak{C}_{\mathsf{I}}$ be its corresponding class of incidence graphs. If $\mathfrak{C}$ is closed under* pumping *and* local merging*, then $\mathrm{Hom}(\mathfrak{C}, \mathcal{G}) = \mathrm{Hom}(\mathfrak{C}, \mathcal{H})$ if, and only if, $\mathrm{Hom}(\mathfrak{C}_{\mathsf{I}}, I_{\mathcal{G}}) = \mathrm{Hom}(\mathfrak{C}_{\mathsf{I}}, I_{\mathcal{H}})$ for all hypergraphs $\mathcal{G}$ and $\mathcal{H}$.*

$\mathfrak{C}$ is closed under pumping, if $H' \in \mathfrak{C}$ for every $H \in \mathfrak{C}$, where H' is created from H by inserting a new vertex into *one* arbitrary hyperedge of H; and closed under local merging, if $H' \in \mathfrak{C}$ for every $H \in \mathfrak{C}$, where H' is created from H by choosing an arbitrary hyperedge e and then merging two vertices u, v that are both contained in e.

It is easy to see that SHD_k is closed under both pumping and local merging, whereas HD_k is only closed under local merging.

▶ **Proposition 2.8.** *Let $k \in \mathbb{N}_{\geq 1}$. The class SHD_k is closed under pumping and local merging, the class HD_k is closed under local merging but* ***not*** *under pumping.*

The following theorem shows that homomorphism indistinguishability over HD_k is not equal to homomorphism indistinguishability over SHD_k. Because $\mathsf{SHD}_k \subseteq \mathsf{HD}_k$, counting homomorphisms from HD_k is more powerful in the sense that it distinguishes more hypergraphs. But we also show that it is unequal to homomorphism indistinguishability over IHD_k, which is unexpected. Since this prohibits us from relating HD_k to any fragment of GC, it is conceivable that this could pose a problem in other scenarios too. Thus, we argue that the notion of strict hypertree depth can be viewed as a reasonable generalisation of tree depth, especially when we recall that $\mathsf{HD}_{k-1} \subseteq \mathsf{SHD}_k \subseteq \mathsf{HD}_k$.

▶ **Theorem 2.9.** *For every $k \in \mathbb{N}_{\geq 1}$ there exist pairs of hypergraphs $(\mathcal{G}_k, \mathcal{H}_k)$ and $(\mathcal{G}'_k, \mathcal{H}'_k)$, such that:*

1. $\mathrm{Hom}(\mathsf{SHD}_k, \mathcal{G}_k) = \mathrm{Hom}(\mathsf{SHD}_k, \mathcal{H}_k)$, *but* $\mathrm{Hom}(\mathsf{HD}_k, \mathcal{G}_k) \neq \mathrm{Hom}(\mathsf{HD}_k, \mathcal{H}_k)$;
2. $\mathrm{Hom}(\mathsf{HD}_k, \mathcal{G}'_k) = \mathrm{Hom}(\mathsf{HD}_k, \mathcal{H}'_k)$, *but* $\mathrm{Hom}(\mathsf{IHD}_k, I_{\mathcal{G}'_k}) \neq \mathrm{Hom}(\mathsf{IHD}_k, I_{\mathcal{H}'_k})$.

(a) $\mathcal{G}_1$. **(b)** $\mathcal{H}_1$. **(c)** $\mathcal{G}'_1$. **(d)** $\mathcal{H}'_1$.

Figure 3 $(\mathcal{G}_1, \mathcal{H}_1)$, $(\mathcal{G}'_1, \mathcal{H}'_1)$ witness Theorem 2.9 for $k = 1$. Circles denote singleton hyperedges.

For $k = 1$ this is easy to see: A connected hypergraph has strict hypertree depth 1 iff it consists of a single hyperedge, whereas a connected hypergraph has hypertree depth 1 if one hyperedge contains all vertices. It is therefore not hard to see that the statement of the theorem holds for $k = 1$ using the hypergraphs depicted in Figure 3. For $k \geq 2$ a similar idea for the construction of $(\mathcal{G}_k, \mathcal{H}_k)$ and $(\mathcal{G}'_k, \mathcal{H}'_k)$ works, but we had to defer the details to the full version due to space constraints.

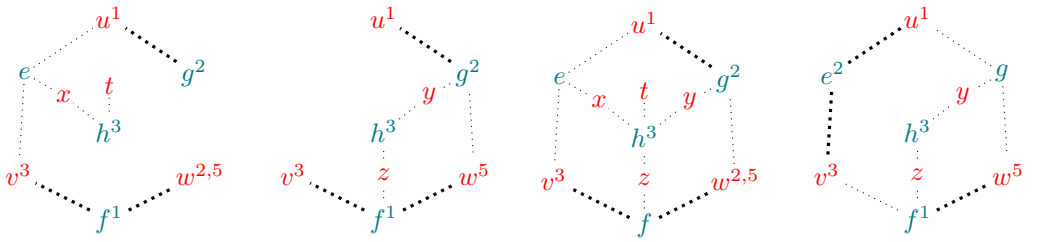
3 k-Labeled Incidence Graphs

Our goal is to give an inductive characterisation of the incidence graphs of strict hypertree depth at most k (and thus also of hypergraphs of strict hypertree depth at most k). The concepts presented in this section were first defined in [27], and we adopt their notation and phrasing for the most part. Note that the k-labeled incidence graphs defined here are inspired by the concept of k-labeled graphs as they are used in [6, 9, 10, 19] and elsewhere. In particular, k-labeled graphs are the main tool used by Dvořák [9] to prove his result. In principle, a *k-labeled incidence graph* is an incidence graphs that has *labels* attached to some of its red and blue vertices. We have an unbounded number of red labels that can be attached to red vertices (though the number of labels actually used must always be finite), but we only have k labels that we can attach to blue vertices. We are allowed to attach multiple labels to the same vertex, but we are not required to use all of them. Every red label has an assigned "guard", which is a blue vertex with a label on it. In practice, we will require every red labeled vertex to be a neighbour of its guard (i.e., we want it to have a *real* guard, as defined in the next paragraph), though it makes the proofs easier if we do not enforce this in the definition itself. This idea is formalised as follows.

A k-labeled incidence graph is a tuple $L = (I, r, b, g)$, where I is an incidence graph, $r\colon \mathbb{N}_{\geq 1} \rightharpoonup \mathcal{R}(I)$, $b\colon [k] \rightharpoonup \mathcal{B}(I)$ and $g\colon \mathbb{N}_{\geq 1} \rightharpoonup [k]$ are partial mappings such that $\mathrm{dom}(r)$ is finite and $\mathrm{dom}(g) = \mathrm{dom}(r)$. We use I_L, r_L, ... to denote the components of L. But to keep the indices from getting overly complicated, we may write $I', r', \ldots$ and $I_i, r_i, \ldots$ instead of $I_{L'}, r_{L'}$ and $I_{L_i}, r_{L_i}, \ldots$, respectively. If it is clear from the context, we may omit the index altogether and simply write I, r, b, g.

We say that L *has real guards (w.r.t. g)*, if for every $i \in \mathrm{dom}(r)$ we have $g(i) \in \mathrm{dom}(b)$ and $(b(g(i)), r(i)) \in E(I)$. A k-labeled incidence graph L is *label-free* if $r_L = b_L = g_L = \varnothing$. We call I the *skeleton* of L. Next, we define some operations on k-labeled incidence graphs.

For any set $X_r \subseteq \mathbb{N}_{\geq 1}$ of finite size ℓ and any tuple $\overline{v} = (v_1, v_2, \ldots, v_\ell) \in \mathcal{R}(I_L)^n$ we write $L[X_r \to \overline{v}]$ to denote a copy of L where we modified r such that $r(i_j) = v_j$ for all j in the enumeration $\langle i_1, \ldots, i_\ell \rangle$ of X_r, i.e., we introduce, and change the placement of, some red labels. Similarly, for any $X_b = \{i_1, \ldots, i_\ell\} \subseteq [k]$ and any $\overline{e} = (e_1, e_2, \ldots, e_\ell) \in \mathcal{B}(I_L)^\ell$ we write $L\langle X_b \to \overline{e} \rangle$ to denote a copy of L where we modified b accordingly. We write $L[X_r \to \bullet]$ ($L\langle X_b \to \bullet \rangle$) to denote a copy of L where we *removed* the red (blue) labels in X_r (X_b). Note, that we remove *just* the labels and *not* the vertices that carry them.



(a) The 3-labeled incidence graph L_1.

(b) The 3-labeled incidence graph L_2.

(c) An example of glueing: $(L_1 \cdot L_2)$.

(d) An example of applying a transition: $L_1[\rightsquigarrow f]$.

Figure 4 3-labeled incidence graphs and operations on them. Labels are encoded as exponents and the guard function is encoded using thicker edges between the red vertex and its guard.

Intuitively, the "product" $(L_1 \cdot L_2)$ or *glueing* of two k-labeled incidence graphs L_1, L_2 is the k-labeled incidence graph L that is created by first taking the disjoint union of L_1 and L_2, followed by repeatedly merging pairs of red (blue) vertices, that carry a shared red (blue) label. By merging we mean that we replace these vertices by a single fresh vertex, which inherits their neighbourhoods and labels. We apply this procedure until there are no more such pairs. The guard function of $(L_1 \cdot L_2)$ is simply $g_{L_1} \cup g_{L_2}$, i.e., in theory, g_{L_1} has precedence over g_{L_2}. In practice, we will require that g_{L_1} and g_{L_2} are compatible, which means the precedence of g_{L_1} will be irrelevant. Note that the order in which we merge vertices does not matter, and that if a vertex carries two or more labels, all vertices carrying any one of these labels will be replaced by a single fresh vertex that carries *all* those labels and inherits *all* neighbourhoods. Finally, for $i \in [2]$ we define mappings $\mathrm{succ}^R_{L_i} \colon \mathcal{R}(I_{L_i}) \to \mathcal{R}(I_L)$ and $\mathrm{succ}^B_{L_i} \colon \mathcal{B}(I_{L_i}) \to \mathcal{B}(I_L)$ such that $\mathrm{succ}^R_{L_i}(v)$ is the red vertex of I_L that corresponds to $v \in \mathcal{R}(I_i)$, and $\mathrm{succ}^B_{L_i}(e)$ is the blue vertex of I_L that corresponds to $e \in \mathcal{B}(I_i)$.

▶ **Example 3.1.** Consider the k-labeled incidence graphs L_1, L_2 according to Figures 4a and 4b. In particular, we have

i	1	2	3	5
$r_1(i)$	u	w	v	w
$r_2(i)$	u	–	v	w

i	1	2	3
$b_1(i)$	f	g	h
$b_2(i)$	f	g	h

and

i	1	2	3	5
$g_1(i)$	2	1	1	1
$g_2(i)$	2	–	1	1

.

The product $(L_1 \cdot L_2)$ is depicted in Figure 4c.

So far, we should not be allowed to remove a blue label from a vertex, if it serves as the guard of a red label. But sometimes we want to *transition* from one (real) guard assignment to another (real) guard assignment. I.e., we want to remove blue labels even if they still guard some red labels, because we guarantee that we introduce new guards for these labels right away. We formalise this operation as a special partial function, that assigns new guards to existing red labels: We call $f \colon \mathbb{N}_{\geq 1} \rightharpoonup [k]$ a *transition for L (for g)*, if $\varnothing \neq \mathrm{dom}(f) \subseteq \mathrm{dom}(g)$ and for all $i \in \mathrm{dom}(g)$ we have that if $g(i) \in \mathrm{img}(f)$, then $i \in \mathrm{dom}(f)$. This means that if f reassigns the blue label guarding the red label i, then f provides a new guard for i. Applying a transition, denoted by $L[\rightsquigarrow f]$, means modifying a copy of L as follows: we want to insert fresh vertices with these blue labels, thus we must first remove all blue labels, that are currently in use, i.e., we must first remove the labels in the set $X_b \coloneqq \mathrm{img}(f) \cap \mathrm{dom}(b_L) \cap \mathrm{img}(g_L)$ from b. Notice that we have to intersect with $\mathrm{dom}(b_L)$ since we do not require L to have real guards. After removing the labels in X_b, we insert $|X_b|$ new blue vertices into I_L, each carrying one of the blue labels in X_b, and introduce an edge between $b(f(i))$ and $r(i)$ for all

$i \in \mathrm{dom}(f)$. Finally, we redefine the guard function as $f \cup g_L$. Note that this procedure can be easily expressed as the product $(M_f \cdot L\langle X_b \to \bullet\rangle)$ for a suitably defined k-labeled incidence graph M_f.

▶ **Example 3.2.** Consider the k-labeled incidence graph L_1 from Example 3.1 and Figure 4a. The partial function $f = \{1 \to 2, 3 \to 2\}$ is a transition for L_1. The result $L_1[\rightsquigarrow f]$ of the application of f on L_1 is depicted in Figure 4d.

We define the class GLI_k^i of k-labeled incidence graphs that can be constructed in a way that at most i blue labels are removed "in series".

▶ **Definition 3.3.** For $k \in \mathbb{N}_{\geq 1}$ and $i \in \mathbb{N}$ we define the set GLI_k^i inductively as follows.

Base case. $L \in \mathsf{GLI}_k^0$ for all k-labeled incidence graphs L with $\mathrm{dom}(r) = \mathcal{R}(I)$, $\mathrm{dom}(b) = \mathcal{B}(I)$ and real guards.
For all $i \in \mathbb{N}$, if $L \in \mathsf{GLI}_k^i$, then $L \in \mathsf{GLI}_k^{i+1}$.

Glueing. Let $L_1 \in \mathsf{GLI}_k^{i_1}$, $L_2 \in \mathsf{GLI}_k^{i_2}$ have compatible guard functions and $L = (L_1 \cdot L_2)$.
Then, $L \in \mathsf{GLI}_k^i$ where $i := \max\{i_1, i_2\}$.

Transitioning. Let $L \in \mathsf{GLI}_k^i$, let f be a transition for L and $L' = L_1[\rightsquigarrow f]$.
Then, $L' \in \mathsf{GLI}_k^{i'}$ where $i' := i + |\,\mathrm{img}(f) \cap \mathrm{img}(b_L) \cap \mathrm{img}(g_L)|$.

Label-Removal. Let $L \in \mathsf{GLI}_k^i$.

(a) For $X_r \subseteq \mathrm{dom}(r)$, $L[X_r \to \bullet] \in \mathsf{GLI}_k^i$.

(b) For $X_b \subseteq \mathrm{dom}(b) \setminus \mathrm{img}(g)$, $L\langle X_b \to \bullet\rangle \in \mathsf{GLI}_k^{i'}$ where $i' := i + |X_b|$.

Finally, we let $\mathsf{GLI}^k := \mathsf{GLI}_k^k$ for every $k \in \mathbb{N}$. ⌟

4 Characterising Hypergraphs of Strict Hypertree Depth at most k

In this section we prove that the inductively defined class GLI^k corresponds precisely to the class ISHD_k.

▶ **Theorem 4.1.** *An incidence graph J has strict hypertree depth at most k if, and only if, there exists a label-free $L \in \mathsf{GLI}^k$ such that $I_L \cong J$.*

In the following, we first show how to construct an incidence graph of strict hypertree depth at most k as the skeleton of a label-free k-labeled incidence graph in GLI^k (Lemma 4.2). Then we show that every label-free $L \in \mathsf{GLI}^k$ has strict hypertree depth at most k (Lemma 4.3). Theorem 4.1 follows directly from the combination of these two Lemmata.

For the rest of this section, let $J \in \mathsf{ISHD}_k$ and let (T, Γ) be a strict elimination forest of height $\leq k$ for J. We can w.l.o.g. assume that J is connected and that T is a tree (Lemma 2.6). Let $\mathcal{R}(J) = \{v_1, v_2, \ldots, v_m\}$ where $m \geq 1$.

(T, Γ) will help us decide when to remove (i.e., eliminate) which label from which blue vertex in the following sense. The core idea is to start with a trivial k-labeled incidence graph for every path from a leaf to the root in the elimination tree. Then we walk bottom-up along these paths and whenever several paths join in a node, we apply red and blue vertex removals in a suitable way on their k-labeled incidence graphs, such that afterwards we can glue them together and receive a k-labeled incidence graph that is isomorphic to the incidence graph induced by the union of said paths.

For this we need the following notions: For a node n in a tree T, the *subtree with stem induced by* n is the tree $\dot{T}_n$ induced on T by the set $\mathrm{P}(n) \cup \{t \in V(T) : n \leq_T t\}$. Recall that $\mathrm{P}(n)$ is the set of nodes on the path from n to the root ω (including n and ω), and

notice that the subtree with stem induced by the root is T, i.e., $T_\omega = T$, and for every leaf n it is the path from the root ω to n. For every node $n \in V(T)$ we define the set $\text{labels}(n) \coloneqq \{i \in [m] : v_i \in \widehat{\Gamma}(n)\}$. To avoid an overload of notation, we will write $\text{labels}(N)$ to denote the set $\bigcup_{n\in N} \text{labels}(n)$ and write $J[\dot{T}_n]$ to abbreviate $J[\widehat{\Gamma}(V(\dot{T}_n))]$. With these notions, we can prove the following lemma via induction.

▶ **Lemma 4.2.** *For every $n \in V(T)$ of level d where $\langle t_1, \dots, t_d\rangle$ is a $\leq_T$-enumeration of $\mathrm{P}(n)$ (i.e., in particular $t_1 = \omega$, $t_d = n$), there exists an $L_n \in \mathsf{GLI}_k^{k-d}$ of the form (I, r, b, g) such that*

(A) $\text{dom}(b) = [d]$ *and* $\text{dom}(r) = \text{labels}(\mathrm{P}(n))$*;*

(B) $g(i) \coloneqq \min\{j \in [d] : v_i \in \widehat{\Gamma}(t_j)\}$ *for every* $i \in \text{dom}(g)$*;*

(C) *There exists an isomorphism (π_R, π_B) between I and $J[\dot{T}_n]$ such that*

- **(i)** $\pi_R(r(i)) = v_i$ *for all* $i \in \text{dom}(r)$*, and*
- **(ii)** $\pi_B(b(j)) = \Gamma(t_j)$ *for all* $j \in \text{dom}(b)$*.*

Notice that, in particular, this lemma states $J \cong I_{L_\omega}$ for the root ω of T. But L_ω is not label-free, since $\text{dom}(b_{L_\omega}) = \{1\}$ and $\text{dom}(r_{L_\omega}) = \text{labels}(\mathrm{P}(\omega)) = \text{labels}(\omega)$. But, the lemma also states that $L_\omega \in \mathsf{GLI}_k^{k-1}$, since $\text{level}(\omega) = 1$. Thus, $J \cong I_{L'}$ for $L' = L_\omega[\text{labels}(\omega)\to\bullet]\langle\{1\}\to\bullet\rangle$, and in particular, $L' \in \mathsf{GLI}_k^k$ is label-free. Thus, this lemma shows the forward direction of Theorem 4.1.

The following lemma can be shown by induction. On a high level, the idea of the proof is to only modify the elimination forest, if blue labels are removed. At that point, we prepend a chain of new nodes to the root(s) of the elimination forest. If we take the product of two k-labeled incidence graphs, we take the union of the forests, and if we remove red labels, we do not alter the forest at all.

▶ **Lemma 4.3.** *For every $L \in \mathsf{GLI}_k^d$ of the form (I, r, b, g) there is a tuple (F, Γ), where F is a forest of height $\leq d$ and Γ is a bijective function from $V(F)$ to $\mathcal{B}(I) \setminus \text{img}(b)$ satisfying condition A. We write $\widehat{\Gamma}(t)$ as a shorthand for $\beta(\Gamma(t))$ and $\widetilde{\Gamma}(t)$ as a shorthand for $\widehat{\Gamma}(t) \setminus \text{img}(r)$.*

(A) *For all $s, t \in V(F)$, and all $v \in \widetilde{\Gamma}(s) \cap \widetilde{\Gamma}(t) \neq \varnothing$ it holds that:*
$v \in \beta(b(j))$ for a $j \in \text{dom}(b)$ or $\wedge(s,t)$ is defined and $v \in \bigcup \widetilde{\Gamma}(\mathrm{P}(\wedge(s,t)))$.

Notice that, if $L \in \mathsf{GLI}_k^d$ is label-free, this guarantees a strict elimination forest (F, Γ) of height d for I_L. This shows the backward direction of Theorem 4.1.

5 The Logic GC^k

This section introduces the logic GC^k as defined in [27] and its restricted fragment GC_k, consisting of all formulas of guard depth at most k. Let k be a positive natural number, that is fixed for this section.

Variables. GC^k uses two different kinds of variables: $\mathsf{VAR}_\mathsf{v} \coloneqq \{\mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3 \dots\}$ to address vertices and $\mathsf{VAR}_\mathsf{e} \coloneqq \{\mathsf{e}_1, \mathsf{e}_2, \dots, \mathsf{e}_k\}$ to address hyperedges. Notice that the number of variables for hyperedges is bounded by k, but unbounded for vertices. We say that a tuple of the form $\overline{\mathsf{v}} = (\mathsf{v}_{i_1}, \dots, \mathsf{v}_{i_\ell}) \in \mathsf{VAR}_\mathsf{v}^\ell$ or $\overline{\mathsf{e}} = (\mathsf{e}_{i_1}, \dots, \mathsf{e}_{i_\ell}) \in \mathsf{VAR}_\mathsf{e}^\ell$ is a v- *or* e-*tuple*, if $i_1 < i_2 < \dots < i_\ell$. We let $\text{vars}(\overline{\mathsf{v}}) \coloneqq \{\mathsf{v}_{i_1}, \dots, \mathsf{v}_{i_\ell}\}$ and $\text{vars}(\overline{\mathsf{e}}) \coloneqq \{\mathsf{e}_{i_1}, \dots, \mathsf{e}_{i_\ell}\}$ respectively. We call $\{i_1, \dots, i_\ell\}$ the *index set* of $\overline{v}$ and $\overline{e}$, respectively.

Logical Guards. The key idea behind GC^k is that on quantification, vertex variables must be *guarded* by hyperedge variables. This is formalised by a partial function $g\colon \mathbb{N}_{\geq 1} \rightharpoonup [k]$ with finite domain (similar to the guard function of a k-labeled incidence graph, cf. Section 3) and its corresponding *logical guard* $\Delta_g \coloneqq \bigwedge_{i \in \mathrm{dom}(g)} E(\mathsf{e}_{g(i)}, \mathsf{v}_i)$. For the special partial function g with empty domain, we let $\Delta_g \coloneqq \top$, which is a special formula that always evaluates to true.

▶ **Definition 5.1.** The logic GC^k is inductively defined along with the *free vertex variables*, the *free hyperedge variables* and the *guard depth*, as formalised by the functions

$$\mathrm{free}_{\mathsf{v}}\colon \mathsf{GC}^k \to \mathcal{P}(\mathsf{VAR}_{\mathsf{v}}), \quad \mathrm{free}_{\mathsf{e}}\colon \mathsf{GC}^k \to \mathcal{P}(\mathsf{VAR}_{\mathsf{e}}), \quad \text{and} \quad \mathrm{gd}\colon \mathsf{GC}^k \to \mathbb{N}.$$

Atomic Formulas. For all $i, i' \in \mathbb{N}_{\geq 1}$ and all $j, j' \in [k]$ the following formulas are in GC^k:

- $\varphi = \mathsf{v}_i = \mathsf{v}_{i'}$ with $\mathrm{free}_{\mathsf{v}}(\varphi) \coloneqq \{\mathsf{v}_i, \mathsf{v}_{i'}\}$ and $\mathrm{free}_{\mathsf{e}}(\varphi) \coloneqq \varnothing$;
- $\varphi = \mathsf{e}_j = \mathsf{e}_{j'}$ with $\mathrm{free}_{\mathsf{v}}(\varphi) \coloneqq \varnothing$ and $\mathrm{free}_{\mathsf{e}}(\varphi) \coloneqq \{\mathsf{e}_j, \mathsf{e}_{j'}\}$;
- $\varphi = E(\mathsf{e}_j, \mathsf{v}_i)$ with $\mathrm{free}_{\mathsf{v}}(\varphi) \coloneqq \{\mathsf{v}_i\}$ and $\mathrm{free}_{\mathsf{e}}(\varphi) \coloneqq \{\mathsf{e}_j\}$.

In all the above cases, $\mathrm{gd}(\varphi) \coloneqq 0$.

Inductive Rules. Let χ, ψ be formulas of GC^k. The following formulas are in GC^k.

- $\varphi = \neg\chi$ with $\mathrm{free}_{\mathsf{v}}(\varphi) \coloneqq \mathrm{free}_{\mathsf{v}}(\chi)$ and $\mathrm{free}_{\mathsf{e}}(\varphi) \coloneqq \mathrm{free}_{\mathsf{e}}(\chi)$, and $\mathrm{gd}(\varphi) \coloneqq \mathrm{gd}(\chi)$;
- $\varphi = (\chi \wedge \psi)$ with $\mathrm{free}_{\mathsf{v}}(\varphi) \coloneqq \mathrm{free}_{\mathsf{v}}(\chi) \cup \mathrm{free}_{\mathsf{v}}(\psi)$ and $\mathrm{free}_{\mathsf{e}}(\varphi) \coloneqq \mathrm{free}_{\mathsf{e}}(\chi) \cup \mathrm{free}_{\mathsf{e}}(\psi)$, and $\mathrm{gd}(\varphi) \coloneqq \max\{\mathrm{gd}(\chi), \mathrm{gd}(\psi)\}$.

Note that by the rules defined so far, $\mathrm{gd}(\Delta_g) = 0$ for all logical guards Δ_g.

We say that $g\colon \mathbb{N}_{\geq 1} \rightharpoonup [k]$ *is a guard function for* φ if $\mathrm{dom}(g) = \{i : \mathsf{v}_i \in \mathrm{free}_{\mathsf{v}}(\varphi)\}$.

Let $n \in \mathbb{N}_{\geq 1}$, let g be a guard function for ψ and $\chi = (\Delta_g \wedge \psi)$. The following formulas are in GC^k for every v-tuple $\overline{\mathsf{v}}$ with $\mathrm{vars}(\overline{\mathsf{v}}) \subseteq \mathrm{free}_{\mathsf{v}}(\chi)$ and every e-tuple $\overline{\mathsf{e}}$ with $\mathrm{vars}(\overline{\mathsf{e}}) \subseteq \mathrm{free}_{\mathsf{e}}(\chi)$:

- $\varphi = \exists^{\geq n} \overline{\mathsf{v}}\,.\,\chi$ with $\mathrm{free}_{\mathsf{v}}(\varphi) \coloneqq \mathrm{free}_{\mathsf{v}}(\chi) \setminus \mathrm{vars}(\overline{\mathsf{v}})$ and $\mathrm{free}_{\mathsf{e}}(\varphi) \coloneqq \mathrm{free}_{\mathsf{e}}(\chi)$, and $\mathrm{gd}(\varphi) \coloneqq \mathrm{gd}(\chi)$;
- $\varphi = \exists^{\geq n} \overline{\mathsf{e}}\,.\,\chi$ with $\mathrm{free}_{\mathsf{v}}(\varphi) \coloneqq \mathrm{free}_{\mathsf{v}}(\chi)$ and $\mathrm{free}_{\mathsf{e}}(\varphi) \coloneqq \mathrm{free}_{\mathsf{e}}(\chi) \setminus \mathrm{vars}(\overline{\mathsf{e}})$, and $\mathrm{gd}(\varphi) \coloneqq \mathrm{gd}(\chi) + |\mathrm{vars}(\overline{\mathsf{e}})|$. ⌟

For convenience, we let $\mathrm{free}(\varphi) \coloneqq \mathrm{free}_{\mathsf{v}}(\varphi) \cup \mathrm{free}_{\mathsf{e}}(\varphi)$ for all $\varphi \in \mathsf{GC}^k$. Formulas of GC^k are evaluated over a hypergraph $\mathcal{H}$ via interpretations $\mathcal{I} = (I_{\mathcal{H}}, \nu_{\mathsf{v}}, \nu_{\mathsf{e}})$ that consist of $\mathcal{H}$'s incidence graph $I_{\mathcal{H}}$ and assignments $\nu_{\mathsf{v}}\colon \mathsf{VAR}_{\mathsf{v}} \to \mathcal{R}(I_{\mathcal{H}})$ and $\nu_{\mathsf{e}}\colon \mathsf{VAR}_{\mathsf{e}} \to \mathcal{B}(I_{\mathcal{H}})$. The semantics of GC^k are as expected and a definition can be found in Section 6 of the full version of [27], thus we do not give one here. A *sentence* is a formula $\varphi \in \mathsf{GC}^k$ that has neither free vertex, nor free hyperedge variables, i.e., $\mathrm{free}(\varphi) = \varnothing$. By GC^k_d we denote the fragment $\{\varphi \in \mathsf{GC}^k : \mathrm{gd}(\varphi) \leq d\}$, and we let $\mathsf{GC}_k \coloneqq \mathsf{GC}^k_k$. We write $\mathcal{G} \equiv_{\mathsf{L}} \mathcal{H}$ to denote that $\mathcal{G}$ and $\mathcal{H}$ satisfy the same sentences in the fragment $\mathsf{L} \subseteq \mathsf{GC}^k$.

For simplicity, we omit logical guards if they are empty or equal to the formula they are guarding. I.e., we may abbreviate subformulas of the form $(\top \wedge \varphi)$ or $(\varphi \wedge \varphi)$ as φ. We may also omit parentheses in the usual way. We write $\exists^{=n}(\overline{\mathsf{x}})\,.\,(\Delta \wedge \varphi)$ as shorthand for $\exists^{\geq n}(\overline{\mathsf{x}})\,.\,(\Delta \wedge \varphi) \wedge \neg\exists^{\geq n+1}(\overline{\mathsf{x}})\,.\,(\Delta \wedge \varphi)$. Clearly, these shorthands change neither the semantics, nor the free variables, nor the guard depth of a formula.

▶ **Example 5.2.** The sentence $\varphi_{\mathcal{G}} \coloneqq \psi_1 \wedge \psi_2 \wedge \psi_3$ describes $\mathcal{G}$ from Example 2.1 up to isomorphism, where

$$\chi_n \coloneqq \bigwedge_{1\leq i<j\leq n} \neg \mathtt{v}_i = \mathtt{v}_j \ \wedge\ \neg\exists^{\geq 1}(\mathtt{v}_{n+1}) \,.\, \Big(E(\mathtt{e},\mathtt{v}_{n+1}) \ \wedge\ \bigwedge_{i\in[n]} \neg \mathtt{v}_{n+1} = \mathtt{v}_i\Big)\,,$$
$$\psi_1 \coloneqq \exists^{=4}(\mathtt{e}) \,.\, \mathtt{e}=\mathtt{e}\,,$$
$$\psi_2 \coloneqq \exists^{=1}(\mathtt{e}) \,.\, \exists^{\geq 1}(\mathtt{v}_1,\mathtt{v}_2,\mathtt{v}_3) \,.\, \Big(\bigwedge_{i\in[3]} E(\mathtt{e},\mathtt{v}_i) \ \wedge\ \chi_3 \ \wedge\ \bigwedge_{i\in[3]} \exists^{=3}(\mathtt{e}) \,.\, E(\mathtt{e},v_i)\Big)\,,$$
$$\psi_3 \coloneqq \exists^{=3}(\mathtt{e}) \,.\, \exists^{\geq 1}(\mathtt{v}_1,\mathtt{v}_2) \,.\, \Big(\bigwedge_{i\in[2]} E(\mathtt{e},\mathtt{v}_i) \ \wedge\ \chi_2 \ \wedge\ \bigwedge_{i\in[3]} \exists^{=3}(\mathtt{e}) \,.\, E(\mathtt{e},v_i)\Big)\,.$$

It is easily verified that $\varphi_{\mathcal{G}} \in \mathsf{GC}^1_2$. χ_n is a helper formula, describing that there are precisely n vertices $\mathtt{v}_1,\dots,\mathtt{v}_n$ in the hyperedge $\mathtt{e}$. ψ_1 describes that there are precisely four hyperedges, ψ_2 describes that precisely one hyperedge contains precisely three vertices, each being contained in precisely 3 hyperedges. Finally, ψ_3 describes that there are exactly 3 hyperedges containing precisely 2 vertices, each being contained in precisely 3 hyperedges. It is not hard to see that, in total, this describes $\mathcal{G}$ up to isomorphism.

Scheidt and Schweikardt [27] prove their result only for the following restricted variant of GC^k, called RGC^k. They mention in the conclusion, that RGC^k and GC^k are equivalent and show in the full version of the paper (Theorem 7.2) how a formula in GC^k can be translated into one in RGC^k. We still need RGC^k since it is used in the formulation and the proof of the two core lemmata of [27] that we want to borrow.

▶ **Definition 5.3** ([27])**.** The restriction RGC^k is inductively defined as follows:

Atomic Formulas. $(\Delta_g \wedge \varphi)$ is in RGC^k for all atomic formulas $\varphi \in \mathsf{GC}^k$ and all guard functions for φ, i.e., all $g\colon \mathbb{N}_{\geq 1} \rightharpoonup [k]$ with $\mathrm{dom}(g) = \{i : \mathtt{v}_i \in \mathrm{free}_{\mathtt{v}}(\varphi)\}$.

Inductive Rules.

- For every formula $(\Delta_g \wedge \varphi) \in \mathsf{RGC}^k$, the formula $(\Delta_g \wedge \neg\varphi)$ is also in RGC^k.
- For $i \in [2]$ and formulas $(\Delta_{g_i} \wedge \psi_i) \in \mathsf{RGC}^k$, the formula $(\Delta_{(g_1 \cup g_2)} \wedge (\psi_1 \wedge \psi_2))$ is in RGC^k, if g_1 and g_2 are compatible.

Let $n \in \mathbb{N}_{\geq 1}$, $(\Delta_g \wedge \varphi) \in \mathsf{RGC}^k$.

- For every $\mathtt{v}$-tuple $\overline{\mathtt{v}}$ with $\mathrm{vars}(\overline{\mathtt{v}}) \subseteq \mathrm{free}_{\mathtt{v}}(\varphi)$ and index set S, the formula $(\Delta_{\tilde{g}} \wedge \chi)$ is in RGC^k, where

  $$\chi \coloneqq \exists^{\geq n}\overline{\mathtt{v}} \,.\, (\Delta_g \wedge \varphi) \quad \text{and } \tilde{g} \text{ is the restriction of } g \text{ to } \mathrm{dom}(g) \setminus S.$$

- For every $\mathtt{e}$-tuple $\overline{\mathtt{e}}$ with $\mathrm{vars}(\overline{\mathtt{e}}) \subseteq \mathrm{free}_{\mathtt{e}}(\Delta_g \wedge \varphi)$ and index set S, the formula

  $$(\Delta_{\tilde{g}} \wedge \exists^{\geq n}\overline{\mathtt{e}} \,.\, (\Delta_g \wedge \varphi))$$

  is in RGC^k, if $\mathrm{dom}(\tilde{g}) = \mathrm{dom}(g)$ and all $i \in \mathrm{dom}(g)$ satisfy

  $$\tilde{g}(i) = g(i) \quad \text{or} \quad \tilde{g}(i) \in S \quad \text{or} \quad \tilde{g}(i) \notin \mathrm{img}(g). \tag{1}$$

Intuitively, formulas in RGC^k always carry the information, which hyperedge variable currently guards which vertex variable and the logical guards are in a certain sense "consistent" (1) along the syntax tree. ⌟

A simple inspection of the inductive proof for Theorem 7.2 in the full version of [27] shows that the guard depth is unaffected by the translation, thus it gives us the following refined result.

▶ **Lemma 5.4.** *For every formula $\varphi \in \mathsf{GC}^k$ and every guard function g for φ, there exists a formula $(\Delta_g \wedge \varphi_g) \in \mathsf{RGC}^k$ such that*

1. $(\Delta_g \wedge \varphi) \equiv (\Delta_g \wedge \varphi_g)$,
2. $\mathrm{free}(\varphi) = \mathrm{free}(\varphi_g)$, *and* $\mathrm{gd}(\varphi) = \mathrm{gd}(\varphi_g)$.

6 Main Result

We are now ready to plug everything together, which yields our main result.

▶ **Theorem 6.1.** *Let $\mathcal{G}$ and $\mathcal{H}$ be hypergraphs and let $k \in \mathbb{N}_{\geq 1}$.*

$$\begin{aligned} \mathcal{G} \equiv_{\mathsf{GC}_k} \mathcal{H} &\iff \mathrm{Hom}(\mathsf{ISHD}_k, I_{\mathcal{G}}) = \mathrm{Hom}(\mathsf{ISHD}_k, I_{\mathcal{H}}) \\ &\iff \mathrm{Hom}(\mathsf{SHD}_k, \mathcal{G}) = \mathrm{Hom}(\mathsf{SHD}_k, \mathcal{H}). \end{aligned}$$

We use the fact that the proofs for the core Lemmata 8.1 and 8.2 in the work by Scheidt and Schweikardt [27] actually give us the following refined results. This is easy to see on inspection of the original proofs (consult Appendix E in the full version of [27]), since there is a one-to-one correspondence between the blue label i and the hyperedge variable e_i in the proofs for both lemmas: whenever a blue label i is removed, the corresponding variable e_i is quantified and vice-versa.

For a k-labeled incidence graph L of the form (I, r, b, g), we let $\mathcal{I}_L := (I, \nu_{\mathsf{v}}, \nu_{\mathsf{e}})$ be defined by $\nu_{\mathsf{v}}(\mathsf{v}_i) := r(i)$ for all $i \in \mathrm{dom}(r)$ and $\nu_{\mathsf{e}}(\mathsf{e}_j) := b(j)$ for all $j \in \mathrm{dom}(b)$.

▶ **Lemma 6.2** (implicit in [27])**.** *Let $L = (I, r, b, g) \in \mathsf{GLI}_k^i$. For every $m \in \mathbb{N}$ there is a formula $\varphi_{L,m}$ with $(\Delta_g \wedge \varphi_{L,m}) \in \mathit{RGC}^k$, $\mathrm{free}_{\mathsf{v}}(\Delta_g \wedge \varphi_{L,m}) = \{\mathsf{v}_i : i \in \mathrm{dom}(r)\}$, $\mathrm{free}_{\mathsf{e}}(\Delta_g \wedge \varphi_{L,m}) = \{\mathsf{e}_j : j \in \mathrm{dom}(b)\}$, and $\mathrm{gd}(\varphi_{L,m}) \leq i$, such that for every k-labeled incidence graph L' with $\mathrm{dom}(b_{L'}) \supseteq \mathrm{dom}(b)$, $\mathrm{dom}(r_{L'}) \supseteq \mathrm{dom}(r)$, and with real guards w.r.t. g we have: $\mathcal{I}_{L'} \models \Delta_g$, and $\mathrm{hom}(L, L') = m \iff \mathcal{I}_{L'} \models \varphi_{L,m}$.*

▶ **Lemma 6.3** (implicit in [27])**.** *Let $\chi := (\Delta_g \wedge \psi) \in \mathit{RGC}^k$ with $\mathrm{gd}(\chi) = \ell$ and let $m, d \in \mathbb{N}$ with $m \geq 1$. There exists a linear combination $Q := \sum_{i \in [q]} \alpha_i L_i$, and sets $\mathrm{dr}_Q = \{i : \mathsf{v}_i \in \mathrm{free}_{\mathsf{v}}(\chi)\}$ and $\mathrm{db}_Q = \{i : \mathsf{e}_i \in \mathrm{free}_{\mathsf{e}}(\chi)\}$, where for all $i \in [q]$:*

$$\alpha_i \in \mathbb{R}, \quad L_i \in \mathsf{GLI}_k^{\ell}, \quad g_i = g, \quad \mathrm{dom}(b_i) = \mathrm{db}_Q, \quad \text{and} \quad \mathrm{dom}(r_i) = \mathrm{dr}_Q;$$

such that for all k-labeled incidence graphs L' with $|\mathcal{B}(I')| = m$, $\max\{|\,\beta(e)| : e \in \mathcal{B}(I')\} \leq d$ and $\mathrm{dom}(b') \supseteq \mathrm{db}_Q$, $\mathrm{dom}(r') \supseteq \mathrm{dr}_Q$, $g' \supseteq g$, and with real guards w.r.t. g we have: $\mathcal{I}_{L'} \models \Delta_g$, and

$$\sum_{i \in [q]} \alpha_i \cdot \mathrm{hom}(L_i, L') = \begin{cases} 1, & \text{if } \mathcal{I}_{L'} \models \chi \\ 0, & \text{if } \mathcal{I}_{L'} \not\models \chi. \end{cases}$$

The proof of Theorem 6.1 works the same way as the one for Theorem 6.1 in [27, Section 8]: the second biimplication is provided by Theorem 2.7 and Proposition 2.8. The first biimplication is shown via contraposition, where the contraposition of the forward direction uses Lemma 6.2 and the one for the backward direction uses Lemma 6.3.

Proof of Theorem 6.1. Let $I = I_{\mathcal{G}}$ and $J = I_{\mathcal{H}}$. If $|\mathcal{B}(I)| \neq |\mathcal{B}(J)|$ then $\mathrm{hom}(I', I) \neq \mathrm{hom}(I', J)$ for the incidence graph $I' \in \mathsf{ISHD}_1$ that consists of a single blue vertex and no red vertices. Similarly, I and J are distinguished by a suitable GC_1-sentence of the form $\exists^{\geq n} \mathsf{e}\,.\,(\mathsf{e} = \mathsf{e})$. If $|\mathcal{B}(I)| = |\mathcal{B}(J)|$, consider their corresponding label-free k-labeled incidence graphs $L_I = (I, \varnothing, \varnothing, \varnothing)$ and $L_J = (J, \varnothing, \varnothing, \varnothing)$.

Assume there is an $I' \in \mathsf{ISHD}_k$ such that $\mathrm{hom}(I', I) = m_1 \neq m_2 = \mathrm{hom}(I', J)$. According to Theorem 4.1, there is a label-free $L \in \mathsf{GLI}^k$ such that $I' \cong I_L$, which means $\mathrm{hom}(L, L_I) = m_1 \neq m_2 = \mathrm{hom}(L, L_J)$. By Lemma 6.2 there exists a formula $(\top \wedge \varphi_{L,m_1}) \in \mathsf{RGC}^k$ with $\mathrm{gd}(\varphi_{L,m_1}) \leq k$ such that $\mathcal{I}_{L_I} \models (\top \wedge \varphi_{L,m_1})$ and $\mathcal{I}_{L_J} \not\models (\top \wedge \varphi_{L,m_1})$. Hence, $\mathcal{I}_{L_I} \models \varphi_{L,m_1}$ and $\mathcal{I}_{L_J} \not\models \varphi_{L,m_1}$, and since $\varphi_{L,m_1} \in \mathsf{GC}_k$, $\mathcal{G} \not\equiv_{\mathsf{GC}_k} \mathcal{H}$.

Assume there is a sentence $\varphi \in \mathsf{GC}^k$ with $\mathrm{gd}(\varphi) = k$ such that $\mathcal{I}_{L_I} \models \varphi$ and $\mathcal{I}_{L_J} \not\models \varphi$. By Lemma 5.4 there exists a formula $(\top \wedge \psi) \in \mathsf{RGC}^k$ with $\mathrm{gd}(\psi) = k$ such that $\mathcal{I}_{L_I} \models (\top \wedge \psi)$ and $\mathcal{I}_{L_J} \not\models (\top \wedge \psi)$. Let $m := |\mathcal{B}(I)| = |\mathcal{B}(J)|$ be the number of hyperedges and let $n \in \mathbb{N}$ such that $|\,\beta(e)| \leq n$ for all $e \in \mathcal{B}(I)$ and all $e \in \mathcal{B}(J)$. According to Lemma 6.3 there exists a linear combination $Q = \sum_{i\in[q]} \alpha_i L_i$ such that $\sum_{i\in[q]} \alpha_i \cdot \hom(L_i, L_I) = 1$ and $\sum_{i\in[q]} \alpha_i \cdot \hom(L_i, L_J) = 0$ and $L_i \in \mathsf{GLI}_k^k$ for all $i \in [q]$. This means there must be an $i \in [q]$ such that $\alpha_i \cdot \hom(L_i, L_I) \neq \alpha_i \cdot \hom(L_i, L_J)$, which means $\hom(L_i, L_I) \neq \hom(L_i, L_J)$. Since $\mathrm{dr}_Q = \mathrm{db}_Q = \varnothing$, L_i is label-free. According to Theorem 4.1, there exists an $I' \in \mathsf{ISHD}_k$ such that $I' \cong I_{L_i}$. Thus, $\hom(I', I) \neq \hom(I', J)$, i.e., $\mathrm{Hom}(\mathsf{ISHD}_k, I) \neq \mathrm{Hom}(\mathsf{ISHD}_k, J)$.

This finishes the proof for the first "iff". The second is provided by the combination of Theorem 2.7 and Proposition 2.8. ◀

7 Final Remarks

This paper solves one of the open questions of Scheidt and Schweikardt [27], who lift a result by Dvořák [9] from graphs to hypergraphs. Dvořák shows that homomorphism indistinguishability over the graphs of tree width at most k is equivalent to indistinguishability over first-order logic with counting quantifiers (C) and $k{+}1$ variables (C^{k+1}). Scheidt and Schweikardt show that homomorphism indistinguishability over the class GHW_k of hypergraphs of generalised hypertree width at most k is equivalent to indistinguishability over the logic GC with k guards (GC^k). Grohe [11] gave a result complementing Dvořák's: C with quantifier depth at most m (C_m) matches homomorphism indistinguishability over graphs of tree depth at most m. An obvious expectation was that the distinguishing power of GC_m would match homomorphism indistinguishability over the class HD_m of hypergraphs of hypertree depth at most m as it is defined by Adler et al. [1]. However, this expectation did not manifest in this exact way. Instead, we proved that the distinguishing power of GC_m matches homomorphism indistinguishability over hypergraphs of *strict* hypertree depth at most m, which is a (mild) restriction of hypertree depth. Combining Theorem 6.1 with the main result of [27] yields the following combined result.

▶ **Theorem 7.1.** *For all hypergraphs $\mathcal{G}$ and $\mathcal{H}$, the following equivalences hold:*

$$\mathcal{G} \equiv_{\mathsf{GC}_k} \mathcal{H} \iff \mathcal{G} \equiv_{\mathsf{SHD}_k} \mathcal{H} \iff I_{\mathcal{G}} \equiv_{\mathsf{ISHD}_k} I_{\mathcal{H}} \quad \text{and}$$
$$\mathcal{G} \equiv_{\mathsf{GC}^k} \mathcal{H} \iff \mathcal{G} \equiv_{\mathsf{GHW}_k} \mathcal{H} \iff I_{\mathcal{G}} \equiv_{\mathsf{IGHW}_k} I_{\mathcal{H}}.$$

We took this unexpected mismatch between GC_k and HD_k as an opportunity to investigate the relationship between HD_k and SHD_k. In Theorem 2.5 we showed that the strict hypertree depth of a hypergraph is at most 1 larger than its hypertree depth.

▶ **Theorem 2.5.** *For all hypergraphs $\mathcal{H}$,* $\mathrm{hd}(\mathcal{H}) \leq \mathrm{shd}(\mathcal{H}) \leq \mathrm{hd}(\mathcal{H})+1$.

To show that homomorphism counts from the class SHD_k are just as expressive as homomorphism counts from the class ISHD_k, which was necessary to prove Theorem 6.1, we used an implicit result by Böker [4], who gives a sufficient set of properties for a class $\mathfrak{C}$ of hypergraphs, such that homomorphism indistinguishability over $\mathfrak{C}$ is the same as homomorphism indistinguishability over the corresponding class $\mathfrak{C}_{\mathsf{I}}$ of incidence graphs. Since HD_k does not have these properties, Böker's result cannot be applied with respect to HD_k and IHD_k. In fact, we showed in Theorem 2.9 that homomorphism indistinguishability over HD_k is *not* the same as homomorphism indistinguishability over IHD_k and furthermore, that it is also not the same as homomorphism indistinguishability over SHD_k.

▶ **Theorem 2.9.** *For every $k \in \mathbb{N}_{\geq 1}$ there exist pairs of hypergraphs $(\mathcal{G}_k, \mathcal{H}_k)$ and $(\mathcal{G}'_k, \mathcal{H}'_k)$, such that:*

1. $\mathrm{Hom}(\mathsf{SHD}_k, \mathcal{G}_k) = \mathrm{Hom}(\mathsf{SHD}_k, \mathcal{H}_k)$, *but* $\mathrm{Hom}(\mathsf{HD}_k, \mathcal{G}_k) \neq \mathrm{Hom}(\mathsf{HD}_k, \mathcal{H}_k)$;
2. $\mathrm{Hom}(\mathsf{HD}_k, \mathcal{G}'_k) = \mathrm{Hom}(\mathsf{HD}_k, \mathcal{H}'_k)$, *but* $\mathrm{Hom}(\mathsf{IHD}_k, I_{\mathcal{G}'_k}) \neq \mathrm{Hom}(\mathsf{IHD}_k, I_{\mathcal{H}'_k})$.

Further Research. It would be very interesting to see if the result by Böker (Theorem 2.7) is tight in the sense that closure under pumping and local merging are sufficient *and required* properties. I.e., whether for every class $\mathfrak{C}$ that misses one of these properties, homomorphism counts over $\mathfrak{C}$ differ from homomorphism counts over the corresponding class $\mathfrak{C}_{\mathsf{I}}$ of incidence graphs in their distinguishing power.

As mentioned in the introduction, this work can be seen as one more step in the search of a "proper" lifting of the k-dimensional Weisfeiler-Leman algorithm to hypergraphs. Given the relationship between Weisfeiler-Leman, C and homomorphism indistinguishability on graphs [5, 7, 8, 9, 10, 11], we believe that the proper lifting should admit a similar relationship to the corresponding hypergraph parameters. Hence, we believe that the distinguishing power of such an algorithm should match homomorphism indistinguishability over the class GHW_k of hypergraphs of generalised hypertree width at most k and thus also indistinguishability by the logic GC^k. Since we believe that GC^k is the natural lifting of C^k in this setting, this paper adds to this picture: The k-dimensional Weisfeiler-Leman algorithm restricted to m iterations should have the same distinguishing power as the intersection of the classes $\mathsf{GHW}_k \cap \mathsf{SHD}_m$. Hence, the mismatch we uncovered in this work might propagate to the Weisfeiler-Leman algorithm.

References

1 Isolde Adler, Tomáš Gavenčiak, and Tereza Klimošová. Hypertree-depth and minors in hypergraphs. *Theoretical Computer Science*, 463:84–95, 2012. doi:10.1016/j.tcs.2012.09.007.

2 Jan Böker, Yijia Chen, Martin Grohe, and Gaurav Rattan. The Complexity of Homomorphism Indistinguishability. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*, volume 138 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 54:1–54:13, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2019.54.

3 Silvia Butti and Víctor Dalmau. Fractional Homomorphism, Weisfeiler-Leman Invariance, and the Sherali-Adams Hierarchy for the Constraint Satisfaction Problem. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, volume 202 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:19, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2021.27.

4 Jan Böker. Color Refinement, Homomorphisms, and Hypergraphs. In Ignas Sau and Dimitrios M. Thilikos, editors, *Graph-Theoretic Concepts in Computer Science - 45th International Workshop, WG 2019, Vall de Núria, Spain, June 19-21, 2019, Revised Papers*, volume 11789 of *Lecture Notes in Computer Science*, pages 338–350. Springer, 2019. doi:10.1007/978-3-030-30786-8_26.

5 Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992. doi:10.1007/BF01305232.

6 Bruno Courcelle. Graph grammars, monadic second-order logic and the theory of graph minors. In Neil Robertson and Paul Seymour, editors, *Graph Structure Theory, Proceedings of a AMS-IMS-SIAM Joint Summer Research Conference on Graph Minors held June 22 to July 5, 1991, at the University of Washington, Seattle, USA*, volume 147 of *Contemporary Mathematics*, pages 565–590. American Mathematical Society, 1993.

7 Anuj Dawar, Tomáš Jakl, and Luca Reggio. Lovász-type theorems and game comonads. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470609.

8 Holger Dell, Martin Grohe, and Gaurav Rattan. Lovász meets Weisfeiler and Leman. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 40:1–40:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.40.

9 Zdeněk Dvořák. On recognizing graphs by numbers of homomorphisms. *Journal of Graph Theory*, 64(4):330–342, 2010. doi:10.1002/jgt.20461.

10 Eva Fluck, Tim Seppelt, and Gian Luca Spitzer. Going deep and going wide: Counting logic and homomorphism indistinguishability over graphs of bounded treedepth and treewidth. In Aniello Murano and Alexandra Silva, editors, *32nd EACSL Annual Conference on Computer Science Logic, CSL 2024, February 19-23, 2024, Naples, Italy*, volume 288 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:17, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2024.27.

11 Martin Grohe. Counting Bounded Tree Depth Homomorphisms. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, pages 507–520, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3373718.3394739.

12 Martin Grohe. Word2vec, Node2vec, Graph2vec, X2vec: Towards a Theory of Vector Embeddings of Structured Data. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS'20, pages 1–16. ACM, 2020. doi:10.1145/3375395.3387641.

13 Martin Grohe. The Logic of Graph Neural Networks. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–17. IEEE, 2021. doi:10.1109/LICS52264.2021.9470677.

14 Martin Grohe, Kristian Kersting, Martin Mladenov, and Erkal Selman. Dimension Reduction via Colour Refinement. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 505–516, Berlin, Heidelberg, 2014. Springer. doi:10.1007/978-3-662-44777-2_42.

15 Martin Grohe, Gaurav Rattan, and Tim Seppelt. Homomorphism tensors and linear equations. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 70:1–70:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2022.70.

16 Neil Immerman and Eric Lander. Describing Graphs: A First-Order Approach to Graph Canonization. In Alan L. Selman, editor, *Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday, July 5, 1988*, pages 59–81. Springer, New York, NY, 1990. doi:10.1007/978-1-4612-4478-3_5.

17 Sandra Kiefer. The Weisfeiler-Leman Algorithm: An Exploration of Its Power. *ACM SIGLOG News*, 7(3):5–27, 2020. doi:10.1145/3436980.3436982.

18 László Lovász. Operations with structures. *Acta Mathematica Academiae Scientiarum Hungaricae*, 18(3):321–328, 1967.

19 László Lovász and Balázs Szegedy. Contractors and connectors of graph algebras. *Journal of Graph Theory*, 60(1):11–30, 2009. doi:10.1002/jgt.20343.

20 Laura Mančinska and David E. Roberson. Quantum isomorphism is equivalent to equality of homomorphism counts from planar graphs. In Sandy Irani, editor, *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 661–672. IEEE, 2020. doi:10.1109/FOCS46700.2020.00067.

21 Yoàv Montacute and Nihil Shah. The Pebble-Relation Comonad in Finite Model Theory. In Christel Baier and Dana Fisman, editors, *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, number 13 in LICS '22, pages 1–11, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3531130.3533335.

22 Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 4602–4609. AAAI Press, 2019. doi:10.1609/aaai.v33i01.33014602.

23 Daniel Neuen. Homomorphism-distinguishing closedness for graphs of bounded tree-width. *CoRR*, abs/2304.07011, 2023. doi:10.48550/arXiv.2304.07011.

24 Gaurav Rattan and Tim Seppelt. Weisfeiler-Leman and Graph Spectra. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 2268–2285. SIAM, 2023. doi:10.1137/1.9781611977554.ch87.

25 David E. Roberson. Oddomorphisms and homomorphism indistinguishability over graphs of bounded degree, 2022. doi:10.48550/arXiv.2206.10321.

26 Benjamin Scheidt. On Homomorphism Indistinguishability and Hypertree Depth, 2024. Full version. doi:10.48550/arXiv.2404.10637.

27 Benjamin Scheidt and Nicole Schweikardt. Counting Homomorphisms from Hypergraphs of Bounded Generalised Hypertree Width: A Logical Characterisation. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*, volume 272 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 79:1–79:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. Full version available at arXiv: arXiv:2303.10980 [cs.LO]. doi:10.4230/LIPIcs.MFCS.2023.79.

28 Tim Seppelt. Logical equivalences, homomorphism indistinguishability, and forbidden minors. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*, volume 272 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 82:1–82:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2023.82.

29 Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research*, 12(77):2539–2561, 2011. URL: https://www.jmlr.org/papers/volume12/shervashidze11a/shervashidze11a.pdf, doi:10.5555/1953048.2078187.

30 Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019. URL: https://openreview.net/forum?id=ryGs6iA5Km.

On the Length of Strongly Monotone Descending Chains over $\mathbb{N}^d$

Sylvain Schmitz
Université Paris Cité, CNRS, IRIF, Paris, France

Lia Schütze
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Abstract

A recent breakthrough by Künnemann, Mazowiecki, Schütze, Sinclair-Banks, and Węgrzycki (ICALP 2023) bounds the running time for the coverability problem in d-dimensional vector addition systems under unary encoding to $n^{2^{O(d)}}$, improving on Rackoff's $n^{2^{O(d \lg d)}}$ upper bound (*Theor. Comput. Sci.* 1978), and provides conditional matching lower bounds.

In this paper, we revisit Lazić and Schmitz' "ideal view" of the backward coverability algorithm (*Inform. Comput.* 2021) in the light of this breakthrough. We show that the controlled strongly monotone descending chains of downwards-closed sets over $\mathbb{N}^d$ that arise from the dual backward coverability algorithm of Lazić and Schmitz on d-dimensional unary vector addition systems also enjoy this tight $n^{2^{O(d)}}$ upper bound on their length, and that this also translates into the same bound on the running time of the backward coverability algorithm.

Furthermore, our analysis takes place in a more general setting than that of Lazić and Schmitz, which allows to show the same results and improve on the 2EXPSPACE upper bound derived by Benedikt, Duff, Sharad, and Worrell (LICS 2017) for the coverability problem in invertible affine nets.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Models of computation

Keywords and phrases Vector addition system, coverability, well-quasi-order, order ideal, affine net

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.153

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/2310.02847

1 Introduction

Well-Quasi-Orders (wqo for short) are a notion from order theory [29, 41] that has proven very effective in many areas of mathematics, logic, combinatorics, and computer science in order to establish finiteness statements. For instance, in the field of formal verification, they provide the termination arguments for the generic algorithms for *well structured transition systems* [1, 23], notably the *backward coverability algorithm* for deciding safety properties [3, 1, 23].

In full generality, one cannot extract complexity bounds from wqo-powered termination proofs. Nevertheless, in an algorithmic setting, one can "instrument" wqos by considering so-called *controlled sequences* [41, 39], and new tight complexity upper bounds for wqo-based algorithms now appear on a regular basis [40, 4, 6, 5, 26, for a few recent examples].

Those complexity upper bounds are however astronomically high, and sometimes actually way too high for the problem at hand. An emblematic illustration of this phenomenon is the backward coverability algorithm for vector addition systems (VAS), which was shown to run in double exponential time by Bozzelli and Ganty [13] based on an original analysis due to Rackoff [37]: the corresponding bounds over the wqo $\mathbb{N}^d$ are Ackermannian [20].


Descending Chains. One way pioneered by Lazić and Schmitz [32] to close such complexity gaps while retaining some of the wide applicability of wqos and well structured transition systems is to focus on the descending chains of downwards closed sets over the wqo at hand. Indeed, one of the equivalent characterisations of wqos is the *descending chain condition* [29, 41], which guarantees that those descending chains are finite.

In themselves, descending chains are no silver bullet: e.g., the controlled descending chains over $\mathbb{N}^d$ are also of Ackermannian length [32, Thm. 3.10]. Nevertheless, these chains sometimes exhibit a form of "monotonicity," which yields vastly improved upper bounds. When applied to a *dual* version of the backward coverability algorithm in well structured transition systems, this allows to recover the same double exponential time upper bound as in [13, 37] for the VAS coverability problem, along with tight upper bounds for coverability in several VAS extensions. The same framework was also the key to establishing tight bounds for coverability in ν-Petri nets [31]. As a further testimony to the versatility of the approach, Benedikt, Duff, Sharad, and Worrell use it in [7] to derive original upper bounds for problems on invertible polynomial automata and invertible affine nets, in a setting that is not strictly speaking one of well structured transition systems.

Fine-grained Bounds for VAS Coverability. The coverability problem in VAS is well-known to be EXPSPACE-complete, thanks to Rackoffs's 1978 upper bound [37] matching a 1976 lower bound by Lipton [34]. The main parameter driving this complexity is the dimension of the system: the problem is in pseudo-polynomial time in fixed dimension d; more precisely, Rackoff's analysis yields a $n^{2^{O(d \lg d)}}$ deterministic time upper bound for d-dimensional VAS encoded in unary [38], by proving the same bound on the length of a covering execution of minimal length. Here, there is a discrepancy with the $n^{2^{\Omega(d)}}$ lower bound on the length of that execution in Lipton's construction – a discrepancy that was already highlighted as an open problem in the early 1980's by Mayr and Meyer [35], and settled in the specific case of reversible systems by Koppenhagen and Mayr [28]. The upper bounds of both Bozzelli and Ganty [13] and Lazić and Schmitz [32] on the complexity of the backward coverability algorithm inherit from Rackoff's $n^{2^{O(d \lg d)}}$ bound and suffer from the same discrepancy.

This was the situation until Künnemann, Mazowiecki, Schütze, Sinclair-Banks, and Węgrzycki [30] showed an $n^{2^{O(d)}}$ upper bound on the length of minimal covering executions of unary encoded d-dimensional VAS, matching Lipton's lower bound [30, Thm. 3.3]. This directly translates into a deterministic algorithm with the same upper bound on the running time [30, Cor. 3.4]. Furthermore, assuming the exponential time hypothesis, Künnemann et al. also show that there does not exist a deterministic $n^{o(2^d)}$ time algorithm deciding coverability in unary encoded d-dimensional VAS [30, Thm. 4.2].

Thinness. The improved upper bound relies on the notion of a *thin* vector in $\mathbb{N}^d$ [30, Def. 3.6] (somewhat reminiscent of the "extractors" of Leroux [33]). The proof of [30, Thm. 3.3] works by induction on the dimension d. By splitting a covering execution of minimal length at the first non-thin configuration, Künnemann et al. obtain a prefix made of distinct thin configurations (which must then be of bounded length), and a suffix starting from a configuration with some components high enough to be disregarded, hence that can be treated as an execution in a VAS of lower dimension, on which the induction hypothesis applies.

Contributions. In this paper, we show that the improved $n^{2^{O(d)}}$ upper bound of Künnemann et al. [30] also applies to the number of iterations of the backward coverability algorithm for d-dimensional VAS encoded in unary (see Theorem 4.2). In order to do so, one could

reuse the approach of Bozzelli and Ganty [13] to lift the improved bound from the length of minimal covering executions to the running time of the backward coverability algorithm, but here we aim for the generality of the framework of [32].

Our main contribution is thus to show in Section 3 that the upper bounds on the length of strongly monotone controlled descending chains of downwards closed sets over $\mathbb{N}^d$ – which include those constructed during the running of the backward coverability algorithm for VAS – can be improved similarly (see Theorem 3.6) when focusing on a suitably generalised notion of thinness. As a byproduct, we observe that thinness is an inherent property of such chains (see Corollary 3.7), rather than an *a priori* condition that – almost magically – yields the improved bound.

We apply our results to the coverability problem in vector addition systems in Section 4.2 – thus providing as promised an alternative to applying Bozzelli and Ganty's approach to Künnemann et al.'s results – and show that the backward coverability algorithm runs in time $n^{2^{O(d)}}$ (see Corollary 4.5) and is therefore conditionally optimal by [30, Thm. 4.2].

As a further demonstration of the versatility of our results, we show in Section 4.3 how to apply them to invertible affine nets, a generalisation of vector addition systems introduced by Benedikt et al. [7], and a good showcase for our techniques. We obtain the same bounds for their coverability problem as in the case of vector addition systems (see Theorem 4.11 and Corollary 4.12), and thereby improve on the 2EXPSPACE upper bound of [7] by showing that the problem is actually EXPSPACE-complete (see Corollary 4.13). Along the way, we will see that the improved upper bounds also apply for other VAS extensions, for which Rackoff's proof scheme had been successfully adapted (see Remarks 4.4 and 4.15), namely strictly increasing affine nets [11], branching VAS [16], and alternating VAS [15].

2 Well-Quasi-Orders and Ideals

We start by introducing the necessary background on well-quasi-orders, descending chains, and order ideals.

Well-Quasi-Orders. A *quasi-order* $(X, \leq)$ comprises a set X and a transitive reflexive relation $\leq \subseteq X \times X$. For a subset $S \subseteq X$, its *downward closure* is the set of elements smaller or equal to some element in S, i.e., $\downarrow S \stackrel{\text{def}}{=} \{x \in X \mid \exists y \in S \,.\, x \leq y\}$. When $S = \{y\}$ is a singleton, we note $\downarrow y$ for this set. A subset $S \subseteq X$ is *downwards-closed* if $S = \downarrow S$. A *well-quasi-order* is a quasi-order $(X, \leq)$ such that all the *descending chains*

$$D_0 \supsetneq D_1 \supsetneq D_2 \supsetneq \cdots \tag{1}$$

of downwards-closed subsets $D_k \subseteq X$ are finite [29, 41].

Conversely, the *upward closure* of a subset $S \subseteq X$ is $\uparrow S \stackrel{\text{def}}{=} \{x \in X \mid \exists y \in S \,.\, y \leq x\}$, and S is *upwards-closed* if $S = \uparrow S$. The complement $X \setminus D$ of a downwards-closed set D is upwards-closed (and conversely), hence wqos have the *ascending chain condition* for chains $U_0 \subsetneq U_1 \subsetneq \cdots$ of upwards-closed sets: they are necessarily finite. Furthermore, any upwards-closed set U over a wqo has a *finite basis* B such that $U = \uparrow B$ [29, 41]; without loss of generality, we can take the elements of B to be minimal and mutually incomparable in U.

A well-studied wqo is $(\mathbb{N}^d, \sqsubseteq)$ the set of d-dimensional vectors of natural numbers along with the component-wise (aka product) ordering [17]; see Figure 1 for an illustration of a descending chain over $\mathbb{N}^2$, which happens to be produced by the backward coverability algorithm for a vector addition system [32, Ex. 3.6].

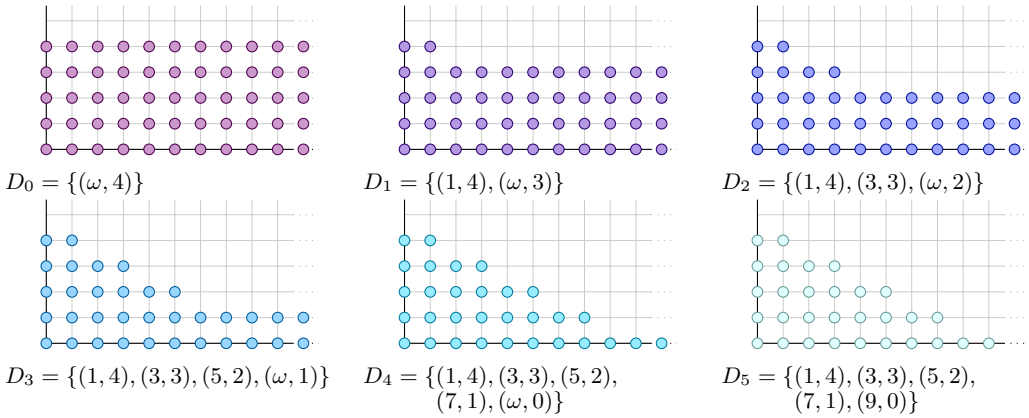


Figure 1 A descending chain $D_0 \supsetneq D_1 \supsetneq \cdots \supsetneq D_5$ over $\mathbb{N}^2$ [32, Ex. 3.6].

Order Ideals. An *order ideal* of X is a downwards-closed subset $I \subseteq X$, which is *directed*: it is non-empty, and if x, x' are two elements of I, then there exists y in I with $x \leq y$ and $x' \leq y$. Alternatively, order ideals are characterised as the *irreducible* non-empty downwards-closed sets of X: an order ideal is a non-empty downwards-closed set I with the property that, if $I \subseteq D_1 \cup D_2$ for two downwards-closed sets D_1 and D_2, then $I \subseteq D_1$ or $I \subseteq D_2$.

Over a wqo $(X, \leq)$, any downwards-closed set $D \subseteq X$ has a canonical decomposition as a finite union of order ideals $D = I_1 \cup \cdots \cup I_n$, where the I_j's are mutually incomparable for inclusion [12, 25]. We write $I \in D$ if I is an order ideal appearing in the canonical decomposition of D, i.e., if it is a maximal order ideal included in D. Then $D \subseteq D'$ if and only if, for all $I \in D$, there exists $I' \in D'$ such that $I \subseteq I'$.

Effective Representations over $\mathbb{N}^d$. Over the wqo $(\mathbb{N}^d, \sqsubseteq)$, the order ideals are exactly the sets of the form $\downarrow \boldsymbol{v} \cap \mathbb{N}^d$ where $\boldsymbol{v}$ ranges over $\mathbb{N}_\omega^d \stackrel{\text{def}}{=} (\mathbb{N} \uplus \{\omega\})^d$, where ω is a new top element [25]. From here on, we will abuse notations and identify an order ideal I of $\mathbb{N}^d$ with the vector $\boldsymbol{v}$ in $\mathbb{N}_\omega^d$ such that $I = \downarrow \boldsymbol{v} \cap \mathbb{N}^d$. See for instance the decompositions in Figure 1.

Let us introduce some notations for the sets of *infinite* and *finite components* of I, namely

$$\omega(I) \stackrel{\text{def}}{=} \{1 \leq i \leq d \mid I(i) = \omega\}\,, \qquad \text{fin}(I) \stackrel{\text{def}}{=} \{1 \leq i \leq d \mid I(i) < \omega\}\,, \tag{2}$$

along with its *dimension* and *finite dimension*, respectively defined as

$$\dim I \stackrel{\text{def}}{=} |\omega(I)|\,, \qquad \text{fdim}\, I \stackrel{\text{def}}{=} |\text{fin}(I)|\,. \tag{3}$$

Note that $\text{fin}(I) = \{1, \ldots, d\} \setminus \omega(I)$ and $\text{fdim}\, I = d - \dim I$. For instance, the order ideal $I = (\omega, 4)$ in the decomposition of D_0 in Figure 1 satisfies $\omega(I) = \{1\}$ and $\dim I = 1$.

The order ideals of $\mathbb{N}^d$, when represented as vectors in $\mathbb{N}_\omega^d$, are rather easy to manipulate [25] – and thus so are the downwards-closed subsets of $\mathbb{N}^d$ when represented as finite sets of vectors in $\mathbb{N}_\omega^d$. For instance,

- $I \subseteq I'$ (as subsets of $\mathbb{N}^d$) if and only if $I \sqsubseteq I'$ (as vectors in $\mathbb{N}_\omega^d$) – which incidentally entails $\omega(I) \subseteq \omega(I')$ and therefore $\dim I \leq \dim I'$; also note that, if $I \subseteq I'$ and $\dim I = \dim I'$, then $\omega(I) = \omega(I')$;
- the intersection of two order ideals is again an order ideal, represented by the vector $I \wedge I'$ defined by $(I \wedge I')(i) \stackrel{\text{def}}{=} \min(I(i), I'(i))$ for all $1 \leq i \leq d$;
- the complement of an order ideal I is the upwards-closed set $\bigcup_{i \in \text{fin}(I)} \uparrow\big((I(i)+1) \cdot \boldsymbol{e}_i\big)$, where $\boldsymbol{e}_i$ denotes the unit vector with "1" in coordinate i and "0" everywhere else.

Proper Ideals and Monotonicity. If $D \supsetneq D'$, then there must be an order ideal $I \in D$ such that $I \notin D'$. Coming back to a descending chain $D_0 \supsetneq D_1 \supsetneq \cdots \supsetneq D_\ell$, we then say that an order ideal I is *proper* at step k, for $0 \leq k < \ell$, if $I \in D_k$ but $I \notin D_{k+1}$; at each step $0 \leq k < \ell$, there must be at least one proper order ideal. In Figure 1, $(\omega, 4)$ is proper at step 0, and more generally $(\omega, 4-k)$ is the only proper order ideal at step $0 \leq k < 5$.

It turns out that the descending chains arising from some algorithmic procedures, including the backward coverability algorithm for VAS, enjoy additional relationships between their proper order ideals. Over $(\mathbb{N}^d, \sqsubseteq)$, we say that a descending chain $D_0 \supsetneq D_1 \supsetneq \cdots$ is

- *strongly monotone* [36, 7] if, whenever an ideal I_{k+1} is proper at some step $k+1$, then there exists I_k proper at step k such that $\dim I_{k+1} \leq \dim I_k$, and
- in particular *ω-monotone* [32] if, whenever an ideal I_{k+1} is proper at some step $k+1$, then there exists I_k proper at step k such that $\omega(I_{k+1}) \subseteq \omega(I_k)$.

The descending chain depicted in Figure 1 is ω-monotone – and thus strongly monotone – with $\omega((\omega, 4-(k+1))) \subseteq \omega((\omega, 4-k))$ for all $4 > k \geq 0$.

Controlled Sequences. While guaranteed to be finite, descending chains over a wqo can have arbitrary length. Nevertheless, their length can be bounded under additional assumptions. We define the *size* of a downwards-closed subset of $\mathbb{N}^d$ and of an order ideal of $\mathbb{N}^d$ as

$$\|D\| \stackrel{\text{def}}{=} \max_{I \in D} \|I\| \,, \qquad\qquad \|I\| \stackrel{\text{def}}{=} \max_{i \in \text{fin}(I)} I(i) \,. \tag{4}$$

In Figure 1, $\|D_0\| = \|D_1\| = \|D_2\| = 4$, $\|D_3\| = 5$, $\|D_4\| = 7$, and $\|D_5\| = 9$.

Given a *control* function $g \colon \mathbb{N} \to \mathbb{N}$, which will always be monotone (i.e., $\forall x \leq y . g(x) \leq g(y)$) and expansive (i.e., $\forall x . x \leq g(x)$) along with an *initial size* $n_0 \in \mathbb{N}$, we say that a descending chain $D_0 \supsetneq D_1 \supsetneq \cdots$ over $\mathbb{N}^d$ is (g, n_0)-*controlled* if, for all $k \geq 0$,

$$\|D_k\| \leq g^k(n_0) \tag{5}$$

where $g^k(n_0)$ is the kth iterate of g applied to n_0 [39]. In particular, $\|D_0\| \leq n_0$ initially. In Figure 1, the descending chain is $(g, 4)$-controlled for $g(x) \stackrel{\text{def}}{=} x + 1$.

3 Main Result

In this section, we establish a new bound on the length of controlled strongly monotone descending sequences. This relies on a generalisation of the notion of *thinness* from Künnemann et al. [30, Def. 3.6] (see Section 3.1), before we can apply thinness in the setting of strongly monotone descending chains and prove our main result in Section 3.2.

3.1 Thinness

Fix a control function g, an initial size n_0, and a dimension $d \geq 0$. Define inductively the bounds on sizes $(N_i)_{0 \leq i \leq d}$ and lengths $(L_i)_{0 \leq i \leq d}$ as follows

$$N_0 \stackrel{\text{def}}{=} n_0 \,, \qquad\qquad N_{i+1} \stackrel{\text{def}}{=} g^{L_i+1}(n_0) \,, \tag{6}$$

$$L_0 \stackrel{\text{def}}{=} 0 \,, \qquad\qquad L_{i+1} \stackrel{\text{def}}{=} L_i + \prod_{1 \leq j \leq i+1} (d - j + 1)(N_j + 1) \,. \tag{7}$$

Beware the abuse of notation, as the bounds above depend on (g, n_0) and d, but those will always be clear from the context.

▶ Remark 3.1 (Monotonicity of $(N_i)_{0\leq i\leq d}$ and $(L_i)_{0\leq i\leq d}$). By definition, for all $0 \leq i < j \leq d$, $0 \leq L_i < L_j$, and because g is assumed monotone expansive, $n_0 \leq N_i \leq N_j$. ⌟

The following definition generalises [30, Def. 3.6] to handle order ideals and an arbitrary control function and initial size.

▶ **Definition 3.2** (Thin order ideal). *Let (g, n_0) be a control function and initial size and $d > 0$ a dimension. An order ideal I of $\mathbb{N}^d$ is* thin *if there exists a bijection $\sigma\colon \mathrm{fin}(I) \to \{1, \ldots, \mathrm{fdim}\, I\}$ such that, for all $i \in \mathrm{fin}(I)$, $I(i) \leq N_{\sigma(i)}$.*

Observe that that, if I' is thin, $I \subseteq I'$, and $\dim I = \dim I'$, then I is thin.

▶ Remark 3.3 (Number of thin order ideals). There cannot be more than $\binom{d}{i} \cdot i! \cdot \prod_{1\leq j\leq i}(N_j+1) = \prod_{1\leq j\leq i}(d-j+1)(N_j+1)$ distinct thin order ideals of finite dimension i. As will become apparent in the proofs, this is what motivates the definition in (7). ⌟

3.2 Thinness Lemma

The crux of our result is the following lemma.

▶ **Lemma 3.4** (Thinness). *Consider a (g, n_0)-controlled strongly monotone descending chain $D_0 \supsetneq D_1 \supsetneq \cdots$ of downwards-closed subsets of $\mathbb{N}^d$. If I_ℓ is a proper order ideal at some step ℓ, then I_ℓ is thin and $\ell \leq L_{\mathrm{fdim}\, I_\ell}$.*

The proof of Lemma 3.4 proceeds by induction over the finite dimension $\mathrm{fdim}\, I_\ell = d - \dim I_\ell$. For the base case where I_ℓ has full dimension $\dim I_\ell = d$, then $I_\ell = (\omega, \ldots, \omega)$ is thin and $D_\ell = \mathbb{N}^d$ is the full space, which can only occur at step $\ell = 0 = L_0$. For the induction step, we first establish thinness with the following claim; note that, as just argued, an order ideal of dimension d is necessarily thin. We then follow with the bound on ℓ to complete the proof of Lemma 3.4.

▷ Claim 3.5. Let $0 \leq d' < d$ and assume that Lemma 3.4 holds for all proper order ideals I' of dimension $\dim I' > d'$. If I is any (not necessarily proper) order ideal of dimension $\dim I = d'$ appearing as a maximal ideal in the descending chain $D_0 \supsetneq D_1 \supsetneq \cdots$, then I is thin.

Proof of Claim 3.5. Let k be a step where I appears in the descending chain $D_0 \supsetneq D_1 \supsetneq \cdots$, i.e., $I \in D_k$, and let us write $I_k \stackrel{\text{def}}{=} I$. If $k > 0$, since $D_k \subseteq D_{k-1}$, there exists an order ideal $I_{k-1} \in D_{k-1}$ such that $I_k \subseteq I_{k-1}$. If $k = 0$, or by repeating this argument if $k > 0$, we obtain a chain of order ideals (with decreasing indices)

$$I_k \subseteq I_{k-1} \subseteq \cdots \subseteq I_0 \tag{8}$$

where $I_m \in D_m$ for all $k \geq m \geq 0$. Every order ideal in that chain must have dimension at least $\dim I_k = d'$ since they all contain I_k. Two cases arise.

1. If every order ideal in the chain (8) has dimension $\dim I_k$, then because the descending chain $D_0 \supsetneq D_1 \supsetneq \cdots$ is (g, n_0)-controlled, we have $\|I_0\| \leq n_0 = N_0$ and we know by Remark 3.1 that I_0 is thin. Since $I_k \subseteq I_0$ and $\dim I_k = \dim I_0$, I_k is also thin.
2. Otherwise there exists a first index K along the chain (8) where the dimension increases, i.e., such that $\dim I_k < \dim I_K$ and $\dim I_m = \dim I_k$ for all $k \geq m > K$. Then I_K is proper, as otherwise D_{K+1} would contain two distinct but comparable order ideals in its canonical decomposition, namely I_K and I_{K+1}: indeed, $I_{K+1} \subseteq I_K$ and $\dim I_{K+1} = \dim I_k < \dim I_K$ imply $I_{K+1} \subsetneq I_K$. By assumption, Lemma 3.4 can be applied to I_K of dimension $\dim I_K > \dim I_k = d'$, thus I_K is thin and $K \leq L_{\mathrm{fdim}\, I_K}$.

Let us now show that I_{K+1} is thin, which will also yield that I_k is thin since $I_k \subseteq I_{K+1}$ and $\dim I_k = \dim I_{K+1}$.
Since $\dim I_{K+1} < \dim I_K$, we let $f \stackrel{\text{def}}{=} \dim I_K - \dim I_{K+1} = \operatorname{fdim} I_{K+1} - \operatorname{fdim} I_K > 0$. As furthermore $I_{K+1} \subseteq I_K$, $\omega(I_{K+1}) \subsetneq \omega(I_K)$ and we let $\{i_1, \dots, i_f\} \stackrel{\text{def}}{=} \omega(I_K) \setminus \omega(I_{K+1}) = \operatorname{fin}(I_{K+1}) \setminus \operatorname{fin}(I_K)$.
Since I_K is thin, there exists a bijection $\sigma\colon \operatorname{fin}(I_K) \to \{1, \dots, \operatorname{fdim}(I_K)\}$ such that $I_K(i) \leq N_{\sigma(i)}$ for all $i \in \operatorname{fin}(I_K)$. We extend σ to a bijection $\sigma'\colon \operatorname{fin}(I_K) \uplus \{i_1, \dots, i_f\} \to \{1, \dots, \operatorname{fdim} I_K + f\}$: we let $\sigma'(i) \stackrel{\text{def}}{=} \sigma(i)$ for all $i \in \operatorname{fin}(I_K)$, and $\sigma'(i_j) \stackrel{\text{def}}{=} \operatorname{fdim} I_K + j$ for all $1 \leq j \leq f$. Let us check that σ' witnesses the thinness of I_{K+1}.

- Because $I_{K+1} \subseteq I_K$, for all those $i \in \operatorname{fin}(I_K)$, $I_{K+1}(i) \leq I_K(i) \leq N_{\sigma(i)} = N_{\sigma'(i)}$.
- Since $K + 1 \leq L_{\operatorname{fdim} I_K} + 1$ and since the descending chain $D_0 \supsetneq D_1 \supsetneq \cdots$ is (g, n_0)-controlled, we have a bound of $g^{L_{\operatorname{fdim} I_K}+1}(n_0) = N_{\operatorname{fdim} I_K + 1}$ on all the finite components of I_{K+1}, and in particular $I_{K+1}(i_j) \leq N_{\operatorname{fdim} I_K + 1}$ for all $1 \leq j \leq f$. By Remark 3.1, we conclude that $I_{K+1}(i_j) \leq N_{\operatorname{fdim} I_K + j} = N_{\sigma'(i_j)}$ for all $1 \leq j \leq f$. ◁

Proof of Lemma 3.4. We have already argued for the base case, so let us turn to the inductive step where $\dim I_\ell < d$. If $\ell > 0$ and since our descending chain is strongly monotone, we can find an order ideal $I_{\ell-1}$ proper at step $\ell - 1$ such that $\dim I_\ell \leq \dim I_{\ell-1}$. Both if $\ell = 0$ or by repeating this argument, we obtain a sequence of order ideals (with decreasing indices)

$$I_\ell, I_{\ell-1}, \dots, I_0 \tag{9}$$

where, for each $\ell > k \geq 0$, I_k is proper at step k, and $\dim I_{k+1} \leq \dim I_k$.

Let us decompose our sequence (9) by identifying the first step L where $\dim I_{L+1} < \dim I_L$; let $L \stackrel{\text{def}}{=} -1$ if this never occurs. After this step, for all $L \geq k \geq 0$, $\dim I_k > \dim I_\ell$. Within the initial segment, for $\ell \geq k > L$, the dimension $\dim I_k$ remains constant equal to $\dim I_\ell$, and the induction hypothesis allows to apply Claim 3.5 and infer that every order ideal I_k in this initial segment, and in particular I_ℓ among them, is thin.

It remains to provide a bound on ℓ. The $\ell - L$ order ideals in the initial segment are thin, and distinct since they are proper, hence by Remark 3.3,

$$\ell \leq L + \prod_{1 \leq i \leq \operatorname{fdim} I_\ell} (d - i + 1)(N_i + 1)\,. \tag{10}$$

If $L \geq 0$ we can apply the induction hypothesis to the proper order ideal I_L of finite dimension $\operatorname{fdim} I_L < \operatorname{fdim} I_\ell$ along with Remark 3.1 to yield $L \leq L_{\operatorname{fdim} I_L} \leq L_{\operatorname{fdim} I_\ell - 1}$ and therefore

$$\ell \leq L_{\operatorname{fdim} I_\ell - 1} + \prod_{1 \leq i \leq \operatorname{fdim} I_\ell} (d - i + 1)(N_i + 1) = L_{\operatorname{fdim} I_\ell}\,. \tag{11}$$

If $L = -1$ then (11) also holds since $L_{\operatorname{fdim} I_\ell - 1} \geq 0 > L$ in (10). ◀

We deduce a general combinatorial statement on the length of controlled strongly monotone descending chains, that generalises and refines [32, Thm. 4.4] thanks to thinness.

▶ **Theorem 3.6** (Length function for strongly monotone descending chains). *Consider a (g, n_0)-controlled strongly monotone descending chain $D_0 \supsetneq \cdots \supsetneq D_\ell$ of downwards-closed subsets of $\mathbb{N}^d$. Then $\ell \leq L_d + 1$.*

Proof. In such a descending chain, either $\ell = 0 \leq L_d + 1$, or $\ell > 0$ and there must be an order ideal I proper at step $\ell - 1$, and I has finite dimension at most d. By Lemma 3.4 and Remark 3.1, $\ell - 1 \leq L_{\operatorname{fdim} I} \leq L_d$ in that case. ◀

3.3 Thin Order Ideals and Filters

Let us conclude this section with some consequences of Lemma 3.4 and Claim 3.5. Whereas thinness was posited *a priori* in the proof of Künnemann et al. [30, Thm. 3.3] and then shown to indeed allow a suitable decomposition of minimal covering executions and to eventually prove their result, here in the descending chain setting it is an inherent property of all the order ideals appearing in the chain, thereby providing a "natural" explanation for thinness.

▶ **Corollary 3.7.** *Consider a (g, n_0)-controlled strongly monotone descending chain $D_0 \supsetneq D_1 \supsetneq \cdots$ of downwards-closed subsets of $\mathbb{N}^d$. Then every order ideal appearing in the chain is thin.*

Corollary 3.7 also entails a form of thinness of the minimal configurations in the complement of the downwards-closed sets D_k. Recall that such a complement is the upward-closure of a finite basis $B_k \stackrel{\text{def}}{=} \min_{\sqsubseteq} \mathbb{N}^d \setminus D_k$. Each element $\boldsymbol{v} \in B_k$ is a vector defining a so-called *(principal) order filter* $\uparrow\boldsymbol{v}$ of $\mathbb{N}^d$. Let us call a vector $\boldsymbol{v} \in \mathbb{N}^d$ *nearly thin* if there exists a permutation $\sigma\colon \{1, \ldots, d\} \to \{1, \ldots, d\}$ such that, for all $1 \leq i \leq d$, $\boldsymbol{v}(i) \leq N_{\sigma(i)} + 1$. We can relate thin order ideals with nearly thin order filters, which by Corollary 3.7 applies to every vector $\boldsymbol{v} \in \bigcup_k B_k$ (see the full version for a proof).

▶ **Proposition 3.8.** *If every order ideal in the canonical decomposition of a downwards-closed set $D \subseteq \mathbb{N}^d$ is thin, then each $\boldsymbol{v} \in \min_{\sqsubseteq} \mathbb{N}^d \setminus D$ is nearly thin.*

4 Applications

We describe two applications of Theorem 3.6 in this section. The first application in Section 4.2 is to the coverability problem in vector addition systems, and relies on the analysis of the backward coverability algorithm done in [32]. Thus we can indeed recover the improved upper bound of Künnemann et al. [30] for the coverability problem in the more general setting of descending chains, and show that the backward coverability algorithm achieves this $n^{2^{O(d)}}$ upper bound (see Corollary 4.5).

The second application in Section 4.3 focuses on the coverability problem in invertible affine nets, a class introduced by Benedikt et al. [7], who analysed the complexity of the problem through a reduction to zeroness in invertible polynomial automata. We give a direct analysis of the complexity of the backward coverability algorithm, which follows the same lines as in the VAS case, and allows to improve on the 2EXPSPACE upper bound shown in [7] for the problem, by showing that it is actually EXPSPACE-complete (see Corollary 4.13). This application additionally illustrates the usefulness of considering strongly monotone descending chains rather than the ω-monotone ones, as the descending chains constructed by the backward algorithm for invertible affine nets are in general not ω-monotone.

As both applications take place in the framework of well-structured transition systems [1, 23], we start with a quick refresher on this framework, the backward coverability algorithm, and its dual view using downwards-closed sets [32] in the upcoming Section 4.1.

4.1 Coverability in Well-Structured Transition Systems

Well-structured transition systems (WSTS) form an abstract family of computational models where the set of configurations is equipped with a well-quasi-ordering "compatible" with the computation steps. This wqo ensures the termination of generic algorithms checking some important behavioural properties like coverability and termination. While the idea can be traced back to the 1980's [21], this framework has been especially popularised through two landmark surveys [1, 23] that emphasised its wide applicability, and new WSTS models keep being invented in multiple areas to this day.

4.1.1 Well-Structured Transition Systems

A *well-structured transition system* (WSTS) [1, 23] is a triple $(X, \to, \leq)$ where X is a set of configurations, $\to \subseteq X \times X$ is a transition relation, and $(X, \leq)$ is a wqo with the following *compatibility* condition: if $x \leq x'$ and $x \to y$, then there exists $y' \geq y$ with $x' \to y'$.

The coverability problem below corresponds to the verification of safety properties, i.e., to checking that no bad configuration can ever be reached from a given initial configuration $s \in X$. Here we are given an error configuration $t \in X$, and we assume that any configuration larger than t is also an error.

▶ **Problem** (Coverability in well-structured transition systems).
input a well-structured transition system $(X, \to, \leq)$ and two configurations s and t in X
question does s *cover* t, i.e., does there exist $t' \in X$ such that $s \to^* t' \geq t$?

4.1.2 The Backward Coverability Algorithm

The first published version of this algorithm seems to date back to [3], where it was used to show the decidability of coverability in vector addition systems extended with reset capabilities, before it was rediscovered and generalised to well-structured transition systems [1].

The Algorithm. Given an instance of the coverability problem, the *backward coverability algorithm* [3, 1, 23] computes (a finite basis for) the upwards-closed set

$$U_* \stackrel{\text{def}}{=} \{x \in X \mid \exists t' \geq t \,.\, x \to^* t'\} \tag{12}$$

of all the configurations that cover t, and then checks whether $s \in U_*$.

The set U_* itself is computed by letting

$$U_0 \stackrel{\text{def}}{=} \uparrow t \,, \qquad\qquad U_{k+1} \stackrel{\text{def}}{=} U_k \cup \mathrm{Pre}_\exists(U_k) \,, \tag{13}$$

where, for a set $S \subseteq X$, $\mathrm{Pre}_\exists(S) \stackrel{\text{def}}{=} \{x \in X \mid \exists y \in S \,.\, x \to y\}$. Then $U_k = \{x \in X \mid \exists t' \geq t \,.\, x \to^{\leq k} t'\}$ is the set of configurations that can cover t in at most k steps. Equation (13) defines a chain $U_0 \subseteq U_1 \subseteq \cdots$ of upwards-closed subsets of X. Furthermore, if $U_\ell = U_{\ell+1}$ at some step, then we have reached stabilisation: $U_\ell = U_{\ell+k} = U_*$ for all k. Thus we focus in this algorithm on ascending chains $U_0 \subsetneq U_1 \subsetneq \cdots$, which are finite thanks to the ascending chain condition of the wqo $(X, \leq)$. In order to turn (13) into an actual algorithm, one needs to make some effectiveness assumptions on $(X, \to, \leq)$, typically that $\leq$ is decidable and a finite basis for $\mathrm{Pre}_\exists(\uparrow x)$ can be computed for all $x \in X$ [23, Prop. 3.5].

A Dual View. Lazić and Schmitz [32] take a dual view of the algorithm and define from (13) a descending chain $D_0 \supsetneq D_1 \supsetneq \cdots$ of the same length where

$$D_k \stackrel{\text{def}}{=} X \setminus U_k \tag{14}$$

for each k; this stops with $D_* = X \setminus U_*$ the set of configurations that do *not* cover t. The entire computation in (13) can be recast in this dual view, by setting

$$D_0 \stackrel{\text{def}}{=} X \setminus \uparrow t \,, \qquad\qquad D_{k+1} \stackrel{\text{def}}{=} D_k \cap \mathrm{Pre}_\forall(D_k) \,, \tag{15}$$

where, for a set $S \subseteq X$, $\mathrm{Pre}_\forall(S) \stackrel{\text{def}}{=} \{x \in X \mid \forall y \in X.(x \to y \implies y \in S)\} = X \setminus (\mathrm{Pre}_\exists(X \setminus S))$. Under some effectiveness assumptions, in particular for manipulating ideal representations over X, this can be turned into an actual algorithm [32, Sec. 3.1].

4.2 Coverability in Vector Addition Systems

Vector addition systems are a well-established model for simple concurrent processes [27] equivalent to Petri nets, with far-reaching connections to many topics in theoretical computer science. In particular, their coverability problem, which essentially captures safety checking, has been thoroughly investigated from both a theoretical [27, 34, 37, 13, 32, 30] and a more practical [19, 8, 24, 10] standpoint.

4.2.1 Vector Addition Systems

A d-dimensional *vector addition system* (VAS) [27] is a finite set $\boldsymbol{A}$ of vectors in $\mathbb{Z}^d$. It defines a well-structured transition system $(\mathbb{N}^d, \to_{\boldsymbol{A}}, \sqsubseteq)$ with $\mathbb{N}^d$ as set of configurations and transitions $\boldsymbol{u} \to_{\boldsymbol{A}} \boldsymbol{u}+\boldsymbol{a}$ for all $\boldsymbol{u}$ in $\mathbb{N}^d$ and $\boldsymbol{a}$ in $\boldsymbol{A}$ such that $\boldsymbol{u}+\boldsymbol{a}$ is in $\mathbb{N}^d$. We work with a unary encoding, and let $\|\boldsymbol{u}\| \stackrel{\text{def}}{=} \max_{1\le i\le d} |\boldsymbol{u}(i)|$ and $\|\boldsymbol{A}\| \stackrel{\text{def}}{=} \max_{\boldsymbol{a}\in\boldsymbol{A}} \|\boldsymbol{a}\|$ for all $\boldsymbol{u} \in \mathbb{Z}^d$ and $\boldsymbol{A} \subseteq \mathbb{Z}^d$ finite.

The coverability problem in vector addition systems was first shown decidable in 1969 by Karp and Miller [27], before being proven EXPSPACE-complete when d is part of the input by Lipton [34] and Rackoff [37]. Note that the problem parameterised by d is trivial for $d = 1$ (a target $\boldsymbol{t}$ is coverable if and only if $\boldsymbol{s} \ge \boldsymbol{t}$ or there exists $\boldsymbol{a} \in \boldsymbol{A}$ with $\boldsymbol{a} > 0$), hence we will assume $d \ge 2$.

4.2.2 Complexity Upper Bounds

The dual backward coverability algorithm of Section 4.1.2 is straightforward to instantiate in the case of a vector addition system. Figure 1 displays the computed descending chain for the 2-dimensional VAS $\boldsymbol{A}_{\div 2} \stackrel{\text{def}}{=} \{(-2, 1)\}$ and target configuration $\boldsymbol{t} \stackrel{\text{def}}{=} (0, 5)$ [32, Ex. 3.6].

▶ **Fact 4.1** ([32, claims 3.9 and 4.3])**.** The descending chain $D_0 \supsetneq D_1 \supsetneq \cdots$ defined by equations (13–15) for a d-dimensional VAS $\boldsymbol{A}$ and a target vector $\boldsymbol{t}$ is (g, n_0)-controlled for $g(x) \stackrel{\text{def}}{=} x + \|\boldsymbol{A}\|$ and $n_0 \stackrel{\text{def}}{=} \|\boldsymbol{t}\|$, and is ω-monotone.

The length of the descending chain defined by equations (13–15) is the main source of complexity for the whole backward coverability algorithm, and we can apply our own Theorem 3.6 instead of [32, Thm. 4.4] in order to prove the following bound on this length, where the combinatorics are somewhat similar to those of [30, Lem. 3.5].

▶ **Theorem 4.2.** *The backward coverability algorithm terminates after at most $n^{2^{O(d)}}$ iterations on a d-dimensional VAS encoded in unary.*

Proof. Let n be the size of the input to the coverability problem; we assume in the following that $n, d \ge 2$. By Fact 4.1 and due to the unary encoding, the descending chain $D_0 \supsetneq D_1 \supsetneq \cdots \supsetneq D_\ell = D_*$ is (g, n_0)-controlled for $g(x) \stackrel{\text{def}}{=} x + n$ and $n_0 \stackrel{\text{def}}{=} n$, and is ω-monotone and thus strongly monotone. By Theorem 3.6, $\ell \le L_d + 1$. Let us bound this value.

▷ Claim 4.3. Let $g(x) \stackrel{\text{def}}{=} x + n$ and $n_0 \stackrel{\text{def}}{=} n$. Then, for all $i \le d$,

$$N_{i+1} = n \cdot (L_i + 2)\,, \qquad\qquad L_i + 4 \le n^{3^i \cdot (\lg d + 1)}\,.$$

Proof of Claim 4.3. In the case of N_{i+1}, by the definition of N_{i+1} in (6), $N_{i+1} = g^{L_i+1}(n_0) = n + (L_i + 1) \cdot n = n \cdot (L_i + 2)$ as desired.

Regarding L_i, we proceed by induction over i. For the base case $i = 0$, $L_0 + 4 = 4 \leq n^{3^0 \cdot (\lg d + 1)}$ since we assumed $n, d \geq 2$. For the induction step, by the definition of L_{i+1} in (7)

$$\begin{aligned} L_{i+1} + 4 &= L_i + 4 + \prod_{0 \leq j \leq i} (d - j)(N_{j+1} + 1) \\ &\leq L_i + 4 + \prod_{0 \leq j \leq i} (d - j) \cdot n \cdot (L_j + 3) \\ &\leq 2 \cdot (dn)^{i+1} \cdot \prod_{0 \leq j \leq i} (L_j + 3) \,. \end{aligned}$$

Here, since $n \geq 2$,

$$2 \cdot (dn)^{i+1} \leq n^{(i+1)(\lg d + 1) + 1}$$

and by induction hypothesis for $j \leq i$

$$\prod_{0 \leq j \leq i} (L_j + 3) \leq n^{\sum_{0 \leq j \leq i} 3^j (\lg d + 1)} \,.$$

Thus, it only remains to see that, since $i > 0$,

$$\begin{aligned} 3^{i+1} \cdot (\lg d + 1) &= (1 + 2 \cdot \sum_{0 \leq j \leq i} 3^j) \cdot (\lg d + 1) \\ &\geq (1 + 3^0 + 3^i) \cdot (\lg d + 1) + \sum_{0 \leq j \leq i} 3^j \cdot (\lg d + 1) \\ &\geq (i + 1) \cdot (\lg d + 1) + 1 + \sum_{0 \leq j \leq i} 3^j \cdot (\lg d + 1) \,. \end{aligned}$$

◁

Thus $L_d + 1 \leq n^{3^d \cdot (\lg d + 1)}$ by Claim 4.3, and this is in $n^{2^{O(d)}}$. ◀

▶ Remark 4.4 (Branching or alternating vector addition systems). The improved upper bound parameterised by the dimension d in Theorem 4.2 also applies to some extensions of vector addition systems, for which Lazić and Schmitz [32] have shown that the backward coverability algorithm was constructing an ω-monotone descending chain controlled as in Fact 4.1, namely

- in [32, claims 6.7 and 6.8] for bottom-up coverability in branching vector addition systems (BVAS) – which is 2EXP-complete [16] – , and
- in [32, claims 5.4 and 5.5] for top-down coverability in alternating vector addition systems (AVAS) – which is 2EXP-complete as well [15]. ⌟

Recall that U_ℓ is the set of configurations that can cover the target $\boldsymbol{t}$ in at most ℓ steps, hence Theorem 4.2 provides an alternative proof for [30, Thm. 3.3]: if there exists a covering execution, then there is one of length in $n^{2^{O(d)}}$, from which an algorithm in $n^{2^{O(d)}}$ follows by [30, Thm. 3.2]. Regarding the optimality of Theorem 4.2, recall that Lipton [34] shows an $n^{2^{\Omega(d)}}$ lower bound on the length of a minimal covering execution, which translates into the same lower bound on the number ℓ of iterations of the backward coverability algorithm [13, Cor. 2]. Finally, this also yields an improved upper bound on the complexity of the (original) backward coverability algorithm. Here, we can rely on the analysis performed by Bozzelli and Ganty [13, Sec. 3] and simply replace Rackoff's $n^{2^{O(d \lg d)}}$ bound on the length of minimal covering executions by the bound from Theorem 4.2.

▶ **Corollary 4.5.** *The backward coverability algorithm runs in time $n^{2^{O(d)}}$ on d-dimensional VAS encoded in unary.*

Proof. Let n be the size of the input to the coverability problem and $U_0 \subsetneq U_1 \subsetneq \cdots \subsetneq U_\ell = U_*$ be the ascending chain constructed by the backward coverability according to (13). By Theorem 4.2, ℓ is in $n^{2^{O(d)}}$.

Let $B_k \stackrel{\text{def}}{=} \min_{\sqsubseteq} U_k$ be the minimal basis at each step k. The algorithm computes B_{k+1} from B_k as per (13) by computing $\min_{\sqsubseteq} \mathrm{Pre}_\exists(\uparrow \boldsymbol{v})$ for each $\boldsymbol{v} \in B_k$, adding the elements of B_k, and removing any non-minimal vector. Thus each step can be performed in time polynomial in n, d, and the number of vectors in B_k. Here, Bozzelli and Ganty's analysis in [13, Sec. 3] shows that $\|\boldsymbol{v}'\| \leq g(\|\boldsymbol{v}\|)$ for all $\boldsymbol{v}' \in \min_{\sqsubseteq} \mathrm{Pre}_\exists(\uparrow \boldsymbol{v})$, yielding a bound of $|B_k| \leq (g^k(n)+1)^d \leq ((\ell+1)\cdot n + 1)^d$, which is still in $n^{2^{O(d)}}$.

We can do slightly better. By Corollary 3.7, all the ideals in the canonical decomposition of $D_k \stackrel{\text{def}}{=} \mathbb{N}^d \setminus U_k$ are thin, and in turn Proposition 3.8 shows that all the vectors in B_k are nearly thin. Accordingly, let us denote by $\mathrm{Fil}^{\mathsf{thin}+1}(\mathbb{N}^d)$ the set of order filters $\uparrow \boldsymbol{v}$ such that $\boldsymbol{v}$ is nearly thin. Then $|B_k| \leq |\mathrm{Fil}^{\mathsf{thin}+1}(\mathbb{N}^d)|$, and the latter is in $n^{2^{O(d)}}$:

$$
\begin{aligned}
|\mathrm{Fil}^{\mathsf{thin}+1}(\mathbb{N}^d)| &\leq d! \cdot \prod_{1 \leq i \leq d} (N_i + 2) \\
&\leq d! \cdot n^d \cdot \prod_{0 \leq i \leq d-1} (L_i + 4) && \text{(by Claim 4.3 on } N_i\text{)} \\
&\leq n^{2d + \sum_{0 \leq i \leq d-1} 3^i \cdot (\lg d + 1)} && \text{(because } d \leq n \text{ and by Claim 4.3 on } L_i\text{)} \\
&\leq n^{3^d \cdot (\lg d + 1)} \,. && (16)
\end{aligned}
$$

Therefore, the overall complexity of the backward coverability algorithm is polynomial in ℓ, $\max_{0 \leq k \leq \ell} |B_k|$, n, and d, which is in $n^{2^{O(d)}}$. ◀

The bounds in $n^{2^{O(d)}}$ for $\|\boldsymbol{v}\| \leq N_d + 1$ for all $\boldsymbol{v} \in \min_{\sqsubseteq} U_k$ and for $|\min_{\sqsubseteq} U_k| \leq |\mathrm{Fil}^{\mathsf{thin}+1}(\mathbb{N}^d)|$ in the previous proof also improve on the corresponding bounds in [44, Thm. 9] and [13, Thm. 2]. Recall that Künnemann et al. [30, Thm. 4.2] show that, assuming the exponential time hypothesis, there does not exist a deterministic $n^{o(2^d)}$ time algorithm deciding coverability in unary encoded d-dimensional VAS, hence the backward coverability algorithm is conditionally optimal.

4.3 Coverability in Affine Nets

Affine nets [22], also known as affine vector addition systems, are a broad generalisation of VAS and Petri nets encompassing multiple extended VAS operations designed for greater modelling power.

4.3.1 Affine Nets

A d-dimensional (well-structured) *affine net* [22] is a finite set $\mathcal{N}$ of triples $(\boldsymbol{a}, A, \boldsymbol{b}) \in \mathbb{N}^d \times \mathbb{N}^{d \times d} \times \mathbb{N}^d$. It defines a well-structured transition system $(\mathbb{N}^d, \to_{\mathcal{N}}, \sqsubseteq)$ with $\mathbb{N}^d$ as set of configurations and transitions $\boldsymbol{u} \to_{\mathcal{N}} A \cdot (\boldsymbol{u} - \boldsymbol{a}) + \boldsymbol{b}$ for all $\boldsymbol{u}$ in $\mathbb{N}^d$ and $(\boldsymbol{a}, A, \boldsymbol{b})$ in $\mathcal{N}$ such that $\boldsymbol{u} - \boldsymbol{a}$ is in $\mathbb{N}^d$. This model encompasses notably

- VAS and Petri nets when (each such) A is the identity matrix I_d,
- *reset nets* [2, 3] when A is component-wise smaller or equal to I_d,
- *transfer nets* [14] when the sum of values in every column of A is one,
- *post self-modifying nets* [43] – also known as *strongly increasing affine nets* [22, 11] – when A is component-wise larger or equal to I_d, and
- *invertible* affine nets [7] when A is invertible over the rationals, i.e., $A \in \mathsf{GL}_d(\mathbb{Q})$.

As in the case of VAS, we will work with a unary encoding, and we let $\|\mathcal{N}\| \stackrel{\text{def}}{=} \max\{\|\boldsymbol{a}\| \mid (\boldsymbol{a}, A, \boldsymbol{b}) \in \mathcal{N}\}$; note that the entries from $\boldsymbol{b}$ and A are not taken into account.

▶ **Example 4.6.** Consider the affine nets

$$\mathcal{N}_1 \stackrel{\text{def}}{=} \left\{\begin{bmatrix}2\\0\end{bmatrix}, \begin{bmatrix}1&0\\0&1\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}\right\} \quad \mathcal{N}_2 \stackrel{\text{def}}{=} \left\{\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1&1\\0&0\end{bmatrix}, \begin{bmatrix}0\\0\end{bmatrix}\right\} \quad \mathcal{N}_3 \stackrel{\text{def}}{=} \left\{\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1&1\\2&0\end{bmatrix}, \begin{bmatrix}0\\0\end{bmatrix}\right\}.$$

Then $\mathcal{N}_1$ defines the same WSTS as the 2-dimensional VAS $\boldsymbol{A}_{\div 2} = \{(-2, 1)\}$. Focusing on the effects of their transition matrices, $\mathcal{N}_2$ performs a transfer from its second component into its first component, while $\mathcal{N}_3$ sums the values of its first two components into the first one, and puts the double of its first component into its second one. ⌟

The coverability problem for reset VAS was first shown decidable in 1978 by Arnold and Latteux [3] using the backward coverability algorithm, and the same algorithm applies to all affine nets [18, 22]. Its complexity is considerable: their coverability problem has already an Ackermannian complexity in the reset or transfer cases [42, 20, 40]. In the strongly increasing case, Bonnet, Finkel, and Praveen [11, Lem. 11 and Thm. 13] show how to adapt Rackoff's original argument to derive an upper bound in $n^{2^{O(d \lg d)}}$ on the length of minimal coverability witnesses, with an EXPSPACE upper bound for the problem when d is part of the input, while in the invertible case, Benedikt et al. [7, Thm. 6] show a 2EXPSPACE upper bound.

Control. Before we turn to the case of invertible affine nets, let us show that the descending chains defined by the backward coverability algorithm for affine nets are controlled, with a control very similar to the VAS case (c.f. Fact 4.1).

▶ **Proposition 4.7.** *The descending chain $D_0 \supsetneq D_1 \supsetneq \cdots$ defined by equations (13–15) for a d-dimensional affine net $\mathcal{N}$ and a target vector $\boldsymbol{t}$ is (g, n_0)-controlled for $g(x) \stackrel{\text{def}}{=} x + \|\mathcal{N}\|$ and $n_0 \stackrel{\text{def}}{=} \|\boldsymbol{t}\|$.*

Proof. Rather than handling $\mathrm{Pre}_\forall$ computations directly, we use the fact that $\mathrm{Pre}_\forall(S) = \mathbb{N}^d \setminus (\mathrm{Pre}_\exists(\mathbb{N}^d \setminus S))$ for all $S \subseteq \mathbb{N}^d$ and the following statement on $\mathrm{Pre}_\exists$ computations.

▷ Claim 4.8. If $\boldsymbol{u}' \in \min_{\sqsubseteq} \mathrm{Pre}_\exists(\uparrow \boldsymbol{u})$, then $\|\boldsymbol{u}'\| \leq \|\boldsymbol{u}\| + \|\mathcal{N}\|$.

Proof of Claim 4.8. In such a situation, there exists a triple $(\boldsymbol{a}, A, \boldsymbol{b}) \in \mathcal{N}$ such that $\boldsymbol{u}' \sqsupseteq \boldsymbol{a}$ and $A \cdot (\boldsymbol{u}' - \boldsymbol{a}) \sqsupseteq \boldsymbol{u} - \boldsymbol{b}$. Let $\boldsymbol{y}$ be defined by $\boldsymbol{y}(i) \stackrel{\text{def}}{=} \max(\boldsymbol{u}(i), \boldsymbol{b}(i)) - \boldsymbol{b}(i)$ for all $1 \leq i \leq d$, thus of size $\|\boldsymbol{y}\| \leq \|\boldsymbol{u}\|$. Then $\boldsymbol{u}' = \boldsymbol{x} + \boldsymbol{a}$ where $\boldsymbol{x}$ is a $\sqsubseteq$-minimal solution of the system of inequalities $A\boldsymbol{x} \sqsupseteq \boldsymbol{y}$.

We are going to show that if $\boldsymbol{x}$ is an $\sqsubseteq$-minimal solution, then $\|\boldsymbol{x}\| \leq \|\boldsymbol{y}\|$. This will yield the result, as then $\|\boldsymbol{u}'\| \leq \|\boldsymbol{y}\| + \|\boldsymbol{a}\| \leq \|\boldsymbol{u}\| + \|\mathcal{N}\|$. Assume by contradiction that $\boldsymbol{x}$ is a $\sqsubseteq$-minimal solution with $\boldsymbol{x}(j) > \|\boldsymbol{y}\|$ for some $1 \leq j \leq d$. Consider $\boldsymbol{x}'$ defined by $\boldsymbol{x}'(j) \stackrel{\text{def}}{=} \|\boldsymbol{y}\|$ and $\boldsymbol{x}'(i) \stackrel{\text{def}}{=} \boldsymbol{x}(i)$ for all $i \neq j$; note that $\boldsymbol{x}' \sqsubsetneq \boldsymbol{x}$. Let us show that $\boldsymbol{x}'$ is also a solution, i.e., that $A\boldsymbol{x}' \sqsupseteq \boldsymbol{y}$: for all $1 \leq i \leq d$,

- if $A(i, j) > 0$ then $\sum_{1 \leq k \leq d} A(i, k) \cdot \boldsymbol{x}'(k) \geq \boldsymbol{x}'(j) \geq \|\boldsymbol{y}\| \geq \boldsymbol{y}(i)$, and
- otherwise $\sum_{1 \leq k \leq d} A(i, k) \cdot \boldsymbol{x}'(k) = \sum_{1 \leq k \leq d} A(i, k) \cdot \boldsymbol{x}(k) \geq \boldsymbol{y}(i)$ since $\boldsymbol{x}$ is a solution.

Thus $\boldsymbol{x}'$ is a solution, contradicting the $\sqsubseteq$-minimality of $\boldsymbol{x}$. ◁

Now, since $D_0 = \mathbb{N}^d \setminus \uparrow \boldsymbol{t}$, $\|D_0\| \leq \|\boldsymbol{t}\| - 1$ by [32, Lem. 3.8]. Regarding the control function, $D_{k+1} = D_k \cap \mathrm{Pre}_\forall(D_k)$ is such that $\|D_{k+1}\| \leq \max(\|D_k\|, \|\mathrm{Pre}_\forall(D_k)\|)$ also by [32, Lem. 3.8]. In turn, regarding $\mathrm{Pre}_\forall(D_k) = \mathbb{N}^d \setminus \mathrm{Pre}_\exists(U_k)$, the minimal elements $\boldsymbol{u}$ of $U_k = \mathbb{N}^d \setminus D_k$ have size $\|\boldsymbol{u}\| \leq \|D_k\| + 1$ still by [32, Lem. 3.8], thus the minimal elements $\boldsymbol{u}'$ of $\mathrm{Pre}_\exists(U_k)$ have size $\|\boldsymbol{u}'\| \leq \|D_k\| + 1 + \|\mathcal{N}\|$ by Claim 4.8, hence $\|\mathrm{Pre}_\forall(D_k)\| \leq \|D_k\| + \|\mathcal{N}\|$ by a last application of [32, Lem. 3.8]. ◀

4.3.2 Invertible Affine Nets

The restriction to invertible affine nets [7] is somehow orthogonal to the usual restrictions to reset/transfer/post self-modifying/... nets. For instance, in Example 4.6, the identity matrix in $\mathcal{N}_1$ is clearly invertible, and the transfer matrix in $\mathcal{N}_2$ is not. More generally, reset nets are never invertible (when they perform resets), and transfer nets are invertible exactly when their matrices are permutation matrices. Nevertheless, some more involved affine nets are invertible, like $\mathcal{N}_3$ in Example 4.6, whose matrix is invertible with inverse $\left[\begin{smallmatrix} 0 & 1/2 \\ 1 & -1/2 \end{smallmatrix}\right]$.

Strong Monotonicity. When dealing with a descending sequence of downwards-closed sets produced by the dual backward coverability algorithm for WSTS, a key observation made in [32] allows to sometimes derive monotonocity. For this, in a WSTS $(X, \to, \leq)$, define $\mathrm{Post}_\exists(S) \stackrel{\text{def}}{=} \{y \in X \mid \exists x \in S \,.\, x \to y\}$. Following [9], for two order ideals I and I', write $I \rightsquigarrow I'$ if I' appears in the canonical decomposition of $\downarrow\mathrm{Post}_\exists(I)$.

▶ **Fact 4.9** ([32, Claim 4.2]). Let $D_0 \supsetneq D_1 \supsetneq \cdots$ be a descending chain of downwards-closed sets defined by equations (13–15). If I_{k+1} is an order ideal proper at step $k+1$, then there exists an order ideal I and an order ideal I_k proper at step k such that $I_{k+1} \rightsquigarrow I \subseteq I_k$.

In the case of affine nets, and identifying order ideals I with vectors in $\mathbb{N}_\omega^d$ with $\omega + n = \omega - n = \omega \cdot n = \omega$ for all n in $\mathbb{N}$, $\downarrow\mathrm{Post}_\exists(I) = \downarrow\{A \cdot (I - \boldsymbol{a}) + \boldsymbol{b} \mid (\boldsymbol{a}, A, \boldsymbol{b}) \in \mathcal{N}, I \sqsupseteq \boldsymbol{a}\}$.

▶ **Proposition 4.10.** *The descending chain $D_0 \supsetneq D_1 \supsetneq \cdots$ defined by equations (13–15) for a d-dimensional invertible affine net $\mathcal{N}$ and a target vector $\boldsymbol{t}$ is strongly monotone.*

Proof. Let I_{k+1} be proper at step $k+1$. By Fact 4.9, there exists an order ideal I and an order ideal I_k proper at step k such that $I_{k+1} \rightsquigarrow_{\mathcal{N}} I \subseteq I_k$. Let us show that $\dim I_{k+1} \leq \dim I$; as $\dim I \leq \dim I_k$ because $I \subseteq I_k$, this will yield the result.

Since $I_{k+1} \rightsquigarrow_{\mathcal{N}} I$, there exists $(\boldsymbol{a}, A, \boldsymbol{b})$ in $\mathcal{N}$ such that $I - \boldsymbol{b} = A \cdot (I_{k+1} - \boldsymbol{a})$. For this to hold, note that for all $i \in \mathrm{fin}(I)$, the ith row of A must be such that $A(i,j) = 0$ for all $j \in \omega(I_{k+1})$. As A is invertible, those $(\mathrm{fdim}\, I)$-many rows must be linearly independent. As just argued, the jth column for each of these rows is made of zeroes whenever $j \in \omega(I_{k+1})$. Thus the remaining $(\mathrm{fdim}\, I_{k+1})$-many columns must make those $\mathrm{fdim}\, I$ rows linearly independent, hence necessarily $\mathrm{fdim}\, I_{k+1} \geq \mathrm{fdim}\, I$, i.e., $\dim I_{k+1} \leq \dim I$. ◀

Observe that the proof of Proposition 4.10 does not work for the transfer net $\mathcal{N}_2$ of Example 4.6: $\left[\begin{smallmatrix} \omega \\ \omega \end{smallmatrix}\right] \rightsquigarrow_{\mathcal{N}_2} \left[\begin{smallmatrix} \omega \\ 0 \end{smallmatrix}\right]$; this is exactly the kind of non-monotone behaviour invertibility was designed to prevent. Also observe that $\left[\begin{smallmatrix} 2 \\ \omega \end{smallmatrix}\right] \rightsquigarrow_{\mathcal{N}_3} \left[\begin{smallmatrix} \omega \\ 4 \end{smallmatrix}\right]$ in the invertible affine net $\mathcal{N}_3$, which is not an ω-monotone behaviour: this illustrates the usefulness of capturing strongly monotone descending chains, as [32, Thm. 4.4 and Cor. 4.6] do not apply.

Complexity Upper Bounds. We are now equipped to analyse the complexity of the backward coverability algorithm in invertible affine nets. Regarding the length ℓ of the chain constructed by the algorithm, by Propositions 4.7 and 4.10 we are in the same situation as in Theorem 4.2 and we can simply repeat the arguments from its proof.

▶ **Theorem 4.11.** *The backward coverability algorithm terminates after at most $n^{2^{O(d)}}$ iterations on d-dimensional invertible affine nets encoded in unary when $d \geq 2$.*

We deduce two corollaries from Theorem 4.11: one pertaining to the complexity of the backward coverability algorithm in dimension d, which mirrors Corollary 4.5, and one for the coverability problem when d is part of the input. Let us start with the backward coverability algorithm.

▶ **Corollary 4.12.** *The backward coverability algorithm runs in time $n^{2^{O(d)}}$ on d-dimensional invertible affine nets encoded in unary when $d \geq 2$.*

Proof. Theorem 4.11 shows that the length ℓ of the ascending chain $U_0 \subsetneq U_1 \subsetneq \cdots \subsetneq U_\ell = U_*$ constructed by the backward coverability algorithm is at most $L_d + 1$, which is in $n^{2^{O(d)}}$.

Let $B_k \stackrel{\text{def}}{=} \min_{\sqsubseteq} U_k$ denote the minimal basis at step k. In order to compute B_{k+1} as per (13), thanks to Claim 4.8, we could essentially argue as in the proof of Corollary 4.5, with the caveat that computing bluntly $\min_{\sqsubseteq} \mathrm{Pre}_\exists(\uparrow \boldsymbol{v})$ for each $\boldsymbol{v} \in B_k$ is dangerously similar to a linear integer programming question and will incur an additional cost.

Alternatively, recall from equation (16) that $\mathrm{Fil}^{\mathsf{thin}+1}(\mathbb{N}^d)$, the set of order filters $\uparrow \boldsymbol{v}$ such that $\boldsymbol{v}$ is nearly thin, has at most $n^{2^{O(d)}}$ elements, and that $|B_k| \leq |\mathrm{Fil}^{\mathsf{thin}+1}(\mathbb{N}^d)|$ by Corollary 3.7 and Proposition 3.8. Thus in order to compute B_{k+1} one can enumerate the nearly thin vectors $\boldsymbol{v}' \in \mathrm{Fil}^{\mathsf{thin}+1}(\mathbb{N}^d)$ and check for each $(\boldsymbol{a}, A, \boldsymbol{b}) \in \mathcal{N}$ such that $\boldsymbol{v}' \sqsupseteq \boldsymbol{a}$ whether there exists $\boldsymbol{v} \in B_k$ such that $A \cdot (\boldsymbol{v}' - \boldsymbol{a}) + \boldsymbol{b} \sqsupseteq \boldsymbol{v}$. Each such check can be performed in time polynomial in $\|\boldsymbol{v}'\| \leq N_d + 1 = n \cdot (L_{d-1} + 2) + 1$, n, d, and $|B_k| \leq |\mathrm{Fil}^{\mathsf{thin}+1}(\mathbb{N}^d)|$. Thus the entire computation can be carried out in $n^{2^{O(d)}}$. ◀

As VAS are a particular case of invertible affine nets, the upper bounds in Corollary 4.12 are optimal assuming the exponential time hypothesis by [30, Thm. 4.2].

Our last result concerns the complexity of coverability in invertible affine nets when d is part of the input. Note that the arguments leading to an algorithm working in space $O(d \lg(n \cdot \ell))$ in the VAS case [30, Thm. 3.2] – which are essentially the same as those used to derive a 2EXPSPACE upper bound for invertible affine nets in [7, Thm. 6] – do not work here, as the configurations along an execution of an affine net can grow exponentially with ℓ.

▶ **Corollary 4.13.** *The coverability problem for invertible affine nets is* EXPSPACE*-complete.*

Proof. The hardness for EXPSPACE follows from the hardness of the coverability problem for VAS [34]. Regarding the upper bound, consider the execution of the classical backward coverability algorithm as defined in equation (13) on an invertible affine net $\mathcal{N}$ with target configuration $\boldsymbol{t}$: this is an ascending chain $U_0 \subsetneq U_1 \subsetneq \cdots \subsetneq U_\ell$ where $U_\ell = U_{\ell+1} = U_*$. The following characterisation of coverability actually holds more generally in WSTS.

▷ Claim 4.14. In an affine net $\mathcal{N}$, $\boldsymbol{s}$ covers $\boldsymbol{t}$ if and only if there exists $\ell' \leq \ell$ and a sequence of configurations $\boldsymbol{t}_0, \ldots, \boldsymbol{t}_{\ell'}$, called a *coverability pseudo-witness*, satisfying

$$\boldsymbol{t}_0 \stackrel{\text{def}}{=} \boldsymbol{t}\,, \qquad \boldsymbol{t}_{k+1} \in \min_{\sqsubseteq} \mathrm{Pre}_\exists(\uparrow \boldsymbol{t}_k)\,, \qquad \boldsymbol{t}_{\ell'} \sqsubseteq \boldsymbol{s}\,. \tag{17}$$

Proof of Claim 4.14. If a coverability pseudo-witness exists, then we claim that for all $\ell' \geq k \geq 0$ there exists $\boldsymbol{s}_k \sqsupseteq \boldsymbol{t}_k$ such that $\boldsymbol{s} = \boldsymbol{s}_{\ell'} \to_{\mathcal{N}} \boldsymbol{s}_{\ell'-1} \to_{\mathcal{N}} \cdots \to_{\mathcal{N}} \boldsymbol{s}_k$, and thus in particular $\boldsymbol{s} \to^*_{\mathcal{N}} \boldsymbol{s}_0 \geq \boldsymbol{t}_0$ for $k = 0$. We can check this by induction over k. For the base case $k = \ell'$, define $\boldsymbol{s}_{\ell'} \stackrel{\text{def}}{=} \boldsymbol{s}$. For the induction step k, since $\boldsymbol{t}_{k+1} \in \mathrm{Pre}_\exists(\uparrow \boldsymbol{t}_k)$ there exists $\boldsymbol{s}'_k \sqsupseteq \boldsymbol{t}_k$ such that $\boldsymbol{t}_{k+1} \to_{\mathcal{N}} \boldsymbol{s}'_k$; by WSTS compatibility and since $\boldsymbol{s}_{k+1} \sqsupseteq \boldsymbol{t}_{k+1}$, there exists $\boldsymbol{s}_k \sqsupseteq \boldsymbol{s}'_k$ such that $\boldsymbol{s}_{k+1} \to_{\mathcal{N}} \boldsymbol{s}_k$.

Conversely, assume that $\boldsymbol{s}$ covers $\boldsymbol{t}$ in $\mathcal{N}$. Then $\boldsymbol{s} \in U_\ell$, and let $\ell' \leq \ell$ be the least index such that $\boldsymbol{s} \in U_{\ell'}$. Then either $\ell' = 0$, i.e., $\boldsymbol{s} \sqsupseteq \boldsymbol{t} = \boldsymbol{t}_0$ and we are done, or $\ell' > 0$. Because $\boldsymbol{s} \in U_{\ell'}$ there must be some $\boldsymbol{t}_{\ell'} \in \min_{\sqsubseteq} U_{\ell'}$ with $\boldsymbol{s} \sqsupseteq \boldsymbol{t}_{\ell'}$, and $\boldsymbol{t}_{\ell'} \notin U_{\ell'-1}$ as otherwise $\boldsymbol{s}$ would be in $U_{\ell'-1}$, contradicting the minimality of ℓ'. In general, if we have found a sequence $(\boldsymbol{t}_j)_{\ell' \geq j \geq k > 0}$ satisfying (17) until rank $k+1$ included and know that $\boldsymbol{t}_k \in (\min_{\sqsubseteq} U_k) \setminus U_{k-1}$, then either $k = 1$ and $\boldsymbol{t}_1 \in \min_{\sqsubseteq} \mathrm{Pre}_\exists(\uparrow \boldsymbol{t}_0)$ by definition of U_0 and U_1 in (13), or $k > 1$

and because $\boldsymbol{t}_k \notin U_{k-1}$, there exists $\boldsymbol{t}_{k-1} \in \min_{\sqsubseteq} U_{k-1}$ such that $\boldsymbol{t}_k \in \min_{\sqsubseteq} \mathrm{Pre}_\exists(\uparrow \boldsymbol{t}_{k-1})$, and $\boldsymbol{t}_{k-1} \notin U_{k-2}$ as otherwise we would have $\boldsymbol{t}_k$ in U_{k-1}. Repeating this process yields a coverability pseudo-witness. ◁

By Claim 4.14, a non-deterministic algorithm for coverability can guess and check the existence of a coverability pseudo-witness. By Theorem 4.11, such a pseudo-witness has a length $\ell' \leq \ell$ in $n^{2^{O(d)}}$. Furthermore, by Claim 4.8 the components in each $\boldsymbol{t}_k$ in such a pseudo-witness are bounded by $\|\boldsymbol{t}\| + \|\mathcal{N}\| \cdot k \leq (\ell+1) \cdot n$, which is still in $n^{2^{O(d)}}$. Thus exponential space suffices. Note that this also holds when we assume the invertible affine net to be encoded in binary, by substituting 2^n for n in the bound $n^{2^{O(d)}}$. ◀

▶ Remark 4.15 (Strictly increasing affine nets). Strictly increasing affine nets [43, 22, 11] are intuitively the affine nets devoid of any form of reset or transfer; in Example 4.6, only $\mathcal{N}_1$ is strictly increasing. All the results we have proven for invertible affine nets in this section – namely in Theorem 4.11 and Corollaries 4.12 and 4.13 – also hold for strictly increasing affine nets, because the descending chains of downwards-closed sets they generate when running the backward coverability algorithm are ω-monotone.

▷ Claim 4.16. The descending chain $D_0 \supsetneq D_1 \supsetneq \cdots$ defined by equations (13–15) for a d-dimensional strictly increasing affine net $\mathcal{N}$ and a target vector $\boldsymbol{t}$ is ω-monotone.

Proof of Claim 4.16. Let I_{k+1} be proper at step $k+1$. By Fact 4.9, there exists an order ideal I and an order ideal I_k proper at step k such that $I_{k+1} \rightsquigarrow_{\mathcal{N}} I \subseteq I_k$. Let us show that $\omega(I_{k+1}) \subseteq \omega(I)$; as $\omega(I) \subseteq \omega(I_k)$ because $I \subseteq I_k$, this will yield the result.

Since $I_{k+1} \rightsquigarrow_{\mathcal{N}} I$, there exists $(\boldsymbol{a}, A, \boldsymbol{b})$ in $\mathcal{N}$ such that $I_{k+1} \sqsupseteq \boldsymbol{a}$ and $I = A \cdot (I_{k+1} - \boldsymbol{a}) + \boldsymbol{b}$. Because $\mathcal{N}$ is strictly increasing, $A = I_d + A'$ for some matrix $A' \in \mathbb{N}^{d \times d}$, hence $I = I_{k+1} - \boldsymbol{a} + A' \cdot (I_{k+1} - \boldsymbol{a}) + \boldsymbol{b}$. Thus $I \sqsupseteq (I_{k+1} - \boldsymbol{a})$ and therefore $\omega(I) \supseteq \omega(I_{k+1})$. ◁

An EXPSPACE upper bound was already shown by Bonnet et al. [11] for the coverability problem, but the $n^{2^{O(d)}}$ bound for the problem parameterised by d is an improvement over the $n^{2^{O(d \lg d)}}$ bounds of [11, Lem. 11 and Thm. 13], and the bounds for the backward coverability algorithm are new. ⌟

References

1 Parosh A. Abdulla, Karlis Čerāns, Bengt Jonsson, and Yih-Kuen Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160(1–2):109–127, 2000. doi:10.1006/inco.1999.2843.

2 Toshiro Araki and Tadao Kasami. Some decision problems related to the reachability problem for Petri nets. *Theoretical Computer Science*, 3(1):85–104, 1976. doi:10.1016/0304-3975(76)90067-0.

3 André Arnold and Michel Latteux. Récursivité et cônes rationnels fermés par intersection. *CALCOLO*, 15(4):381–394, 1978. doi:10.1007/BF02576519.

4 A. R. Balasubramanian. Complexity of controlled bad sequences over finite sets of $\mathbb{N}^d$. In *Proceedings of LICS 2020*, pages 130–140. ACM, 2020. doi:10.1145/3373718.3394753.

5 A. R. Balasubramanian. Complexity of coverability in depth-bounded processes. In *Proceedings of CONCUR 2022*, volume 243 of *Leibniz International Proceedings in Informatics*, pages 17:1–17:19. LZI, 2022. doi:10.4230/LIPIcs.CONCUR.2022.17.

6 A. R. Balasubramanian, Timo Lang, and Revantha Ramanayake. Decidability and complexity in weakening and contraction hypersequent substructural logics. In *Proceedings of LICS 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470733.

7 Michael Benedikt, Timothy Duff, Aditya Sharad, and James Worrell. Polynomial automata: Zeroness and applications. In *Proceedings of LICS 2017*, pages 1–12. IEEE, 2017. doi:10.1109/LICS.2017.8005101.

8 Michael Blondin, Alain Finkel, Christoph Haase, and Serge Haddad. Approaching the coverability problem continuously. In *Proceedings of TACAS 2016*, volume 9636 of *Lecture Notes in Computer Science*, pages 480–496. Springer, 2016. doi:10.1007/978-3-662-49674-9_28.

9 Michael Blondin, Alain Finkel, and Pierre McKenzie. Handling infinitely branching well-structured transition systems. *Information and Computation*, 258:28–49, 2018. doi:10.1016/j.ic.2017.11.001.

10 Michael Blondin, Christoph Haase, and Philip Offtermatt. Directed reachability for infinite-state systems. In *Proceedings of TACAS 2021*, volume 12652 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2021. doi:10.1007/978-3-030-72013-1_1.

11 Rémi Bonnet, Alain Finkel, and M. Praveen. Extending the Rackoff technique to affine nets. In *Proceedings of FSTTCS 2012*, volume 18 of *Leibniz International Proceedings in Informatics*, pages 301–312. LZI, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.301.

12 Robert Bonnet. On the cardinality of the set of initial intervals of a partially ordered set. In *Infinite and finite sets: to Paul Erdős on his 60th birthday, Vol. 1*, volume 10 of *Coll. Math. Soc. János Bolyai*, pages 189–198. North-Holland, 1975.

13 Laura Bozzelli and Pierre Ganty. Complexity analysis of the backward coverability algorithm for VASS. In *Proceedings of RP 2011*, volume 6945 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2011. doi:10.1007/978-3-642-24288-5_10.

14 Gianfranco Ciardo. Petri nets with marking-dependent arc cardinality: Properties and analysis. In *Proceedings of Petri Nets 1994*, volume 815 of *Lecture Notes in Computer Science*, pages 179–198. Springer, 1994. doi:10.1007/3-540-58152-9_11.

15 Jean-Baptiste Courtois and Sylvain Schmitz. Alternating vector addition systems with states. In *Proceedings of MFCS 2014*, volume 8634 of *Lecture Notes in Computer Science*, pages 220–231. Springer, 2014. doi:10.1007/978-3-662-44522-8_19.

16 Stéphane Demri, Marcin Jurdziński, Oded Lachish, and Ranko Lazić. The covering and boundedness problems for branching vector addition systems. *Journal of Computer and System Sciences*, 79(1):23–38, 2013. doi:10.1016/j.jcss.2012.04.002.

17 Leonard Eugene Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics*, 35(4):413–422, 1913. doi:10.2307/2370405.

18 Catherine Dufourd, Alain Finkel, and Philippe Schnoebelen. Reset nets between decidability and undecidability. In *Proceedings of ICALP 1998*, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115. Springer, 1998. doi:10.1007/BFb0055044.

19 Javier Esparza, Ruslán Ledesma-Garza, Rupak Majumdar, Philipp Meyer, and Filip Niksic. An SMT-based approach to coverability analysis. In *Proceedings of CAV 2014*, volume 8559 of *Lecture Notes in Computer Science*, pages 603–619. Springer, 2014. doi:10.1007/978-3-319-08867-9_40.

20 Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and primitive-recursive bounds with Dickson's Lemma. In *Proceedings of LICS 2011*, pages 269–278. IEEE, 2011. doi:10.1109/LICS.2011.39.

21 Alain Finkel. A generalization of the procedure of Karp and Miller to well structured transition systems. In *Proceedings of ICALP 1987*, volume 267 of *Lecture Notes in Computer Science*, pages 499–508. Springer, 1987. doi:10.1007/3-540-18088-5_43.

22 Alain Finkel, Pierre McKenzie, and Claudine Picaronny. A well-structured framework for analysing Petri net extensions. *Information and Computation*, 195(1):1–29, 2004. doi:10.1016/j.ic.2004.01.005.

23 Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001. doi:10.1016/S0304-3975(00)00102-X.

24 Thomas Geffroy, Jérôme Leroux, and Grégoire Sutre. Occam's Razor applied to the Petri net coverability problem. *Theoretical Computer Science*, 750:38–52, 2018. doi:10.1016/j.tcs.2018.04.014.

25 Jean Goubault-Larrecq, Simon Halfon, Prateek Karandikar, K. Narayan Kumar, and Philippe Schnoebelen. The ideal approach to computing closed subsets in well-quasi-orderings. In *Well-Quasi-Orders in Computation, Logic, Language and Reasoning*, volume 53 of *Trends in Logic*, pages 55–105. Springer, 2020. doi:10.1007/978-3-030-30229-0_3.

26 Lucie Guillou, Corto Mascle, and Nicolas Waldburger. Parameterized broadcast networks with registers: from NP to the frontiers of decidability. In *Proceedings of FoSSaCS 2024*, volume 14575 of *Lecture Notes in Computer Science*, pages 250–270, 2024. doi:10.1007/978-3-031-57231-9_12.

27 Richard M. Karp and Raymond E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969. doi:10.1016/S0022-0000(69)80011-5.

28 Ulla Koppenhagen and Ernst W. Mayr. Optimal algorithms for the coverability, the subword, the containment, and the equivalence problems for commutative semigroups. *Information and Computation*, 158(2):98–124, 2000. doi:10.1006/inco.1999.2812.

29 Joseph B. Kruskal. The theory of well-quasi-ordering: A frequently discovered concept. *Journal of Combinatorial Theory, Series A*, 13(3):297–305, 1972. doi:10.1016/0097-3165(72)90063-5.

30 Marvin Künnemann, Filip Mazowiecki, Lia Schütze, Henry Sinclair-Banks, and Karol Węgrzycki. Coverability in VASS revisited: Improving Rackoff's bound to obtain conditional optimality. In *Proceedings of ICALP 2023*, Leibniz International Proceedings in Informatics, pages 131:1–131:20. LZI, 2023. doi:10.4230/LIPIcs.ICALP.2023.131.

31 Ranko Lazić and Sylvain Schmitz. The complexity of coverability in ν-Petri nets. In *Proceedings of LICS 2016*, pages 467–476. ACM, 2016. doi:10.1145/2933575.2933593.

32 Ranko Lazić and Sylvain Schmitz. The ideal view on Rackoff's coverability technique. *Information and Computation*, 277:104582, 2021. doi:10.1016/j.ic.2020.104582.

33 Jérôme Leroux. Vector addition system reversible reachability problem. *Logical Methods in Computer Science*, 9(1), 2013. doi:10.2168/LMCS-9(1:5)2013.

34 Richard J. Lipton. The reachability problem requires exponential space. Technical Report 62, Department of Computer Science, Yale University, 1976. URL: http://www.cs.yale.edu/publications/techreports/tr63.pdf.

35 Ernst W. Mayr and Albert R. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46(3):305–329, 1982. doi:10.1016/0001-8708(82)90048-2.

36 Dmitri Novikov and Sergei Yakovenko. Trajectories of polynomial vector fields and ascending chains of polynomial ideals. *Annales de l'Institut Fourier*, 49(2):563–609, 1999. doi:10.5802/aif.1683.

37 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

38 Louis E. Rosier and Hsu-Chun Yen. A multiparameter analysis of the boundedness problem for vector addition systems. *Journal of Computer and System Sciences*, 32(1):105–135, 1986. doi:10.1016/0022-0000(86)90006-1.

39 Sylvain Schmitz. *Algorithmic Complexity of Well-Quasi-Orders*. Habilitation thesis, École Normale Supérieure Paris-Saclay, 2017. URL: http://tel.archives-ouvertes.fr/tel-01663266.

40 Sylvain Schmitz. The parametric complexity of lossy counter machines. In *Proceedings of ICALP 2019*, volume 132 of *Leibniz International Proceedings in Informatics*, pages 129:1–129:15. LZI, 2019. doi:10.4230/LIPIcs.ICALP.2019.129.

41 Sylvain Schmitz and Philippe Schnoebelen. Algorithmic aspects of WQO theory. Lecture notes, 2012. URL: http://cel.archives-ouvertes.fr/cel-00727025.

42 Philippe Schnoebelen. Revisiting Ackermann-hardness for lossy counter machines and reset Petri nets. In *Proceedings of MFCS 2010*, volume 6281 of *Lecture Notes in Computer Science*, pages 616–628. Springer, 2010. doi:10.1007/978-3-642-15155-2_54.

43 Rüdiger Valk. Self-modifying nets, a natural extension of Petri nets. In *Proceedings of ICALP 1978*, volume 62 of *Lecture Notes in Computer Science*, pages 464–476. Springer, 1978. doi:10.1007/3-540-08860-1_35.

44 Hsu-Chun Yen and Chien-Liang Chen. On minimal elements of upward-closed sets. *Theoretical Computer Science*, 410(24):2442–2452, 2009. doi:10.1016/j.tcs.2009.02.036.

FO Logic on Cellular Automata Orbits Equals MSO Logic

Guillaume Theyssier
I2M, CNRS, Université Aix-Marseille, France

Abstract

We introduce an extension of classical cellular automata (CA) to arbitrary labeled graphs, and show that FO logic on CA orbits is equivalent to MSO logic. We deduce various results from that equivalence, including a characterization of finitely generated groups on which FO model checking for CA orbits is undecidable, and undecidability of satisfiability of a fixed FO property for CA over finite graphs. We also show concrete examples of FO formulas for CA orbits whose model checking problem is equivalent to the domino problem, or its seeded or recurring variants respectively, on any finitely generated group. For the recurring domino problem, we use an extension of the FO signature by a relation found in the well-known Garden of Eden theorem, but we also show a concrete FO formula without the extension and with one quantifier alternation whose model checking problem does not belong to the arithmetical hierarchy on group $\mathbb{Z}^2$.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Logic; Theory of computation → Formal languages and automata theory

Keywords and phrases MSO logic, FO logic, cellular automata, domino problem, Cayley graphs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.154

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://hal.science/hal-04558369`

Acknowledgements We thank anonymous referees for their feedback and their suggestions to improve the presentation. We also warmly thank N. Pytheas Fogg for their hints about domino problems on regular graphs and numerous stimulating discussions that inspired this work.

1 Introduction

Symbolic dynamics was historically introduced as the study of one-dimensional infinite words representing discretized orbits of smooth dynamical systems through a finite partition of space [35, 34]. It has since then been largely extended to higher dimensions and arbitrary Cayley graphs of finitely generated groups, and seen rich developments going way beyond the initial motivations. The field of symbolic dynamics would now be better described as the study of sets of configurations (*i.e.* coloring of a graph) which can be defined by local uniform constraints (subshift of finite type, sofic subshift, etc) and maps on configurations acting by a uniform and local update rule (cellular automata, and morphisms between subshifts, which are the continuous maps commuting with translations [26, 12]). A fascinating aspects of these objects is that they are very simple to define, yet can produce very complex behaviors which make them challenging to analyze. They can be considered as a reasonable modeling tool [13], but more importantly, they constitute a natural model of computation for which undecidablity and computational hardness can arise in the most simple and seemingly unrelated questions in a spectacular way [28, 30, 25, 9, 29].

In symbolic dynamics, a major trend is to relate the properties of the considered objects (cellular automata or subshifts) to the structure of the underlying graph they are defined on (usually Cayley graphs of finitely generated groups). An emblematic example is the domino problem: since the breakthrough undecidability result of Berger [9], a lot of works

have focused on the characterization of Cayley graphs of groups for which the domino problem is undecidable [4, 5, 8, 7] or other graphs with less symmetries [20, 27]. Concerning cellular automata (CA), properties of the global map that can be expressed in first-order (FO) logic (or, said differently, FO properties of their orbit graph) are already challenging. For instance, injectivity and surjectivity problems were shown decidable on $\mathbb{Z}$ [1] and on context-free graphs [36], but undecidable on $\mathbb{Z}^2$ [29]. Other FO properties of CA were studied in relation to the graph structure: the Gotschalk conjecture [12, 23] asks whether the property "injectivity implies surjectivity" is true for all CA on all group. Besides, the garden of Eden theorem [12] characterizes amenability among finitely generated groups by the fact that another simple FO property[1] is true for all CA on this group.

It turns out that many central problems considered in symbolic dynamics can actually be rephrased in *monadic second order logic* (MSO). It was for instance noticed in [36] for injectivity and surjectivity of CA. MSO is a logical language that has received enormous interest, probably for its balance between expressivity and decidability in many cases. More precisely, for graph properties, algorithmic metatheorems [36, 15, 17] and reciprocals [32, 33] relates the complexity of the MSO model checking problem to the structure of considered graph or graph family. The key parameter at play here is treewidth [39] which is related to the non-existence of arbitrarily large grid minors [40]: bounded treewidth provides positive algorithmic results, while arbitrarily large grid minors often allows lower bounds or undecidability results. For instance, on a Cayley graph of a finitely generated group, the MSO model checking problem is decidable if and only if the group is virtually free (meaning having a free group as finite index subgroup) [33].

Of course, positive algorithmic results for MSO apply to the particular problems studied in symbolic dynamics, but lower bounds or undecidability results for MSO are not directly transferable. For instance, on the graph $\mathbb{Z}^2$, undecidability of MSO model checking follows from a straightforward encoding of Turing machines, while undecidability of the domino problem [9] or of injectivity problem of CA [29] requires detailed and non-straightforward constructions that involves ideas and tools of independent interest (aperiodic tile sets and space filling curves for instance). Moreover, it is still open to our knowledge whether both problems are undecidable on any Cayley graph of group which is not virtually free. More generally, much work remains to close the gap between the global understanding of MSO logic on arbitrary graphs and the particular MSO fragments at sake in symbolic dynamics that are mostly understood on Cayley graphs of some group families.

Contributions of the paper. In this paper we explore properties of CA on arbitrary labeled graphs (finite or infinite). To do this, we introduce a definition of local rules of CA that doesn't use any explicit reference to the local structure of the graph as it is classically done, but instead just relies on the notion of bounded labeled walks and multiset of possible states read at the end of these walks. In particular, we don't require the graph to be uniform nor to have bounded degree, but our notion exactly corresponds to the classical one on Cayley graphs of finitely generated groups. Moreover, a fixed CA local rule provides a well-defined CA global map on any graph with same label sets. We can thus explore the influence of the graph separately from the local rule. For instance, we can specify a graph property by a CA local rule f and a FO property ϕ of CA orbits: the set of graphs on which the local rule f induces a global map that satisfies ϕ. Our main result (theorems 8, 9 and 12) is then an equivalence between this approach and MSO logic, on arbitrary labeled graphs. Precisely, for every class $\mathcal{C}$ of graphs, the following two conditions are equivalent:

[1] As detailed in Section 5, it uses an additional relation in the FO signature which doesn't break the main point of our approach, an equivalence with MSO.

- there exists an MSO formula Ψ, such that $G \in \mathcal{C} \iff G \models \Psi$,
- there exists a pair (ϕ, f), where ϕ is a FO formula and f is a CA local rule, such that $G \in \mathcal{C}$ if and only if the global map induced by f on G satisfies ϕ.

Moreover, the pair (ϕ, f) can be effectively constructed from Ψ, and conversely.

Said differently, (FO,CA) pairs and MSO formulas define exactly the same graph languages, and the corresponding model checking problems are many-one equivalent on any fixed graph. We believe that this new characterization of MSO is particularly relevant in the context of symbolic dynamics.

First, FO properties of orbits of CA are a conjugacy invariant and were much studied as mentioned above. We obtain a characterization of the decidability of FO model checking for CA orbits on Cayley graphs (Corollary 14) and we show that undecidability can be obtained with a fixed FO formula (Corollary 16) exactly on non virtually free f.g. groups, which should be put in perspective with the Ballier-Stein conjecture [6]. We also obtain undecidability of a satisfiability problem for CA on finite graphs for a fixed FO formula (Corollary 17).

Besides, when fixing an arbitrary FO formula and letting the CA vary, we get fragments of MSO that make sense beyond the examples directly motivated by CA theory. In particular, we show in Section 5 that on Cayley graphs of finitely generated groups, such fragments do not depend on the choice of generators, and that the domino problem and its variants (seeded and recurring) are equivalent to the model checking problem of some simple fixed FO formula (Theorem 19). For this we use an additional relation in the case of the recurring domino problem that remains MSO-expressible (Lemma 18). Finally we obtain a FO formula with just one quantifier alternation whose model checking problem does not belong to the arithmetical hierarchy when fixing the graph $\mathbb{Z}^2$ (Theorem 20).

Warm-up examples. To fix ideas and give some intuition on how FO logic on CA orbits can be used to express graph properties, let us consider two well-known MSO properties and give an informal translation into a pair made of a FO formula and a CA local rule.

▶ **Example 1** (k-Colorable graphs)**.** Fix k and consider an undirected graph $G = (V, E)$. Consider the CA local rule with state set $S = \{0, \dots, k-1\}$ such that a vertex in state i changes its state to $i + 1 \bmod k$ if it has a neighbor in state i and remains in state i otherwise. It can be checked that the CA induced on G by this local rule has a fixed-point if and only if G admits a proper vertex coloring with k colors (*i.e.* a coloring where no two neighboring vertices have the same color).

▶ **Example 2** (Connected graphs)**.** Consider an undirected graph $G = (V, E)$. Consider the state set $S = \{0, 1, a_0, a_1, a_2\}$ and the CA local rule such that a vertex in state $i \in \{0, 1\}$ becomes $1 - i$, a vertex in state a_i becomes 0 if it has a neighbor in $\{0, 1\}$ and $a_{i+1 \bmod 3}$ otherwise. We claim that G is connected if and only if the CA induced on G by this local rule has no periodic orbit of minimal period 6, which is obviously an FO property of orbits.

Comparison with another characterization of MSO by automata. Several previous works established equivalence results between logic formalism and automata theory in the context of MSO languages of graphs. As mentioned above, [36] already noticed that some cellular automata properties can be translated into MSO. Following this, [44] and [42] introduced tiling and automata recognizers that are equivalent to (small) fragments of MSO. In [37], alternating distributed graph automata are introduced that recognize exactly the languages of graphs definable in MSO logic. These distributed automata are close to CA in the sense that they run on configurations (coloring of the input graph by states) and use finite local

memory and local communication between neighboring vertices. However, they are highly non-deterministic (alternating) and their accepting mechanism uses both initialization of the run to a particular configuration, and a global knowledge of the final configuration reached (precisely the set of states present in this configuration). Restrictions of this model to deterministic or non-deterministic automata (instead of alternating) gives strictly weaker fragments.

Our goal here is not to define a single automata model equivalent to MSO. Instead our approach motivated by symbolic dynamics uses deterministic CA on one hand and quantifier alternations in a separated FO formula which plays also the role of the accepting condition on the other hand. A key aspect is that we thus get natural fragments of MSO by fixing the FO formula and letting the CA rule vary. This also makes a strong difference with distributed alternating automata in the accepting mechanism since the FO formula does not offer any direct means of initializing some computation on a particular configuration, nor to detect presence of some particular states in a final configuration.

2 Formal definitions

Graphs. A (Σ, Δ)-labeled graph is a graph $G = (V, (E_\delta)_{\delta\in\Delta}, L)$, which can be finite or infinite, where $L : V \to \Sigma$ is the vertex labeling and $E_\delta \subseteq V \times V$ are the edges labeled by δ. In such a graph, given some finite word $w = w_1 \cdots w_k \in \Delta^*$, a path labeled by w is a sequence of vertices $v_1, \ldots, v_{k+1}$ such that, for any $1 \leq i \leq k$, $(v_i, v_{i+1}) \in E_{w_i}$. All graphs considered in this paper are simple, meaning that there is at most one edge of a given label between two given vertices[2]. Such a graph is said to be *connected* if it is connected as an undirected and unlabeled graph, *i.e.* if $G = (V, E)$ is connected where $(v, v') \in E$ if either $(v, v') \in E_\delta$ or $(v', v) \in E_\delta$ for some $\delta \in \Delta$. The set of connected graphs is denoted by $\mathcal{C}$. Σ will in some case be a singleton and can therefore be silently omitted: we speak about Δ-labeled graph in this case. An important class of graphs studied in symbolic dynamics is that of Cayley graphs of finitely generated (f.g.) groups. Given a f.g. group $(\Gamma, \cdot)$ and a (finite) set of generators Δ (including their inverses), the associated Cayley graph is the Δ-labeled graph where $(\gamma, \gamma') \in E_\delta$ if and only if $\gamma' = \gamma \cdot \delta$. An undirected graph G_1 is a minor of another undirected graph G_2 if it can be obtained from G_2 by deleting edges and vertices, and by contracting edges (*i.e.* identifying the vertices incident to the edge without creating multiple edges).

Cellular automata. Given a finite set of states S and a set of vertices V, a configuration is an element of S^V. It can be seen as a coloring of vertices by S. A CA is a map from configurations to configurations, that is induced by a uniform local rule. It is generally studied as a dynamical system through its set of orbits, which are sequences of configurations obtained by iterating the map from an initial configuration.

CA are usually defined over a fixed Cayley graph of a (f.g.) group [12]. Following this classical approach, the local rule defining a CA is formally a lookup table and is bound to a particular graph as it relies on local patterns defined over bounded balls of the graph. It also requires the graph to be uniform and of bounded degree. More general definitions were proposed that don't stick to a particular graph [3, 2], but they still rely on the hypothesis of bounded degree and use a particular labeling by port numbers.

[2] The hypothesis that our graphs are simple will be used in sections 4 and 5. We choose to adopt this hypothesis across the entire paper for simplicity and clarity, however the main results from Section 3 should also hold without this hypothesis.

We introduce in this section a simple definition of CA on arbitrary labeled graphs. The key advantage of our formalism is that a given local rule actually defines a CA on *any* labeled graph for fixed label sets. We can therefore fix a local rule and asks on which graphs the corresponding CA has a given property (as sketched in Section 1): a pair made of a local rule and a property of CA dynamics actually defines a property of graphs. Moreover, on Cayley graphs of f.g. groups our formalism is equivalent to the classical one. The CA local rules we consider can intuitively be seen as finite memory and finite distance exploring machines working as follows in parallel from each vertex v: they walk from v following all possible Δ-labeled walks up to some length r, and harvest states seen at the end of these walks and count their occurrences up to some constant k, then they decide from this information (a multiset) the new state at vertex v. From this point of view, the edge labeling by Δ acts as local directions that give more information on the position in the graph given a labeled walk, while vertex labeling σ gives some level of non-uniformity as in non-uniform cellular automata [18].

Let us now formalize this definition. Given $v \in V$ and a word $w \in \Delta^*$, we denote by $\mathcal{R}^w(v)$ the set of vertices reachable from v by a path labeled by w. If $w = \epsilon$ (empty word), then $\mathcal{R}^w(v) = \{v\}$ by definition. If G is the Cayley graph of a f.g. group, then $\mathcal{R}^w(v)$ is always a singleton, however there are generally several paths reaching the same vertex.

The k-capped multisets over set X are the multisets where no cardinality is greater than k, and are denoted $\text{MS}^k(X) = \{0, \dots, k\}^X$. Given a multiset $m \in \mathbb{N}^X$, we denote by $\mathsf{cap}^k(m)$ the k-capped multiset such that $\mathsf{cap}^k(m)(x) = \max(m(x), k)$. We denote by $A^{\leq r}$ the words of length at most r over alphabet A including the empty word ϵ. Given a set of states S and a configuration $c \in S^V$ and $v \in V$, the k-capped pattern of radius r at v in c is the k-capped multiset $P(c, v, r, k) \in \text{MS}^k\big(\Delta^{\leq r} \times S\big)$ defined by:

$$P(c, v, r, k) = \mathsf{cap}^k\big((w, s) \mapsto \#\{v' \in \mathcal{R}^w(v) : c_{v'} = s\}\big).$$

▶ **Definition 3** (CA local rules and global maps). *A CA local rule for (Σ, Δ)-labeled graphs of state set S, radius r and using k-capped multisets ($k \geq 1$) is a map*

$$f : \Sigma \times \text{MS}^k\big(\Delta^{\leq r} \times S\big) \to S.$$

For any (Σ, Δ)-labeled graph with vertices V, the global CA map $F_{G,f} : S^V \to S^V$ associated to the local map f and graph G is then defined by $F_{G,f}(c)_v = f(\sigma(c_v), P(c, v, r, k))$ for any configuration $c \in S^V$ and any vertex $v \in V$.

In the sequel, when considering a local map f, it always implicitly comes with a specified state set S and values of r and k defining its domain and image sets. We are mainly interested in the case where Σ is a singleton (uniform CA), but incorporating Σ in our definition allows non-uniformity in the local rule as in non-uniform CA [18].

▶ Remark 4. On a Cayley graph of f.g. group, $P(x, v, r, k)$ gives all the information about configuration x restricted to the ball in the graph centered in v and with radius r: indeed, in this case, $P(x, v, r, k)(q, w) = 1$ if and only if the unique vertex $v' \in \mathcal{R}^w(v)$ is such that $x_{v'} = q$, and $\mathcal{R}^w(v)$ describes the entire ball when w enumerates $\Delta^{\leq r}$. From this observation it follows that in the case of Cayley graphs and when Σ is a singleton, the global CA maps from Definition 3 are exactly the classical global CA maps (see for instance [12]).

Definition 3 explains how a given CA local rule f induces a CA global map $F_{G,f}$ on a given graph. $F_{G,f}$ is the main object of study in CA theory, as it represents a dynamical system.

▶ **Example 5** (Two definitions of Game of Life)**.** Consider the famous Game of Life CA $F : \{0,1\}^{\mathbb{Z}^2} \to \{0,1\}^{\mathbb{Z}^2}$ [10, 22]. In this example Σ is a singleton and thus ignored to simplify notations. First, let G_1 be the Cayley graph of $\mathbb{Z}^2$ with generators $n = (0,1)$, $e = (1,0)$ and their inverses. Let M be the following set of words in $\Delta^{\leq 2}$: n, n^{-1}, e, e^{-1}, ne, ne^{-1}, $n^{-1}e$, $n^{-1}e^{-1}$. Now define the local rule f_1 of radius 2 and using 4-capped multiset by

$$f_1(\mu) = \begin{cases} 1 & \text{if } \mu(1,\epsilon) = 0 \text{ and } \sum_{w\in M} \mu(1,w) = 3 \\ 1 & \text{if } \mu(1,\epsilon) = 1 \text{ and } 2 \leq \sum_{w\in M} \mu(1,w) \leq 3 \\ 0 & \text{otherwise.} \end{cases}$$

One can check that $F_{G_1,f_1} = F$. Now consider the undirected and unlabeled graph $G_2 = (\mathbb{Z}^2, E)$ with $((i,j),(i',j')) \in E$ if $|i-i'| \leq 1$ and $|j-j'| \leq 1$ and $(i,j) \neq (i',j')$. Here we denote $\Delta = \{u\}$. We then define another local rule f_2 of radius 1 and using 4-capped multiset as follows:

$$f_2(\mu) = \begin{cases} 1 & \text{if } \mu(1,\epsilon) = 0 \text{ and } \mu(1,u) = 3 \\ 1 & \text{if } \mu(1,\epsilon) = 1 \text{ and } 2 \leq \mu(1,u) \leq 3 \\ 0 & \text{otherwise.} \end{cases}$$

One can again check that $F_{G_2,f_2} = F$.

Logics. To make the exposition more concise, we suppose some familiarity with standard concepts of formal logic (variables, assignments, quantification, free variables, etc). MSO logic uses first-order variables (usually denoted by lower-case letters) representing vertices and second-order variables (usually denoted by upper-case letters) representing sets of vertices. To help reading, relations in formulas will use infix notation ($x\,R\,y$) while relation in the meta-language will use the set notation ($(x,y) \in R$).

▶ **Definition 6** (MSO formulas and their semantics)**.** *The set MSO formulas over label sets (Σ,Δ) is the set of atomic formulas:*

- *$x\,L\,\sigma$ for x a first-order variable and $\sigma \in \Sigma$ (meaning x has label σ),*
- *$x\,E_\delta\,x'$ for x and x' first-order variables and $\delta \in \Delta$ (meaning that there is an edge labeled δ from x to x'),*
- *$x = x'$ for first-order variables x and x' (meaning that x is equal to x'),*
- *$x \in X$ for first-order variable x and second-order variable X (meaning that x belongs to set X),*

closed by the usual logic connectives ($\vee$, $\wedge$, $\neg$) and quantifiers ($\forall$, $\exists$). Given an MSO formula Ψ, a (Σ,Δ)-labeled graph G and an assignment α of free variables of Ψ we define the semantics in the standard way starting from the obvious meaning of atomic formula above (see [17] for an in-depth introduction). We write $(G,\alpha) \models \Psi$ when Ψ is true on G with assignment α. If Ψ has no free variable, we simply write $G \models \Psi$ when Ψ is true on G.

We will sometimes use substitution of relations with formulas defining them. For instance we can write $\Psi(X, \Psi_R(x_1,x_2))$, where Ψ is a formula using an additional relation symbol R, to denote the MSO formula obtained by substituting Ψ_R for R in Ψ (with the usual precaution of renaming variables if necessary, see [17]).

We now define FO logic over orbits of CA: they are just formulas allowing quantifications over configurations and using two relations, equality and application of one step of the CA global rule.

▶ **Definition 7** (FO formulas and their semantics). *The set of FO formulas is made of atomic formulas:*

- $y = y'$ *(meaning that configuration y is equal to configuration y'),*
- $y \to y'$ *(meaning that the global CA map leads to y' in one step starting from y),*

and closed under the usual logic connectives and quantifiers. Given a FO formula ϕ, a CA global map $F : S^V \to S^V$ and an assignment β of free variables of ϕ to configurations from S^V, we write $(F, \beta) \models \phi$ to denote that F satisfies ϕ with assignment β following the obvious semantics of formulas starting from the relations above. When ϕ has no free variable, we just write $F \models \phi$.

For a CA global map $F : S^V \to S^V$ and a FO formula ϕ with free variables $(y^1, \dots, y^n)$, we use the shortcut $F \models \phi(c^1, \dots, c^n)$ for configurations $c^1, \dots, c^n \in S^V$ to express that $(F, \beta) \models \phi$ where β is the assignment given by $y^i \mapsto c^i$ for $1 \leq i \leq n$. We will also use the FO shortcut $y^0 \to^k_{\neq} y^k$ to represent the FO formula expressing that y^0 leads to y^k in k steps and the $k+1$ configurations involved in this partial orbit are pairwise different:

$$y^0 \to^k_{\neq} y^k \stackrel{\text{def}}{=} \exists y^1, \dots, \exists y^k : \bigwedge_{0 \leq i < k} y^i \to y^{i+1} \wedge \bigwedge_{i \neq j} y^i \neq y^j.$$

Notation convention: we will always use letter Ψ for MSO formulas, x or X for MSO variables, α for MSO assignments, ϕ for FO formulas, y for FO variables, G for graphs, f for CA local rules, F for CA global maps and c for configurations. We use notation c_v to denote state of configuration at vertex v, that's why we prefer the exponent notation $c^1, c^2, \dots$ to denote several configurations.

Combining graphs, CA and logics. The above definitions suggest various definitions of sets of objects (or languages): the graph language $\mathcal{G}(\Psi) = \{G : G \models \Psi\}$, the graph language $\mathcal{G}(\phi, f) = \{G : F_{G,f} \models \phi\}$, the set of CA local rules $\mathcal{CA}(\phi, G) = \{f : F_{G,f} \models \phi\}$, where we use the notation convention above, and where Σ and Δ are fixed so graphs are actually (Σ, Δ)-graphs and CA local rules are rules for such graphs. Moreover, $\mathcal{CA}(\phi, G)$ can be seen as decision problems where inputs are given as local maps of CA (model checking problem of ϕ on G).

3 Translation results

3.1 From FO/CA pairs to MSO

Whatever the state set, a CA configuration can be represented as a tuple of vertex sets: we can code the state at a vertex by the number of sets it belongs to among the tuple. This way, FO variables can easily be translated into tuples of second-order MSO variables undergoing the same quantification and we get an onto map from possible assignments of the tuple of second-order MSO variables onto possible assignments of the corresponding FO variable.

Under that coding, equality of configurations translates into a simple MSO formula with just one universal first-order quantifier. It remains to show that the other relation in the signature of FO, relation $\to$ which represents the application of one step of the CA global map, can also be translated into MSO: this boils down to checking that at each vertex the local rule is correctly applied, which itself boils down to counting up to some constant occurrences of states that can be reached by a labeled walk of bounded length.

▶ **Theorem 8.** *There is a recursive translation τ from pairs (ϕ, f) made of a FO formula ϕ and a CA local rule f to MSO formulas such that the following equivalence holds for any graph G: $F_{f,G} \models \phi \iff G \models \tau(\phi, f)$.*

3.2 From MSO to FO/CA pairs

This converse translation is less straightforward. Let us first give a simplified overview by considering an MSO formula Ψ in prenex normal form, and describing its translation into a CA local rule f together with a FO formula ϕ.

We will use binary configurations (*i.e.* elements of $\{0,1\}^V$) to code either second-order variable assignments (a set coded by its indicator function) or first-order variable assignments (a singleton coded by its indicator function). More generally, we can code several variable assignments in a configuration made of several binary components, *i.e.* configurations over a product alphabet $S = \{0,1\} \times \cdots \times \{0,1\}$. Given a configuration made of a product of binary components coding an assignment of several variables, the truth of an atomic formula using these variables over this assignment can be checked by a CA local rule in a distributed manner. With slightly more work and using a particular FO property of orbits, we can actually test any quantifier free formula in a distributed manner.

The CA local rule together with the FO formula we are going to construct will essentially enforce an erasing process that starts from a configuration made of a product of binary components (to code an assignment of all MSO variables at once) and then removes components of the product one by one at successive steps until reaching a fixed point: on one hand having all information about variables assignment at the start allows to check the truth of the quantifier-free matrix of the MSO formula as hinted before, and, on the other hand, having components to disappear individually in successive steps allows to make a FO quantification over configurations following exactly the MSO quantification over variables.

For instance, taking MSO formula $\Psi = \forall X_1, \exists x_2, \forall X_3, R(X_1, x_2, X_3)$, we construct a FO formula that is essentially of the form:

$$\forall y_1, \exists y_2, \forall y_3 : y_3 \to y_2 \to \ y_1 \circlearrowright$$

and a CA local rule that will ensure that $y_1 \approx a_1$, $y_2 \approx (a_1, a_2)$ and $y_3 \approx (a_1, a_2, a_3)$ where a_1 is an assignment for X_1, a_2 is an assignment for x_2 and a_3 is an assignment for X_3. It will also ensure, when in configuration y_3, that the assignments (a_1, a_2, a_3) satisfy $R(X_1, x_2, X_3)$. It is important to note that successive choices of assignments of variables y_1, y_2 and y_3 corresponds, up to a simple product encoding, to successive choices of assignments for variables X_1, x_2 and X_3. Non-deterministic choices of successive assignments are possible in this construction because they correspond to going backward in time in the canonical orbit enforced by the FO formula above: there is no contradiction with the determinism of cellular automata.

To turn this overview into a concrete construction, several technical points have to be addressed:

- the simplified behavior described above only works on some well-formed configuration; as usual in CA constructions, we will use local error detection and special error states to mark orbits of bad configurations and distinguish them from good ones: here we use two error states that oscillate with period two in order to ensure that any orbit reaching a fixed point has successfully passed all error detection mechanisms.
- to code first-order variables, binary configurations need to have exactly one vertex in state 1 and the local nature of CA prevents from verifying this (it cannot a priori distinguish a configuration with a single 1 from a configuration with two 1s arbitrarily far away, not to mention the case of non-connected graphs). Our construction handles this through the FO formula to be satisfied using a sibling configurations counting trick combined with a particular behavior of the CA which uses additional layers of states.

  The erasing process of the CA mentioned above will therefore take several steps for first-order variables, and only one step for second-order variables.

- checking a quantifier-free formula given an assignment of its variables encoded in a configuration is generally not doable in one step by a CA, especially on non-connected graphs that prevent the CA from communicating between components (think of the example: $x \in X \vee y \in X$) ; to solve this problem our construction will once again rely on a combination of FO logic over several steps and a particular behavior of the CA.

We first give a solution to these technical problems that works when we restrict to connected graphs. This construction is a little simpler than the general case that we address later, and it has the benefit to induce a better controlled dependence of the FO formula on the MSO formula (an aspect that will turn out to be useful in Section 4).

If Ψ is an MSO formula in prenex normal form with quantifier prefix $Q_1, \dots, Q_n$, its prefix signature is the word describing the alternations of quantifiers taking into account both the type of quantification and the order of quantified variables. More precisely, it is the word over alphabet $\{\forall, \exists\} \times \{1, 2\}$ obtained as follows: first map each quantifier to the alphabet according to the actual type and order, then remove any repetition of consecutive identical letters.

▶ **Theorem 9.** *There are two recursive transformations τ_{FO} from MSO formulas to FO formulas and τ_{CA} from MSO formulas to CA local rules such that, for any MSO formula Ψ, the pair made of $\phi = \tau_{FO}(\Psi)$ and $f = \tau_{CA}(\Psi)$ verifies:*

1. *for any connected graph G the following equivalence holds: $G \models \Psi \iff F_{G,f} \models \phi$,*
2. *if Ψ is prenex then ϕ depends only on its prefix signature.*

Let $\Psi = \mathbf{Q}_1 x_1^1, \dots, \mathbf{Q}_1 x_1^{k_1}, \mathbf{Q}_2 x_2^1, \dots, \mathbf{Q}_2 x_2^{k_2}, \dots, \mathbf{Q}_n x_n^1, \dots, \mathbf{Q}_n x_n^{k_n}, R(x_1^1, \dots, x_n^{k_n})$ be any MSO formula in prenex normal form where $\mathbf{Q}_1, \dots, \mathbf{Q}_n$ are the n quantifiers types (either $\forall$ or $\exists$ and either first or second-order) forming the prefix signature, variables $x_i^1, \dots, x_i^{k_i}$ are bound by quantifier of type $\mathbf{Q}_i$ and $R(x_1^1, \dots, x_n^{k_n})$ is the matrix of the prenex normal form (i.e. a quantifier free formula). We use this numbering of variables grouped by quantifiers type to obtain a more compact FO formula that only depends on the prefix signature of Ψ. Let's write $R(x_1^1, \dots, x_n^{k_n})$ in disjunctive normal form:

$$R(x_1^1, \dots, x_n^{k_n}) = \bigvee_{1 \le j \le d} C_j(x_1^1, \dots, x_n^{k_n})$$

where each clause C_j is a conjunction of terms which are atomic formula or negation thereof using variables $x_1^1, \dots, x_n^{k_n}$. Let $\mathcal{O}$ be the set of i such that $\mathbf{Q}_i$ is a first-order quantifier.

Structure of configurations. For each $1 \le i \le n$, let $\omega(i) = \left|\mathcal{O} \cap \{1, \dots, i\}\right|$ and define $\lambda(i) = i + 2 \cdot \omega(i)$. As it will become clear below, $\lambda(n)$ denotes the length of an orbit along which n particular configurations will be identified. Configurations along this orbit will use distinct state sets, and $\lambda(i)$ will also denotes the number of layers of the i-th configuration. We first introduce sets S_l for $1 \le l \le \lambda(n)$ that will be used to hold variable assignments and translate MSO quantification over variables of first or second-order into FO quantification over configurations. S_l is a product of l *layers* each of the form $\{0,1\}^{k_i}$ (*variable* layer) or $\{1, \dots, k_i\}$ (*choice* layer) for some i, or $\{0, 1\}$ (*control* layer). Intuitively, variable layers will hold MSO variables assignments, and choice and control layers are used only for first-order variables as a control mechanism. Sets S_l are precisely defined as follows:

- $S_1 = \{0,1\}^{k_1}$ and if $1 \in \mathcal{O}$ then $S_2 = S_1 \times \{1, \dots, k_1\}$ and $S_3 = S_2 \times \{0, 1\}$,
- for $1 \le i < n$, $S_{\lambda(i)+1} = S_{\lambda(i)} \times \{0,1\}^{k_{i+1}}$ and if $i + 1 \in \mathcal{O}$ then $S_{\lambda(i)+2} = S_{\lambda(i)+1} \times \{1, \dots, k_{i+1}\}$ and $S_{\lambda(i)+3} = S_{\lambda(i)+2} \times \{0, 1\}$.

If $j = \lambda(i) \leq l$ with $i \notin \mathcal{O}$ or $j = \lambda(i) - 2 \leq l$ with $i \in \mathcal{O}$ then the j-th layer of S_l is a *variable* layer, denoted as $V_i(S_l)$, and intuitively represents an assignment for the tuple of variables $x_i^1, \ldots, x_i^{k_i}$. For $1 \leq j \leq k_i$, we denote by $V_i^j(S_l)$ the j-th binary component of $V_i(S_l)$ which intuitively represents an assignment for variable x_i^j. For $i \in \mathcal{O}$ and $j = \lambda(i) - 1 \leq l$, the jth layer of S_l is a choice layer, denoted $\chi_i(S_l)$. Other layers, precisely j-th layers with $j = \lambda(i)$ with $i \in \mathcal{O}$, are *control* layers and denoted $K_i(S_l)$. Choice layer χ_i together with control layer K_i are used to ensure that the corresponding variable layer V_i correctly encodes a k_i-tuple of assignments of first-order variables, *i.e.* is a k_i-tuple of binary configurations each having exactly one position in state 1.

We denote by π the natural projection from S_l onto S_{l-1} (for any $2 \leq l \leq \lambda(n)$) which removes the last (l-th) layer of elements of S_l.

A well-formed configuration where all vertices are in a state from $S_{\lambda(n)}$ intuitively contains an assignment for all variables involved in formula Ψ (provided the control mechanism for first-order variables to be detailed below has been successful). We need to implement a distributed check of the truth of quantifier-free formula R on such an assignment. The key is to ensure that some clause C_j from the disjunctive normal form of R is chosen uniformly on the entire graph and to check everywhere that each terms of C_j is locally correct given the assignment. For $1 \leq j \leq d$, let $T_j = S_{\lambda(n)} \times \{j\}$ (recall that d is the number of clauses in the disjunctive normal form of $R(x_1^1, \ldots, x_n^{k_n})$). We again use notation π to denote the natural projection from T_j onto $S_{\lambda(n)}$, which removes the last component of states. T_j sates will be used to check clause C_j.

We can now define the state set of the CA $\tau_{CA}(\Psi)$ as

$$S = \{e_0, e_1\} \cup \bigcup_{1 \leq l \leq \lambda(n)} S_l \cup \bigcup_{1 \leq j \leq d} T_j$$

where e_1 and e_2 are distinct elements from the rest of S. We say that the *type* of an element of S is l if it belongs to S_l, *error* if it is e_0 or e_1 and *j-truth-check* if it belongs to T_j. We also naturally extend the notation V_i, χ_i and K_i for any state $s \in T_j$ by $V_i(s) = V_i(\pi(s))$, $\chi_i(s) = \chi_i(\pi(s))$ and $K_i(s) = K_i(\pi(s))$.

A configuration $c \in S^V$ is *valid* if the following conditions hold:

- states of all pairs of neighboring vertices of G are of the same type, and not of error type;
- choice layer χ_i of all pairs of neighboring vertices of G are equal;
- at each vertex v the control layers have zeros where the corresponding variable layers indicated by the choice layers have, precisely: $K_i(c_v) \leq V_i^{\chi_i(c_v)}(c_v)$ for all i such that $K_i(c_v)$ is defined (intuitively, a 1 in a control layer is authorized only if their is a 1 in the 'chosen' component of the corresponding variable layer);

Note that this definition of validity is purely local. For μ a capped multiset (second argument of the local rule of a CA), we write $\texttt{valid}(\mu)$ to express the local validity conditions above on μ: the first two item are checked on each pair of states (s, s') such that $\mu(s, \epsilon) \geq 1$ and $\mu(s', \delta) \geq 1$ for some $\delta \in \Delta$; the third item is checked on state s such that $\mu(s, \epsilon) \geq 1$.

CA local rule. The behavior of the CA local rule $f = \tau_{CA}(\Psi)$ can intuitively be described as follows:

- check the local validity of the configuration and if not generate states of error type that alternate with period 2 (between e_0 and e_1),
- apply projection π on states of type l with $l \geq 2$ and let states of type 1 unchanged,
- on states of type j-truth-check, verify that the assignment of variables $x_1^1, \ldots, x_n^{k_n}$ coded in variable layers V_i^j are such that $C_j(x_1^1, \ldots, x_n^{k_n})$ holds, and apply π if it is the case, or generate an error state otherwise.

Of course, the behavior on states of type j-truth-check above has to be understood locally since we are defining a CA. The implementation of this distributed truth check is as follows: each term $t(x_a^b, x_p^q)$ appearing in clause C_j (an atomic formula or its negation), where x_a^b is a first-order variable, is checked only at any vertex having a V_a^b component at 1, otherwise it is considered true by default. More precisely, for a pair (σ, μ) made of a vertex label and a capped multiset (arguments of the local rule f), we write $(\sigma, \mu) \models_{loc} C_j$ (clause C_j is locally valid) if the local state (unique s such that $\mu(s, \epsilon)$) is of type j-truth-check and the neighboring states also, and if all terms of C_j are locally true according to the previous rule. It turns out that $(\sigma, \mu) \models_{loc} C_j$ can be checked by a local rule of CA of radius 1 and using 1-capped multisets (*i.e.* sets). Precisely, all possible terms of clause C_j are treated as follows (denoting again s the unique state such that $\mu(s, \epsilon) \geq 1$):

- $x_a^b \, L \, \sigma'$ (resp. its negation) is true if and only if $V_a^b(s) = 0$ or if $\sigma = \sigma'$ (resp. $\sigma \neq \sigma'$),
- $x_a^b \, E_\delta \, x_p^q$ (resp. its negation) is true if and only if $V_a^b(s) = 0$ or if $1 \in \{V_p^q(q') : \mu(q', \delta) \geq 1\}$ (resp. 1 does not belong to this set),
- $x_a^b = x_p^q$ (resp. its negation) is true if and only if $V_a^b(s) = 0$ or $V_p^q(s) = 1$ (resp. $V_p^q(s) = 0$),
- $x_a^b \in x_p^q$ (resp. its negation) is true if and only if $V_a^b(s) = 0$ or $V_p^q(s) = 1$ (resp. $V_p^q(s) = 0$).

For t a term, we write $(\sigma, \mu) \models_{loc} t$ if t is locally true according to the above definition. Recall that x_a^b is a first order variable and the rest of the construction will ensure that, on configurations that matter, there will always exist a node at which $V_a^b(s) = 1$ so these tests will actually check that the assignments of variables encoded in the configuration do satisfy the term as desired.

The CA local rule f is then defined as follows (denoting again s the unique state such that $\mu(s, \epsilon) \geq 1$):

$$f(\sigma, \mu) = \begin{cases} e_{1-i} & \text{if } s = e_i \\ e_0 & \text{otherwise, and if } \neg \texttt{valid}(\mu), \\ e_0 & \text{otherwise, and if } s \text{ has type } j\text{-truth-check and } (\sigma, \mu) \not\models_{loc} C_j, \\ s & \text{otherwise, and if } s \text{ is of type 1}, \\ \pi(s) & \text{otherwise.} \end{cases}$$

FO formula. Most of the task of the FO formula is to check that n configurations are well-positioned in an orbit leading to a fixed-point. However, along this orbit, we also have to make checks to ensure that layers corresponding to first-order variables are well formed. For $i \in \mathcal{O}$, the CA behavior already ensures (by generating error states if not) that choice layers χ_i are uniform and that control layers K_i are upper-bounded by the chosen corresponding variable layer V_i^j. In this context, the check is done as follows (intuitively, variable y^i represents a configuration of type $\lambda(i)$ at each vertex, for some $i \in \mathcal{O}$):

$$\texttt{goodFOVAR}(y^i) \stackrel{\text{def}}{=} \forall y, \forall y', (y^i \to^2_{\neq} y' \wedge y \to^2_{\neq} y') \Rightarrow \texttt{\#siblings}(y) = 1,$$

where the formula $\texttt{\#siblings}(y) = 1$ expresses that there is exactly 1 configuration other than y with same image as y and can be written explicitly in FO as follows:

$$\exists y_s, \exists y_+, y_s \to y_+ \wedge y \to y_+ \wedge y_s \neq y \wedge (\forall y' : y' \to y_+ \Rightarrow (y' = y \vee y' = y_s)).$$

The idea is that a variable layer V_i is good if, for any choice j made in choice layer χ_i, there are only 2 possible ways to correctly complete the control layer K_i, because there is exactly one vertex v at which V_i^j is 1 and therefore at which K_i can be freely chosen to be 0 or 1.

Let us now define formulas to deal with the global structure of the orbit leading to a fixed point:

$$\texttt{seq}_1(y) \stackrel{\text{def}}{=} \begin{cases} y \to y & \text{if } 1 \notin \mathcal{O}, \\ \exists y^0, y \to^2_{\neq} y^0 \wedge y^0 \to y^0 \wedge \texttt{goodFOVAR}(y) & \text{if } 1 \in \mathcal{O}, \end{cases}$$

and for any $2 \leq i \leq n$

$$\texttt{seq}_i(y, y^+) \stackrel{\text{def}}{=} \begin{cases} y^+ \to y & \text{if } i \notin \mathcal{O} \\ y^+ \to^3_{\neq} y \wedge \texttt{goodFOVAR}(y^+) & \text{if } i \in \mathcal{O} \end{cases}.$$

Then, denote by $\texttt{good}_i(y^1, \ldots, y^i)$ for each $1 \leq i \leq n$ the formula:

$$\texttt{good}_i(y^1, \ldots, y^i) \stackrel{\text{def}}{=} \texttt{seq}_1(y^1) \wedge \bigwedge_{2 \leq k \leq i} \texttt{seq}_k(y^{k-1}, y^k).$$

The truth check for R has to make a non-deterministic choice of clause C_j to then use the distributed truth check implemented in the CA, we therefore define the following formula to be used in ϕ: $\texttt{truth}(y) \stackrel{\text{def}}{=} \exists y' \to y$ which make sense when y represents a configuration everywhere of type $\lambda(n)$.

As we show later, formula $\texttt{good}_i(y^1, \ldots, y^i)$ paired with CA F ensures that the configuration assigned to y^i is a well-formed configuration of type $\lambda(i)$ that holds an assignment for variables $(x_i^1, \ldots, x_i^{k_i})$ through its variable components $V_i^1, \ldots, V_i^{k_i}$. We use these formulas in ϕ to make restricted domain FO quantifications that exactly correspond to well-formed configurations that hold assignments of the corresponding MSO variables.

To make the formula ϕ more readable, we use the following syntactic sugar to express restricted domain quantification. If ϕ_D and ϕ are formulas containing y as free variable, then:

- $\exists y \in \phi_D, \phi$ stands for $\exists y, \phi_D \wedge \phi$,
- $\forall y \in \phi_D, \phi$ stands for $\forall y, \phi_D \Rightarrow \phi$.

We can finally define FO formula $\phi = \tau_{FO}(\Psi)$:

$$\phi \stackrel{\text{def}}{=} \mathbf{Q}'_1 y^1 \in \texttt{good}_1(y^1), \mathbf{Q}'_2 y^2 \in \texttt{good}_2(y^1, y^2), \ldots, \mathbf{Q}'_n y^n \in \texttt{good}_n(y^1, \ldots, y^n), \texttt{truth}(y^n)$$

where the FO quantifier $\mathbf{Q}'_i$ is $\exists$ if the MSO quantifier $\mathbf{Q}_i$ is existential, and $\forall$ if $\mathbf{Q}_i$ is universal.

Correctness of the construction. First, it can be checked that ϕ only depends on the prefix of Ψ and not on R. Second, the construction of f and ϕ are clearly computable from Ψ. The proof of Theorem 9 then relies on two lemmas. The first one ensures that $\texttt{good}_i(\ldots)$ predicates correctly translate assignments of FO variables quantified in ϕ into assignments of MSO variables quantified in Ψ and conversely.

▶ **Lemma 10.** *Let G be a (Σ, Δ)-labeled graph which is connected. Consider configurations $c^1, \ldots, c^i$ with $1 \leq i \leq n$ such that $F_{G,f} \models \texttt{good}_i(c^1, \ldots, c^i)$, then the following holds:*

1. *c^k is of type $\lambda(k)$ at each vertex, for $1 \leq k \leq i$;*
2. *for $1 \leq l \leq i$ and $1 \leq j \leq k_l$, variable component $V_l^j(c^k)$ is the same for all k with $l \leq k \leq i$, and it is such that exactly one vertex is in state 1 when $l \in \mathcal{O}$;*

3. *if $i < n$, for any assignment α of variables $(x_{i+1}^1, \dots, x_{i+1}^{k_{i+1}})$ there exists c^{i+1} such that*

$$F_{G,f} \models \texttt{good}_{i+1}(c^1, \dots, c^{i+1})$$

and $V_{i+1}^j(c^{i+1}) = \alpha(x_{i+1}^j)$ for all $1 \leq j \leq k_{i+1}$.

From the above lemma, if $F_{G,f} \models \texttt{good}_i(c^1, \dots, c^i)$ then c^i codes an assignment for all MSO variables x_l^j for $1 \leq l \leq i$ and $1 \leq j \leq k_l$ through components V_l^j of c^i. Precisely, when $l \in \mathcal{O}$ variable x_l^j is assigned to the unique vertex v such that $V_l^j(c_v^i) = 1$, and when $l \notin \mathcal{O}$ variable x_l^j is assigned to the set of vertices $\{v : V_l^j(c_v^i) = 1\}$. We denote this assignment by α_{c^i}. Moreover, α_{c^i} is an extension of assignment α_{c^j} for any $1 \leq j < i$.

Next lemma ensures that $\texttt{truth}(\dots)$ predicate correctly codes truth of formula $R(\dots)$ (matrix of Ψ) through the previous assignment translation.

▶ **Lemma 11.** *Under the hypothesis of Lemma 10, it holds that $F_{G,f} \models \texttt{truth}(c^n)$ if and only if $(G, \alpha_{c^n}) \models R(x_1^1, \dots, x_n^{k_n})$.*

The proof of Theorem 9 then consists in applying inductively the definition of truth by assignments of variables simultaneously in Ψ and ϕ, use Lemma 11 as base case and Lemma 10 for the induction step to translate assignments between MSO variables and FO variables.

Generalizing to arbitrary graphs. In the previous construction, we use the fact that considered graphs are connected in two places: to ensure uniformity of choice layers χ_i and to ensure a uniform choice of j for testing clause C_j with states of type j-truth-check. When generalized to possibly disconnected graphs, such uniformity conditions cannot be checked by the CA alone, simply because the CA has no possibility to communicate between connected components. We can compensate this impossibility by slightly changing the behavior of F and adding new FO constraints in the definition of ϕ. The price to pay is that the new definition of ϕ will depend on all parts of Ψ, not only its prefix signature.

First, the case of choice layers χ_i can be solved easily by de-grouping variables x_i^1 to $x_i^{k_i}$, *i.e.* renumbering variables in the prefix of Ψ by letting $k_i = 1$ and taking $\sum_{1 \leq i \leq n} k_i$ as the new value of n. Then, choice layers become trivial (they contain just one state) and are therefore always uniform by definition.

To solve the case of truth check, we introduce a general pre-image counting trick to ensure that a configuration c is uniform by a FO property. First, each state s of the alphabet S used by c is associated to a distinct prime number p_s, and there is a probing mechanism that selects a set of vertices and allows exactly p_s predecessors at selected vertices which are in state s, and only 1 at vertices which are not selected. The trick to check that c is s-uniform then consists in counting the number of pre-images up to $\max_{s' \in S} p_{s'}$: whatever the set of selected vertices, it should always be a power of p_s.

▶ **Theorem 12.** *There are two recursive transformations τ_{FO} from MSO formulas to FO formulas and τ_{CA} from MSO formulas to CA local rules such that for any MSO formula Ψ, and any graph G the following equivalence holds: $G \models \Psi$ if and only if $F_{G,\tau_{CA}(\Psi)} \models \tau_{FO}(\Psi)$.*

Using notations from the construction of Theorem 9, let us now describe precisely the modifications required to generalize to arbitrary graphs. As explained above, we assume $k_i = 1$ for all $1 \leq i \leq n$, so choice layers χ_i are always trivial and uniform. States of type l for $1 \leq l \leq \lambda(n)$ are identical as in the construction of Theorem 9. However, we need additional states to implement the truth check for the matrix R of formula Ψ, and it will spread over 3 time steps of the CA. The key is to ensure that some clause C_j from the disjunctive normal form of R is chosen uniformly on the entire graph and to check everywhere

that each terms of C_j is locally correct given the assignment. Let $2 = p_1 < p_2 < \cdots < p_d$ be the first d prime numbers. For $1 \leq j \leq d$, let $T_j^0 = S_{\lambda(n)} \times \{j\}$ and $T_j^1 = T_j^0 \times \{0,1\}$ and $T_j^2 = T_j^1 \times \{1, \ldots, p_j\}$. T_j^1 is used to *mark* vertices, while T_j^2 is used to alter the number of pre-images depending on j and the mark. We use the notation π to denote at the same time the natural projection from T_j^0 onto $S_{\lambda(n)}$, or from T_j^1 onto T_j^0 or from T_j^2 onto T_j^1, which removes the rightmost component of states. T_j^0 sates will be used to check clause C_j, while states from T_j^1 and T_j^2 will be used to guarantee through a pre-image counting trick that the same choice of j is made on the entire graph, thus ensuring correctness of the truth check of formula R.

We can now define the state set of the CA local rule $\tau_{CA}(\Psi)$ as

$$S = \{e_0, e_1\} \cup \bigcup_{1 \leq l \leq \lambda(n)} S_l \cup \bigcup_{1 \leq j \leq d} T_j^0 \cup T_j^1 \cup T_j^2.$$

We say that the *type* of an element of S is l if it belongs to S_l, *error* if it is e_0 or e_1 and (j,m)-*truth-check* if it belongs to T_j^m for $1 \leq j \leq d$ and $m = 0, 1$ or 2.

A configuration $c \in S^V$ is *valid* if the following conditions hold:

- states of all pairs of neighboring vertices of G are of the same type, and not of error type;
- at each vertex v the control layers have zeros where the corresponding variable layers indicated by the choice layers have, precisely: $K_i(c_v) \leq V_i^{\chi_i(c_v)}(c_v)$ for all i such that $K_i(c_v)$ is defined (intuitively, a 1 in a control layer is authorized only if their is a 1 in the 'chosen' component of the corresponding variable layer);
- for a state $(s, m, w) \in T_j^2$ where $s \in T_j^0$, $m \in \{0,1\}$ and $w \in \{1, \ldots, p_j\}$, it must be the case that $w = 1$ whenever $m = 0$ (this condition expresses intuitively, that only marked vertices can generate pre-images and it will allow through a pre-image counting trick in the FO formula to ensure that the choice to check truth of clause j is coherent on the entire graph).

We write $\texttt{valid}(\mu)$ when the capped multiset μ represents a locally valid neighborhood according to the above conditions.

The modified CA local rule f is almost identical as the one from Theorem 9 and defined as follows (denoting again s the unique state such that $\mu(s, \epsilon) \geq 1$):

$$f(\sigma, \mu) = \begin{cases} e_{1-i} & \text{if } s = e_i \\ e_0 & \text{otherwise, and if } \neg\texttt{valid}(\mu), \\ e_0 & \text{otherwise, and if } s \text{ has type } (j,0)\text{-truth-check and } (\sigma,\mu) \not\models_{loc} C_j, \\ s & \text{otherwise, and if } s \text{ is of type 1,} \\ \pi(s) & \text{otherwise.} \end{cases}$$

The key aspect of this new construction is that the correctness of the distributed truth check implemented in the CA above by states of type (j,m)-truth check rely on a modification of the considered FO formula. Let $\texttt{preimg}_j(y)$ be a FO formula expressing that the number of pre-images of y is either $> p_d$ or a multiple of p_j. We use the following modified definition of formula $\texttt{truth}(y)$:

$$\texttt{truth}(y) \stackrel{\text{def}}{=} \bigvee_{1 \leq j \leq d} \exists y^j : y^j \to y \wedge \left(\forall y' : y' \to y^j \Rightarrow \texttt{preimg}_j(y')\right)$$

which intuitively makes sense when y represents a configuration everywhere of type $\lambda(n)$, so y^j represents of configuration of type $(j,0)$-truth check everywhere. We this new definition of $\texttt{truth}(y)$, we introduce a strong dependence of the FO-formula on the matrix part of Ψ (by the presence of d for instance), which was not the case in Theorem 9.

The final FO formula ϕ is defined exactly as in Theorem 9 but using this modified version of $\texttt{truth}(y)$:

$$\phi \stackrel{\text{def}}{=} \mathbf{Q}'_1 y^1 \in \texttt{good}_1(y^1), \mathbf{Q}'_2 y^2 \in \texttt{good}_2(y^1, y^2), \ldots, \mathbf{Q}'_n y^n \in \texttt{good}_n(y^1, \ldots, y^n) : \texttt{truth}(y^n)$$

where the FO quantifier $\mathbf{Q}'_i$ is $\exists$ if the MSO quantifier $\mathbf{Q}_i$ is existential, and $\forall$ if $\mathbf{Q}_i$ is universal.

The proof of Theorem 12 is based on two lemmas adapted from Lemma 10 and 11, and can be copied word for word from the proof of Theorem 9, but simply removing the connectedness hypothesis.

4 Consequences on FO model checking for CA

A set of (Σ, Δ)-labeled graphs, or graph language, is *MSO-definable* if it is of the form $\mathcal{G}(\Psi)$ for some MSO formula Ψ. It is *FOCA-definable* if it is of the form $\mathcal{G}(\phi, f)$ for some FO formula ϕ and some CA local rule f. From Theorem 12, we get the following immediate corollary.

▶ **Corollary 13.** *MSO-definable and FOCA-definable graph languages are the same.*

Since the translations given in Section 3 are effective, we also obtain equivalence of model checking problems. Using [33], this gives a characterization of decidability of FO model checking for CA orbits on f.g. groups.

▶ **Corollary 14.** *On any fixed graph, FO model checking for CA is many-one equivalent to MSO model checking. In particular, FO-model checking for CA on a f.g. group Γ is decidable if and only if Γ is virtually free.*

We can get a more precise result on graphs of bounded degree, but we need an additional lemma in order to apply Theorem 9. It is well-known that undecidability in MSO can be obtained using the MSO-definability of grids and encoding Turing-computations on it [17]. Moreover, MSO on bounded degree graphs actually allows to code quantification on edge sets and not only vertex sets [16], so that we can express the grid minor relation (and not only the fact that a fixed graph is a minor). The following lemma doesn't use any new idea, but it ensures the fact that all this encoding process can be done within a fixed prefix signature. It also gives a variant of the construction for Σ^1_1-hardness (see [41]) using infinite grids and the recurring domino problem [25].

A $n \times n$-grid (or simply a grid when n is not specified), is the directed finite graph with vertices $\{(i, j) : 1 \leq i, j \leq n\}$ and edges the set of pairs $\big((i, j), (i + 1, j)\big)$ (the east edges) for $1 \leq j \leq n$ and $1 \leq i < n$, and pairs $\big((i, j), (i, j + 1)\big)$ (the north edges) for $1 \leq j < n$ and $1 \leq i \leq n$. A ∞-grid is the infinite directed graph with vertices $\mathbb{N} \times \mathbb{N}$ and same north/east adjacency relation.

▶ **Lemma 15.** *Fix some D. There exists a fixed quantifier prefix signature ρ for MSO such that, for any graph G of degree at most D that contains arbitrarily large grid as minors, deciding whether a given MSO formula in prenex form with prefix signature ρ is satisfied on G is undecidable. There also exists a fixed quantifier prefix signature ρ' for MSO such that, for any graph G of degree at most D that contains a ∞-grid as minor, deciding whether a given MSO formula in prenex form with prefix signature ρ' is satisfied on G is Σ^1_1-hard.*

This lemma together with Theorem 9 gives undecidability of model checking for a fixed FO formula expressed in the following corollary. For the first part of the corollary, we use the grid minor theorem [40] to translate the statement of the lemma in terms of treewidth [39],

and for the second part, it is known that the Cayley graph of a f.g. group which is not virtually free has a thick end (see [6]), hence it contains a ∞-grid as a minor by Halin's grid theorem [24]. The decidable part of the Corollary comes from [14, 36].

▶ **Corollary 16.** *Fix some D. There is a fixed FO formula ϕ such that for any connected graph G of degree at most D, the set $\mathcal{CA}(G, \phi)$ is computable if and only if G has finite treewidth. Moreover, there is a FO formula ϕ' such that for any Cayley graph G of a f.g. group with at most D generators, the set $\mathcal{CA}(G, \phi)$ is:*

- *computable if the group is virtually free,*
- *Σ^1_1-hard otherwise.*

In the context of modeling, it makes sense to consider distributed dynamical systems over arbitrary finite graphs. For instance automata networks (which are non-uniform CA on arbitrary finite graphs) are a well-established model for its use in the study of gene regulation networks [43, 31]. The theory of automata networks has largely grown around FO properties of orbits (typically fixed points) and their crucial dependence on the graph [38]. However, although many results deal with computational complexity in automata networks [11, 21], no natural undecidability result appeared so far to our knowledge. By our translation result from MSO, we can import Trakhtenbrot's theorem [19] to obtain an undecidability result for FO properties of CA orbits on finite graphs. It can intuitively be formulated as follows in the context of modeling: it is undecidable to know whether there is some finite interaction graph on which a given local interaction law (CA) induces a given dynamical property (FO). By the way, this corollary doesn't need Trakhtenbrot's theorem since we have all the expressive power of MSO (not only FO logic on graphs), and we can obtain it for a fixed FO formula. For instance, it follows directly from Theorem 9 and the techniques of Lemma 15 (see also [17, Theorem 5.6]).

▶ **Corollary 17.** *There exists a FO formula ϕ such that the following problem is undecidable: given some input CA local rule f, decide whether there exists a finite graph G with $F_{G,f} \models \phi$.*

5 Cayley graphs and domino problems

The case of Cayley graphs of f.g. groups is particular for our approach in two relevant ways. First, we can express in MSO that a set of vertices is infinite.

▶ **Lemma 18.** *For any $D \geq 1$, there is an MSO formula $\Psi(X)$ such that, on any Cayley graph G of some f.g. group with D generators and any assignment α, it holds that $(G, \alpha) \models \Psi(X)$ if and only if $\alpha(X)$ is infinite.*

From this lemma, it makes sense to extend the signature of FO logic with the addition of new relation $\overset{\infty}{=}$ on configurations which is at the heart of the 'Garden of Eden' theorem [12]: we write $c \overset{\infty}{=} c'$ whenever $\{v : c_v \neq c'_v\}$ is finite. We denote by FO($\overset{\infty}{=}$) the extension of FO signature by adding relation $\overset{\infty}{=}$. By Lemma 18, this extension remains within MSO. Precisely, in any fixed Cayley graph of a f.g. group and by a straightforward extension of Theorem 8, we can compute from any formula in FO($\overset{\infty}{=}$) and CA local rule, an equivalent MSO formula. We deduce that FO($\overset{\infty}{=}$) model checking for CA is decidable on some f.g. group exactly when MSO model checking is, and exactly when FO model checking for CA is.

Besides, if Γ is a f.g. group and G_1 and G_2 two Cayley graphs of Γ with two different sets of generators Δ_1 and Δ_2, then the definable CA global maps $F : S^\Gamma \to S^\Gamma$ are the same on G_1 and G_2. More precisely, there is a computable translation τ on CA local maps such that for any local map f for G_1, it holds: $F_{G_1,f} = F_{G_2,\tau(f)}$. This simply comes from the fact that we can translate Δ_1-walks into equivalent Δ_2-walks.

Therefore, if ϕ is a FO formula, we have that the sets $\mathcal{CA}(\phi, G_1)$ and $\mathcal{CA}(\phi, G_2)$ are actually Turing-equivalent. Said differently, by Theorem 8, any FO formula (actually any FO($\overset{\infty}{=}$) formula) defines a fragment of MSO logic whose model checking problem's Turing degree is independent of the choice of generators on a f.g. group Γ. It turns out that such fragments naturally capture the domino problem and its classical variants.

Given some finite set S, a domino specification $\mathcal{D}$ is a set of pairs $D_\delta \subseteq S^2$ for each $\delta \in \Delta$. A configuration $c \in S^V$ is said $\mathcal{D}$-valid for some graph if for any $v, v' \in V$ it holds: $(v, v') \in E_\delta$ implies $(c_v, c_{v'}) \in R_\delta$. The domino problem on a fixed graph, consists in deciding given $\mathcal{D}$ whether there exists a $\mathcal{D}$-valid configuration. The seeded domino problem consists in deciding given $\mathcal{D}$ and $s_0 \in S$ whether there exists a $\mathcal{D}$-valid configuration where s_0 occurs at some vertex. Finally, the recurring domino problem consists in deciding given $\mathcal{D}$ and $s_0 \in S$ whether there exists a $\mathcal{D}$-valid configuration where s_0 occurs infinitely often.

▶ **Theorem 19.** *Fix any Cayley graph G of any f.g. group, then:*

- *domino problem $\equiv_T \exists x, x \to x$,*
- *seeded domino problem $\equiv_T \exists x, \exists y, x \to x \wedge y \to x \wedge x \neq y$,*
- *recurring domino problem $\equiv_T \exists x, \exists y, x \to x \wedge y \to x \wedge \neg(x \overset{\infty}{=} y)$,*

where '$\equiv_T \phi$' means Turing-equivalent to the set $\mathcal{CA}(\phi, G)$ (model checking of ϕ for CAs on G).

The recurring domino problem is Σ^1_1-hard on $\mathbb{Z}^2$ [25], as well as the model checking of the corresponding FO($\overset{\infty}{=}$) formula from Theorem 19. It is just an existential formula, but it crucially uses relation $\overset{\infty}{=}$. We can actually also obtain Σ^1_1-hardness on $\mathbb{Z}^2$ with a pure FO formula with just one quantifier alternation, using pre-image counting trickery to check finiteness of a set and a reduction from the recurring domino problem.

▶ **Theorem 20.** *The problem $\mathcal{CA}(\phi, \mathbb{Z}^2)$ is Σ^1_1-hard where ϕ is the following formula:*

$$\phi \overset{def}{=} \exists y, y \to y \wedge \forall y', \forall y_1, \forall y_2, \forall y_3, (y' \neq y \wedge y' \to y \wedge \bigwedge_i y_i \to y') \Rightarrow \bigvee_{i \neq j} y_i = y_j.$$

6 Perspectives

We see several interesting research directions inspired by the approach taken in this work.

First, we believe that the dependence of ϕ on the degree or the number of generators in Corollary 16 is an artifact that can be removed with more work in the proof of Lemma 15. The same proof techniques should also provide hardness result at any level of the analytical hierarchy.

Then, this corollary should be put into perspective with the Ballier-Stein conjecture [6] saying that the domino problem on a f.g. group is decidable if and only if the group is virtually free. On one hand, it seems natural to ask whether the recurring domino problem (or its equivalent FO formula from Theorem 19) can play the role of formula ϕ' in Corollary 16. On the other hand, N. Pytheas Fogg pointed us simple examples of 4-regular graphs having an ∞-grid as subgraph on which the domino problem is decidable. So formula ϕ in Corollary 16 cannot be the FO formula expressing the existence of a fixed point (Turing-equivalent to the domino problem), and we wonder how simple such formula ϕ can be. Actually, we can ask a similar question for Corollary 17.

In general, we believe that the Turing degrees of FO-model checking problems for various concrete formulas is worth being investigated. As mentioned above, the Turing degree of all such model checking problems for a fixed FO formula is independent of the choice of

generators on f.g. groups, and we wonder how they change when changing the group among non virtually free groups. Injectivity of CA is a natural candidate that received little attention to our knowledge since the seminal result on $\mathbb{Z}^2$ [29].

Finally, we believe that there exists a fixed CA rule f for which the FO-model checking problem is undecidable on graph $\mathbb{Z}^2$ (the rule is fixed, the formula is given as input). While we see the proof ingredient to obtain this specifically for $\mathbb{Z}^2$, we have no idea of whether it is always the case that undecidability of FO model checking for CA orbits can be obtained for a fixed CA rule on any f.g. which is not virtually free.

References

1 S. Amoroso and Y.N. Patt. Decision procedures for surjectivity and injectivity of parallel maps for tessellation structures. *Journal of Computer and System Sciences*, 6(5):448–464, October 1972. doi:10.1016/s0022-0000(72)80013-8.

2 Pablo Arrighi and Gilles Dowek. Causal graph dynamics. *Information and Computation*, 223:78–93, February 2013. doi:10.1016/j.ic.2012.10.019.

3 Pablo Arrighi, Simon Martiel, and Vincent Nesme. Cellular automata over generalized cayley graphs. *Mathematical Structures in Computer Science*, 28(3):340–383, May 2017. doi:10.1017/s0960129517000044.

4 Nathalie Aubrun, Sebastián Barbieri, and Emmanuel Jeandel. *About the Domino Problem for Subshifts on Groups*, pages 331–389. Springer International Publishing, 2018. doi:10.1007/978-3-319-69152-7_9.

5 Nathalie Aubrun, Sebastián Barbieri, and Etienne Moutot. The Domino Problem is Undecidable on Surface Groups. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*, volume 138 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 46:1–46:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2019.46.

6 Alexis Ballier and Maya Stein. The domino problem on groups of polynomial growth. *Groups, Geometry, and Dynamics*, 12(1):93–105, March 2018. doi:10.4171/ggd/439.

7 Laurent Bartholdi. The domino problem for hyperbolic groups, 2023. arXiv:2305.06952.

8 Laurent Bartholdi and Ville Salo. Simulations and the lamplighter group. *Groups, Geometry, and Dynamics*, 16(4):1461–1514, November 2022. doi:10.4171/ggd/692.

9 R. Berger. The undecidability of the domino problem. *Mem. Amer. Math Soc.*, 66, 1966.

10 Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways for your Mathematical Plays*, volume 2. Academic Press, 1982. chapter 25.

11 Florian Bridoux, Amélia Durbec, Kevin Perrot, and Adrien Richard. Complexity of fixed point counting problems in boolean networks. *Journal of Computer and System Sciences*, 126:138–164, June 2022. doi:10.1016/j.jcss.2022.01.004.

12 T. Ceccherini-Silberstein and M. Coornaert. *Cellular automata and groups*. Springer Monographs in Mathematics. Springer-Verlag, Berlin, 2010. doi:10.1007/978-3-642-14034-1.

13 Bastien Chopard and Michel Droz. *Cellular Automata Modeling of Physical Systems*. Cambridge University Press, December 1998. doi:10.1017/cbo9780511549755.

14 Bruno Courcelle. The monadic second-order logic of graphs, ii: Infinite graphs of bounded width. *Mathematical Systems Theory*, 21(1):187–221, December 1988. doi:10.1007/bf02088013.

15 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, March 1990. doi:10.1016/0890-5401(90)90043-h.

16 Bruno Courcelle. The monadic second order logic of graphs vi: on several representations of graphs by relational structures. *Discrete Applied Mathematics*, 54(2-3):117–149, October 1994. doi:10.1016/0166-218x(94)90019-1.

17 Bruno Courcelle and Joost Engelfriet. *Graph structure and monadic second-order logic. A language-theoretic approach.* Encyclopedia of Mathematics and its applications, Vol. 138. Cambridge University Press, June 2012. Collection Encyclopedia of Mathematics and Applications, Vol. 138.

18 Alberto Dennunzio, Enrico Formenti, and Julien Provillard. Non-uniform cellular automata: Classes, dynamics, and decidability. *Information and Computation*, 215:32–46, June 2012. doi:10.1016/j.ic.2012.02.008.

19 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Springer Berlin Heidelberg, 1995. doi:10.1007/978-3-662-03182-7.

20 Louis Esperet, Ugo Giocanti, and Clément Legrand-Duchesne. The structure of quasi-transitive graphs avoiding a minor with applications to the domino problem, 2023. doi:10.48550/arXiv.2304.01823.

21 Guilhem Gamard, Pierre Guillon, Kevin Perrot, and Guillaume Theyssier. Rice-Like Theorems for Automata Networks. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*, volume 187 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2021.32.

22 Martin Gardner. Mathematical games. *Scientific American*, 223(4):120–123, October 1970. doi:10.1038/scientificamerican1070-120.

23 Walter Gottschalk. *Some general dynamical notions*, pages 120–125. Springer Berlin Heidelberg, 1973. doi:10.1007/bfb0061728.

24 R. Halin. Über unendliche wege in graphen. *Mathematische Annalen*, 157(2):125–137, April 1964. doi:10.1007/bf01362670.

25 David Harel. *Recurring Dominoes: Making the Highly Undecidable Highly Understandable*, pages 51–71. Elsevier, 1985. doi:10.1016/s0304-0208(08)73075-5.

26 G. A. Hedlund. Endomorphisms and Automorphisms of the Shift Dynamical Systems. *Mathematical Systems Theory*, 3(4):320–375, 1969.

27 Benjamin Hellouin de Menibus, Victor H. Lutfalla, and Camille Noûs. *The Domino Problem Is Undecidable on Every Rhombus Subshift*, pages 100–112. Springer Nature Switzerland, 2023. doi:10.1007/978-3-031-33264-7_9.

28 Michael Hochman and Tom Meyerovitch. A characterization of the entropies of multi-dimensional shifts of finite type. *Annals of Mathematics*, 171(3):2011–2038, 2010. doi:10.4007/annals.2010.171.2011.

29 Jarkko Kari. Reversibility and surjectivity problems of cellular automata. *Journal of Computer and System Sciences*, 48(1):149–182, February 1994. doi:10.1016/s0022-0000(05)80025-x.

30 Jarkko Kari. Rice's theorem for the limit sets of cellular automata. *Theoretical Computer Science*, 127(2):229–254, May 1994. doi:10.1016/0304-3975(94)90041-8.

31 S.A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437–467, March 1969. doi:10.1016/0022-5193(69)90015-0.

32 Stephan Kreutzer and Siamak Tazari. On brambles, grid-like minors, and parameterized intractability of monadic second-order logic. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 354–364. SIAM, 2010. doi:10.1137/1.9781611973075.30.

33 Dietrich Kuske and Markus Lohrey. Logical aspects of cayley-graphs: the group case. *Annals of Pure and Applied Logic*, 131(1-3):263–286, January 2005. doi:10.1016/j.apal.2004.06.002.

34 Douglas Lind and Brian Marcus. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, December 2020. doi:10.1017/9781108899727.

35 Marston Morse and G. A. Hedlund. Symbolic dynamics. *Amer. J. Math.*, 3:286–303, 1936.

36 David E. Muller and Paul E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985. doi:10.1016/0304-3975(85)90087-8.

37 Fabian Reiter. Distributed graph automata. In *2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science*. IEEE, July 2015. doi:10.1109/lics.2015.27.

38 Adrien Richard. Fixed points and connections between positive and negative cycles in boolean networks. *Discrete Applied Mathematics*, 243:1–10, July 2018. doi:10.1016/j.dam.2017.12.037.

39 Neil Robertson and P.D Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, September 1986. doi:10.1016/0196-6774(86)90023-4.

40 Neil Robertson and P.D Seymour. Graph minors. v. excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, August 1986. doi:10.1016/0095-8956(86)90030-4.

41 H. Rogers. *Theory of Recursive Functions and Effective Computability*. MIT Press, 1987.

42 Thomas Schwentick and Klaus Barthelmann. *Local normal forms for first-order logic with applications to games and automata*, pages 444–454. Springer Berlin Heidelberg, 1998. doi:10.1007/bfb0028580.

43 René Thomas. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, 42(3):563–585, December 1973. doi:10.1016/0022-5193(73)90247-6.

44 Wolfgang Thomas. *On logics, tilings, and automata*, pages 441–454. Springer Berlin Heidelberg, 1991. doi:10.1007/3-540-54233-7_154.

Regular Expressions with Backreferences and Lookaheads Capture NLOG

Yuya Uezato
CyberAgent, Inc., Tokyo, Japan
National Institute of Informatics, Tokyo, Japan

Abstract

Backreferences and lookaheads are vital features to make classical regular expressions (REGEX) practical. Although these features have been widely used, understanding of the unrestricted combination of them has been limited. Practically, most likely, no implementation fully supports them. Theoretically, while some studies have addressed these features separately, few have dared to combine them. Those few studies showed that the amalgamation of these features significantly enhances the expressiveness of REGEX. However, no acceptable expressivity bound for REWBLk – REGEX with backreferences and lookaheads – has been established. We elucidate this by establishing that REWBLk coincides with **NLOG**, the class of languages accepted by log-space nondeterministic Turing machines (NTMs). In translating REWBLk to log-space NTMs, negative lookaheads are the most challenging part since it essentially requires complementing log-space NTMs in nondeterministic log-space. To address this problem, we revisit Immerman–Szelepcsényi theorem. In addition, we employ log-space nested-oracles NTMs to naturally handle nested lookaheads of REWBLk. Utilizing such oracle machines, we also present the new result that the membership problem of REWBLk is **PSPACE**-complete.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Complexity classes

Keywords and phrases Regular Expression, Automata Theory, Nondeterministic Log-Space

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.155

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Long Version with Appendix*: `https://arxiv.org/abs/2404.17492`

Funding This work was supported by JST, CREST Grant Number JPMJCR21M3.

Acknowledgements I thank anonymous reviewers for their detailed comments on a previous version of this paper. I also sincerely thank anonymous reviewers for their careful reading and invaluable comments on the current version. All comments helped me to significantly improve the presentation of this paper and the clarity of proofs and constructions.

1 Introduction

Backreferences and lookaheads are practical extensions for classical regular expressions (REGEX). REGEX with backreferences – *REWB* – can represent the non context-free language $L = \{w\#w : w \in \{a,b\}^*\}$ with the REWB expression $E = ((a+b)^*)_x \ \# \ \backslash x$. Roughly telling about this expression, we save a substring matched with $(a+b)^*$ into the variable x, and later, we refer back to the matched string using $\backslash x$; therefore, E represents L. REWB is a classical calculus [2], and there are some results:

1. Schmid showed that **REWB**, the class of languages accepted by REWB, is contained in **NLOG**, the class of languages accepted by log-space nondeterministic Turing machines (NTMs) [23, Lemma 18].
2. Recently, Nogami and Terauchi showed that **REWB** is contained in the class of indexed languages, **IL** (an extension of context-free languages [1]) [20].
3. The membership problem of REWB – for an input REWB expression E and an input word w, deciding if E accepts w – is **NP**-complete [2].


On REGEX with lookaheads, there are two kinds of lookaheads. A *positive* lookahead $?(E)$ checks if the rest of the input can be matched by E *without* consuming the input. A *negative* lookahead $!(E)$ checks if the rest of the input *cannot* be matched by E without consuming the input. For example, the expression $?(E)F + !(E)G$ runs an expression F if E goes well with the rest of the input and runs an expression G otherwise. Thus, it can be read as `if` E `then` F `else` G and is helpful in writing practical applications. Although lookaheads are useful, they do not alter the REGEX expressiveness. This fact immediately follows from the result of alternating Turing machines that the language class of alternating finite automata corresponds to that of usual finite automata [6].

Now, it is a natural question: *How expressive are REWB with lookaheads?*

The class REWB with lookaheads – REWBLk – was studied by Chida and Terauchi [7, 8]. They have shown (a) **REWBLk**, the class of languages accepted by REWBLk, is closed under intersection and complement (it is immediately shown using lookaheads); (b) REWB is a proper subclass of REWB(+) – REWB with positive lookaheads – and REWB(−) – REWB with negative lookaheads; (c) the language emptiness problem of REWB(−) is undecidable. They also developed a new class of automata called *positive lookaheads memory automata (PLMFA)* and proved that REWB(+) equals PLMFA.

However, the following two key questions remain unresolved:

1. Which known language classes are related to **REWBLk**?
2. What is the computational complexity of the membership problem of REWBLk– a problem deciding if E accepts w for an input expression E and an input string w?

We solve these questions by presenting the following tight results:

(I) **REWBLk** = **NLOG**. Besides, **REWB** already contains an **NLOG**-complete language.

(II) The membership problem of REWBLk is **PSPACE**-complete.

Together with existing results, our results are summarized in the following table:

	Language Class	Membership Problem
REGEX + lookaheads	= **Regular** [6]	**P**-c [18]
REWB	$\subseteq$ **IL** [20], incomparable to **CFL** [4, 5], $\subseteq$ **NLOG** [23], $\ni$ **NLOG**-c language (I)	**NP**-c [2]
REWBLk	= **NLOG** (I)	**PSPACE**-c (II)

where **NLOG**-c is short for **NLOG**-complete, and the same applies to the others.

1.1 Difficulty in Translating REWBLk to Log-space NTMs

To investigate a hard part of translation from REWBLk to log-space NTMs, let us consider the following language, which is a well-known **NLOG**-complete language:

$$L_{\text{reach}} = \{s\,\#\,x_1 \to y_1 \# \cdots \# x_n \to y_n \#\, t : s, t, x_i, y_i \in V^*, \text{and there is a path from } s \text{ to } t\}$$

where the part $\#x_1 \to y_1 \# \cdots \# x_n \to y_n \#$ means the directed graph with direct edges $x_1 \to y_1$, $x_2 \to y_2$, and so on. The following REWBLk expression E_{reach} recognizes L_{reach}:

$$E_{\text{reach}} = (V^*)_{\text{Cur}}\,\#\,\left(?(\Sigma^*\,\#\,\backslash\text{Cur} \to (V^*)_{\text{Cur}}\,\#)\right)^*\,\Sigma^*\,\#\,\backslash\text{Cur}$$

where $\Sigma = V \cup \{\#, \to\}$. It first captures s into the variable Cur and walks on graphs while repeatedly evaluating the part $?(\cdots)$. Each evaluation makes a nondeterministic one-step move on the graph. The part $\Sigma^*\,\#\,\backslash\text{Cur}$ checks if we reach the goal t.

It is not difficult to structurally translate E_{reach} to a log-space NTM M such that $L(M) = L(E_{\mathrm{reach}})$. Now, using M, let us translate the if-then-else expression $?(E_{\mathrm{reach}})F + !(E_{\mathrm{reach}})G$ for some expressions F and G to a log-space NTM N. Let w be an input word $s\#edges\#t$. For the part $?(E_{\mathrm{reach}})F$, we run M in N; if M accepts w, we proceed to simulate F.

However, for the other part $!(E_{\mathrm{reach}})G$, we encounter problems:

(A) We need to check if all possible walks of M starting from s do not reach t.

(B) Walking paths starting from s and aiming for t become infinitely long, and thus there are infinitely many walks (branching) to be checked.

Therefore, we cannot run M directly in N to handle the negative lookahead $!(E_{\mathrm{reach}})$.

Our Idea: Immerman–Szelepcsényi Theorem & Log-space Nested-Oracles NTMs

To address the above problems, we leverage Immerman–Szelepcsényi theorem [16, 27]. This theorem states that the class **NLOG** is closed under complement; i.e., there exists a log-space NTM $\overline{M}$ such that $L(\overline{M}) = \Sigma^* \setminus L(M)$. Therefore, in our machine N, we run $\overline{M}$ for the part $!(E_{\mathrm{reach}})G$: If $\overline{M}$ eventually accepts w, then we proceed to simulate G.

On the other hand, REWBLk permits nested lookaheads, such as $?(\cdots!(\cdots?(\cdots)\cdots)\cdots)$. To handle them naturally, we employ log-space NTMs with *nested oracles* [16, 25, 19]. These machines can easily simulate REWBLk and are translated to log-space NTMs by Immerman–Szelepcsényi theorem; thus, **REWBLk** = **NLOG** holds. Moreover, oracle machines are crucial to showing that the membership problem of REWBLk is **PSPACE**-complete. To this end, we also give the new result that the membership problem of such machines is in **PSPACE**.

Structure of Paper

The rest of the paper is structured as follows. Section 2 discusses related work. Section 3 reviews REWBLk and demonstrates that **REWB** already contains an **NLOG**-complete language. Section 4 illustrates the expressiveness of REWBLk: (1) **NLOG** $\subseteq$ **REWBLk**; (2) the membership problem of REWBLk is **PSPACE**-hard; (3) REWB(+) and REWB(−) represent languages $\notin$ **IL**; and (4) the emptiness problems of REWB(+) and REWB(−) are undecidable even if $\Sigma = \{a\}$. Section 5 reviews log-space nested-oracles NTMs and their language class (= **NLOG**), and shows our new result: their membership problem is in **PSPACE**. Section 6 establishes **REWBLk** $\subseteq$ **NLOG** and that the membership problem of REWBLk is in **PSPACE**. Section 7 concludes this paper by giving open problems.

2 Related Work

As discussed in the Introduction, Chida and Terauchi have formalized REWBLk and its semantics [7, 8]. To our knowledge, their study is the first theoretical exploration into the simultaneous treatment of backreferences and lookaheads. Surprisingly, there has been no prior theoretical research on the topic despite their longstanding and widespread use. They introduced PLMFA (positive lookahead MFA) by expanding MFA (memory finite automata), which Schmid presented for studying REWB in [24]. One of their main results is the equivalence of PLMFA to REWB(+), established through translations between PLMFA and REWBLk. Nevertheless, they did not address: (i) a relationship between REWBLk and existing known language classes; (ii) the complexity of the membership problem of REWBLk. In contrast, we show that REWBLk captures **NLOG** and the membership problem of REWBLk is **PSPACE**-complete.

As highlighted in the Introduction, for REWB, Schmid showed **REWB** $\subseteq$ **NLOG** [23], and Nogami and Terauchi showed **REWB** $\subseteq$ **IL** [20]. Schmid also introduced MFA and showed that MFA corresponds to REWB [24]. About the relationship between **NLOG** and **IL**, it is worth noting that:

- **NLOG** $\not\subseteq$ **IL**. It is shown as follows. The language $L_{2^{2^n}} = \{a^{2^{2^n}} : n \in \mathbb{N}\}$ clearly belongs to **NLOG**. However, $L_{2^{2^n}} \notin$ **IL** by pumping lemmas for indexed languages [14, 11].
- **IL** $\not\subseteq$ **NLOG** unless **NLOG** = **NP**. It is shown as follows. **IL** can represent the language $L_{3\mathrm{SAT}}$, whose words are true 3-CNF formulas [22]. However, $L_{3\mathrm{SAT}} \notin$ **NLOG** unless **NLOG** = **NP**.

In the present paper, we take their result further and show that **REWBLk** = **NLOG** and that **REWB** already contains an **NLOG**-complete language.

We also refer to modern REGEX engines that (partially) support both backreferences and lookaheads. Several programming languages (for example, Perl, Python, PHP, Ruby, and JavaScript) and .NET framework support these features. However, their support is limited in both syntax and semantics. First, expressions like $(\backslash x \backslash x)_x$ and $(\backslash x + \backslash x)_x$ are rejected by most implementations because the variable x appears more than once in single captures. Next, in most implementations, the expression $F = (?((\backslash x a)_x))^* \backslash x$, which represents $\{\epsilon, a, a^2, \ldots\}$, does not match with a^2 and a^3, so on. It is due to conservative loop-detecting semantics. Such a semantics is standardized in ECMAScript [9]. This semantics works for F as follows. First, it unfolds the Kleene-* of $(?((\backslash x a)_x))^*$ as $(\epsilon + \underline{?((\backslash x a)_x)}(?((\backslash x a)_x))^*)$. Next, it enters the underline part and updates the variable x without consuming any input characters. Then, it tries to evaluate $(?((\backslash x a)_x))^*$ *again* at the same input position. At this point, many REGEX engines think that we enter an infinite loop and so stop unfolding the Kleene-*. Consequently, F only matches with ϵ (without loop unfolding) and a (with a single loop unfolding).

We can rephrase this situation as follows: (1) the amalgamation of lookaheads with variables induces side effects without consuming any characters; (2) however, the loop-detecting semantics overlooks such side effects and changes behaviors from the naive semantics. On the other hand, such conservative semantics work well for REGEX, REGEX with lookaheads, and REWB since they do not induce such side effects.

This paper presents a translation between REWBLks and log-space NTMs, enabling the development of REGEX engines that fully support backreferences and lookaheads. Notably, such engines run in *polynomial time* (for a fixed expression) since **NLOG** $\subseteq$ **P**.

3 Preliminaries: REWBLk

We review the syntax and semantics of REWBLk [7, 8] step-by-step below.

3.1 Regular Expressions with Backreferences and Lookaheads

We first give the syntax of REWBLk over an alphabet Σ and variables $\mathcal{V}$:

$$E ::= \sigma \mid \epsilon \mid E + E \mid E\,E \mid E^* \mid \underbrace{(E)_v}_{\text{capture}} \mid \underbrace{\backslash v}_{\text{backreference}} \mid \underbrace{?(E)}_{\text{positive lookahead}} \mid \underbrace{!(E)}_{\text{negative lookahead}}$$

where $\sigma \in \Sigma$ and $v \in \mathcal{V}$ is a variable. The first line defines classical regular expressions, REGEX. We consider the following subclasses in this paper: *REWB* (REGEX with captures and backreferences), *REWB(+)* (REWB with positive lookaheads $?(E)$), *REWB(−)* (REWB with negative lookaheads $!(E)$).

Semantics of REGEX

We first give a semantics for REGEX. To accommodate variables and lookaheads, configurations for REGEX are 4-tuples $\langle E, w, p, \Lambda\rangle$ where

- E is an expression to be executed;
- w is an input string and will not change throughout computation;
- p is a 0-origin position on w ($0 \leq p \leq |w|$). We write $w[p]$ for the symbol on the position p. It should be noted that $p = |w|$ is allowed to represent that we consume all the input.
- $\Lambda : \mathcal{V} \to \Sigma^*$ is an assignment from variables $\mathcal{V}$ to substrings of w.

We write $\mathcal{C}$ for the set of configurations. To denote all the results obtained by computing, we use a semantic function $[\![\cdot]\!] : \mathcal{C} \to \mathcal{P}(\mathbb{N} \times (\mathcal{V} \to \Sigma^*))$ where $\mathcal{P}(X)$ is the power set of X. On $[\![\langle E, w, p, \Lambda\rangle]\!] = \{\langle p_1, \Lambda_1\rangle, \ldots, \langle p_n, \Lambda_n\rangle\}$, each pair $\langle p_i, \Lambda_i\rangle$ means that, after executing E on w from p under Λ, we move to the position p_i and obtain an assignment Λ_i. On the basis of this idea, we define a semantics for each rule of the REGEX part:

$$[\![\langle \sigma, w, p, \Lambda\rangle]\!] = \begin{cases} \{\langle p+1, \Lambda\rangle\} & \text{if } p < |w| \text{ and } w[p] = \sigma, \\ \emptyset & \text{otherwise,} \end{cases} \qquad [\![\langle \epsilon, w, p, \Lambda\rangle]\!] = \{\langle p, \Lambda\rangle\},$$

$$\begin{aligned} [\![\langle E_1 + E_2, w, p, \Lambda\rangle]\!] &= [\![\langle E_1, w, p, \Lambda\rangle]\!] \cup [\![\langle E_2, w, p, \Lambda\rangle]\!], \\ [\![\langle E_1 E_2, w, p, \Lambda\rangle]\!] &= \bigcup_{\langle p', \Lambda'\rangle \in [\![\langle E_1, w, p, \Lambda\rangle]\!]} [\![\langle E_2, w, p', \Lambda'\rangle]\!], \\ [\![\langle E^*, w, p, \Lambda\rangle]\!] &= \bigcup_{i=0}^{\infty} [\![\langle E^i, w, p, \Lambda\rangle]\!] \quad \text{where } E^0 = \epsilon,\ E^i = \overbrace{EE\cdots E}^{i}. \end{aligned}$$

We note that our semantic function $[\![\langle E, w, p, \Lambda\rangle]\!]$ is inductively defined on the lexicographic ordering over the star height of E and the expression size of E. The start height and expression size of REWBLK is defined in the usual way.

We also note that each $[\![\langle E, w, p, \Lambda\rangle]\!]$ forms a finite set because the value of each variable x must be a substring of w, and also p is bounded as $0 \leq p \leq |w|$.

Semantics of REWB

A capture expression $(E)_x$ attempts to match the input string with E. If it succeeds, the matched substring is stored in the variable x.

$$[\![\langle\, (E)_x, w, p, \Lambda\,\rangle]\!] = \left\{\left\langle\ p', \Lambda'\left[\, x \mapsto w[p..p')\,\right]\ \right\rangle : \langle p', \Lambda'\rangle \in [\![\langle E, w, p, \Lambda\rangle]\!]\right\}$$

where $w[p..q)$ is the string $w[p]\, w[p+1] \cdots w[q-1]$.

A backreference $\backslash x$ refers to the substring stored previously by evaluating some $(E)_x$.

$$[\![\langle \backslash x, w, p, \Lambda\rangle]\!] = [\![\langle \Lambda(x), w, p, \Lambda\rangle]\!].$$

Semantics of Lookaheads

Positive lookaheads $?(E)$ run E from the current input without consuming any input. Although the change in head position is undone after running E, the modification to variables in E is not. So, we can also call it destructive lookahead.

$$[\![\langle ?(E), w, p, \Lambda\rangle]\!] = \left\{\langle p, \Lambda'\rangle : \langle p', \Lambda'\rangle \in [\![\langle E, w, p, \Lambda\rangle]\!]\right\}.$$

Negative lookaheads $!(E)$ also run E without consuming any input. If E does not match anything, we invoke a continuation. Compared with positive lookaheads $?(E)$, both the previous head position and the previous values of variables are recovered.

$$[\![\langle !(E), w, p, \Lambda\rangle]\!] = \begin{cases} \{\langle p, \Lambda\rangle\} & \text{if } [\![\langle E, w, p, \Lambda\rangle]\!] = \emptyset, \\ \emptyset & \text{otherwise.} \end{cases}$$

Readers may wonder why positive lookaheads modify variables while negative ones do not. This asymmetry arises because, in negative lookaheads, all computations uniformly fail, leaving no suitable configuration for altering variables. On the other hand, using the non-destructive property of the negative lookaheads, we can also define non-destructive positive lookaheads by $!(!(E))$. This expression executes E from the current position without any variable modifications.

Remark: Special character \$. Using a negative lookahead, we define $\$ = !(\Sigma)$ to check the end of the input. \$ is one of the most important applications of negative lookaheads.

▶ **Definition 1.** *The language of a REWBLk E, $L(E)$, is defined as follows:*

$$L(E) = \left\{ w \in \Sigma^* : \langle p, \Lambda\rangle \in [\![\langle E, w, 0, \iota\rangle]\!],\ p = |w| \right\} \quad \textit{where } \forall x \in \mathcal{V}.\, \iota(x) = \epsilon.$$

We can also consider another definition using $\gamma(x) = \bot$ instead of ι, where γ indicates that all variables are initially undefined. Although some real-world regular expression engines adopt that definition, we adopt the above ι-definition since it is tedious to initialize all variables x using $(\epsilon)_x$. The results discussed in this paper will not change regardless of which one is used. There is another formalization that excludes labels that appear multiple times in a single group, for instance $(\backslash x \backslash x)_x$. We discuss such a restriction, which we call reference restriction, in the immediately following section.

3.2 Reference Restriction and Normalization

Our formalization of REWBLk allows that variable references appear inside their definitions; for example, we allow expressions like $(\backslash x \backslash x)_x$. On the other hand, many studies on REWB do not allow them; i.e., expressions like $(\backslash x \backslash x)_x$ are prohibited [5, 23, 10, 20].

To the best of the author's knowledge, it remains unclear whether the restriction alters the expressive power of REWB. However, on REWBLk, the restriction does not change the language class of REWBLk. Here, we formalize the restriction and then present our normalization, which converts REWBLk expressions to language-equivalent restricted forms.

We use the function Var that receives an expressions E and returns the set of all the backreference variables inside E. For example,

$$\textsc{Var}(\ \backslash x \backslash x\ ) = \{x\},\ \textsc{Var}(\ (\backslash x \backslash y)_z\ ) = \{x, y\},\ \textsc{Var}(\ a\, b\, c\, ?(\ (\backslash w (\backslash x)_z)_y\ )\ ) = \{w, x\}.$$

It can be easily defined inductively on the expression size of E.

We also define a restriction, *reference restriction*. An REWBLk expression E satisfies the reference restriction condition if, for all the capture subexpressions $(F)_x$ of E, $x \notin \textsc{Var}(F)$ holds. For example, the expression $a\, b\, c\, ?((\backslash w (\backslash x)_z)_y)$ satisfies the condition. On the other hand, the expressions $(\backslash x \backslash x)_x$ and $a\, b\, c\, ?((\backslash w (\backslash y)_z)_y)$ do not.

▶ **Theorem 2.** *For any REWBLk expression E, there is an expression E' that satisfies the reference restriction condition and $L(E) = L(E')$.*

Proof. It suffices to perform variable renaming like alpha-conversion in lambda calculus to remove patterns $(\cdots x \cdots)_x$. Formally, we apply the following renaming function $\mathcal{R}$ from the innermost of E to the outermost:

$$\mathcal{R}(F) = \begin{cases} ?((F')_y)(\backslash y)_x & \text{where } y \text{ is a fresh variable} \quad \text{if } F = (F')_x, \\ F & \text{otherwise.} \end{cases}$$

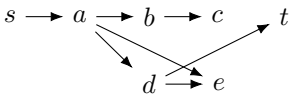
The expression $?((F')_y)(\backslash y)_x$ is equivalent to $(F')_x$ because it first stores the result of running F' to a fresh variable y and then puts it to the original variable x. Thus, the innermost-to-outermost applying $\mathcal{R}$ changes expressions to ones, which satisfy the restriction condition, without change their languages.

For example, an expression $(aa)_x(\backslash x \backslash x)_x$ is translated to $(aa)_x?((\backslash x \backslash x)_y)(\backslash y)_x$ by introducing a variable y. Another expression $(a)_x(b)_y(?[(\backslash x \backslash x)_x]aa\backslash y)_x$ is first translated to $(a)_x(b)_y(?[?((\backslash x \backslash x)_\alpha)(\backslash \alpha)_x]aa\backslash y)_x$ by introducing α. Then, introducing β, it is translated to $(a)_x(b)_y?((?[?((\backslash x \backslash x)_\alpha)(\backslash \alpha)_x]aa\backslash y)_\beta)(\backslash \beta)_x$. ◀

We will use this theorem in Section 6 to translate REWBLk to nested-oracles machines.

3.3 NLOG-Complete Language Accepted by REWB

Hartmanis and Mahaney proposed a decision problem called *TAGAP*, which is the topological sorted version of the reachability problem of directed *acyclic* graphs (DAG) [13]. Let us consider a word $w = \ulcorner x_1 \to y_1 \urcorner \ulcorner x_2 \to y_2 \urcorner \cdots \ulcorner x_n \to y_n \urcorner$, which represents a DAG. We call w *topologically sorted* if: for all pairs of $a \to b$ and $b \to c$ of w, $a \to b$ appears before $b \to c$ in w. The following example represents a DAG and one of its topologically sorted representation:



, $\ulcorner s \to a \urcorner \ulcorner a \to d \urcorner \ulcorner a \to e \urcorner \ulcorner d \to t \urcorner \ulcorner a \to b \urcorner \ulcorner d \to e \urcorner \ulcorner b \to c \urcorner$.

We define the language for TAGAP as follows:

$$L_{\text{TAGAP}} = \{s \# R \# t : R \text{ is a topologically sorted repr. of } G,\ t \text{ is reachable from } s \text{ in } G\}.$$

Hartmanis and Mahaney showed that this language is **NLOG**-complete [13, Theorem 3]. Since we only consider the topologically sorted representation, there is no longer a need to explore the entire input many times. In the above example, we can reach t from s by nondeterministically finding edges $s \to a$, $a \to d$, and $d \to t$ in this order by one-way scanning. Indeed, we can show the following theorem.

▶ **Theorem 3.** *$L_{TAGAP} \in$ **REWB**.*

Proof. The following expression E_{TAGAP} clearly recognizes L_{TAGAP}:

$$E_{\text{TAGAP}} = (V^*)_{\text{Cur}} \# \left(\Sigma^* \ulcorner \backslash\text{Cur} \to (V^*)_{\text{Cur}} \urcorner\right)^* \Sigma^* \# \backslash\text{Cur},$$

where V is an alphabet for vertices. ◀

4 Expressiveness of REWBLk

In this section, we present some theorems about the expressiveness of REWBLk.

4.1 Unary Non-Indexed Language

We consider the single exponential numerical language $L_{1exp} = \{a^{2^k} : k \in \mathbb{N}\}$ over the unary alphabet $\Sigma = \{a\}$. The language L_{1exp} is represented by the REWB(+) expression $E_{1exp} = ?(a)_x\,(?(\backslash x \backslash x)_x)^*\,\backslash x$. The part $?(a)_x$ initializes $x = a$ (i.e., $x = a^0$), and the Kleene-$*$ part iteratively doubles x.

Furthermore, we can represent the doubly exponential language $L_{2exp} = \{a^{2^{2^k}} : k \in \mathbb{N}\}$ by the following REWBLK expression E_{2exp}:

$$\begin{aligned} E(\alpha,\beta) &= \big(?[(\backslash\alpha\, a)_\alpha]\ ?[(\backslash\beta\backslash\beta)_\beta]\big)^*, \qquad \text{(it adds } a \text{ to } \alpha \text{ and doubles } \beta) \\ E_{2exp} &= ?((a)_m)\ E(n,m)\ ?((a)_x)\ E(y,x)\ ?(a^*\,?(\backslash m\$)\,?(\backslash y\$))\ \backslash x. \end{aligned}$$

It searches the numbers n, m, x, y that satisfy $2^n = m$, $2^y = x$, and $m = y$; so, $x = 2^{2^n}$. While unfolding the Kleene-* of $E(n,m)$ and $E(y,x)$, $2^n = m$ and $2^y = x$ hold. The part $?(a^*\,?(\backslash m\$)\,?(\backslash y\$))$ checks if $m =_?\ y$ by utilizing the negative lookahead expression $\$ = !(a)$.

We emphasize the known result $L_{2exp} \notin \mathbf{IL}$, which is shown by the pumping or shrinking lemma for indexed languages [14, 11]. Since we can carry out a similar construction of E_{2exp} in REWB(+) and REWB(−), we have the following result.

▶ **Theorem 4.** *REWB(+) and REWB(−) can represent unary non-indexed languages.*

Proof (Sketch). Due to the page limitation, we provide a proof sketch for the REWB(−) part and put the complete proof in the Appendix of the long version. Let us consider the following REWB(−) expression:

$$\begin{aligned} E'(\alpha,\beta) &= \big((\backslash\alpha\, a)_\alpha\ (\backslash\beta\backslash\beta)_\beta\big)^*, \\ E'_{2exp} &= (a)_m\ E'(n,m)\ (a)_x\ E'(y,x)\ !(!(\backslash m\$))\ !(!(\backslash y\$))\ a^*. \end{aligned}$$

While the expression E'_{2exp} resembles E_{2exp}, it lacks positive lookaheads. Let us explain E'_{2exp} step-by-step:

1. The expressions $(a)_m$ and $(a)_x$ initialize m and x by a as with E_{2exp}.
2. The subexpression $E'(n,m)$ repeatedly expands variables n and m as with E_{2exp}. So, executing $E'(n,m)$ *actually* consumes inputs as follows without positive lookaheads:

$$\underline{a^1}_n\ \underline{a^{2^1}}_m\ \underline{a^2}_n\ \underline{a^{2^2}}_m\ \underline{a^3}_n\ \underline{a^{2^3}}_m \cdots \underline{a^i}_n\ \underline{a^{2^i}}_m \cdots$$

3. The same holds for the expression $E'(y,x)$.

The part $!(!(\backslash m\$))$ is a non-destructive positive lookahead by $\backslash m\$$ that is simulated by double negative lookaheads; therefore, the part $!(!(\backslash m\$))\ !(!(\backslash y\$))$ requires $m = y$. If we pass the assertion, we consume the rest input by a^*. By replacing positive lookaheads with actual consuming, the language $L(E'_{2exp})$ *grows faster* (the notion *fast growth* is formalized in the Appendix) than L_{2exp}. This property implies that $L(E'_{2exp})$ is not an indexed language. ◀

It states that, even if restricted to unary languages, positive or negative lookaheads make REWB expressive and incomparable to **IL**. We can also show the undecidability of the emptiness problem, which is checking if $L(E) = \emptyset$ for a given expression E, of REWB(+) and REWB(−) over $\Sigma = \{a\}$.

▶ **Theorem 5.** *The emptiness problems of REWB(+) over a unary alphabet and REWB(−) over a unary alphabet are undecidable.*

The two undecidability results can be shown by translating the Post Correspondence Problem to the emptiness problems. Due to the page limitation, we put the proof of Theorem 5 in the Appendix of the long version.

4.2 Simulating Two-Way Multihead Automata by REWBLk

We show that REWBLk can simulate *two-way multihead automata*, which are a classical extension of automata and capture **NLOG** [12].

Simulating Two-Way One-Head Automata

We start from two-way *one*-head automata. Let $\mathcal{A} = (Q, q_{\text{init}}, q_{\text{acc}}, \Sigma, \Delta)$ be a two-way automata where $Q = \{q_0, q_1, \ldots, q_{|Q|-1}\}$ is a finite set of states, $q_{\text{init}} \in Q$ is the initial state, $q_{\text{acc}} \in Q$ is the accepting state, Σ is an input alphabet, and Δ is a set of transition rules. Each transition rule is of the form $p \xrightarrow{\tau/\theta} q$ where $p, q \in Q$, $\tau \in (\Sigma \cup \{\vdash, \dashv\})$, and $\theta \in \{-1, 0, 1\}$. The component θ indicates a head moving direction: (1) if $\theta = -1$ (resp. $\theta = 1$), we move the scanning head left (resp. right); (2) if $\theta = 0$, we *do not* move the scanning head.

For an input $w \in \Sigma^*$, we run $\mathcal{A}$ on the extended string $\vdash w \dashv$, which are surrounded by the left and right end markers. A configuration of $\mathcal{A}$ for $\vdash w \dashv$ is a tuple (q, i) where $q \in Q$ and $0 \leq i < 2 + |w|$, and thus the current scanning symbol is $(\vdash w \dashv)[i]$.

A transition rule $\delta = p \xrightarrow{\tau/\theta} q \in \Delta$ gives a labelled transition relation $\overset{\delta}{\Rightarrow}$ as follows:

$$(p, i) \overset{\delta}{\Rightarrow} (q, i + \theta) \quad \text{if } (\vdash w \dashv)[i] = \tau \text{ and } 0 \leq i + \theta < |w| + 2$$

The word w is accepted by $\mathcal{A}$ if the initial configuration $(q_{\text{init}}, 0)$, reading $\vdash$, has a computation path to a configuration with the accepting state q_{acc}. We now define the language of $\mathcal{A}$ as follows:

$$L(\mathcal{A}) = \{w : (q_{\text{init}}, 0) \overset{\delta_1}{\Rightarrow} (q_1, i_1) \overset{\delta_2}{\Rightarrow} \cdots \overset{\delta_n}{\Rightarrow} (q_{\text{acc}}, i_n)\}.$$

To simulate $\mathcal{A}$ by REWBLk, we use some variables $L, R, S \in \Sigma^*$. Intuitively, each variable means the following:

- If $L = w$ (resp. $R = w$), $\mathcal{A}$ is located on $\dashv$ (resp. $\vdash$). Otherwise, $\exists \sigma \in \Sigma.\, w = L\sigma R$. So, L (resp. R) means the left (resp. right) part of w.
- The length of S denotes the index i of the current state q_i of $\mathcal{A}$.

We formalize the above intuition as the following simulation $\sim$ between (q, i) and $\langle L, R, S\rangle$:

$$\begin{array}{ll} (q_j, 0) \sim \langle \epsilon, w, S\rangle & \text{if } |S| = j, \\ (q_j, |w| + 1) \sim \langle w, \epsilon, S\rangle & \text{if } |S| = j, \\ (q_j, i) \sim \langle L, R, S\rangle & \text{if } 1 \leq i \leq |w|,\ \exists \sigma.\ w = L\,\sigma\,R,\ |L| = i - 1, \text{ and } |S| = j. \end{array}$$

To represent all states, we need $|w| \geq |Q| - 1$. So, we mainly consider to represent $L(\mathcal{A}) \setminus L'$ where $L' = \{w \in L(\mathcal{A}) : |w| < |Q| - 1\}$. For instance, our simulation proceeds as follows:

$$\begin{array}{lll} (1) & \vdash^{q_0} ab \dashv \sim \langle L = \epsilon,\ R = ab,\ S = \epsilon\rangle, & (\text{if } R = w, \text{ then } L = \epsilon \text{ and } \mathcal{A} \text{ is on } \vdash) \\ (2) & \vdash a^{q_1} b \dashv \sim \langle L = \epsilon,\ R = b,\ S = a\rangle, & (\text{move right from (1)}) \\ (3) & \vdash ab^{q_1} \dashv \sim \langle L = a,\ R = \epsilon,\ S = a\rangle, & (\text{move right from (2)}) \\ (4) & \vdash ab \dashv^{q_1} \sim \langle L = ab,\ R = \epsilon,\ S = a\rangle, & (\text{if } L = w, \text{ then } R = \epsilon \text{ and } \mathcal{A} \text{ is on } \dashv) \\ (5) & \vdash ab^{q_2} \dashv \sim \langle L = a,\ R = \epsilon,\ S = ab\rangle, & (\text{move left from (4) and change } q_1 \text{ to } q_2) \\ (6) & \vdash a^{q_2} b \dashv \sim \langle L = \epsilon,\ R = b,\ S = ab\rangle. & \end{array}$$

As initialization, we use the expression $E_{\text{init}} = (\epsilon)_L\ ?((\Sigma^*)_R\,\$)\ (\epsilon)_S$, which sets $L = S = \epsilon$ and $R = w$ for the input w. To move the head right, we use the following expression:

$$E_{+1} = ?((\backslash L\Sigma)_L\ \Sigma\,(\Sigma^*)_R\,\$) + ?((\backslash L\Sigma)_L\,(\epsilon)_R\,\$).$$

The part $?((\backslash L\Sigma)_L\,\Sigma\,(\Sigma^*)_R\,\$)$ first attempts to expand L toward the right and then updates R. Similarly, we define the expression E_{-1} to move the head left as follows:

$$E_{-1} = ?((\Sigma^*)_L\,\Sigma\,(\Sigma\backslash R)_R\,\$) + ?((\epsilon)_L\,(\Sigma\backslash R)_R\,\$).$$

To check if the current scanning symbol is $\dashv$, $\vdash$, or $\sigma \in \Sigma$, we use the following expressions:

$$E_{\dashv} = ?(\backslash L\$),\ E_{\vdash} = ?(\backslash R\$),\ E_{\sigma} = ?(\backslash L\,\sigma\,\backslash R\$).$$

To check if the current state is q_i, we use the expression $E_{q_i} = ?\big(\Sigma^*\ ?(\backslash S)\$\ ?(\Sigma^i)\$\big)$. To change the current state q_i to q_j, we use the expression $E_{i\text{-to-}j} = E_{q_i}\,?(\Sigma^*\,(\Sigma^j)_S\,\$)$.

Now, each transition rule δ is simulated by the following expression $E(\delta)$ defined as:

$$E(q_i \xrightarrow{\tau/0} q_j) = E_{i\text{-to-}j}\ E_\tau, \quad E(q_i \xrightarrow{\tau/\theta} q_j) = E_{i\text{-to-}j}\ E_\tau\ E_\theta\ \ (\theta \in \{-1,+1\}).$$

Finally, the following expression $E_{\mathcal{A}}$ simulates $\mathcal{A}$, and $L(E_{\mathcal{A}}) = L(\mathcal{A})$ holds:

$$E_{\mathcal{A}} = E_{L'} + ?(E_{q_{\text{init}}}\,(E(\delta_1) + E(\delta_2) + \cdots + E(\delta_n))^*\,E_{q_{\text{acc}}})\,\Sigma^*$$

where $E_{L'}$ is a regular expression for the finite language L', and $\Delta = \{\delta_1, \delta_2, \ldots, \delta_n\}$.

We summarize the our translation as follows.

▶ **Lemma 6.** *For a two-way one-head automata $\mathcal{A}$, there is an expression $E_{\mathcal{A}}$ with 3 variables (L, R, and S) such that $L(E_{\mathcal{A}}) = L(\mathcal{A})$.* Especially, *the expression uses negative lookaheads only in the form of $\$$.*

Simulating Two-Way Multihead Automata

We extend the above argument to two-way *multihead* automata $\mathcal{M}$ [12, 15]. Compared to two-way one-head automata $\mathcal{A}$, $\mathcal{M}$ has multiple-heads on input strings. We write K for the number of heads. The difference between $\mathcal{A}$ and $\mathcal{M}$ are the following:

- Each configuration of $\mathcal{M}$ is a tuple $(q, i_1, i_2, \ldots, i_K)$ where q is the current state and i_j is the j-th head position.
- Each transition rule is $p \xrightarrow{(\tau_1,\ldots,\tau_K)/(\theta_1,\ldots,\theta_K)} q$ where $p, q \in Q$, $\tau_j \in \Sigma \cup \{\vdash, \dashv\}$ is used for inspecting the scanned symbol by j-th head, and θ_j denotes the head moving direction for the j-th head.

We define a transition relation $\Rightarrow$ in the same way as for $\mathcal{A}$. Let $\delta = p \xrightarrow{(\tau_1,\ldots,\tau_K)/(\theta_1,\ldots,\theta_K)} q$ be a rule and $C = (p, i_1, i_2, \ldots, i_K)$ be a valid configuration. If $\forall 1 \le j \le K.\,(\vdash w \dashv)[i_j] = \tau_j$, then we have $C \overset{\delta}{\Rightarrow} (q, i_1 + \theta_1, i_2 + \theta_2, \ldots, i_K + \theta_K)$. We define the language as follows:

$$L(\mathcal{M}) = \{w : (q_{\text{init}}, 0, 0, \ldots, 0) \overset{\delta_1}{\Rightarrow}\overset{\delta_2}{\Rightarrow} \cdots \overset{\delta_n}{\Rightarrow} (q_{\text{acc}}, i_1, i_2, \ldots, i_K)\}.$$

We can show the following lemma by simply extending our above construction for two-way one-head automata.

▶ **Lemma 7.** *For a given two-way K-head automata $\mathcal{M}$, we have an expression $E_{\mathcal{M}}$ with $3K$ variables such that $L(E_{\mathcal{M}}) = L(\mathcal{M})$. $E_{\mathcal{M}}$ uses negative lookaheads only in the form of $\$$.*

Proof. As with the two-way one-head automata, for each i-th head, we prepare variables L_i, R_i, and S_i. For each i-th head, by using the variables for i, we give expressions E^i_π where $\pi \in \{+1, -1, \sigma, \vdash, \dashv\}$. Employing the same E_{q_i} and $E_{i\text{-to-}j}$, we can give $E(\delta)$ for each transition rule δ of $\mathcal{M}$ and so $E_{\mathcal{M}}$. ◀

It is well-known that the class of languages accepted by two-way multihead automata corresponds to **NLOG** [12]; so, we have the following theorem.

▶ **Theorem 8.** $\mathbf{NLOG} \subseteq \mathbf{REWBLk}$.

4.3 Encoding Quantified Boolean Formula Problem

We translate the **PSPACE**-complete problem **QBF**, checking if a *quantified boolean formula* (QBF) is true, into the membership problem of REWBLK. We only consider QBFs in CNF since QBF restricted to CNF remains **PSPACE**-complete [3]. For instance, let us consider the following QBF Q and translate it to the equivalent one Q' by replacing $\forall$ with $\neg\exists\neg$:

$$Q : \forall a.\, \exists b.\, \forall c. \forall d.\, (a \vee b \vee c) \wedge (\overline{b} \vee c \vee d) \Rightarrow Q' : \neg\exists a.(\neg\exists b.(\neg\exists c.(\exists d.(\neg((a \vee b \vee c) \wedge (\overline{b} \vee c \vee d)))))).$$

In order to check if Q is true, we first structurally translate Q' into the following $E_{Q'}$:

$$\begin{aligned} E(v) &= \big((T)_v(F)_{\overline{v}} \;+\; (T)_{\overline{v}}(F)_v\big) \quad \text{where } v \text{ is a propositional variable,} \\ E_{Q'} &= !(E(a)\; !(E(b)\; !(E(c)\; E(d)\; !((\backslash a + \backslash b + \backslash c)(\overline{\backslash b} + \backslash c + \backslash d))))), \end{aligned}$$

where we replace $\neg$ with $!$, $\exists x$ with $E(x)$, x with $\backslash x$, $\overline{x}$ with $\overline{\backslash x}$, and $\vee$ with $+$.

We then check $w = TF\,TF\,TF\,TF\,TT \in_? L(E_{Q'})$. We explain the string w using the annotated version $T_1F_2T_3F_4T_5F_6T_7F_8T_9T_{10}$: (1) the first two characters T_1F_2 makes the two cases where $(a = T, \overline{a} = F)$ or $(a = F, \overline{a} = T)$; (2) similarly, T_3F_4 (resp. T_5F_6 and T_7F_8) works for b and $\overline{b}$ (resp. $c, \overline{c}$ and $d, \overline{d}$) ; (3) by T_9, we check if the expression $(a \vee b \vee c)$ holds (in the negative context); (4) by T_{10}, we also check if $(\overline{b} \vee c \vee d)$ holds. Thus, Q is true iff $w \in L(E_Q)$.

On the basis of the above translation using $E(v)$, we can translate every CNF-QBF Q to the corresponding expression E_Q and give the membership problem $TFTF \ldots TF\,TT \ldots T \in_? L(E_Q)$ in polynomial time for the size of Q. It implies the following result.

▶ **Theorem 9.** *The membership problem of REWBLK is **PSPACE**-hard.*

5 Log-space Nested-Oracles Nondeterministic Turing Machines

As we have stated in the Introduction, we utilize log-space nested-oracles NTMs. We will translate REWBLK to them in the next section.

We first review log-space NTMs. Here we especially consider *c-bounded k-working-tapes* log-space NTM $M = (k, c, Q, q_{\text{init}}, Q_F, \Sigma, \Gamma, \square, \Delta)$. Each component of M means:

- k is the number of working tapes $T_1, T_2, \ldots, T_k$.
- c is used to bound the size of working tapes. It will be defined precisely below.
- Q is a finite set of states, q_{init} is the initial state, and $Q_F \subseteq Q$ is a set of accepting states.
- Σ is an input alphabet.
- Γ is a working tape alphabet. $\square \in \Gamma$ is the blank symbol for working tapes.
- Δ is a set of transition rules. Each rule is either $p \xrightarrow{\tau \mid \theta} q$ or $p \xrightarrow[T_i]{\kappa \mapsto \kappa' \mid \theta} q$ where $p, q \in Q$, $\tau \in \Sigma \cup \{\epsilon, \vdash, \dashv\}$, $\kappa, \kappa' \in \Gamma \cup \{\vdash, \dashv\}$, and $\theta \in \{-1, +1, 0\}$.

Let $\vdash w \dashv$ ($w \in \Sigma^*$) be a string surrounded by the left and right end markers. Valid configurations of M for $\vdash w \dashv$ are tuples $\langle q, i, (T_1, i_1), \ldots, (T_k, i_k)\rangle$ where

- $q \in Q$ is the current state. $i \in \mathbb{N}$ $(0 \le i < |w| + 2)$ is the current head position on $\vdash w \dashv$.
- $T_x \in (\vdash \Gamma^C \dashv)$ where $C = c \cdot \lceil \log |w| \rceil$ is the x-th working tape surrounded by the end markers. $\lceil \cdot \rceil$ is the ceiling function to integers; for example, $\lceil \log 3 \rceil = \lceil 1.584 \ldots \rceil = 2$.
  Remark: The tape capacity C is determined by the parameter c and the input w.
- i_x is the x-th tape head position on T_x $(0 \le i_x < C + 2)$.

We write $\mathbf{Valid}_M(w)$ (or, simply $\mathbf{Valid}(w)$) for the set of valid configurations for the input w. It is clear that $|\mathbf{Valid}(w)| = |Q| \times (|w|+2) \times (|\Gamma|^C \times (C+2))^k$ where $C = c \cdot \lceil \log |w| \rceil$.

For an input string w, we write $\mathcal{I}(w)$ to denote the initial configuration on $\vdash w \dashv$:

$$\mathcal{I}(w) = \langle q_{\text{init}},\ 0,\ (\vdash \square^C \dashv, 0), \ldots, (\vdash \square^C \dashv, 0)\rangle \text{ where } C = c \cdot \lceil \log |w| \rceil.$$

Let $\xi = \langle p, i, (T_1, i_1) \ldots, (T_x, i_x), \ldots, (T_k, i_k)\rangle$ be a valid configuration on $\vdash w \dashv$. For each transition rule δ, we define a labelled transition relation $\overset{\delta}{\Rightarrow}$ on *valid* configurations as follows:

$$\frac{\delta = p \xrightarrow{\tau \mid \theta} q \qquad (\vdash w \dashv)[i] = \tau \ (\text{if } \tau \neq \epsilon)}{\xi \overset{\delta}{\Rightarrow} \langle q,\ i + \theta,\ (T_1, i_1), \ldots, (T_k, i_k)\rangle} \qquad \frac{\delta = p \xrightarrow[T_x]{\kappa \mapsto \kappa' \mid \theta} q \qquad \kappa = T_x[i_x]}{\xi \overset{\delta}{\Rightarrow} \langle q, i, \ldots (T_x[i_x] := \kappa', i_x + \theta) \ldots\rangle}$$

where $T_x[i_x] := \kappa'$ is the new working tape obtained by writing κ' to the position i_x.

We also simply write $\xi \Rightarrow \xi'$ if there is a transition rule $\delta \in \Delta$ such that $\xi \overset{\delta}{\Rightarrow} \xi'$.

We write $\mathbf{NLOG}(c, k)$ for the set of c-bounded k-working-tapes log-space NTMs. If c and k is not important, by abusing notation, we simply write $\mathbf{NLOG}$. For $M \in \mathbf{NLOG}$ and an input string w, we write $M(w, \xi)$ to denote the set of valid and acceptable configurations that are reachable from a valid configuration ξ on $\vdash w \dashv$:

$$M(w, \xi) = \{\xi' \ :\ \xi \Rightarrow^* \xi',\ \xi' = \langle q_{\text{acc}}, i, \mathcal{T}\rangle,\ q_{\text{acc}} \in Q_F\},$$

where $\mathcal{T} = (T_1, i_1) \ldots (T_k, i_k)$ is a sequence of pairs of a working tape and an index. Now the language $L(M)$ is defined as:

$$L(M) = \{w : M(w, \mathcal{I}(w)) \neq \emptyset\}.$$

Here we state a useful proposition, which will be used below sometimes.

▶ **Proposition 10.** *Let $M \in \mathbf{NLOG}(c, k)$. For any input w, to represent a single valid configuration or store $|\mathbfit{Valid}(w)|$, we need an extra $O(c \cdot k)$-bounded working tape.*

Proof. Since $|\mathbf{Valid}(w)| = |Q| \times (|w| + 2) \times (|\Gamma|^C \times (C + 2))^k$ where $C = c \cdot \lceil \log |w| \rceil$, $\log |\mathbf{Valid}(w)| = (k \cdot C) \log |\Gamma| + \cdots = O(k \cdot c) \log |w|$. So, we need an $O(c \cdot k)$-bounded tape. ◀

On log-space NTMs, we can solve the problem-***(B)*** in Section 1.1.

▶ **Proposition 11.** *Let $M \in \mathbf{NLOG}(c, k)$. There exists $N \in \mathbf{NLOG}(O(c \cdot k), k + 1)$ such that $L(M) = L(N)$ and, for any input w, all computations of N starting from w eventually halt.*

Proof. The number of reachable configurations is bounded by $\mathcal{B} = |\mathbf{Valid}_M(w)|$. So, we can safely ignore all paths P whose length $> \mathcal{B}$ without changing the accepting language. To check if the current path length $> \mathcal{B}$, we need an $O(c \cdot k)$-bounded tape by Proposition 10. ◀

We note that properties, like Proposition 11, are insufficient to show that a language class is closed under complement.[1] Thus, the problem-(***A***) in Section 1.1 is essentially hard; indeed, it is the interesting part of Immerman–Szelepcsényi theorem [16, 27]. In the following subsection, we revisit their theorem along with introducing log-space nested-oracles NTM.

[1] For example, we can translate any nondeterministic pushdown automata to real-time ones, which do not have ϵ-transitions. However, the class of context free languages is not closed under complement.

5.1 Augmenting NTM with Nested Oracles

We extend log-space NTM with finitely nested oracles (or subroutines) to naturally handle nested lookaheads of REWBLk. Similar to our definition of log-space NTMs, we consider c-bounded k-tapes log-space nested oracles NTMs.

To allow nested oracle calling, we inductively define our machines. First, as the base case, we write $\mathbf{OLOG}^0(c,k) = \mathbf{NLOG}(c,k)$ to denote machines without oracles. Next, as the induction step, we define $\mathbf{OLOG}^{x+1}(c,k)$ using $\mathbf{OLOG}^x(c,k)$ as follows:

- Each $M \in \mathbf{OLOG}^{x+1}(c,k)$ is a tuple $(k, c, Q, q_{\text{init}}, Q_F, \Sigma, \Gamma, \square, \Delta)$.
- There are new transition rules, *oracle transition rules*, of the form $p \xrightarrow{\in N} q$ and $p \xrightarrow{\notin N} q$ where $N \in \mathbf{OLOG}^y(c,k)$ and $y \leq x$. Their semantics will be defined immediately later.

We write $\mathbf{OLOG}^\omega(c,k)$ for $\bigcup_{i=0}^{\infty} \mathbf{OLOG}^i(c,k)$. If c and k are not important, we simply write as $\mathbf{OLOG}^n$ and $\mathbf{OLOG}^\omega$. To denote the nesting level of machines $M \in \mathbf{OLOG}^\omega(c,k)$, we inductively define the function *depth* as follows:

$$\mathit{depth}(M) = \begin{cases} 0 & \text{if } M \in \mathbf{OLOG}^0(c,k), \\ 1 + \max\{\mathit{depth}(N) : p \xrightarrow{\in N} q,\ p' \xrightarrow{\notin N} q \in \Delta(M)\} & \text{otherwise.} \end{cases}$$

Now, we define the semantics of oracle transition rules as follows:

$$\frac{p \xrightarrow{\in N} q \qquad N(w, \langle q^N_{\text{init}}, i, \mathcal{T}\rangle) \ni \langle r,\ j,\ \mathcal{U}\rangle}{\langle p, i, \mathcal{T}\rangle \Rightarrow \langle q, i, \mathcal{U}\rangle} \qquad \frac{p \xrightarrow{\notin N} q \qquad N(w, \langle q^N_{\text{init}}, i, \mathcal{T}\rangle) = \emptyset}{\langle p, i, \mathcal{T}\rangle \Rightarrow \langle q, i, \mathcal{T}\rangle}$$

where q^N_{init} is the initial state of N, $\mathcal{T}$ and $\mathcal{U}$ is a sequence of pairs of a working tape and an index $(T_1, i_1)\dots(T_k, i_k)$, and the function

$$N(w, \xi) = \{\xi' : \xi \Rightarrow^* \xi', \xi' = \langle q, i, \mathcal{T}\rangle, q \in Q_F(N)\}$$

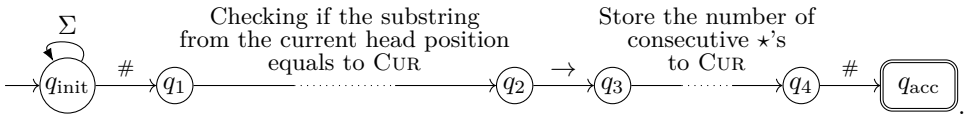
is defined inductively on the depth of machines.

The semantics of $p \xrightarrow{\in N} q$ means that: (1) we call an oracle (subroutine) N for $\vdash w \dashv$ with the current position i and the current working tapes $\mathcal{T}$ as its initial working tapes; and (2) if N accepts w, then we enter a state q with the original position i and working tapes $\mathcal{U}$ of N's accepting configuration. The semantics of $p \xrightarrow{\notin N} q$ means that, if N does not accept w, we enter a state q with the original position and working tapes $\mathcal{T}$.[2]

▶ **Example 12.** Using log-space nested-oracles NTMs, we simulate the following REWBLk:

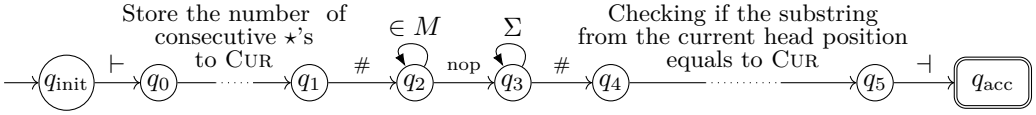
$$E_{\text{reach}} = (V^*)_{\text{Cur}}\,\#\,\left(?(\Sigma^*\,\#\setminus\text{Cur} \to (V^*)_{\text{Cur}}\,\#)\right)^*\,\Sigma^*\,\#\setminus\text{Cur}.$$

For the sake of simplicity, we assume that $V = \{\star\}$ is a unary alphabet. The subexpression $(\Sigma^*\,\#\setminus\text{Cur} \to (V^*)_{\text{Cur}}\,\#)$ is simulated by the following log-space NTM $M \in \mathbf{NLOG}$:



[2] For simplicity, our oracle formalization differs from traditional treatments [17, 21, 26, 3] in some points: (1) we omit the use of oracle tapes, and (2) we allow inheriting configurations from called oracles. Despite these differences, our definition is adequate for Theorem 13 and for REWBLk in Section 6.

Using M, we give the following machine $N \in \mathbf{OLOG}^1$, clearly accepting $L(E_{\text{reach}})$:

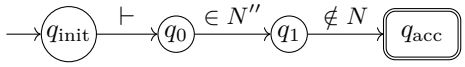


where edges labelled with "nop", $\xrightarrow{\text{nop}}$, mean transitions that only change states and do not depend scanning symbol. It just a syntax sugar because we can define $p \xrightarrow{\text{nop}} q$ by $p \xrightarrow{\epsilon \mid 0} q$.

▶ **Example 13.** We can also accept the language of non-reachability problems:

$$L_{\text{non-reach}} = \{s \# x_1 \to y_1 \# \cdots \# x_n \to y_n \# t : \text{there is no path from } s \text{ to } t\}.$$

To recognize this language, we use the following $M_{\text{non-reach}} \in \mathbf{OLOG}^2$:



where $N'' \in \mathbf{NLOG}$ recognizes the language defined by $V^* \# (V^* \to V^* \#)^* V^*$.

5.2 Collapsing OLOG$^\omega$ by Immerman–Szelepcsényi Theorem

Thanks to Proposition 11, we can give a decision procedure to check $w \in_? L(M_{\text{non-reach}})$ for our above examples. However, it is not clear that $\mathbf{OLOG}^2 =_? \mathbf{NLOG}$ and more generally $\mathbf{OLOG}^\omega =_? \mathbf{NLOG}$. For example, is there a log-space NTM N that recognizes $L_{\text{non-reach}}$?

Fortunately, the class **NLOG** is closed under complement, **NLOG** = **co-NLOG**. This result is known as Immerman–Szelepcsényi theorem [16, 27]. We employ their proof to collapse $\mathbf{OLOG}^x$ for some x to log-space NTMs $\mathbf{OLOG}^0 = \mathbf{NLOG}$.

▶ **Lemma 14.** *Let $M \in \mathbf{NLOG}(c, k)$. There is a machine $\overline{M} \in \mathbf{NLOG}(O(c \cdot k), k + \partial)$ where $L(\overline{M}) = \Sigma^* \setminus L(M)$ and ∂ is independent of M, c, and k.*

Proof. We review Immerman's original construction in [16]. For the reader who would like to know more detailed explanation about his construction, we recommend some literature [26]. His construction consists of the following two parts:

- Let START be a configuration of M. First, we compute the number C, the total number of configurations reachable from START.
- Next, using C, we check if there is a path from START to an acceptable configuration.

The first part is accomplished by the following pseudocode [16, Lemma 2].

```
global w; // input string

// For configurations x and x', we check if x ⇒ x'
def one_step_M(x, x'):
    foreach δ ∈ Δ(M): // Δ(M) is the set of transition rules of M
        if x ⇒^δ x': return True;
    return False;

// calculate the total number of configurations reachable from START
def counting_M(START):
    cur ← 1; // the number of reachable configurations within ≤ dist steps
```

```
    for(dist ← 0,next ← 0; dist < |Valid_M(w)|; dist += 1, cur ← next, next ← 0):
        foreach x ∈ Valid_M(w):
            count ← 0; found_x ← false;
            foreach y ∈ Valid_M(w):
                z ← START; // search a path from START to y
                for(i ← 0; z ≠ y & i < dist; i += 1):
                    z' ← Nondeterministically generated configuration;
                    if one-step(z, z'): z ← z';
                    else: break;
                if z = y:
                    count += 1;
                    if one-step(y, x): { next += 1; found_x ← true; break; }
            if ¬found_x & count ≠ cur: Halt and reject;
    return cur;
```

These functions require extra working tapes at least for variables of δ, *cur*, *dist*, *next*, x, *count*, y, i, z, and z'. By Proposition 10, for each variable, we need an $O(c \cdot k)$-bound working tape.

The second part is accomplished by the following pseudocode [16, Lemma 1].

```
// Judge whether an acceptable configuration can be reached from START.
// If so, it returns such a configuration y by SOME(y).
// Otherwise, we return the nothing by NONE.
def judge_M(START):
    C ← counting_M(START);
    count ← 0;
    foreach x ∈ Valid_M(w):
        y ← START;
        for(i ← 0; i ≤ C; i += 1):
            y' ← Nondeterministically generated configuration;
            if one-step(y, y'):
                y ← y';
                if y is an accepting configuration: return SOME(y);
                if y = x: { count += 1; break; }
            else: break;
    if count = C: return NONE;
    else: Halt and reject;
```

This function also requires extra $O(c \cdot k)$-bound working tapes.

Now we can build $\overline{M} \in \mathbf{NLOG}(O(c \cdot k), k + \partial)$ as a log-space NTM that simulates the function `judge` and then accepts inputs if $\texttt{judge}_M(\textsc{Start})$ is NONE. ◀

Repeatedly applying Immerman's construction collapses nested oracle machines to machines without oracles [16, Corollary 2].

▶ **Theorem 15.** *Let $M \in \boldsymbol{OLOG}^n(c, k)$ be a log-space n-nested-oracles NTM. There exists a log-space NTM $N \in \mathbf{NLOG}(O(c \cdot k^n), O(k \cdot n))$ such that $L(M) = L(N)$.*

Proof. We eliminate oracle transitions from the innermost to the outermost for M as follows. We replace $p \xrightarrow{\in N} q$ with $N \in \mathbf{OLOG}^0$ with multiple transition rules that perform: (1) save the head position H to an extra tape; (2) run N; (3) if we reach an accepting configuration

$\langle q_f, \mathcal{T}\rangle$, then we continue $\langle q, H, \mathcal{T}\rangle$. Similarly, we replace $p \xrightarrow{\notin N} q$ with $N \in \mathbf{OLOG}^0$ with multiple transition rules using $\overline{N}$ obtained by Immerman's construction. We emphasize that each generation of $\overline{N}$ increases c and k in the order of the statement of Lemma 14. ◀

5.3 Membership Problem of OLOG$^\omega$

We now show that the membership problem of log-space nested-oracles machines is in **PSPACE**. We first formally state our membership problem.

▶ **Definition 16** (Membership problem of **OLOG**$^\omega$). *The membership problem of* ***OLOG***$^\omega$ *is a decision problem of the following form:*
Inputs *Binary encoded integers c and k. A machine $M \in \boldsymbol{OLOG}^\omega(c,k)$. A word $w \in \Sigma^*$.*
Output *If $w \in_? L(M)$, return Yes. Otherwise, No.*

To show that the problem belongs to **PSPACE**, we would like to employ Theorem 15. However, it is not feasible because the theorem gives a log-space $O(c \cdot k^{|M|})$-bounded tapes machine N in general; i.e., N demands $O(c \cdot k^{|M|}) \cdot \log|w|$ space. Thus, we cannot simulate N in polynomial size in c, k, $|M|$, and $|w|$. To address this problem, we adopt Immerman's construction for **OLOG**$^\omega$ in an interpreter style.

▶ **Theorem 17** (Membership problem of **OLOG**$^\omega$ belongs to **PSPACE**). *Let w be an input word and $M \in \boldsymbol{OLOG}^\omega(c,k)$ be an input machine where c and k are binary encoded. We can decide if $w \in_? L(M)$ in polynomial space in c, k, $|w|$, and $|M|$.*

Proof. First, we extend the function `one-step` for oracle transitions as follows:

```
def one-step_{M_i}(x, x'): // return True if x ⇒ x'. Otherwise, False.
    foreach δ ∈ Δ(M_i):
        if δ is a non-oracle transition & x ⇒^δ x': return True;
        else: // δ is an oracle transition
            (p, i, T) ← x; // extract state, position, and tape contents from x
            if δ = p --∈N--> q:
                match judge_N(⟨q^N_init, i, T⟩): // pattern matching
                case Some(⟨p', i', U⟩) -> return (x' =_? ⟨q, i, U⟩);
                case None -> return False;
            if δ = p --∉N--> q:
                match judge_N(⟨q^N_init, i, T⟩):
                case Some(⟨p', i', U⟩) -> return False;
                case None -> return (x' =_? ⟨q, i, T⟩);
    return False;
```

Next, we generate the codes of $\texttt{one-step}_{M_i}$, $\texttt{counting}_{M_i}$, and $\texttt{judge}_{M_i}$ for all oracle machines M_i that appears in M. Such generation is carried out in polynomial-time for $|M|$. The total size of generated code is also polynomial in $|M|$.

We can also provide an interpreter for the generated code in polynomial time for $|M|$. While this interpreter needs a call stack for function calls, its depth is bounded by $\mathit{depth}(M) \leq |M|$. Additionally, the size of each stack frame is bounded by $O((c{\cdot}k)\log|w|)$ by Proposition 10. From the above argument, we can check $w \in_? L(M)$ using (nondeterministic) polynomial space with respect to c, k, $|w|$, and $|M|$. ◀

We will use this theorem to show that the membership problem of REWBLk belongs to **PSPACE** in the following section. To this end, we put remarks about this theorem.

Remark (Drop k from Theorem 17). We can decide $w \in_? L(M)$ in polynomial space for c, $|w|$, and $|M|$. Compared with Theorem 17, this refined version does not depend on k. This is because $k \leq |M|$ holds, even when k is binary encoded.

Of course, to establish this property, we assume a natural restriction that all tapes $T_1, \ldots, T_k$ are used in some transition rules. Indeed, if $M \in \mathbf{OLOG}^\omega(c, k)$ has a working tape T_i with $1 \leq i \leq k$ that is not engaged by M, a better parameter k' (where $k' < k$) should be employed, ensuring $M \in \mathbf{OLOG}^\omega(c, k')$. Given such constraints, $|M| \geq k \cdot \log(k) \geq k$ since k tapes necessitate $\log(k)$-space for tape identification.

Remark (Cannot drop c from Theorem 17). On the other hand, we cannot drop c from Theorem 17. Namely, we cannot decide $w \in_? L(M)$ in polynomial space for $|w|$ and $|M|$ because c is not bounded by Poly($|M|$) in general. (Please recall that $k \leq |M|$.) It can be understood from the following argument, which establishes $c = \Omega(2^{|M|})$. Let us consider a machine M that runs as follows:

- First, M fills a tape T_1 with N 1's by using states $q_1, q_2, \ldots, q_N$.
- Next, M interprets T_1 as a binary number t_1 and fills a tape T_2 with t_1 1's.
- Finally, M writes the contents of T_2 N times to T_3.
- For example, if $N = 3$, then M makes $T_1 = 111$ and then makes $T_2 = 1111111 = 1^{2^3-1}$ and then $T_3 = 3\,T_2 = 1^{3\cdot(2^3-1)} = 1^{N\cdot(2^N-1)}$.

The size of M satisfies $|M| = O((\log N)N)$ since it contains binary-encoded N states; thus, the length of T_3, $|T_3|$, satisfies $|T_3| = \Omega(2^{|M|})$. By appending the content of T_3 onto another tape $\log|w|$ times (where w is the input word), we establish that $c = \Omega(2^{|M|})$.

However, fortunately we can assume $c = 1$ when simulating REWBLK by $\mathbf{OLOG}^\omega$ as we will see below. It is crucial for showing that the membership problem of REWBLK belongs to **PSPACE**.

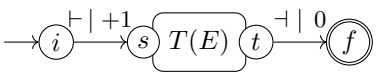
6 From REWBLk to Log-space NTM via OLOG$^\omega$

We finally show **REWBLk** $\subseteq$ **NLOG** by translating REWBLKs to $\mathbf{OLOG}^\omega$.

▶ **Theorem 18.** *Given a REWBLK expression E, we can translate it to $M \in \mathbf{OLOG}^\omega(1, O(|E|))$ in polynomial time in the size of $|E|$ where $L(E) = L(M)$.*

Proof. We inductively translate a REWBLK expression E to an $\mathbf{OLOG}^\omega$ $T(E)$ such that $w \in L(E)$ if and only if $T(E)$ successfully runs through w when started with w instead of $\vdash w \dashv$.

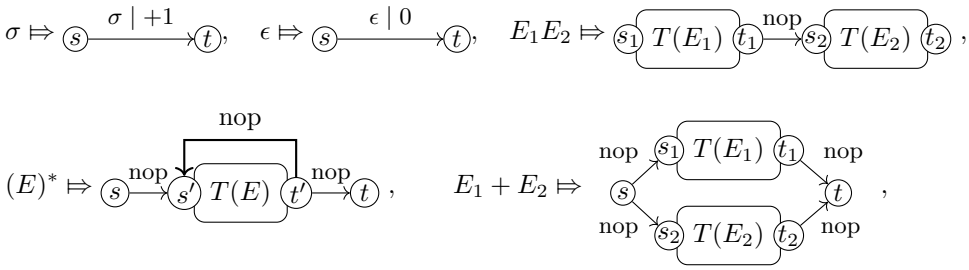
As we will see below, the generated $T(E)$ has a unique source state s, which does not have incoming edges, and a unique sink state t, which does not have outgoing edges. Please recall that E accepts an input w if it consumes all the input, i.e., if we have $\langle p, \Lambda\rangle \in [\![\langle E, w, 0, \iota\rangle]\!]$ with $p = |w|$. To build our M, we add two transition rules to $T(E)$ to treat the end markers $\vdash$ and $\dashv$ as follows:

$M =$ 

where i and f are initial and accepting states of M.

Translating REGEX Part

First, we give a translation for the REGEX as follows:

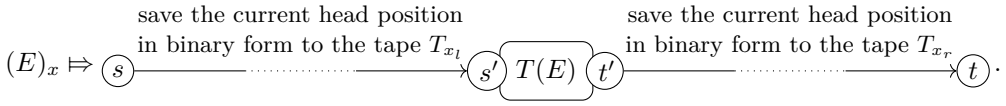


where the edges labelled with "nop" are the same ones used in Example 12, which just change states. This translation is identical to the McNaughton–Yamada–Thompson algorithm, which is well-known and found in textbooks of automata theory.

Translating Backreference and Capturing Expressions

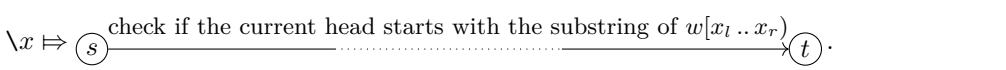
Using Theorem 2, we can assume that every variable x does not appear inside an expression capturing x; i.e., we avoid patterns such that $(\cdots x \cdots)_x$. The following translation heavily depends on the theorem.

We now focus on the part $(E)_x$ of REWB:



In order to keep the start position of the variable x, we first copy the current head position to the special working tape T_{x_l} in binary form. Then, we execute the expression E by running from the state s to t. Finally, we record the new head position into the working tape T_{x_r}. Now, $w[x_l \,..\, x_r) = w[x_l]w[x_l+1]\cdots w[x_r-1]$ is a substring matched with the expression E where x_l and x_r are the numbers corresponding to the contents of T_{x_l} and T_{x_r}.

Next, we focus on the part $\backslash x$ of REWB:



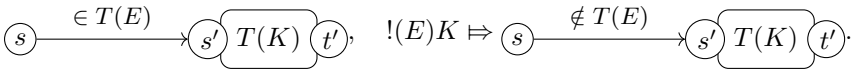
As we have seen above, the substring $w[x_l \,..\, x_r)$ denotes the value of the variable x. This checking task is accomplished using an extra tape T_{tmp} without changing T_{x_l} and T_{x_r}.

Remark: Why do we need Theorem 2. Let us consider an expression $E = (a)_x\ (\backslash x \backslash x)_x$ and run it for an input string a. We underline $a \notin L(E)$. By interpreting $(a)_x$, we set the position 0 to T_{x_l} and 1 to T_{x_r}. On the translation of $(\backslash x \backslash x)_x$, we first save the current head position 1 to the tape T_{x_l} and then proceed to the part $\backslash x \backslash x$. It should be noted that, at this point, T_{x_l} denotes 1; so, we cannot correctly recover the captured content by $(a)_x$. When meeting $\backslash x$, we check if the substring between $T_{x_l}(=1)$ and $T_{x_r}(=1)$ starts from the current head position. Since the substring is the empty string ϵ, we go through the part $\backslash x \backslash x$ and accepts a incorrectly.

To prevent us from this situation, we use Theorem 2. It rewrites E to $(a)_x\ ?((\backslash x \backslash x)_y)(\backslash y)_x$, and the rewritten expression is safely interpreted thanks to the variable y.

Translating Positive and Negative Lookaheads

For our construction, we need that lookahead are augmented with a continuation K, for instance $?(E)K$ and $!(E)K$. If a given expression does not satisfy this property, we add the expression ϵ, which always succeeds. For example, $(E_1E_2!(E_3))^*E_4 \Rightarrow (E_1E_2(!(E_3)\epsilon))^*E_4$. Now, we can easily translate $?(E)K$ and $!(E)K$ using oracle transition rules as follows:



By the semantics of our nested oracle machines, it is clear that $L(E) = L(M)$. ◀

As corollaries of Theorem 18, we have the following main results.

▶ **Corollary 19.** **REWBLk** = **NLOG**.

Proof. We already have **NLOG** ⊆ **REWBLk** by Theorem 8. It is an immediate result from Theorem 15 and 18 that **REWBLk** ⊆ **NLOG**. ◀

▶ **Corollary 20.** *The membership problem of REWBLk is* ***PSPACE****-complete.*

Proof. By Theorem 9, the membership problem of REWBLk is **PSPACE**-hard.

The result that the problem belongs to **PSPACE** is shown as follows. Let E be an input REWBLk expression and w be an input string. By Theorem 17, we translate $M \in \mathbf{OLOG}^{\omega}(1, O(|E|))$ such that $L(E) = L(M)$. This translation is carried out in polynomial time in $|E|$; so, $|M| = \mathrm{Poly}(|E|)$. By Theorem 17 and its remarks, we can check if $w \in_? L(M)$ in polynomial space in $|w|$ and $|M|$. This also derives that we can check if $w \in_? L(E)$ in polynomial space in $|w|$ and $|E|$. ◀

7 Future Work and Conclusion

We have shown that the language class REWBLk – regular expressions plus backreferences and lookaheads (without any restrictions) – captures the class **NLOG**. Our result closes the expressiveness about REWBLk. Furthermore, we have shown that the membership problem of REWBLk is **PSPACE**-complete. On the other hand, it remains unclear whether REWB(+) and REWB(−) are proper subclasses of REWBLk. For example, we conjecture that REWB(+) cannot recognize the language $L_{\text{prime}} = \{a^n : n \text{ is a prime number}\}$.[3] It is known that the language can be represented by REWB(−). Also, we conjecture that some **NLOG**-languages cannot be recognized by REWB(−).

References

1 A. V. Aho. Indexed grammars – an extension of context free grammars. *J. ACM*, 15(4):647–671, 1968.

2 A. V. Aho. Algorithms for finding patterns in strings. In J. V. Leeuwen, editor, *Algorithms and Complexity*, Handbook of Theoretical Computer Science, pages 255–300. Elsevier, 1990.

3 S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[3] REWB(−) recognizes L_{prime} as follows: (1) we define $E_{\text{composite}} = (aa\,a^*)_w \backslash w \backslash w^*\$$, which recognizes composite numbers; (2) then, we define $E_{\text{non-prime}} = a\$ + E_{\text{composite}}$, which recognizes non-prime numbers; (3) finally, we define $E_{\text{prime}} = !(E_{\text{non-prime}})\ a^*\ \$$. Readers interested in conducting a prime test with E_{prime} can try the following expression in C#: `@"^(?!((a$)|((?<W>(aa+))(\k<W>+)$))).*$"`.

4 M. Berglund and B. van der Merwe. Re-examining regular expressions with backreferences. *Theoretical Computer Science*, 940:66–80, 2023.
5 C. Câmpaanu, K. Salomaa, and S. Yu. A formal study of practical regular expressions. *International Journal of Foundations of Computer Science*, 14(06):1007–1018, 2003.
6 A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.
7 N. Chida and T. Terauchi. On lookaheads in regular expressions with backreferences. In *FSCD 2022*, volume 228, pages 15:1–15:18. Schloss Dagstuhl, 2022.
8 N. Chida and T. Terauchi. On lookaheads in regular expressions with backreferences. *IEICE Transactions on Information and Systems*, E106.D(5):959–975, 2023.
9 ECMAScript community. Ecmascript 2023 language specification. `https://262.ecma-international.org/14.0/#sec-runtime-semantics-repeatmatcher-abstract-operation`.
10 D. D. Freydenberger and M. L. Schmid. Deterministic regular expressions with back-references. *Journal of Computer and System Sciences*, 105:1–39, 2019.
11 R. H. Gilman. A shrinking lemma for indexed languages. *Theoretical Computer Science*, 163(1):277–281, 1996.
12 J. Hartmanis. On non-determinancy in simple computing devices. *Acta Informatica*, 1(4):336–344, 1972.
13 J. Hartmanis and S. Mahaney. Languages simultaneously complete for One-way and Two-way log-tape automata. *SIAM Journal on Computing*, 10(2):383–390, 1981.
14 T. Hayashi. On derivation trees of indexed grammars—an extension of the uvwxy-theorem—. *Publications of the Research Institute for Mathematical Sciences*, 9(1):61–92, 1973.
15 M. Holzer, M. Kutrib, and A. Malcher. Complexity of multi-head finite automata: Origins and directions. *Theoretical Computer Science*, 412(1):83–96, 2011.
16 N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.
17 N. Immerman. *Descriptive Complexity*. Springer Verlag, 1998.
18 T. Jiang and B. Ravikumar. A note on the space complexity of some decision problems for finite automata. *Information Processing Letters*, 40(1):25–31, 1991.
19 K.-J. Lange, B. Jenner, and B. Kirsig. The logarithmic alternation hierarchy collapses: $A\Sigma_2^{\mathcal{L}} = A\Pi_2^{\mathcal{L}}$. In *ICALP 87*, pages 531–541. Springer, 1987.
20 T. Nogami and T. Terauchi. On the expressive power of regular expressions with backreferences. In *MFCS 2023*, volume 272, pages 71:1–71:15. Schloss Dagstuhl, 2023.
21 C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
22 W. C. Rounds. Complexity of recognition in intermediate level languages. In *SWAT'73*, pages 145–158, 1973.
23 M. L. Schmid. Inside the class of regex languages. *International Journal of Foundations of Computer Science*, 24(07):1117–1134, 2013.
24 M. L. Schmid. Characterising regex languages by regular languages equipped with factor-referencing. *Information and Computation*, 249:1–17, 2016.
25 U. Schöning and K. W. Wagner. Collapsing oracle hierarchies, census functions and logarithmically many queries. In *STACS 88*, pages 91–97. Springer, 1988.
26 M. Sipser. *Introduction to the theory of computation*. Cengage Learning, third edition, 2013.
27 R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

Verification of Population Protocols with Unordered Data

Steffen van Bergerem
Humboldt-Universität zu Berlin, Germany

Roland Guttenberg
Technische Universität München, Germany

Sandra Kiefer
University of Oxford, UK

Corto Mascle
LaBRI, Université de Bordeaux, France

Nicolas Waldburger
IRISA, Université de Rennes, France

Chana Weil-Kennedy
IMDEA Software Institute, Madrid, Spain

Abstract

Population protocols are a well-studied model of distributed computation in which a group of anonymous finite-state agents communicates via pairwise interactions. Together they decide whether their initial configuration, i. e., the initial distribution of agents in the states, satisfies a property. As an extension in order to express properties of multisets over an infinite data domain, Blondin and Ladouceur (ICALP'23) introduced population protocols with unordered data (PPUD). In PPUD, each agent carries a fixed data value, and the interactions between agents depend on whether their data are equal or not. Blondin and Ladouceur also identified the interesting subclass of immediate observation PPUD (IOPPUD), where in every transition one of the two agents remains passive and does not move, and they characterised its expressive power.

We study the decidability and complexity of formally verifying these protocols. The main verification problem for population protocols is well-specification, that is, checking whether the given PPUD computes some function. We show that well-specification is undecidable in general. By contrast, for IOPPUD, we exhibit a large yet natural class of problems, which includes well-specification among other classic problems, and establish that these problems are in ExpSpace. We also provide a lower complexity bound, namely coNExpTime-hardness.

2012 ACM Subject Classification Theory of computation → Verification by model checking; Theory of computation → Distributed computing models

Keywords and phrases Population protocols, Parameterized verification, Distributed computing, Well-specification

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.156

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/2405.00921 [7]

Funding *Steffen van Bergerem*: This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 431183758 (gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 431183758).
Sandra Kiefer: This research was supported by the Glasstone Benefaction, University of Oxford [Violette and Samuel Glasstone Research Fellowships in Science 2022] as well as Jesus College in Oxford, UK.
Chana Weil-Kennedy: This work was supported by the grant PID2022-138072OB-I00, funded by MCIN, FEDER, UE and partially supported by PRODIGY Project (TED2021-132464B-I00) funded by MCIN and the European Union NextGeneration.

Acknowledgements This project started at and has benefitted substantially from the research camp Autobóz 2023 in Kassel, Germany. We would like to thank the host, sponsors, and organisers of the research camp for bringing us together.

1 Introduction

Population protocols (PP) model distributed computation and have received a lot of attention [1, 2, 9, 12, 14, 18] since their introduction in 2004 [3]. In a PP, a collection of indistinguishable mobile agents with constant-size memory communicate via pairwise interactions. When two agents meet, they exchange information about their states and update their states accordingly. The agents collectively compute whether their input configuration, i. e., the initial distribution of agents in each state, satisfies a certain predicate. For a PP to compute a predicate, the protocol must be *well-specified*, i. e., for every initial configuration, all fair runs starting in this configuration must converge to the same answer. It was shown that PP compute exactly the predicates of Presburger arithmetic [4]. Moreover, well-specification is known to be decidable but as hard as the reachability problem for Petri nets [17]. Note that deciding well-specification is a problem that concerns *parameterised verification* in the sense of [8, 15], i. e., one must decide that something holds with respect to every value of the parameter. Here the parameter is the number of agents that are present – the PP must converge to one answer for every initial configuration, no matter the number of agents.

Population protocols with unordered data (PPUD) were introduced by Blondin and Ladouceur as a means to compute predicates over arbitrarily large domains [10]. In this setting, each agent holds a read-only datum from an infinite set $\mathbb{D}$. When interacting, agents may check (dis)equality of their data. While PP can compute properties like "there are more than 5 agents in state q_1", PPUD can express, e. g., "there are more than 2 data with 5 agents each in state q_1". In [10], the authors construct a PPUD computing the absolute majority predicate, i. e., whether a datum is held by more than half of the agents. They also characterise the expressive power of *immediate observation PPUD* (IOPPUD), a subclass of interest in which interactions are restricted to observations. That is, in every interaction, one of the two agents is passive and does not change its state. The decidability and complexity of the main verification question for PPUD, namely well-specification, is left open in Blondin's and Ladouceur's article [10]. It is the subject of this paper.

Contributions. We start by showing that well-specification is *undecidable* for PPUD. This follows from a reduction from 2-counter machines; in fact, the presence of data allows us to encode zero-tests. Contrasting this, we show that deciding well-specification is in ExpSpace for IOPPUD. To this end, we define *generalised reachability expressions* (GRE) and establish that, for IOPPUD, deciding whether the set of configurations that satisfy a given GRE is empty is in ExpSpace. This decidability result is powerful; indeed, this emptiness problem subsumes classic verification problems like reachability and coverability, as well as parameterised verification problems such as well-specification and correctness, where the latter asks whether a given protocol computes a given predicate. Lastly, we exhibit a coNExpTime lower bound for deciding emptiness of GRE for IOPPUD.

Related work. For a recent survey of the research on verification of PP (without data), see [16]. In particular, the well-specification problem for PP is known to be decidable, but as hard as Petri net reachability [17] and therefore Ackermann-complete [13, 24, 25]. In their seminal paper on the computational power of PP, Angluin, Aspnes, Eisenstat, and Ruppert

also introduced five subclasses of PP that model one-way communication [4]. One of these is immediate observation population protocols (IOPP), which correspond to IOPPUD without data. The complexity of well-specification for all five subclasses is determined in [18]. In particular, the paper shows that well-specification for IOPP is PSpace-complete. IOPP were modelled by immediate observation Petri nets, where classic parameterised problems can be decided in polynomial space. The notion of generalised reachability expression was first phrased in this setting, and one of the consequences is that the emptiness problem of GRE for IOPP is PSpace-complete [29]. Our result shows that adding data to the model as in [9] (and extending GRE naturally) pushes the emptiness problem between coNExpTime and ExpSpace.

While the introduction of data in the PP model happened recently [10], a similar approach has been studied in the related model of Petri nets, under the name of *data nets*. In this setting, the classic problem of coverability (or control-state reachability) is decidable but non-primitive recursive [23] and in fact $\mathbf{F}_{\omega^\omega}$-complete [27]. While PPUD can be encoded into data nets, our results show that the problems that we study cannot be reduced to coverability. Another related model is formed by broadcast networks of register automata (BNRA) [20], an extension of reconfigurable broadcast networks (RBN) with data. RBN subsume IOPP [6], and consequently BNRA subsume IOPPUD. However, the complexity of coverability in BNRA is known to be $\mathbf{F}_{\omega^\omega}$-complete, hence non-primitive recursive, and more complex problems quickly become undecidable [20]. These hardness results contrast with the ExpSpace membership.

Organisation. In Section 2, we introduce the models of PPUD and IOPPUD, the notion of GRE, and we state our main results. We prove undecidability of well-specification for PPUD in Section 3. The next sections are dedicated to the study of IOPPUD. In Section 4, we establish bounds on the number of observed agents. In Section 5, we introduce the technical notions of boxes and containers and use the bounds from the previous section to translate GRE into containers. We present the complexity bounds for emptiness of GRE in Section 6.

2 Population Protocols and Main Results

We use the notation $[m, n] \coloneqq \{\ell \in \mathbb{N} \mid m \leq \ell \leq n\}$ for $m, n \in \mathbb{N}$ and $[m, +\infty) \coloneqq \{\ell \in \mathbb{N} \mid m \leq \ell\}$.

2.1 Population Protocols with Unordered Data

We fix an infinite *data* domain $\mathbb{D}$, an infinite set of *agents* $\mathbb{A}$, and a function $\mathbf{dat}\colon \mathbb{A} \to \mathbb{D}$ such that $\mathbf{dat}^{-1}(d)$ is infinite for all $d \in \mathbb{D}$. For $d \in \mathbb{D}$, a *d-agent* is an agent $a \in \mathbb{A}$ with $\mathbf{dat}(a) = d$.

▶ **Definition 1.** *A* population protocol with unordered data *(PPUD) is a tuple* (Q, Δ, I, O) *where Q is a finite set of* states*, $\Delta \subseteq Q^2 \times \{=, \neq\} \times Q^2$ the set of* transitions*, $I \subseteq Q$ the set of* initial states*, and $O\colon Q \to \{\top, \bot\}$ the* output function*.*

The size of a PPUD $\mathcal{P}$, written $|\mathcal{P}|$, is its number of states. We fix a PPUD (Q, Δ, I, O).

A *configuration* is a function $\gamma\colon \mathbb{A} \to Q \cup \{*\}$ such that $\gamma(a) \in Q$ only holds for finitely many agents $a \in \mathbb{A}$ (the agents *appearing* in γ). We denote by Γ the set of all configurations, and by $\Gamma_{\mathsf{init}} \coloneqq \{\gamma \in \Gamma \mid \forall a \in \mathbb{A},\, \gamma(a) \notin Q \setminus I\}$ the set of *initial configurations*. Given $\gamma \in \Gamma$ and $d \in \mathbb{D}$, we let $\gamma_d^{\#}\colon Q \to \mathbb{N}$ be the function that maps each state q to the number of d-agents in q in γ.

Given $\gamma, \gamma' \in \Gamma$, we write $\gamma \to \gamma'$, and call it a *step* from γ to γ', when there are states $q_1, q_2, q_3, q_4 \in Q$ and two distinct agents $a_1, a_2 \in \mathbb{A}$ such that $((q_1, q_2), \bowtie, (q_3, q_4)) \in \Delta$, $\gamma(a_1) = q_1$, $\gamma(a_2) = q_2$, $\gamma'(a_1) = q_3$, $\gamma'(a_2) = q_4$, $\gamma(a) = \gamma'(a)$ for all $a \in \mathbb{A} \setminus \{a_1, a_2\}$, and, additionally, if $\bowtie$ is an equality (resp. disequality), then $\mathbf{dat}(a_1) = \mathbf{dat}(a_2)$ (resp. $\mathbf{dat}(a_1) \neq \mathbf{dat}(a_2)$). A *run* ρ is a (finite or infinite) sequence of consecutive steps $\rho\colon \gamma_1 \to \gamma_2 \to \gamma_3 \to \dots$. We write $\rho\colon \gamma \xrightarrow{*} \gamma'$ to denote that ρ is a finite run from γ to γ', and simply $\gamma \xrightarrow{*} \gamma'$ to denote the existence of such a run. For every $\gamma \in \Gamma$, let $\mathsf{Post}^*(\gamma) \coloneqq \{\gamma' \in \Gamma \mid \gamma \xrightarrow{*} \gamma'\}$ and $\mathsf{Pre}^*(\gamma) \coloneqq \{\gamma' \in \Gamma \mid \gamma' \xrightarrow{*} \gamma\}$. A run ρ *covers* a state $q \in Q$ if there is a configuration γ in ρ such that $\gamma(a) = q$ for some agent a.

In accordance with [3] and [10], we consider a run $\gamma_1 \to \gamma_2 \to \dots$ *fair* if it is infinite[1] and for every configuration γ with $\left|\{i \in \mathbb{N} \mid \gamma_i \xrightarrow{*} \gamma\}\right| = \infty$, it holds that $\left|\{i \in \mathbb{N} \mid \gamma_i = \gamma\}\right| = \infty$. That is, every infinitely often reachable configuration also occurs infinitely often along the run. For $b \in \{\top, \bot\}$, a *b-consensus* is a configuration γ in which, for all agents $a \in \mathbb{A}$ appearing in γ, it holds that $O(\gamma(a)) = b$. A fair run $\rho\colon \gamma_1 \to \gamma_2 \to \cdots$ *stabilises* to $b \in \{\top, \bot\}$ if there is an $n \in \mathbb{N}$ such that for every $i \geq n$, γ_i is a b-consensus. A protocol is *well-specified* if, for every initial configuration $\gamma_0 \in \Gamma_{\mathsf{init}}$, there is $b \in \{\top, \bot\}$ such that all fair runs starting in γ_0 stabilise to b. The *well-specification problem* for PPUD asks, given a PPUD $\mathcal{P}$, whether $\mathcal{P}$ is well-specified. Given a PPUD $\mathcal{P}$ and a function $\Pi\colon \Gamma_{\mathsf{init}} \to \{\top, \bot\}$, $\mathcal{P}$ *computes* Π if, for every $\gamma_0 \in \Gamma_{\mathsf{init}}$, every fair run of $\mathcal{P}$ starting in γ_0 stabilises to $\Pi(\gamma_0)$.

▶ **Example 2.** Consider the following PPUD $\mathcal{P}$. Its set of states is $Q = \{\ell_0, \ell_1, \mathsf{f}_0, \mathsf{f}_1, \mathsf{dead}\}$, with $I = \{\ell_1\}$, $O(\ell_0) = O(\mathsf{f}_0) = \top$, $O(\ell_1) = O(\mathsf{f}_1) = O(\mathsf{dead}) = \bot$ and its transitions are:

$$\forall b, b' \in \{0,1\},\ (\ell_b, \ell_{b'}) \mapsto (\ell_{b \oplus b'}, \mathsf{f}_{b \oplus b'}) \qquad \forall b, b' \in \{0,1\},\ (\ell_b, \mathsf{f}_{b'}) \mapsto (\ell_b, \mathsf{f}_b)$$
$$\forall q, q' \in Q,\ (q, q') \mapsto_{=} (\mathsf{dead}, \mathsf{dead}) \qquad \forall q \in Q,\ (q, \mathsf{dead}) \mapsto (\mathsf{dead}, \mathsf{dead})$$

where $\mapsto_{=}$ denotes that the data of the agents must be equal, $\mapsto$ without subscript means no condition on data (or equivalently, the transition exists both for equality and disequality), and $\oplus$ denotes the XOR operator. $\mathcal{P}$ is well-specified and computes the function Π that is equal to $\top$ whenever there is an even number of appearing data and they all have exactly one corresponding agent. To see this, if there are two agents of equal datum, then all fair runs eventually have all agents on dead and stabilise to $\bot$. Otherwise, there will eventually be a single agent in $\{\ell_0, \ell_1\}$, and it will be on ℓ_b if and only if the number of agents has parity b, in which case all other agents will eventually go to f_b and the run stabilises to $\bot$ if $b = 1$ (odd number of agents) and to $\top$ if $b = 0$ (even number of agents).

A more interesting but also more complex example is the majority protocol described in [10, Section 3]; it computes whether a datum has the absolute majority, i. e., strictly more agents than all other data combined.

Well-specification is the fundamental verification problem for population protocols. However, as we will see in Section 3, this problem is undecidable for PPUD.

▶ **Theorem 3.** *The well-specification problem for PPUD is undecidable.*

This motivates the study of the restricted class of *immediate observation* PPUD.

[1] One often considers that a finite run $\gamma_f \xrightarrow{*} \gamma_\ell$ is fair when there is no γ such that $\gamma_\ell \to \gamma$. In the following, we rule out this possibility by implicitly assuming that, for all $q_1, q_2 \in Q$ and $\bowtie \in \{=, \neq\}$, it holds that $((q_1, q_2), \bowtie, (q_1, q_2)) \in \Delta$, and ignoring the trivial cases of runs with at most one agent.

2.2 Immediate Observation Protocols

Immediate observation protocols [4] are a restriction of population protocols where, when two agents interact, one of the two agents does not change its state. The restriction of the model with data to immediate observation was first considered in [10].

▶ **Definition 4.** *An* immediate observation population protocol with unordered data *(or* IOPPUD*) is a PPUD* $\mathcal{P} = (Q, \Delta, I, O)$ *where every transition* $\delta \in \Delta$ *is of the form* $(q_1, q_2, \bowtie, q_1, q_3)$*, with* $q_1, q_2, q_3 \in Q$ *and* $\bowtie \in \{=, \neq\}$*, i. e., the first agent does not change its state.*

For IOPPUD, we denote a transition $(q_1, q_2, \bowtie, q_1, q_3)$ by $q_2 \xrightarrow{\bowtie q_1} q_3$. If we have a step $\gamma \to \gamma'$ with transition $q_2 \xrightarrow{\bowtie q_1} q_3$ that involves agents $a, a_o \in \mathbb{A}$ where a is the agent moving from q_2 to q_3 and a_o is the agent in q_1, we denote it by $\gamma \xrightarrow{\bowtie a_o}_a \gamma'$. We say that agent a *observes* agent a_o, and call a_o the *observed* agent. Intuitively, a "observes" a_o and reacts, whereas a_o may not even know it has been observed.

▶ **Example 5.** Consider the following IOPPUD $\mathcal{P} = (Q, \Delta, I, O)$, with $Q \coloneqq \{q_0, q_1, q_2, q_3\}$, $I \coloneqq \{q_0, q_1\}$, $O(q_3) = \top$, $O(q) = \bot$ for all $q \neq q_3$, and transitions in Δ as follows:

$$q_0 \xrightarrow{=q_1} q_2 \quad q_1 \xrightarrow{=q_0} q_2 \quad q_2 \xrightarrow{\neq q_2} q_3 \qquad \forall q \in \{q_1, q_2\}, \forall \bowtie \in \{=, \neq\}, q \xrightarrow{\bowtie q_3} q_3$$

This protocol is well-specified: from $\gamma_0 \in \Gamma_{\mathsf{init}}$, all fair runs stabilise to $\top$ if two data have agents on both q_0 and q_1, and all fair runs stabilise to $\bot$ otherwise. Indeed, if there is a datum with agents on both q_0 and q_1, by fairness eventually an agent with this datum is sent to q_2; if there are two such data, then eventually some agent covers q_3, and then all agents are sent to q_3 and the run stabilises to $\top$. Conversely, if it is not the case, then q_3 cannot be covered and all fair runs stabilise to $\bot$.

Let $\rho\colon \gamma_{start} \xrightarrow{*} \gamma_{end}$ be a run. Agent a_o is *internally observed* (resp. *externally observed*) in ρ if ρ contains a step of the form $\gamma_1 \xrightarrow{=a_o}_a \gamma_2$ (resp. $\gamma_1 \xrightarrow{\neq a_o}_a \gamma_2$); it is *observed* if one of the two cases holds. Similarly, a datum d is *observed* in ρ if an agent a with $\mathbf{dat}(a) = d$ is observed in ρ; we define similarly a datum being internally or externally observed.

While the set of functions that can be computed by PPUD remains an open question, it is known that IOPPUD exactly compute interval predicates [10], defined as follows. Let S be a finite set. A *simple interval predicate* over S is a formula ψ of the form $\dot{\exists}\mathsf{d}_1, \ldots, \mathsf{d}_m, \bigwedge_{q \in S} \bigwedge_{j=1}^{m} \#(q, \mathsf{d}_j) \in [A_{q,j}, B_{q,j}]$ where, for all $q \in S$ and $j \in [1, m]$, we have $A_{q,j} \in \mathbb{N}$ and $B_{q,j} \in \mathbb{N} \cup \{+\infty\}$. The dotted quantifiers quantify over *pairwise distinct* data. Formally, given a protocol $\mathcal{P}$ with set of states Q such that $S \subseteq Q$ and given $\gamma \in \Gamma$, the predicate ψ is *satisfied* by γ if there exist pairwise distinct data $d_1, \ldots, d_m \in \mathbb{D}$ such that for all $q \in S$ and $j \in [1, m]$, it holds that $\gamma^{\#}_{d_j}(q) \in [A_{q,j}, B_{q,j}]$ (resp. $\gamma^{\#}_{d_j}(q) \in [A_{q,j}, B_{q,j})$ in the case that $B_{q,j} = +\infty$). An *interval predicate* over S is a Boolean combination φ of simple interval predicates over S; we define that φ is *satisfied* by a configuration γ if the simple interval predicates satisfied by γ satisfy the Boolean combination.

▶ **Theorem 6** ([10], Theorem 18 and Corollary 29)**.** *Given a finite set* I*, the functions computed by IOPPUD with set of initial states*[2] I *are exactly the interval predicates over* I*.*

▶ **Example 7.** The protocol described in Example 2 and the majority protocol of [10, Section 3] cannot be turned into immediate observation protocols, as they compute functions that cannot be expressed as interval predicates. The immediate observation protocol from Example 5 computes the following interval predicate, which is actually a simple interval predicate:

$$\dot{\exists}\mathsf{d}_1, \mathsf{d}_2, (\#(q_0, \mathsf{d}_1) \geq 1) \wedge (\#(q_1, \mathsf{d}_1) \geq 1) \wedge (\#(q_0, \mathsf{d}_2) \geq 1) \wedge (\#(q_1, \mathsf{d}_2) \geq 1).$$

[2] This does not limit the number of states of said protocols, as their set of states Q may be larger than I.

Given a simple interval predicate $\psi = \dot{\exists}\mathsf{d}_1, \ldots, \mathsf{d}_m, \bigwedge_{q \in I} \bigwedge_{j=1}^{m} \#(q, \mathsf{d}_j) \in [A_{q,j}, B_{q,j}]$, we define its *width* as m, its *height* h as the maximum of all finite $A_{q,j}$ and $B_{q,j}$, and its *size* as $|I| \cdot m \cdot \mathsf{log}(h)$. We also define the *width* (resp. *height*) of an interval predicate as the maximum of the widths (resp. heights) of its simple interval predicates, and its *size*, measuring the space taken by its encoding, as the sum of their sizes plus its number of Boolean operators.

▶ Remark 8. In [10], predicates refer to an input alphabet Σ, which is converted into initial states using an input mapping. For convenience, we have not included the input alphabet in our model, which is why we arbitrarily fix a set of initial states I in Theorem 6.

2.3 Generalised Reachability Expressions

We define a general class of specifications, called generalised reachability expressions, which are formulas constructed using interval predicates as atoms and using union, complement, Post^*, and Pre^* as operators. This concept is inspired by [29, Section 2.4], although our choice of atoms is more general and adapted to the data setting.

▶ **Definition 9.** *Let $\mathcal{P} = (Q, \Delta, I, O)$ be a protocol.*
Generalised Reachability Expressions *(GRE) over $\mathcal{P}$ are produced by the grammar*

$$E ::= \varphi \mid E \cup E \mid \overline{E} \mid \mathsf{Post}^*(E) \mid \mathsf{Pre}^*(E),$$

where φ ranges over interval predicates over Q.

Given a GRE E, we define the set of configurations defined by E, denoted $[\![E]\!]_{\mathcal{P}}$, as the set containing all configurations of $\mathcal{P}$ that satisfy the formula, where the predicates are interpreted as above and the other operators are interpreted naturally (the overline denotes set complementation). This set is denoted $[\![E]\!]$ when $\mathcal{P}$ is clear from context.

The *length* $|E|$ of a GRE E is its number of operators. Letting $\varphi_1, \ldots, \varphi_k$ be the interval predicates used as atoms in E, the *norm* $||E||$ of E is the maximum of the heights and widths of the φ_i. Its *size* is the sum of the sizes of the φ_i plus $|E|$. The *emptiness problem for GRE* asks, given as input a protocol $\mathcal{P}$ and a GRE E over $\mathcal{P}$, whether $[\![E]\!]_{\mathcal{P}} = \emptyset$. We will show in Section 6 that, for IOPPUD, this problem is decidable.

▶ **Theorem 10.** *The emptiness problem for GRE over IOPPUD is in ExpSpace.*

We now argue that this decidability result is powerful, as it implies decidability of many classic problems on IOPPUD. We start with well-specification. We use the notation $\dot{\forall}\mathsf{d}, \varphi$ as a short form for $\neg\dot{\exists}\mathsf{d}, \neg\varphi$. Given a PPUD $\mathcal{P}$ and $b \in \{\top, \bot\}$, let $\mathsf{Out}_b := \dot{\forall}\mathsf{d}, \bigwedge_{q \notin O^{-1}(\{b\})} \#(q, \mathsf{d}) = 0$ be the GRE for b-consensus configurations; moreover, let $\mathsf{Stable}_b := \overline{\mathsf{Pre}^*(\overline{\mathsf{Out}_b})}$ be the GRE for *stable* b-consensus, i. e., configurations from which all runs lead to a b-consensus.

▶ **Proposition 11.** *Let $\mathcal{P}$ be a PPUD, $E_{ws} := \Gamma_{\mathsf{init}} \cap \mathsf{Pre}^*(\overline{\mathsf{Pre}^*(\mathsf{Stable}_\top)}) \cap \mathsf{Pre}^*(\overline{\mathsf{Pre}^*(\mathsf{Stable}_\bot)})$. $\mathcal{P}$ is well-specified if and only if $[\![E_{ws}]\!]_{\mathcal{P}} = \emptyset$.*

Proof. First, $\Gamma_{\mathsf{init}} = [\![\dot{\forall}\mathsf{d}, \bigwedge_{q \notin I} \#(q, d) = 0]\!]$ and E_{ws} is indeed a GRE over $\mathcal{P}$. For every $\gamma \in \Gamma$, $\mathsf{Post}^*(\gamma)$ is finite as all configurations reachable from γ have the same number of agents. Therefore, a fair run ρ that visits $\mathsf{Pre}^*(\mathcal{S})$ infinitely often for $\mathcal{S} \subseteq \Gamma$ must visit $\mathcal{S}$ infinitely often. Let $\gamma_0 \in \Gamma_{\mathsf{init}}$ and $b \in \{\top, \bot\}$; it suffices to prove that there is a fair run from γ_0 that does *not* stabilise to b if and only if $\gamma_0 \in [\![\mathsf{Pre}^*(\overline{\mathsf{Pre}^*(\mathsf{Stable}_b)})]\!]$. If, from γ_0, one can reach $\gamma \notin [\![\mathsf{Pre}^*(\mathsf{Stable}_b)]\!]$, then one can build a fair run from γ_0 that first goes to γ, and

then forever performs arbitrary steps in a fair way; since $\gamma \notin [\![\mathsf{Pre}^*(\mathsf{Stable}_b)]\!]$, it will stay in $[\![\mathsf{Pre}^*(\overline{\mathsf{Out}_b})]\!]$, so by fairness it visits $[\![\overline{\mathsf{Out}_b}]\!]$ infinitely often and does not stabilise to b. Conversely, if there is a fair run that does not stabilise to b, then it never visits $[\![\mathsf{Stable}_b]\!]$, hence by fairness it eventually stops visiting $[\![\mathsf{Pre}^*(\mathsf{Stable}_b)]\!]$; this proves that it visits a configuration $\gamma \in [\![\overline{\mathsf{Pre}^*(\mathsf{Stable}_b)}]\!]$, and γ is reachable from γ_0 hence $\gamma_0 \in [\![\mathsf{Pre}^*(\overline{\mathsf{Pre}^*(\mathsf{Stable}_b)})]\!]$. ◀

Many other problems can be expressed as emptiness problems for GRE; we list a few.

- The *correctness* problem for IOPPUD asks, given an IOPPUD $\mathcal{P} = (Q, I, O, \Delta)$ and an interval predicate φ over I, whether $\mathcal{P}$ computes φ. This can be equivalently phrased as $[\![\varphi^{-1}(b) \cap \mathsf{Pre}^*(\overline{\mathsf{Pre}^*(\mathsf{Stable}_b)})]\!] = \emptyset$ for all $b \in \{\top, \bot\}$, where $\varphi^{-1}(b)$ is the set of initial configurations that φ maps to b. Note that the previous expression is a GRE because, in Definition 9, we chose as atoms interval predicates such as φ.
- The *set-reachability* problem (called cube-reachability in [5]) asks, given two sets of configurations $\mathcal{S}_1, \mathcal{S}_2$, whether $\mathcal{S}_2$ is reachable from $\mathcal{S}_1$; this typically expresses safety problems where $\mathcal{S}_2$ represents "bad configurations" that must not be reached. If $\mathcal{S}_1 = [\![E_1]\!]$ and $\mathcal{S}_2 = [\![E_2]\!]$, then this amounts to checking whether $[\![E_1 \cap \mathsf{Pre}^*(E_2)]\!]$ is empty.
- The *home-space problem* asks, given a protocol $\mathcal{P}$ and a set of configurations $\mathcal{H}$, whether $\mathcal{H}$ can be reached from every configuration reachable from an initial configuration. If $\mathcal{H}$ can be expressed as a GRE E, then it suffices to check whether $[\![\mathsf{Post}^*(\Gamma_{\mathsf{init}})]\!] \subseteq [\![\mathsf{Pre}^*(E)]\!]$. This problem has been studied in Petri nets [22], but also in probabilistic settings, for example in [11] for asynchronous shared-memory systems; indeed, in systems with uniform probabilistic schedulers where $\mathsf{Post}^*(\gamma_0)$ is finite for every initial configuration γ_0, this problem is equivalent to asking whether the probability of reaching $\mathcal{H}$ is equal to 1.

Theorem 10 entails that, for IOPPUD, all these problems are decidable and in ExpSpace.

3 Undecidability of Verification of Population Protocols with Unordered Data

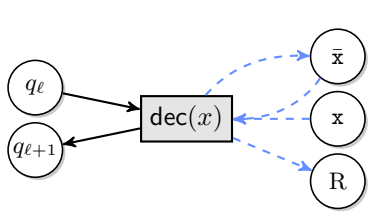
In this section, we establish that the most fundamental verification problem for PPUD, i. e., the well-specification problem, is already undecidable.

▶ **Theorem 3.** *The well-specification problem for PPUD is undecidable.*

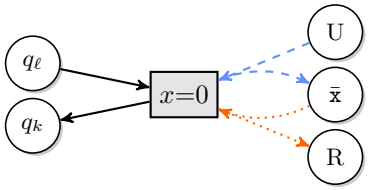
We proceed by reduction from the halting problem for 2-counter machines with zero-tests, a famously undecidable problem [26]. Here, we give a proof sketch. The detailed reduction can be found in [7], which is the full version of this paper.

We fix a 2-counter machine and build a protocol $\mathcal{P}$ which is *not* well-specified if and only if the counter machine halts. A 2-*counter machine* performs increments, decrements and zero-tests on two counters. The main difficulty are the zero-tests. Let us first recall how increments and decrements are simulated in many prior undecidability results for population protocols and Petri nets [21, 27]. The protocol has a control part $Q_{CM} := \{q_i \mid i \in [1, n]\}$ where a single *instruction agent* evolves; this part has one state per instruction of the machine. Increments and decrements are simulated as follows: The instruction agent interacts with states of $\{R\} \cup \{\mathtt{x}, \mathtt{y}\}$, where R is a *reservoir state* and $\mathtt{x}$ and $\mathtt{y}$ are states in which the number of agents represents the value of the counters. For example an increment on counter x moves one agent from the reservoir R to $\mathtt{x}$ and advances the instruction agent to the next instruction. The reservoir is hence implicitly assumed to start with arbitrarily many agents.

The main difficulty is that one does not want to take the $= 0$ branch of a zero-test when the value of the counter is non-zero. Actually, similar to [21, 27], we will not prevent the existence of such runs. Instead, our protocol will have "violating" runs which take the wrong



(a) Gadget for performing dec(x). Agents of the blue dashed edges must have the same datum.



(b) Gadget for x=0 branch of "`if` x=0 `then goto` k". The same agent takes the two blue dashed edges and the same agent takes the orange dotted edges.

Figure 1 For simplicity, we use Petri net notation: circles are states, rectangles are Petri net transitions. To encode this into our protocols, we split each transition into pairwise interactions.

branch of a zero-test, but our well-specification check will consider only *violation-free* runs. The correctness of the reduction is then established in two steps: The CM halts if and only if some *violation-free* run to the halting state exists, and this is true if and only if our protocol is not well-specified. We establish the connection between non-well-specification and the existence of a *violation-free* run in our protocol.

In the first place, we guarantee that every initial configuration of our protocol has a fair run stabilising to $\bot$, so that $\mathcal{P}$ is not well-specified if and only if there exists a fair run which does not stabilise to $\bot$[3]. Second, we introduce *violation detection*, a mechanism which guarantees that *fair* runs which contain a violation stabilise to $\bot$, hence preserving well-specification. To do so, we add a sink state $q_\bot$, which has output $\bot$ and is attracting, i. e., all other states have a transition to $q_\bot$ available when observing that $q_\bot$ is non-empty. Violation detection then entails adding transitions into $q_\bot$ that will be available infinitely often if the run (or its initial configuration) contained a violation. By fairness, any run containing a violation will then eventually put an agent into $q_\bot$, and hence, because $q_\bot$ is attracting, the run ends in a deadlock with all agents in $q_\bot$. In particular, any fair run containing a violation will output $\bot$ as claimed. There are two types of violation detection.

First, we want to only mark those runs as violation-free that start in initial configurations where $U \in Q$ has at most one agent of each datum. To do so, we make agents remember whether their initial state was U or not (by encoding it into the state space), and, from every state, we add a transition to $q_\bot$ such that this transition is enabled when an agent who started in U observes another agent of same datum that also started in U.

Second, we want to detect violations which consist in falsely simulating a zero-test, as discussed above. Here our technique shares some similarities with [27]. Let $c \in \{x, y\}$ be a counter; for every zero-test of the counter machine, we add two types of transitions to the protocol. The first type simulates the $c \neq 0$ branch and can be taken by the instruction agent upon interacting with some agent on state $\mathtt{c}$; by contrast, the $c = 0$ branch can always be taken. However, if it was taken with $c \neq 0$, then violation detection will eventually detect this. For this mechanism, we introduce a counter control state $\bar{\mathtt{c}} \in Q$. At any point in time, $\bar{\mathtt{c}}$ contains one agent, similar to the instruction agent. The crux of our violation detection is that only agents which share the datum with the agent in $\bar{\mathtt{c}}$ will be allowed to move in and out of state $\mathtt{c}$, as illustrated in Figure 1a.

The $= 0$ branch of a zero-test is depicted in Figure 1b. It replaces the agent on $\bar{\mathtt{c}}$ with an agent with fresh datum from state U. Thus, when the $c = 0$ branch is taken, any remaining agent in $\mathtt{c}$ is stuck in $\mathtt{c}$ as it will never again share datum with the agent in $\bar{\mathtt{c}}$. Violation detection then sends an agent in $\mathtt{c}$ to $q_\bot$ upon observing an agent in $\bar{\mathtt{c}}$ with different datum.

[3] This can be done with the addition of a fresh state that is the only initial state and that has internal transitions to all former initial states and an internal transition to a sink state that has output $\bot$.

Now that we have violation detection in place, it only remains to explain the connection to halting. The halting instruction q_n in Q_{CM} is the only state with output $\top$. Hence, any run not outputting $\bot$ must contain an agent in the halting instruction at some point, and be violation-free by the above. That is, the counter machine reached the halting state without violations. Conversely, if the machine halts, one can build a finite run that puts an agent into the halting state without any violation occurring. The corresponding configuration is then a deadlock, and hence the extension to an infinite run (by staying there forever) is a fair run not outputting $\bot$. This proves that well-specification is undecidable for PPUD, which motivates restricting ourselves to immediate observation PPUD.

4 An Analysis of Immediate Observation Protocols with Data

To obtain our complexity bounds on the emptiness problem for GRE, we first show some transformations on runs that allow us to bound the number of observed agents. All runs that we consider in this section are finite, and we therefore write them as $\gamma_1 \to \cdots \to \gamma_m$ or $\gamma_{start} \xrightarrow{*} \gamma_{end}$. In the rest of this section, we fix an IOPPUD $\mathcal{P} = (Q, \Delta, I, O)$.

We introduce some notation for agents in runs. Let $\rho\colon \gamma_1 \xrightarrow{*} \gamma_m$ and $d \in \mathbb{D}$. We let $\mathbb{A}_\rho$ be the set of agents appearing in ρ, and set $\mathbb{A}_\rho^d := \{a \in \mathbb{A}_\rho \mid \mathbf{dat}(a) = d\}$. We let $\mathbb{A}_{\rho,o}^d$ be the set of agents with datum d that are observed in ρ, i. e., the $a_o \in \mathbb{A}_\rho^d$ such that there exists a step $\gamma \xrightarrow{\bowtie a_o}_a \gamma'$ in ρ. For all $q_1, q_m \in Q$, we let $\mathbb{A}_{\rho,q_1,q_m}^d$ be the set of agents with datum d that start in q_1 and end in q_m, i. e., the $a \in \mathbb{A}_\rho^d$ such that $\gamma_1(a) = q_1$ and $\gamma_m(a) = q_m$. Moreover, we let $\mathbb{D}_\rho := \{d \in \mathbb{D} \mid \mathbb{A}_\rho^d \neq \emptyset\}$ be the set of data appearing in ρ. We may omit ρ in the subscript if the run is clear from the context.

4.1 Bounds on the Number of Observed Agents per Datum

Let $\rho\colon \gamma_1 \to \cdots \to \gamma_m$ be a run. For $i \in [1, m]$, we call $\gamma_i \to \gamma_{i+1}$ the *i-th step* in ρ. Let $\rho[\to i]$ (resp. $\rho[i \to]$) denote the prefix of ρ ending on its i-th configuration (resp. the suffix of ρ starting on its i-th configuration). Let $a, b \in \mathbb{A}_\rho$. Agent a is *active* in the i-th step if $\gamma_i \xrightarrow{\bowtie a_o}_a \gamma_{i+1}$ for some agent a_o. Otherwise, a is *idle* in that step. We say b *copies* a in ρ if after every step $\gamma_i \xrightarrow{\bowtie a_o}_a \gamma_{i+1}$ in ρ via some transition t, there is a step $\gamma_{i+1} \xrightarrow{\bowtie a_o}_b \gamma_{i+2}$ via t and, additionally, b is idle in every step not immediately following an active step of a.

The following lemma allows us to add agents to a run that copy an agent of the same datum.

▶ **Lemma 12** (Agents copycat). *Let $\rho\colon \gamma_{start} \xrightarrow{*} \gamma_{end}$ be a run. Let $a \in \mathbb{A}_\rho$ and $\tilde{a} \in \mathbb{A} \setminus \mathbb{A}_\rho$ with $\mathbf{dat}(a) = \mathbf{dat}(\tilde{a})$. Then there exist configurations $\tilde{\gamma}_{start}, \tilde{\gamma}_{end}$ and a run $\tilde{\rho}\colon \tilde{\gamma}_{start} \xrightarrow{*} \tilde{\gamma}_{end}$ such that:*

- **(i)** *$\mathbb{A}_{\tilde{\rho}} = \mathbb{A}_\rho \uplus \{\tilde{a}\}$, and for all $a' \in \mathbb{A}_\rho$, $\tilde{\gamma}_{start}(a') = \gamma_{start}(a')$ and $\tilde{\gamma}_{end}(a') = \gamma_{end}(a')$;*
- **(ii)** *$\tilde{\gamma}_{start}(\tilde{a}) = \gamma_{start}(a)$ and $\tilde{\gamma}_{end}(\tilde{a}) = \gamma_{end}(a)$;*
- **(iii)** *$\tilde{a}$ is not observed in $\tilde{\rho}$.*

Proof. We let $\tilde{\gamma}_{start}$ be such that $\tilde{\gamma}_{start}(\tilde{a}) = \gamma_{start}(a)$ and $\tilde{\gamma}_{start}(a') = \gamma_{start}(a')$ for all $a' \neq \tilde{a}$. We construct $\tilde{\rho}$ by going through ρ step by step, making $\tilde{a}$ copy a: whenever ρ takes a step $\xrightarrow{\bowtie a_o}_a$, then we take this step followed by step $\xrightarrow{\bowtie a_o}_{\tilde{a}}$ to $\tilde{\rho}$. We can do so because $\mathbf{dat}(\tilde{a}) = \mathbf{dat}(a)$ and because agent $a_o \neq a$ has not moved and thus can be observed again. These are the only steps where $\tilde{a}$ is involved, hence it is never observed. ◀

The following result shows that, given a run ρ, we can construct a new run with a small subset of the agents of $\mathbb{A}_\rho$ such that, for all $d \in \mathbb{D}$ and all states q_1 and q_2, if there is a d-agent starting in q_1 and ending in q_2 in ρ, then this is also true in the new run. We refer to [7] for proof details.

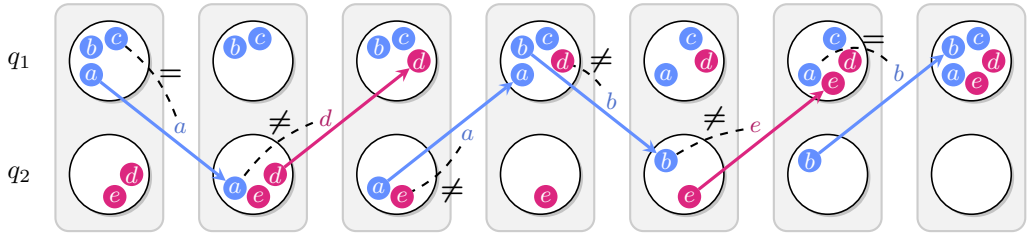


Figure 2 An example of a run with six steps on a protocol with two states q_1, q_2. $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}$ denote agents; $\mathsf{a}, \mathsf{b}, \mathsf{c}$ have the same datum and d, e have the same datum. Dashed lines are observations.

▶ **Lemma 13** (Agents core). *Let $\rho\colon \gamma_{start} \xrightarrow{*} \gamma_{end}$ be a run. Then there exist configurations γ'_{start}, γ'_{end} and a run $\rho'\colon \gamma'_{start} \xrightarrow{*} \gamma'_{end}$ with $\mathbb{A}_{\rho'} \subseteq \mathbb{A}_\rho$ such that:*

(i) *for all $a \in \mathbb{A}_{\rho'}$, $\gamma'_{start}(a) = \gamma_{start}(a)$ and $\gamma'_{end}(a) = \gamma_{end}(a)$;*
(ii) *for all $d \in \mathbb{D}$ and $q_s, q_e \in Q$, if $\mathbb{A}^d_{\rho,q_s,q_e} \neq \emptyset$, then $\mathbb{A}^d_{\rho',q_s,q_e} \neq \emptyset$;*
(iii) *for all $d \in \mathbb{D}$, we have $|\mathbb{A}^d_{\rho'}| \leq |Q|^3$.*

Proof sketch. We adapt the bunch argument from the case of IO protocols without data [19]. Suppose there is $d \in \mathbb{D}$ and $q_s, q_e \in Q$ such that $|\mathbb{A}^d_{\rho,q_s,q_e}| > |Q|$. Let $\mathcal{R}$ be the set of states visited by agents of $\mathbb{A}^d_{\rho,q_s,q_e}$ in ρ. Notice that $|\mathcal{R}| \leq |Q|$. We define a family $(a_q)_{q\in\mathcal{R}}$ of pairwise distinct agents such that reducing $\mathbb{A}^d_{\rho,q_s,q_e}$ in ρ to $(a_q)_{q\in\mathcal{R}}$ still yields a valid run.

We iterate through $\mathcal{R}$ as follows. Let $q \in \mathcal{R}$ and let f be the first moment q is reached in ρ, i. e., the minimal index such that there exists an $a \in \mathbb{A}^d_{\rho,q_s,q_e}$ with $\gamma_f(a) = q$. Let ℓ be the last moment q is occupied in ρ, i. e., the maximal index such that there exists an $a \in \mathbb{A}^d_{\rho,q_s,q_e}$ with $\gamma_\ell(a) = q$. Let α_q be the agent in $\mathbb{A}^d_{\rho,q_s,q_e}$ that reaches q first, i. e., $\gamma_f(\alpha_q) = q$, and let β_q be the agent in $\mathbb{A}^d_{\rho,q_s,q_e}$ that leaves q last, i. e., $\gamma_l(\beta_q) = q$. Note that these agents do not have to be distinct. We pick a fresh agent $a_q \notin \mathbb{A}_\rho$ with $\mathbf{dat}(a_q) = d$ and modify ρ as follows. We let a_q copy α_q in $\rho[\to f]$, then a_q stays idle until β_q leaves q (for the last time) and then a_q copies β_q in $\rho[\ell \to]$. We do this for every $q \in \mathcal{R}$.

Then, for every step in which an a_o in $\mathbb{A}^d_{\rho,q_s,q_e}$ is observed in state q, let a_q be observed instead, i. e., replace steps $\xrightarrow{\bowtie a_o}_a$ with $\xrightarrow{\bowtie a_q}_a$. Finally, remove all the agents of $\mathbb{A}^d_{\rho,q_s,q_e}$ from the run, and identify (or substitute) each a_q with a distinct agent in $\mathbb{A}^d_{\rho,q_s,q_e}$, so that $(a_q)_{q\in\mathcal{R}} \subseteq \mathbb{A}^d_{\rho,q_s,q_e}$. We do this for every $d \in \mathbb{D}$ and $q_s, q_e \in Q$ such that $|\mathbb{A}^d_{\rho,q_s,q_e}| > |Q|$. ◀

▶ **Example 14.** Consider the run ρ depicted in Figure 2. Applying Lemma 13 on ρ yields a new run ρ' with 4 agents instead of 5. Indeed, let d denote the datum of a, b and c; we have $|\mathbb{A}^d_{\rho,q_1,q_1}| = |\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}| = 3$ whereas $|Q| = 2$. In ρ, agents a and b successively go from q_1 to q_2 and back to q_1. In ρ', these two agents are replaced by a single agent (named b again) who goes to q_2 on the first step and only leaves q_2 on the last step. In ρ', the new agent b is observed by d in the second step, and by e in the penultimate step.

4.2 Bounds on the Number of Observed Data

Given a run and a datum d appearing in it, we define the *trace* of d in ρ as the function $\mathtt{tr}^d_\rho\colon Q^2 \to \mathbb{N}$ such that for all $q_1, q_2 \in Q$, it holds that $\mathtt{tr}^d_\rho(q_1, q_2) = |\mathbb{A}^d_{\rho,q_1,q_2}|$. For each pair of states q_1, q_2, the trace counts the number of d-agents starting in q_1 and ending in q_2. For example, the trace of the run ρ of Example 14, with d the datum of agents a, b and c, is such that $\mathtt{tr}^d_\rho(q_1, q_1) = 3$ and $\mathtt{tr}^d_\rho(q, q') = 0$ for all $(q, q') \neq (q_1, q_1)$. The trace is the information we need to copy data: if there is a datum d with trace tr in a run, then we can add data to the run that mimic d and have the same trace. The following lemma echoes Lemma 12.

▶ **Lemma 15** (Data copycat). *Let $\rho\colon \gamma_{start} \xrightarrow{*} \gamma_{end}$ be a run. Let $d \in \mathbb{D}_\rho$ and $\tilde{d} \in \mathbb{D} \setminus \mathbb{D}_\rho$. Then there exist configurations $\tilde{\gamma}_{start}$, $\tilde{\gamma}_{end}$ and a run $\tilde{\rho}\colon \tilde{\gamma}_{start} \xrightarrow{*} \tilde{\gamma}_{end}$ such that:*

(i) *$\mathbb{A}_{\tilde{\rho}} = \mathbb{A}_\rho \uplus \mathbb{A}^{\tilde{d}}_{\tilde{\rho}}$, and for all $a \in \mathbb{A}_\rho$, $\tilde{\gamma}_{init}(a) = \gamma_{init}(a)$ and $\tilde{\gamma}_{end}(a) = \gamma_{end}(a)$,*

(ii) *$\mathtt{tr}^{\tilde{d}}_{\tilde{\rho}} = \mathtt{tr}^{d}_{\rho}$ and $\mathtt{tr}^{d'}_{\tilde{\rho}} = \mathtt{tr}^{d'}_{\rho}$ for all $d' \neq \tilde{d}$,*

(iii) *$\tilde{d}$ is not externally observed in $\tilde{\rho}$.*

Proof. For all $q_s, q_e \in Q$ and all $a \in \mathbb{A}^d_{q_s,q_e}$, we add an agent $\tilde{a}$ with datum $\tilde{d}$ in q_s at the start. We do this in a way similar to Lemma 12: after every step $\xrightarrow{\neq a_o}_a$ in ρ, we insert a step $\xrightarrow{\neq a_o}_{\tilde{a}}$, and after every step $\xrightarrow{=a_o}_a$ in ρ, we insert a step $\xrightarrow{=\tilde{a}_o}_{\tilde{a}}$. We thus maintain the fact that each added agent $\tilde{a}$ is in the same state as its counterpart a. In particular, they are in the same state at the end of the run. This yields a run $\tilde{\rho}$ with $\mathtt{tr}^{\tilde{d}}_{\tilde{\rho}} = \mathtt{tr}^{d}_{\rho}$, and such that for all $d' \neq \tilde{d}$, $\mathtt{tr}^{d'}_{\tilde{\rho}} = \mathtt{tr}^{d'}_{\rho}$. Since $\tilde{d} \notin \mathbb{D}_\rho$, it is not externally observed in $\tilde{\rho}$. ◀

Like we showed for the agents, we show that we can reduce the number of data in a run. We lift the proof strategy of Lemma 13 from agents to data, exploiting the sets of data with equal traces. We refer to [7] for proof details.

▶ **Lemma 16** (Data core). *Let $\rho\colon \gamma_{start} \xrightarrow{*} \gamma_{end}$ be a run and let K be a number such that there are at most K agents of each datum in ρ. Then there exist configurations γ'_{start}, γ'_{end}, a run $\rho'\colon \gamma'_{start} \xrightarrow{*} \gamma'_{end}$, and a subset of data $\mathbb{D}_{\rho'} \subseteq \mathbb{D}_\rho$ such that:*

(i) *for all $d \in \mathbb{D}_{\rho'}$ and all agents a of datum d, $\gamma_{start}(a) = \gamma'_{start}(a)$ and $\gamma_{end}(a) = \gamma'_{end}(a)$,*

(ii) *for all $d \in \mathbb{D}_\rho$, there exists $d' \in \mathbb{D}_{\rho'}$ such that $\mathtt{tr}^{d'}_{\rho'} = \mathtt{tr}^{d}_{\rho}$,*

(iii) *$|\mathbb{D}_{\rho'}| \leq (K+1)^{|Q|^3+|Q|^2}$.*

Proof sketch. We define the notion of split trace. The split trace of a datum d at the i-th configuration of a run ρ maps every triple of states (q_1, q_2, q_3) to the number of d-agents that are in q_1 at the start of ρ, then in q_2 in the i-th configuration, and finally in q_3 at the end. Since there are at most K agents per datum, there are at most $(K+1)^{|Q^2|}$ possible traces and $M = (K+1)^{|Q^3|}$ possible split traces.

For every trace tr, if there are more than M data that have trace tr in ρ, we apply a similar argument to Lemma 13: we select one datum for each possible split trace, and use it to cover all external observations of agents whose datum matches that split trace. We remove the other data, and show that this is still a valid run. The bound on the total number of data comes from the number of traces and split traces. ◀

▶ **Corollary 17.** *For every run $\rho\colon \gamma_{start} \xrightarrow{*} \gamma_{end}$, there exists a run $\tilde{\rho}\colon \gamma_{start} \xrightarrow{*} \gamma_{end}$ such that for all $d \in \mathbb{D}$, it holds that $|\mathbb{A}^d_{\tilde{\rho},o}| \leq |Q|^3$ and that agents of at most $(|Q|^3+1)^{|Q|^3+|Q|^2}$ different data are externally observed.*

Proof. We first apply Lemma 13 to ρ to obtain $\rho^{(1)}\colon \gamma^{(1)}_{start} \xrightarrow{*} \gamma^{(1)}_{end}$ over the same data such that for all $d \in \mathbb{D}$, it holds that $|\mathbb{A}^d_{\rho^{(1)}}| \leq |Q|^3$. Then we apply Lemma 16 to obtain $\rho^{(2)}\colon \gamma^{(2)}_{start} \xrightarrow{*} \gamma^{(2)}_{end}$ with at most $(|Q|^3+1)^{|Q|^3+|Q|^2}$ data. By Lemma 13-(i) and Lemma 16-(i), the remaining agents have the same initial and final states in ρ and $\rho^{(2)}$. It remains to put back the agents and data we removed, without increasing the number of externally observed data or observed agents per datum.

By Lemma 16-(ii), every trace of a datum in $\rho^{(1)}$ appears as the trace of a datum in $\rho^{(2)}$. Thus, it is possible to re-add data of $\mathbb{D}_{\rho^{(1)}}$ to $\mathbb{D}_{\rho^{(2)}}$ using repeated applications of Lemma 15. By Lemma 15-(iii), this does not add any external observation. So we obtain a run $\tilde{\rho}^{(1)}$ from

$\gamma^{(1)}_{start}$ to $\gamma^{(1)}_{end}$ such that at most $(|Q|^3+1)^{|Q|^3+|Q|^2}$ data are externally observed by Lemma 15-(iii). Recall that there are at most $|Q|^3$ agents per datum in $\gamma^{(1)}_{start}$ by Lemma 13-(iii); in particular there are at most $|Q|^3$ observed agents per datum in $\tilde{\rho}^{(1)}$.

By Lemma 13-(ii), for each datum d and states q_s, q_e, if there is a d-agent a such that $\gamma_{start}(a) = q_s$ and $\gamma_{end}(a) = q_e$ then there is an agent a' such that $\gamma^{(1)}_{start}(a') = q_s$ and $\gamma^{(1)}_{end}(a') = q_e$ in ρ. Therefore, due to Lemma 13-(ii), we can apply Lemma 12 repeatedly to add back the missing agents in $\tilde{\rho}^{(1)}$ and obtain a run $\tilde{\rho}$ from γ_{start} to γ_{end}. By Lemma 12-(iii), this does not add any observation. As a result, we obtain a run from γ_{start} to γ_{end} in which at most $(|Q|^3+1)^{|Q|^3+|Q|^2}$ data are externally observed and for all datum d, at most $|Q|^3$ d-agents are observed. ◀

5 From Expressions to Containers

In this section, we define the technical notions of boxes and containers, which are meant to represent sets of configurations defined by counting agents and data up to some thresholds. In Proposition 21, we will prove that the set of configurations defined by a generalised reachability expression E can be described as a union of containers whose thresholds are exponential in the length of E and polynomial in its norm. To do so, we will leverage the bounds on the number of observed agents from Section 4 to bound the description of the GRE $\mathsf{Post}^*(F)$ with respect to the one of GRE F. The key result of Proposition 21 will be used in Section 6 to obtain the decidability of the emptiness problem for GRE.

5.1 Equivalence of Predicates and Containers

In this subsection, we fix an IOPPUD $\mathcal{P} = (Q, \Delta, I, O)$.

Let $n, M \in \mathbb{N}$. An *n-box* is a vector $\mathsf{b}\colon Q \to [0, n]$. Given a configuration γ and a datum $d \in \mathbb{D}$, we define *the n-box of d in γ* as $\lceil \gamma, d \rceil^n \colon Q \to [0, n]$ such that for all $q \in Q$, $\lceil \gamma, d \rceil^n(q) = \min\{n, \gamma^{\#}_d(q)\}$; in words, the n-box of d truncates the number of agents of d if it exceeds n. We write $\mathbf{Boxes}_n$ for the set of all n-boxes. We define the equivalence relation $\equiv_n$ over $\Gamma \times \mathbb{D}$ by $(\gamma_1, d_1) \equiv_n (\gamma_2, d_2)$ whenever $\lceil \gamma_1, d_1 \rceil^n = \lceil \gamma_2, d_2 \rceil^n$. An equivalence class of $\equiv_n$ is a set of the form $\{(\gamma, d) \in \Gamma \times \mathbb{D} \mid \lceil \gamma, d \rceil^n = \mathsf{b}\}$ for $\mathsf{b} \in \mathbf{Boxes}_n$; we represent such an equivalence class for $\equiv_n$ by the associated n-box b.

To lift this concept to data, we count the number of data with the same n-box up to bound M. The *(n, M)-container* of a configuration γ is the function $\lceil \gamma \rceil^{n,M} \colon \mathbf{Boxes}_n \to [0, M]$ such that $\lceil \gamma \rceil^{n,M}(\mathsf{b}) = \min\{M, |\{d \in \mathbb{D} \mid \lceil \gamma, d \rceil^n = \mathsf{b}\}|\}$ for all $\mathsf{b} \in \mathbf{Boxes}_n$. We define the equivalence relation $\equiv_{n,M}$ over Γ by $\gamma_1 \equiv_{n,M} \gamma_2$ whenever $\lceil \gamma_1 \rceil^{n,M} = \lceil \gamma_2 \rceil^{n,M}$. An equivalence relation for $\equiv_{n,M}$ is the preimage of some (n, M)-container by the previously described function; we represent such an equivalence class by the associated (n, M)-container. Figure 3 illustrates the function mapping a given configuration to its container.

> In all the following, we use the terms n-boxes and (n, M)-containers to designate both the vectors and the equivalence classes of $\equiv_n$ and $\equiv_{n,M}$ that they represent. For instance, we write *union of n-boxes* for the union of the corresponding equivalence classes of $\equiv_n$.

The partition of Γ into (n, M)-containers becomes finer as n and M grow.

▶ **Lemma 18.** *Let $n_1, n_2, M_1, M_2 \in \mathbb{N}$. If $n_1 \leq n_2$ and $M_1 \leq M_2$, then every (n_1, M_1)-container is a union of (n_2, M_2)-containers.*

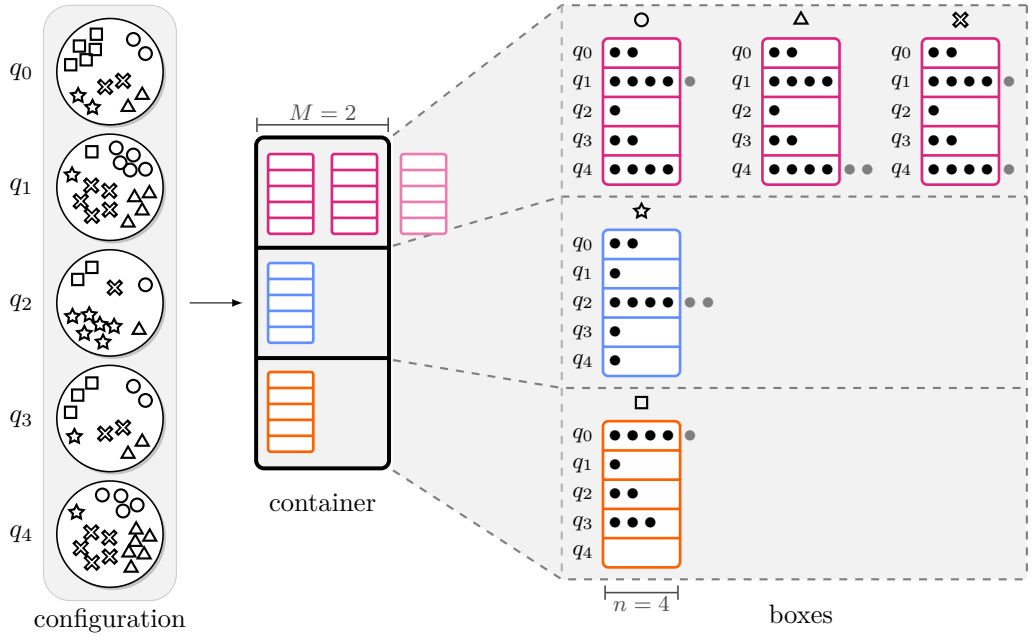


Figure 3 How a configuration is mapped to a $(4, 2)$-container. Here, the protocol has five states $q_0, \dots, q_4$. Five distinct data appear in the configuration, and they are represented using symbols.

Algorithmically, we represent an n-box as a list of appearing states with associated numbers from $[1, n]$ encoded in binary. Similarly, we represent an (n, M)-container as a list of appearing n-boxes with associated numbers from $[1, M]$ encoded in binary.

In fact, interval predicates exactly describe finite unions of containers.

▶ **Proposition 19.** *The sets of configurations defined by interval predicates of height at most n and width at most M are exactly the sets formed by unions of (n, M)-containers.*

Proof sketch. For the translation from predicates to containers, consider a simple interval predicate $\dot{\exists}\mathsf{d}_1, \dots, \mathsf{d}_M, \bigwedge_{q\in Q}\bigwedge_{j=1}^{M} \#(q, \mathsf{d}_j) \in [A_{q,j}, B_{q,j}]$ of height n. This predicate cannot distinguish data mapped to the same n-box, hence cannot distinguish configurations in the same equivalence class for $\equiv_{n,M}$, i. e., (n, M)-containers. The same directly extends to interval predicates.

For the other direction, we prove that a given (n, M)-container can be expressed as an interval predicate of height at most n and width at most M. To do so, given a box $\mathsf{b} \in \mathbf{Boxes}_n$ and $m \leq M$, we define the simple interval predicate $\varphi_{\mathsf{b},\geq m}$ expressing that at least m data are mapped to box b. Formally, $\varphi_{\mathsf{b},\geq m} \coloneqq \dot{\exists}\mathsf{d}_1, \dots, \mathsf{d}_m, \bigwedge_{q\in Q}\bigwedge_{j=1}^{m} \#(q, \mathsf{d}_j) \in [A_q, B_q]$, where, for all $q \in Q$, $A_q \coloneqq \mathsf{b}(q)$, $B_q \coloneqq \mathsf{b}(q)$ if $\mathsf{b}(q) < n$ and $B_q \coloneqq +\infty$ if $\mathsf{b}(q) = n$. This predicate has height at most n and width at most M. A Boolean combination of such predicates allows us to express an (n, M)-container. We refer to [7] for a detailed proof. ◀

We therefore have two equivalent representations. Both are useful: interval predicates allow us to express properties more naturally, but containers are more convenient for the proofs in the remainder of this section. While they are equally expressive, each can be much more succinct than the other, as stated below. We refer to [7] for details.

▶ Remark 20. Containers can be exponentially more succinct than interval predicates, while interval predicates can be doubly exponentially more succinct than unions of containers.

5.2 A Translation from Expressions to Containers

Based on the translation from interval predicates to containers from Proposition 19, we can now show that for all generalised reachability expressions E over an IOPPUD $\mathcal{P}$, the set $[\![E]\!]_{\mathcal{P}}$ is a union of (n, M)-containers with n and M bounded in terms of E and $\mathcal{P}$.

▶ **Proposition 21.** *There is a polynomial function* $\mathtt{poly}\colon \mathbb{N} \to \mathbb{N}$ *such that for all IOPPUD* $\mathcal{P}$ *and GRE* E*, the set* $[\![E]\!]_{\mathcal{P}}$ *is a union of* $\left(||E|| \cdot \left(\mathtt{poly}(|\mathcal{P}|)\right)^{|E|}, ||E||^{\mathtt{poly}(|\mathcal{P}|)\cdot|E|^2}\right)$*-containers.*

The detailed proof of Proposition 21 can be found in [7]. We show the result by structural induction on E. The base case, when E is an interval predicate, is provided by Proposition 19. For the induction step, handling Boolean operators is straightforward; the difficulty lies in operators Pre^* and Post^*. This is handled by the following lemma, which relies on the bounds from Section 4.

Equivalence classes for fixed values of n and M do not behave well with respect to the reachability relation, in the sense that it can happen that $\gamma_{start} \xrightarrow{*} \gamma_{end}$ and $\gamma_{start} \equiv_{n,M} \chi_{start}$, but there is no $\chi_{end} \equiv_{n,M} \gamma_{end}$ such that $\chi_{start} \xrightarrow{*} \chi_{end}$. However, this will hold if we take some margin on the equivalence relation of configurations at the start; the following two functions express this margin. For all $n, M \in \mathbb{N}$, let $f(n) \coloneqq (n + |\mathcal{P}|^3) \cdot |\mathcal{P}|$ and $g(n, M) \coloneqq \big(M + (|\mathcal{P}|^3 + 1)^{|\mathcal{P}|^3+|\mathcal{P}|^2}\big)(n+1)^{|\mathcal{P}|}$.

The following lemma states that, if a set of configurations C cannot distinguish $\equiv_{n,M}$-equivalent configurations, then $\mathsf{Pre}^*(C)$ cannot distinguish $\equiv_{f(n),g(n,M)}$-equivalent configurations. In other words, if C is a union of $\equiv_{n,M}$-equivalence classes, (i. e., of (n, M)-containers), then $\mathsf{Pre}^*(C)$ is a union of $\equiv_{f(n),g(n,M)}$-equivalence classes.

▶ **Lemma 22.** *For all* $n, M \in \mathbb{N}$ *and all configurations* $\gamma_{start}, \gamma_{end}, \chi_{start} \in \Gamma$*, if there is a run* $\rho\colon \gamma_{start} \xrightarrow{*} \gamma_{end}$ *and* $\gamma_{start} \equiv_{f(n),g(n,M)} \chi_{start}$*, then there is a configuration* $\chi_{end} \in \Gamma$ *with* $\gamma_{end} \equiv_{n,M} \chi_{end}$ *and a run* $\pi\colon \chi_{start} \xrightarrow{*} \chi_{end}$*.*

Proof sketch. We first apply Corollary 17 to ρ, so that we can assume that ρ has a limited number of externally observed data and of observed agents per datum.

In this proof sketch, we first handle the case with only one datum. Then, we explain how to generalise this. We refer to [7] for proof details.

Suppose that all agents in γ_{start} and χ_{start} share a single datum d, and suppose $(\gamma_{start}, d) \equiv_{f(n)} (\chi_{start}, d)$. Let $\mathbb{A}_\gamma$ and $\mathbb{A}_\chi$ be the agents in γ_{start} and χ_{start}, respectively. For all $q, q' \in Q$, we set $\mathbb{A}_\gamma^{q\to} \coloneqq \{a \in \mathbb{A}_\gamma \mid \gamma_{start}(a) = q\}$, $\mathbb{A}_\chi^{q\to} \coloneqq \{a \in \mathbb{A}_\chi \mid \chi_{start}(a) = q\}$, $\mathbb{A}_\gamma^{\to q'} \coloneqq \{a \in \mathbb{A}_\gamma \mid \gamma_{end}(a) = q'\}$, and $\mathbb{A}_\gamma^{q\to q'} \coloneqq \mathbb{A}_\gamma^{q\to} \cap \mathbb{A}_\gamma^{\to q'}$.

Our aim is to assign to each agent in χ_{start} an agent in γ_{start} to mimic. To do so, we construct a mapping $\nu\colon \mathbb{A}_\chi \to \mathbb{A}_\gamma$ such that

(A) for all $a \in \mathbb{A}_\chi$, we have $\chi_{start}(a) = \gamma_{start}\big(\nu(a)\big)$,
(B) for all $a' \in \mathbb{A}_\gamma$ observed in ρ, we have $\nu^{-1}(a') \neq \emptyset$, and
(C) for all $q' \in Q$, we have $|\nu^{-1}(\mathbb{A}_{\gamma,d}^{\to q'})| = |\mathbb{A}_{\gamma,d}^{\to q'}|$, or $|\nu^{-1}(\mathbb{A}_{\gamma,d}^{\to q'})| \geq n$ and $|\mathbb{A}_{\gamma,d}^{\to q'}| \geq n$.

We build ν separately on each set $\mathbb{A}_\chi^{q\to}$ by defining, for each $q \in Q$, a mapping $\nu_q\colon \mathbb{A}_\chi^{q\to} \to \mathbb{A}_\gamma^{q\to}$. Let $q \in Q$. As $(\chi_{start}, d) \equiv_{f(n)} (\gamma_{start}, d)$, either $|\mathbb{A}_\gamma^{q\to}| = |\mathbb{A}_\chi^{q\to}|$, or both $|\mathbb{A}_\gamma^{q\to}|$ and $|\mathbb{A}_\chi^{q\to}|$ are at least $f(n)$. If $|\mathbb{A}_\gamma^{q\to}| = |\mathbb{A}_\chi^{q\to}|$, we let ν_q form a bijection between $\mathbb{A}_\chi^{q\to}$ and $\mathbb{A}_\gamma^{q\to}$. Consider now the second case, where $|\mathbb{A}_\gamma^{q\to}|$ and $|\mathbb{A}_\chi^{q\to}|$ are at least $f(n)$. We aim at selecting, for every $q' \in Q$, a set $A_{q\to q'} \subseteq \mathbb{A}_\gamma^{q\to q'}$ of agents that must be copied in π. If $|\mathbb{A}_\gamma^{q\to q'}| \leq n$, then we let $A_{q\to q'} \coloneqq \mathbb{A}_\gamma^{q\to q'}$. Otherwise, we first put in $A_{q\to q'}$ all agents in $\mathbb{A}_\gamma^{q\to q'}$ that are observed in ρ, at most $|\mathcal{P}|^3$ in total by Corollary 17. If $|A_{q\to q'}| < n$, we add

arbitrary agents from $\mathbb{A}_\gamma^{q\to q'}$ to $A_{q\to q'}$ until $|A_{q\to q'}| \geq n$. Either way, we have selected $A_{q\to q'}$ of size at most $|\mathcal{P}|^3 + n$ for each q', hence at most $f(n)$ agents in total. For every q', we have $|A_{q\to q'}| \leq |\mathbb{A}_\gamma^{q\to q'}|$, and either $|A_{q\to q'}| = |\mathbb{A}_\gamma^{q\to q'}|$ or the two sets have size more than n.

We now build ν_q such that its image over $\mathbb{A}_\chi^{q\to}$ is $\bigcup_{q'\in Q} A_{q\to q'}$. We build this in two steps. First, we assign to each $\bigcup_{q'\in Q} A_{q\to q'}$ one antecedent by ν_q in $\mathbb{A}_\chi^{q\to}$. This is possible because $|\bigcup_{q'\in Q} A_{q\to q'}| \leq f(n) \leq |\mathbb{A}_\chi^{q\to}|$. We then identify some q'' such that $|A_{q\to q''}| > n$ and map all remaining agents of $\mathbb{A}_\chi^{q\to}$ to an arbitrary agent in $A_{q\to q''}$. Such a q'' exists because $|\mathbb{A}_\gamma^{q\to}| \geq f(n) \geq n \cdot |\mathcal{P}|$, so there is a q' such that $|\mathbb{A}_\gamma^{q\to q'}| \geq n$, and hence $|A_{q\to q'}| \geq n$ by construction.

This concludes the construction of ν. It remains to prove that ν fulfils Items A–C. Items A and B are immediate from the definition. We prove Item C. Let $q' \in Q$. We distinguish two cases:

- if $|\mathbb{A}_\gamma^{q\to q'}| < n$ for all $q \in Q$, then for all q, we have $|\nu^{-1}(\mathbb{A}_\gamma^{q\to q'})| = |\nu^{-1}(A_{q\to q'})| = |A_{q\to q'}| = |\mathbb{A}_\gamma^{q\to q'}|$, so $|\nu^{-1}(\mathbb{A}_\gamma^{\to q'})| = |\mathbb{A}_\gamma^{\to q'}|$;
- if $|\mathbb{A}_\gamma^{q\to q'}| \geq n$ for some $q \in Q$, then $|A_{q\to q'}| \geq n$, so $|\nu^{-1}(\mathbb{A}_\gamma^{q\to q'})| \geq n$, and thus both $|\nu^{-1}(\mathbb{A}_\gamma^{\to q'})|$ and $|\mathbb{A}_\gamma^{\to q'}|$ are at least n.

We construct a run π from χ_{start} by copying ρ as follows. For each step of ρ where an agent a performs some transition t, we make $|\nu^{-1}(a)|$ steps in π so that all agents in $\nu^{-1}(a)$ perform transition t one by one. If a observed some agent a', there is a'' in π that can be observed because $\nu^{-1}(a') \neq \emptyset$: we made sure to map an agent to each observed agent in ρ.

For the general case with more data, we similarly construct two mappings μ and ν. First we define μ, which maps each datum d of χ_{start} to one of γ_{start} such that $(\gamma_{start}, \mu(d)) \equiv_{f(n)} (\chi_{start}, d)$. Then, for each datum d, ν maps each agent a with datum d of χ_{start} to one with datum $\mu(d)$ of γ_{start}.

Once μ and ν are defined, we build a run from χ_{start} to a configuration χ_{end} in which each agent a mimics the behaviour of $\nu(a)$ in ρ. We make sure that agents (resp. data) observed in ρ have agents (resp. data) mapped to them, so that we can take the same transitions in ρ and π. The construction of ν ensures that, for all data d, we have $(\gamma_{end}, \mu(d)) \equiv_{f(n)} (\chi_{end}, d)$. The construction of μ ensures that $\gamma_{end} \equiv_{f(n),g(n,M)} \chi_{end}$. ◀

6 Decidability and Complexity Bounds

6.1 Decidability in Exponential Space

In this section, we use the results on GRE from Section 5 to provide an ExpSpace upper bound for the emptiness problem for GRE. In the following, we assume that the representation of a GRE E takes $|E| + \log(||E||)$ space.

We first prove that we can decide membership of a configuration (encoded in a naive way) in a GRE in PSpace. A configuration is represented *data-explicitly* if it is represented as a list of vectors of $\mathbb{N}^Q$, one vector for each datum. The *size* of this representation is $k \cdot |\mathcal{P}| \cdot log(m)$ where k is the number of data and m is the number of agents appearing in γ.

▶ **Proposition 23.** *The following problem is decidable in* PSpace*: given a PPUD $\mathcal{P}$, a GRE E, and a configuration γ described data-explicitly, decide if $\gamma \in [\![E]\!]_{\mathcal{P}}$.*

We refer to the full version [7] for the proof. It uses a relatively straightforward induction on E to show that this problem can be decided in polynomial space using a recursive algorithm (with a polynomial whose degree does not depend on E). For the case where $E = \mathsf{Post}^*(F)$, we rely on the fact that the numbers of agents and data remain the same throughout a run;

we therefore can guess the configuration γ' such that $\gamma' \in [\![F]\!]_{\mathcal{P}}$ (which can be checked with a recursive call) and $\gamma \xrightarrow{*} \gamma'$ (which can be checked by exploration of the graph containing configurations with as many agents and data as γ). The case $E = \mathsf{Pre}^*(F)$ is similar.

Proposition 23 allows us to check if a given configuration of a PPUD is in the set described by a GRE[4]. In the case of IOPPUD, Proposition 21 allows us to search for a witness configuration within some bounded set, yielding decidability.

▶ **Theorem 10.** *The emptiness problem for GRE over IOPPUD is in* ExpSpace.

Proof. Suppose $[\![E]\!]_{\mathcal{P}}$ is not empty. By Proposition 21, it contains an (n, M)-container cont with $n \coloneqq ||E|| \cdot \mathtt{poly}(|\mathcal{P}|)^{|E|}$ and $M \coloneqq ||E||^{\mathtt{poly}(|\mathcal{P}|)\cdot|E|^2}$. We construct a configuration $\gamma \in \mathsf{cont}$ as follows. For each n-box b, we select $\mathsf{cont}(\mathsf{b})$ many data such that over all n-boxes, the selected data are pairwise distinct. Then, for each n-box b, each state $q \in Q$ of $\mathcal{P}$, and each datum d_{b} selected for b, we put $\mathsf{b}(q)$ many agents with datum d_{b} in q. Note that the configuration γ is in cont, and the number of agents it contains it at most $n \cdot |\mathcal{P}| \cdot |\mathbf{Boxes}_n| \cdot M$. We have $|\mathbf{Boxes}_n| = (n+1)^{|\mathcal{P}|} = ||E|| \cdot \mathtt{poly}(|\mathcal{P}|)^{|E||\mathcal{P}|}$. We assumed at the beginning of Section 6 that the encoding of E uses memory $|E| + \mathsf{log}(||E||)$. As a result, n, M, $|\mathcal{P}|$ and $|\mathbf{Boxes}_n|$ are all at most exponential in the size of the input. Therefore, if $[\![E]\!]_{\mathcal{P}}$ is not empty, then it contains a configuration with at most exponentially many agents. We can guess the data-explicit description of such a configuration in non-deterministic exponential space, and then check that the guessed configuration is in $[\![E]\!]_{\mathcal{P}}$ in exponential space by Proposition 23 (we apply the PSpace algorithm on an exponential input). As a result, deciding emptiness of $[\![E]\!]_{\mathcal{P}}$ is in NExpSpace, which is identical with ExpSpace. ◀

6.2 A Lower Complexity Bound

We now provide the following lower complexity bound.

▶ **Theorem 24.** *The emptiness problem for GRE over IOPPUD is* coNExpTime*-hard.*

Proof sketch. We proceed by reduction from the problem of tiling an exponentially large grid, a NExpTime-complete problem [28], to the complement of the emptiness problem for GRE. We refer to [7] for proof details.

A *tiling instance* is a tuple $(2^n, \mathcal{C}, \mathcal{T})$, with $n \geq 1$, $\mathcal{C}$ a finite set of *colours* with special colour white, and $\mathcal{T} = \{t_1, \ldots, t_m\} \subseteq \mathcal{C}^4$ a finite set of *tiles*. We can view a tile as a square whose four edges are coloured. The *tiling problem* asks whether there is a *tiling*, that is, a mapping $\tau \colon [0, 2^n{-}1] \times [0, 2^n{-}1] \to \mathcal{T}$ such that the colours of neighbouring tiles match and the borders of the grid are white.

Given a tiling instance $(2^n, \mathcal{C}, \mathcal{T})$, we build an instance $(\mathcal{P}, E)$ of the emptiness problem for GRE. In $\mathcal{P}$, witness tilings can be encoded in the configurations, and we construct $(\mathcal{P}, E)$ such that $[\![E]\!]$ contains exactly the configurations that correspond to a correctly encoded witness tiling. More precisely, $\gamma \in [\![E]\!]$ when:

(Cond1) for all $(i, j) \in [0, 2^n - 1]^2$, some datum encodes coordinates (i, j) and a tile type;
(Cond2) for all $(i, j) \in [0, 2^n - 1]^2$, there is at most one datum encoding (i, j);
(Cond3) the mapping $[0, 2^n - 1]^2 \to \mathcal{C}$ defined by the data is a tiling.

The GRE E will be of the form of a conjunction, i. e., a list of *constraints* that the configuration must satisfy. Our first constraint is $\overline{\mathsf{Pre}^*(\mathsf{Pres}(q_\perp))}$ where $q_\perp$ is a special error state and $\mathsf{Pres}(q)$ is the GRE expressing that some agent is in q. This forbids, in $[\![E]\!]$, configurations from which $q_\perp$ can be covered.

[4] This implies that the emptiness problem for GRE over PPUD, while undecidable due to Theorem 3, is semi-decidable: one can simply enumerate all configurations and test membership for each of them.

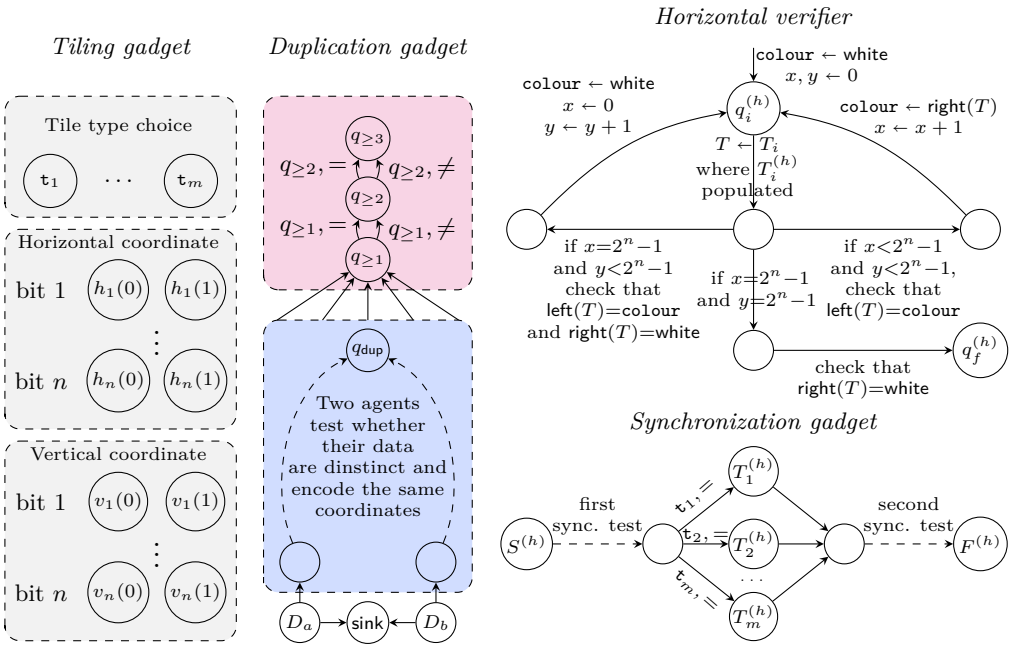


Figure 4 Partial depiction of the protocol constructed in Theorem 24.

(Cond1) is obtained using the *tiling gadget* in Figure 4. States $\mathtt{t}_1, \ldots, \mathtt{t}_m$ represent the available tiles of $\mathcal{T}$, and coordinate states allow for a binary representation of the horizontal and vertical coordinates of a square in the grid. For a datum d, the agents of datum d in the coordinate states encode the position of the square corresponding to d, and an agent of datum d in state $\mathtt{t}_i$ indicates that the square in the grid corresponding to d should be coloured according to tile t_i. Configurations in $[\![E]\!]$ are not allowed to have two agents of same datum playing the same role; otherwise, one of them may observe the other and go to $q_\perp$. In particular, each datum has at most $2n+1$ agents in the tiling gadget.

To obtain (Cond2), we use a *duplication gadget*, partially represented in Figure 4. We enforce that any configuration in $[\![E]\!]$ has one agent of each datum in D_a, one in D_b and none in the rest of the duplication gadget. The blue part implements a test (depicted in [7, Figure 5b]) where two agents of distinct data, one from D_a and one from D_b, may test that their data encode the same coordinates; if this is the case, they may go to q_{dup}. If there are more than two agents in the blue part, this test is not reliable but $q_{\geq 3}$ can be covered. (Cond2) can therefore be achieved by enforcing that configurations in $[\![E]\!]$ are *not* in $\mathsf{Pre}^*(\mathsf{Pres}(q_{\mathsf{dup}}) \cap \overline{\mathsf{Pre}^*(\mathsf{Pres}(q_{\geq 3}))})$.

Finally, we explain how (Cond3) is achieved; we describe only how the horizontal (left-right) borders are verified. We use a gadget, named *horizontal verifier* in Figure 4. In this gadget, a single agent, called *verifier*, is in charge of verifying that colours of left-right borders match. The verifier uses $2n$ auxiliary agents to encode two variables $x, y \in [0, 2^n - 1]$ in binary. Again, transitions to $q_\perp$ detect when two agents play the same role, so that there is only one verifier and so that variables x and y can be implemented faithfully. The initialisation $x = y = 0$ is enforced as a constraint in E. We now sketch how the verifier reads the encoded tiling; to do that, it must synchronise with the datum encoding (x, y).

This is done using the synchronisation gadget of Figure 4. In $[\![E]\!]$, all agents in the synchronisation gadget are in $S^{(h)}$. Moreover, we add a constraint in E so that $\gamma \in [\![E]\!]$ requires that there is a run from γ where all agents in the synchronisation gadget end in

$F^{(h)}$ and where the verifier ends in $q_f^{(h)}$. The synchronisation tests guarantee that, whenever there is an agent in $T_i^{(h)}$, this agent's datum encodes square (i, j) where i is equal to the current value of x and j is equal to the current value of y. The synchronisation is challenging to design because the values of x and y may change throughout a run and only one bit can be tested at a time. However, as proved in [7, Lemma 37], this can be achieved by having a first synchronisation test that checks equality of bits from most to least significant, and a second test that checks equality from least to most significant. ◀

6.3 Discussion on Complexity Gaps

We now discuss some complexity gaps left open by this paper. First, there remains a complexity gap for the emptiness problem for GRE, which is known to be between coNExpTime (Theorem 24) and ExpSpace (Theorem 10). Closing the gap appears challenging. On one hand, if the problem is below ExpSpace, then this probably requires developing new techniques. On the other hand, proving ExpSpace-hardness does not seem easy. In particular, the synchronisation techniques from Theorem 24 assumes that each datum synchronises only once with the verifier. This synchronisation technique would not suitable for, e. g., multiple interactions between the head and the cells of a Turing machine.

Another, arguably more important open question is the exact complexity of well-specification, which is only known to be between PSpace (model without data, [19]) and ExpSpace (Theorem 10). On the one hand, it is unclear whether relevant configurations can be stored in polynomial space.

▷ Claim 25. The number of data that need to be considered for well-specification may be exponential.

The claim is formalised and proven in [7]. As a consequence, proving that the problem is in PSpace cannot be achieved with a procedure that explicitly stores configurations. On the other hand, in order to build a reduction from the tiling problem as in Theorem 24, we need a new idea to enforce that *at most* one datum encodes each tile. In Theorem 24, we had states q_{dup} and $q_{\geq 3}$ and duplication meant being able to cover q_{dup} and, at the same, forbid that $q_{\geq 3}$ can ever be covered in the future. We do not know how to encode this constraint when working with an instance of well-specification.

7 Conclusion

We have studied the verification of population protocols with unordered data [10], an extension of population protocols where agents carry data from an infinite unordered set. We first proved that the well-specification problem is undecidable (Theorem 3), which then led us to consider the restriction to protocols with immediate observation. This subclass was defined in [10], where the authors proved that these protocols compute exactly the interval predicates. We defined a general class of problems on this model, which consists in deciding the existence of a configuration satisfying a so-called generalised reachability expression; this class of problems subsumes many classic problems, one of which is well-specification. Despite its generality, we showed the problem to be decidable in exponential space (Theorem 10); we also provided a coNExpTime lower bound. A remaining open question is the exact complexity of well-specification for immediate observation population protocols with unordered data, which is located between PSpace (model without data, [19]) and ExpSpace (Theorem 10).

References

1 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-space trade-offs in population protocols. In *28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 2560–2579, 2017. doi:10.1137/1.9781611974782.169.

2 Dan Alistarh and Rati Gelashvili. Recent algorithmic advances in population protocols. *SIGACT News*, 49(3):63–73, 2018. doi:10.1145/3289137.3289150.

3 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *33rd Annual ACM Symposium on Principles of Distributed Computing, PODC 2004*, pages 290–299. ACM, 2004. doi:10.1145/1011767.1011810.

4 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Comput.*, 20(4):279–304, 2007. doi:10.1007/S00446-007-0040-2.

5 A. R. Balasubramanian, Lucie Guillou, and Chana Weil-Kennedy. Parameterized analysis of reconfigurable broadcast networks. In *Foundations of Software Science and Computation Structures - 25th International Conference, FoSSaCS 2022*, pages 61–80, 2022. doi:10.1007/978-3-030-99253-8_4.

6 A. R. Balasubramanian and Chana Weil-Kennedy. Reconfigurable broadcast networks and asynchronous shared-memory systems are equivalent. In *12th International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2021*, pages 18–34, 2021. doi:10.4204/EPTCS.346.2.

7 Steffen van Bergerem, Roland Guttenberg, Sandra Kiefer, Corto Mascle, Nicolas Waldburger, and Chana Weil-Kennedy. Verification of population protocols with unordered data. *CoRR*, abs/2405.00921, 2024. arXiv:2405.00921.

8 Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015. doi:10.2200/S00658ED1V01Y201508DCT013.

9 Michael Blondin, Javier Esparza, Stefan Jaax, and Philipp J. Meyer. Towards efficient verification of population protocols. *Formal Methods Syst. Des.*, 57(3):305–342, 2021. doi:10.1007/S10703-021-00367-3.

10 Michael Blondin and François Ladouceur. Population protocols with unordered data. In *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023*, pages 115:1–115:20, 2023. doi:10.4230/LIPICS.ICALP.2023.115.

11 Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier, and Daniel Stan. Reachability in networks of register protocols under stochastic schedulers. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, pages 106:1–106:14, 2016. doi:10.4230/LIPICS.ICALP.2016.106.

12 Philipp Czerner, Roland Guttenberg, Martin Helfrich, and Javier Esparza. Fast and succinct population protocols for Presburger arithmetic. *J. Comput. Syst. Sci.*, 140:103481, 2024. doi:10.1016/J.JCSS.2023.103481.

13 Wojciech Czerwinski and Lukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 1229–1240, 2021. doi:10.1109/FOCS52979.2021.00120.

14 Robert Elsässer and Tomasz Radzik. Recent results in population protocols for exact majority and leader election. *Bull. EATCS*, 126, 2018. URL: http://bulletin.eatcs.org/index.php/beatcs/article/view/549/546.

15 Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, pages 1–10, 2014. doi:10.4230/LIPICS.STACS.2014.1.

16 Javier Esparza. Population protocols: Beyond runtime analysis. In *Reachability Problems - 15th International Conference, RP 2021*, pages 28–51, 2021. doi:10.1007/978-3-030-89716-1_3.

17 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Informatica*, 54(2):191–215, 2017. doi:10.1007/S00236-016-0272-3.

18 Javier Esparza, Stefan Jaax, Mikhail A. Raskin, and Chana Weil-Kennedy. The complexity of verifying population protocols. *Distributed Comput.*, 34(2):133–177, 2021. doi:10.1007/S00446-021-00390-X.

19 Javier Esparza, Mikhail A. Raskin, and Chana Weil-Kennedy. Parameterized analysis of immediate observation Petri nets. In *Application and Theory of Petri Nets and Concurrency - 40th International Conference, PETRI NETS 2019*, pages 365–385, 2019. doi:10.1007/978-3-030-21571-2_20.

20 Lucie Guillou, Corto Mascle, and Nicolas Waldburger. Parameterized broadcast networks with registers: from NP to the frontiers of decidability. In *Foundations of Software Science and Computation Structures - 27th International Conference, FoSSaCS 2024*, pages 250–270, 2024. doi:10.1007/978-3-031-57231-9_12.

21 Petr Jancar. Undecidability of bisimilarity for Petri nets and some related problems. *Theor. Comput. Sci.*, 148(2):281–301, 1995. doi:10.1016/0304-3975(95)00037-W.

22 Petr Jancar and Jérôme Leroux. The semilinear home-space problem is Ackermann-complete for Petri nets. In *34th International Conference on Concurrency Theory, CONCUR 2023*, pages 36:1–36:17, 2023. doi:10.4230/LIPICS.CONCUR.2023.36.

23 Ranko Lazic, Thomas Christopher Newcomb, Joël Ouaknine, A. W. Roscoe, and James Worrell. Nets with tokens which carry data. In *28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, ICATPN 2007*, pages 301–320, 2007. doi:10.1007/978-3-540-73094-1_19.

24 Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 1241–1252, 2021. doi:10.1109/FOCS52979.2021.00121.

25 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019*, pages 1–13, 2019. doi:10.1109/LICS.2019.8785796.

26 Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., 1967.

27 Fernando Rosa-Velardo and David de Frutos-Escrig. Decidability and complexity of Petri nets with unordered data. *Theor. Comput. Sci.*, 412(34):4439–4451, 2011. doi:10.1016/J.TCS.2011.05.007.

28 François Schwarzentruber. The complexity of tiling problems. *CoRR*, abs/1907.00102, 2019. arXiv:1907.00102.

29 Chana Weil-Kennedy. *Observation Petri Nets*. PhD thesis, Technical University of Munich, Germany, 2023. URL: https://nbn-resolving.org/urn:nbn:de:bvb:91-diss-20230320-1691161-1-3.

Domain Reasoning in TopKAT

Cheng Zhang
Boston University, MA, USA

Arthur Azevedo de Amorim
Rochester Institute of Technology, NY, USA

Marco Gaboardi
Boston University, MA, USA

Abstract

TopKAT is the algebraic theory of Kleene algebra with tests (KAT) extended with a top element. Compared to KAT, one pleasant feature of TopKAT is that, in relational models, the top element allows us to express the domain and codomain of a relation. This enables several applications in program logics, such as proving under-approximate specifications or reachability properties of imperative programs. However, while TopKAT inherits many pleasant features of KATs, such as having a decidable equational theory, it is incomplete with respect to relational models. In other words, there are properties that hold true of all relational TopKATs but cannot be proved with the axioms of TopKAT. This issue is potentially worrisome for program-logic applications, in which relational models play a key role.

In this paper, we further investigate the completeness properties of TopKAT with respect to relational models. We show that TopKAT is complete with respect to (co)domain comparison of KAT terms, but incomplete when comparing the (co)domain of arbitrary TopKAT terms. Since the encoding of under-approximate specifications in TopKAT hinges on this type of formula, the aforementioned incompleteness results have a limited impact when using TopKAT to reason about such specifications.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Programming logic

Keywords and phrases Kleene algebra, Kleene Algebra With Tests, Kleene Algebra With Domain, Kleene Algebra With Top and Tests, Completeness, Decidability

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.157

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding *Cheng Zhang*: National Science Foundation Grant No. 2040249 and No. 2314324
Arthur Azevedo de Amorim: National Science Foundation Grant No. 2314323
Marco Gaboardi: National Science Foundation Grant No. 2040249 and No. 2314324

1 Introduction

Kleene algebra with tests (KAT) is an algebraic framework that extends Kleene algebra with an embedded Boolean algebra to model control structures like if-statement and while-loops [20]. This extension enables us to reason about several properties of imperative programs. For example, one of the key early results in the area was that KAT can encode Hoare logic, in the sense that any proof in the logic's propositional fragment can be carried out faithfully using KAT equations [27, 21].

Some applications, however, require us to look beyond KAT. For example, Zhang et al. [41] recently proved that KAT alone cannot be used to encode incorrectness logic [31, 7] – a close cousin of Hoare logic with applications in bug finding [25, 36]. A similar result was proved by Struth [39], who showed that KAT cannot encode weakest liberal preconditions. If we view a program as a relation between its input and output states, both of these limitations


arise from KAT's lack of power to encode the (co)domain of a relation. Indeed, Möller et al. [30] proved that incorrectness logic could be encoded by extending KAT with a codomain operation. Independently, Zhang et al. provided a similar encoding [41] by extending KAT with a top element, which can be used to express inequalities between codomains. They dubbed the resulting algebraic structure a TopKAT.

The present paper investigates the expressive power of TopKAT as a tool for (co)domain reasoning. As noted by Zhang et al. [41], one limitation of TopKAT is that it is not expressive enough to derive all valid equations between relations. More precisely, Zhang et al.'s encoding of incorrectness logic interprets the top element of the algebra as the complete relation, which relates all pairs of program states. Under this interpretation, the inequality $p\top p \geq p$ is valid, but unprovable using the theory of TopKAT [41]. This a potential issue when using TopKAT to reason in incorrectness logic: though Zhang et al.'s encoding covers all the rules of propositional incorrectness logic, there could be inequalities about (co)domain that fall outside this fragment and cannot be established solely by the theory of TopKAT.

Pous et al. [34, 35] were able to make some progress on the issue, by showing we can obtain a complete axiomatic system for relational models TopKATs by adding in the inequality $p\top p \geq p$ as an additional axiom. In this paper, we look at the question from a different angle, instead of working with a more complex theory, we show that the original theory of TopKAT is complete with respect to relational models for (co)domain comparisons, namely the inequalities of the form $\top t_1 \geq \top t_2$ or $t_1\top \geq t_2\top$ where t_1, t_2 are KAT terms. Since these inequalities suffice to encode incorrectness logic, this completeness result lays a solid foundation for encoding program logics in TopKAT. We have also showed that this completeness result is tight, in the sense that it does not extend to the case where t_1 and t_2 contain the top element, by explicitly constructing two TopKAT terms that witness the incompleteness.

The result above is enabled by the homomorphic structure of the reduction [41, 33] from TopKAT to KAT. This discovery also let us shorten the proofs of previous results [41], and enables systematic generation of TopKAT complete interpretations from complete interpretations of KAT. We believe that this new representation of the reduction technique could also be of independent interest.

Structure of this paper and contributions. In Section 2, we present several previous results on KAT and TopKAT. Inspired by universal algebra [4], we characterize fundamental concepts, like interpretation and completeness, using homomorphisms. In Section 3, we uncover additional structure of the reduction technique [24, 33] in the case of TopKAT: the reduction from TopKAT to KAT is a TopKAT homomorphism. This discovery not only allows us to simplify several previous results [41] by avoiding tedious induction proofs; but also enables the techniques used in the later section. Section 4 presents the completeness results of TopKAT with respect (co)domain comparison. The codomain completeness result is proven by an equality that connects codomain operation with the language interpretation, and the domain completeness is then proven by applying the codomain completeness result to the opposite TopKAT.

2 Preliminaries

2.1 Extensions of Kleene algebra And Their Models

A *Kleene algebra* is an idempotent semiring with a star operation, written p^*, that satisfies the following *unfolding*, *left induction*, and *right induction* rules:

$$p^* = 1 + pp^* = 1 + p^*p, \qquad pr + q \leq r \implies p^*q \leq r, \qquad rp + q \leq r \implies qp^* \leq r;$$

the ordering here is the conventional ordering in idempotent semirings: $p \leq q \triangleq p + q = q$. It is known that the right-hand version of unfolding and induction rule can be removed while preserving the same equational theory [23]. Yet, we will focus on the standard definition of KA in this paper.

▶ **Lemma 1.** *Following are well-known facts in Kleene algebra*

- *All the Kleene algebra operations preserve order.*
- *The following equations are true for the star operation:*

$$p^* \cdot p^* = p^* \qquad\qquad (p^*)^* = p^*.$$

A Kleene algebra with tests (KAT) is a Kleene algebra with an embedded Boolean algebra, where the conjunction, disjunction, and identities in the Boolean algebra coincide with the addition, multiplication, and the identities of Kleene algebra. We refer to elements of this embedded Boolean algebra as *tests*.

Given an algebraic theory, we can construct its *free model* over a finite set Σ, called the *alphabet* [4]. The free model consists of all the terms formed by Σ modulo provable equivalences of the algebra. The operations of the free model are obtained by lifting the term-level operations to equivalence classes.

The above construction can be extended to the case of KAT and TopKAT, suppose that we are given two disjoint finite sets K (the *action alphabet*) and B (the *test alphabet*). Elements of K and B are called *primitive actions and primitive tests*, respectively. KAT terms over the alphabet K, B are defined with the following grammar:

$$t \triangleq b \in B \mid p \in K \mid 1 \mid 0 \mid t_1 + t_2 \mid t_1 \cdot t_2 \mid t^* \mid \overline{t_b},$$

where t_b does not contain primitive actions. The *free KAT* over K, B, written $\mathsf{KAT}_{K,B}$, consists of terms over K, B modulo provable KAT equivalences. The tests of the free KAT are Boolean terms, i.e. terms formed by primitive tests and Boolean operations modulo Boolean axioms. A similar construction applies to TopKAT, where an additional symbol $\top$ was added as the largest element in the theory; we denote the free TopKAT over K, B as $\mathsf{TopKAT}_{K,B}$. We sometimes omit the alphabets K and B when they are irrelevant or can be inferred.

In the paper, we frequently consider terms modulo provable equalities, i.e. in the context of its corresponding free model. For example, given $t_1, t_2 \in \mathsf{KAT}$, we will say $t_1 = t_2$ when they are provably equal using the theory of KAT. Although the free model seems trivial, it leads to simpler and more modular proofs of some properties of algebraic theories, as we will see in Section 3.

Other important models that we will use in this paper are language (Top)KATs and relational (Top)KATs, which we review here. An *atom* (short for "atomic test") over a test alphabet $B = \{b_1, b_2, \ldots, b_n\}$ is a sequence of the form

$$\hat{b_1} \cdot \hat{b_2} \cdot \cdots \cdot \hat{b_n} \text{ where } \hat{b_i} \in \{b_i, \bar{b_i}\}.$$

We denote atoms as $\alpha, \beta, \gamma, \ldots$ and the set of all atoms as At.

A *guarded string* (or *guarded word*) over K, B is an alternation between atoms and primitive actions that starts and ends in atoms:

$$\alpha_0 p_1 \alpha_1 \cdots p_n \alpha_n \text{ where } p_i \in K, \alpha_i \in \text{At};$$

where each action is "guarded" by an atom. A guarded string is similar to a program trace, where each program state is denoted by an atom; and primitive actions will cause a transition

between program states. We denote the set of all guarded strings over alphabet K, B as $GS_{K,B}$, and we will omit the alphabet K, B when it is irrelevant or can be inferred from context. The notation αs denotes a guarded string starting with atom α with the rest of the string s; similarly, $s\alpha$ denotes a guarded string that ends with atom α with rest of the string being s.

▶ **Definition 2** (Language/trace KAT [24]). *The* language KAT *(also called "*trace KAT*") over an alphabet K, B is denoted as $\mathcal{G}_{K,B}$, or simply $\mathcal{G}$ if no confusion can arise.*

The elements are sets of guarded strings (called guarded languages*), and the tests are sets of atoms. The additive identity 0 is the empty set, and the multiplicative identity 1 is the set of all the atoms* At*. The addition operator is set union, and the multiplication operator is defined as follows:*

$$S_1 \diamond S_2 \triangleq \{s_1 \alpha s_2 \mid s_1\alpha \in S_1, \alpha s_2 \in S_2\}.$$

The star operation is defined non-deterministically iterating the multiplication operator:

$$S^* \triangleq \bigcup_{i\in\mathbb{N}} S^i \text{ where } S^0 = \mathrm{At}, S^{k+1} = S \diamond S^k.$$

Another useful type of KAT are relational ones, where each element is a relation $R \subseteq X \times X$ over a fixed set X. In applications, the set X typically represents the set of all possible program states, and each relation R represents a program by relating each possible input to the corresponding output.

▶ **Definition 3** (Relational KAT). *A relational KAT is a KAT $\mathcal{R}$ consists of relations over a fixed set X (though $\mathcal{R}$ need not contain every relation over X), and it is closed under the following operations. The tests are all the relations that are subsets of the identity relation. The additive identity 0 is the empty set, and the multiplicative identity is the identity relation:*

$$1 \triangleq \{(x, x) \mid x \in X\}.$$

The addition operator is set union, and the multiplication operation is relational composition:

$$R_1; R_2 = \{(x, z) \mid \exists y \in X, (x, y) \in R_1, (y, z) \in R_2\}.$$

Finally, the star operation is defined as:

$$R^* \triangleq \bigcup_{i\in\mathbb{N}} R^i \text{ where } R^0 = 1, R^{k+1} = R; R^k.$$

We denote the class of all relational KATs as REL*.*

TopKAT extends the theory of KAT with the largest element $\top$, i.e. $\top \geq p$ for all elements p. The *language TopKAT* over an alphabet K, B has the same carrier and operations as $\mathcal{G}_{K_\top,B}$, where $K_\top$ is the set K joined with a new primitive action $\top$; and the largest element is the full language $GS_{K_\top,B}$.

The *relational TopKAT* is a relational KAT that contains the complete relation:

$$\top \triangleq \{(x, y) \mid x, y \in X\};$$

we denote the set of all relational TopKATs as TopREL. It is known that there are equations that are valid in relational TopKAT, but are not derivable by the axioms of TopKAT [41]; however, by adding the axiom $p\top p \geq p$, the theory becomes complete over relational

TopKATs [34, 35]. In this paper, instead of working with a more complex theory, we will show that TopKAT without any additional axiom already suffices for the purpose of encoding domain comparisons. Indeed, TopKAT is complete with respect to domain comparison inequalities, which can be used to encode both incorrectness logic and Hoare logic.

In this paper, we will use dom and cod to denote the conventional (co)domain operators on relations, namely, for any relation R:

$$\mathrm{dom}(R) \triangleq \{x \mid \exists y, (x,y) \in R\} \qquad \mathrm{cod}(R) \triangleq \{y \mid \exists x, (x, y \in R)\}.$$

To demonstrate how TopKAT models (co)domain comparisons, we take any relational TopKAT $\mathcal{R}$ and two relations $R_1, R_2 \in \mathcal{R}$, and we denote the complete relation as $\top$:

▶ **Lemma 4** (TopKAT encodes (co)domain comparison).

$$R_1\top \supseteq R_2\top \iff \mathrm{dom}(R_1) \supseteq \mathrm{dom}(R_2) \qquad \top R_1 \supseteq \top R_2 \iff \mathrm{cod}(R_1) \supseteq \mathrm{cod}(R_2)$$

If we regard R_1 and R_2 as the input output relation of two programs, which is typically encoded by KAT terms, we can see that $R_1\top \supseteq R_2\top$ reflects that the domain of R_1 is larger than the domain of R_2; and similarly for the inequality $\top R_1 \supseteq \top R_2$. Thus, given two KAT terms $t_1, t_2 \in \mathsf{KAT}_{K,B}$, we call inequalities like $t_1\top \geq t_2\top$ *domain comparison inequalities*, and $\top t_1 \geq \top t_2$ *codomain comparison inequalities*. Notice that the term $\top t_1$ is a shorthand for $\top \cdot i(t_1)$, where i is the inclusion function $\mathsf{KAT}_{K,B} \hookrightarrow \mathsf{TopKAT}_{K,B}$. In the rest of the paper, we will sometimes leave this inclusion function implicit. These two forms of inequalities will be the focus of our completeness results in Section 4.

We also know another class of TopKATs named *general relational TopKATs*, which is denoted as TopGREL. The top element of general relational TopKAT is not necessarily the complete relation, but the largest relation in the model. All equations in the general relational TopKAT can be derived using the theory of TopKAT.

However, the completeness of TopGREL came at the cost of expressive power: every predicate that is expressible using general relational TopKAT is already expressible using relational KAT [41], so the extension with top, in the case of general relational TopKAT, does not grant any extra expressive power. In Theorem 13, we show that this result is a simple corollary of our new reduction result.

We are also interested in maps between models: A *KAT homomorphism* f is a map between two KATs $\mathcal{K}$ and $\mathcal{K}'$ s.t. it preserves the sorts and operations: given a test b in $\mathcal{K}$ then $f(b)$ is a test in $\mathcal{K}'$; and all the KAT operations (complement, identities, addition, multiplication, and star) are preserved:

$$\begin{aligned} f &: \mathcal{K} \to \mathcal{K}' \\ f(\bar{b}) &= \overline{f(b)} \\ f(1) &= 1 \\ f(0) &= 0 \\ f(p+q) &= f(p) + f(q) \\ f(p \cdot q) &= f(p) \cdot f(q) \\ f(p^*) &= f(p)^*. \end{aligned}$$

Similarly, a *TopKAT homomorphism* is a KAT homomorphism that preserves the largest element.

2.2 Interpretation, Completeness, and Injectivity

Consider a KAT equation such as $p \cdot b \cdot \bar{b} = 0$. To determine its validity in a particular KAT $\mathcal{K}$, we need to assign meaning to it by interpreting each primitive as an element in $\mathcal{K}$; that is, by defining a map $\hat{I}$ of type $K + B \to \mathcal{K}$. Such a map $\hat{I} : K + B \to \mathcal{K}$ induces a unique KAT homomorphism $I : \mathsf{KAT}_{K,B} \to \mathcal{K}$ inductively defined on the term as follows:

$$\begin{aligned} I(p) &\triangleq \hat{I}(p) && \text{where } p \in K + B \\ I(\overline{t_b}) &\triangleq \overline{I(t_b)} && t_b \text{ does not contain primitive actions} \\ I(t_1 + t_2) &\triangleq I(t_1) + I(t_2) \\ I(t_1 \cdot t_2) &\triangleq I(t_1) \cdot I(t_2) \\ I(t^*) &\triangleq I(t)^* \end{aligned} \tag{1}$$

In fact, every KAT homomorphism from a free model arises this way: there is a bijection between functions of type $K + B \to \mathcal{K}$ and KAT homomorphisms of type $\mathsf{KAT}_{K,B} \to \mathcal{K}$, for any KAT $\mathcal{K}$. Because the homomorphism I and the function $\hat{I}$ are equivalent, we will refer to them interchangeably as *KAT interpretations* and denote both of them as I.

The above result enables us to define a homomorphism from the free KAT just by defining its action on the primitives; saving us time to check the equations that a homomorphism must satisfy. It also allows us to prove that two interpretations are equal by arguing that they map the primitives to equal values.

Given a KAT $\mathcal{K}$, and two terms $t_1, t_2 \in \mathsf{KAT}_{K,B}$ we say that $\mathcal{K} \models t_1 = t_2$ if

$$\forall I : \mathsf{KAT}_{K,B} \to \mathcal{K}, I(t_1) = I(t_2).$$

In particular, for two terms in the free model $t_1, t_2 \in \mathsf{KAT}_{K,B}$, $\mathsf{KAT}_{K,B} \models t_1 = t_2$ is equivalent to $t_1 = t_2$. For a collection of models K, we say that $\mathsf{K} \models t_1 = t_2$ if for all $\mathcal{K} \in \mathsf{K}$, $\mathcal{K} \models t_1 = t_2$. For example, $\mathrm{REL} \models t_1 = t_2$ means that $t_1 = t_2$ is valid in all relational KATs. All the above notations and terminologies can be similarly extended to TopKAT.

Theories like KAT and TopKAT are designed to model practical programs, so it is important to know if they can model all the desirable equations between programs. If the theory of KAT can derive all the equalities for a particular interpretation I, namely:

$$\mathsf{KAT}_{K,B} \models t_1 = t_2 \iff I(t_1) = I(t_2),$$

we say that the theory of KAT is *complete* with respect to I. Recall that $\mathsf{KAT}_{K,B} \models t_1 = t_2$ is equivalent to $t_1 = t_2$; thus, by definition, an interpretation I is complete if and only if it is injective. One of such interpretation is the guarded string interpretation $G : \mathsf{KAT}_{K,B} \to \mathcal{G}_{K,B}$ [24], defined by lifting the following action on the primitives:

$$G(b) = \{\alpha \mid b \text{ appears positively in } \alpha\}, \qquad G(p) = \{\alpha p \beta \mid \alpha, \beta \in \mathrm{At}\}.$$

In several previous works, the term "free model" refers to the range (set of reachable elements) of a complete interpretation. Since a complete interpretation is an injective homomorphism, such interpretation induces an isomorphism on its range, thus our definition of free model is equivalent to these definitions.

Many previous proofs can also be explained by seeing complete interpretations as injective homomorphisms: the proof for completeness of relational KATs constructs an injective homomorphism h from a language KAT into a relational KAT [24]. Since both G and h

are injective homomorphisms, $h \circ G$ is also an injective homomorphism, hence a complete interpretation. Since $h \circ G$ is a relational interpretation:

$$\mathsf{KAT}_{K,B} \models t_1 = t_2 \implies \mathrm{REL} \models t_1 = t_2 \implies h \circ G(t_1) = h \circ G(t_2);$$

then the completeness of $h \circ G$ implies $(h \circ G)(t_1) = (h \circ G)(t_2) \iff \mathsf{KAT}_{K,B} \models t_1 = t_2$. Hence,

$$\mathsf{KAT}_{K,B} \models t_1 = t_2 \iff \mathrm{REL} \models t_1 = t_2,$$

i.e. the theory of KAT is complete with respect to relational KAT.

Besides using composition of injective homomorphisms, another technique commonly used to prove injectivity is to construct a left inverse: if a (Top)KAT homomorphism $f : \mathcal{K} \to \mathcal{K}'$ has a left inverse homomorphism $g : \mathcal{K}' \to \mathcal{K}$ i.e. $g \circ f = id_{\mathcal{K}}$, then f is injective. Notice that g does not need to be a homomorphism for f to be injective, however, in the case where f is an interpretation, g being a homomorphism makes the equality $g \circ f = id_{\mathcal{K}}$ easier to check. Because both $g \circ f$ and $id_{\mathcal{K}}$ are all interpretations, they are equal if and only if they have the same action on all the primitives.

Finally, we provide a shorthand for domain reasoning. For two terms $t_1, t_2 \in \mathsf{KAT}$, we write

$$\mathrm{REL} \models \mathrm{dom}(t_1) \geq \mathrm{dom}(t_2),$$

when $\mathrm{dom}(I(t_1)) \supseteq \mathrm{dom}(I(t_1))$ for all relational KAT interpretations I; and similarly for relational TopKAT and general relational TopKAT. Then Lemma 4 implies the following:

▶ **Lemma 5.** *For two KAT terms $t_1, t_2 \in \mathsf{KAT}_{K,B}$:*

$$\mathrm{TopREL} \models t_1 \top \geq t_2 \top \iff \mathrm{REL} \models \mathrm{dom}(t_1) \geq \mathrm{dom}(t_2)$$
$$\mathrm{TopREL} \models \top t_1 \geq \top t_2 \iff \mathrm{REL} \models \mathrm{cod}(t_1) \geq \mathrm{cod}(t_2)$$

3 Reduction, A New Perspective

Our goal in this section is to construct a complete interpretation for TopKAT, by reducing its theory to that of plain KAT. In other words, any equation between two TopKAT terms is logically equivalent to another equation between a pair of corresponding KAT terms. While this result is not new [41, 42, 34], we present a more streamlined proof that hinges on the universal properties of free KATs and TopKATs, without relying explicitly on language models. Similar to previous works, we obtain the decidability of the equational theory of TopKAT as a corollary of reduction. However, because of the new notion of reduction, our decidability result no longer depends on the completeness of the language TopKAT. Moreover, our technique helps us to construct complete models and interpretations simply by computation, as well as simplifying proofs of other results about TopKAT.

3.1 Reduction on free models

We first note that any free KAT over an alphabet K, B is also a TopKAT, where the largest element is $(\sum K)^*$. This fact can be seen by straightforward induction.

▶ **Lemma 6.** *Every free KAT over alphabet K, B forms a TopKAT.*

Proof. Since $\mathsf{KAT}_{K,B}$ is a KAT, we only need to show the term $(\sum K)^*$ is the largest element of $\mathsf{KAT}_{K,B}$, i.e.

$$(\sum K)^* \geq t, \forall t \in \mathsf{KAT}_{K,B}.$$

The above fact can be shown by induction on t; some algebraic manipulations below use facts in Lemma 1:

- $(\sum K)^* \geq 1$ (by unfolding rule), thus $(\sum K)^*$ is larger than $0, 1$ and every Boolean term.
- $(\sum K)^*$ is larger than $\sum K$, which is larger than every primitive action.
- Given two terms t_1 and t_2, assume $(\sum K)^*$ is larger than both. Because $(\sum K)^* = (\sum K)^* + (\sum K)^*$ and addition preserves order,
$$(\sum K)^* = (\sum K)^* + (\sum K)^* \geq t_1 + t_2$$
- Given two terms t_1 and t_2, assume $(\sum K)^*$ is larger than both. Because $(\sum K)^* = (\sum K)^* \cdot (\sum K)^*$ and multiplication preserves order,
$$(\sum K)^* = (\sum K)^* \cdot (\sum K)^* \geq t_1 \cdot t_2.$$
- Given a term t, if $(\sum K)^* \geq t$, then $(\sum K)^* \geq t^*$. Since $(\sum K)^* = ((\sum K)^*)^*$ and star preserves order:
$$(\sum K)^* = ((\sum K)^*)^* \geq t^*.$$ ◀

Since every free KAT is a TopKAT, every KAT interpretation $I : \mathsf{KAT} \to \mathcal{K}$ induces a sub-KAT $\mathbf{Im}(I) \subseteq \mathcal{K}$, and this sub-KAT happens to be a *TopKAT*. Specifically, the image of $(\sum K)^*$ in $\mathcal{K}$ is the largest element of $\mathbf{Im}(I)$, and the restricted $I : \mathsf{KAT} \to \mathbf{Im}(I)$ is a TopKAT homomorphism.

This gives us a powerful tool to construct complete TopKAT interpretations. Since we already know that the KAT interpretations $G : \mathsf{KAT} \to \mathcal{G}$ and $h \circ G : \mathsf{KAT} \to \mathbf{Im}(h)$ are injective TopKAT homomorphisms, we can construct complete TopKAT interpretations by *composition*, if we can construct an injective TopKAT interpretation r of type $\mathsf{TopKAT}_{K,B} \to \mathsf{KAT}_{K_\top,B}$:

$$\mathsf{TopKAT}_{K,B} \xrightarrow{r} \mathsf{KAT}_{K_\top,B} \xrightarrow{G} \mathcal{G}_{K_\top,B}, \quad \mathsf{TopKAT}_{K,B} \xrightarrow{r} \mathsf{KAT}_{K_\top,B} \xrightarrow{G} \mathcal{G}_{K_\top,B} \xrightarrow{h} \mathbf{Im}(h).$$

In fact, such an injective homomorphism can be obtained by lifting the embedding map $K + B \hookrightarrow \mathsf{KAT}_{K_\top,B}$:

$$r : K + B \to \mathsf{KAT}_{K_\top,B}$$
$$r(p) \triangleq p.$$

This homomorphism coincides with the *reduction maps* of the same name in previous works [41, 35]. More concretely, we can picture r as simply replacing the symbol $\top$ in a TopKAT term with $(\sum K_\top)^*$, the largest element in $\mathsf{KAT}_{K_\top,B}$.

We will show that r is injective by constructing a left inverse for it. In fact, the left inverse $[-]_\top$ simply interprets the $\top$ primitive in $\mathsf{KAT}_{K_\top,B}$ as the largest element.

▶ **Lemma 7.** *The map $[-]_\top : \mathsf{KAT}_{K_\top,B} \to \mathsf{TopKAT}_{K,B}$, where each term is mapped to its corresponding equivalence class, is defined by lifting the following action on the primitives:*

$$[p]_\top \triangleq p \qquad \text{if } p \in K + B$$
$$[\top]_\top \triangleq \top.$$

The map $[-]_\top$ is a TopKAT homomorphism.

Proof. Because this map defined by lifting on the primitives, it is automatically a KAT homomorphism. All we need to show is that $[-]_\top$ preserves the top element, that is $[(\sum K_\top)^*]_\top = (\sum K_\top)^*$ is the largest element in $\mathsf{TopKAT}_{K,B}$.

By construction of $\mathsf{TopKAT}_{K,B}$, $\top$ is the largest element in $\mathsf{TopKAT}_{K,B}$. Thus, to prove that $(\sum K_\top)^*$ is also the largest element in $\mathsf{TopKAT}_{K,B}$, it suffices to prove $(\sum K_\top)^* \geq \top$:

$$(\sum K_\top)^* \geq \sum K_\top = \top + \sum K \geq \top. \qquad \blacktriangleleft$$

▶ **Theorem 8** (Reduction). *$[-]_\top$ is the right inverse of r: $[-]_\top \circ r = id_{\mathsf{TopKAT}_{K,B}}$. More explicitly for all $t \in \mathsf{TopKAT}_{K,B}$:*

$$\mathsf{TopKAT}_{K,B} \models [r(t)]_\top = t.$$

Proof. Since $[-]_\top \circ r : \mathsf{TopKAT}_{K,B} \to \mathsf{TopKAT}_{K,B}$ is a TopKAT interpretation, the action on the primitives uniquely determines the interpretation: because both r and $[-]_\top$ are identity on the primitives, therefore $[-]_\top \circ r$ is the identity interpretation on $\mathsf{TopKAT}_{K,B}$. ◀

The above theorem matches one of the soundness condition of reductions in previous works [41, 24, 33], which was typically proven by a monolithic induction on the structure of terms. Our approach, on the other hand, relies on establishing fine-grained algebraic properties, like Lemmas 6 and 7; then the theorem follows simply by computing the action of $[-]_\top \circ r$ on primitives.

Since r has a right inverse, it is indeed the injective interpretation we desired, and it is also a complete interpretation:

$$\mathsf{TopKAT}_{K,B} \models t_1 = t_2 \iff r(t_1) = r(t_2),$$

With the completeness of r, we can already show the complexity of TopKAT. The complexity results echos previous proofs [41, 35], but we are able to obtain this result without completeness of TopKAT language interpretation, which is essential in previous proofs.

▶ **Corollary 9** (Complexity). *Given two terms $t_1, t_2 \in \mathsf{TopKAT}_{K,B}$, deciding whether these two terms are equal is PSPACE-complete.*

Proof. Deciding KAT equality is a sub-problem of deciding TopKAT equality, and KAT equality is PSPACE-hard [6]; therefore TopKAT equality is PSPACE-hard.

To decide the equality of t_1, t_2, we first remove all the redundant primitives that do not appear in t_1, t_2 from the alphabet K, B. Then we compute $r(t_1)$ and $r(t_2)$, each taking polynomial space (of $|t_1| + |t_2|$) to store; and we use the standard algorithm [6] to decide whether $r(t_1) = r(t_2)$ in $\mathsf{KAT}_{K_\top,B}$, this will also take polynomial space. Hence, the decision procedure for TopKAT equality in PSPACE.

Thus deciding TopKAT equality is PSPACE-complete. ◀

3.2 Computing the complete interpretations

Designing complete interpretations and models was not always easy. In fact, in previous works [42], the authors made a mistake in the definition of language TopKAT, which was fixed later [41] by suggestion of Pous et al. [34]. However, with the results in Section 3.1, we can construct the complete interpretation just by composition, and compute the complete model by computing the range of the complete interpretation.

We already know that there are two complete interpretations of TopKAT defined as follows:

$$\mathsf{TopKAT}_{K,B} \xrightarrow{r} \mathsf{KAT}_{K_\top,B} \xrightarrow{G} \mathcal{G}_{K_\top,B}, \quad \mathsf{TopKAT}_{K,B} \xrightarrow{r} \mathsf{KAT}_{K_\top,B} \xrightarrow{G} \mathcal{G}_{K_\top,B} \xrightarrow{h} \mathbf{Im}(h),$$

with a complete language model $\mathcal{G}_{K_\top,B}$, and a complete model consisting of relations $\mathbf{Im}(h)$.

The operations in these models can be recovered by computing these maps. For example, the multiplication operation in the language TopKAT can be computed as follows:

$$G \circ r(t_1 \cdot t_2) = G(r(t_1) \cdot r(t_2)) = G(r(t_1)) \diamond G(r(t_2)).$$

Since r does not change the multiplication operation, the multiplication in the language TopKAT is the same as in language KAT. In fact, as r does not change any operation in KAT, most operations in language TopKAT are the same as language KAT. Thus, we only need to compute the top element in language TopKAT.

The top element in language TopKAT can be computed in the same fashion:

$$G \circ r(\top) = G((\sum K_\top)^*) = GS_{K_\top,B},$$

i.e. the top element is just the complete language.

▶ **Corollary 10.** *The language TopKAT inherits all the operations in language KAT, except the top element, which is defined as the full language. And such models are complete with $G \circ r$ as a complete interpretation.*

In the same way, we know that complete models consisting of relations (a.k.a. general relational TopKAT) will have the same operations as relational KATs. However, in this case the characterization of the computed top: $h \circ G \circ r(\top)$ is not as simple as the full language, but we know it is the largest relation in the range of $h \circ G \circ r$:

▶ **Corollary 11.** *The general relational TopKAT inherits all the operations in relational KAT, except the top element is the largest relation. And such models are complete with $h \circ G \circ r$ as a complete interpretation.*

Finally, to investigate whether we can use general relational TopKAT to encode incorrectness logic, we will provide a short proof that general relational TopKATs are as expressive as relational KATs [41]; that is, every property on relations that can be encoded using general relational TopKAT, is already encodable in the relational KAT. Hence, adding a top element does not give extra expressive power in general relational TopKAT.

The original proof [41, Lemma 2] encodes every TopKAT term using a KAT term, and then uses two pages to prove the soundness of this encoding. Here we show the aforementioned encoding is simply the reduction r.

▶ **Definition 12.** *Given two terms $t_1, t_2 \in \mathsf{TopKAT}$, and n primitives $p_1, p_2, \ldots, p_n \in K + B$, we say that an n-ary predicate P is* expressible *by equation $t_1 = t_2$ for a class of TopKATs K when for all interpretations I into TopKATs in K, the following equivalence holds:*

$$I(t_1) = I(t_2) \iff P(I(p_1), I(p_2), \ldots, I(p_n)).$$

▶ **Theorem 13** (Expressiveness of general relational TopKAT)**.** *Given an alphabet K, B, an n-ary predicate P on relations, the predicate P over primitives $p_1, p_2, \ldots, p_n \in K$ is expressible in general relational TopKAT if and only if it is expressible in relational KAT.*

Proof. A predicate expressible in relational KAT is also expressible in general relational TopKAT using the same pair of terms, we only need to show the converse. Assume a predicate P is expressible in general relational TopKAT, then there exists two TopKAT terms $t_1, t_2 \in \mathsf{TopKAT}_{K,B}$ s.t. for all general relational TopKAT interpretations $I_\top$:

$$I_\top(t_1) = I_\top(t_2) \iff P(I_\top(p_1), I_\top(p_2), \ldots, I_\top(p_n));$$

We take an arbitrary relational KAT interpretation I from $\mathsf{KAT}_{K_\top,B}$. Notice $\mathbf{Im}(I)$, the range of I, is a relational KAT with the largest element $I((\sum K)^*)$, i.e. $\mathbf{Im}(I)$ is a general relational TopKAT. Because I is a KAT interpretation, it preserves all the KAT operations and the largest element. Hence, I is a TopKAT homomorphism from $\mathsf{KAT}_{K_\top,B}$ to $\mathbf{Im}(I)$.

Then we can construct $I \circ r : \mathsf{TopKAT}_{K,B} \to \mathbf{Im}(I)$, a general relational interpretation:

$$\begin{aligned} I(r(t_1)) = I(r(t_2)) &\iff I \circ r(t_1) = I \circ r(t_2) \\ &\iff P(I \circ r(p_1), \ldots, I \circ r(p_n)) \quad I \circ r \text{ is a TopGREL interpretation} \\ &\iff P(I(p_1), \ldots, I(p_n)) \qquad\qquad r(p_i) = p_i \end{aligned}$$

Thus the two KAT terms $r(t_1), r(t_2) \in \mathsf{KAT}_{K_\top,B}$ also can express the predicate P. ◀

Since the image of I is not necessarily a relational TopKAT, where the top element is interpreted as the complete relation, the above trick does not work for relational TopKAT. It is also known that relational TopKAT is strictly more expressive than general relational TopKAT, since relational TopKAT can encode incorrectness logic, where general relational TopKAT cannot [41].

4 (Co)domain Completeness

In general, TopKAT is not complete over relational models, which are crucial for applications in program logics [41]. However, it was later showed that we can obtain a complete theory for relational models by simply adding the axiom $p\top p \geq p$ to the theory of TopKAT [35].

In this paper, we take a different approach than Pous et al. [35]: instead of extending the TopKAT framework, we will restrict the completeness result. In particular, the encoding of incorrectness logic and Hoare Logic in TopKAT [41] relies only on the ability of TopKAT to compare the domain and codomain of two relations. This raises the question of whether TopKAT suffices for proving such properties; that is, whether the following completeness results hold: for $t_1, t_2 \in \mathsf{KAT}_{K,B}$ (i.e. $\top$ does not appear in t_1 and t_2)

$$\begin{aligned} \text{REL} \models \mathrm{cod}(t_1) \geq \mathrm{cod}(t_2) &\iff \mathsf{TopKAT} \models \top t_1 \geq \top t_2 \qquad \text{codomain completeness} \\ \text{REL} \models \mathrm{dom}(t_1) \geq \mathrm{dom}(t_2) &\iff \mathsf{TopKAT} \models t_1\top \geq t_2\top \qquad \text{domain complete} \end{aligned}$$

In this section, we prove that these equivalences hold, even without the additional axiom. However, they do *not* hold if we allow terms that contain top. For example, let $t_1 \triangleq p\top p$, and $t_2 \triangleq p$. Since $p\top p \geq p$ holds in relational TopKAT, thus $\mathrm{dom}(p\top p) \geq \mathrm{dom}(p)$. However, $p\top p\top \geq p\top$ is not provable in TopKAT, because the inequality is not valid with the language interpretation. The incompleteness of codomain comparison can also be shown using the same example.

4.1 Codomain completeness

The core insight to prove the domain completeness result is to construct a specific relational interpretation $h \circ i \circ G$, where its codomain is equivalent to the complete TopKAT interpretation $G \circ r$:

$$\mathrm{cod}(h \circ i \circ G(t)) = G \circ r(\top t),$$

where i is the natural inclusion homomorphism $i : \mathcal{G}_{K,B} \hookrightarrow \mathcal{G}_{K_\top,B}$, that maps every language to itself; and h is the classical embedding of language KAT into relational KAT [24], which we will recall as follows:

$$h(L) = \{(s, s \diamond s') \mid s \in GS, s' \in L\}.$$

Although i will not change the outcome of G, it will add a new primitive action $\top$ to the alphabet, hence changing the outcome of h. Such addition will equate the codomain of $h \circ i \circ G(t)$ with the complete TopKAT interpretation $G \circ r$ of $\top t$. The proof of this equality is by simply computing both sides of the equation.

▶ **Lemma 14.** *For any term $t \in \mathsf{KAT}_{K,B}$,*

$$\mathrm{cod}(h \circ i \circ G(t)) = G \circ r(\top t).$$

Proof. We explicitly write out the domain and codomain of the functions in the relational KAT interpretation $h \circ i \circ G$ for the ease of the reader:

$$\mathsf{KAT}_{K,B} \xrightarrow{G} \mathcal{G}_{K,B} \xrightarrow{i} \mathcal{G}_{K_\top,B} \xrightarrow{h} \mathcal{P}(\mathcal{G}_{K_\top,B} \times \mathcal{G}_{K_\top,B}).$$

In this case, h is a KAT homomorphism from $\mathcal{G}_{K_\top,B}$:

$$h(S) = \{(s, s \diamond s_1) \mid s \in GS_{K_\top,B}, s_1 \in S\}.$$

Since the reduction r preserves terms without $\top$, let $t \in \mathsf{KAT}_{K,B}$ (i.e. t does not contain $\top$),

$$G \circ r(\top) = GS_{K_\top,B} \qquad\qquad G \circ r(t) = G(t).$$

Therefore, for any term $t \in \mathsf{KAT}_{K,B}$

$$\begin{aligned}
\mathrm{cod}(h \circ i \circ G(t)) &= \{s\alpha s_1 \mid s\alpha \in GS_{K_\top,B}, \alpha s_1 \in G(t)\} \\
&= GS_{K_\top,B} \diamond G(t) \\
&= (G \circ r(\top)) \diamond (G \circ r(t)) \\
&= G \circ r(\top t).
\end{aligned}$$

◀

Lemma 14 established a connection between the codomain operator and the language interpretation of TopKAT. Then by completeness of the language interpretation, we will obtain the completeness of codomain comparison.

▶ **Theorem 15** (Codomain completeness)**.** *Given two terms $t_1, t_2 \in \mathsf{KAT}_{K,B}$ (i.e. terms without $\top$), then codomain comparison is complete:*

$$\mathrm{REL} \models \mathrm{cod}(t_1) \geq \mathrm{cod}(t_2) \iff \mathsf{TopKAT} \models \top t_1 \geq \top t_2.$$

Proof. Given the natural inclusion homomorphism: $i : \mathsf{KAT}_{K,B} \to \mathsf{KAT}_{K_\top,B}$, we show that the following are equivalent:

1. $\mathrm{REL} \models \mathrm{cod}(t_1) \geq \mathrm{cod}(t_2)$.
2. $\mathrm{cod}(h \circ i \circ G(t_1)) \geq \mathrm{cod}(h \circ i \circ G(t_2))$.
3. $\mathsf{TopKAT} \models \top t_1 \geq \top t_2$.

We first show that $1 \implies 2$, by definition, $\mathrm{REL} \models \mathrm{cod}(t_1) \geq \mathrm{cod}(t_2)$ implies $\mathrm{cod}(I(t_1)) \geq \mathrm{cod}(I(t_2))$ for all relational KAT interpretations I. Because $h \circ i \circ G$ is a relational KAT interpretation, so $1 \implies 2$.

We show 2 $\implies$ 3, which uses the equality discussed above, and proved in Lemma 14:

$$\begin{aligned}
&\mathrm{cod}(h \circ i \circ G(t_1)) \geq \mathrm{cod}(h \circ i \circ G(t_2)) \\
\iff\ & G \circ r(\top t_1) \geq G \circ r(\top t_2) && \text{Lemma 14} \\
\iff\ & \mathsf{TopKAT} \models \top t_1 \geq \top t_2. && \text{Completeness of } G \circ r
\end{aligned}$$

Finally, we show 3 $\implies$ 1, by Lemma 5:

$$\mathsf{TopKAT} \models \top t_1 \geq \top t_2 \implies \mathrm{TopREL} \models \top t_1 \geq \top t_2 \implies \mathrm{REL} \models \mathrm{cod}(t_1) \geq \mathrm{cod}(t_2). \quad \blacktriangleleft$$

4.2 Domain completeness

The domain completeness result can be derived from codomain completeness by observing properties of opposite TopKAT and the converse operator $(-)^\vee$, both of which we will recall below.

For every TopKAT $\mathcal{K}$, we can construct the opposite TopKAT $\mathcal{K}^{\mathrm{op}}$ by reversing the multiplication operation, keeping the sorts and other operations unchanged:

$$p \mathbin{\hat{\cdot}} q \triangleq q \cdot p,$$

where $\hat{\cdot}$ is multiplication in $\mathcal{K}^{\mathrm{op}}$ and $\cdot$ is multiplication in $\mathcal{K}$. By definition, $(-)^{\mathrm{op}}$ is a involution, that is $(\mathcal{K}^{\mathrm{op}})^{\mathrm{op}} = \mathcal{K}$. Furthermore, $(-)^{\mathrm{op}}$ is a TopKAT functor, this means all TopKAT homomorphisms $h : \mathcal{K} \to \mathcal{K}'$ can be lifted to a TopKAT homomorphism on the opposite TopKAT $h^{\mathrm{op}} : \mathcal{K}^{\mathrm{op}} \to \mathcal{K}'^{\mathrm{op}}$. The lifting h^{op} is point-wise equal to h:

$$\forall p \in \mathcal{K}, h^{\mathrm{op}}(p) \triangleq h(p).$$

The fact that h^{op} is a TopKAT homomorphism can be proven by unfolding the definition, and the functor laws are satisfied because h^{op} is point-wise equal to h.

There are two important homomorphisms involving opposite TopKAT:

$$\begin{aligned}
&(-)^\vee : (X \times X)^{\mathrm{op}} \to (X \times X) && \mathrm{op} : \mathsf{TopKAT}_{K,B} \to \mathsf{TopKAT}^{\mathrm{op}}_{K,B} \\
&(R)^\vee = \{(b,a) \mid (a,b) \in R\}, && \forall p \in K + B, \mathrm{op}(p) = p.
\end{aligned}$$

The $(-)^\vee$ is the relational converse operator, the rules of homomorphism can simply be proven by unfolding of definitions. The crucial property of $(-)^\vee$ is that it flips the domain and codomain:

$$\mathrm{dom}(R^\vee) = \mathrm{cod}(R). \tag{2}$$

Hence, allowing us to flip the result about codomains and apply it to domains.

op is a homomorphism from free TopKAT to its opposite TopKAT; it can be defined by lifting the embedding function $K + B \hookrightarrow \mathsf{TopKAT}_{K,B}$ on primitives. Intuitively, given a term $t \in \mathsf{TopKAT}$, $\mathrm{op}(t)$ will flip all the multiplications in t recursively.

▶ **Lemma 16.** *the left inverse of op can be obtained by lifting itself through the* $(-)^{\mathrm{op}}$ *functor,*

$$\mathrm{op}^{\mathrm{op}} : \mathsf{TopKAT}^{\mathrm{op}} \to (\mathsf{TopKAT}^{\mathrm{op}})^{\mathrm{op}} = \mathsf{TopKAT}.$$

Recall $\mathrm{op}^{\mathrm{op}}$ *is pointwise equal to* op*, thus* $\mathrm{op}^{\mathrm{op}} \circ \mathrm{op} : \mathsf{TopKAT} \to \mathsf{TopKAT}$ *is the identity interpretation because it preserves all the primitives. Thus,* op *has a left inverse, hence it is injective:*

$$t_1 = t_2 \iff \mathrm{op}(t_1) = \mathrm{op}(t_2).$$

Finally, since the elements in $\mathsf{TopKAT}^{\mathrm{op}}$ are the same as TopKAT, which are TopKAT terms modulo provable TopKAT equalities, theorems about TopKAT terms are also true for elements in $\mathsf{TopKAT}^{\mathrm{op}}$. In particular, codomain completeness (Theorem 15) also holds in $\mathsf{TopKAT}^{\mathrm{op}}$: for all terms $t_1, t_2 \in \mathsf{TopKAT}$,

$$\top \cdot \mathrm{op}(t_1) \geq \top \cdot \mathrm{op}(t_2) \iff \mathrm{REL} \models \mathrm{cod}(\mathrm{op}(t_1)) = \mathrm{cod}(\mathrm{op}(t_2)). \tag{3}$$

▶ **Theorem 17** (Domain Completeness). *For all terms $t_1, t_2 \in \mathsf{KAT}$, the following equivalence hold:*

$$\mathrm{REL} \models \mathrm{dom}(t_1) = \mathrm{dom}(t_2) \iff \mathsf{TopKAT} \models t_1\top \geq t_2\top.$$

Proof. $\Longleftarrow$ direction is trivial by Lemma 5; and $\Longrightarrow$ direction can be derived as follows: let I be some relational interpretation, then $I^{\mathrm{op}}(\mathrm{op}(-))^\vee$ is also a relational interpretation:

$$I^{\mathrm{op}}(\mathrm{op}(-))^\vee : \mathsf{TopKAT} \xrightarrow{\mathrm{op}} \mathsf{TopKAT}^{\mathrm{op}} \xrightarrow{I^{\mathrm{op}}} (X \times X)^{\mathrm{op}} \xrightarrow{(-)^\vee} (X \times X).$$

Thus, we let I range over all relational interpretations:

$$\begin{aligned}
&\mathrm{REL} \models \mathrm{dom}(t_1) \supseteq \mathrm{dom}(t_2) \\
&\Longrightarrow \forall I, \mathrm{dom}(I(t_1)) \supseteq \mathrm{dom}(I(t_2)) \\
&\Longrightarrow \forall I, \mathrm{dom}(I^{\mathrm{op}}(\mathrm{op}(t_1))^\vee) \supseteq \mathrm{dom}(I^{\mathrm{op}}(\mathrm{op}(t_2))^\vee) && \text{specialize } I \text{ as } I^{\mathrm{op}}(\mathrm{op}(-))^\vee \\
&\Longrightarrow \forall I, \mathrm{cod}(I^{\mathrm{op}}(\mathrm{op}(t_1))) \supseteq \mathrm{cod}(I^{\mathrm{op}}(\mathrm{op}(t_1))) && \text{Equation (2)} \\
&\Longrightarrow \forall I, \mathrm{cod}(I(\mathrm{op}(t_1))) \supseteq \mathrm{cod}(I(\mathrm{op}(t_1))) && I^{\mathrm{op}} \text{ is pointwise equal to } I \\
&\Longrightarrow \top \cdot \mathrm{op}(t_1) \geq \top \cdot \mathrm{op}(t_2) && \text{Equivalence (3)} \\
&\Longrightarrow \mathrm{op}(\top \cdot t_1) \geq \mathrm{op}(\top \cdot t_2) && \text{Definition of op} \\
&\Longrightarrow t_1\top \geq t_2\top && \text{Lemma 16}
\end{aligned}$$ ◀

▶ Remark 18. Alternatively, Theorem 17 can also be proven by constructing the following h':

$$h' : \mathcal{G}_{K,B} \to \mathcal{P}(\mathcal{G}_{K,B} \times \mathcal{G}_{K,B})$$
$$h'(S_1) \triangleq \{(s_1\alpha s, \alpha s) \mid s_1\alpha \in S_1, \alpha s \in GS_{K,B}\}.$$

Then the proof would mirror that of Theorem 15, replacing h with h' and replacing cod with dom. However, the proof of Theorem 17 reveals more properties of maps like $(-)^\vee$ and op, thus we choose to present the current proof of Theorem 17 instead of the alternative proof.

5 Related Works

Extensions of Kleene algebra and reduction. soon after the completeness of Kleene algebra was proven [18], it was realized that adding an embedded Boolean algebra can help reasoning about control structures, such system is referred to as Kleene algebra with tests (KAT) [24, 6]. Later KAT was further extended to reason about failure [26], indicator variables [13], domain [9], networks [1], and relational reasoning [3]. Kleene algebra has also been extended to reason about concurrency, as concurrent Kleene algebra [14, 17] and concurrent Kleene algebra with observations [16]. Many of these extensions can be seen as Kleene algebra with extra hypotheses [5, 11]. Although many hypotheses make the theory undecidable [19, 22, 11], many useful hypotheses can be eliminated via reduction [33]. Thus, our new perspective on reduction could potentially lead to streamlining of various previous proofs, and more general proofs of completeness results.

Top element. Tarski's relational algebra [40] contains the addition, mulitiplication, and identity operation of KA; in addition, relational algebra also include a top element. Hence attempts to incorporat Kleene star into relational algebra effectively create a super theory of TopKAT. Unfortuantly, several attempts at these algebras turn out to be undecidable because of the presence of intersection and converse operations [2, 32]. With the intersection and converse operators removed, top element is proven to be individually useful in Kleene algebra: for example, Mamouras [26] uses the top element to forget program states, and Antonopoulos et al. [3] uses top to design forward simulation rules for relational verification, and claim that relational incorrectness logic [29] can be encoded using BiKAT extended with top. The completeness and decidability of TopKAT was first studied by Zhang et al. [41], and concluded that TopKAT is not complete with relational models. Later, Pous et al. [34, 35] showed that both TopKA and TopKAT is complete with relational model with one additional axiom: $p\top p \geq p$, and the theory remains PSPACE-complete, like KAT and TopKAT. In this paper, we showed that TopKAT without the additional axiom is complete for a specific form of inequalities, namely when top only appears in the front or the end of the term. Although this form of inequalities seem restrictive, they are enough to encode both Hoare and incorrectness logic [41].

Domain in KAT. The study of axiomatizing (co)domain in KAT has a long and rich history. Domain semiring [10] and Kleene algebra with domain [9] were two popular yet different axiomatizations of (co)domain in Kleene algebra with tests. These two axiomitizations turn out to coincide in a large class of semirings [12]. Various applications for domain in KAT have been discovered, including modeling program correctness, predicate transformers, temporal logics, termination analysis, and many more [8]. Many of these applications can even be efficiently automated [15]. However, although the free relational model of these theories has been characterized [28], the search for general complete interpretation remains unfruitful. The complexity of these theories was recently shown to be EXPTIME-complete [37], a worse complexity class than PSPACE-complete for TopKAT.

6 Conclusion And Open Problems

In this paper, we exploit the homomorphic structure of reduction to simplify the proof of various previous results [41]. We have also showed that TopKAT is complete with respect to (co)domain comparison in the relational models, which lays a solid foundation for the use of TopKAT in (co)domain reasoning.

However, there are still several interesting unsolved problems about TopKAT. Most of the incorrectness logic rules are written using hypotheses, for example, the sequencing rule:

$$\frac{[a]\ p\ [b] \qquad [b]\ q\ [c]}{[a]\ p \cdot q\ [c]}$$

corresponds to the implication $\top ap \leq \top b \wedge \top bp \leq \top c \implies \top apq \leq \top c$. Although each individual inequality in the implication fits the desired form $\top t_1 \geq \top t_2$. it is unclear whether implications of the form

$$\top t_{11} \leq \top t_{12} \wedge \top t_{21} \leq \top t_{22} \wedge \cdots \wedge \top t_{n1} \leq \top t_{n2} \implies \top t_1 \leq \top t_2$$

are complete with relational TopKAT or decidable.

Recently, there is an efficient fragment of KAT proposed, named *Guarded Kleene algebra with tests* [38] or *GKAT*. This fragment not only enjoys nearly-linear time equality checking, but also soundly models probabilistic computations as well. It would be interesting to see whether the completeness and decidability result of TopKAT can be extended to GKAT, and whether the efficiency of GKAT will persist with the addition of top.

References

1 Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. Netkat: semantic foundations for networks. *ACM SIGPLAN Notices*, 49(1):113–126, January 2014. doi:10.1145/2578855.2535862.

2 Hajnal Andréka and Szabolcs Mikulás. Axiomatizability of positive algebras of binary relations. *Algebra universalis*, 66(1-2):7–34, October 2011. doi:10.1007/s00012-011-0142-3.

3 Timos Antonopoulos, Eric Koskinen, Ton Chanh Le, Ramana Nagasamudram, David A. Naumann, and Minh Ngo. An Algebra of Alignment for Relational Verification. *Proceedings of the ACM on Programming Languages*, 7(POPL):20:573–20:603, January 2023. doi:10.1145/3571213.

4 Stanley Burris and H. P. Sankappanavar. *A Course in Universal Algebra*. Number 78 in Graduate Texts in Mathematics. Springer-Verlag, New York, 1981.

5 Ernie Cohen. Hypotheses in kleene algebra, March 1995.

6 Ernie Cohen, Dexter Kozen, and Frederick Smith. The Complexity of Kleene Algebra with Tests. Technical Report, Cornell University, USA, June 1996.

7 Edsko de Vries and Vasileios Koutavas. Reverse Hoare Logic. In Gilles Barthe, Alberto Pardo, and Gerardo Schneider, editors, *Software Engineering and Formal Methods*, volume 7041, pages 155–171. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-24690-6_12.

8 Jules Desharnais, Bernhard Möller, and Georg Struth. Modal kleene algebra and applications – a survey. In *Journal on Relational Methods in Computer Science*, pages 93–131, 2004.

9 Jules Desharnais, Bernhard Möller, and Georg Struth. Kleene algebra with domain. *ACM Transactions on Computational Logic*, 7(4):798–833, October 2006. doi:10.1145/1183278.1183285.

10 Jules Desharnais and Georg Struth. Internal axioms for domain semirings. *Science of Computer Programming*, 76(3):181–203, March 2011. doi:10.1016/j.scico.2010.05.007.

11 Amina Doumane, Denis Kuperberg, Damien Pous, and Pierre Pradic. Kleene algebra with hypotheses. In *22nd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, Proc. FoSSaCS 2019, Prague, Czech Republic, 2019. Springer.

12 Uli Fahrenberg, Christian Johansen, Georg Struth, and Krzysztof Ziemiánski. Domain semirings united. *arXiv:2011.04704 [cs]*, March 2021.

13 Niels Bjørn Bugge Grathwohl, Dexter Kozen, and Konstantinos Mamouras. Kat + b! In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, pages 1–10, New York, NY, USA, July 2014. Association for Computing Machinery. doi:10.1145/2603088.2603095.

14 Tony Hoare, Stephan van Staden, Bernhard Möller, Georg Struth, and Huibiao Zhu. Developments in concurrent kleene algebra. *Journal of Logical and Algebraic Methods in Programming*, 85(4):617–636, June 2016. doi:10.1016/j.jlamp.2015.09.012.

15 Peter Höfner and Georg Struth. Automated Reasoning in Kleene Algebra. In Frank Pfenning, editor, *Automated Deduction – CADE-21*, Lecture Notes in Computer Science, pages 279–294, Berlin, Heidelberg, 2007. Springer. doi:10.1007/978-3-540-73595-3_19.

16 Tobias Kappé, Paul Brunet, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. *Concurrent Kleene Algebra with Observations: From Hypotheses to Completeness*, volume 12077 of *Lecture Notes in Computer Science*, pages 381–400. Springer International Publishing, Cham, 2020. doi:10.1007/978-3-030-45231-5_20.

17 Tobias Kappé, Paul Brunet, Alexandra Silva, and Fabio Zanasi. Concurrent kleene algebra: Free model and completeness. In Amal Ahmed, editor, *Programming Languages and Systems*, Lecture Notes in Computer Science, pages 856–882, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-319-89884-1_30.

18 D. Kozen. A completeness theorem for kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, May 1994. doi:10.1006/inco.1994.1037.

19 Dexter Kozen. *Kleene algebra with tests and commutativity conditions*, volume 1055 of *Lecture Notes in Computer Science*, pages 14–33. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996. doi:10.1007/3-540-61042-1_35.

20 Dexter Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997. doi:10.1145/256167.256195.

21 Dexter Kozen. On hoare logic and kleene algebra with tests. *ACM Transactions on Computational Logic*, 1(1):60–76, July 2000. doi:10.1145/343369.343378.

22 Dexter Kozen. On the complexity of reasoning in kleene algebra. *Information and Computation*, 179(2):152–162, December 2002. doi:10.1006/inco.2001.2960.

23 Dexter Kozen and Alexandra Silva. Left-handed completeness. *Theoretical Computer Science*, 807:220–233, February 2020. doi:10.1016/j.tcs.2019.10.040.

24 Dexter Kozen and Frederick Smith. *Kleene algebra with tests: Completeness and decidability*, volume 1258 of *Lecture Notes in Computer Science*, pages 244–259. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997. doi:10.1007/3-540-63172-0_43.

25 Quang Loc Le, Azalea Raad, Jules Villard, Josh Berdine, Derek Dreyer, and Peter W. O'Hearn. Finding real bugs in big programs with incorrectness logic. *Proceedings of the ACM on Programming Languages*, 6(OOPSLA1):1–27, April 2022. doi:10.1145/3527325.

26 Konstantinos Mamouras. *Equational Theories of Abnormal Termination Based on Kleene Algebra*, volume 10203 of *Lecture Notes in Computer Science*, pages 88–105. Springer Berlin Heidelberg, Berlin, Heidelberg, 2017. doi:10.1007/978-3-662-54458-7_6.

27 Ernest G. Manes and Michael A. Arbib. *Algebraic Approaches to Program Semantics*. Springer New York, New York, NY, 1986.

28 Brett McLean. Free Kleene algebras with domain. *Journal of Logical and Algebraic Methods in Programming*, 117:100606, December 2020. doi:10.1016/j.jlamp.2020.100606.

29 Toby Murray. An Under-Approximate Relational Logic. *Archive of Formal Proofs*, March 2020.

30 Bernhard Möller, Peter O'Hearn, and Tony Hoare. *On Algebra of Program Correctness and Incorrectness*, volume 13027 of *Lecture Notes in Computer Science*, pages 325–343. Springer International Publishing, Cham, 2021. doi:10.1007/978-3-030-88701-8_20.

31 Peter W. O'Hearn. Incorrectness logic. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–32, January 2020. doi:10.1145/3371078.

32 Damien Pous. On the Positive Calculus of Relations with Transitive Closure. In *35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.3.

33 Damien Pous, Jurriaan Rot, and Jana Wagemaker. On tools for completeness of kleene algebra with hypotheses. In *Relational and Algebraic Methods in Computer Science: 19th International Conference, RAMiCS 2021, Marseille, France, November 2–5, 2021, Proceedings*, pages 378–395, Berlin, Heidelberg, November 2021. Springer-Verlag. doi:10.1007/978-3-030-88701-8_23.

34 Damien Pous and Jana Wagemaker. Completeness theorems for kleene algebra with top. In Bartek Klin, Slawomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory (CONCUR 2022)*, volume 243 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CONCUR.2022.26.

35 Damien Pous and Jana Wagemaker. Completeness theorems for kleene algebra with tests and top. *CoRR*, April 2023. arXiv:2304.07190.

36 Azalea Raad, Josh Berdine, Hoang-Hai Dang, Derek Dreyer, Peter O'Hearn, and Jules Villard. *Local Reasoning About the Presence of Bugs: Incorrectness Separation Logic*, volume 12225 of *Lecture Notes in Computer Science*, pages 225–252. Springer International Publishing, Cham, 2020. doi:10.1007/978-3-030-53291-8_14.

37 Igor Sedlár. On the complexity of kleene algebra with domain. In *Relational and Algebraic Methods in Computer Science: 20th International Conference, RAMiCS 2023, Augsburg, Germany, April 3–6, 2023, Proceedings*, pages 208–223, Berlin, Heidelberg, April 2023. Springer-Verlag. doi:10.1007/978-3-031-28083-2_13.

38 Steffen Smolka, Nate Foster, Justin Hsu, Tobias Kappé, Dexter Kozen, and Alexandra Silva. Guarded kleene algebra with tests: verification of uninterpreted programs in nearly linear time. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–28, January 2020. doi:10.1145/3371129.

39 Georg Struth. On the expressive power of kleene algebra with domain. *CoRR*, August 2015. arXiv:1507.07246.

40 Alfred Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6(3):73–89, September 1941. doi:10.2307/2268577.

41 Cheng Zhang, Arthur Azevedo de Amorim, and Marco Gaboardi. On incorrectness logic and kleene algebra with top and tests. *CoRR*, August 2022. doi:10.48550/arXiv.2108.07707.

42 Cheng Zhang, Arthur Azevedo de Amorim, and Marco Gaboardi. On incorrectness logic and kleene algebra with top and tests. *Proceedings of the ACM on Programming Languages*, 6(POPL):29:1–29:30, January 2022. doi:10.1145/3498690.